Singapore Institute of Management

UOW AUSTRALIA

CSCI-323 Modern Artificial Intelligence

Group Project - The Traveling Salesman Problem Comparison between Q-Learning and Dynamic Programming

(Group 53)

	Member Name	UOW-ID
1	Peh Yi Xiong	8080987
2	Kieron Tyas Tang	7671878
3	Kho Li Ming	8335102
4	Chew Wayne	8492839
5	Lim Yi Xiang	8466075

1. Introduction

The Traveling Salesman Problem is a classic optimization problem in computer science. It asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"

Traveling salesman problem is a combinatorial optimization problem, where one has to find an optimal solution in a finite list of solutions itself. In this case, we will attempt to find the most optimal route (shortest distance route) among the list of possible routes that can visit each city once and return right back to the starting city.

The Travelling Salesman Problem has numerous real-world applications, making it a significant challenge across various fields. In logistics and transportation, the traveling salesman problem is commonly applied to routing delivery trucks, planning efficient transportation routes, and optimizing supply chains. By finding optimal solutions, companies can achieve substantial cost savings and reduce fuel consumption(Sbihi, Abdelkader; Eglese, Richard W. (2007)). In manufacturing. It plays a crucial role in minimizing the travel time of machines or robots during the assembly of products, leading to faster and more efficient production processes (Eskandarpour, Majid; Dejax, Pierre; Miemczyk, Joe; Péton, Olivier (2015)).

2. Background theory

Theoretical Foundation

Traveling salesman problem was formulated in the 19th century by William Rowan Hamilton and Thomas Kirkman, gained interest in the 1930s and became popular during the 1950s-1960s. During those times, this problem came to be called by one mathematician as a Hamilton game, and also whose solution is the shortest Hamilton path in a weighted graph. (Biggs, Lloyd, and Wilson)

Theoretical developments

Over the years, as mathematicians came to tackle this problem, they created certain different solutions, some of them which will be discussed below, with one being our chosen method.

Approaches to tackle the problem

- **Brute Force Approach:** We simply calculate all the possible paths and narrow it down to the shortest path. But this process quickly becomes impractical with large datasets.

- **Dynamic Programming:** Breaking down the problem into smaller subproblems, and storing the results, allows the computer to avoid a lot of redundant computation that would happen in the Brute Force Approach. This still takes a bit of time and resources to compute, but is overall better than the brute force approach in terms of time complexity. With Dynamic programming it is possible to get the best possible result.

- **Q - Learning:** Q-Learning is a reinforcement learning Algorithm used to find optimal solutions in environments where an agent learns to make decisions through trial and error. Core idea of Q-Learning is to estimate the value of taking a specific action in a given state, which is represented by a Q-value.

3. Solutions, evaluation, and discussions

3.1. Solutions

For this project, we chose to implement **Dynamic Programming** and **Reinforcement Learning (Q-Learning)** to solve the Traveling Salesman Problem . These algorithms were selected as part of an exploratory approach to understanding different methods for solving combinatorial optimization problems like the Traveling Salesman Problem. Dynamic Programming was chosen as one of our solutions because the Traveling Salesman Problem exhibits both optimal substructure and overlapping subproblems, which were key factors in our decision to use Dynamic Programming. Q-Learning was chosen for its ability to learn and adapt to complex environments through reinforcement learning.

3.1.1. Dynamic Programming

Dynamic Programming (DP) is a method for solving complex problems by breaking them down into simpler subproblems. By solving each subproblem only once and storing the results, it avoids redundant computations, leading to more efficient solutions across a wide range of problems (Jain, 2024).
The optimal substructure of the Traveling Salesman Problem is evident in the way the optimal route to visit all cities can be constructed from the optimal tours of smaller subsets of cities. The overlapping subproblems are encountered when solving the Traveling Salesman Problem using brute force, where the program may recompute the same path multiple times while considering different combinations of cities. This leads to inefficiencies and redundancies.

Although Dynamic Programming does not eliminate the exponential nature of the Traveling Salesman Problem, it significantly improves the efficiency of finding an optimal solution compared to brute force methods.

There are two approaches to Dynamic Programming: Tabulation(Bottom-Up Approach) and Memoization(Top-Down Approach).

