The Travelling Salesman problem with Simulated Annealing

Joachim Pomper

February 2021

Contents

1 Basics

In this project the solution to the Travelling Salesman Problem with the help of a stochastic optimization algorithm, called Simulated Annealing is investigated. The focus lies on different aspects and approaches of the Simulated Annealing method rather than producing an optimal, fast running algorithm. The code in section 7 is therefore not fully optimized and quite general. In some cases possible optimizations are pointed out.

1.1 Travelling Salesman Problem

The intention of the Travelling Salesman Problem is to find the shortest connection between a discrete set of points. In our case we want to find a round-trip between L points x_i, placed within a square with an edge length of 1. The distance between the points is measured by the euclidean metric of $\mathbb{R}^2$. This can be seen as a discrete optimization problem where possible configurations are represented by a finite series of points in $\mathbb{R}^2$

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_L) \tag{1.1}$$

and the cost-function to be minimized is given by

$$f(\boldsymbol{x}) = \|x_1 - x_L\|_2 + \sum_{i=1}^{L-1} \|x_{i+1} - x_i\|_2 \tag{1.2}$$

1.2 Simulated Annealing

The Simulated Annealing algorithm is inspired by the physical process of a liquid freezing out to from a crystal. When cooled slowly, the molecules can arrange in optimal patterns, minimizing the free energy of the crystal. The advantage, in comparison to for example gradient or steepest descent methods, is that there is always a finite probability to go to configurations of higher cost and therefor overcome wells and barriers in the optimization-terrain to find a better minimum. In order to find a rather good minimum of our cost-function over the given discrete configuration space, the cost of a configuration is interpreted as an artificial energy. To minimize this energy, we follow these steps:

i) We propose a new configuration.

ii) If the energy is lower, we accept the proposed configuration as a new configuration.

iii) If the energy is higher, we accept the proposed only with a certain probability.

The last point guarantees that we can leave a local minimum and proceed in finding a better one. While the algorithm runs through a lot of different configurations, the best solution found yet is stored separately. We introduce an artificial inverse temperature β, which we use to control how much the energy can change when accepting a configuration with higher energy. Increasing this β value during the simulation leads to a reduction of movement in the configuration space until the algorithm eventually gets stuck in a local minimum. In order to understand the various aspects of this algorithm we want to consider some of the crucial points.

1.2.1 Probability Distribution

We have to choose a probability distribution for the acceptance probability. A common choice for the so called "Classic Simulated Annealing" algorithm is a Boltzmann-factor. Then the probability to accept a proposed configuration is given by

$$p_{accept} = \min\left(1, \mathrm{e}^{-\beta\Delta E}\right) \tag{1.3}$$

where ΔE is the difference in energy between the proposed and the current configuration.

$$\Delta E = f(\boldsymbol{x}_{proposed}) - f(\boldsymbol{x}_{current}) \tag{1.4}$$

This choice of acceptance probability is connected to the Metropolis Hasting Algorithm with a canonical probability distribution

$$p(\boldsymbol{x}) = \frac{\mathrm{e}^{-\beta f(\boldsymbol{x})}}{Z} \tag{1.5}$$

and a symmetric proposal probability. Note that this is not the only way to choose the acceptance probability. In this project the influence of the choice of the form for the acceptance probability is not further discussed.

1.2.2 Proposing a new configuration

The choice of how to propose a new state is crucial for the quality of the results and the convergence properties. When proposing a new state it is important to consider that every configuration can be reached (ergodicity). Also, the movement through configuration space and the rate of acceptance depend very much on the states proposed.

1.2.3 Minimum and maximum of the inverse temperature

The choice of the minimum value of the inverse temperature is essential. If too big, it will not be possible to move freely in configuration space at the beginning and the result will depend strongly on the initial condition. A rule of thumb is to choose β_{min} in such a way that the rate of acceptance is higher than 80% at the beginning. The choice of the maximum value influences the convergence of the algorithm. In principle one does not have to choose a fixed value. One can increase the value of β further, until the algorithm appears to stop improving the result.

1.2.4 Criteria of convergence

It is rather hard to find a good criterion of convergence. One could check the number of optimization steps where the current minimal energy value did not change and consider the algorithm as converged if this number is bigger then some threshold value. Another possibility, which is investigated in this project, is to set a boundary for a minimal rate of acceptance. If the rate of acceptance becomes very low, it is not very likely that we will discover a better minimum and the algorithm becomes very inefficient anyways. This threshold value however strongly depends on the method to propose new states.

