FUNSEARCH ON TRAVELING SALESMAN PROBLEM CS5491 AI PROJECT

MACheng
59046021

SuZehao
58818678

ZhouJiayu
58909853

ABSTRACT

The Traveling Salesman Problem (TSP) is a typical NP-hard problem in combinatorial optimization and has very important applications in many aspects. Although many heuristics exist, it is still challenging to efficiently find optimal or near-optimal solutions for large instances. This project explores the application of FunSearch, an LLM-driven evolutionary framework, in automatically discovering novel and effective heuristic functions for solving TSP instances. We define a specification for a heuristic TSP solver, where the core prioritization function evolved by FunSearch is used to provide guidance for the selection of the next city. `bayg29.tsp` dataset serves as a benchmark case, where FunSearch successfully identifies a heuristic function that significantly outperforms the initial baseline heuristic function provided in the framework specification, thus demonstrating the potential of LLM-based evolutionary search for discovering high-performance algorithms for complex optimization problems.

1 INTRODUCTION

The Traveling Salesman Problem (TSP) requires finding a shortest route that allows it to visit each city in a given table once and finally return to the origin city. As a typical NP hard combinatorial optimization problem, its complexity grows exponentially with the number of cities. The exact solution is difficult to compute except for a small number of small-scale instances.The TSP serves as a benchmark for optimization techniques, which are often used in life in various fields such as logistics and planning.

Traditional methods include more refined branch and bound methods that are only feasible for smaller problems, approximation algorithms that guarantee performance, and various heuristics and metaheuristics such as the nearest neighbor method. These heuristics, while effective to varying degrees for problem solving, may ignore unconventional algorithmic strategies.

FunSearch proposes a new paradigm by utilizing the code generation capabilities of Large Language Models (LLMs) within an evolutionary framework. It generates, evaluates and refines program code in an iterative manner, discovering new solutions in a vast program space. The core idea of the method is to maintain a population of programs, evaluate their performance on relevant inputs, and use the best performing programs to motivate the LLM to generate new and improved versions.

This project applies the FunSearch methodology to TSP. our goal is to automatically discover high-performance heuristic functions (prioritized) to guide constructive TSP solvers. We utilize the `bayg29.tsp` instance from the TSPLIB library as our primary test case and demonstrate FunSearch's ability to evolve simple heuristics into more efficient heuristics.

2 METHOD

Our approach integrates the FunSearch framework with specific components tailored for solving the TSP.

2.1 FUNSEARCH FRAMEWORK

The framework runs on two core components:

- **LLM Sampler (Code Generator):** Generates candidate Python functions (in our case prioritized functions) based on existing high-performance functions and hints built into the problem specification.
- **Evaluator and Sandbox:** compiles and executes the generated functions in a secure sandbox environment, evaluates their performance on problems entered into the TSP instance, and filters out poorly performing or incorrect solutions.

2.2 PROBLEM SPECIFICATION FOR TSP

We define a Python-based specification for FunSearch and build the TSP solution process around an evolvable heuristic function. The main components are as follows:

`heuristic_tsp_solver`: acts as the main solver function. Iteratively constructs the tour from multiple initial cities to ensure robustness. The function calls `priority_func` at each step and determines the next city to visit based on the current city, the distance matrix, and the state of the cities visited.

`two_opt_improvement`: optional local search function for improvement steps, the main role of which is to apply the tour generated by the heuristic solver to further improve the solution.

`calculating_tour_distance`: utility function for calculating the total length of a given tour.

`priority_func`priority: core function decorated with `@@funsearch.evolve`. FunSearch focuses its evolutionary search on improving this function. It takes as input the current city, the complete distance matrix, and the set of visited cities, and returns an array containing the priority scores for the next visit to each unvisited city, with lower scores giving higher priority. We provide a simple initial version based on distance, connectivity and completion factor as a starting point.

`evaluate`: The main evaluation function qualified with `@funsearch.run`. It calls `heuristic_tsp_solver` with the current priority function generated by LLM and returns the negative of the total distance of the generated tour.The goal of FunSearch is to maximize this score, thus effectively minimizing the tour distance.

