Voyageur de Commerce

Physique Numérique et Projet
Sorbonne Université
S1 2021-22

Sebastian Castedo, Hannah Vormann

Contents

1 Introduction

The Travelling Salesman Problem (TSP) is one of the best known optimisation problems in computer science today and has many well known applications in a range of fields. The problem can be stated as follows:

Given a list of cities with distances between them, a travelling salesman wants to find the shortest path that visits each city exactly once and returns to the first city.

This problem was first formulated mathematically by Hamilton in the 19th century [1] and with the advent of computers and computational complexity theory in the 20th century the problem began to be investigated computationally. Initially, brute force techniques were used but people very quickly realised that as the number of cities increases, it becomes virtually impossible to get exact solutions as the number of possible paths grows rapidly (see Table 1). The next era of this problem saw a variety of approximate solutions developed with many techniques using ideas from statistical mechanics as we will explore. The TSP is an NP-Complete problem meaning that the problem cannot be decided by a deterministic Turing Machine in polynomial time. The worst case running time for an algorithm that solved this problem increases super-polynomially with the number of nodes. This was proved by Karp in 1972 [2].

Cities	**Paths**
3	1
5	12
7	360
9	20 160
15	43 Bn
20	$6 \cdot 10^{19}$
71	$6 \cdot 10^{102}$

Table 1: Number of possible paths for maps with a certain number of cities to be visited. This illustrates the combinatorial explosion of the number of possible paths as the number of cities increases.

In its simplest form the TSP has a number of applications for logistics companies trying to find the shortest way to deliver goods to a list of places where the cities represent customers. The manufacture of microchips requires holes to be drilled in the printed circuit board (PCB) in order to attach components to it. The path of the drill bit can be optimised with TSP where the cities represent the locations where holes need to be drilled. The concept of distance can also be changed to accommodate other applications such as DNA sequencing where distance is represented by similarity measure between DNA fragments.[9]

This report is organized as follows: In section 2 we will give an overview of the theory behind the algorithm of Monte-Carlo Metropolis used in our approach to the TSP. Our results for three different example maps are then shown and discussed in section 3. The report closes with a conclusion in section 4. Our *Fortran 90* code is provided in the Annex.

2 Theory

2.1 Monte Carlo Metropolis-Hastings Algorithm

The goal of this problem is to try and find the global minimum of the total distance function. As stated earlier, finding this global minimum is very difficult for a large number of cities so we will use approximate techniques in order to try and arrive at a minimum that is close to the global minimum.
For our TSP algorithm we used the *Metropolis-Hastings algorithm* which is a 1st order *Markov Chain Monte-Carlo* method.
The outline of the algorithm goes as follows:

1. Randomise the initial order of positions to visit.
2. Select a random position and swap it with another random position.
3. Evaluate the change in total distance δD.
4. If the distance decreased: Keep the changed positions.
5. If the distance increased, apply the criterion of Metropolis:
   (a) Keep the changed positions with probability
   $$p = e^{\frac{-\delta D}{k_B T}} \tag{1}$$
   where k_B is the Boltzmann constant and T is a "fictitious temperature"
   (b) Or reject the changed positions with probability 1-p.
6. Repeat from 2.

The random order of the points in the used map (Step 1.) was generated in the following way: The two-dimensional coordinates were read from a text file in a certain order. Then, two random numbers between 1 and the number of points in the map were drawn indicating the indices of two points to be exchanged in the sequence. This process was repeated five times the number of points in the map to ensure a sufficient randomization.

The power of this method lies in the fact that it allows us to obtain non-Gaussian probability distributions of the final path configuration around the initial path [4]. Normally, when looking at a simple *random walk* (random changes of position with a uniform probability distribution), the distribution of the final position/path will always go towards a Gaussian distribution centered around the initial position/path configuration due to the *central limit theorem*. In contrast, the criterion of Metropolis allows us to obtain any probability distribution depending on which explicit expression is chosen for p. This allows to systematically go towards an "optimized" path (which here means a path with a lower total path length) instead of guessing completely randomly.

In general, the "landscape" of the total path length as a function of the order in which the points are visited can be very complicated and have several local minima. Therefore, the solution can be sensitive to the random initial order and it is crucial to run the algorithm several times with different initial conditions in order to find a "good" minimum.

2.2 Simulated Annealing

The "fictitious temperature" in equation 1 is a technique carried over from Statistical Physics but in this context there is no real "temperature" to speak of. Instead, it is used here as a measure of the likelihood that a new path with a longer total distance than the one from the previous iteration is accepted. Step 5 a. is crucial in escaping a local minimum as it allows you to temporarily select a "bad" step to find a better minimum.
A way to improve the above algorithm is by incorporating *Simulated Annealing* instead of holding

the temperature/probability p constant over the course of the optimization. Annealing originally refers to a technique used by blacksmiths whereby in order to create stronger materials they cool them down slowly. In a similar way, in simulated annealing the aim is to slowly decrease the temperature over iterations so that a global minimum is found. We decided to study five different cooling schedules according to which the temperature T_k at the k'th iteration is defined as follows:

- Constant Temperature:
$$T_k = T_0 \ \forall \ k$$
where T_0 is the initial temperature.
- Linear Temperature:
$$T_k = T_0 - k\frac{T_0 - 1}{N}$$
where N is the total number of iterations
- Exponential Temperature:
$$T_k = T_0 \cdot 0.8^{k/\alpha}$$
where α is a constant factor. This is also known as *Exponential multiplicative cooling* and was first proposed by Kirkpatrick, Gelatt and Vecchi in 1983 [5].
- Sigmoid Temperature:
$$T_k = \frac{T_0}{0.5 + e^{(k-\alpha)/\beta}}$$
where α and β are constants.
- Stepwise Temperature:
Alternating phases of constant temperature linked by linear increase or decrease of temperature

Keep in mind that the temperature here is a fictitious quantity. A "high"/"low" temperature is a temperature for which the probability p of accepting a change of positions, which does not decrease the total path length, is high/low. As can be seen from eq. 1 this depends on the typical scale of δD and might be different for different examples.

2.3 Maps

In this project we investigated three different two-dimensional maps of differing sizes.
First, we investigated forty equally spaced points on a unit circle generated as follows:

$$x_i = cos\left(i \cdot \frac{360}{40}\right) \tag{2}$$
$$y_i = sin\left(i \cdot \frac{360}{40}\right) \qquad \text{for } i \in [1, 40] \ . \tag{3}$$

The x and y coordinates here have not further specified dimensions of distance. This provides a simple uniform shape to test our algorithm on with a well-known and intuitive global optimum: going from one point to the next following the shape of the circle in one sense (see fig. 1). The global minimum can be calculated according to this and is 6.276727658.
Next we used two maps which came from an online database and which usefully included some best known solutions to test our algorithm against. The first is a map of the capitals of the 48 US States (excluding Alaska and Hawaii) [6] with a best known solution of 10628 km [8] and the second is a bigger map of 127 beer gardens in the German city of Augsburg [6] with a best known solution of 118282 m [8]).

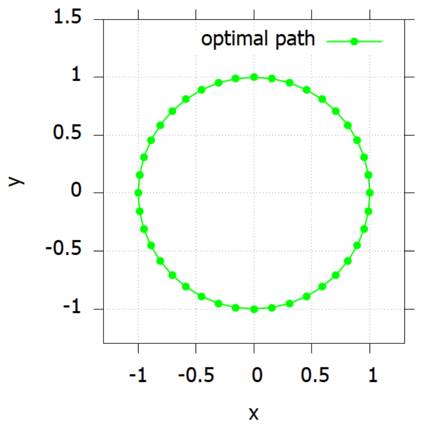


Figure 1: Forty points on a unit circle. The shortest possible path and therefore the global optimum is to go around the circle in one sense (green line).

3 Results

In this project we investigated different annealing temperatures and schedules for points on a unit circle, 48 US Capitols and 127 Beer Gardens in a German city. We also investigated the effect that the number of iterations had on our minimum distance in the beer garden example. The results for the different maps are presented in this section.

