Optimizing Travelling Salesman Problem Solutions Through FunSearch Technology

LIANG Tianyi **LIU Yungeng** **CHEN Meng**
Department of Computer Science
City University of Hong Kong

1 Problem Description

1.1 Background of the Problem

The Traveling Salesman Problem (TSP) is a classic combinatorial optimization challenge that aims to find the shortest possible route for a salesman to visit a set of cities and return to the starting city, visiting each city only once. Specifically, the TSP is described as follows: Given a set of cities (nodes) and the distances between each pair of cities (edge weights). The goal is to find a route that starts from the initial city, passes through each city exactly once, and returns to the initial city, all while minimizing the total length of the route.

1.2 Why Choose TSP Problem

Theoretical Significance and Practical Value:
The Traveling Salesman Problem (TSP) is not only a quintessential problem in theoretical computer science and operations research but also boasts extensive practical applications. The optimization concepts and methods involved in TSP are applicable to multiple fields such as logistics, transportation, and supply chain management. Studying this problem allows for a deeper understanding of core concepts and technologies in combinatorial optimization, graph theory, and algorithm design.
Challenge:
TSP is categorized as an NP-hard problem, lacking a known exact solution within polynomial time. This significant challenge not only fuels enthusiasm for problem-solving and creative thinking but also facilitates the learning and mastering of advanced algorithms and mathematical theories.
Career Development:
Solving complex algorithmic problems like the TSP is a crucial skill for many technical roles, including data scientists and algorithm engineers. Undertaking this project can significantly enhance personal competitiveness in the job market, making it easier to secure employment in the future.

2 Method Explainations

2.1 FunSearch Framework Overview

Funsearch primarily consists of five components: Specification, Pretrained LLM, Evaluation, Prompt, and Program Database(4):
Specification: It has been discovered in research that Funsearch's performance can significantly improve if an initial solution is provided at runtime.
Pretrained LLM: The LLM acts as the creative core of FunSearch, responsible for suggesting improvements to the functionality presented in the prompt and forwarding them for evaluation.
Evaluation: Programs generated by the LLM are evaluated and scored based on a set of inputs.
Program Database: The program database stores a set of correct programs. These programs are then sampled to create prompts. Preserving and encouraging diversity in the

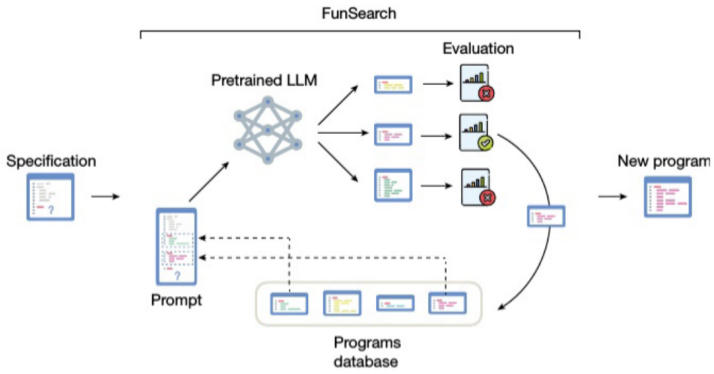

Figure 1: the process of Funsearch

database is crucial for exploration and avoiding local optima. Researchers in the paper use a genetic algorithm to ensure the flow of information.

2.2 Detailed Execution Process of FunSearch

2.2.1 Overview of the FunSearch Implementation for TSP

In tackling the Traveling Salesman Problem, the FunSearch method employs a cycle that iteratively refines a solution. It starts with a Specification that defines how to assess and evolve potential solutions through the evaluate and priority functions. Initial solutions populate a set of 'islands'—diverse starting points to explore the problem space. The Pre-trained Language Model (LLM) then generates new code iterations, particularly focusing on enhancing the priority function which guides the search for efficient routes.
These new iterations are executed and scored within a Sandbox environment, ensuring safety and efficiency. High-scoring solutions update the Program Database, which in turn informs the creation of new prompts. These prompts guide the LLM in its next round of code generation, effectively closing the loop. Selective integration across islands allows for the cross-pollination of strategies, gradually improving the priority function's performance with each iteration. This process iterates until the evolving solutions converge on an optimal or satisfactory resolution for the TSP.

