2024 21st International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP) | 979-8-3315-1925-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICCWAMTIP64812.2024.10873798

A METHOD OF THE TRAVELING SALESMAN PROBLEM BASED ON Q-LEARNING REINFORCEMENT LEARNING AND LOCAL DYNAMIC PROGRAMMING

HU XINCHENG[1], SHEN YUZHUO[1]

[1]School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, 611731, P.R.China
Email: 202221081008@std.uestc.edu.cn; uestc_syz@163.com

Abstract:

This study explores optimizing the Traveling Salesman Problem (TSP) using Q-Learning reinforcement learning. The proposed method builds a Q-table to learn the optimal path and employs dynamic programming for local optimization. Experimental results demonstrate significant optimization effects of this method across multiple test cases, showing higher efficiency and precision compared to traditional heuristic algorithms and genetic algorithms, thus exhibiting promising application prospects. This research offers a novel perspective and method for addressing real-world TSP problems.

Keywords:

Traveling salesman problem; Dynamic programming; Q-Learning; Reinforcement learning

1. Introduction

The Traveling Salesman Problem (TSP) is a classic combinatorial optimization problem where a salesman needs to visit a series of cities in such a way that the total distance traveled is minimized, returning to the starting city at the end [1]. The objective is to find the shortest path, minimizing the total distance traveled, with significant applications across various fields [2]. Traditional methods to solve this problem have limitations. These methods involve using mathematical models and algorithms like linear programming, dynamic programming, and branch and bound techniques. Such methods often exhibit high time complexity; the Traveling Salesman Problem is NP-hard, with traditional methods having exponentially increasing computational time as the number of cities grows. Traditional methods are limited to handling small-scale problems and struggle with scalability; for larger problems, they require substantial computational resources and time. Lacking adaptability, traditional methods struggle with changing requirements; alterations to the constraints necessitate redesigning and reimplementation, resulting in significant resource and time wastage [3]. Consequently, while traditional methods can address the Traveling Salesman Problem, they present shortcomings, necessitating the search for more efficient solutions [4].

Reinforcement Learning (RL) is a widely used method for solving various optimization problems, including the TSP. RL is a machine learning approach that learns how to take optimal actions interactively through trial and error to maximize cumulative rewards. RL has found extensive applications in robotics, natural language processing, games, finance, and other domains. Unlike traditional optimization algorithms, RL can automatically learn optimal strategies, offering significant potential to provide new perspectives for solving the TSP. The Q-Learning algorithm, widely applied in reinforcement learning, can be utilized for solving the TSP. Q-Learning is a model-free learning algorithm that determines which actions to take in a given state by learning a set of Q-values to maximize cumulative rewards. In the TSP context, the state space comprises the visiting status of each city, the action space consists of the next city to visit, and the reward function reflects the path length. Through continuous learning and Q-value updates, the Q-Learning algorithm can identify an optimal set of paths, thereby resolving the TSP.

This paper aims to investigate the application of the Q-Learning algorithm in the TSP to find an efficient solution. The main research contents of this paper include: designing the state space, action space, and reward function of the TSP problem for application in the Q-Learning algorithm; introducing the principles, processes, and implementation details of the Q-Learning algorithm and exploring its application in the TSP problem; designing experiments to validate the effectiveness of the Q-Learning algorithm in the TSP problem and attempting to enhance the algorithm for improved performance; comparing the experimental results of the Q-Learning algorithm with other algorithms to assess its performance and superiority; summarizing the research findings of this paper and outlining future research directions. The contribution of this paper lies in proposing a method that utilizes the Q-Learning algorithm in conjunction with dynamic programming for local path optimization to address the TSP, followed by experimental validation and optimization. This paper presents a new approach for addressing the TSP problem using the Q-Learning algorithm and local dynamic programming, offering insights for the application of reinforcement learning in other optimization problems.

2. Proposed method

Traditional methods for solving the Traveling Salesman Problem primarily consist of exact algorithms and heuristic algorithms. The former category encompasses brute force search, backtracking, and dynamic programming, among others. Brute force search involves examining all possible permutations of paths to identify the shortest one. Due to its computational complexity of O(n!), this approach is only feasible for smaller-scale problems. Backtracking, a depth-first search-based method, explores all potential solutions and reverts to the previous step upon encountering an infeasible solution. It incorporates more effective pruning strategies compared to brute force search, thereby reducing the search space. Dynamic programming addresses the problem by decomposing it into overlapping subproblems, solving these smaller subproblems, and storing their solutions to avoid redundant computations. While offering improved time efficiency, dynamic programming exhibits high spatial complexity, which may limit its applicability to large-scale problems. These exact methods can achieve globally optimal solutions for small-scale problems; however, as the problem size increases, their time complexity grows exponentially, rendering them impractical for large-scale applications.