3.1 Circle

First, we investigated the forty points on a unit circle as it is a simple and intuitive example to set up and test our algorithm. The following temperature schedules were applied in order to probe their effect on the final result:

- constant temperature
  - T_0=300 K
  - T_0=100 K
  - T_0=10 K
  - T_0=2 K
  - T_0=0 K
- linear cooling with T_0= 100 K
- stepwise cooling: 300 K $\rightarrow$ 2 K $\rightarrow$ 50 K $\rightarrow$ 10 K $\rightarrow$ 1 K
- exponential cooling with T_0= 100 K and $\alpha = 1$

For each of them the algorithm was run three times in order to explore different random initial paths. This number was kept low because the problem is relatively simple and thus we decided to rather focus on exploring a variety of different temperature schedules here. For the next, more complicated map of the US capitols, however, this number was increased (see section 3.2).

In fig. 2 the path length over the course of the optimization is shown for the run which provided the lowest final path length for each of the different temperature schedules.

First, one can clearly see that for "high" constant temperatures like 300 K (red curve) and 100 K (dark green curve) the total distance fluctuates very strongly around the initial value without ever going towards a minimum. This is because the probability for accepting a "bad" change of the

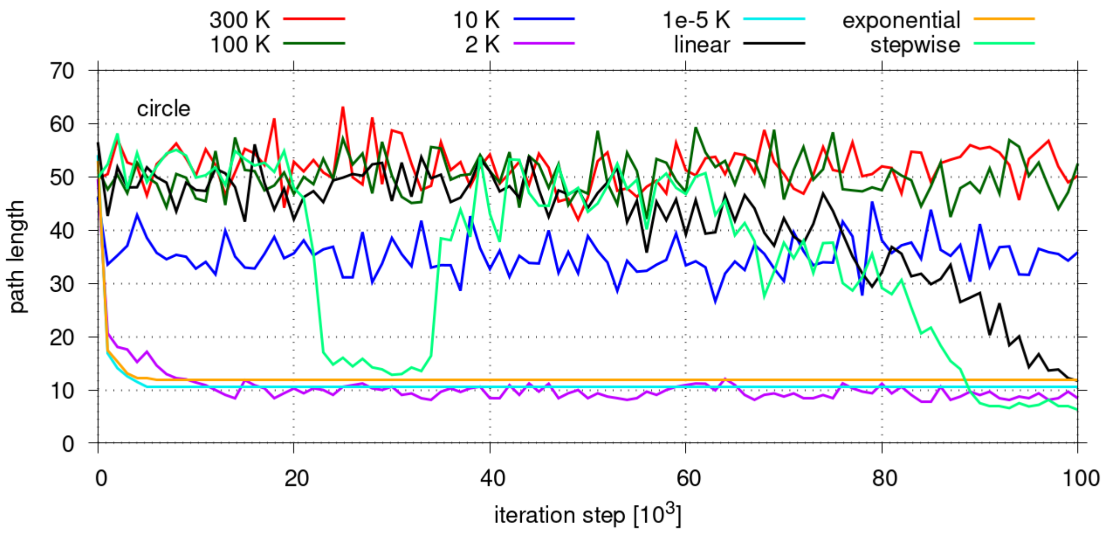
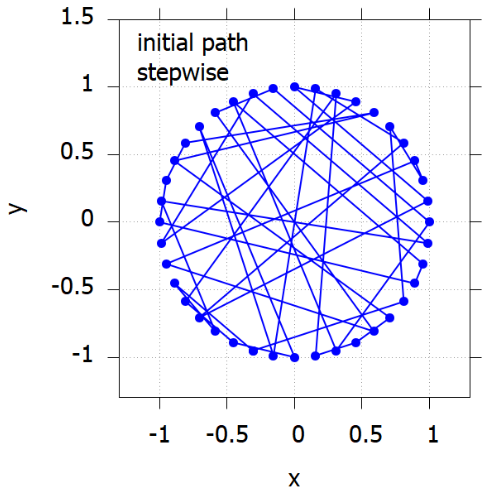
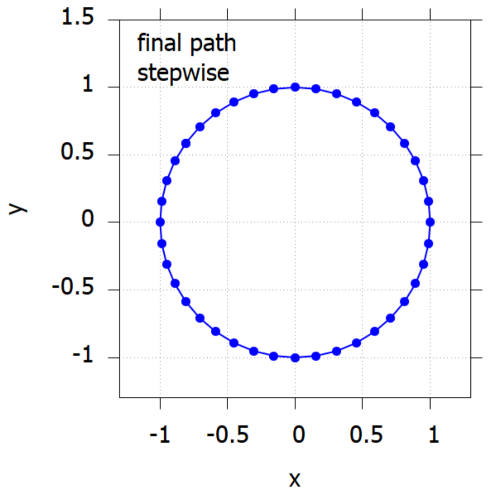


Figure 2: Total path length as a function of the iteration step of the run with the smallest final path length for the different temperature schedules for the circle.

path remains constantly high. So every time the total path length decreases a little, we again step out of this minimum only a few iterations later and never manage to closely approach a minimum. The same is true for an intermediate constant temperature of 10 K but the effect is a little less pronounced so that in the end we get a slightly better result. However, the final path length is still approximately 30 units higher than the global minimum of 6.276727658.
In contrast, with very low constant temperatures we get reasonable results. For 2 K (pink curve) and 10^{-5} K (light blue curve) the path length decreases drastically within the first few iterations because we have a very low probability of accepting changes to the path that would increase its length. This is also why once we attained a relatively small value it does not significantly change anymore for the rest of the optimization. However, one can observe an interesting small difference between the graphs for 2 K and 10^{-5} K: After the initial decrease, the light blue curve for 10^{-5} K does not change any more because none of the random position changes further decreases the path length and p is vanishingly small so that in practice no further changes are being accepted. On the other hand, p is slightly higher for 2 K so that a few temporarily "bad" changes are accepted which allow to get over some small "hills" in the path length landscape and explore some slightly lower minima. This is why in the end, after 10^5 iterations, the result for 2 K is slightly better than for 10^{-5} K. Surprisingly the best 2 K result is also the second best result overall making this constant temperature schedule better suited here than the linear and exponential simulated annealing.
The path length for the best run with linear cooling (black curve) in the beginning behaves similarly to the ones with a constant high temperature since we also start with a relatively high temperature of 100 K and the temperature decreases slower than for the exponential or sigmoid cooling scheme. However, during the last 20 000 steps the path length decreases systematically while still showing significant fluctuations. After 10^5 iterations the final result is of the same order of magnitude as for the other cooling schedules and the constant low temperatures. Contrary to the other temperature schedules, the path length here significantly decreases until the end of the optimization. Therefore increasing the total number of iterations might improve the results for linear cooling.
Additionally, from the orange curve we can clearly see the effect of an exponential decrease in temperature: During the first few iterations, the path length falls off rapidly because we apparently found some very efficient changes. But already after a few iterations (approximately 5), the path length does not change anymore because the temperature is already so low that no more "bad" changes get accepted but on the other side we also do not find any changes that further decrease the path length. It is the same situation as for the lowest constant temperature (light blue curve).
Finally; we get our overall best result with a stepwise cooling. One can clearly see the stepwise



(a) initial path



(b) final path

Figure 3: Random initial path (a) and optimized final path (b) to visit all 40 points on the unit circle obtained using stepwise cooling. The final path is indeed the global minimum for this map.

behavior of p in the light green curve: First, after switching to a low temperature after the first ninth of the total iterations, we find a minimum of the path length. Then again we go back to a higher temperature which results in the acceptance of some "bad" changes so that the path length again increases strongly. However, when during the last four ninth iterations the temperature stepwise goes down again and we accept less and less "bad" changes, we converge towards an even lower minimum than before. This means that the intermediate, temporary increase in temperature allowed to step out of a local minimum and finally converge towards a better minimum. In this case, we were even able to reproduce the global minimum with the stepwise cooling. The best result obtained with this method was indeed 6.2767276582275944 which is equal to the analytic global minimum for our unit circle within numerical inaccuracy. The initial and final path of this best run are visualized in fig. 3a and fig. 3b, respectively. Comparing fig. 3b to fig. 1 shows that the final path here is indeed the same one as the analytical/logically inferred best solution.