2.2.2 Specific Implementation

The detailed execution of the FunSearch process for the TSP problem is unfolded across several critical components, each contributing to the iterative refinement and evolution of solutions. Below we detail the functionality and the role of each component within the FunSearch ecosystem.

2.2.3 Overview of the FunSearch Implementation for TSP

Specification:
The Specification is the foundational blueprint of the problem-solving process.The specific description of our specification algorithm is as follows:
—
Specification:
The Specification is the foundational blueprint of the problem-solving process. The specific description of our specification algorithm is as follows:

```
Table 3-2 Heuristic Algorithm for the Traveling Salesman Problem

Algorithm for TSP based on a complex priority function
Input: Distance matrix 'distances' between cities, priority function 'priority_fn'
```

```
Output: Sequence array of visited cities 'tour', total travel distance 'total_distance'
(1) Initialize the number of cities 'num_cities' to the length of the distance matrix
'distances'
(2) Create and initialize an array 'visited[num_cities]' with all entries set to 'False'
(3) Set the starting city 'current_city' to 0 and mark 'visited[current_city]' as 'True'
(4) Initialize the travel path 'tour' with 'current_city' included in the array
(5) Initialize the total distance 'total_distance' to 0
(6) While the length of 'tour' array is less than 'num_cities':
for 'i' = 1 to 'num_cities' - 1 do:
(a) Use the priority function 'priority_fn' to calculate the
priority 'priorities' for each unvisited city
(b) Set the 'priorities' for visited cities to infinity
(c) Select the city with the smallest value in 'priorities' as 'next_city'
(d) Mark 'visited[next_city]' as 'True'
(e) Append 'next_city' to the 'tour' array
(f) Add 'distances[current_city, next_city]' to 'total_distance'
(g) Update 'current_city' to 'next_city' end for
(7) Append the starting city to the end of the 'tour' array,
update 'total_distance' to include the distance back to the starting city
(8) Return 'tour' and 'total_distance'
```

In summary, our heuristic determines which cities to visit next in the traveling salesman problem by prioritizing each city based on its calculated distance from the current city and other factors. The algorithm initializes a city as the starting point, and then iteratively selects the unvisited city with the highest priority to construct a travel route. In this process, the algorithm continues to optimize the path until all cities are covered, and finally returns to the starting point, forming a closed loop.

2.2.4 Pretrained LLM

The Pretrained Language Model (LLM) is a key part of improving the way we solve the Traveling Salesman Problem (TSP) in the FunSearch system. It uses a specific setup to handle data and communicate over the internet.

Data Processing

The LLM has a special function that cleans up the text it generates. This function cuts out any parts that are not actual code, such as descriptions or unnecessary symbols. This ensures that the output is clean and ready to use.

Network Requests and Response Handling

The system talks to the LLM through HTTP requests to an external API, specifically "api.chatanywhere.com.cn." This setup allows the LLM to continue building on existing code. The system is also designed to handle errors well. It can retry sending a request if the first try fails due to issues like a bad internet connection. This makes sure that the system works smoothly and reliably, even when there are network problems.

These steps help the LLM provide useful code changes that directly help improve solutions for the TSP, ensuring that the FunSearch process is efficient and effective.

2.2.5 Evaluation(1)(2)

The evaluation component of the FunSearch project is pivotal for testing and verifying the effectiveness of code solutions generated for the Traveling Salesman Problem (TSP). It comprises two main elements: the Sandbox and the evaluator, which together ensure that the solutions are not only effective in solving the problem but also efficient and safe to run.

Sandbox: The Sandbox acts as a controlled environment where the generated code is executed. It is crafted to prevent any potentially harmful operations by the code, such as unauthorized internet access or excessive use of system resources. Moreover, it incorporates a timeout mechanism to halt any code that runs longer than the predefined limit, addressing the issue of infinite loops or excessively long computations.