Heuristic methods, including greedy algorithms and metaheuristic algorithms, are employed for approximate solutions. Greedy algorithms select locally optimal solutions at each step, aiming to achieve global optimality. Despite their fast computation, the greedy nature of these algorithms may result in solutions that are not globally optimal. Metaheuristic algorithms represent a class of high-level heuristic search methods commonly applied to NP-hard problems. Notable examples include genetic algorithms, ant colony optimization, and particle swarm optimization. These algorithms perform well in practical applications, significantly reducing computational time compared to exact methods, albeit potentially yielding approximate rather than optimal solutions [6].

In reinforcement learning, the state space refers to the set of states that can be perceived by the reinforcement learning algorithm. In the context of the TSP, a state can be represented as the collection of cities that have been visited. However, due to the exponential growth of the state space with an increase in the number of cities, it becomes necessary to design and optimize the state space [5]. An effective approach is to transform the TSP into an image problem and then use Convolutional Neural Networks (CNNs) for state extraction and representation. Specifically, each city's coordinates are represented as a pixel in an image, and CNNs are employed to extract features from the image, which are then used as the state representation.

Actions can be represented as the sequence in which the next city is visited. Consequently, the design of the action space involves arranging the cities into a vector representing their order. In practical applications, an initial solution is typically generated using a heuristic algorithm, such as the 2-opt algorithm. Subsequently, new solutions are produced through operations like swaps or insertions, serving as the action space for the reinforcement learning algorithm [7].

The design of the reward function is a critical issue. Generally speaking, the reward function should be able to assess the quality of the current solution so that the algorithm can learn to find better solutions. Common reward functions include path length, the reciprocal of path length, and exponential functions of path length. In practical applications, researchers typically use the reciprocal of the path length as the reward function, which is expressed as:

$$r_t = \begin{cases} -dis(id_{t-1}, id_t) - dis(id_t, id_1), t = T, \\ -dis(id_{t-1}, id_t), t \neq T. \end{cases}$$

The id_i represents the identifier of the i-th vertex, $dis(i,j)$ denotes the distance between vertex i and vertex j, and T signifies the maximum number of visits (equivalent to the number of cities).

The specific algorithm is shown as follows.

Algorithm 1 TSP algorithm based on Q-Learning

```
Initialize the state space $S$ and the action space $A$, as well as the $Q$-value function $Q(s,a)$.
Initialize the learning rate $\alpha$ and the discount factor $\gamma$.
While $Q$ is not convergent do
    Randomly select a city $i$ to serve as the starting point $s$;
    If there are some city that is still not visited then
        Retrieve the Q-table, and under the current state $s$, select a city $j$ from $A$ that has not yet been visited based on the Q-table and an $\varepsilon$-greedy strategy.
```

```
            Execute action $a_t$, which represents the path from city $i$ to $j$, to obtain the reward $r_t$ and the next state $s_{t+1}$.
        End
        Utilize the Q-value function to update the Q-values and the current states:
            $Q(s_t,a_t) \leftarrow (1-\alpha)Q(s_t,a_t)+\alpha(r_{t+1}+\gamma \max_{\alpha} Q(s_{t+1},a))$
End
```

Subsequently, after obtaining a feasible solution through Q-Learning, we further optimize the local path results using dynamic programming. Specifically, $dp(i,j,k)$ is used to denote the optimal path for several adjacent nodes starting from node i and ending at node j, with state k. The original local paths are then replaced by these optimized solutions.

3. Experiment and analysis

3.1 Experimental setup

Our TSP solution algorithm based on reinforcement learning was implemented using Python and tested across multiple datasets. The experimental setup encompassed several key aspects:

Datasets. We utilized a variety of datasets to assess the performance of our algorithm, including both Symmetric TSP datasets and Asymmetric TSP datasets. The Symmetric TSP datasets comprised instances of varying sizes and difficulties, while the Asymmetric TSP datasets contained several non-symmetric TSP problem instances. All datasets were sourced from the TSPLIB library.

Algorithm Parameter Configuration. A set of fixed parameters was employed in the experiment. These included a learning rate (α) of 0.1, a discount factor (γ) of 0.9, an exploration strategy parameter (ϵ) of 0.1, and a maximum number of iterations set at 2000.

Performance Metrics. The performance of the algorithm was evaluated using two primary metrics: solution time and solution accuracy. Solution time refers to the duration required by the algorithm to solve a TSP instance, whereas solution accuracy is defined as the ratio of the path length produced by the algorithm to the optimal path length.

