Monte Carlo Optimisation: The traveling salesman problem

Mikhail Tiuterev, Philipp von Campenhausen

March 24, 2024

1 Introduction

The traveling salesman problem is a classic computational problem, where a salesman has to visit a number of cities in a closed loop, but the order is not predetermined. The problem then lies in the question of the optimal order of visits with respect to some metric, for example distance traveled or time spent.
The problem, when abstracted to "find the best order", has wide-ranging applications, relevant for navigation or even chip design. The high relevance for large industries makes the economic incentive for solving the problem large, leading to significant amounts of research done in this field.

2 Description of the problem

2.1 General problem

While the namesake of the problem is rather specific, it can be greatly abstracted to a problem of graph theory. In this representation, the vertices of the graph would be the cities, and the (weighted) edges the possible connections between the cities, where the weight associated with that edge is the cost of that road in the original problem.

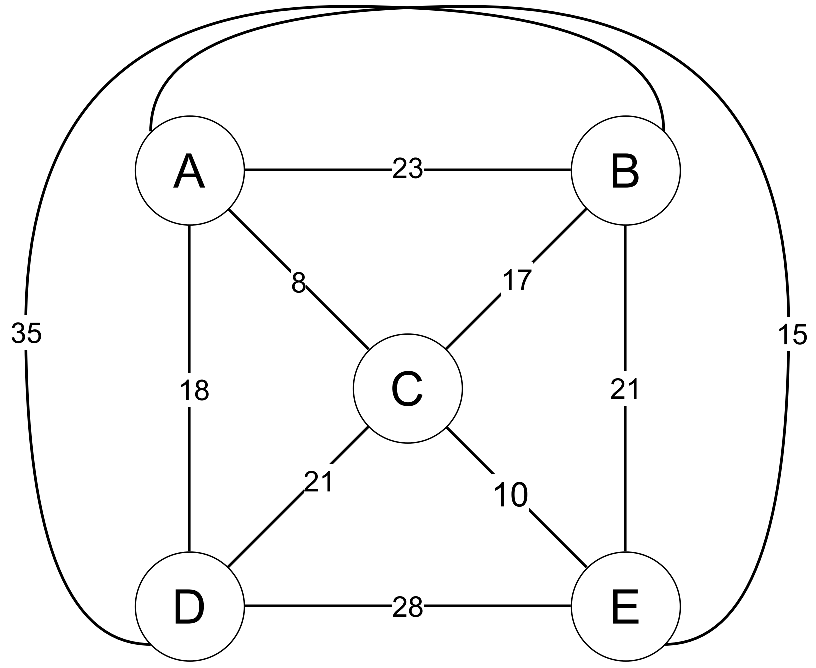


Figure 1: Example graph for a traveling salesman problem. The cities are the vertices and the numbers on the edges the associated weights/cost

An example of such a graph can be seen in Figure 1. There, the "cities" A through E are interconnected with "roads", each with an associated cost. The shown case is part of the most general form, where every vertex is connected to every other vertex. So for this, the problem would be formulated as: "what is the order in which to visit all nodes A, B, C, D and E such that the accumulated total weight, i.e. the sum of the weights of visited edges, is minimal". For such a small problem with only 5 vertices, the solution can be found sufficiently quickly, by generating every possible permutation and checking the total costs. For the shown example, this simple brute force method yields the order $A -> C -> B -> E -> D -> A$ with the total cost of 92. This is not the only solution, as every cyclic permutation yields the same cost, as well as traveling the same route in reverse.

2.2 Our specific problem

The previous example of a simple weighted graph is the most general variant of this problem. To show the use of Monte Carlo optimization, we will discuss a more involved and specific variant.
The setting of the problem is a description of an actual traveling salesman, with certain parameters, that will impact the optimal solution. We will later vary these parameters to see how they influence the solution.
The problem is described with the following parameters:

- The problem takes place on a collection of cities.
  - Cities are characterized by an `X`- and `Y`-coordinate. Therefore, the first measure of cost of travel between two cities is the Euclidean distance.
  - Each city has a "package weight" associated with it. The salesman is selling his product in every city, and the weight of the product he sells differs for each city. The weight he has to carry increases the cost of travel, favoring cities with a larger package weight to come first in the route, as there he could quickly dump a large amount of the total weight.
- A salesman intends to travel through all these cities in a closed loop. With this salesman, there are multiple parameters associated, which all impact the optimal solution:
  - He has a base speed, which dictates how quickly he travels between cities
  - There is a fuel consumption per kilometer and a price of that fuel
  - He is paid an hourly wage
  - He has two slowdown factors:
    1. While starting with the total weight of all packages, he slowly loses that weight. That speeds up his travels. To accommodate this, there is a penalty term on the travel time that scales with the fraction of the total weight that he is currently carrying
    2. There are times with more and times with less traffic. For this, there is a simple slowdown factor which takes the current time into account
  - He has a flat waiting time, a time he loses in each city.