2.3 LLM INTEGRATION

We implemented an LLMAPI class inheriting from sampler.LLM.

- **Model:** it passes the `api.bltcy.ai`endpoint to access GPT-4o models.
- **Prompting:** Added a specific directive `_additional_prompt` to direct LLM to generate more complex and varied Python functions that focus on the code output without providing descriptive text.
- **Output parsing:** A helper function `_trim_preface_of_body` has been implemented to preprocess the LLM output, removing extraneous text and function signatures that are often generated by non-plain code after the model is completed, and isolating the function body for execution.

2.4 SANDBOX IMPLEMENTATION

To ensure safe execution of LLM-generated code, we implemented a custom sandbox class that inherits from `api.bltcy.ai`evaluator.Sandbox.

- **Timeout:** Use Python's threading module to run the evaluation function in a separate thread with a pre-set timeout of `evaluate_timeout_seconds` to prevent infinite loops or long computation times.
- **Output Limiting:** During execution, the standard output `sys.stdout` is redirected to the `StringIO` buffer. This prevents overprinting and allows capturing a limited amount of output for debugging purposes, while enforcing a maximum output size of `_max_output_size`.
- **Error Handling:** Captures exceptions that occur during in-thread code execution and reports errors without causing the FunSearch main process to crash.

2.5 Data Handling and Preprocessing

TSP instance files were processed as follows:

- **Parsing Coordinates:** The `parse_display_coords` function extracts city coordinates from the `DISPLAY_DATA_SECTION` section of the standard TSP file.
- **Distance Matrix Calculation:** The `coordinates_to_distance_matrix` function calculates the Euclidean distance between all city pairs and generates the distance matrix required by the solver.

