Leveraging Transformer Model and Heuristic Strategies for Solving the Traveling Salesman Problem

Denis Levchenko[1] and Éric D. Taillard[1*]

[1*]HEIG-Vd, University of Applied Sciences and Arts of Western Switzerland, CH-1401, Yverdon-les-Bains, Switzerland.

*Corresponding author(s). E-mail(s): eric.taillard@heig-vd.ch;
Contributing authors: levchenkodenis@gmail.com;

Abstract

Transformer-based neural networks have emerged as powerful tools for combinatorial optimization problems, such as the Traveling Salesman Problem (TSP). However, their high computational demands during training raise concerns about scalability. This paper explores the use of POPMUSIC, a fast heuristic, to replace resource-intensive training with lightweight edge scoring. The study compares several sampling techniques —greedy search, beam search, and randomized selection (inspired by ant colony optimization)— both guided by POPMUSIC-generated edge frequencies and transformer outputs. Additionally, it evaluates how a transformer model, trained on uniform TSP instances of fixed size, generalizes to clustered and larger instances. The results demonstrate that randomized construction consistently outperforms beam search for both POPMUSIC and transformer outputs. While the pre-trained transformer generalizes well to larger and structurally different instances, traditional heuristics still surpass neural networks for large-scale TSPs. The findings highlight promising directions for hybrid methods that combine neural scoring with advanced heuristic selection strategies.

Keywords: Transformer Performance, Combinatorial Optimization, Traveling Salesman Problem, Beam Search, Ant Colony Optimization

Preprint: 2026/04/10

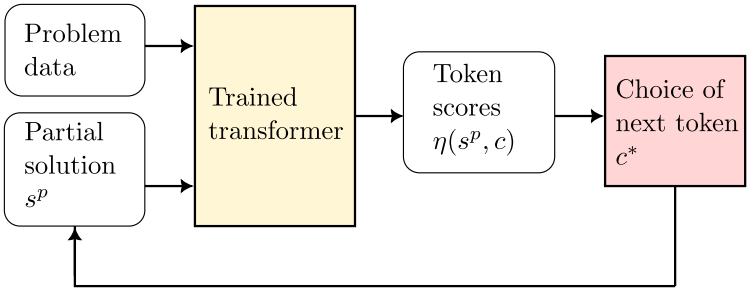

Fig. 1 Schematic illustration of an auto-regressive transformer-based neural net solver.

1 Introduction

Transformer-based neural networks [1] have profoundly reshaped the field of artificial intelligence, especially in natural language processing. Originally introduced to improve performance in machine translation, these models rely on an attention mechanism [2] that enables them to handle long-range dependencies within sequences without requiring a recurrent structure. The transformer processes all elements in a sequence in parallel, assigning each a weight based on its contextual relevance to the others.

Today, transformer-based neural networks form the backbone of modern AI tools, particularly in tasks such as machine translation, text generation, and conversational agents. In these applications, they are typically used in an auto-regressive manner: text is generated one element at a time (e.g., a word or token), with each new word predicted based on the preceding context. The same transformer model is reused at each step, with the previous output integrated into the next input, allowing for smooth and coherent sequence generation.

These models are general-purpose and can be applied beyond natural language processing tasks [3]. More recently, they have been adapted for solving difficult optimization problems [4], particularly in contexts where solutions must be generated sequentially, in a structured or combinatorial fashion. For instance, auto-regressive transformer-based models have been applied to classic combinatorial optimization problems such as the Traveling Salesman Problem (TSP), task scheduling[5], and the design of logic or molecular circuits. In these cases, the model learns to build a solution step by step, selecting at each iteration the next action or component to add, similar to text generation. This approach leverages the powerful generalization and contextual learning capabilities of transformers while retaining an auto-regressive framework well suited for producing optimal or near-optimal solutions.

While non-auto-regressive models allow for significantly faster training and inference by generating multiple elements simultaneously, this typically comes at the cost of solution quality [6].

Figure 1 schematically illustrates the operational structure of a transformer-based neural network solver. This architecture is characteristic of models used in natural language processing, but it can also be adapted to tackle combinatorial optimization

problems. The central component represents a neural network with a Transformer architecture, organized into multiple attention layers.

This network takes as input both the problem data and a partial solution. The Transformer encodes these inputs, and the decoding process yields a score for each token, representing its suitability to extend the given partial solution.

These scores are then fed into a selection module, depicted as the rightmost block in Figure 1. The purpose of this module is to select the next token to incorporate into the solution, guided by the scores generated by the Transformer. The interaction between the network and the selection module mirrors the structure of a typical auto-regressive sequence generation process.

While various architectural modifications have been proposed to tailor such models to optimization tasks, a comprehensive review falls outside the scope of this article. We focus instead on the behavior of a pre-trained network, assumed to be capable of assigning relevance scores to candidate tokens for the purpose of extending a partial solution.

The primary goal of this article is to reexamine these two components —the neural network and the selection module— in the context of solving combinatorial optimization problems, using the Traveling Salesman Problem (TSP) as a testbed. This problem offers an ideal experimental ground: it is well-defined, widely studied, and backed by a vast body of literature. It is important to emphasize that our objective is not to propose a new high-performance method for solving the TSP, but rather to use it as a benchmark for exploring alternative designs in optimization solvers. Our goal is neither to propose a new neural network architecture nor to address the issue of inductive bias. We will use a pre-trained network and study whether it can be exploited in a different way.

1.1 The Traveling Salesman Problem (TSP)

The TSP is a classical combinatorial problem that involves finding the shortest possible tour allowing a salesman to visit a set of cities exactly once and return to the starting point. In graph-theoretic terms, this corresponds to finding a minimum-weight Hamiltonian cycle in a complete weighted graph. Despite its simple formulation, the TSP is NP-hard and remains a benchmark problem in optimization.

Exact methods can solve many instances efficiently, including the *Concorde* solver [7], which remains a reference. Though older, this software leverages integer linear programming solvers (such as *CPLEX* or *Gurobi*) and benefits from ongoing advances in that domain.

On the heuristic side, the POPMUSIC method introduced by [8] offers a very fast, quasi-linear complexity solution strategy that can generate large-scale random solutions. While these individual solutions are of modest quality, the diversity of edges covered across a small set of generated solutions captures a large portion of edges found in high-quality solutions. This property was exploited in [9] as a pre-processing step, leading to one of the most efficient methods to date in terms of both solution quality and computation time.

1.2 Neural Network Solvers: Promises and Limitations

Among the transformer-based approaches proposed for the optimization of routing problems, we can mention [10–15]. Among these references, [12] achieved the best solution quality for the TSP. This work is one of the most innovative solutions in AI for optimization. The authors of this reference have made the model they trained available. The numerical experiments reported in this paper therefore pay particular attention to this reference. Since we directly use a pre-trained model, all details regarding the architecture and training process are available in the original reference, ensuring full reproducibility.

However, despite their promise, these methods are not competitive with traditional solvers for several reasons:

- Training the network requires a substantial computational investment, often involving weeks of processing on massively parallel GPUs.
- Inference time increases quadratically with the number of cities, making the approach unsuitable for large-scale problems.
- Memory limitations constrain these models to instances with only a few hundred cities.

Despite these limitations, the transformer's time and accuracy performance is remarkably competitive for small instances (50–100 cities), which were the focus of the paper. In this context, the approach matches the performance of state-of-the-art classical methods — refined over more than 70 years — using a relatively simple and directly adapted transformer architecture, developed in a short time. Following the publication of this work, several subsequent studies [11, 14, 15] successfully applied similar architectures to other combinatorial problems, achieving comparable results and encountering analogous limitations. While neural network solvers are not yet practical for widespread use, this research direction is highly promising, especially given the rapid advancements in GPU hardware.

1.3 Research Questions Explored in This Study

This study explores the following questions:

- Can the heavy network training be replaced with a fast learning by identifying the most relevant tokens from concrete problem instances using classical optimization methods?
- Can the token selection strategy (currently based on greedy heuristics or beam search) be improved by drawing inspiration from other techniques like ant colony optimization?
- Can the solver from [12], trained on Euclidean instances of 50 and 100 cities uniformly distributed in the unit square, generalize to larger or structurally different instances?