3.2 US Capitals

For the map of the 48 Contiguous US states' capitals (excluding Juneau, Alaska and Honolulu, Hawaii) six different temperature schemes were used: First, the temperature was held constant at a "high" temperature (1000 K), then at a "low" temperature (2 K). Subsequently different cooling schemes were applied: linear cooling with $T_0 = 1000\ K$, exponential cooling with $T_0 = 3000\ K$ and $\alpha = 5 \cdot 10^4$, as well as sigmoid cooling with $\alpha = 4 \cdot 10^4$ and $\beta = 10^4$. Lastly, a stepwise cooling scheme with the steps 3000 K $\rightarrow$ 1000 K $\rightarrow$ 50 K $\rightarrow$ 400 K $\rightarrow$ 2 K was applied. All nine intervals (alternating between constant temperature and linear increase/decrease) were equally sized. The temperature as a function of the iteration step k for the different cooling schemes is shown in fig. 4. For simplicity, from here on the Boltzmann constant was fixed to $k_B = 1$ as it only rescales the temperature here. Since the temperature is fictitious its absolute scale is irrelevant.
The number of iterations was fixed to $N = 10^5$.
For every cooling strategy five runs with different random initial paths were performed.

In table 2 the results are summarized. As a comparison, the optimal solution for this particular problem is known and given with a path length of 10628 km.[8]
By far the worst results are obtained with a constant high temperature of T=1000 K.
A lower constant temperature of T=2 K with an average acceptance rate approximately ten times lower already gives much better results, although they are still worse than the ones of the more sophisticated cooling schedules. This is also due to the low probability p to accept changes in the

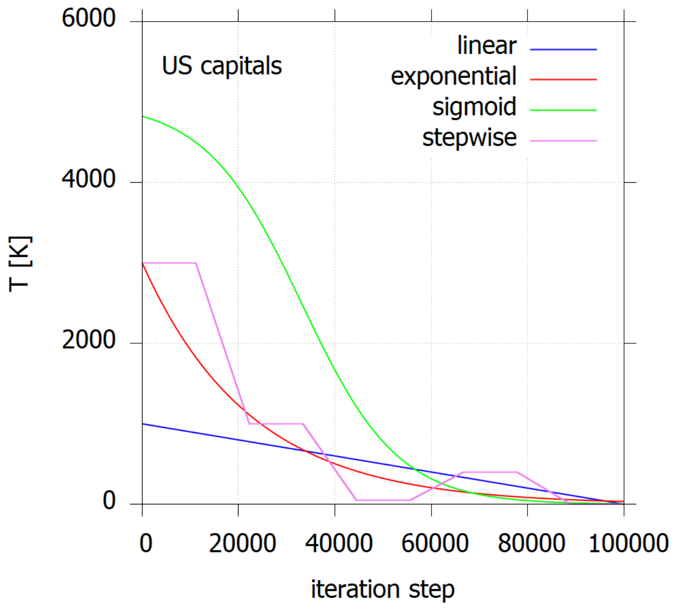


Figure 4: Cooling schemes for the map of the capitals of the 48 US states.

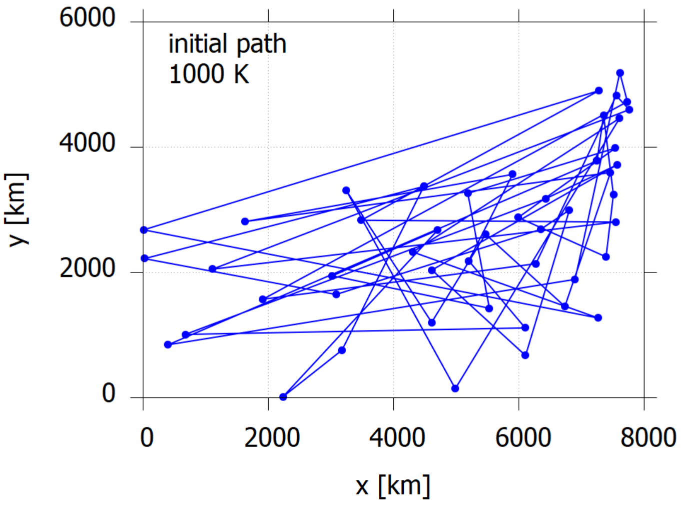
order of the visited cities that (for the present moment) lead to an increase in the path length. This makes it hard to escape a local minimum (also see explanations to fig. 11 below).
Including a cooling scheme generally improves the found solutions:
The best results (lowest average final path length, highest average decrease in path length, and lowest optimal path length out of five runs with different initial configurations) are here obtained with a linear decrease in temperature over the course of the simulation although it is the most "simple" one.
The next best solutions are provided by a stepwise, then exponential, and finally sigmoid cooling. However, exponential cooling gives the highest average δD out of the three. This shows that five runs might be too few to obtain good statistics; the initial conditions for the other strategies might just have "by accident" been better starting configurations.
Nevertheless, in general all the applied annealing strategies provide similar results, especially if one takes into account the standard deviations around the given averaged values of the five runs with different initial paths.
The highest acceptance rate is yield by sigmoid cooling (circa 30%). This shows impressively that solutions obtained using a cooling scheme are superior to ones using a constant temperature: In general, a high acceptance probability leads to many changes to the path that do not lead to a decrease but an increase in path length. However, the changes accepted here with the sigmoid cooling scheme seem to be significantly better than for a constant T=1000 K where still a lot but already 10% less changes are accepted overall but the results are much worse.
All in all, we also have to conclude that even though the path length could be decreased significantly compared to a random initial path, we could not get very close to the optimal solution with the applied methods: Our best solution of a path length of 34909 km with linear cooling is still approximately 3.3 times higher than the global minimum of 10628 km.[8]

In figures 5 to 10 the initial and final path of the run that provided the lowest final path length d_{opt} for every temperature scheme are shown. The coordinates are given in km and the absolute values are chosen in a way that the northwesternmost capital (Olympia, Washington) has a x-coordinate of 0 km and the southwesternmost capital (Phoenix, Arizona) a y-coordinate of 0 km. However, they are not of any physical importance here as we are only interested in distances.

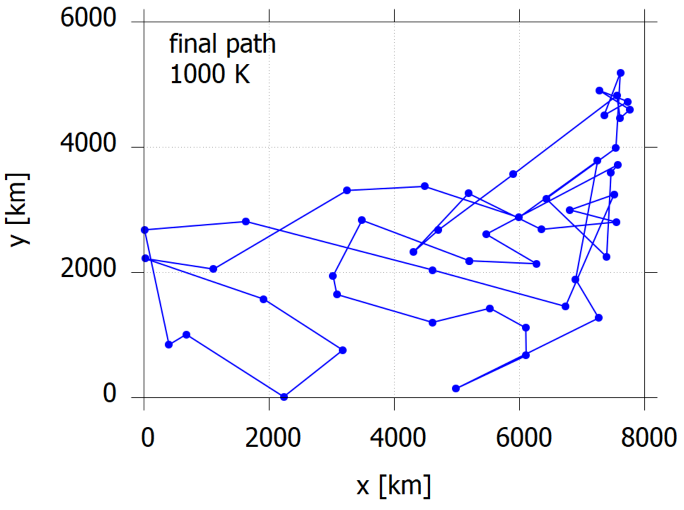
Cooling Strategy	$\overline{d}_{\text{final}}$ [km]	$\overline{\delta D}$ [km]	d_{opt} [km]	av. %acceptance
constant: T=1000 K	71000 ± 8000	-94000 ± 5000	60752.03549	20.65 ± 0.16
constant: T=2 K	46000 ± 4000	-115000 ± 15000	41524.04111	2.26 ± 0.04
linear	36200 ± 1000	-132000 ± 5000	34909.12616	10.22 ± 0.15
exponential	37300 ± 1600	-125400 ± 10400	35508.39983	15.00 ± 0.12
sigmoid	38800 ± 1400	-109000 ± 12000	36684.65766	28.40 ± 0.18
stepwise	36400 ± 1600	-116000 ± 9000	35092.65127	16.30 ± 0.14

Table 2: Results for the US capitals: average final path length $\overline{d}_{\text{final}}$, average decrease of path length in one run $\overline{\delta D}$, best result d_{opt}, and average percentage of accepted position swaps for each of the applied cooling strategies. For the means, the standard deviation is given as a measure of dispersion, respectively.