1.2.5 Cooling strategies

At last one can think of different strategies for increasing the inverse temperature. A too quick cooling increases the chance of getting stuck at some non optimal minimum. Just like a crystal takes on a polymorphic structure. Too slow cooling will lead to long computational times. The ideal cooling process can be very specific to the problem. As an example consider the following case of the Travelling salesman problem, where we have some cities with a lot oh houses each. In order to find a way to visit all houses in an optimal way the salesman has to deal with to different questions. In which order and where will he enter the cities and then in which order does he visit the houses in each city. The two questions reflect different scales of energy. The first one will be decided at small β values, where large changes in energy are possible. The second is decided at higher inverse temperatures. Cooling too fast at the beginning will therefore lead to a possibly bad decision for the order of visiting the towns. Also one could try to stop the algorithm at some point and use the found configuration as starting point at some lower finite inverse temperature. This could be understand as some sort of tempering of the system.

2 Experiment: Choosing different ways to propose new configurations

First we want to investigate the influence of the choice of different ways to propose a new configuration. We consider the following four methods:

P1 : Two randomly picked points in the chain are swapped.

P2 : Two points in the chain are picked randomly and the order of all points in between, including the points picked, is inverted.

P3 : A random section of random length is taken out and inserted at some other point of the chain.

P4 : Two randomly picked neighboring points in the chain are swapped.

To test these methods, we try to find the minimum of the cost function for the pointset in figure 1. The set is constructed in such a way, that the exact solution to this problem can be estimated from geometric and convex hull considerations. The global minimum is assumed to be

$$\min f(\boldsymbol{x}) = 3.131408622 \tag{2.1}$$

The pointset consists of 29 points, grouped in 3 cluster, so that there a two different length scales for the problem.
As cooling strategy we use a linear increase of the inverse temperature given by

$$\beta_n = 0.05 + 0.05n \tag{2.2}$$

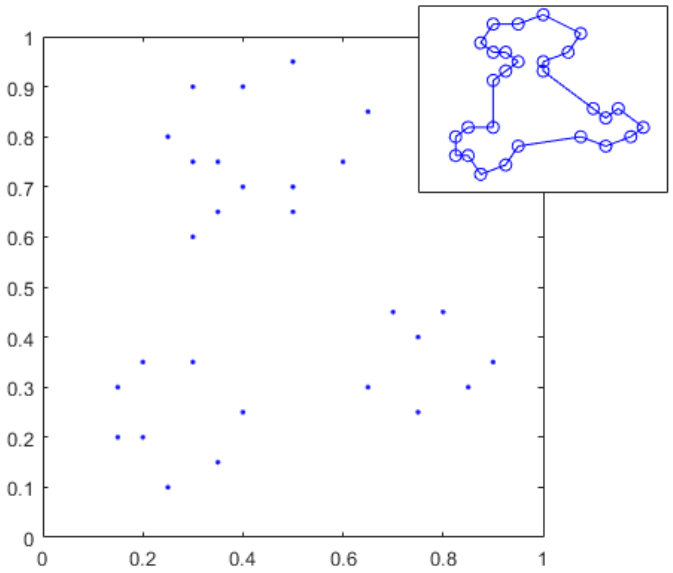

Figure 1: Pointset used to test the different proposal methods. The solution for the global minimum by geometric considerations is shown in the right corner.

We propose N_s different configuration before changing the inverse Temperature again. Changing the value of β will be called a cooling-step, while the act of proposing a new configuration will be called a sweep. A cooling-step therefore consists of N_s sweeps and when changing the inverse temperature N_b times, we perform $N_b \cdot N_s$ sweeps in total. A good rule of thumb seems to choose $N_s \approx L^2$. We therefore fix

$$N_s = 900 \tag{2.3}$$

for the following calculations. Figure 2 shows the result of such an optimization for each proposal method. The Method P4 seems to get stuck in some local minimum fairly easy. This is quite understandable, since the configurations proposed by this method differ very little from each other. Therefore the movement in configuration space is too slow.
For the further analysis two convergence criteria are introduced:

C1 : The minimum is assumed to be good, if the mean value of the acceptance rate of the last 10 cooling-steps drops below 1%.