A detailed response to the first question would require considerable work beyond the scope of this article. For example, training a neural network by biasing it with data

obtained from optimization heuristics would require studies on the network architecture and significant computational resources. In this paper, we will limit ourselves to utilizing an existing neural network solver and a classical, ultra-fast optimization method.

1.4 Structure of the Article

Section 2 explores how the POPMUSIC method can be used to identify edges likely to belong to a high-quality TSP tour. First, an empirical study assesses how many solutions need to be generated to cover all the edges in an optimal or near-optimal solution. Second, we analyze the quality of solutions obtained using only the frequency of edge occurrences from POPMUSIC-generated tours.

Section 3 investigates various edge selection strategies for constructing a solution. We start with a study of the limitations of the classical greedy nearest-neighbor approach and assess whether beam search significantly improves solution quality when using either edge length or frequency as selection criteria. We then propose improvements and introduce an alternative strategy inspired by ant colony algorithms.

Section 4 presents a synthesis of numerical results, comparing the performance of the transformer-based solver from [12], particularly on instances of up to 200 clustered cities —outside the model's original training distribution.

Section 5 concludes with a general discussion and outlines promising directions for improving the integration of neural models in combinatorial optimization.

2 POPMUSIC as a Direct Learning Method

POPMUSIC (Partial OPtimization Metaheuristic Under Special Intensification Conditions) is a general decomposition methodology for combinatorial optimization problems, introduced by [16, 17]. It is also part of the broader class of matheuristics, combining heuristic and exact optimization methods.

The core idea of POPMUSIC is partial optimization: instead of solving the entire problem, "parts" or sub-components of a solution are selected and optimized locally. These optimizations are iterated until no significant improvement is achieved, typically resulting in a local optimum. This approach is especially suitable for large-scale instances where global optimization is too expensive or impractical.

POPMUSIC has proven effective across a range of problems, including the capacitated vehicle routing problem (CVRP), location-routing problems, clustering problems, and graph problems such as the maximum weight stable set. Success hinges on judicious decomposition into relatively independent parts, around which meaningful sub-problems can be constructed.

In the TSP, POPMUSIC can be applied by considering each city as a part, and defining proximity between cities by the number of edges separating them in the tour. Sub-problems then involve optimizing sub-paths (consecutive segments of the tour), while keeping the endpoints fixed. This approach, studied notably by [9], enables rapid local improvements. More recently, [8] proposed a fast POPMUSIC heuristic of $O(n \log n)$ complexity for the TSP, capable of generating reasonably good solutions even for billion-city instances.

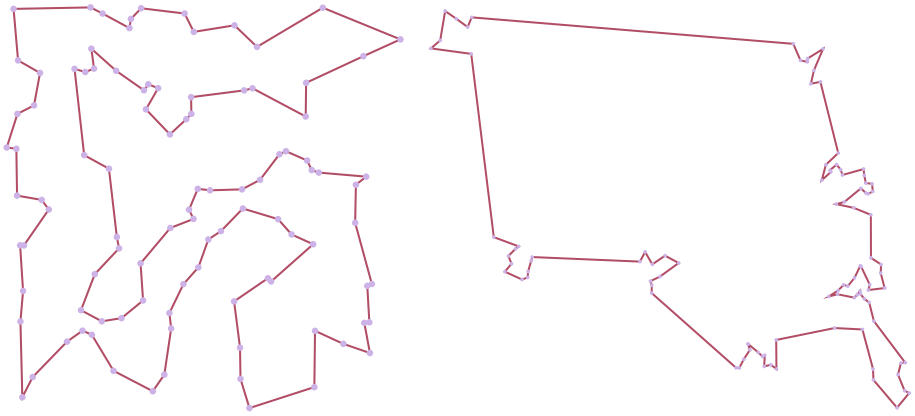

Fig. 2 An improved solution of 0.426% for uniform instance number 8534 with 100 cities published by [12] (left), and the best known solution for the first clustered instance with 100 cities generated for this article (right).

2.1 Methodology and Instance Generation

In this paper, we focus on small instances, as the neural network model used for comparison was trained only on instances of up to 100 cities. Accordingly, we generated datasets with 100 and 200 cities, for which near-optimal solutions could be reliably identified.

Two instance types were considered: uniformly distributed cities in a square, and clustered cities. For the latter, we randomly selected 10 or 20 cluster centers and then assigned each city to one of these centers with a small random offset. The square has size 100,000, and city coordinates are integers, allowing for unambiguous instance definitions. Euclidean distances were computed in double-precision floating point for tour quality evaluation. We generated 10,000 instances of each type for the 100-city problem and 1,000 for the 200-city problem. Instances with 50-city problem were also generated but ultimately excluded from the numerical results due to their simplicity, which hindered comparison with larger problems.

Each instance was solved 10 times using LKH software [18], with high parameters for the number of considered neighbors. Other randomized heuristics confirmed that the best solutions were consistently reproducible. While we cannot guarantee optimality, results strongly suggest that nearly all instances were solved optimally or very close to optimum.

These instances are available on https://mistic.iict-heig-vd.ch/taillard/problemes.dir/tsp/tsp_taillard.zip and provided with the best solutions found. Figure 2 compares the shape of the uniform and clustered instances.

Why not using existing benchmarks?

For the instances with uniformly distributed cities, we could have reused those from [12]. On the one hand, using new instances allowed us to perform a cross-validation of the results reported in that reference. On the other hand, the approach presented

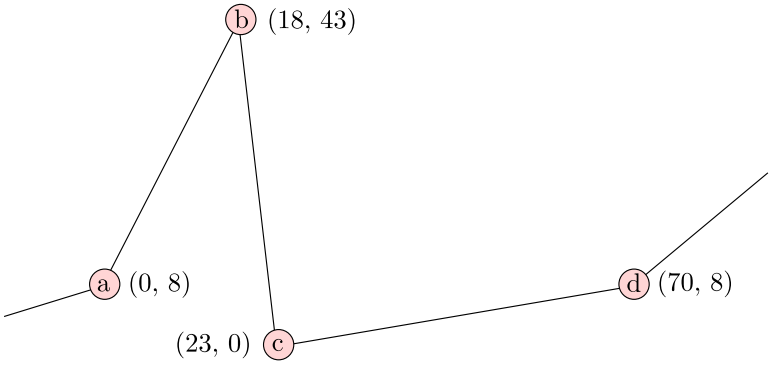

Fig. 3 Length of path a–b–c–d is lower than that of a–c–b–d, but the difference lower than 10^{-7}.

above appears more reliable than using "exact" solvers like *Concorde*, which assume integer distances for eliminating branches that improve the solution by less than one unit.

Indeed, by comparing the lengths of the "optimal" solutions reported by [12] with those obtained through heuristic methods, it became apparent that the proportion of instances *not* optimally solved in this reference is approximately 36.5% for the 50-city instances, 73.4% for the 100-city instances, and 97.3% for the 200-city instances. The gap to the optimum can be as large as 0.4% for some instances. Using the tour lengths from [12] could therefore skew the results provided.

Figure 3 illustrates a situation in which it is difficult to determine, using only 64-bit integer representations, whether the city ordering a–b–c–d or a–c–b–d is preferable. Given that the difference in total length is less than 10^{-7}, a naive approach might involve scaling all coordinates by a factor of 10^8 to ensure that the difference exceeds one unit. However, this would lead to integer overflows during the computation of hypotenuses.

The LKH solver with default parameters finds solutions on average 0.0006% above the best-known solutions for uniformly generated 100-city problems. LKH finds the best-known solution in 99.18% of cases. These values are significantly better than the results obtained with *Concorde*, as reported by [12], for similar instances (0.045% gap, best-known solution found in 26.6% of cases).

For clustered instances of the same size, the results with LKH and default parameters are slightly worse. The best-known solutions are found in 96.67% of cases, with an average gap of 0.0076%. Solving the 10,000 clustered instances with 100 cities using LKH (10 runs per instance) takes about 1 hour (sequential execution on an i7 6700K CPU).

LKH scales very well. For uniform TSP instances with 200 cities, it finds the best-known solution in 99.3% of cases, with an average optimality gap of 0.00034%, requiring approximately 24 minutes to solve 1,000 instances. For clustered TSP instances of the same size, the best-known solution is found in 86.4% of cases, with an average gap of 0.037%, and a total computation time of about 42 minutes for 1,000 instances (10 runs per instance).