In the end this means the base travel time τ_{base} between two cities, with coordinates $(X_1,\ Y_1)$ and $(X_2,\ Y_2)$ is calculated as

$$\tau_{\text{base}} = \frac{\sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}}{v_{\text{base}}} \tag{1}$$

where v_{base} is the before mentioned base speed. To this speed, there is then a total slowdown factor applied, which is the sum of the weight-caused slowdown factor (η_{weight}) and the time-based slowdown factor (η_{time}). These factors are explicitly calculated as:

$$\begin{aligned}\eta_{\text{weight}} &= \frac{m_{\text{current}}}{m_{\text{maximum}}} \cdot \lambda_{\text{weight}} \\ \eta_{\text{time}} &= \left(1 \cdot \mathcal{N}(\mu_1,\, \sigma_1^2) + 2 \cdot \mathcal{N}(\mu_2,\, \sigma_2^2)\right) \cdot \lambda_{\text{time}}\end{aligned}$$

Here m_{current} is the weight still left, m_{maximum} is the total weight of all packages, i.e. the maximum weight the salesman carries and λ_{weight} a parameter to control the influence of this slowdown.
The dependence of η_{time} on the current time is shown in Figure 2. $\mathcal{N}$ denotes a Gaussian function, with the mean μ_X and the variance σ_X^2. For the means, we chose $\mu_1 = 8$ and $\mu_2 = 17$ to simulate rush hours in the morning and the evening. λ_{time} is again used to tune the impact of this factor.

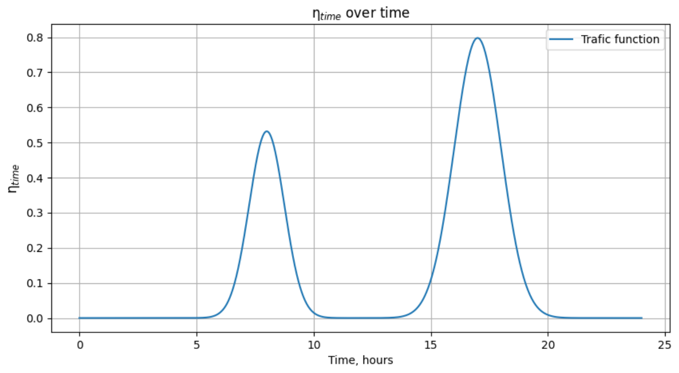


Figure 2: The dependence of the parameter η_{time} over time. The maximum values correspond to the appearance of traffic and the period of maximum traffic congestion

With these two factors, the total slowdown coefficient is calculated:

$$\eta_{\text{total}} = 1 + \eta_{\text{time}} + \eta_{\text{weight}} \tag{2}$$

With (1) and (2) we can then finally calculate the “slowed travel time” τ_{slowed} and this combined with the aforementioned flat waiting time τ_{wait} gives the total travel time τ_{total} for any two cities, time and current weight:

$$\begin{aligned}\tau_{\text{slowed}} &= \tau_{\text{base}} \cdot \eta_{\text{total}} \\ \tau_{\text{total}} &= \tau_{\text{slowed}} + \tau_{\text{wait}}\end{aligned} \tag{3}$$

For the comparison of different routes, we will optimize the monetary cost. This, in the end, is the sum of fuel cost and wage of the salesman. For the fuel cost (C_{fuel}), we used the traveled distance and fuel consumption, which starts with a base fuel consumption ($c_{\text{fuel,base}}$) and scales with the current weight and the cost of fuel per liter $\lambda_{\text{fuelcost}}$:

$$C_{\text{fuel}} = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2} \cdot \underbrace{c_{\text{fuel,base}} \cdot (1 + 0.001 \cdot m_{\text{current}})}_{\text{total fuel consumption}} \cdot \lambda_{\text{fuelcost}} \tag{4}$$

For the total wage of the salesman C_{wage}, we simply use an hourly wage λ_{wage} as a parameter to scale the total time:

$$C_{\text{wage}} = \tau_{\text{total}} \cdot \lambda_{\text{wage}} \tag{5}$$

The total cost C_{total}, which is then the variable used to characterize the optimal route, is then simply calculated as the sum of (4) and (5):

$$C_{\text{total}} = C_{\text{fuel}} + C_{\text{wage}} \tag{6}$$

For a whole route the total costs of each path between two cities will be summed.
With these parameters and this total cost function, we are then equipped to find the optimal route for different configurations.