First of all, one can easily see that for all the temperature schedules, the algorithm of Monte-Carlo Metropolis could find a path that is significantly shorter than the random initial path. However, for the two runs with constant temperatures (see fig. 5 and 6) it is clearly visible that the final paths are not the best possible solution even without looking at the numbers: In both cases there are still some "unnecessary" turning points/ways back and forth, especially for the very high acceptance rate (1000 K).
For the other four runs including cooling (see fig. 7 to 10) one cannot easily decide which one provides the best solution. This is in line with the results in table 2 discussed above.

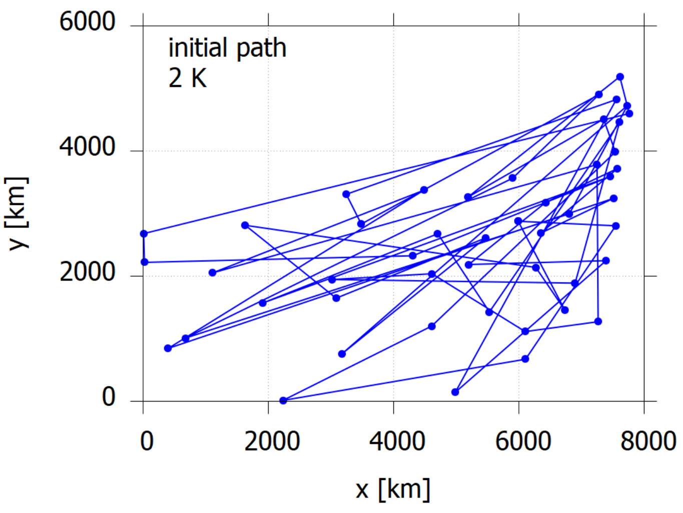


(a) initial path

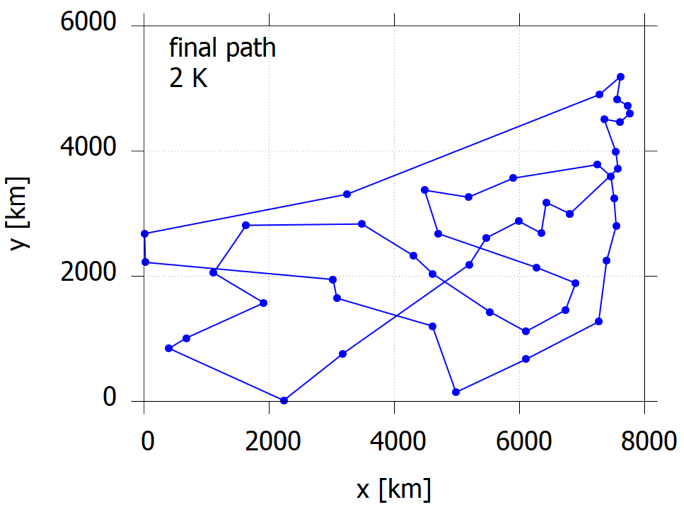


(b) final path

Figure 5: Random initial path (a) and optimized final path (b) to visit all 48 US capitals obtained using a constant "high temperature" of T=1000 K.

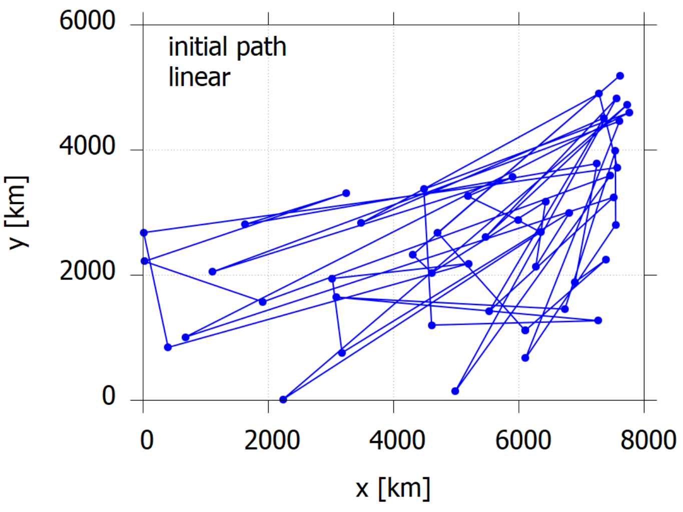


(a) initial path

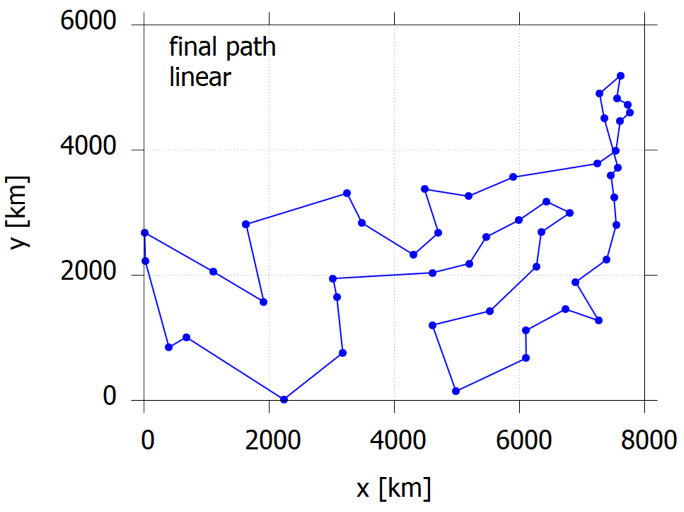


(b) final path

Figure 6: Random initial path (a) and optimized final path (b) to visit all 48 US capitals obtained using a constant "low temperature" of T=2 K.

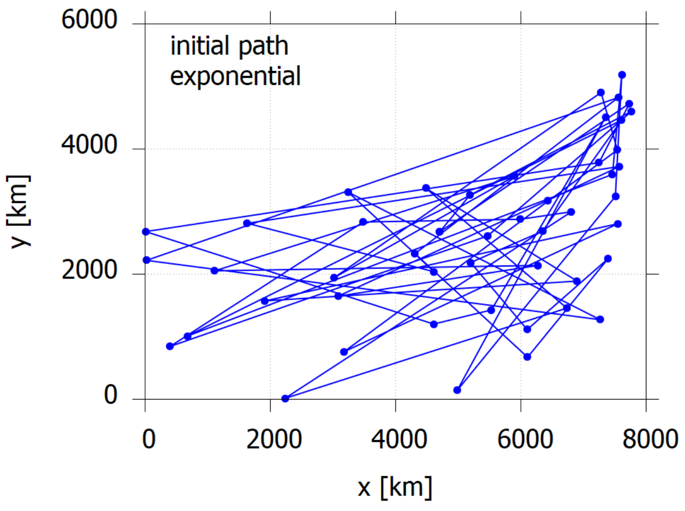


(a) initial path

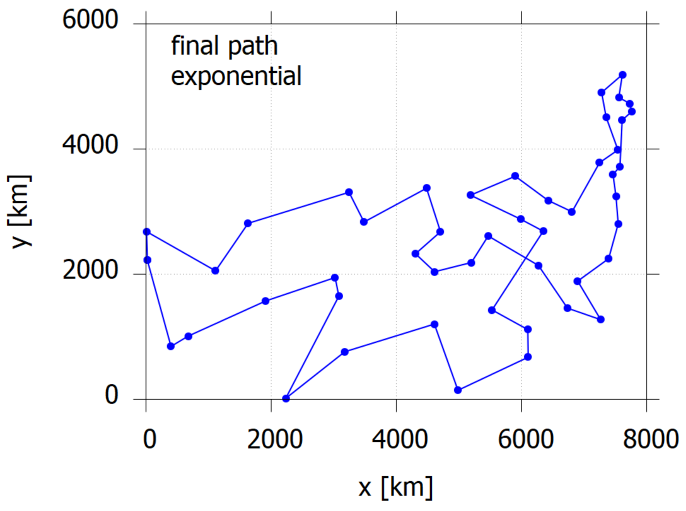


(b) final path

Figure 7: Random initial path (a) and optimized final path (b) to visit all 48 US capitals obtained using linear cooling.

