An optimal solution for the Travelling Salesman Problem

Abstract

In the following study we attempted to derive an optimal solution for the Travelling Salesman Problem, also known as TSP, by implementing two distinct method: the Hungarian Method, and by Simulated Annealing. During the process, we tried to improve our methods to get better results and optimal routes in various ways. We found that the Simulated Annealing gives more optimal routes for all cases than the Hungarian Method, moreover, we concluded that the latter does not work for too big or complex systems.

1 Introduction

One of the most challenging optimisation problems in mathematics is the so called Travelling Salesman Problem (TSP), which was first formulated and examined in the 1930s by Karl Menger in Vienna and Harvard. In the TSP, one seeks for an optimal route between $n > 0$ cities visiting each city only once such that given the cost of the route(in our case, it is the distance) between the ith and the jth city ($i,j \leq n$ and $i \neq j$), the total cost of the journey is minimised. The TSP is an NP hard problem, and no solution has been found for it yet, however throughout the last century several algorithms have been invented to find an optimal solution for such problem, some better than others. [1] Two of these are the Hungarian Method (*algorithm A*) and the Simulated Annealing (*algorithm B*), which we discuss below.

2 Hungarian Method

2.1 Structure

The Hungarian Method is an efficient way to solve assignment problems, and the TSP can be viewed as a zeros-assignment problem. To begin with, we are going to form a cost-matrix based on the distances between cities, the rows represent the origins and the columns stand for the destination of the travels between cities. By definition, the salesman cannot visit a city twice or more, thus we set the diagonal of our matrix to *infinity* to prevent looping.***(1)*** First of all, for each row/column we perform row and column reduction by selecting the minimum value of them and subtracting it from all entries in the row/column. By doing so, we derive our *reduced-matrix* with at least one zero entry in each row and column. Now our aim is to cover the zero entry cells with the least lines possible, given all lines must be horizontal or vertical. To generalise this, one must do row and column scanning, as follows: ***(2)*** first look for rows with single zero entries, **mark** the cell with the single zero and cross out the corresponding column. After scanning each row, do the same for each column, crossing out the rows of the cell with the single zero. Note that by crossing out rows/columns, we are going to ignore these rows/columns for subsequent scans. By considering only the remaining entries of the matrix we derive our *remaining-matrix*. If there is none or more than one zero entry in a row/column we skip the given row/column. Repeat scanning on the *remaining-matrix* until we have no zeros left in the *remaining-matrix*.***(3)*** Note that if we end up covering every single row and column as a result of the procedure, we can simply focus on covering either every row or every column, but we shall not cover all of the rows and columns.***(4)*** Once we have no zeros left in the *remaining-matrix*, check whether or not the number of crossed out rows and columns is equal to the number of cities. If so, then we found **an optimal solution**. On the other hand, if not, then select the minimum value of the *remaining-matrix*: *remaining-minimum*, and subtract it from all entries in the *remaining-matrix*. Now bring back the crossed out rows and columns, and add the *remaining-minimum* to the entries which had been crossed out twice, that is: one vertical and one horizontal line. As a result, we derived our **new** *reduced-matrix*, and with this, we start again from ***(2)***, until we find an optimal solution.***(5)*** Once an optimal solution has been found, that is, the number of crossed out rows and columns is equal to the number of cities, one can compose a complete optimal route through all cities as follows. Starting from the top left **marked** cell, which must contain zero, we attempt to create a route through all cities: the marked cell *(m,n)* tells us that the salesman goes from the mth city to the nth city, thus n will become our new m, and in the row indexed by n we find the next marked cell, which tells us where to go next. One follows this procedure until a complete optimal route is constructed. [2]

2.2 Fine-tuning

During the implementation of this method we have introduced several improvements to tackle arising problems and to improve performance such as:

- If rows and columns in *remaining-matrix* contain only two or no zeros in them, one might get stuck on row and column scanning, generating an infinite loop of scanning. In such case, mark a cell with zero entry which has no other marked cells in the corresponding row or column, and cross out its row, thus we have created columns with single zeros, and we can proceed forward with the scanning.
- During the process of finding an optimal route in ***(5)*** the distribution of marked cells might be in such a way that the salesman gets into a loop of some cities $m < n$. In this situation, we take cells with the next smallest element of the latest *reduced-matrix* into consideration, and attempt to build a route using the marked cells and these new cells. We repeat this procedure until a complete optimal route is found.
- We used online Hungarian Method calculator tools and actual paper-based calculations to set examples of matrices for TSP and work through them. By comparing these outcomes with ours, we managed to make some changes to our program in order to get the accurate results.

2.3 Limitations

The biggest challenge with this implementation was to handle complex matrices. For almost each tour file we had different issues arising during the execution of the method. We managed to solve most of these, and thus for simple matrices with less cities, such as 12, 17, 21, 26, 42, 48, the method appeared to work, nevertheless for more complex systems our experiments raised errors and even got into infinite loops. The reason for this could be due to faulty or poor coding, since there were several issues and special cases which could not always be fully handled. Another reason for the lack of an optimal tour for large amount of cities could be that the Hungarian Method simply might not be generalisable for any number of cities. As an example, during experimentations with 58 cities or 180 cities one can find that the program ran out of iterations during a *while loop* looking for the optimal tour by taking the next smallest element of the final *remaining-matrix* into consideration.

