**Master of AI**

Faculty of Engineering & IT

42172 AT1 Mini AI Application Example

UTS CRICOS 00099F

**Link to solution notebook:**

https://github.com/Freddiefine777/optimization-problem.git

**Part:**

**1**

**Author**:

Enda fan

25608579

**Date:**

2024.09.13

**Semester:**

1

**Year:**

2024

**University of Technology Sydney**

Table of Contents

[Table of Contents 2](#_Toc177118542)

[Summary or Executive Summary 3](#_Toc177118543)

[Introduction 4](#_Toc177118544)

[Report Body 5](#_Toc177118545)

[Methods/AI techniques used 5](#_Toc177118546)

[Implementation of AI techniques 6](#_Toc177118547)

[Comparisons of different AI techniques 10](#_Toc177118548)

[Conclusions and Recommendations 12](#_Toc177118549)

[List of References 14](#_Toc177118550)

[Individual reflection 15](#_Toc177118551)

## Summary or Executive Summary

The Travelling Salesman Problem (TSP) was the first very challenging problem I encountered in my AI course, and it was the first time I used AI algorithms to solve a real-world problem. When I was studying the course Data Structures in my undergraduate studies, I also learned at that time about the methods to find the shortest paths in graphs, that is, the Depth-First Search Algorithm (DFS), the Breadth-First Algorithm (BFS), and the Dijkstra Algorithm, which I commonly use, so it was very natural for me to think of using these three algorithms at that time when I saw the problem. Still, after I had gone through my in-class and post-course study, I found that these two problems are very different.

On the one hand, finding the shortest path in the graph, that is, using the three algorithms I learned in my undergraduate study to solve this problem, only need to find the shortest path from each point to all other points. At the same time, the travelling salesman problem also needs to find the shortest path through all the closed-loop paths, so the previous method is not feasible. On the other hand, the travelling salesman problem is a classic optimization problem, in my understanding, the optimisation problem is the process of optimising a function, for example, we use a recogniser to identify traffic lights: when it sees a red light we want it to output a very high (close to one hundred) score, and when it sees a green light we want it to output a very low (close to zero) score, and the difference in the scores between the red light and the green light is our function, and we want this function to be as high as it can be because this means that we will get better recognition results and that we will not be able to get better recognition results. We want this function to be as high as possible, because this means that our recognition results will be good, and the process of making the value of this function higher and higher is the process of optimisation.

So, after I studied the lesson of week 3, I learned about the two more popular ways to solve this problem. It just so happens that my hometown: Chengdu, Sichuan Province, China, is a city with a long history, a very beautiful city, and one of my favourite cities, and I'd like to use the solution of the Travelling Salesman Problem to design a route for the most convenient excursion to many of the famous attractions in my hometown.

The next two algorithms that I used to solve the travelling salesman problem for this assignment are described, they are all commonly used optimisation and search algorithms.

In the end, I compare the performance and applicability scenarios of the hill-climbing algorithm and the simulated annealing algorithm, explain the advantages and disadvantages of these two algorithms in different contexts, and provide reasonable suggestions based on experimental findings.

## Introduction

地图

描述已自动生成My hometown Chengdu is a very famous tourist city in China that attracts millions of tourists every year. The problem of this assignment was to plan the most time-efficient route to visit Chengdu, which included 20 attractions and locations that I highly recommend.

*Figure 1 Chengdu China*

表格

描述已自动生成I used Google Maps to get the coordinates for each of these locations but most of them are in the centre of Chengdu, the difference between their latitude and longitude is not particularly clear, so I only chose the last three places of their longitude, last two places of their latitude to work with. After doing these steps, I used KNIME to draw a graph of these locations.