	Unif. 100	Clust. 100	Unif. 200	Clust. 200
Time [ms, i7 6700K]	0.34	0.48	1.43	2.04
Average deviation [%]	2.37	0.93	3.06	1.30
Deviation, best of 100	0.085	0.0065	0.73	0.056

Table 1 Results for POPMUSIC with parameters as suggested in [8]: Computational time to produce a solution and relative deviation to best solution. Instances with 100 and 200 cities, uniformly generated in a square or clustered.

	Unif. 100	Clust. 100	Unif. 200	Clust. 200
Time [ms]	0.16	0.22	0.39	0.53
Average deviation [%]	4.34	2.53	7.37	8.08
Deviation, best of 100	0.55	0.045	2.92	0.297
Correct edges[%]	77.5	84.5	73.9	81.4

Table 2 Results for degraded POPMUSIC. Sub-path with only 50 cities for $n = 100$ and 64 cities for $n = 200$. Computational time, deviation to best solution and proportion of edges of the best solution identified.

2.2 Solution Quality from POPMUSIC

The fast version of POPMUSIC with parameters recommended in [8] (optimizing 100 to 200-city paths using Lin-Kernighan neighborhood) can very quickly generate high-quality solutions. Selecting the best among 100 runs —which takes only fractions of a second— typically yields solutions within 1% of the optimum, as shown in Table 1. For clustered instances, near-optimal solutions are achieved in most cases.

As shown in [8], POPMUSIC-generated solutions for uniform 10^9-city instances deviate by about 10% from the optimum. In the present paper, we aim to assess the extrapolation capabilities of a direct learning approach to larger instances. Since POPMUSIC was designed to handle much larger instances than those typically addressed by neural solvers, we intentionally degraded the performance of POPMUSIC by limiting the length of the optimized sub-paths. Table 2 reports the same statistics as before but for this degraded algorithm. Additionally, we have included a row indicating the proportion of generated edges that match those of the best solutions known.

Tables 1 and 2 provide the computation times of POPMUSIC for reference. The goal of this paper is not to compare these times to those of the solution produced by the solver of [12]. Interested readers can find them in that reference. It should be noted that a comparison is difficult because the times indicated in Tables 1 and 2 are relative to a sequential execution using a single CPU core, while those of [12] are for a highly parallelized execution on a GPU.

Figure 4 shows the proportion of instances for which all edges of a near-optimal solution appear in the union of edges from p POPMUSIC-generated solutions. The results indicate that edges covered by 100 solutions almost surely ($> 99\%$) include all those from the best solution, even though none of the individual solutions are optimal.

These results suggest that the approach effectively identifies promising edges in Euclidean TSP problems —a property exploited in recent versions of the LKH solver.

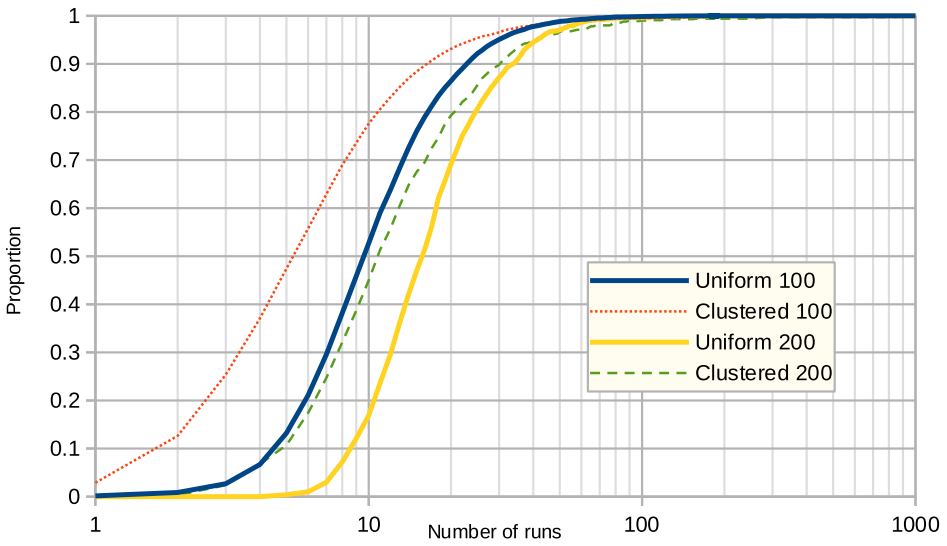

Fig. 4 Proportion of instances for which all edges of sub-optimal solution are found, as a function of the number of degraded POPMUSIC solutions generated.

Cities	Distr.	POPMUSIC		Consensus 20		Consensus 100		Consensus 1000	
		Mean	Median	Mean	Median	Mean	Median	Mean	Median
100	uniform	4.34	4.31	1.22	1.01	0.973	0.728	1.31	1.01
100	cluster	2.53	2.42	0.551	0.111	0.460	0.049	0.528	0.046
200	uniform	7.37	7.34	1.44	1.29	1.10	0.872	1.72	1.42
200	cluster	8.08	8.12	1.82	0.498	1.47	0.329	1.92	0.535

Table 3 Mean and median deviation (expressed in percent above best solution known) for degraded POPMUSIC, and consensus(p) for $p = 20, 100, 1000$

2.3 Consensus Solution

To assess the potential of this approach as a learning method, we evaluated the quality of solutions obtained by considering only edge occurrence frequencies from POPMUSIC solutions. We define consensus(p) as the solution maximizing the sum of edge occurrence frequencies over p generated solutions.

Table 3 compares deviations from a near-optimal solution for three values of p, contrasting them with results from the degraded POPMUSIC version. Both mean and median deviations are provided. For non-uniform instances in particular, the median is often significantly lower than the mean, indicating a strong asymmetry in the result distribution—as illustrated in Figure 5, which provides the proportion of runs deviating from the best-known solution by less than a given value.

It is noteworthy that the consensus solutions obtained from 1,000 individual solutions are not superior to those derived from only 20 solutions. Figure 4 shows that

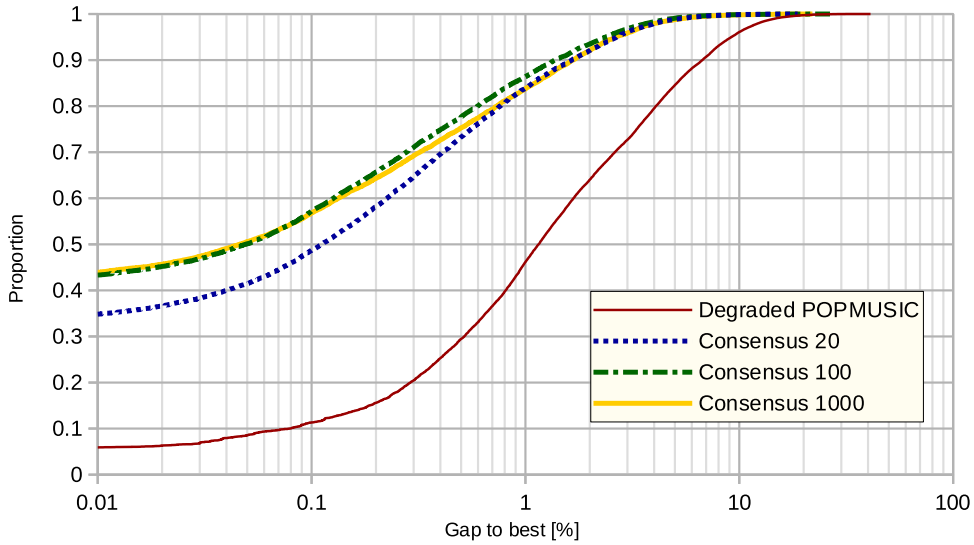

Fig. 5 Proportion of runs deviating from the best-known solution by less than a given value for 10,000 TSP instances with 100 cities and 10 clusters. The instances have been solved with degraded POPMUSIC (10 clusters, max path length 50) and consensus(p) for $p = 20, 100, 1000$

.

with 20 solutions, between 7% and 30% of the instances exhibit missing optimal edges. With 1,000 solutions, nearly all edges of the best solution are generated; however, a significant number of suboptimal edges are also introduced. This additional "noise" degrades the quality of the consensus solutions.