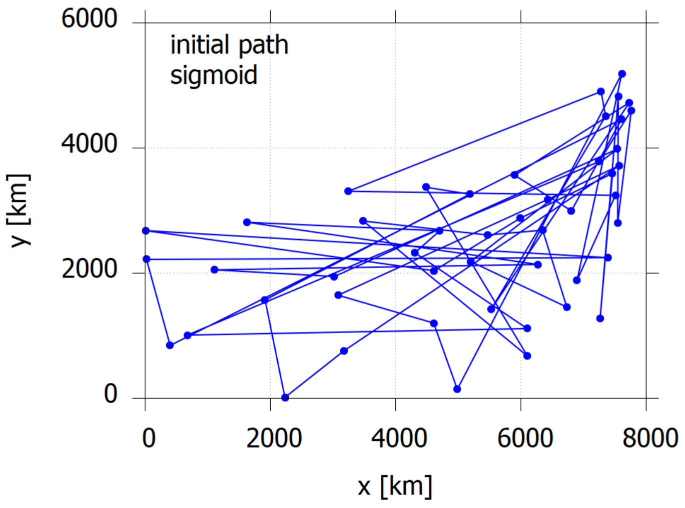


(a) initial path

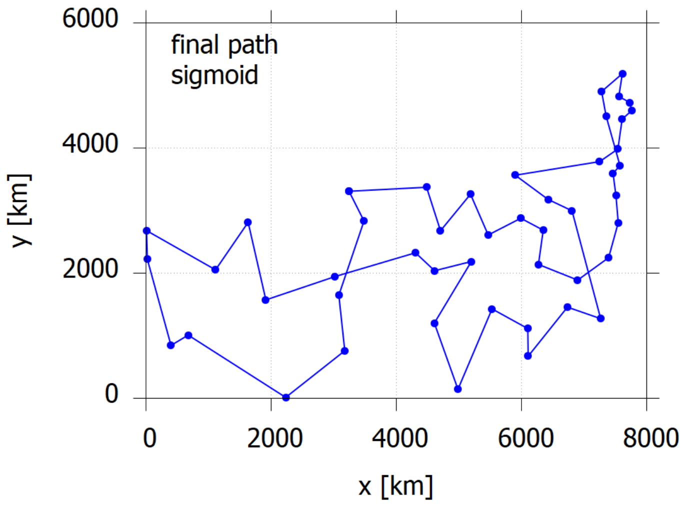


(b) final path

Figure 8: Random initial path (a) and optimized final path (b) to visit all 48 US capitals obtained using exponential cooling.

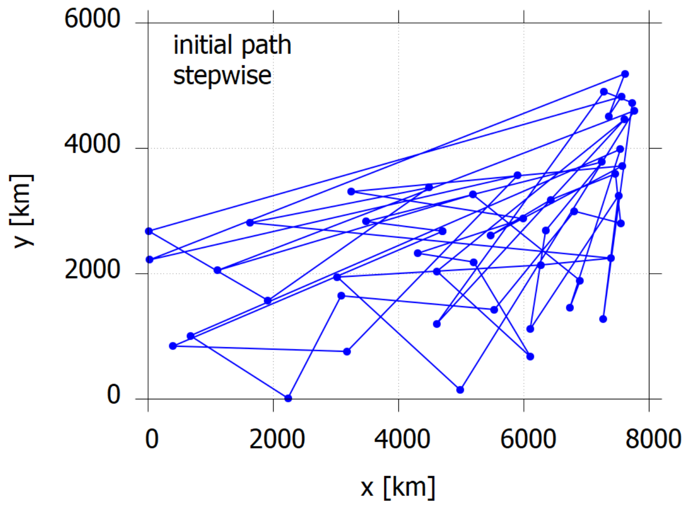


(a) initial path

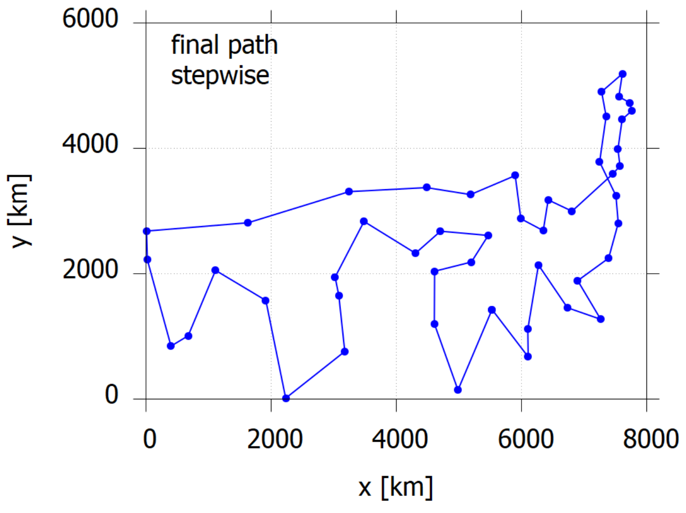


(b) final path

Figure 9: Random initial path (a) and optimized final path (b) to visit all 48 US capitals obtained using sigmoid cooling.



(a) initial path



(b) final path

Figure 10: Random initial path (a) and optimized final path (b) to visit all 48 US capitals obtained using stepwise cooling.

Fig. 11 shows the path length as a function of the current iteration step for the runs with the lowest final path length for the different temperatures.
In the beginning during the first few iterations, there is a fast decrease of the total path length except for the sigmoid and stepwise cooling, for which the first position swaps do not seem to significantly decrease the path length but make it fluctuate around the starting value. The reason for this is, that the temperature remains very high for these two strategies in the beginning, even higher than the one with the constant temperature of 1000 K (see fig. 4). Therefore, a high number of position swaps are accepted which do not necessarily decrease the path length but are rather random.
This trend is also clearly visible further into the simulations: the higher the temperature, the stronger are the fluctuations of the path length and vice versa. For example, for the run with a constant and very low temperature of 2 K, the path length does not significantly change anymore after the first falloff in the very beginning. We seem to be stuck in a local minimum and do not manage to get out of it until the end of the simulation because "disadvantageous" changes are accepted with an extremely low probability.
For all the other strategies fluctuations decrease in the course of the simulation after some time according to their cooling effects and the path length does not significantly change anymore in the end (from approximately $80 \cdot 10^3$ iterations on). The only exception here is the red curve for the run with a constant high temperature of 1000 K. As the temperature remains high until the end, a lot of "bad" position changes are accepted again and again, so that the algorithm never really converges towards a minimum. However, this demonstrates that the chosen number of $N = 10^5$ iterations in total is high enough and that we do not expect to get significantly better results by increasing N.

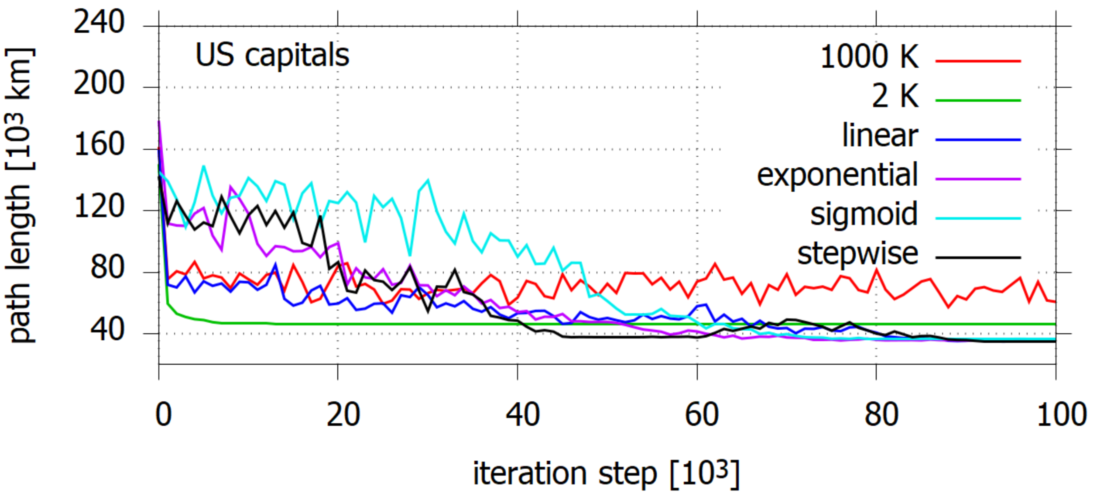


Figure 11: Total path length as a function of the iteration step of the run with the smallest final path length for the different temperature schedules for the US capitals.