*Figure 2 LocationX and LocationY of Chengdu*

图表, 散点图

描述已自动生成

*Figure 3 Graph of Locations*

I used the hill-climbing algorithm and the simulated annealing algorithm to solve this problem. Firstly, I set the Chengdu Museum as a starting point, and randomised the order of cities to form an initial path, in these two algorithms, they optimise the path by constantly modifying the initial solution when they meet two situations: 1. the path improves less than some predetermined value. 2. after a certain number of iterations, the path no longer improves. And I used Matplotlib to draw a picture to perspective their optimisation processes.

## Report Body

### Methods/AI techniques used

The Travelling Salesman Problem (TSP) is a travelling problem in which each city on a map must be visited. The goal is to find a tour with a cost less than C (or in the optimised version, a tour with the lowest cost). Great efforts have been made to improve the capabilities of the TSP algorithms. These algorithms can also be extended to handle collections of vehicles. For example, the search and optimisation algorithm for school bus routes in Boston saved $5 million, reduced traffic and air pollution, and saved time for drivers and students. In addition to planning journeys, search algorithms have been used for tasks such as planning the movement of automated circuit board drilling machines and stocking machines on the shop floor (Russell, Stuart, & Peter Norvig, 2021). Optimization algorithms are a class of mathematical or computational methods used to find an optimal solution to a problem by minimising or maximising an objective function under given conditions, i.e., to find the method that minimises the cost.

To solve this problem, I chose to use the two algorithms: The Hill-climbing algorithm and the Simulated annealing algorithm, which I learned in class.

Hill-climbing algorithm: In my opinion, this optimisation algorithm is like a blind man climbing a mountain, he can't see the whole picture of the mountain, not to mention that he doesn't know where the top of the mountain is, and he can only use his crutches to explore up the mountain step by step and continue exploring after he finds a maximum slope to go up one step, so that he can reach the top of the mountain, and then, our optimisation is also finished. The problem with this algorithm is that when there are several peaks, we don't know which peak we are walking on, and we don't know if we have reached the highest one. However, because this algorithm is very simple and easy to implement, it is still the most widely used optimisation method in AI.

Simulated annealing algorithm: This algorithm is based on the phenomenon of annealing in physics, in simple terms, when we encounter a patch of mountains, using a mountain climbing algorithm is not guaranteed to reach the highest peaks because it goes a locally optimal solution, which does not want to continue exploring, whereas using a simulated annealing algorithm allows us to jump out of this solution, permitting us to find a better or worse solution, and over time, this acceptance of other solutions probability gradually decreases, and the particle is more stable as the temperature decreases during simulated annealing. However, this algorithm is more difficult to implement and requires a higher choice of parameters for the cooling process.

### Implementation of AI techniques

文本

描述已自动生成We store the LocationsX and LocationsY of these twenty locations in a dictionary called Location, which makes it easier for us to make subsequent calls.

*Figure 4 LocationsX and LocationsY of 20 locations*

Before starting the execution of the algorithm, create a function called define\_map whose main task is to generate a list of cities and a matrix of distances between them. it is necessary to generate a list containing the names of all the cities (all\_cities) and a dictionary of the distances between the cities (distances) and to calculate and store the Euclidean distance between each pair of cities. Finally, the distance between each pair of cities is output. Euclidean distances are commonly used to calculate straight-line distances between two points. Among other things, we used the NumPy library because I wanted to use the formula to calculate the Euclidean distance between two cities, which is their straight-line distance. The formula is:

After this, print out the distance between each city.

文本

描述已自动生成*Figure 5 Calculate the Euclidean distance between each city*

We are designing a function called generate\_k\_neighbouring\_paths, the core idea of which is to generate multiple neighbouring paths through a local inverse order operation as a way of exploring local variations in the solution. With the idea of generating multiple neighbouring paths, these neighbouring solutions can be used in optimization algorithms to find potentially better ones.

Suppose path= [A, B, C, D, E] and k=2.

In the first iteration: randomly generate left = 1, right = 3, after that the paths [B, C, D] are reversed to get the new path [A, D, C, B, E]. Add to neighbouring\_paths.

