Demonstrating Program Evolution on the Traveling Salesman Problem

Demonstration Track

Solomon Andryushenko
Yandex School of Data Analysis
Moscow, Russia
solomon.andryushenko@proton.me

Artem Dzhalilov
Yandex School of Data Analysis
Moscow, Russia
artem.dzhalilov@gmail.com

Yaroslav Pelekhov
Yandex School of Data Analysis,
Moscow State University
Moscow, Russia
yaroslavpelekhov@gmail.com

Pavel Agafonov
Yandex School of Data Analysis,
HSE University
Moscow, Russia
pavelagafonov80@gmail.com

Maria Ivanova
Yandex School of Data Analysis,
Applied AI Institute
Moscow, Russia
ivanova.m.pe@gmail.com

ABSTRACT

We present a demonstration of the outcomes of large-scale program evolution applied to the Traveling Salesman Problem (TSP). Using the OpenEvolve evolutionary programming framework [6], inspired by AlphaEvolve [5], we evolve complete, executable TSP solvers and evaluate them empirically over thousands of evolutionary iterations. The demonstration focuses on the concrete results produced by evolution, including improvements in solution quality metrics and computational efficiency relative to strong existing baselines. Through quantitative performance trajectories, population-level evolutionary dynamics, and real-time visualization of evolved solvers, we show that program evolution can reliably discover competitive, non-trivial TSP algorithms and systematically improve them through iterative selection, highlighting its potential as a general framework for automated algorithm discovery.

KEYWORDS

Large Language Model (LLM); Traveling Salesman Problem (TSP)

ACM Reference Format:
Solomon Andryushenko, Artem Dzhalilov, Yaroslav Pelekhov, Pavel Agafonov, and Maria Ivanova. 2026. Demonstrating Program Evolution on the Traveling Salesman Problem: Demonstration Track. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 3 pages. https://doi.org/10.65109/NWXS1507

1 INTRODUCTION

The Traveling Salesman Problem (TSP) is a classical NP-hard combinatorial optimization problem with applications in routing, logistics, and scheduling. Due to its practical importance and computational difficulty, TSP has motivated extensive research on algorithmic techniques that aim to improve solution quality while controlling runtime, particularly for large-scale instances.

Recent approaches span multiple paradigms, including exact solvers such as Concorde [1], high-performance heuristics such as Lin-Kernighan-Helsgaun (LKH) [3], as well as supervised learning (Att-GCRN) [2] and unsupervised learning (UTSP) [4] methods that combine neural models with search procedures such as Monte Carlo Tree Search (MCTS). In particular, some learning-based approaches rely on neural networks to produce heatmaps that estimate the probability distribution of each edge being part of the optimal solution, which are used to guide MCTS in solution finding. However, Xia et al. [7] demonstrate that simple non-neural baselines, such as nearest-neighbor based methods, can outperform more complex neural heatmap guided approaches. Despite substantial prior work, achieving strong performance on large TSP instances continues to require carefully engineered solver logic that balances exploration, refinement, and computational cost.

This demonstration provides evidence that program evolution guided by large language models (LLMs) can autonomously improve such solver logic when applied directly to full algorithm implementations. Rather than introducing a new handcrafted heuristic or learning architecture, we focus on demonstrating the empirical outcomes of an automated evolutionary process that iteratively modifies and evaluates complete TSP solvers under a fixed evaluation protocol. To the best of our knowledge, this is the first work demonstrating LLM-driven program evolution for large-scale Euclidean TSP. The demonstration is conducted on 1,000-node Euclidean TSP instances, a regime in which even modest algorithmic improvements result in measurable differences in solution quality and runtime behavior. The source code and the demo video are available on GitHub[1] and YouTube[2] respectively.

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/NWXS1507

[1] github.com/strangecreator/openevolve-project/tree/tsp-example/examples/tsp_tour_minimization
[2] https://youtu.be/XAZW770jQZ8

2 EVOLUTIONARY FRAMEWORK

The system demonstrated in this work is built on OpenEvolve [6], an open-source framework inspired by the AlphaEvolve [5] paradigm for evolutionary program search. OpenEvolve provides the infrastructure required to maintain an evolving archive of candidate programs and to coordinate program generation, empirical evaluation and selection. Within each evolutionary iteration, OpenEvolve invokes an LLM to propose concrete code-level modifications to an existing program. These modifications are applied directly, producing a new candidate program that can be executed by OpenEvolve without manual intervention. Each candidate program is evaluated using a task-specific evaluation procedure that measures performance under fixed resource constraints. Based on the observed evaluation results, OpenEvolve updates its archive by retaining program variants that improve upon previously observed performance while discarding weaker candidates.

In this demonstration, OpenEvolve is configured to operate directly over a complete, executable TSP solver by defining a common interface through which candidate programs receive TSP instances and return complete tours. Evolution is initialized using a solver derived from the nearest-neighbor guided MCTS procedure [4, 7], which is inserted into the OpenEvolve archive as the initial individual program. Starting from this baseline, OpenEvolve employs an LLM to generate incremental code-level modifications, which are evaluated under combined score (average tour length and total time). In our experiments, we used the DeepSeek-Reasoner model accessed via API.