3.3 Beer Gardens in Augsburg, Germany

For the map of 127 beer gardens in the German city of Augsburg five different annealing schedules were used: A constant temperature of 350 K, a linear temperature descending from 350 K to 0 K, a stepwise cooling scheme with the steps 3000 K $\rightarrow$ 1000 K $\rightarrow$ 50 K $\rightarrow$ 400 K $\rightarrow$ 2 K, an exponential cooling with $T_0 = 3000\ K$ and $\alpha = 5 \cdot 10^4$, and lastly a sigmoid cooling with $\alpha = 4 \cdot 10^4$ and $\beta = 10^4$. The temperature as a function of the iteration step k for the different cooling schemes can also be seen in fig 4.
As in the previous section, the Boltzmann constant was fixed to $k_B = 1$ but unlike the previous map, the number of iterations was varied, and the initial map remained the same by setting a fixed seed using a function given in a previous assignment (See TP3 MU4PY108 Physique numérique et projet). Varying the number of iterations was done to investigate the effects of the total number of iterations on runs with the different annealing temperatures. Keeping the initial maps fixed allowed us to investigate the effectiveness of different annealing schedules for a particular example. Figure 13 shows the final path length for different cooling schedules. As a comparison, the optimal solution for this particular problem is known and given with a path length of 118282 m [8]. From the figure, one can clearly see that for all cases one needs iterations of at least 10^5 in order to get any meaningful reduction in distance. For the case of 10^3, most annealing schedules had decreased by around half with the sigmoid case only fractionally decreasing. For the exponential case, after 10^5 iterations there was no decrease in distance which led to it being the worst cooling schedule overall, this is due to the fact that the acceptance rate at this point was very small $(5 \cdot 10^{-3})$ and so no new jumps were made. The best result was the stepwise annealing schedule which achieved a minimum distance of 138856 m which is only 17.4% higher than the best known solution.

Looking at the acceptance rates (Figure 12) for the different cooling temperatures we can see the different strategies that each takes. Exponential, Linear and constant cooling start off with a high acceptance which is then reduced rapidly thus converging on a particular minimum quickly. Sigmoid annealing reduces its acceptance more slowly but also ends up with a very low acceptance. The issue with these approaches is that they do not necessarily converge on the correct minimum and solutions that slowly drop their acceptance rate and keep it higher were the most likely to fall into a lower minimum, as seen in the stepwise case. A reason why the stepwise cooling schedule performed the best is that it decreased and increased the temperature over time. The increasing stages allowed it to escape bad local minima and thus converge on a better solution.

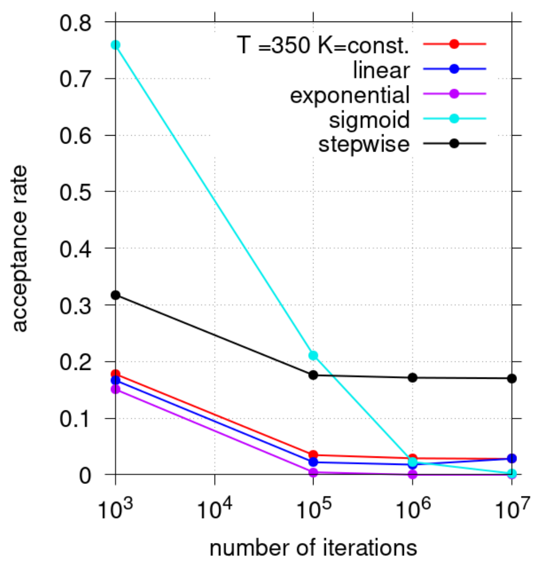


Figure 12: Acceptance rate as a function of the total number of iterations for the best runs of the different temperature schedules for the beer garden map.

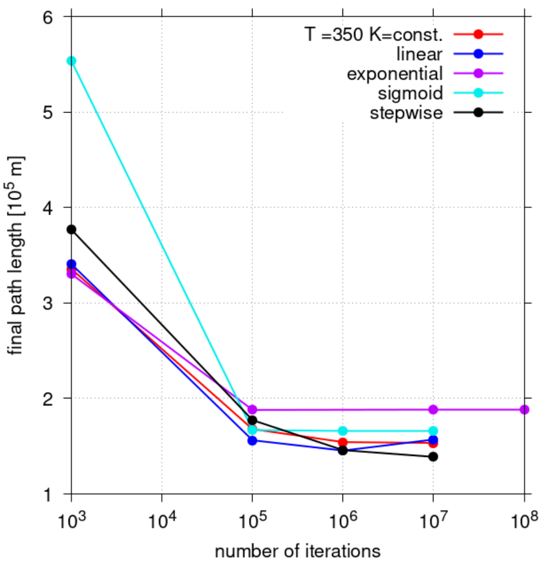


Figure 13: Final path length for the beer garden map as a function of the total number of iterations for the different temperature schedules.

Figure 14 shows the initial path length with a distance of 606550 m. The final paths are shown in figure 15 of the run that provided the lowest final path length for every temperature scheme. Again , although the coordinates are given in km the geography of the points in the plane are not of importance as we are only interested in the distances and relative positions of the beer gardens. In general you can see that the final paths are much cleaner than the initial path and you can clearly observe a decrease in distance when comparing them. However, the figures still contain many intersecting paths which would usually point to the fact that a global minimum has not been achieved but as our best solution was only 17.4% off the best known solution the reason for the unclear paths is likely the high number of nodes.

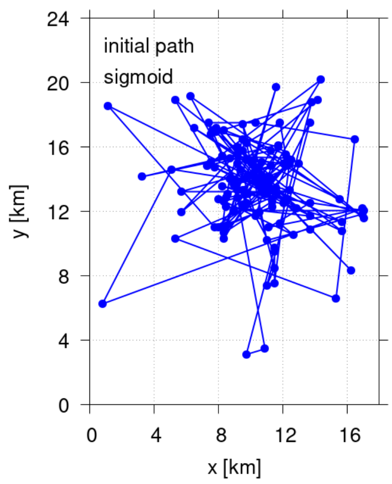
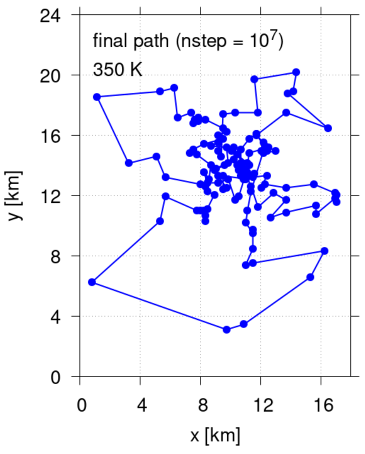
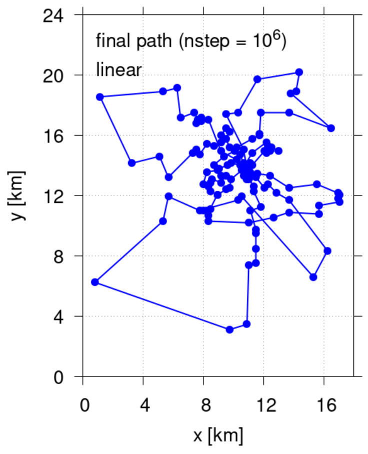


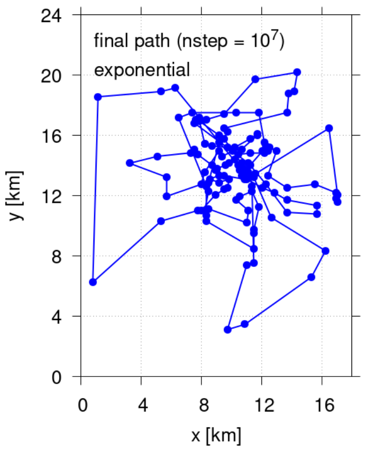
Figure 14: Initial path for the different runs for the beer garden map. As a random seed was chosen for comparability, it is the same for all.