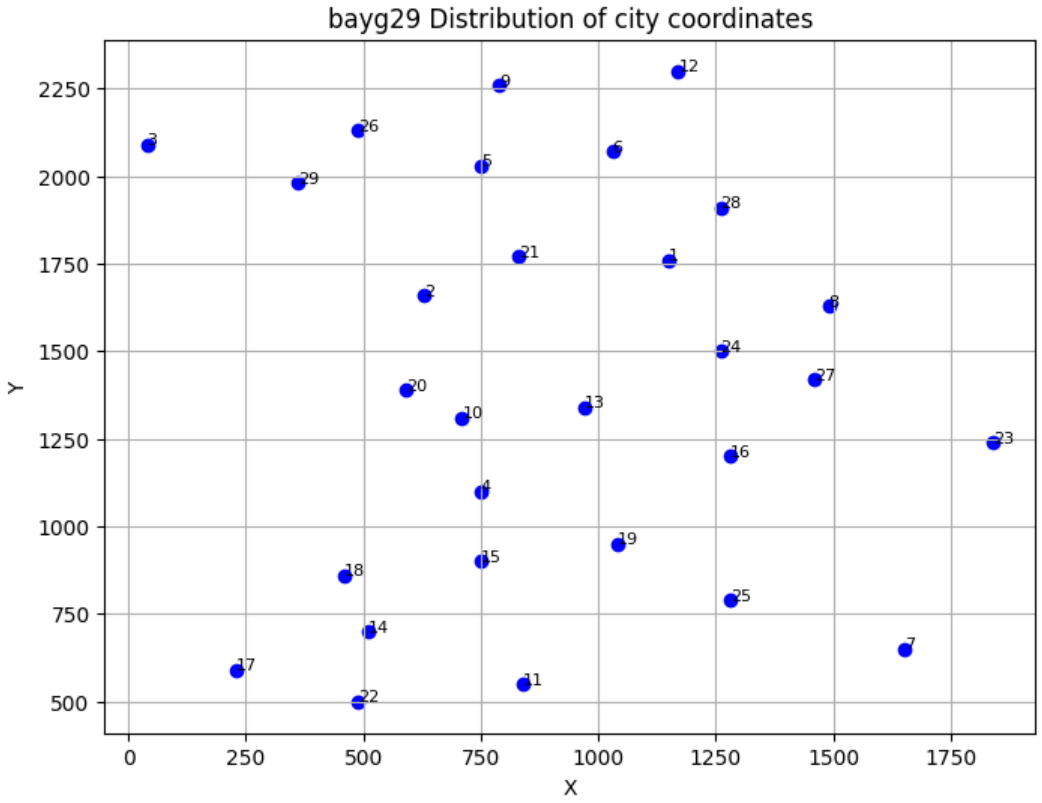

Figure 1: Distribution of city coordinates for bayg29 instance.

3 Results and Discussion

We applied the configured FunSearch framework to the `bayg29.tsp` instance.

3.1 Baseline Performance

The initial prioritization function provided in the specification was first evaluated based on a weighted sum of distances, average connectivity to unvisited cities, and a completion factor. Running the evaluation function on the `bayg29.tsp` distance matrix using this baseline heuristic yields a negative total distance score of -9837.41, which is the starting point for the evolutionary search. For reference, a simple nearest-neighbor heuristic on the same dataset yields a total distance of 10211.18. This suggests that the initial canonical heuristic has been somewhat effective, but is still far from optimal.

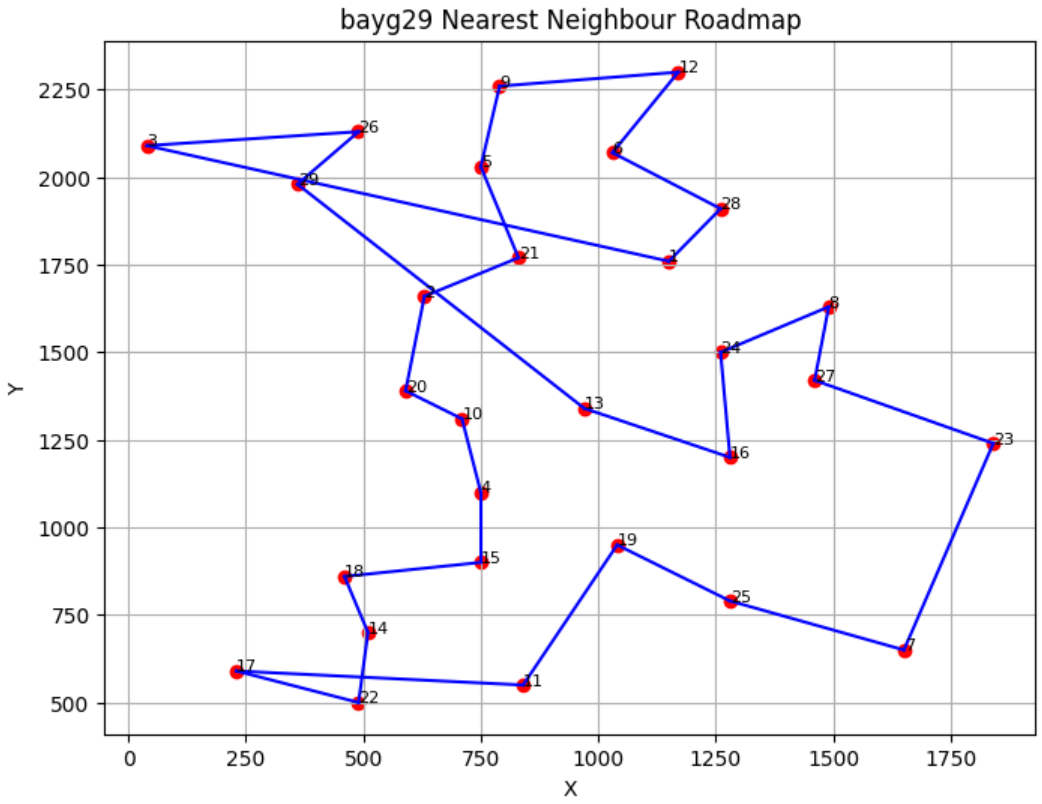

Figure 2: Nearest neighbour connectivity roadmap for bayg29 TSP instance.

