IDAHO STATE UNIVERSITY
COMPUTER SCIENCE

Project Group: 6 April 25, 2024

CS 4412 Advanced Algorithms: Traveling Salesperson Group Project

by

Saim Ishtiaq Shipra Kumari Jeongeun Lee
Victoria Weir

Abstract:

The Traveling Salesperson Problem (TSP) is a classic optimization problem with applications in various domains. This project aims to explore different algorithmic approaches to solving the TSP and compare their performance empirically. We implemented a greedy algorithm as a baseline and developed a custom algorithm to achieve high accuracy approximate solutions. Through empirical analysis, we evaluated the solution quality, runtime, and scalability of each algorithm on hard-level TSP instances. Our findings provide insights into the trade-offs between solution quality and computational complexity in solving the TSP.

1 Introduction

The TSP involves finding the shortest possible route that visits each city exactly once and returns to the origin city. Due to its NP-complete nature, solving the TSP optimally becomes computationally infeasible for large problem instances. Hence, various approximation algorithms and heuristics have been devised to find near-optimal solutions efficiently. In this project, we implement and compare a greedy algorithm, a custom algorithm, and a branch and bound algorithm (from a previous project) for solving the TSP. By analyzing their performance on hard-level TSP instances, we aim to gain insights into the effectiveness and scalability of different approaches.

2 Methods

- **Greedy Algorithm Implementation:** We implemented a simple greedy algorithm that iteratively selects the nearest unvisited city until all cities are visited. This algorithm serves as a baseline for comparison.
- **Branch and Bound Algorithm:** We utilized a Branch and Bound algorithm, similar to the implementation in a previous project, for solving the TSP optimally. This algorithm systematically explores the search space of possible tours, pruning branches that are guaranteed to lead to suboptimal solutions. While providing optimal solutions for small instances, the computational complexity of Branch and Bound increases exponentially with the number of cities.
- **Custom Algorithm Development:** In addition to the greedy algorithm and Branch and Bound, we devised a custom algorithm, Simulated Annealing Algorithm approach.

2.1 Greedy Algorithm Implementation:

Initial Solution: Initialize an empty route where a list of cities and a time allowance as input. Select a random starting city and add it to the route.

Main Loop : There are still unvisited cities, so find the nearest one to the last city in the route. Add this city to the route. Repeat this, but the algorithm runs until the

time allowance is exceeded. For each iteration, a random permutation of cities is generated. When all cities are visited then come to the first city when the route was started. Hence, it will complete the route. This algorithm does not find an optimal solution but gets a good solution.

2.2 Branch and Bound Algorithm:

Initialization: Start with an empty route and a list of cities as input. Initialize the initial cost matrix representing the distances between cities. Calculate the lower bound using the reduced cost matrix technique.

Setup Phase: Set up the priority queue to store partial tours. Initialize it with the initial state (empty tour, lower bound).

Main Loop: While the priority queue is not empty, explore the state space by dequeuing a partial tour from the priority queue. For each partial tour, generate child states by adding one more city to the tour. Calculate the lower bound for each child state and enqueue them into the priority queue. Prune branches that exceed the current best solution.

Termination: Terminate the algorithm when the priority queue is empty or when the time limit is reached.

Performance Metrics: Track performance metrics such as the quantity of generated and pruned states, queue size, and the final solution quality.

The Branch and Bound algorithm systematically explores the search space of possible tours, pruning branches that are guaranteed to lead to suboptimal solutions. While providing optimal solutions for small instances, the computational complexity of Branch and Bound increases exponentially with the number of cities.

2.3 Simulated Annealing Algorithm

Initial Solution: Start with an initial solution, which could be a random tour that visits each city exactly once.

Temperature Initialization: Set an initial temperature, which controls the probability of accepting worse solutions.

Iteration: We will not stop until we find a solution

In this we start with a random solution, like a route for a road trip. We then make small changes to this route, like rearranging the order of cities. After each change, we calculate how good the new route is by checking its total distance. If the new route is better, we always accept it. But if it's worse, we might still accept it, but with a lower chance as we "cool down" over time. This helps us explore different routes widely at first and then focus on improving the best ones as we "cool down," eventually finding a good solution.