In the second iteration: reinitialise neighbouring\_path = [A, B, C, D, E]. Assume again that left = 2 and right = 4 this time, and that the paths [C, D, E] are reversed to get the new route [A, B, E, D, C]. Add the route to the neighbouring\_paths list.

So, the neighbouring\_paths returned may be [[A, D, C, B, E], [A, B, E, D, C]].

*Figure 6* 文本

描述已自动生成*Generate K neighbouring paths*

We design a function called cost\_fun to calculate and save the distance between two cities, cost\_fun is used to evaluate the merit of the current path when we use simulated annealing and hill-climbing algorithms, the algorithms call cost\_fun to calculate the cost of the current solution and the neighbouring solutions and to decide whether to accept a new solution or not.

*Figure* 图形用户界面, 文本, 应用程序

描述已自动生成*7 Cost\_fun function*

We design two functions: 1. shuffled: randomly shuffles the order of the elements of a list. But random.shuffle can only operate on lists, so first convert the iterable to a list, disrupt and return. This allows the solution to be chosen randomly with the same cost. 2. argmin\_random\_tie: calculates the cost of each path using cost\_fun and returns the path with the smallest cost.

文本

描述已自动生成*Figure 8 shuffled function and argmin\_random\_tie function*

#### 文本 描述已自动生成Hill Climbing Algorithm

*Figure 9 Hill Climbing Algorithm*

1. city\_list: The input list of cities, representing the initial solution of the algorithm, i.e. the starting path.

2. neighbour\_function: A function to generate all neighbouring paths to the current path. These neighbouring paths are usually generated by small local changes (e.g. swapping the positions of two cities).

3. cost\_fun: A function that calculates the cost (e.g., total distance) of a given path. The goal of the algorithm is to minimise this cost.

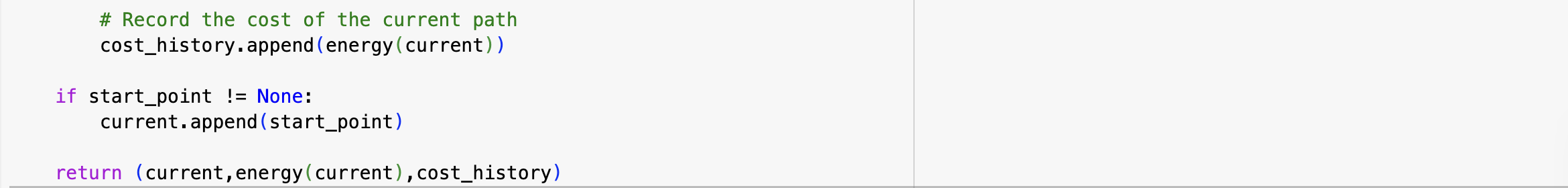
4. times: The maximum number of consecutive non-improvements allowed by the algorithm before termination. If no better solution is found in TIME's iterations, the algorithm will stop to avoid an endless search.

5. iteration: The maximum number of iterations, which limits the total running time of the algorithm.

6. start\_point: Optional parameter specifying the starting point. If given, this point will be added to the end of the final path at the end of the algorithm to form a closed loop.

In the while loop, we first find the proximity solution of the current path and find the shortest one among them, if we find a better solution, we replace the current path with this better path and reset the count, if we don't find a better solution, we increase the count, when several consecutive iterations don't find a better solution, the algorithm considers that a locally optimal solution has been found, and exits the loop.