Tabulation

Tabulation handles smaller subproblems first and builds up to the overall solution. In the context of the Traveling Salesman Problem, the result to travel to each city is calculated, then the result is used to calculate the result for traveling to two cities, this process is continued until all cities are calculated. The results are then stored in a table to avoid redundant calculations. Tabulation uses the Iterative approach, avoiding the potential issues with deep recursion and has lesser overhead from function calls.

Memoization

Memoization starts by solving the main problem and breaking it down into sub-problems recursively. The solutions for these problems are computed, the results are stored in to avoid redundant calculations. In the context of the Traveling Salesman Problem, the whole path is calculated for each recursive call and the result in a table.Memoization uses a recursive approach, hence, if the number of cities is large, the recursion might become very deep, leading to a lot of recursive function calls.

For our project, we chose the **memoization** method because it's easier to implement, and there's no risk of stack overflow errors, given that our system can handle up to 20 cities without issues.

We implement memoization by storing the minimum cost of visiting all cities starting from a given city and a bitmask to track which cities have been visited. The base case returns the cost of returning to the starting city after all cities are visited. It reconstructs the optimal path based on the stored information.

3.1.2. Reinforcement Learning (Q Learning)

In this study, we implemented a Q-Learning algorithm to tackle the Traveling Salesman Problem. The algorithm was designed with the following key components:

- **State and Action Representation:** The Traveling Salesman Problem was modeled as a reinforcement learning task, where each city represents a state. The available actions from each state correspond to the cities that have not yet been visited. To enforce this, a **masking** mechanism was employed, which marked cities that had already been visited as unavailable (masked). This ensured that the Q-Learning algorithm did not revisit cities, thereby maintaining the validity of the route being constructed.
- **Q-Table Initialization:** A Q-Table was initialized to store the expected rewards for state-action pairs, with rewards set as the negative distance between each pair of cities. Initially, all entries in the Q-Table were set to zero, reflecting a lack of prior knowledge about the environment.
- **Epsilon-Greedy Strategy:** This strategy balances exploration and exploitation during the learning process. At each step, a random number is generated; if it is less than epsilon, the algorithm explores a random action (city), otherwise, it exploits by choosing the action with the highest Q-value. The mask was applied during this process to ensure that only unvisited cities were considered for the next action.
- **Epsilon-Decaying Strategy:** To further optimize learning, the epsilon value was initially set high to encourage exploration and then gradually decayed to 0.1. This approach allowed the algorithm to explore a diverse set of routes in the early stages and focus on exploiting the best routes as the learning progressed.
- **Learning Process:** The algorithm utilized Bellman's equation for optimal decision-making (TechTarget, 2024). The Q-Table was updated recursively using the equation. To ensure the algorithm converges to a near-optimal solution, the learning process was terminated after 10,000 iterations with no improvement in the route cost.

  Bellman's equation: (TechTarget, 2024)
  $$Q(s,a) = Q(s,a) + \alpha * (r + \gamma * \max(Q(s',a')) - Q(s,a))$$

  The equation breaks down as follows:

- *Q(s, a)* represents the expected reward for taking action, *a,* in state, *s*.
- *r is the actual reward received for that action.*
- *s′* refers to the next state.
- *α* is the learning rate.
- *γ* is the discount factor.
- *max(Q(s', a'))* is the highest expected reward for all possible actions in state s′.

Parameter Tuning:
To optimize the performance of the Q-Learning algorithm, a range of hyperparameters were tested to find the most effective configuration. The key hyperparameters adjusted include:

- **EPOCHS:** The number of epochs was set to 500,000 to ensure extensive training and convergence of the Q-Table.
- **LEARNING_RATE (α):** Set to 0.01, this controls the rate at which the Q-Table values are updated.
- **GAMMA (γ):** A discount factor of 0.99 was used to weigh future rewards, promoting long-term optimization.
- **INITIAL_EPSILON:** Starting at 0.7, this high initial value encouraged exploration in the early stages of learning.
- **FINAL_EPSILON:** Reduced to 0.1, this final value balanced exploration and exploitation towards the end of the learning process.
- **EPSILON_DECAY:** A decay rate of 0.0001 was applied to gradually decrease epsilon, ensuring a smooth transition from exploration to exploitation.
- **MAX_NO_IMPROVE_EPISODES:** The algorithm was configured to stop after 10,000 episodes with no improvement in route cost, preventing unnecessary computations once a near-optimal solution was achieved.