Stop the algorithm when the temperature falls below a certain threshold or after a predetermined number of iterations.

2.3.1 More Information about this Algorithm:

- When we start, the temperature is high, so we can try out many different routes.
- We're okay with accepting slightly longer routes sometimes, especially when the temperature is high, because it helps us explore more options and avoid getting stuck in bad routes.
- As we go on, the temperature decreases, and we become more picky. We only accept better routes, like cooling metal to make it stronger.
- Eventually, when the temperature is very low, we stop accepting worse routes altogether and focus on improving the best one we've found.

3 Empirical Analysis:

3.1 Greedy Algorithm Implementation:

Time Complexity: For Initialization : $0(1)$ Main Loop: For generating a random permutation, it is $O(nlogn)$ where *n* is number of cities. $O(n^2)$(iterating over n cities and finding the nearest unvisited city at each step). The overall time complexity of the greedy algorithm for the Traveling Salesman Problem (TSP) can be approximated as $O(n^2)$.

Space Complexity: The total space complexity of the greedy algorithm for TSP is $O(n)$ due to the storage required for the input data, permutation, and route.

3.2 Branch and Bound Algorithm:

3.2.1 reduceCostMatrix:

Time Complexity:

- Iterating over each row of the cost matrix contributes $O(n^2)$ time complexity, where $O(n)$ is the number of cities.
- Finding the minimum cost in each row requires $O(n)$ operations, resulting in a total time complexity of $O(n^3)$ for row operations.
- Similarly, iterating over each column and finding the minimum cost in each column also contributes $O(n^3)$ time complexity.
- Overall, the time complexity of the reduceCostMatrix function is $O(n^3)$ due to the nested loops and minimum finding operations.

Space Complexity:

- The space complexity of the reduceCostMatrix function primarily depends on the space required to store the cost matrix itself.
- Since the function operates in-place on the given cost matrix, the additional space complexity is negligible.
- Therefore, the space complexity is O(1), indicating constant space usage regardless of the size of the cost matrix.

3.2.2 Overall:

Time Complexity: Setup Phase: $O(n^3)$ due to reduceCostMatrix being called. Initialization: $O(logn)$, where n is the number of states in the priority queue. Loop:

- Exploring child states: $O(n^2)$
- Updating results: $O(n^2)$

- Total time spent: $O(1)$

Overall Time Complexity: $O(n^3)$, where n is the number of cities.

Space Complexity: From the reduceCostMatrix, $O(1)$, the priority queue, $O(n)$, to other space variables, $O(1)$, the total is $O(n)$.

3.3 Simulated Annealing Algorithm:

Time Complexity:

- $O(n^2)$, where n is the number of cities where while loop runs until the stopping criteria are met.
- Each operation to change the route and generate a new TSPSolution object inside the loop takes $O(n)$ time.
- $accept$ and $probabilityAccept$ Function will have $O(1)$.
- Overall the dominant factor is the simAnnealing function, which has a time complexity of $O(n^2)$.

Space Complexity: $O(n^2)$, where n is the number of cities.

- $O(n^2)$ space utilization is a result of the simAnnealing function and related data structures (such as temp and $temp_bssf$).
- More space is needed for the storing of variables and results, which is constant, and the input data (cities), which takes $O(n)$ space.

4 Results:

Some notes about the table, the branch and bound algorithm time limit is not 60 but a larger time limit such as 300 seconds or 5 minutes. For the random algorithm, we used the default algorithm provided with the original framework (this acts as baseline for the greedy algorithm).

For the results in this table they were done in Hard level. For the greedy algorithm report improvements were found by putting the improvement over random as a

fraction (calculate this as greedyCost/randomCost).