3.2 FunSearch Discovery Process

The number of sampling steps for FunSearch was set `global_max_sample_num = 10`, and multiple independent parallel search processes drove the LLM to generate changes in the prioritization function. These changes were evaluated using a sandbox, and the execution logs showed iterative improvements in the best scores found at different search nodes.

3.3 Best Discovered Heuristic and Performance

During the FunSearch run on `bayg29.tsp`, the best score achieved was -372.02. This is a significant improvement over the initial baseline score of -9837.41. The corresponding priority functions contain more complex logic than the baseline function, including factors such as congestion estimation, weather condition penalties modeled through random factors, and dynamic weighting based on travel progress. While some of the generated functions lead to errors, the framework successfully filters out these errors and focuses on valid, high-performing candidates.

3.4 Insights

Results show that FunSearch, guided by a specified evaluation framework, can automatically explore the program space of TSP heuristic functions with significantly better results than non-initial heuristic solutions.LLM is able to generate functions with more complex logic, combining different factors including distance, connectivity, progress, randomness, and simulation of externalities in novel ways. The dramatic increase in scores highlights the potential for discovering effective, potentially non-intuitive algorithmic components through this LLM-driven evolutionary approach. The sandboxing mechanism proved to be critical in dealing with potential error codes generated by the LLM.

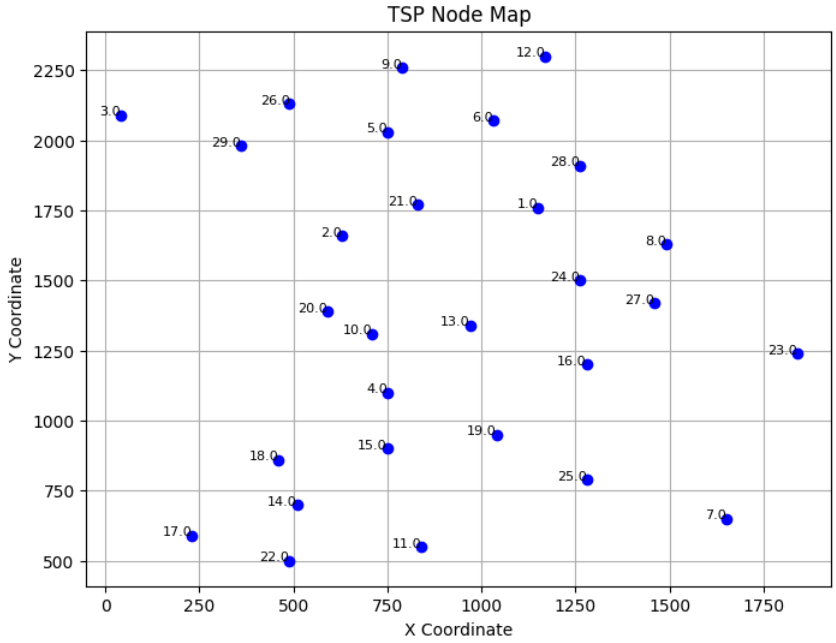

Figure 3: Spatial distribution of nodes in a TSP instance.

4 Conclusion

This project demonstrates the application of the FunSearch framework to the traveling salesman problem. By defining a specification centered on an evolvable heuristic prioritization function and implementing the necessary LLM sampling and sandbox evaluation components, we achieve automatic discovery of improved TSP heuristics. For specific instances, FunSearch evolves an initial heuristic function that significantly reduces the computed tour length.

These findings highlight the ability of LLM-driven evolutionary search to discover new algorithmic strategies in complex combinatorial optimization domains such as TSP, potentially surpassing human-designed heuristics. Future work includes extending the runtime of FunSearch, applying it to a wider range of TSP instances with different characteristics, exploring different solver skeletons in the specification, and comparing the discovered heuristics with existing state-of-the-art TSP algorithms.