The operational steps in the Sandbox are as follows:

- **Process Initialization:** For each piece of code, a separate process is initiated. This isolation helps in managing the execution safely and independently.
- **Execution Constraints:** Each process is monitored for its execution time. If the process exceeds the set time limit (specified in `timeout_seconds`), it is terminated to prevent resource drainage.
- **Result Collection:** Upon completion of the execution within the allowed time, the output is retrieved. If the process terminates or fails, it is recorded as unsuccessful.

Evaluator: The evaluator's role is to assess the outputs generated by the Sandbox. It evaluates the code based on specific metrics such as the efficiency of the solution and its compliance with the TSP requirements. This evaluation is crucial for determining which solutions are viable and which need further refinement or should be discarded.
The evaluation steps involve:

- **Output Assessment:** The evaluator reviews the results from the Sandbox, checking if the outputs meet the predefined criteria for success.
- **Metric Calculation:** It calculates various metrics from the execution, such as the computational efficiency and the quality of the TSP solution.
- **Feedback Integration:** Successful strategies are identified and used to inform further iterations of the solution generation process.

This evaluation mechanism ensures a robust feedback loop within the FunSearch framework, enabling continuous refinement of solutions based on empirical data and performance metrics. At the end of the evaluation phase, solutions that meet the set standards progress to the next cycle of refinement, gradually improving the effectiveness and efficiency of the TSP solutions.

2.2.6 Program Database and Prompt

The Program Database and the creation of prompts are key parts of how FunSearch works. They are closely linked in managing and improving the code solutions we develop.
Program Database
The Program Database keeps and updates a collection of different program versions, organized into groups called "islands." Each island has its own set of programs that get better over time through changes like updates and selections based on how well they perform.
In simpler terms, the database picks which programs might be good to keep based on their scores, using a special rule (the softmax function) that tends to favor better-performing programs. This helps ensure that over time, the programs we use and refine are those more likely to give us the best results.
Prompt Generation
Prompts are special instructions or starting points that we give to the language model (LLM). They are made using the best programs we have, selected from these islands. The idea is to take the strongest programs, put them into a prompt, and then ask the LLM to make them even better.
This step is crucial because it guides the LLM on what problems to solve and how to try solving them. It's like giving the LLM a direction or focus in its task of creating new and better code.
Integration and Continuous Evolution(3)
Every new piece of code the LLM creates from a prompt is tested. If it works well, it's added to our collection in the Program Database. This makes our code collection like a living thing that grows and improves over time, constantly learning and adapting.
We keep multiple islands to make sure we have lots of different types of code being

developed. This variety is important because it helps us avoid getting stuck with only one way of solving problems, keeping our solutions fresh and effective.
In summary, the Program Database and Prompt are at the heart of how FunSearch finds new and better ways to solve problems. They make sure we are always improving our solutions and exploring new possibilities. This setup is crucial for tackling tough challenges and getting better results with each attempt.

3 Results

Dataset visualization
With a cleansed and structured dataset in place, we proceeded to a preliminary visualization phase. Visualization serves a dual purpose in our study; it provides an intuitive understanding of the problem space and verifies the integrity of our data through graphical representations.
For the visualization, we plotted the locations of the nodes (cities) involved in the TSP instances. This depiction not only confirmed the accuracy of the dataset post-cleaning but also offered a visual confirmation of the complexity inherent in TSP. The plots illustrated the non-triviality of the task, with scattered nodes and the absence of apparent simplistic patterns, underscoring the necessity for sophisticated approaches to tackle the problem efficiently.
We performed simple visualizations of the “att48.tsp”, "a280.tsp" and "ali535.tsp" instances, as illustrated below:

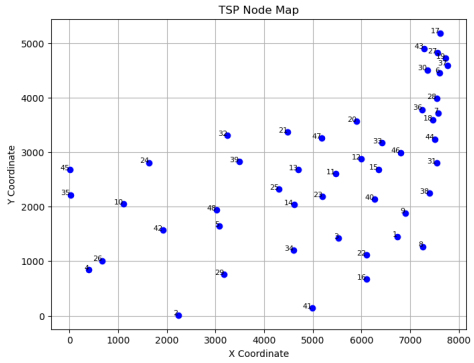

Figure 2: att48.tsp

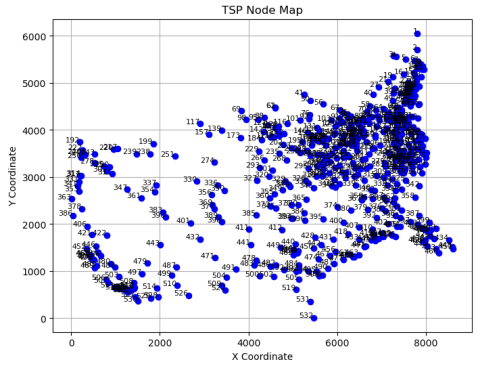

Figure 3: a280.tsp

3.1 Final Results

Our FunSearch project saw great success in improving the Traveling Salesman Problem solution. We started with a score that represented the total travel distance, which was quite high at -838152. After running 100 training cycles, we significantly improved the score to -2915.
In simpler terms, this means that the route our solution suggests for the salesman to travel from city to city got a lot shorter. This big improvement shows how well our system learned and got better at solving the problem. It’s like teaching someone a faster way to get through a maze, and they keep finding quicker paths every time they try.
This progress is a result of the smart way FunSearch works. It makes little changes each time, learns which changes lead to better routes, and keeps those while throwing out the rest. It’s like finding the best pieces of a puzzle one by one until you see the whole picture.
So, in the end, we achieved a route that is much more efficient than what we started with, and this shows the power of using FunSearch for problems like this.

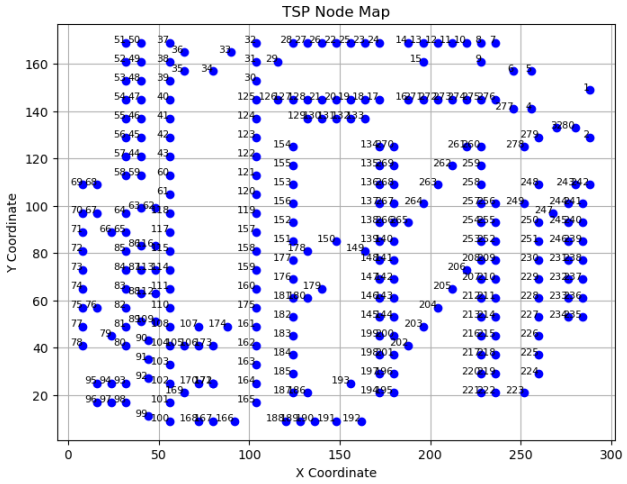

Figure 4: ali535.tsp

3.2 Insights

The FunSearch framework, with its iterative and evolutionary approach, demonstrates its efficacy in refining strategies over time. It suggests that for problems where the landscape is vast and filled with potential routes, an exploratory method that continually learns and adapts is not just beneficial but perhaps essential.

Moreover, the success of this project highlights the value of collaboration between different components of the AI system. The integration of a Pretrained Language Model with evolutionary algorithms and safety measures such as the Sandbox creates a symbiotic environment where each part contributes to the strength and flexibility of the whole.

In practical terms, the outcomes of this project offer promising implications for various applications. The principles applied here can be adapted to other complex scheduling and routing problems across industries like logistics, networking, and urban planning.

In conclusion, the FunSearch project for the TSP stands as a clear indicator of the robustness of iterative learning systems and their ability to produce concrete, impactful results. It invites us to consider the broader possibilities of AI in problem-solving and to continue pushing the boundaries of what these intelligent systems can achieve.

References

[1] Y. Bang, S. Cahyawijaya, N. Lee, et al. A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity. *arXiv preprint arXiv:2302.04023*, 2023.

[2] A. Borji. A categorical archive of chatgpt failures. *arXiv preprint arXiv:2302.03494*, 2022.

[3] X. Chen, M. Lin, N. Schärli, et al. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.

[4] B. Romera-Paredes, M. Barekatain, A. Novikov, et al. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024.