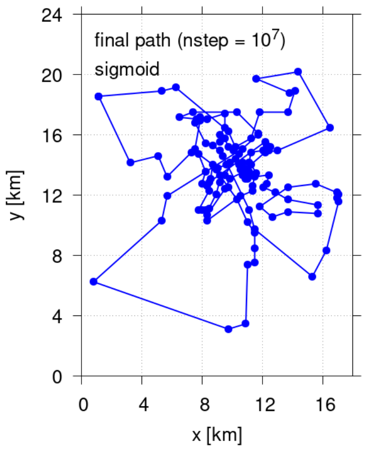
(a) 350 K, 10^7 iterations



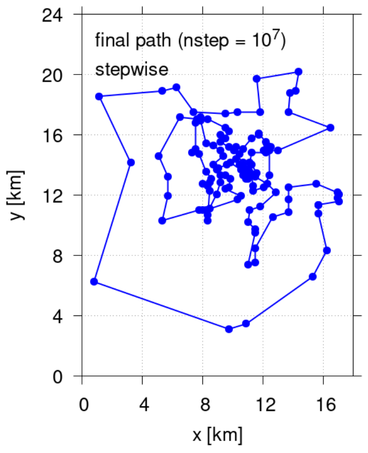
(b) linear, 10^6 iterations



(c) exponential, 10^7 iterations



(d) sigmoid, 10^7 iterations



(e) stepwise, 10^7 iterations

Figure 15: Final paths to visit the beer gardens of the runs with the number of iterations that yield the smallest final path length for the different temperature schedules.

4 Conclusion and Outlook

All in all, for our simple test map of forty points on a unit circle, constant high acceptance probabilities of "bad" changes did not manage to significantly decrease the path length compared to a random initial path. In contrast, a relatively small and constant probability lead to reasonable results. A very low but non-zero value even lead to better results than most of the strategies including cooling.
The best result was obtained with a stepwise changing probability because it was able to escape from non-optimal local minima by temporarily increasing p again after a first decrease. In this fashion we were even able to reproduce the known global optimum for this simple map of simply following the contour of the circle in one sense or the other.

For our first bigger and more complex map of the capitals of the 48 Contiguous US states we can conclude that for a constant temperature a low value generally gives better results than a very high value but bears the risk of getting stuck in a local minimum.
Furthermore, significantly better results could be obtained introducing a cooling strategy. The four different ones that were tested for the circle and used again here for this more complicated map did not show any major differences in their respective results. However, none of the methods managed to get close to the global optimum.

For the map of the beer gardens in Augsburg we conclude that a stepwise simulated annealing schedule achieved the smallest distance due to its ability to escape bad local minima as explained above in the circle example. We observed that iterations of at least 10^5 were required to achieve an acceptable score and that higher iterations generally decreased the path difference but by smaller amounts.

To improve our results, the number of runs with different initial conditions could be increased in the future in order to "explore" a wider range of the distance landscape to find a "good" minimum and obtain better statistics.
We also could have investigated a greater number of simulated annealing schedules and fine tune the parameters of the schedules we already explored here in order to match the best known solutions given by the dataset. Due to limited time and because we wanted to explore a wider range of different maps and parameters this was out of the scope for this project.

There are a lot of possible variations on the classical TSP which include applications to realistic problems. An historically important example is the arranging of school bus routes which motivated M. Flood, one of the pioneers of the TSP. Other examples are the scheduling of service calls at cable firms, the delivery of meals to homebound persons, the scheduling of stacker cranes in warehouses, the routing of trucks for parcel post pickup, and many more.[9]
Possible variations include:

- *Multiple TSP*: Multiple traveling salesmen share out the cities that have to be visited between them but they still all start and stop in the same place. The goal is to minimize their summed path lengths. One can apply the restriction of *nonlazy* salesmen meaning that journeys covering less than two cities are forbidden.[10]
- *Generalized TSP:* The cities are clustered and the salesmen only has to go to at least one city of each cluster.[11]
- Restrictions on the "routes": Our traveling salesmen here always traveled as the crow flies but in more realistic scenarios one can include barriers, one-way-streets, or blind-alleys and so on.
- *Prize collecting TSP:* The traveller sells items in every city for a different prize or a different amount according to the local economic situation but also has to pay different prizes to travel from one city to another (e.g. train/airplane ticket, ...) and a penalty for cities he fails to visit. This comes also with a change of the optimization criterion: the objective is now to find the route that maximizes the profit.[12]

- Including time windows: Time schedules are a common restriction in applications of the TSP, for example in the tour planning of a technician who indicates a certain time window when he will come to his customer's houses.
- *Online TSP:* Not all the places are known in the beginning but get only announced from time to time and the salesman has to decide on how to best include them into his route. This is for example the case in the tour planning of a breakdown service (usually a *Multiple Online TSP with time restrictions*).[13]

The TSP is still relevant and of interest for real life applications today. For example the shipping agency *UPS* introduced the so-called ORION (On-Road Integrated Optimization and Navigation) System in 2013 in order to reduce delivery times instead of leaving it completely up to their drivers.[14]

There are also more generalized applications where the notion of "distance" and "cities/places" are changed but the problem itself remains the same and therefore allows for the use of the same algorithms. Some more abstract examples of applications are:

- Genetic Engineering: Embedding a collection of DNA target strings in one universal string trying to minimize the length of the universal string. The "cities" here are the target strings and the "distances" are the amount of non-overlapping nucleobases between two of them.[9]
- Design of microchips: Minimizing the length of scan chains for timing and power reasons.[9]
- Clustering a data array in order to maximize the sum of all horizontally or vertically adjacent elements of the array.[15]
- Scheduling jobs on machines while minimizing the total processing time.[15]

References

[1] Biggs, Lloyd, & Wilson. *Graph Theory, 1736–1936*, (1986)

[2] R. M. Karp. *Reducibility Among Combinatorial Problems*, (1972)

[3] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller. *Teller Equation of State Calculations by Fast Computing Machines*, (1953)

[4] Master de Physique MU4PY108 (Physique numérique et projet): *Cours III : Processus aléatoires et méthode de Monte Carlo.* Sorbonne Université (2021), Paris

[5] Kirkpatrick Gelatt Vecchi. *Optimization by simulated annealing* (1983)

[6] Padberg, Rinaldi: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/att48.tsp (last accessed: 03/12/2021)

[7] Juenger, Reinelt: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/bier127.tsp (last accessed: 03/12/2021)

[8] G. Reinelt. *Best known solutions to symmetric TSPs* (02/1997). http://elib.zib.de/pub/mp-testdata/tsp/tsplib/stsp-sol.html (last accessed: 06/12/2021)

[9] https://www.math.uwaterloo.ca/tsp/apps/index.html (last accessed: 07/12/2021)

[10] T. Bektas. *Omega, Elsevier*, vol. 34(3), 209-219 (2006)

[11] C.E. Noon. *The generalized traveling salesman problem.* PhD Thesis (1988), University of Michigan Library

[12] E. Balas. *NETWORKS* vol. 19 (1989) 621-636 doi.org/10.1002/net.3230190602

[13] P.-C. Chen, E. D. Demaine, C.-S. Liao, H.-T. Wei (2019) arXiv:1907.00317

[14] Wired.com (2013): https://www.wired.com/2013/06/ups-astronomical-math/ (last accessed: 07/12/2021)

[15] J.K. Lenstra & A.H.G. Rinnooy Kan (1975) *Operational Research Quarterly* (1970-1977), vol. 26(4), 717–733. doi.org/10.2307/3008306