2.2.1 Initial state of our problem

To be able to compare different configurations we need a static problem, and so we once generate a list of 30 cities, with their respective coordinates and the weights of the package that needs to be dropped there. We then treat this as the problem we want to solve. A plot of the different cities with an initial guess for the route is shown in Figure 3.

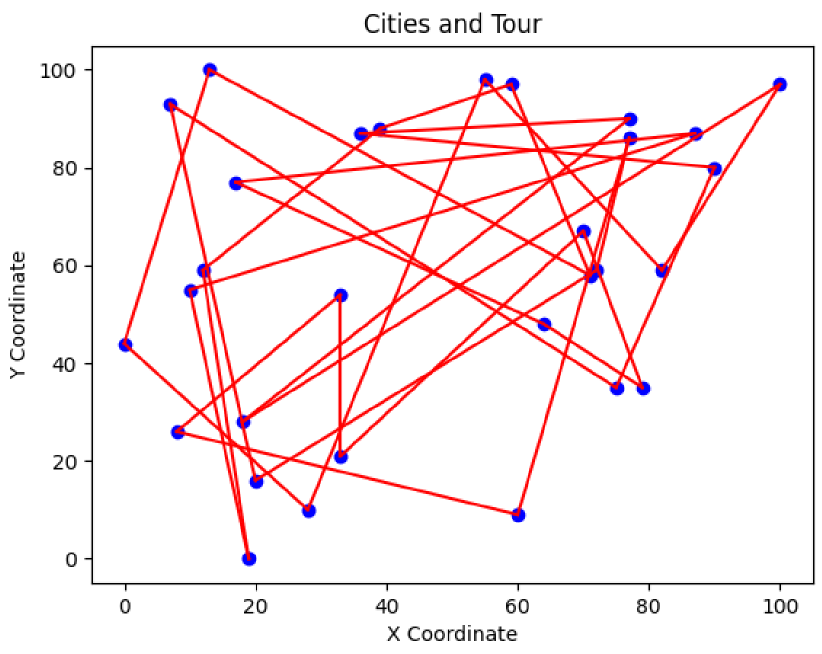


Figure 3: Locations of cities used in our analysis with an initially guessed route. The blue dots represent the cities, and the red line connecting them all is the closed loop that is the chosen route.

The chosen route is one of the $(30-1)!/2 \approx 4.42 \times 10^{30}$ possible routes (where the same route in the reverse order, and the same route with different starting points are treated as the same route). This shows how the problem quickly becomes non-trivial to solve for larger numbers of cities.

3 Route optimization

3.1 Overview

There is a first significant consideration to be made when choosing a solving algorithm. The traveling salesman problem only describes the problem itself, but depending on the use case, you can either be interested in finding a sufficiently good answer quickly, or finding the definitively best answer in the shortest possible time.
With the different areas of interest for the problem, and the large number of different use cases, there is a wide range of different approaches. These range from classical computers to quantum computation, and from math/graph-theory based approaches to statistical simulations.
As the title suggests, this project will focus on Monte Carlo simulations. For this, we will mainly follow *Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm, V. Černý, 1985[2]* and *Optimization of the time-dependent traveling salesman problem with Monte-Carlo methods, Bentner et al., 2001[1].*

3.2 Monte Carlo Markov Chain Optimizations

In the project, we have utilized two different Monte Carlo Markov Chain approaches: Random Walk and Simulated Annealing algorithms. Both of them are related to the Metropolis Hastings algorithm class, where algorithms are based on the accept/reject method but with the different acceptance probability. Both of these and their application in the project will now be laid out

3.2.1 Random Walk

The Random Walk Metropolis-Hastings (RWMH) algorithm is a Markov Chain Monte Carlo (MCMC) method commonly used for sampling from complex probability distributions. It is an extension of the Metropolis-Hastings algorithm that proposes new states by adding a random noise to the current state.