These parameters were selected based on their impact on the learning efficiency and the quality of the resulting routes, leading to improved performance in solving the Traveling Salesman Problem.

3.2. Evaluation

Experimental Setup:
The algorithm was tested on subsets of cities from the ‘xqf131’ Traveling Salesman Problem dataset. The number of cities varied across experiments to assess the scalability and robustness of the Q-Learning approach. The true best-known cost for each instance was used as a benchmark for evaluating the algorithm’s performance.

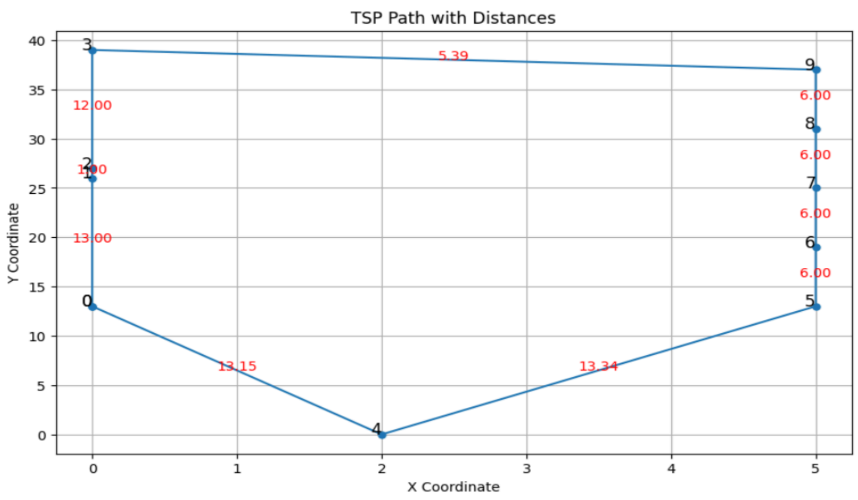
For both algorithms, we set up the environment similarly. Firstly we load the xqf131 Traveling Salesman Problem dataset. Next, To compute the minimum cost, the algorithm uses Euclidean distance based on the x and y coordinates of cities from the xqf131 dataset, storing the results in a distance matrix. This ensures the cost metric accurately reflects the geometric distances between cities.

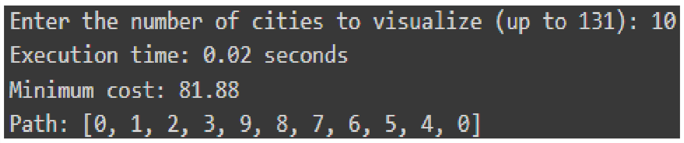
Performance Metrics:
The performance of the Q-Learning algorithm was primarily evaluated using the following metrics:

- **Minimum Cost:** The primary metric was the minimum cost (total route distance) of the route generated by the algorithm. This cost is calculated as the sum of the Euclidean distances between consecutive cities in the route. This metric directly measures the effectiveness of the Q-Learning approach in finding a near-optimal solution to the Traveling Salesman Problem.
- **Execution Time:** The second key metric was the time taken by the algorithm to find the minimum cost route. This metric reflects the efficiency of the algorithm in terms of computational resources and speed.