5 Annex

5.1 Fortran 90 Code

In the following, our *Fortran 90* code to find solutions to the TSP is provided. As an example a version is given which would use the 48 US capitals as a map.

```fortran
module param
   implicit none
   integer, parameter :: n_cities = 48, T0 = 3000  ! nombre de villes, temperature
    initiale
   real(8), dimension(n_cities,2) :: xvec, xvec_new  ! positions des villes dans l'
    ordre
end module param



program main
   use param
   implicit none

   real(8) :: distance, diff, temperature_function  ! fonctions

   integer :: m, istep
   integer :: nstep = 100000  ! nombre d'iterations
   real(8), parameter :: kB = 1  ! constante de Boltzmann
   real(8) :: T  ! temperature fictive [K]
   integer :: compteur_ta  ! compteur de tirages acceptes

   real(8) :: D, dD  ! distance, difference en distance entre deux parcours
   real(8) :: s1, s2, s  ! nombres aleatoires
   integer :: k1, k2  ! indices des points a echanger

   ! initialiser xvec
   call initialize_xvec()

   ! ouvrir des fichiers pour y ecrire quelques resultats plus tard
   open(1, file='pos_init_us.res')
   open(2, file='dist_us.res')
   open(3, file='pos_fin_us.res')

   ! ecrire dans un fichier les positions initiales
   do m = 1, n_cities
      write(1,*) xvec(m,1), xvec(m,2)
   enddo
   close(1)

   ! calculer et ecrire a l'ecran la distance initiale
   D = distance()
   write(*,*) "distance initiale =", D, "km"

   write(2,*) "0", D

   ! initialiser le compteur de tirages acceptes
   compteur_ta = 0


   !!! BOUCLE MONTE-CARLO !!!
   do istep = 1, nstep

      T = temperature_function(istep, nstep)

      ! tirage des points au hasard
      call random_number(s1)
      call random_number(s2)
      k1 = floor(n_cities*s1)+1
      k2 = floor(n_cities*s2)+1

      ! tirage des points a echanger dans l'ordre
      xvec_new = xvec
      xvec_new(k1,1) = xvec(k2,1)
      xvec_new(k1,2) = xvec(k2,2)
      xvec_new(k2,1) = xvec(k1,1)
      xvec_new(k2,2) = xvec(k1,2)

      ! calcul de la difference de distance dD induite par l'echange
      dD = diff(k1, k2)

      ! application du critere de Metropolis
      if (dD.lt.0) then
         xvec = xvec_new
         D = D + dD
         compteur_ta = compteur_ta + 1

      else
         call random_number(s)
         if (s.le.exp(-dD/(kB*T))) then
            xvec = xvec_new
            D = D + dD
            compteur_ta = compteur_ta + 1
         endif

      endif

      ! ecrire (tous les 1000 pas) la nouvelle distance dans un fichier
      if (mod(istep,1000).eq.0) then
         write(2,*) istep, D
      endif

   enddo

   close(2)


   ! ecrire a l'ecran la distance finale et le taux d'acceptation des tirages
   write(*,*) "distance finale =", D, "km"
   write(*,*) "taux d'acceptation des tirages =", real(compteur_ta)/nstep

   ! ecrire dans un fichier les positions finales
   do m = 1, n_cities
      write(3,*) xvec(m,1), xvec(m,2)
   enddo
   close(3)

end program main



! ---------------------------------------------------------------------------------
! ------- subroutine pour lire la carte et initialiser un chemin randomise -------
! ---------------------------------------------------------------------------------
subroutine initialize_xvec()
   use param
   implicit none
   real(8) :: b1, b2  ! nombres aleatoires
   integer :: b, i, c1, c2  ! integers pour boucles et nombres aleatoires
   real(8), dimension(1,2) :: temp

   ! lire les coordonnees des villes dans un ordre initial
   open(4, file='datasets/us48.txt')
   do i = 1, n_cities
      read(4,*) xvec(i,1), xvec(i,2)
   enddo

   ! randomiser l'ordre
   do b = 0 , (n_cities*5)
      call random_number(b1)
      call random_number(b2)
      c1 = floor(n_cities*b1 + 1)
      c2 = floor(n_cities*b2 + 1)

      temp(1,1) = xvec(c1,1)
      temp(1,2) = xvec(c1,2)
      xvec(c1,1) = xvec(c2,1)
      xvec(c1,2) = xvec(c2,2)
      xvec(c2,1) = temp(1,1)
      xvec(c2,2) = temp(1,2)
   enddo

   close(4)

end subroutine initialize_xvec



! ---------------------------------------------------------------------------------
! -- fonction pour calculer la difference de longueur totale entre deux chemins --
! ----       dont les villes a positions k1 et k2 ont ete echangees            ----
! ---------------------------------------------------------------------------------
real(8) function diff(k1, k2)
   use param
   implicit none
   integer :: k1, k2, i
   real(8) :: distance, distance_new

   diff = 0.0
   distance = 0.0
   distance_new = 0.0

   do i = 1, n_cities-1
        distance = distance &
                   + sqrt((xvec(i,1)-xvec(i+1,1))**2 &
                          + (xvec(i,2)-xvec(i+1,2))**2)
   enddo
   distance = distance &
              + sqrt((xvec(n_cities,1)-xvec(1,1))**2 &
                     + (xvec(n_cities,2)-xvec(1,2))**2)


   do i = 1, n_cities-1
        distance_new = distance_new &
                       + sqrt((xvec_new(i,1)-xvec_new(i+1,1))**2 &
                              + (xvec_new(i,2)-xvec_new(i+1,2))**2)
   enddo
   distance_new = distance_new &
                  + sqrt((xvec_new(n_cities,1)-xvec_new(1,1))**2 &
                         + (xvec_new(n_cities,2)-xvec_new(1,2))**2)

   diff = distance_new - distance

end function diff


! ---------------------------------------------------------------------------------
! ------------ fonction pour calculer la longueur totale d'un chemin -------------
! ---------------------------------------------------------------------------------
real(8) function distance()
   use param
   implicit none
   integer :: i

   distance = 0.0

   do i = 1, n_cities-1
      distance = distance &
                 + sqrt((xvec(i,1)-xvec(i+1,1))**2 &
                        + (xvec(i,2)-xvec(i+1,2))**2)
   enddo
   distance = distance &
              + sqrt((xvec(n_cities,1)-xvec(1,1))**2 &
                     + (xvec(n_cities,2)-xvec(1,2))**2)

end function distance



! ---------------------------------------------------------------------------------
! ----- fonction pour calculer la temperature au pas d'iteration istep selon -----
! --  le schema choisi (constant, pas a pas, lineaire, exponentiel ou sigmoide) --
! ---------------------------------------------------------------------------------
real(8) function temperature_function(istep, nstep)
   use param
   implicit none
   integer :: istep, nstep
   real :: linear_temp, exp_temp, const_temp, stepwise_temp, sigmoid_temp

   linear_temp = T0 - istep * (T0-1)/nstep
   exp_temp    = T0 * ((0.8)**(istep/5000))  ! Kirkpatrick, Gelatt, Vecchi (1983)
   const_temp = T0
   sigmoid_temp = 2500 / (0.5 + exp((istep*1.0-40000)/10000))

   ! stepwise cooling
   if (istep.lt.nstep/9) then
      stepwise_temp = 3000
   else if (istep.lt.(2*nstep/9).and.istep.ge.(nstep/9)) then
      stepwise_temp = 3000 - (istep-nstep/9)*2000/(nstep/9)
   else if (istep.lt.nstep/3.and.istep.ge.(2*nstep/9)) then
      stepwise_temp = 1000
   else if (istep.lt.(4*nstep/9).and.istep.ge.(nstep/3)) then
         stepwise_temp = 1000 - (istep-nstep/3)*950/(nstep/9)
   else if (istep.lt.(5*nstep/9).and.istep.ge.(4*nstep/9)) then
       stepwise_temp = 50
    else if (istep.lt.(2*nstep/3).and.istep.ge.(5*nstep/9)) then
       stepwise_temp = 50 + (istep-5*nstep/9)*350/(nstep/9)
    else if (istep.lt.(7*nstep/9).and.istep.ge.(2*nstep/3)) then
       stepwise_temp = 400
    else if (istep.lt.(8*nstep/9).and.istep.ge.(7*nstep/9)) then
        stepwise_temp = 400 - (istep-7*nstep/9)*398/(nstep/9)
    else if (istep.ge.(8*nstep/9)) then
       stepwise_temp = 2
    endif


   temperature_function = sigmoid_temp  ! choisir la temperature/le cooling

end function temperature_function
```