	Random		Greedy	
City	Time (Sec)	Path Length	Path Length	% of Random
15	0.00102	18935	20387	1.08
30	0.033	48725	34842	0.72
60	29.09	83277	86015	1.03
100	TB	TB	TB	TB
200	TB	TB	TB	TB

Table 1: Results Part 1

	Branch and Bound			Simulated Annealing		
City	Time(Sec)	Path Length	% of Greedy	Time(Sec)	Path Length	% of Greedy
15	1.69	9316	0.46	0.33	11144	0.55
30	600	14304	0.41	0.63	18892	0.54
60	TB	TB	TB	1.30	28525	0.33
100	TB	TB	TB	2.15	40492	-
200	TB	TB	TB	4.28	56015	-

Table 2: Results Part 2

4.1 Analysis:

As seen in the tables, we tested different algorithms which are Random, Greedy, Branch and Bound, and Simulated Annealing for solving the Traveling Salesman Problem (TSP). For the Random, the result demonstrates varying performance in terms of solution quality and efficiency across different city sizes. Greedy algorithm shows improvements over the Random with 1.08 percentage. The Branch and Bound algorithm performs well when given more time (600 seconds), finding better solutions than other algorithms. However, it becomes impractical for larger problems like cities with more than 30. Simulated Annealing algorithm offers a promising strategy for optimization, delivering decent solutions within reasonable time with any city size.

In summary, the Branch and Bound algorithm provides optimal solutions for small instances but struggles with scalability. On the other hand, the Simulated Annealing algorithm presents a promising approach for addressing larger TSP instances.

By exploring routes iteratively, Simulated Annealing can efficiently navigate complex solution spaces. Further research is warranted to refine and optimize Simulated Annealing for even larger TSP instances while maintaining acceptable solution quality and efficiency.

5 Future Work:

1. Parameter Tuning: Experiment with different values for parameters such as the initial temperature, cooling rate, and stopping temperature in the simulated annealing process. Fine-tuning these parameters could lead to improved performance and better-quality solutions.

2. Adaptive Parameters: Implement adaptive strategies for adjusting the parameters of the simulated annealing algorithm during the optimization process. For example, dynamically adjusting the temperature schedule based on the progress of the search or the characteristics of the problem instance.

3. Advanced Cooling Schedules: Investigate more sophisticated cooling schedules beyond the standard geometric or exponential cooling. Adaptive cooling schedules or non-linear cooling schedules could be explored to tailor the annealing process to the specific characteristics of the problem instance.

6 Conclusions

Overall, this project implements various algorithms for solving the Traveling Salesperson Problem (TSP). Through empirical analysis and comparison of the Branch and Bound, Greedy, and Simulated Annealing algorithm, we have gained a comprehensive understanding of their performance characteristics across different problem sizes.

The Greedy demonstrates moderate improvements over the Random, with a 1.08 percent increase. However, its inability to find optimal solutions and scalability limitations restrict its applicability to smaller problem sizes.

Conversely, the Branch and Bound algorithm shows particular performance in finding optimal solutions for small TSP instances. However, its complexity grows exponentially, making it difficult to scale up, which means it is not practical for larger

problems.

In contrast, the Simulated Annealing algorithm presents a promising strategy for optimizing TSP solutions. This algorithm efficiently navigates complex solution spaces while delivering decent solutions within reasonable time. Although it may not provide the most optimal solutions, its ability to handle larger problem instances with solution quality and efficiency makes it a feasible approach for real-world applications.

Finally, for possible future work, we would work on modifying the simulated annealing algorithm to better ensure that the algorithm provides a more enhanced performance than it currently does. One way of accomplishing this would be to experiment with different values for the parameters such as the initial temperature, cooling rate, etc in the simulated annealing process to observe and inspect for any noticeable improvement in the algorithm. In addition to this, we could also try to implement adaptive strategies for adjusting the parameters of the simulated annealing algorithm during the optimization process. Finally, we could also investigate and experiment with more sophisticated cooling schedules beyond the standard geometric or exponential coding to see if they make a positive difference.

In conclusion, the Simulated Annealing algorithm offers a promising avenue for addressing larger TSP instances efficiently. Further research and optimization of Simulated Annealing could lead to enhanced performance and broader applicability in solving large-scale TSP problems as mentioned above.

7 References

https://www2.cose.isu.edu/~bodipaul/courses/sp24/4412/projects/project6_files/TSPGroup_python.php

https://www.youtube.com/watch?v=35fzyblVdmA