Although we did not perform a fine-tuned optimization of the parameter p, setting it to 100 yields consensus solutions that are marginally better than those obtained with either 20 or 1,000 solutions.

The quality of the resulting solutions is quite respectable, especially considering that they were generated without taking distances into account. This supports the idea that solution generation using POPMUSIC closely resembles a learning process.

3 Sampling techniques

Among the approximate techniques for combinatorial optimization (CO) problems, two main families can be identified: local search methods and tree search methods [19]. Local search methods explore the search space by transitioning from one complete solution to another, guided by a local neighborhood structure. In contrast, tree search methods follow a constructive approach, modeling the search space as a tree in which each path from the root to a leaf corresponds to the step-by-step construction of

	Unif. 100	Clust. 100	Unif. 200	Clust. 200
Nearest Neighbor	24.71	19.92	25.76	22.95
Best Insertion	4.65	1.67	5.88	1.83

Table 4 Average gaps in percent off the best solution for greedy heuristics on different instance classes.

a solution. The space is explored by incrementally extending partial solutions — each node in the tree— one token at a time. This process is repeated until complete solutions are reached.

The simplest way to choose the next token to add to a partial solution is through a greedy strategy: selecting the token assigned the highest score by the transformer model. To clarify and establish notation, greedy search begins with a partial solution s^p_{init}, which is trivially initialized. For the TSP, two strategies are commonly used to build a solution. Either the partial solutions are open paths and the initial partial solution s^p_{init} is just a single starting city (often chosen arbitrarily), or the partial solution are tours and s^p_{init} is initialized with two cities (for example, the two closest or the two farthest apart).

At each step, the partial solution s^p is extended by an element c^* from the set $\mathcal{N}(s^p)$ of candidates not yet included in s^p. The element c^* maximizes a scoring function $\eta(c, s^p)$, which evaluates the suitability of adding c to the current partial solution. This process is repeated until a complete solution is constructed.

For the TSP, using the first strategy (open path), the function $\eta(c, s^p)$ can be the inverse of the distance between the last city in s^p and candidate city c. This corresponds to the nearest neighbor heuristic. With the second strategy, $\eta(c, s^p)$ may quantify the inverse of the additional detour required to insert city c at the best possible position between two cities already in the partial tour —this is known as the best insertion heuristic.

In this paper, we focus on the nearest neighbor heuristic, as it is the one employed in the transformer-based TSP solver proposed by [12]. It should be noted, however, that this greedy heuristic is far from optimal. Table 4 compares the average performance of the nearest neighbor and best insertion heuristics, showing that the latter yields significantly higher-quality solutions. Given the transformer's ability to learn from nearest neighbor-based construction, it would be valuable to investigate its performance when trained using an insertion-based heuristic. Such a shift would likely be non-trivial, as tokens would then need to encode not only the city to add but also the position within the partial tour.

The effectiveness of any greedy method heavily depends on the design of the scoring function $\eta(c, s^p)$. Ant Colony Optimization (ACO) aims to improve this function by incorporating an artificial pheromone component associated with each candidate element, independently of the current partial solution s^p, and updated via a dedicated learning mechanism. In the TSP context, a pheromone value τ_e is associated with each edge e, and the scoring function becomes

$$\eta(c, s^p) = \tau_e^a / d_e^b$$

where e is the edge connecting the last city in s^p to city c, d_e is the weight of edge e, and the powers a and b are tunable parameters.

Except for certain simple problems (e.g., minimum spanning tree, shortest path), where greedy algorithms yield exact solutions, their main limitation lies in lack of foresight. During early stages, decisions are guided by high scoring elements, but the quality often degrades as the set $\mathcal{N}(s^p)$ shrinks. This myopic behavior can be mitigated in several ways:

3.1 Randomization

A common enhancement to greedy constructive heuristics is to inject randomness, generating multiple candidate solutions and randomly selecting one of the best among them. This approach is employed in GRASP (Greedy Randomized Adaptive Search Procedure, [20]) and ACO (Ant Colony Optimization, [21]). GRASP selects the next element randomly from the top α candidates (according to $\eta(c, s^p)$), where α is a tunable parameter. ACO methods select candidate c with a probability proportional to its score $\eta(c, s^p)$ (also known as *Bernoulli sampling* in machine learning), until a complete solution is built. This is repeated N_a times (by N_a "ants"), then the pheromone trail parameters are updated based on the quality of the solutions, and the algorithm starts again. This iterates until a stopping condition is reached, e.g., a fixed number of iterations, and the best overall solution is selected. In the simplest implementation of ACO, there is no update of the pheromone parameters at all, the probabilistic nature of the algorithm is enough to ensure exploration of the search space.

3.2 Beam search

Instead of relying on randomness to explore the search space, beam search systematically expands the partial solution s^p by selecting the top k_{ext} candidates at each step, according to the scoring function η. At each iteration, up to k_{bw} partial solutions are retained in the beam, where k_{bw} denotes the *beam width*. These are selected based on a possibly distinct ranking function γ that evaluates the quality of partial solutions, while all others are discarded. Each retained solution is then expanded again by its k_{ext} most promising continuations in the subsequent step. This process continues until complete solutions are obtained, from which the best one can be selected from the final beam.

Beam search can thus be viewed as a generalization of greedy search, resembling a breadth-first search with restricted width, and shares conceptual similarities with branch-and-bound techniques.

Although beam search is widely used in fields such as scheduling, natural language processing, and other areas, many sources —especially within the machine learning literature [22–24]— consider only a restricted form of beam search. In particular, the influential work on transformer-based TSP solvers [12] employs a version where $k_{ext} = k_{bw}$, i.e., the number of expansions per step equals the beam width.

For concreteness, we present the full beam search algorithm in Algorithm 1, very similar to how it appears in [19]. After the beam search is complete, the best solution from the final beam $\mathcal{B}_c$ can be selected using γ or a different metric. The $\gamma(s^p)$ value

could be an upper bound on some metric for a complete solution obtainable from the partial solution s^p. Alternatively, for example in machine learning applications, $\eta(c, s^p)$ could be the probability of the option c (e.g. a word for natural language processing) as given by the model, and, setting $\gamma(s^p_{init}) = 1$, we can inductively define

$$\gamma(s^p \text{ extended by } c) = \gamma(s^p) \cdot \eta(c, s^p)$$

Very often, $\eta(c, s^p)$ is the log-likelihood instead of the probability of completing the partial solution s^p with option c. Then $\gamma(s^p_{init}) = 0$ and $\gamma(s^p \text{ extended by } c) = \gamma(s^p) + \eta(c, s^p)$.

In practice, Algorithm 1 can be implemented with a priority queue for $\mathcal{B}_{ext}$ to improve performance.

Algorithm 1 Full beam search

Require: A trivial partial solution s^p_{init}, beam width k_{bw}, maximum number of extensions k_{ext}, function η to rank extensions, function γ to rank partial solutions, $\mathcal{B} \leftarrow \{s^p_{init}\}$, $\mathcal{B}_c \leftarrow \emptyset$

Ensure: A set of candidate solutions $\mathcal{B}_c$

while $\mathcal{B} \neq \emptyset$ **do**
  $\mathcal{B}_{ext} \leftarrow \emptyset$
  for $s^p \in \mathcal{B}$ **do**
    $count \leftarrow 1$
    while $count \leq k_{ext}$ **AND** $\mathcal{N}(s^p) \neq \emptyset$ **do**
      $c^* \leftarrow \arg\max_{c \in \mathcal{N}(s^p)} \eta(c, s^p)$
      $s^{p*} \leftarrow$ extend s^p by c^*
      $\mathcal{N}(s^p) \leftarrow \mathcal{N}(s^p) \setminus \{c^*\}$
      if s^{p*} is extensible **then**
        $\mathcal{B}_{ext} \leftarrow \mathcal{B}_{ext} \cup \{s^{p*}\}$
      else
        $\mathcal{B}_c \leftarrow \mathcal{B}_c \cup \{s^{p*}\}$
      end if
      $count \leftarrow count + 1$
    end while
  end for
  $\mathcal{B} \leftarrow$ select the $\min(k_{bw}, |\mathcal{B}_{ext}|)$ best partial solutions from $\mathcal{B}_{ext}$ according to γ
end while
return $\mathcal{B}_c$

4 Numerical Results

This section starts by specifying the settings used for the various construction methods. The edge frequencies obtained with POPMUSIC can be exploited either deterministically with beam search, or probabilistically with an ant system-like

procedure. These two constructive methods can also be used based on the probabilities obtained from the transformer decoder. This section concludes with a table summarizing the performance of these different options.