In the RWMH algorithm, at each iteration, a candidate state x' is proposed by swapping 2 cities in the optimal salesman's cities visiting order x. Then, an acceptance probability $\alpha(x, x')$ is calculated based on the ratio of the probability density function (pdf) of the proposed state to the pdf of the current state. If the acceptance probability is greater than a uniform random number u between 0 and 1, the candidate state is accepted, and the Markov chain moves to that state. Otherwise, the current state remains unchanged.

The RWMH algorithm proceeds as follows:

Algorithm 1 Random Walk Metropolis-Hastings Algorithm

```
1: Create an initial proposal for the optimal route
2: Calculate the total cost of the current optimal route ($x$)
3: for $i = 1$ to $N$ do
4:     Sample two distinct city indices city_1 and city_2 uniformly at random from the range of city indices
5:     Create a new route by swapping the cities at indices city_1 and city_2 in the current optimal route
6:     Calculate the total cost of the new route ($x'$)
7:     Calculate the acceptance probability:
           $\alpha(x, x') = \min(1, exp(x - x'))$
8:     Generate a uniform random number $u \sim \text{Uniform}(0, 1)$
9:     if $u \leq \alpha(x, x')$ then
10:        Accept the candidate state: $x \leftarrow x'$
11:    end if
12: end for
13: return Samples from the target distribution
```

3.2.2 Simulated Annealing

The Simulated Annealing Metropolis-Hastings (SAMH) algorithm combines the concepts of simulated annealing with the Metropolis-Hastings algorithm. It is in some ways a complicated random walk algorithm. In the SAMH algorithm, at each iteration, a candidate state x' is proposed from a proposal distribution. The acceptance probability $\alpha(x, x')$ is calculated based on the energy difference between the current state x and the proposed state x', normalized by the current temperature T_i according to the Metropolis-Hastings rule. The temperature T_i is lowered logarithmically by $T_{i+1} = \beta \cdot T_i$ with a cooling factor α (usually $0.8 \leq \beta \leq 0.999$) until the system freezes at a very low temperature.

The SAMH algorithm proceeds as follows:

Algorithm 2 Simulated Annealing Metropolis-Hastings Algorithm

```
1: Same as for the Random Walk
2: Initialize the Temperature ($T_1$)
3: for $i = 1$ to $N$ do
4:     Same as for the Random Walk
5:     Calculate the acceptance probability:
           $\alpha(x, x') = \min\left(1, \exp\left(\frac{x - x'}{T_i}\right)\right)$
6:     Generate a uniform random number $u \sim \text{Uniform}(0, 1)$
7:     if $u \leq \alpha(x, x')$ then
8:         Accept the candidate state: $x \leftarrow x'$
9:     end if
10:    Update the temperature according to the cooling schedule: $T_{i+1} \leftarrow \beta \cdot T_i$
11: end for
12: return Samples from the target distribution
```

3.3 Mean Value Graphs

To analyze the performance of the algorithms and to investigate the impact of the different parameters on the solution, we will plot the total cost of the current solution against the current temperature in the algorithm. This serves as a proxy for the performance of the algorithm as it progresses.
Since the search for the minimum of the cost is a random process, we consider the time (or temperature) dependence of not one specific solution, but the average value. A representation of this method is shown in Figure 4.

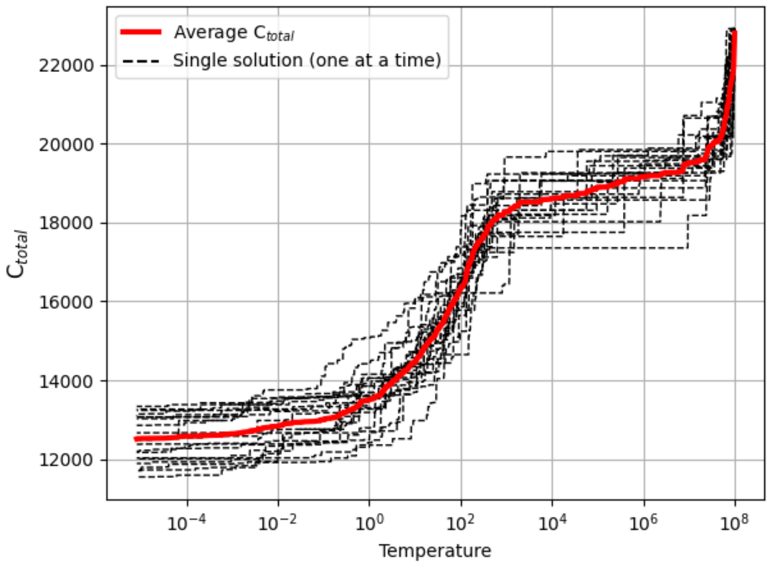


Figure 4: C_{total} over time. The 20 black dashed lines are the singular evaluated solutions. The solid red line shows the average over all these different runs.