3.3. Results

Dynamic Programming

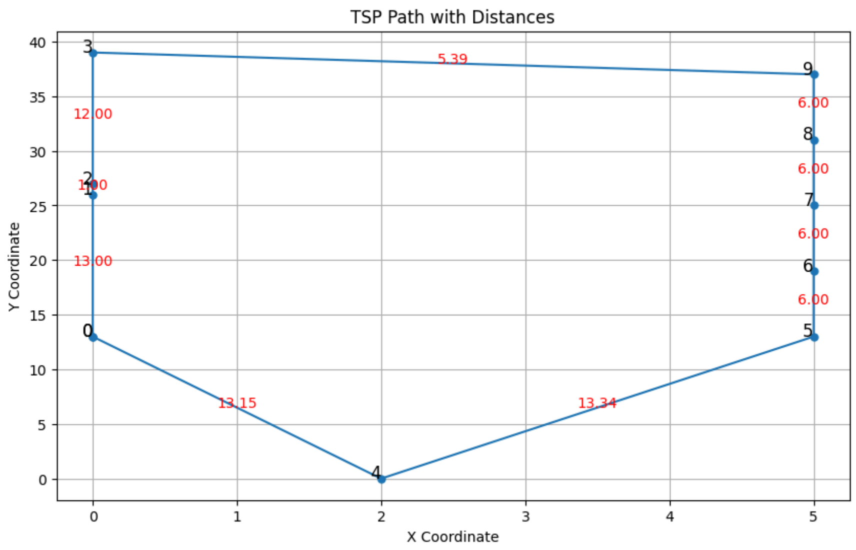


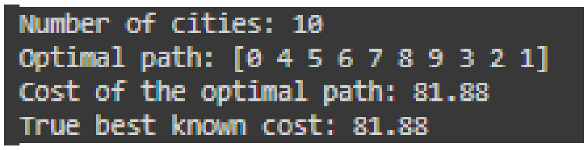


```
Enter the number of cities to visualize (up to 131): 10
Execution time: 0.02 seconds
Minimum cost: 81.88
Path: [0, 1, 2, 3, 9, 8, 7, 6, 5, 4, 0]
```

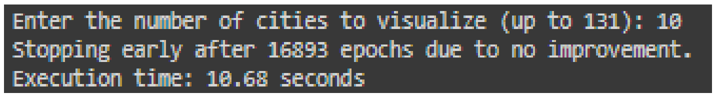
Dynamic Programming has successfully converged to the best-known distance, achieving this result in 0.02 seconds.

Reinforcement Learning(Q-Learning)





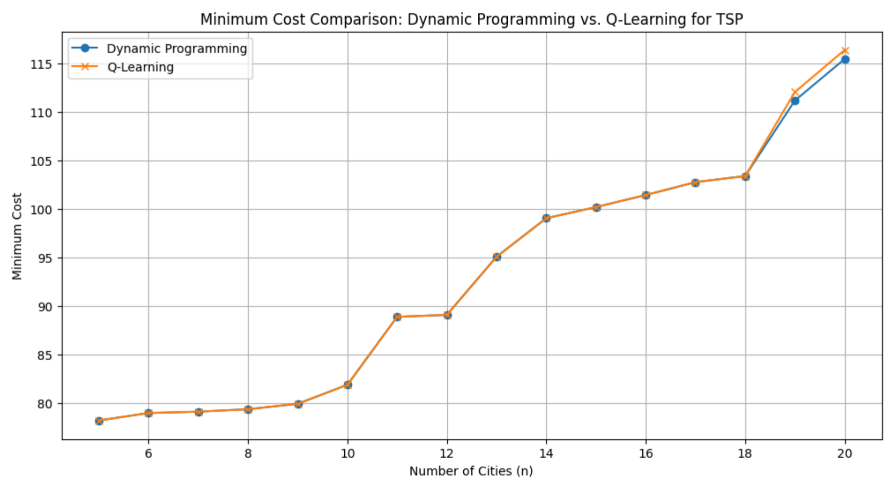
```
Number of cities: 10
Optimal path: [0 4 5 6 7 8 9 3 2 1]
Cost of the optimal path: 81.88
True best known cost: 81.88
```