4.1 POPMUSIC Constructions

For the numerical experiments with POPMUSIC outputs, we consider an η-function of the form $\eta(c, s^p) = \tau_e^a / d_e^b$, where a and b are two parameters, e denotes the edge connecting the last city of the partial tour s_p to city c, d_e is the length of that edge, and τ_e quantifies the usefulness of including edge e in a solution. The value of τ_e equals the empirical frequency with which edge e appears in 100 degraded POPMUSIC solutions.

We examine two ways of exploiting η: deterministic beam search and a probabilistic ant-colony procedure. Beam search is governed by two parameters, k_{ext} and k_{bw}.

The classical nearest-neighbor heuristic is recovered with $a = 0, b > 0$ and $k_{ext} = k_{bw} = 1$, while a simple beam search using only the distance information is obtained with $a = 0, b > 0$ and $k_{ext}, k_{bw} > 1$.

4.1.1 Beam-Search Parameter Tuning

We first studied the influence of k_{ext} and k_{bw} when τ_e is given by POPMUSIC edge frequencies. Experiments began on 50-city instances to speed up computation, but the findings did not transfer to larger problems. This is why we discarded these small instances in our numerical results.

Originally, for a partial solution s^p and candidate city c, we employed the γ function

$$\gamma(s^p \text{ extended by } c) = \gamma(s^p) + \eta(c, s^p)$$

throughout the tour construction, and also to select the best complete tour from the final beam $\mathcal{B}_c$ given by the algorithm. That is, we selected the solution that maximized the sum of the η values along the tour. With this approach, we found heavy dependence on the value of k_{ext} for a fixed k_{bw}, with low k_{ext} giving much better results, the best obtained at $k_{ext} = 2$, while at the same time running significantly faster. While using the same γ for final solution selection is the only sensible option for some problems, such as in natural language processing, for the TSP a much more natural choice is to simply select the shortest tour from the final beam. This was also done by [12]. This approach gave even better results, and the difference in performance between different values of k_{ext} was minimal.

Tree Parzen Estimators (TPE) were used to tune a and b. We found that it is usually preferable to rely solely on edge frequencies and ignore lengths ($b = 0$); however, the differences must be heavily smoothed by choosing a very small a (e.g., $a = 0.1$). Clustered instances are generally easier than uniform ones. Table 5 reports the numerical values obtained with $k_{ext} = k_{bw} = 1000$ for the two parameter sets $(a, b) = (0.1, 0)$ and $(0, 1)$ (i.e., the nearest neighbor heuristic). Overall, beam-search performance is far from satisfactory, except perhaps on very small instances, and —surprisingly— can be worse than the greedy procedure it is supposed to refine.

Intuitively, this behavior can be explained as follows: Beam search starts with an arbitrary city and discovers a multitude of paths where the first edges are indeed

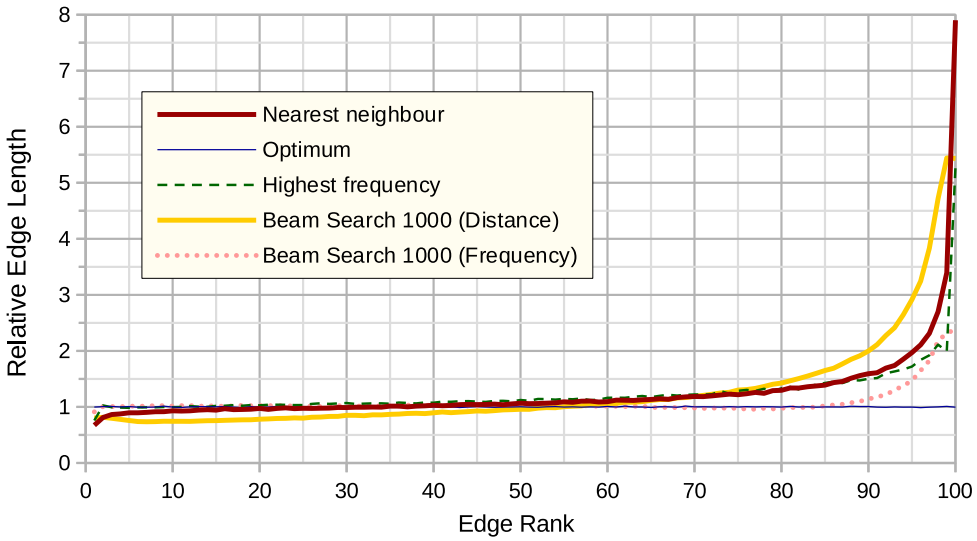

Fig. 6 Uniform TSP with 100 cities. Average length of the edges produced by heuristics as a function of their position in the solutions obtained. These lengths have been normalized relative to the average length of the edges in the optimal solutions. "Highest frequency" refers to a greedy algorithm that works with the frequency of the edges produced by POPMUSIC.

shorter, on average, than those of the optimal solution. However, since the number of these paths is limited by the beam width, paths that begin with sequences similar to those of the optimal solution may be eliminated as more cities are added. Towards the end of the algorithm, the choice of cities becomes increasingly limited, forcing poor choices. In other words, the optimal length of a path over the first half of the cities, added to the optimal length of a path over the remaining half of the cities, results in a longer value than that observed by a simple greedy method.

This behavior is clearly shown in Figure 6, which gives the average length of the ith edge of the solution, relative to the average length of an edge from the optimal solutions. Rank 1 represents the first edge added to the solution, and rank 100 represents the last. We can see that the first edges added by all heuristics are indeed shorter than those of the optimal solutions. This is true for both length-based criteria and frequency of appearance of the edge obtained with POPMUSIC. However, the last edges are significantly longer. For example, the last edge added by the nearest neighbor heuristic is nearly 8 times longer. With Beam search based on distances, the last edge is "only" 5 times longer, but the previous 20 are considerably longer than those of the nearest neighbor heuristic.

So, even with $k_{ext} = k_{bw}$, beam search is short-sighted. For example, consider a TSP with just 50 cities and a large $k_{bw} = 2256$ ($2256 = 48 \times 47$). Let's assume that city the starting city is fixed. In the first step, all options are explored, and all 49 are kept in $\mathcal{B}$. In the second step, we explore all 49×48 options and begin pruning. By

the third step, in the worst-case scenario, it may happen that, after pruning, we are left with 48×47 partial solutions in $\mathcal{B}$, all of which have the same first edge. In this case, the first edge becomes fixed, and no subsequent information can change it.

With a minimal $k_{ext} = 2$, only the two best extensions are kept per branch, so with the same k_{bw}, in the worst case, the first edge is fixed only after 11 steps, allowing a more diverse exploration of the search space. This is an extreme example, but $\mathcal{B}$ gets filled very quickly for large instances.

4.1.2 Ant System Randomized Solution Construction

For probabilistic construction, the optimal (a, b) values differ markedly from those for beam search. We retained $a = 17, b = 7$ for the results in Table 5. To compare fairly with beam search and the neural solver, we generated 1000 solutions per instance and kept the best. This ant-colony-style mechanism yields solutions that clearly outperform both the greedy heuristic and beam search, yet a non-negligible gap to optimality remains —especially on uniformly generated instances. Hence, the good performance reported for ant-colony optimization in the literature is likely due less to the pheromone-learning mechanism than to the systematic application of local search. Indeed, the consensus solutions in Table 3 indicate that relying solely on edge-frequency information can potentially lead to much better tours.

4.2 Construction with a Transformer

The neural network was trained on uniform 100-city instances in the unit square; we employed the model released by [12]. An initial test on fresh 100-city uniform instances produced very poor results, as the network appears unable to handle homothetic scaling. After rescaling coordinates to lie within the unit square, we were able to reproduce solutions whose average quality matches impressive performance published by [12].