#### Simulated Annealing Algorithm

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

*Figure 10 Simulated Annealing Algorithm*

1. init: the initial travelling path, which is the order in which a city is arranged

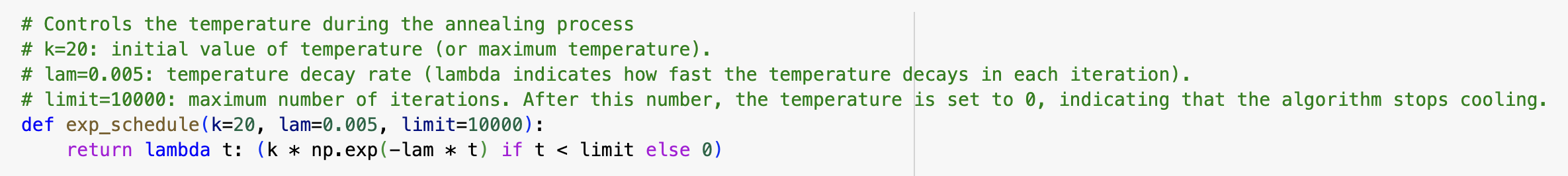
2. energy: used to evaluate the advantages and disadvantages of each solution

3. neighbor: generates neighbours of the current solution, that is, a new path obtained by randomly exchanging the positions of two cities in the current path.

4. schedule: used to control the temperature decay in the simulated annealing algorithm 5. start\_point is the optional starting city (to ensure that the path is a closed loop)

In the while loop, the probability of accepting a suboptimal solution is mainly reduced by gradually decreasing the temperature. At higher temperatures, the algorithm becomes more active, and there is a certain probability of accepting a suboptimal solution, thus jumping out of the local optimum. When the temperature is reduced to near zero, the algorithm prefers to select the neighbouring optimal solution to the current solution and eventually converges to the global or local optimal solution.

In this algorithm, I would like to specifically introduce the exponential decay formula:

The k is the initial value of the temperature, and the current temperature can be calculated by the number of iterations t. Based on this formula, we can infer that 1. the higher the initial temperature is, the more information the algorithm can explore. 2. the larger , the faster the temperature decays; the smaller , the slower the temperature decays.

*Figure 11 Function exp\_schedule*

In the simulated annealing algorithm, the temperature controls the acceptance probability of the solution. When the temperature is high, the algorithm prefers to accept suboptimal solutions (even if they are worse) to jump out of the local optimum, whereas when the temperature decreases, the algorithm gradually accepts only the better solutions, allowing the solution to reach the optimum.

### 图表 描述已自动生成Comparisons of different AI techniques

*Figure 12 Cost over iterations in Hill Climbing Algorithms*

The vertical axis represents the cost of the path, with higher cost indicating worse paths and lower cost indicating better paths. The horizontal axis represents the number of iterations. As we can see from the figure, the path cost starts at about 2900, decreases as the algorithm iterates, and finally be stable at around 2300. The number of iterations starts at 0 and increases as the algorithm tries to optimize the path and reduce the cost. Finally, at about 140 iterations, the algorithm stops updating, indicating that it has found a local optimal solution.

In the first few iterations, the cost drops sharply, indicating that a better path was found quickly in the early stages. Starting from the 20th iteration, the cost no longer drops as quickly as before, indicating that it is becoming increasingly difficult to improve the path and may have approached the local optimal solution. Starting from about the 60th iteration, the cost becomes relatively stable and almost no longer decreases, indicating that the algorithm has entered the local optimum and cannot find a better solution through the current operation, indicating that the algorithm may have been trapped in the local optimum and cannot be further optimized.

This is the disadvantage of the mountain climbing algorithm because the mountain climbing algorithm only tries to select the optimal solution within the current neighbourhood, so it is easy to fall into the local optimum, and not necessarily find the global optimal solution, just like the blind man I mentioned climbing the mountain this process, it does not know it to the top of the mountain, so there is no guarantee that it is the highest of this one.

图表

低可信度描述已自动生成But the advantages are also very obvious, first of all, it is very easy to implement compared to the simulated annealing algorithm, and secondly, it only through the iteration 140 times, the cost reaches about 2300, and in our later comparison of the simulated annealing algorithm, we will find that, the difference between the two is not particularly large, and the final results are close to each other.

