GEPA-TSP: Specializing Lin–Kernighan Heuristics to Target Instance Distributions

Reuben Narad [1]

Abstract

Large language models (LLMs) with reflective evolution have recently been used as search procedures over programs and prompts, offering a new way to adapt existing code to specific workloads. We study this idea in an operations research setting, tuning Concorde (a state-of-the-art exact TSP solver) by evolving the details of its pre-packaged Lin-Kernighan (LK) heuristic. Using GEPA, a reflective prompt evolution algorithm with separate actor and reflector LLMs, we build a sandbox that regenerates and benchmarks candidate heuristics. We target three 400-node TSP distributions of increasing structural complexity: uniform Euclidean, clustered Euclidean with in-cluster discounts, and a road-network derived from Seattle's map. Across many runs, we observe a single robust win on Seattle instances, where GEPA found a change that improves solver time by 5%. On other distributions, the same modification consistently regresses performance. Our results frame GEPA-style LLM search as a distribution-specific tuning method for existing heuristics, and highlight both the possibility and difficulty of beating carefully engineered baselines.

1. Introduction

Automated scientific discovery is an emerging area of interest, aiming to accelerate scientific research by automating parts of the work of an investigator. Instead of testing single experiments or code variations at a time, LLMs can run large batches of these tasks: testing many hypotheses, trying many variants of a codebase, or exploring many proof attempts in a loop. Many of these problems can be viewed as search over a large space of hypotheses, programs, or algorithmic components.

In such settings, the searching agent must keep past rollouts in context. If the agent does not know which experiments or variants worked previously, it will search in an inefficient, unguided way that ignores available feedback, making progress in a huge space effectively intractable. At the same time, fully remembering every experiment is impossible: the search space is too large to keep all history in context at once. This tension naturally motivates mechanisms that maintain a compact but useful long-term memory of past attempts.

Prompt learning and evolution provide one such mechanism. In approaches such as the DSPy framework (**?**), the context of an agent as a major component of its performance, treating it as a learned, mutable object. Using a second "reflector" LLM, that context can be optimized to a given reward by being evolved over time based on the results of its earlier attempts. The Genetic Pareto (GEPA) (**?**) algorithm decides which examples to keep in the reflector's context by maintaining a Pareto frontier of past candidates. Similar prompt-evolution ideas have begun to appear in automated discovery systems more broadly, including work like AlphaEvolve, where long-lived records of past trials guide future exploration.

Heuristics in operations research (OR) are a natural testbed for this paradigm. We study the traveling salesman problem (TSP) by taking Concorde and focusing on its Lin–Kernighan (LK) heuristic, which we allow an LLM to modify. We view the search process as testing different configurations or versions of this LK component. Concretely, we frame this as a task within the GEPA framework: a student model proposes rewrites of the LK heuristic, we recompile Concorde, run it on a benchmark of TSP instances solving to optimality, and use its runtime (relative to vanilla Concorde) and number of branch-and-bound search nodes as a reward signal to decide which examples to keep in the reflector model's context. The reflector model then uses its in-context examples—actor prompts, the resulting heuristic code, and their measured performance—to propose changes to the actor's prompt.

Our contributions are threefold. First, we present a reproducible sandbox for LK block injection in Concorde

[1]University of Washington. Correspondence to: Reuben Narad <rnarad@uw.edu>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

with full artifact logging, designed to provide the reflector with rich feedback for rewriting the actor's prompt. Second, we curate a benchmark of non-Euclidean and Euclidean TSP instance distributions that expose differences between highly tuned baselines and under-optimized regimes. Third, we provide empirical evidence that a reflective LLM loop can discover a small improvement to Concorde on a structured travel-time distribution, while failing to improve and often degrading performance on classical Euclidean benchmarks. We release code, data, and all candidate LK variants to support further work on OR-driven automated science.

2. Related Work

- **Classical LK and Concorde.** Foundational heuristics date to Lin–Kernighan's effective local search for TSP (Lin & Kernighan, 1973) and Helsgaun's LKH implementation (Helsgaun, 2000); Concorde's implementation and engineering remain the reference standard (Applegate et al., 2006).
- **Learning to optimize solvers.** A growing line of work learns heuristics or policies for combinatorial optimization (Bello et al., 2017; Kool et al., 2019; Deudon et al., 2018; d. O. da Costa et al., 2020; Kerschke et al., 2018); our setting follows the same spirit but targets distribution-specific LK tweaks.
- **LLM-guided code evolution.** ReEvo frames LLMs as reflective hyper-heuristics that iteratively refine algorithms (Ye et al., 2024); related work on reflective prompt evolution and LM pipeline optimization includes GEPA and DSPy (Agrawal et al., 2025; Khattab et al., 2023), as well as automated algorithm configuration and selection (Hutter et al., 2009; **?**; Kerschke et al., 2019); our GEPA loop similarly mutates and tests LK code but with a sandboxed, deterministic TSP pipeline.