It was interesting to see whether a model trained on uniform instances could generalize to data coming from a different distribution, namely "clustered" instances, where majority of cities are clustered around a few centers, sparsely populated in between them.

Similarly, we also tested the model trained on instances of size $n = 100$ on instances of size 200 to see whether these models can generalize to larger instances than what they saw in training.

The results obtained with the probabilities produced by the model are summarized in the 3 last rows of Table 5. These results are both encouraging and disappointing. On the one hand, the model provided sensible solutions for these new types of instances it never saw in training. They are of better quality than the simplest heuristics like nearest neighbor would provide (roughly $20-30\%$ off optimum for these sizes). On the other hand, the performance is an order of magnitude worse than with the uniform $n = 100$ instances that the model was trained on, and very far off the best heuristics which provide near optimal solutions quickly. Additionally, very costly (taking about 500 times longer) beam search could only marginally improve on the greedy solution: 9.74% off optimum with $k_{bw} = k_{ext} = 1000$ vs 10.4% for the greedy solution on average for clustered $n = 200$ cities instances.

Based on initial findings discussed in Section 4.1.1, we extensively experimented with beam search using smaller k_{ext} instead of the original $k_{ext} = k_{bw}$, but did not find any significant improvement over the original settings.

4.2.1 Ant Systems

As an alternative to beam search, we explored "ant systems" sampling, i.e., simple Bernoulli sampling of the transformer decoder output probabilities, done once for each city in the sequence. The construction of a whole tour is repeated many times (e.g. $N_a = 50$) in parallel (to save time) or sequentially (to save GPU memory) and simply select the shortest overall length.

For $n = 50$ clustered cities, ant systems were achieving good performance already for $N_a = 20$ ants: a gap of 1.69% to the best solution, c.f. 2.68% for greedy decoding. However, additional improvement came at a high computational cost.

Top-p sampling

We also tried *top-p* sampling with $p = 0.9$ that is nowadays widely used by modern (also transformer-based) LLMs instead of the simple Bernoulli sampling. Top-p sampling sorts the outputs in descending order of probabilities, then adds up the probabilities until a threshold p is reached (usually something like $p = 0.9$). When the threshold is reached, we keep only the top ones that contributed to the sum, readjust the probabilities and sample. This avoids ever selecting the "very bad" choices that have very low probabilities. Over long sequences like $n = 50$ and higher, for simple Bernoulli such "bad" selection will happen at least once with probability 0.995. Interestingly, top-p sampling was performing worse for us than simple Bernoulli sampling and is thus excluded from final results.

We believe the latter, as well as high cost in the number of ants for minimal improvement of simple Bernoulli came because of highly concentrated (and not very good) probability outputs of the transformer decoder. The probability distributions are very sharp and as a result ant systems don't get a lot of variety between each other, preventing them from obtaining better solutions. This can be mitigated using temperature scaling.

Temperature sampling

Temperature sampling adjusts the logits that the neural net outputs before they are converted into probabilities by multiplying them by a "temperature" T. $T < 1$ makes the distribution sharper, $T > 1$ makes it flatter. LLMS use $T < 1$ when an exact answer is needed, like when asking for a fact, and $T > 1$ when more diversity in the answer is wanted, like for a creative, non-repetitive paragraph. For us, $T > 1$ should give more diversity in the obtained tours. Results show best performance with $T = 2$ for uniform 100 cities that the model was trained on, but $T = 1$ (i.e simple Bernoulli sampling) for clustered 100-city instances, $T = 0.5$ for uniform 200-city instances, $T = 0.15$ for clustered 200-city instances.

$\eta - function$	Sampling technique	Unif. 100	Clust. 100	Unif. 200	Clust. 200
d_e	Greedy	24.71	19.92	25.76	22.95
d_e	Beam search	25.95	20.45	31.1	22.6
$\tau_e^{0.1}$	Beam search	6.35	6.08	21.9	33.3
d_e^{17}/τ_e^7	Ants, best of 1000	4.00	**1.13**	9.16	**4.65**
p^{dec}	Greedy	1.45	12.1	5.68	10.4
p^{dec}	Beam search	0.46	7.45	4.18	9.74
p^{dec}	Ants, best of 1000	**0.45**	5.25	**3.7**	8.77

Table 5 Average gaps in percent off best solution for various approaches on different instance classes. Beam search was run with $k_{bw} = k_{ext} = 1000$.

Computational experiment summary

Table 5 summarizes the results for the different approaches. The first column shows the η-function used to rank the options for the next city to add to the partial tour, d_e being the edge distance, τ_e the edge frequency from POPMUSIC, and p^{dec} the probability from the transformer decoder. The second column shows the sampling technique that used the corresponding η-function to build a solution. Thus, the first row represents the greedy nearest-neighbor heuristic; the second row simple beam search using just the distance information; third and fourth rows represent the best results with beam search and ant systems using POPMUSIC frequencies; and the last three rows represent the greedy, beam search, and ant systems using the transformer decoder probabilities. Temperature scaling as described above was used for the transformer decoder probabilities in the ant systems case.

Some of the averages given in Table 5 can be directly compared to those provided in other references. However, they offer only a partial view of the results obtained. To complement them, we provide graphically the cumulative proportion of solutions obtained by the various methods, based on the deviation from the best-known solutions. Figures 7 to 10 show these proportions for the 4 types of instances. These figures can be used, for example, to obtain the median results by reading the value on the horizontal axis where the curves reach a proportion of 0.5.

The outcomes, summarized in Table 5, lead to the following observations:

- The transformer handles uniform instances substantially larger than those seen in training quite satisfactorily.
- The η values returned by the transformer enable greedy construction of markedly better solutions than those based on a mix of edge frequencies and lengths.
- Performance degrades on clustered instances —as expected, given their absence from the training set— but remains respectable relative to the nearest-neighbor heuristic.
- Replacing beam search with randomized construction further improves the solutions obtained.

Why transformers provide better token values?

Using greedy search (argmax) as the sampling technique, the transformer achieves solutions within less than 1.5% of the optimum for uniform 100-city instances in a

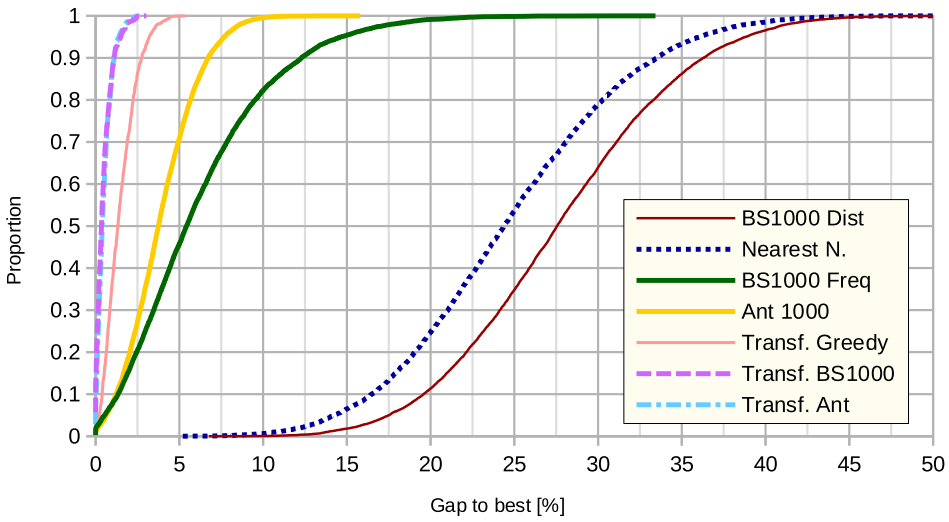

Fig. 7 Proportion of runs deviating from the best-known solution by less than a given value for TSP with 100 cities uniformly distributed within a square. The following abbreviations are used: *BS1000 Dist*: Beam Search based on edge length with a beam width of 1000. *Nearest N*: Nearest Neighbor (based on edge length). *BS1000 Freq*: Beam Search based on edge frequency with a beam width of 1000. *Ant1000*: Best solution obtained from 1000 randomized constructions with probability proportional to d_e^{17}/τ_e^7. *Transf. Greedy*: Solution obtained by a greedy construction based on probabilities produced by the transformer. *Trans. BS1000*: Beam Search based on probabilities produced by the transformer with a beam width of 1000. *Transf. Ant*: Best solution obtained from 1000 randomized constructions with probability produced by the transformer.