In the shown figure, the average graph we obtained (shown in red) has smooth curves, which compared to the 20 jagged singular runs (shown as multiple black dashed lines) is better suited for analysis of the results. In addition, this averaging reduces the impact of degenerate cases. For example, the possibility of falling into the global minimum early on or never falling into it. For all results shown in this report, we will average over 20 runs, unless stated otherwise.
The shape of the graph is of importance, as we can see that the algorithm does not constantly improve the cost, but seems to stay on a plateau at high temperatures, and only as the temperature decreases beyond a certain point the cost decreases as well.
This behavior can be explained by analyzing the algorithm (see 2). The temperature plays a vital role in determining the acceptance probability, which is written as:

$$\alpha(x, x') = \min\left(1, \exp\left(\frac{x - x'}{T_i}\right)\right)$$

When the temperature T_i is very high compared to $x - x'$ the fraction becomes $T_i \gg x - x' \implies \frac{x-x'}{T_i} \approx 0$. As $\exp(0) = 1$ the acceptance probability becomes 1, irregardless of the specific values for x' and x. This means that for high temperatures the algorithm does not favor a decrease in total cost and therefore is in a regime of random walk. Only when the temperature is low enough, the different values for x' and x get resolved and the algorithm starts to improve the result.

4 Computational Results

4.1 Comparison of different slowdown factors

We started by analyzing the impact of different factors for λ_{time}, i.e. the scaling of the time-caused slowdown. The results for $\lambda_{\text{time}} = 1.01,\ 2,\ 5$ are shown in Figure 5.

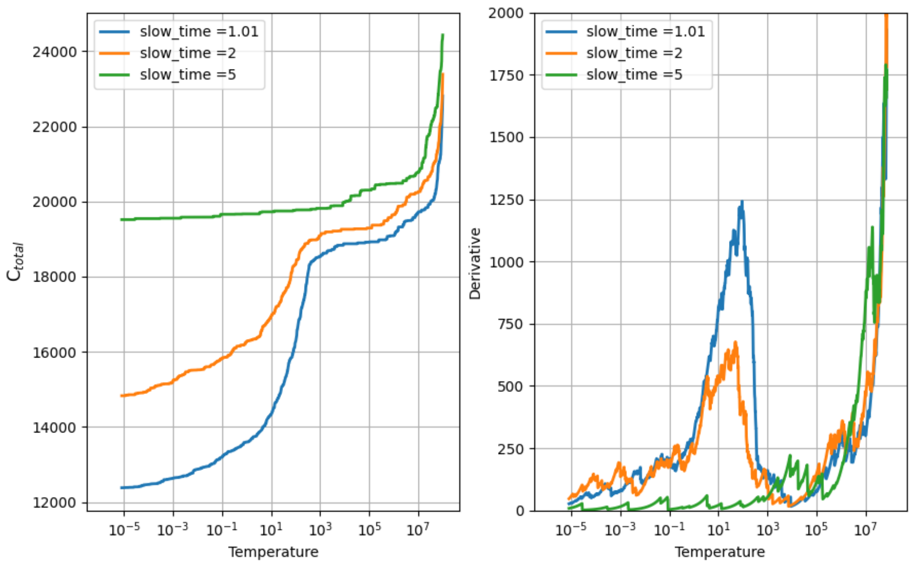


Figure 5: The dependence of the result on the variation of the indicator of highway congestion. The left figure shows the progress of the algorithm for different values of λ_{time} and the right figure shows the absolute change in cost for the solutions.