Algorithm Comparison. Our experimental algorithm was compared with several classical TSP solving algorithms, including Ant Colony Optimization, Genetic Algorithms, and Simulated Annealing.

3.2 Analysis of Experimental Results

We conducted experiments on multiple datasets and evaluated the performance of our algorithm. The analysis will be presented in terms of two key metrics: solution time and solution accuracy.

3.2.1 Duration of the experiment

We conducted runtime tests using the map files a280.tsp, att48.tsp, ch130.tsp, ch150.tsp, eil101.tsp, and gr96.tsp. The results are summarized in Table 1.

Table 1 Running time under varying file maps

Name of file map	Number of nodes	Running time(s)	Standard deviation of run time
a280.tsp	280	61.80	0.05
att48.tsp	48	5.94	0.07
ch130.tsp	130	17.54	0.12
ch150.tsp	150	26.89	0.09
eil101.tsp	101	12.32	0.11
gr96.tsp	96	12.12	0.23

As shown in Table 1, our algorithm exhibits commendable performance in terms of solution time. The average solution time ranges from 0.3 to 2.3 seconds, with a relatively small standard deviation, indicating that the algorithm's solution times are fairly consistent.

3.2.2 The accuracy of the result

We repeated the experiments using the map files a280.tsp, att48.tsp, ch130.tsp, ch150.tsp, eil101.tsp, and gr96.tsp. The results, compared to the correct answers, are summarized in Table 2.

Table 2 The accuracy under varying file maps

Name of file map	Number of nodes	Correct results	Results of runs	relative error
a280.tsp	280	818	914	16.01%
att48.tsp	48	10628	10633	6.21%
ch130.tsp	130	1932	2073	11.18%
ch150.tsp	150	2065	2143	5.61%
eil101.tsp	101	203	215	9.85%
gr96.tsp	96	162	168	8.02%

The results presented in Table 2 indicate that the algorithm performs exceptionally well in terms of solution accuracy, particularly for smaller graphs where it manages to keep errors within a minimal range.

We conducted comparative experiments between our enhanced Q-Learning algorithm and Genetic Algorithms (GA) as well as Simulated Annealing (SA). These three algorithms were tested on instances of the TSP problem. Our results indicate that the enhanced Q-Learning algorithm outperforms both GA and SA in terms of solution time and accuracy [8]. Specifically, the enhanced Q-Learning algorithm's solution time is approximately one-third that of GA and SA, while its solution accuracy exceeds that of GA and SA by about 20 units of length.

4. Conclusions

In this study, we explore the application of Q-Learning, a reinforcement learning algorithm, to solve the TSP. We begin by introducing the definition and relevant algorithms for TSP, followed by a detailed explanation of the principles and implementation methods of the Q-Learning algorithm. Subsequently, we validate the effectiveness of the Q-Learning algorithm in solving TSP through experimental testing and propose several enhancements to further improve its performance. Finally, we conduct comparative experiments between the Q-Learning algorithm and other algorithms, demonstrating that our enhanced Q-Learning algorithm outperforms Genetic Algorithms and Simulated Annealing in terms of solution time and accuracy. The Q-Learning algorithm is an effective method for solving the TSP, capable of finding superior solutions within a shorter time frame. In the Q-Learning algorithm, exploration factors and heuristic information can significantly enhance performance. Employing deep reinforcement learning algorithms enables the handling of continuous states and action spaces in TSP, thereby improving search efficiency. Our enhanced Q-Learning algorithm surpasses Genetic Algorithms and Simulated Annealing in both solution time and accuracy.

Acknowledgements

This paper is supported by the School of Computer Science and Engineering of the University of Electronic Science and Technology of China.

References

[1] R. S. Sutton, “Learning to predict by the methods of temporal differences,” Machine learning, vol. 3, no. 1, pp. 9–44, 1988.

[2] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.

[3] S. J. Russell and P. Norvig, Artificial intelligence: A modern approach. Prentice Hall Press, 2010.

[4] C. J. Watkins and P. Dayan, “Q-learning,” Machine learning, vol. 8, no. 3-4, pp. 279–292, 1992.

[5] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” Journal of artificial intelligence research, vol. 4, pp. 237–285, 1996.

[6] M. Dorigo and L. M. Gambardella, “Ant colony system: A cooperative learning approach to the traveling salesman problem,” IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, pp. 53–66, 1997.

[7] Z. Michalewicz and D. B. Fogel, How to solve it: Modern heuristics. Springer Science & Business Media, 2004.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, ..., and S. Petersen, “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–533, 2015