*Figure 12 Cost over iterations in Simulated Annealing*

The vertical axis represents the cost of the path. The lower the cost, the better the path is and the algorithm is constantly optimising the path, and the horizontal axis represents the number of iterations of the algorithm. In the simulated annealing algorithm, as the number of iterations increases, the algorithm keeps optimising the path. The figure shows that the initial cost is around 2900, after about several iterations, the cost decreases rapidly and finally stabilise at around 2200, the number of iterations of the simulated annealing algorithm ranges from 0 to 10000.

The cost decreases rapidly at the beginning of the graph. This shows that the simulated annealing algorithm explores the path more randomly in the initial stage and finds a better solution quickly. After the cost drops to around 2200, it starts to stabilise. This indicates that the algorithm has found a better path and is no longer accepting higher cost solutions.

In the first few iterations, the algorithm jumps out of the local optimum by accepting worse solutions and finding a better solution quickly, which could explain why the cost drops so quickly. However, as the temperature decreases, the algorithm becomes less exploratory prefers more stable paths and avoids choosing suboptimal solutions, so we can see a very long flat line in the later iterations, which leads to a stabilisation of the cost.

The advantages and disadvantages of simulated annealing can be clearly found firstly, it requires more iterations and more computational time to reach the final solution. Secondly, a successful simulated annealing is very dependent on the design parameters such as initial temperature and decay rate. However, it performs better in global search, especially when the solution space is complex and there are multiple locally optimal solutions, which also solves the problem of a blind man climbing a mountain without being able to climb to the top of the tallest one as if an eye had been added to this blind man.

I don't think there's a problem with choosing either algorithm because our data isn't that overwhelming, and it's probably not very obvious in terms of time spent, but I'd prefer to go with the hill-climbing algorithm because I think it's very easy to implement and I'm not particularly stringent in terms of how much I get to spend at the end of the day, and the hill-climbing algorithm is going to be much shorter as well.

## Conclusions and Recommendations

As we can see from our assignments, the hill climbing algorithm is more efficient in small-scale problems and can quickly find a better solution for scenarios that require a fast response. The simulated annealing algorithm, on the other hand, performs better in optimisation problems with a complex solution space because it can jump out of the local optimum and can get closer to the global optimum, although it converges more slowly.

In choosing between the two, I believe that the hill climbing algorithm is suitable for applications in scenarios where the solution space is small or where local optimality does not affect the results. It can be used in preliminary optimisation or as part of a heuristic algorithm to quickly find a better solution. If the performance of the hill-climbing algorithm needs to be improved, multiple initial solutions can be randomly generated to avoid falling into a local optimum. Whereas in complex problems the simulated annealing algorithm can find a solution close to the global optimum by exploring more of the solution space, the parameter settings for simulated annealing can be further adjusted to the specific problem in terms of the temperature decay rate and the initial temperature k.

In practical applications, it is recommended to choose the appropriate algorithm based on the complexity of the problem and the need for a globally optimal solution.

## List of References

Russell, Stuart, and Peter Norvig. (2021). Artificial Intelligence: a Modern Approach, Global Edition, Pearson Education Ltd.

## Individual reflection

This assignment is still difficult for me because this is the first time I have used AI algorithms to solve a real-world problem. I chose this problem because I want to design this route for my hometown Chengdu, and I also hope more people can know about my hometown. Based on this idea and the example provided, I chose to use the hill climbing algorithm and simulated annealing algorithm learnt in lab 3 to solve this problem.

I think there are three things I need to improve in my later assignments: firstly, I think I need to do my homework faster in the future, I did it too slowly this time and probably did it on and off for almost two weeks, my efficiency must be improved a little bit. The second thing is to read the requirements of the assignment before you start to write it, especially according to the scoring criteria, otherwise, you have to rewrite it like I did this time, and the last thing is not to pursue high-difficulty tasks, because I was ready to implement three algorithms together with lab 3 at the beginning, but then I found that the gene algorithm never ran out, and I never changed it correctly, and that wasted a lot of my time.