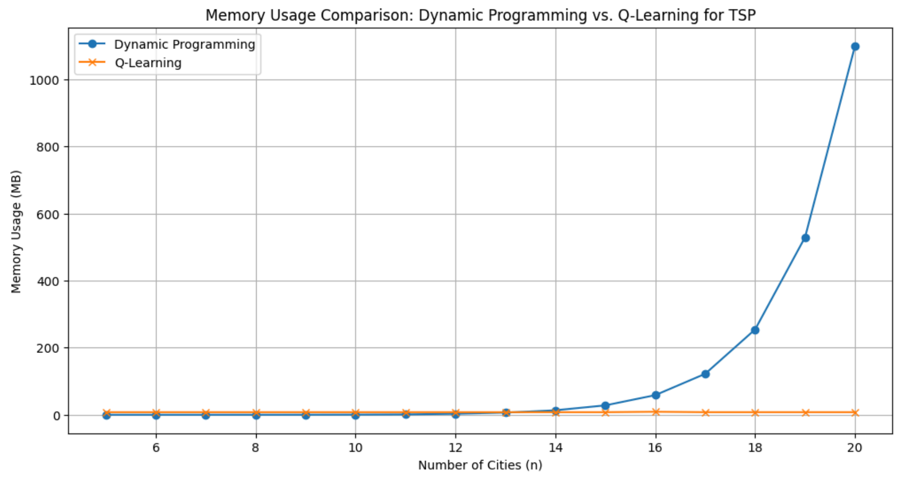
```
Enter the number of cities to visualize (up to 131): 10
Stopping early after 16893 epochs due to no improvement.
Execution time: 10.68 seconds
```

Q-Learning has successfully converged to a solution with the best known distance within 16893 epochs and took 10.68 seconds to execute.

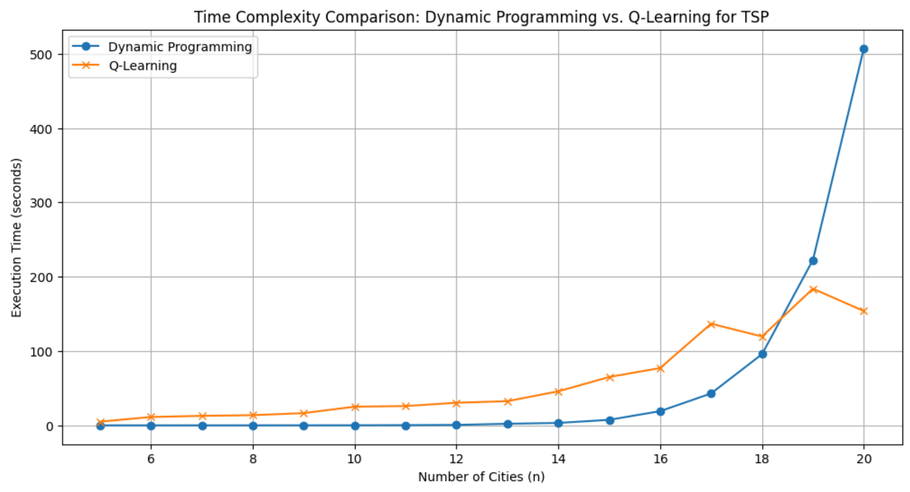
Performance between Dynamic Programming & Q-Learning



Dynamic Programming guarantees an optimal minimum cost by exhaustively evaluating all routes, while Q-Learning provides an approximate minimum cost that may improve with training but does not always guarantee the optimal solution.



Dynamic Programming has a space complexity of O(n×2ⁿ), due to the need to store the cost of visiting subsets of cities. In contrast, the Q-Learning has a space complexity of O(n²) as it requires maintaining a Q-table with entries for each possible state-action pair, making it more scalable in terms of memory usage for larger instances of Traveling Salesman Problem.



Dynamic Programming has a time complexity of O(n²×2ⁿ), considering all subsets of cities to find the optimal solution. Q-Learning, with a time complexity of O(episodes×n²), scales better with larger n but may need many episodes to converge and doesn't always ensure an optimal solution.

4. Conclusion

Dynamic Programming, with its memoization technique, offered a systematic approach to solving the Traveling Salesman Problem by breaking down the problem into smaller, manageable subproblems. This method, although effective in reducing redundant computations, still faced challenges with scalability as the number of cities increased. However, it consistently produced optimal solutions for the problem sizes we tested, demonstrating its reliability for smaller Traveling Salesman Problem instances.

Q-Learning, on the other hand, showcased its adaptability in tackling complex combinatorial optimization problems. The algorithm's ability to learn and improve through experience allowed it to find competitive solutions but imperfect, even with the inherent challenges of exploration versus exploitation. By fine-tuning the hyperparameters, we achieved a balance between exploration and exploitation, resulting in solutions that closely approximated the true optimal paths. Nonetheless, the random nature of Q-Learning led to some variability in the results, highlighting the importance of sufficient training and careful parameter selection. Throughout this journey, we encountered long code execution times, especially when tuning hyperparameters in Q-Learning. These challenges pushed us to learn how to optimize performance while balancing exploration and exploitation.