The lines of different colors show the progress of the algorithm for the different parameters. One would assume that a larger slowdown factor would result in a higher total cost. This is confirmed in the results, where the curve corresponding to the highest factor remains at larger total costs for all temperatures. Another interesting observation can be made in the temperature dependence. The curves for $\lambda_{\text{time}} = 1.01, 2$ show a pronounced dip in C_{total} at $T \approx 10^2 - 10^3$, as it was explained before. This dip is not present for $\lambda_{\text{time}} = 5$.
From this, we can deduce that for a high enough λ_{time}, the solution rapidly becomes optimal. This also can be interpreted as that when λ_{total} is high enough, the relative difference between two solutions appears to shrink, making progress in C_{total} slower as the algorithm progresses. We can also seem that the shape of the curve appears to shift to lower temperatures for higher λ_{time}. This is due to the same effect as before, where the different solutions do not differ from one another as much for larger λ_{time}. When the difference between different C_{total} gets smaller, the temperature also has to become smaller to resolve that difference. This means that the large dip also occurs at lower energies for higher λ_{time}.

4.2 Comparison of different starting times

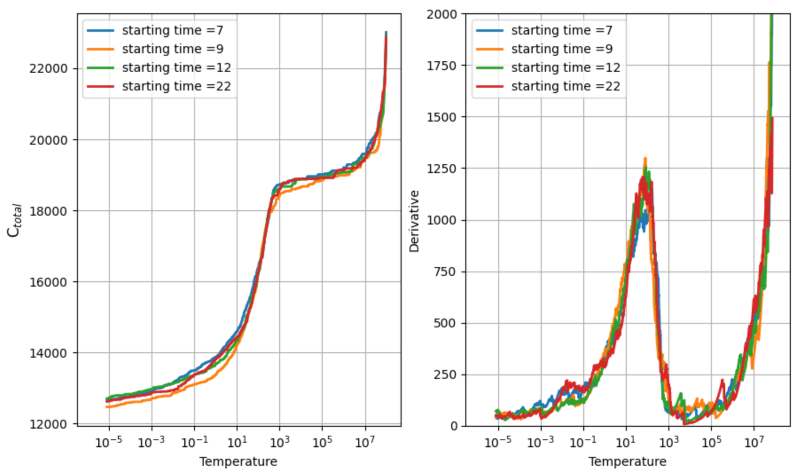
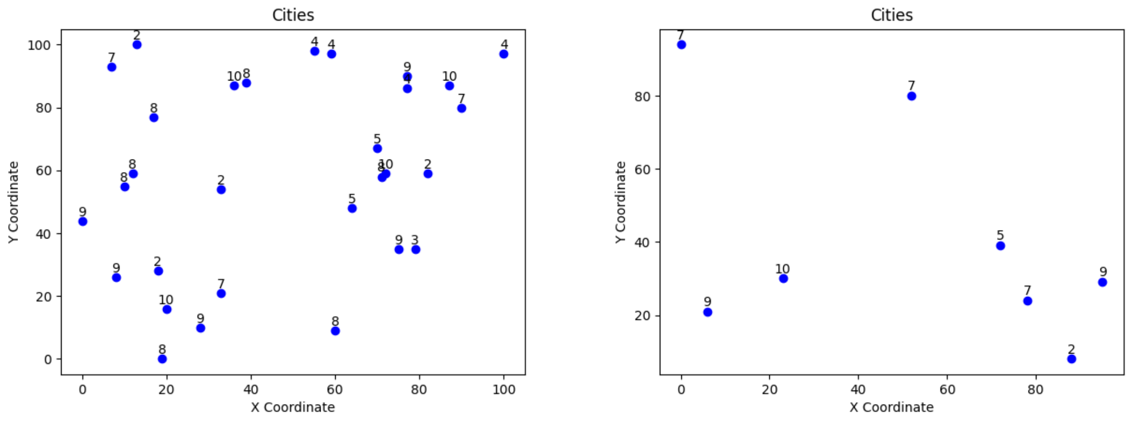


Figure 6: The dependence of the salesman's starting time. As before, the different colored lines represent the progress of the algorithm for different values of a parameter, in this case the time at which the salesman starts his travel.

Figure 6 shows the impact of the starting time on the finding of the solution. One would expect different starting times to yield different results, as the time of the day at which the salesman starts would impact if, when and where the salesman would encounter traffic jams. As the graphs show, the progress appears to be independent of the starting time, contrary to our initial expectation. We expect the cause to be that the chosen configuration of cities simply leads to the total travel time being over 24 h, meaning that irregardless of the starting time, the salesman will always encounter both traffic jams.
To test this, we created a second set of cities, consisting of 8 cities. The estimated time to complete this set is approximately 5 hours.