.

fraction of a second. This stands in stark contrast to the nearly 25% optimality gap of the nearest-neighbor heuristic, the classical equivalent of greedy search.

We attribute this superior performance to the auto-regressive nature of the transformer model. As the route is constructed, the model receives the entire sequence of already visited cities, in order, as input at each step. Through reinforcement learning, the transformer learns to dynamically rank candidate cities so as to minimize the total tour length. Crucially, model weights are updated only when an improvement is found during training.

This approach contrasts sharply with classical methods like nearest-neighbor or beam search, which rely on static η-functions for city-pair rankings. In those methods, the partial tour cannot influence the η-rankings, as they are fixed from the start. In the transformer, however, η-rankings are re-evaluated at every step, allowing for adaptive, context-aware decision-making.

We believe this auto-regressive mechanism is also responsible for the transformer's robust generalization to clustered and larger instances — never seen during training — which can be further enhanced using randomized sampling, as demonstrated in our

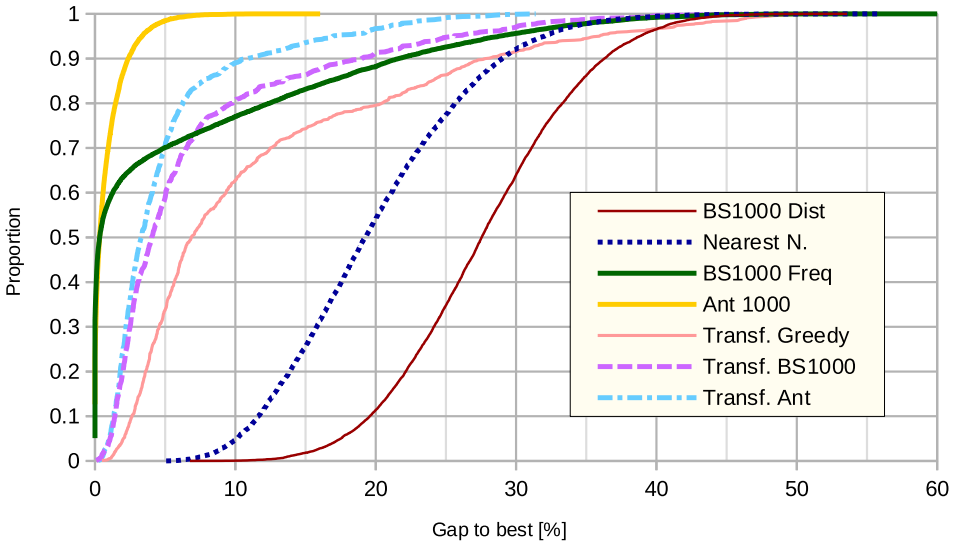

Fig. 8 Proportion of runs deviating from the best-known solution by less than a given value for TSP with 100 cities in clusters.

.

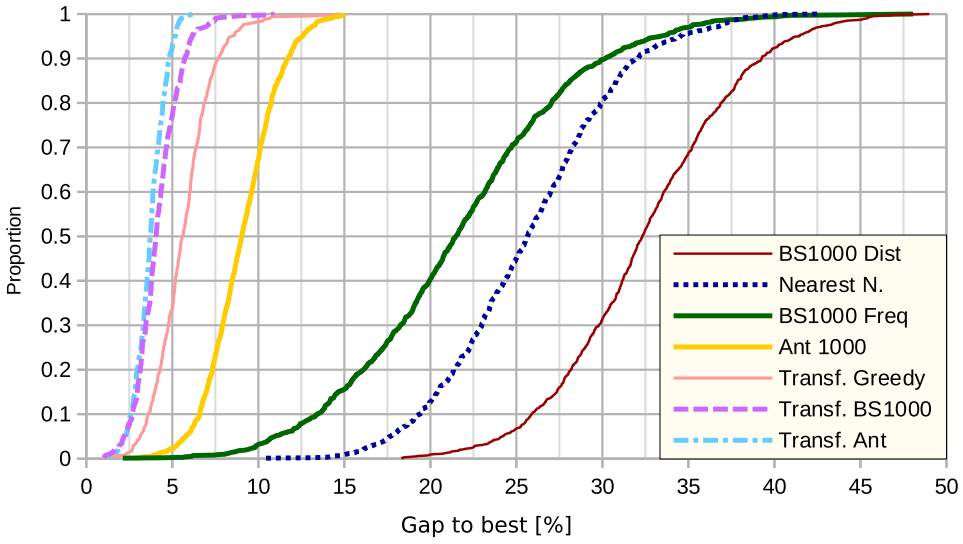

Fig. 9 Proportion of runs deviating from the best-known solution by less than a given value for TSP with 200 cities uniformly distributed within a square.

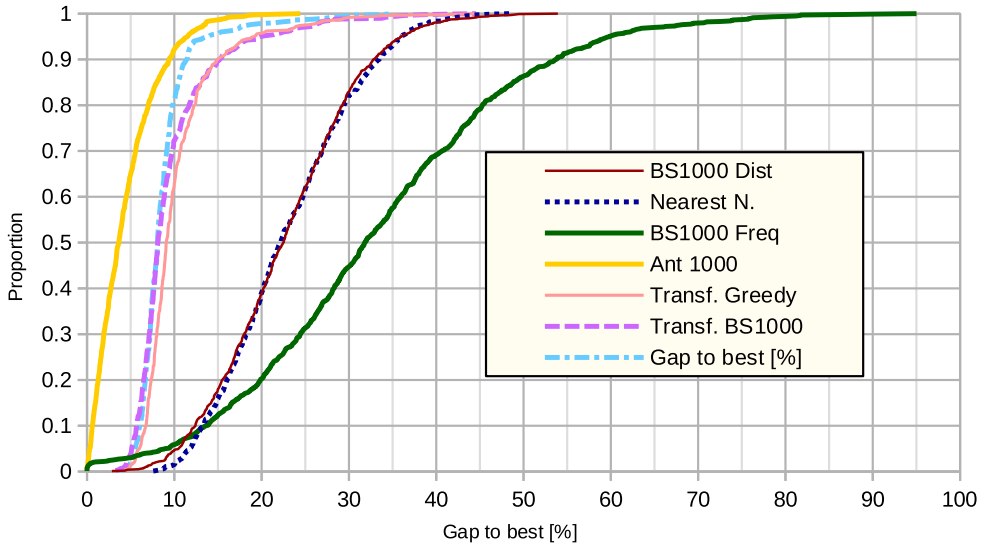

Fig. 10 Proportion of runs deviating from the best-known solution by less than a given value for TSP with 200 cities in clusters.

experiments. Beyond this, transformers (and neural networks in general) are renowned for their generalization capabilities [25], a key factor in their widespread success.

Computational time and algorithmic complexity

POPMUSIC and the Transformer-based solver use very different hardware. POPMUSIC has been implemented for a single CPU thread. It has a time complexity of $O(n \log n)$ and linear memory complexity.

The empirical complexity of LKH, also implemented for a single CPU thread, depends on the selected computation options. With the default options, the time complexity is quadratic. For Euclidean problems, the time complexity can be reduced to $O(n \log n)$ by computing a Delaunay triangulation. The latest versions of LKH integrate POPMUSIC. By choosing this option, the complexity can empirically be reduced to $O(n \log n)$, even for non-Euclidean problems. LKH options allow specifying the reading of a distance matrix. Using such options necessarily implies quadratic complexity, both in time and memory.

The constructive algorithms (nearest neighbor, ant, transformer) have a quadratic time complexity. This complexity must be multiplied by k_{bw} when beam search is used. A parallel implementation of beam search can use k_{bw} processors, achieving an acceleration of the same order of magnitude. This is done in [12], who mentions an inference time of $O(Ln^2)$ for a problem with n cities, using a neural network with L layers.

5 Conclusions

This paper demonstrates that generating TSP solutions using POPMUSIC resembles a form of rapid learning. Within a fraction of a second, and for instances with a few hundred cities, it is possible to generate data reflecting approximately 80% of the edges present in optimal solutions. However, effectively leveraging this data within constructive procedures remains challenging.