During the Q&A session, one particularly insightful question was raised: "In real-world problems, the distance between two points may vary depending on factors like traffic, road conditions, or fuel costs, which may change dynamically. How do you incorporate this in your algorithm?" This question highlighted the complexities of applying theoretical models to practical scenarios. It made us consider how our algorithms could be adapted to account for dynamically changing variables. While our current implementation assumes static distances, incorporating real-world factors would involve adjusting our algorithms to update distance matrices in real-time and re-evaluating paths as conditions change. This adds another layer of complexity and underscores the need for algorithms that can adapt to real-time data, such as its application in supply chain, something we plan to explore in future work. We were also asked to explain the concept of NP-hard problems in the context of the Traveling salesman problem. A problem is classified as NP-hard if It is at least as hard as the hardest problems in NP (nondeterministic polynomial time) and Every problem in NP can be reduced to it in polynomial time. Brute Forcing becomes infeasible as it has a factorial growth which is way beyond non polynomial time. Verifying whether a given route is the shortest among all possible routes requires comparing it with potentially exponentially many other routes, which is computationally infeasible for large inputs. There is no known polynomial-time algorithm to solve Travelling Salesman Problem for all instances

This project not only enhanced our technical skills in algorithm design and optimization but also deepened our understanding of Q-Learning and Dynamic Programming. It highlighted the importance of continual learning and adaptability when addressing complex challenges. Overall, the experience strengthened our foundation in advanced algorithms and provided valuable insights into real-world applications, lessons that will guide us in future projects.

Acknowledgements

Contributions	
Chew Wayne 8492839	20% (Coding, Slides, Report)
Peh Yi Xiong 8080987	20% (Coding, Slides, Report)
Kho Li Ming 8335102	20% (Coding, Slides, Report)
Lim Yi Xiang 8466075	20% (Coding, Slides, Report)
Kieron Tyas Tang 7671878	20% (Coding, Slides, Report)
	Total: 100%

References

Graph Theory, 1736–1936 by Biggs, Lloyd, and Wilson

Sbihi, Abdelkader; Eglese, Richard W. (2007). "Combinatorial optimization and Green Logistics" (PDF). *4OR*. **5** (2): 99–116. doi:10.1007/s10288-007-0047-3. S2CID 207070217. Archived (PDF) from the original on 2019-12-26. Retrieved 2019-12-26.

Eskandarpour, Majid; Dejax, Pierre; Miemczyk, Joe; Péton, Olivier (2015). "Sustainable supply chain network design: An optimization-oriented review" (PDF). *Omega*. **54**: 11–32. doi:10.1016/j.omega.2015.01.006. Archived (PDF) from the original on 2019-12-26. Retrieved 2019-12-26.

Hobé, Alex; Vogler, Daniel; Seybold, Martin P.; Ebigbo, Anozie; Settgast, Randolph R.; Saar, Martin O. (2018). "Estimating fluid flow rates through fracture networks using combinatorial optimization". *Advances in Water Resources*. **122**: 85–97. arXiv:1801.08321. Bibcode:2018AdWR..122...85H. doi:10.1016/j.advwatres.2018.10.002. S2CID 119476042. Archived from the original on 2020-08-21. Retrieved 2020-09-16.

Rohe, Andre. "xqf131 TSP Dataset." *VLSI* TSP *Instances*, Forschungsinstitut für Diskrete Mathematik, Universität Bonn. Available from TSP LIB, https://www.math.uwaterloo.ca/TSP/vlsi/index.html#XQF131. Accessed August 21, 2024.

Jain, S. (2024, July 30). *Dynamic Programming or DP*. GeeksforGeeks. Retrieved August 19, 2024, from https://www.geeksforgeeks.org/dynamic-programming/

"Q-learning." *TechTarget*, Retrieved August 21, 2024, from https://www.techtarget.com/searchenterpriseai/definition/Q-learning.