3 Simulated Annealing

3.1 Structure

Simulated Annealing is a probabilistic algorithm for finding a good global maximum or minimum for an optimisation problem. This is a popular and efficient approach usually used to find a good optimal route for the TSP, since its strength is that it can escape from local maxima/minima, while many other algorithms might get trapped at these critical points. [3]

The technique is that starting with an initial tour, we swap cities randomly, and analyse the resulting tour. If it is better than the previous one, then we change our current tour for this new one, nonetheless predefined if it is not better, it still can become the new tour, based on some predefined heat function, which decreases by time, thus the probability of setting a worse tour to the new tour decreases as time passes. ***(1)*** To successfully implement this, firstly we create a cost-matrix of the system such as for the Hungarian Method. ***(2)*** Next, we define a random initial tour to start with and calculate the the corresponding cost(total distance travelled). ***(3)***. Then we randomly choose 2 cities, reverse the order of the cities between them, and calculate the new cost of this new tour. If the new cost is less than the previous one, then this new tour becomes the current tour, and we do step ***(3)*** again. The key point of this method is that the number of iterations of step ***(3)*** and the probability of taking a worse tour over a better one is based on some heat function and on the difference between the two tours, hence it allows the system to get out of local minima. As step ***(3)*** is repeated, the likelihood of changing the better tour for a worse one decreases, furthermore the recurrence of ***(3)*** stops when the temperature drops below a predefined *stopping-temperature*, T_s. In this study, we use the following heat function:

$$T_n = \frac{T_{n-1}}{\beta}, \tag{1}$$

where $n \geq 1$ and $n \in \mathbb{N}$, moreover T_n is the current temperature, T_{n-1} is the previous one, and β is some predefined constant which is used to decrease the temperature by a small amount for every iteration. The probability of taking a worse tour over a better one is

$$P = \frac{1}{e^{|\triangle E|/T_n}}, \tag{2}$$

where $\triangle E$ is the difference between the cost of the tours we are comparing: one before, and one after reversing the route between two randomly selected cities. Early on, when the temperature is high, the probability of taking a worse tour over a better one is high, thus it prevents getting caught at local minima. This probability however decreases exponentially, as the temperature decreases. The process detailed above runs until T_n reaches T_s, then halts, and outputs the final tour found. Thus we derived an optimal tour for the TSP using Simulated Annealing.

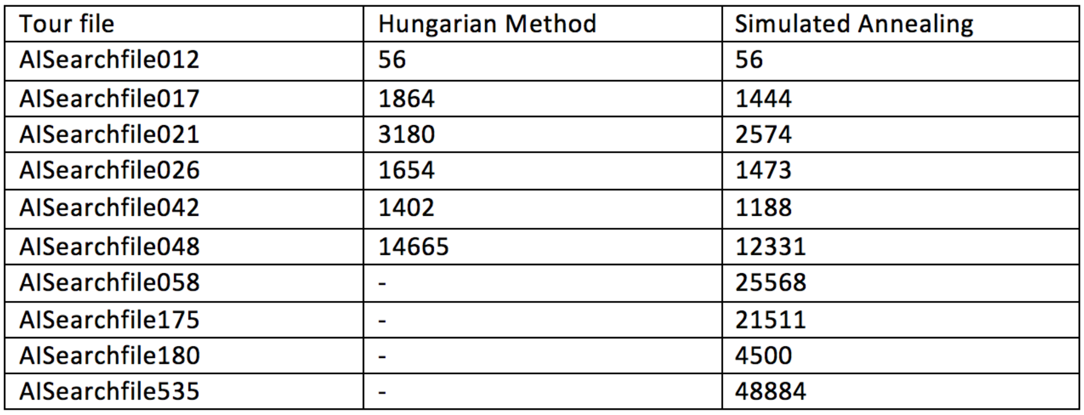
3.2 Fine-tuning

To improve the efficiency and accuracy of the implementation, we have done the following:

- We set the initial parameters carefully such that the iterations run for a sufficient amount of times, P in (2) decreases slow enough: $T_0 = 10000$, $\beta = 1.00001$, and $T_s = 0.00001$. Having investigated results with various parameters several times, we have found that, statistically, for smaller T_0, and greater β and T_s the method produces worse tours.
- To further improve efficiency we applied a basic greedy algorithm and we set the resulting tour as the initial tour for the annealing. By doing this, we already start with a somehow optimal tour, thus a better tour is more likely to be found compared to the case when we start with a random initial tour.
- Finally, we put the whole implementation in a loop of 100 iterations and chose the best out of the produced tours, for each tour file, to derive more accurate tours.

During implementation and experimentation of this method we did not come across any serious issue or special case.

4 Results and conclusion



Tour file	Hungarian Method	Simulated Annealing
AISearchfile012	56	56
AISearchfile017	1864	1444
AISearchfile021	3180	2574
AISearchfile026	1654	1473
AISearchfile042	1402	1188
AISearchfile048	14665	12331
AISearchfile058	-	25568
AISearchfile175	-	21511
AISearchfile180	-	4500
AISearchfile535	-	48884

Figure 1: Table of lengths of optimal tours for the discussed methods.

On the table in Figure 1 one can clearly see that the tour-length outcomes of the Simulated Annealing were better in multiple ways; the program successfully ran for all tour files, no matter how complex the cost-matrix was. Furthermore it also achieved better results than the Hungarian Method, apart from the case of *AISearchfile012*, where both techniques showed the same result. As we go along into more complex systems, we see worst results as we go through the implementation of the Hungarian Method, plus in some extreme cases we do not even get any results. In addition, the Simulated Annealing algorithm could be further improved by introducing a logarithmic heat function instead.

In conclusion, using the Simulated Annealing is a relatively good and efficient approximation when one is looking for the best possible tour available in the TSP. On the other hand, the Hungarian Method gives poor results for optimal tours, but this could be improved by further development of the code.

5 References

References

[1] http://www.math.uwaterloo.ca/tsp/history/index.html

[2] http://www.universalteacherpublications.com/univ/ebooks/or/ch6/travsales.htm

[3] http://katrinaeg.com/simulated-annealing.html