Our experiments with the transformer model trained on uniform 100-city instances reveal strong generalization capabilities:

- It produces competitive solutions for instances twice as large.
- It handles non-uniform instances —absent from the training set — with respectable performance relative to the nearest-neighbor heuristic.
- The token scores output by the transformer enable greedy construction of significantly better solutions than those based on edge frequencies and lengths alone.

Constructive Methods: Beam Search vs. Randomized Approaches

Without relying on local search, the most promising constructive strategy identified is a randomized method inspired by artificial ant systems. In contrast, beam search proves effective only for very small instances and becomes inefficient when edges are selected based solely on length, requiring more computational resources and potentially degrading solution quality. Neural solvers, however, output token scores that can be effectively exploited via beam search, suggesting they have learned more relevant decision criteria than traditional greedy heuristics. This insight motivates further investigation into how such scoring mechanisms are constructed and how they might inspire the design of more effective constructive heuristics.

Future Directions: Randomized Construction as a Key Improvement

The most promising research direction is to replace beam search with randomized construction methods, such as those used in ant colony systems. Our TSP experiments show that randomized construction, for the same computational effort, yields better solutions on average. Moreover, exploring neural architectures that not only predict the next token to append but also determine its optimal insertion position could further enhance performance. For the TSP, greedy heuristics based on "best insertion" already outperform "nearest neighbor" approaches, and similar principles could be integrated into neural solvers.

Limitations and Scalability Challenges

We have demonstrated that the transformer model trained by [12] on uniformly distributed 100-city instances exhibits strong generalization capabilities. Specifically, it produces competitive solutions on instances twice as large, as well as on non-uniform instances not encountered during training.

While neural methods show potential, traditional algorithms remain more efficient for the TSP in terms of both computational complexity and solution quality. Training and inference for larger instances (e.g., 200+ cities) pose significant challenges, and

current GPU limitations (8GB memory) restrict parallel processing. Scalability to instances with thousands of cities remains an open question. Furthermore, training on non-uniform instances was beyond the scope of this work. Future research should investigate whether networks trained on diverse instance types can maintain solution quality.

Broader Applications and Generalization

The framework presented here can be adapted to other combinatorial optimization problems. Neural transformers could optimize production and logistics processes, especially where traditional methods struggle with complex or automatically collected data. As data availability grows, continuous improvement of neural networks becomes increasingly feasible. Other studies [11, 14, 15] have already applied neural networks to various routing problems. It would be valuable to explore whether the observations made for the TSP generalize to other combinatorial optimization challenges.

Acknowledgements. Not applicable

Author contributions. All authors contributed equally to this manuscript

Funding. This manuscript was partially supported by HEIG-VD, Grant numbers HES-SO 128609 and 137143.

Data availability. The datasets used for the numerical experiments are hosted on a publicly accessible webpage.

Competing interests. There are no potential competing interests in the manuscript.

References

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention Is All You Need (2023). https://arxiv.org/abs/1706.03762

[2] Vig, J., Belinkov, Y.: Analyzing the Structure of Attention in a Transformer Language Model (2019). https://arxiv.org/abs/1906.04284

[3] Schuurmans, D., Dai, H., Zanini, F.: Autoregressive Large Language Models are Computationally Universal (2024). https://arxiv.org/abs/2410.03170

[4] Berto, F., Hua, C., Park, J., Kim, M., Kim, H., Son, J., Kim, H., Kim, J., Park, J.: RL4CO: a unified reinforcement learning for combinatorial optimization library. In: NeurIPS 2023 Workshop: New Frontiers in Graph Learning (2023). https://openreview.net/forum?id=YXSJxi8dOV

[5] Smit, I.G., Zhou, J., Reijnen, R., Wu, Y., Chen, J., Zhang, C., Bukhsh, Z., Zhang, Y., Nuijten, W.: Graph neural networks for job shop scheduling problems: A survey. Computers and Operations Research **176**, 106914 (2025) https://doi.org/10.1016/j.cor.2024.106914

[6] Xiao, Y., Wang, D., Li, B., Chen, H., Pang, W., Wu, X., Li, H., Xu, D., Liang, Y., Zhou, Y.: Reinforcement Learning-based Non-Autoregressive Solver for Traveling Salesman Problems. https://arxiv.org/abs/2308.00560 (2023)

[7] Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study. Princeton University Press, Princeton, NJ, USA (2006)

[8] Taillard, É.D.: A Linearithmic Heuristic for the Travelling Salesman Problem. EURO Journal of Operational Research **297**(2), 442–450 (2022) https://doi.org/10.1016/j.ejor.2021.05.034 . Available online June 2021

[9] Taillard, É.D., Heslgaun, K.: POPMUSIC for the travelling salesman problem. EURO Journal of Operational Research **272**(2), 420–429 (2019) https://doi.org/10.1016/j.ejor.2018.06.039

[10] Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural Combinatorial Optimization with Reinforcement Learning (2017). https://arxiv.org/abs/1611.09940

[11] Kool, W., Hoof, H., Welling, M.: Attention, Learn to Solve Routing Problems! (2019). https://arxiv.org/abs/1803.08475

[12] Bresson, X., Laurent, T.: The Transformer Network for the Traveling Salesman Problem. https://arxiv.org/abs/2103.03012 (2021)

[13] Goh, Y.L., Lee, W.S., Bresson, X., Laurent, T., Lim, N.: Combining Reinforcement Learning and Optimal Transport for the Traveling Salesman Problem. https://arxiv.org/abs/2203.00903 (2022)

[14] Wu, Y., Song, W., Cao, Z., Zhang, J., Lim, A.: Learning improvement heuristics for solving routing problems. IEEE Transactions on Neural Networks and Learning Systems **33**(9), 5057–5069 (2022) https://doi.org/10.1109/TNNLS.2021.3068828

[15] Zhu, T., Shi, X., Xu, X., Cao, J.: An accelerated end-to-end method for solving routing problems. Neural Networks **164**, 535–545 (2023) https://doi.org/10.1016/j.neunet.2023.05.003

[16] Taillard, É.D., Voss, S.: POPMUSIC: Partial optimization metaheuristic under special intensification conditions. In: Ribeiro, C., Hansen, P. (eds.) Essays and Surveys in Metaheuristics, pp. 613–629. Kluwer Academic Publishers, New York (2001). https://mistic.iict-heig-vd.ch/taillard/articles.dir/TaillardVoss2001.pdf

[17] Taillard, É.D., Voß, S.: In: Martí, R., Pardalos, P.M., Resende, M.G.C. (eds.) POPMUSIC, pp. 903–919. Springer, Cham (2025). https://doi.org/10.1007/978-3-032-00385-0_31

[18] Helsgaun, K.: Helsgaun's implementation of Lin-Kernighan. Version LKH-2.0.11 (2025). http://webhotel4.ruc.dk/\texttildelowkeld/research/LKH/

[19] Blum, C.: Beam-aco—hybridizing ant colony optimization with beam search: an application to open shop scheduling. Computers & Operations Research **32**(6), 1565–1591 (2005) https://doi.org/10.1016/j.cor.2003.11.018

[20] Resende, M.G.C., Ribeiro, C.C.: Optimization by GRASP. Springer, New York (2016). https://doi.org/10.1007/978-1-4939-6530-4

[21] Dorigo, M., Stützle, T.: Ant Colony Optimization. The MIT Press, Cambridge, US-MA (2004). https://doi.org/10.7551/mitpress/1290.001.0001

[22] Vijayakumar, A., Cogswell, M., Selvaraju, R., Sun, Q., Lee, S., Crandall, D., Batra, D.: Diverse beam search for improved description of complex scenes. Proceedings of the AAAI Conference on Artificial Intelligence **32**(1) (2018) https://doi.org/10.1609/aaai.v32i1.12340

[23] Meister, C., Vieira, T., Cotterell, R.: Best-first beam search. Transactions of the Association for Computational Linguistics **8**, 795–809 (2020) https://doi.org/10.1162/tacl_a_00346

[24] Li, J., Jurafsky, D.: Mutual Information and Diverse Decoding Improve Neural Machine Translation (2016). https://arxiv.org/abs/1601.00372

[25] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge, MA (2016). http://www.deeplearningbook.org