3. Method: GEPA for Lin–Kernighan

3.1. Background: How Concorde Works

Concorde is an exact branch-and-bound solver that is heavily optimized for the TSP. It starts from the standard integer programming formulation of the TSP and relaxes it to a linear program. Because the LP is convex, it can be solved quickly. If the optimal LP solution happens to be integral, it is also an optimal solution to the original integer program.

When the LP solution is fractional, Concorde selects an edge whose LP value lies strictly between 0 and 1 and branches on it. One child node adds a constraint forcing this edge to be in the tour, and the other child node adds a constraint forcing it to be excluded. Recursively solving these subproblems yields a branch-and-bound tree. Alongside this tree search, Concorde runs a heuristic to maintain a best-known tour. Any branch whose LP relaxation cannot beat the heuristic tour length is pruned. The stronger this heuristic baseline, the more aggressively Concorde can prune, and the faster it converges to the optimal tour. Concorde ships with a carefully engineered Lin–Kernighan (LK) heuristic to provide this baseline.

3.2. Turning the Heuristic into a GEPA Task

In Concorde's codebase, the LK heuristic is implemented as a well-defined C file, which we treat as a modular component. We build a workflow that replaces this LK code with a new heuristic block, recompiles Concorde, and then solves a fixed validation set of TSP instances. For each run, we log Concorde's textual output, the total wall-clock time to solve the validation set, and the number of branch-and-bound nodes explored during the search. In GEPA, we use the total time and the node count as rewards.

For the purposes of GEPA, we view this reward as a function of the actor's prompt. Let $\mathcal{P}$ denote the space of actor prompts and $\mathcal{H}$ the space of heuristic code blocks. The actor LLM is a (stochastic) mapping

$$A : \mathcal{P} \to \mathcal{H}, \qquad h = A(p),$$

which takes a prompt $p \in \mathcal{P}$ and produces a candidate heuristic $h \in \mathcal{H}$ (the LK replacement). Given a fixed validation set D_{val} of TSP instances, the Concorde pipeline defines an evaluation map

$$C : \mathcal{H} \times \mathcal{D} \to \mathbb{R}^2 \times \mathcal{T}, \qquad C(h, D_{\text{val}}) = (t, n, \tau),$$

where t is the total runtime on D_{val}, n is the total number of branch-and-bound nodes, and $\tau \in \mathcal{T}$ denotes the emitted trace (logs and solver output). The overall black-box objective that GEPA interacts with is the composition

$$F : \mathcal{P} \to \mathbb{R}^2 \times \mathcal{T}, \qquad F(p) = C(A(p), D_{\text{val}}) = (t, n, \tau).$$

We are interested in minimizing both t and n jointly: a candidate (t', n') is preferred to (t, n) when it Pareto-dominates it, i.e., $t' \leq t$ and $n' \leq n$ with at least one strict inequality. In practice, GEPA maintains a set of non-dominated examples in this two-dimensional objective space and uses them as in-context training data for the reflector. The actor LLM, code generation, compilation, and Concorde evaluation are internal details of this composite map; GEPA only sees prompts, their resulting (t, n) pairs, and the logged artifacts used for reflection.

- Non-Euclidean: structured_seattle_time (400 nodes) from OSM travel-time shortest paths (val/test splits of 20/50 instances).

- Euclidean: uniform_val/test (400 nodes) and clustered_val/test (400 nodes); metadata and seeds released.
- Other splits (toy20/200, tsplib_random) maintained for smoke/regression; not central to main findings.

4. Experimental Setup

4.1. Benchmarks and Data

- Concorde is fast! For instances up to 200 nodes, solution timne was so fast (¡1s, 1 bbnode) that noise was the only differentiating factor. - Thus, we focused on 400 node instances, checking that indeed Concorde typically used 4-6 BB nodes in the solving process - For our experiments, we defined 3 types of TSP instance of increasing structure, with generators to sample new instances:

- 2d Euclidian, Uniform sampling (basic TSP) - Clusters: For a given instance, we define the clusters (2-4, with centers and covariances drawn uniformly). Mostly Euclidian, but with a 50% in-cluster travelling discount (to add structure) - - For a given GEPA candidate, the reward was

4.2. Gepa Details

- Models: student gpt-5-nano, reflector gpt-5-mini; reflection batch 2–3; 20 metric calls.
- Evaluation: per-instance repeats (3–5 on val), CPU affinity when available, timeouts off for reported runs; artifacts under `runs/`.
- Baseline: Concorde default LK rebuilt in the same sandbox; baseline repeats higher (5) for a stable reference.
- Variance: report per-instance averages; note that nontrivial gains require reproducible settings (affinity, binary hash).