TSP solver implementations typically span multiple source files. To support long runs, we avoid including full program history in the LLM context. After each modification, we store a concise natural-language summary of the change and reuse it as context in subsequent steps together with the current solver implementation.

Each solver is evaluated on 128 independently sampled Euclidean TSP instances with 1,000 nodes, where node coordinates are drawn uniformly from the unit square. Solver performance is assessed primarily based on tour length, together with runtime-related statistics collected during execution. These measurements are used by OpenEvolve to determine whether a candidate solver improves upon existing solutions stored in the archive. By employing a consistent instance distribution and evaluation protocol, observed performance differences can be attributed to changes in solver behavior rather than variations in the problem setup.

3 DEMONSTRATION AND RESULTS

The results shown in the demonstration are obtained from a completed evolutionary run comprising 8,165 iterations. Each iteration requires approximately 2 mins for candidate generation, followed by about 10 mins for parallel evaluation of the resulting solver on 48 TSP instances randomly sampled from a 128-instance TSP-1000 test set [2]. All evaluations are performed on a 12-core CPU. The demonstration is organized into two stages. In the first stage, replayed evolutionary runs are presented, allowing users to observe how solver performance evolves over time under the OpenEvolve framework. The interface visualizes the progression of average tour length across evolutionary iterations, making it possible to track how successive solver variants improve as a result of evolutionary updates. In the second stage, the demonstration focuses on single-instance tour search using the best evolved solver obtained from the completed evolutionary run. During the evolutionary stage, solver performance improves steadily starting from the nearest-neighbor guided MCTS initialization. As evolution proceeds, successive code-level modifications generated by the LLM lead to measurable reductions in the average tour length under the fixed evaluation protocol. The visualization of performance trajectories enables direct inspection of how incremental code modifications influence solver behavior and solution quality over evolutionary time. In the single-instance tour search stage, the interface visualizes the current candidate tour produced by the active solver, the best-so-far tour discovered during the search process, and the trajectory of tour length as the search progresses. This view highlights the dynamic behavior of the evolved solver when applied to an individual 1,000-node Euclidean TSP instance and provides an intuitive illustration of its refinement process.

Table 1: Evolved solver results w.r.t. existing baselines, tested on 128 instances with 1000 nodes.

Method	Type	TSP-1000	
		Avg. tour length	Time
Concorde	Exact solver	23.12	6.65h
LKH3	Heuristic	23.12	38.09m
Att-GCRN [2]	SL + MCTS	23.52	43.94s
Rethink MCTS [7]	KNN + MCTS	23.63	3.34m
UTSP [4]	UL + MCTS	23.39	2.67m
Evolved TSP	Program evolution	**23.30** ±0.019	28.27m

To place the demonstrated results in context, Table 1 summarizes representative performance figures reported for the TSP-1000 test dataset [2]. The table includes exact, heuristic, and learning-based methods commonly used as reference points, together with the best average tour length achieved by the evolved solver shown in the demonstration. On 128 independently sampled 1,000-node Euclidean TSP instances, the evolved solver achieves a best average tour length of 23.30. While not a definitive state-of-the-art result given the specific LLM evolution budget, this performance is competitive with strong existing methods and demonstrates that large-scale program evolution can discover high-quality TSP solvers under comparable settings.

Beyond aggregate performance metrics, the system enables inspection of intermediate solver variants stored in the OpenEvolve archive. Examination of these intermediate solvers reveals systematic changes in search behavior across evolutionary stages, including differences in convergence speed, stability, and refinement dynamics. This qualitative inspection complements the quantitative results and provides insight into how solver behavior evolves as code modifications accumulate over the course of evolution.

Conclusion. This demonstration presents empirical results of applying program evolution to the Traveling Salesman Problem using the OpenEvolve framework. By evolving full solver implementations and evaluating them at scale, the system produces competitive, fully executable solvers and illustrates how large-scale program evolution can systematically improve solution quality on large Euclidean TSP instances under the evaluated protocol.

ACKNOWLEDGMENTS

The authors thank Vasily Volovich for editing the demonstration video using Adobe Premiere Pro.

REFERENCES

[1] David Applegate, Robert Bixby, Vasek Chvatal, and William Cook. 2006. Concorde TSP Solver.

[2] Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. 2021. Generalize a small pre-trained model to arbitrarily large tsp instances. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 7474–7482.

[3] Keld Helsgaun. 2000. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European journal of operational research* 126, 1 (2000), 106–130.

[4] Yimeng Min, Yiwei Bai, and Carla P Gomes. 2023. Unsupervised learning for solving the travelling salesman problem. *Advances in neural information processing systems* 36 (2023), 47264–47278.

[5] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. 2025. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131* (2025).

[6] Asankhaya Sharma. 2024. OpenEvolve. https://huggingface.co/blog/codelion/openevolve. Hugging Face Blog, accessed 2026-01-04.

[7] Yifan Xia, Xianliang Yang, Zichuan Liu, Zhihao Liu, Lei Song, and Jiang Bian. 2024. Position: Rethinking post-hoc search-based neural approaches for solving large-scale traveling salesman problems. *arXiv preprint arXiv:2406.03503* (2024).