(a) Bigger landscape with 30 cities

(b) Smaller landscape with 8 cities

Figure 7: Comparison of bigger landscape and smaller landscape with fewer cities. The blue dots represent the cities and the numbers above the dots the package weights associated with each city.

Figure 7 shows a comparison of the two used landscapes. 7a is a diagram of the big landscape consisting of 30 cities. This set of cities was used for most of the analysis. In comparison to that, 7b shows the smaller landscape with only 8 cities.
We can then run the algorithm on this smaller landscape, and now we determine the minimal total cost found for

all the starting times, instead of the decrease in total cost over the decrease in temperature. The resulting diagram is shown in Figure 8.

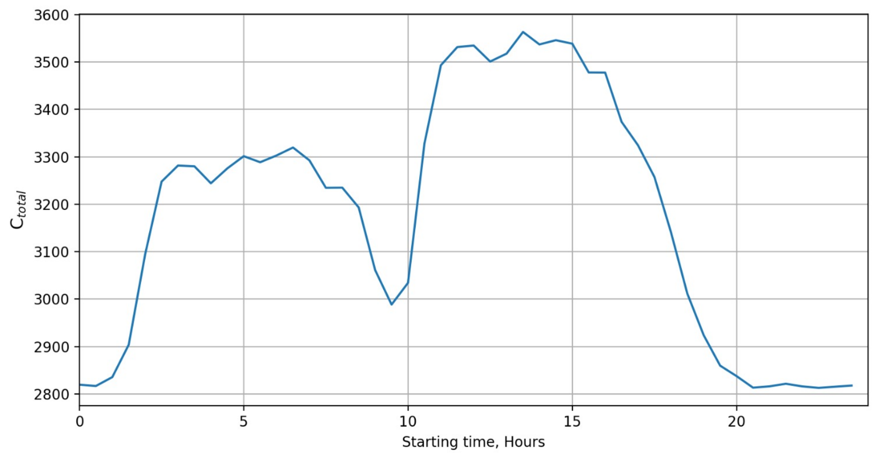


Figure 8: The dependence of the algorithm's final result over the starting time on the reduced number of cities.

The figure shows a base level of $C_{\text{total}} \approx 2800$ for starting times from $\approx$ 8 pm to $\approx$ 1 am. This appears to be the lowest possible total cost for the configuration of cities, without any traffic jams.
The graph then shows two large peaks, one in the morning and one in the afternoon. These then correspond to the two traffic jams we have implemented into our problem. Interestingly, the total cost seems to stay rather constant in these peaks. This can be understood, as the total cost being mostly impacted by whether *if* the salesman encounters a traffic jam, and not *when*. To derive this, we can analyze the plateaus and see that they have a width of $\approx 5\,\text{h}$. This exactly corresponds to the approximate total travel time and then means that as long as a traffic jam falls into the working hours of the salesman, the total cost will be significantly increased.
It is also of note that the second peak, corresponding to the traffic jam around 5 pm, is higher, i.e. has a higher total cost associated with it. This can be understood in combination with the impact of the traffic jams, as it is shown in Figure 2, where the second jam simply was set to be stronger.
This analysis can then be used to send the salesman on his journey at the optimal time, in this case either after 8 pm or, if normal working hours are preferred, just before 10 am.

5 Conclusion

In our paper, we studied the time-dependent traveling salesman problem as defined by us, by introducing a salesman whose speed is depended on the amount of product weight and the traffic jams that occur twice during the day.
We were able to study the effects of different parameters in the problem on the final solution, as well as on the algorithm's performance.
The results confirmed our expectations, e.g. that stronger traffic jams increase the total cost, or that certain starting times can be used to dodge occurring traffic jams.
The method of plotting current total cost against temperature during the simulation provided insights into the performance of the simulated annealing algorithm. It showed the importance of the temperature, how it allows a random walk in the beginning, and how it resolves the differences between different routes.
The project showed how a Monte Carlo simulation of the traveling salesman problem can be used to find "good" solutions in short timespans, as the complexity of the problem quickly grows with its size.
The analysis of the performance of different starting times also showed how these methods can be used to obtain answers to non-trivial questions, with real-world benefits.

References

[1] Johannes Bentner et al. "Optimization of the time-dependent traveling salesman problem with Monte Carlo methods". In: Physical Review E 64.3 (2001), p. 036701.

[2] Vladimír Černỳ. "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm". In: Journal of optimization theory and applications 45 (1985), pp. 41–51.