5. Results: TSP Adaptation

- **Uniform (Euclidean, test 50 inst.):** baseline runtime $4.223\,\mathrm{s}$ / BB 3.84 vs GEPA best $4.521\,\mathrm{s}$ / BB 3.96 ($+7.1\%$ runtime, $+3.1\%$ BB). Plots: `out/gepa_uniform_n400_mean_std.png`; summaries: `runs/eval/eval/20251202T224329Z_uniform_test_baseline`, `runs/eval/eval/20251202T225646Z_uniform_test_gepa_iter40`.
- **Clustered (Euclidean, test 50 inst.):** baseline $4.405\,\mathrm{s}$ / BB 4.6 vs GEPA $4.544\,\mathrm{s}$ / BB 5.2 ($+3.1\%$ runtime, $+13.0\%$ BB). Plots: `out/gepa_clustered_20251129T211605Z.png`; summaries: `runs/eval/eval/20251202T230700Z_clustered_test_baseline`, `runs/eval/eval/20251202T231054Z_clustered_test_gepa_iter30`.
- **Seattle (travel-time, test 50 inst.):** baseline $3.958\,\mathrm{s}$ / BB 3.68 vs GEPA $3.768\,\mathrm{s}$ / BB 3.55 (-4.8% runtime, -3.5% BB); steady improvement in the rollout (see `out/gepa_structured_seattle_time_n400_time_smoothed_final.png`). Summaries: `runs/eval/eval/20251202T231521Z_seattle_time_test_baseline` (latest `runs/eval/eval/20251202T233939Z_seattle_time_test_baseline_latest`) and `runs/eval/eval/20251202T231850Z_seattle_time_test_gepa_iter31`.
- Overall: GEPA slows Euclidean cases relative to the tuned baseline, but yields a modest Seattle speedup while slightly reducing BB nodes, underscoring distribution-specific effects.

6. Discussion

- GEPA excels when the baseline is not already tuned: non-Euclidean Seattle benefits; tuned Euclidean baselines do not.
- The learned tweak targets buffer/flush overhead; it may hurt when coherent batches are valuable (Euclidean).
- Reproducibility is essential: affinity, binary hashes, and artifact logging prevent confounding from build drift.
- Deployment: treat GEPA as per-distribution autotuning—run briefly on your workload, adopt the candidate if it beats your baseline.

7. Conclusion

- GEPA can specialize LK for specific TSP distributions, yielding modest gains on non-Euclidean travel-time data while leaving tuned Euclidean baselines unchanged or slightly worse.
- Future work: multi-objective rewards, better diversity in proposals, and extensions beyond TSP.
- Release: code, data, and all candidate artifacts for reproducibility.

References

Agrawal, L. A., Tan, S., Soylu, D., Ziems, N., Khare, R., Opsahl-Ong, K., Singhvi, A., Shandilya, H., Ryan, M. J., Jiang, M., Potts, C., Sen, K., Dimakis, A. G., Stoica,

I., Klein, D., Zaharia, M., and Khattab, O. GEPA: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025. URL https://arxiv.org/abs/2507.19457.

Applegate, D. L., Bixby, R. E., Chvátal, V., and Cook, W. J. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. Neural combinatorial optimization with reinforcement learning. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1611.09940.

d. O. da Costa, P. R., Rhuggenaath, J., Zhang, Y., and Akcay, A. Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In *Proceedings of The 12th Asian Conference on Machine Learning*, volume 129 of *Proceedings of Machine Learning Research*, pp. 465–480. PMLR, 2020. URL https://proceedings.mlr.press/v129/costa20a.html.

Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., and Rousseau, L. Learning heuristics for the TSP by policy gradient. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2018)*, volume 10848 of *Lecture Notes in Computer Science*, pp. 170–181. Springer, 2018. doi: 10.1007/978-3-319-93031-2_12.

Helsgaun, K. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000. doi: 10.1016/S0377-2217(99)00284-2.

Hutter, F., Hoos, H. H., Leyton-Brown, K., and Stützle, T. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36: 267–306, 2009. doi: 10.1613/jair.2861.

Kerschke, P., Kotthoff, L., Bossek, J., Hoos, H. H., and Trautmann, H. Leveraging TSP solver complementarity through machine learning. *Evolutionary Computation*, 26(4):597–620, 2018. doi: 10.1162/evco_a_00215.

Kerschke, P., Hoos, H. H., Neumann, F., and Trautmann, H. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45, 2019. doi: 10.1162/evco_a_00242.

Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., Sharma, A., Joshi, T. T., Moazam, H., Miller, H., Zaharia, M., and Potts, C. DSPy: Compiling declarative language model calls into self-improving pipelines. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2310.03714.

Kool, W., van Hoof, H., and Welling, M. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1803.08475.

Lin, S. and Kernighan, B. W. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973. doi: 10.1287/opre.21.2.498.

Ye, H., Wang, J., Cao, Z., Berto, F., Hua, C., Kim, H., Park, J., and Song, G. Reevo: Large language models as hyper-heuristics with reflective evolution. *arXiv preprint arXiv:2402.01145*, feb 2024. URL https://arxiv.org/abs/2402.01145. NeurIPS 2024.