C2 : The minimum is assumed to be good, if the minimum is the same for the last 100 cooling-steps.

To have comparable results, we will now perform the optimization 100 times for the proposal method P1 to P3. In figure 3 the mean values of the best approximations for the minimum are plotted for each proposal method. The β-values where the convergence criteria would have stopped the algorithm are also marked in figure 3.
In table 1 the results for the best minimum found are also summarized for the different convergence criteria and proposals. Note that the quantities with a ˆ-symbol denote the sample-mean

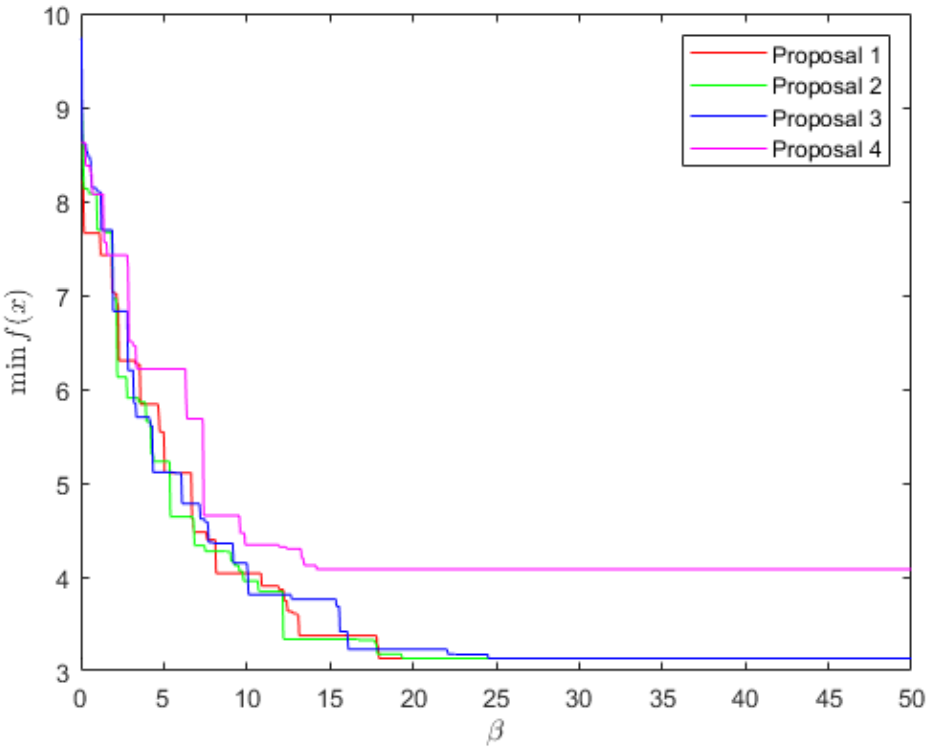

Figure 2: Plot of the best minimum found in dependence of the inverse temperature.

of the 100 optimization runs. The sample-deviation $\hat{\sigma}$ given by

$$\hat{\sigma}^2(\theta) = \frac{1}{N_b - 1} \sum_{i=1}^{N_b} (\theta_i - \hat{\theta})^2 \tag{2.4}$$

should not be confused with the estimate of the deviation of the sample-mean from the true mean, given by

$$\frac{\hat{\sigma}(\theta)}{\sqrt{N_b}}$$

We see that the proposal method P2 gives the best results. The variance of 0 occurs due to the algorithm calculating 100 times the same solution. This solution coincides with our geometric estimate of the global minimum. At last we can have a look at the rate of acceptance, which is visualized in figure 4 and 5 . We see that the choice $\beta_{min} = 0.05$ is okay, because the acceptance rate is way above 80%. We could even increase it up to $\beta_{min} = 1$. The rate of acceptance drastically decreases and eventually becomes less than 1% on average. Even thought the acceptance rate drops fastest for P3, we see that the algorithm nevertheless converges last for P3, when it comes to the criterion C2. Also C1 would stop the algorithm way to early for P3. Here a different threshold value for the convergence criterion C1 must be found. For P2 both criteria seem to work rather good. Also P2 has the perk that the change in energy can be calculated very efficiently, providing a possibility to implement a fast algorithm.

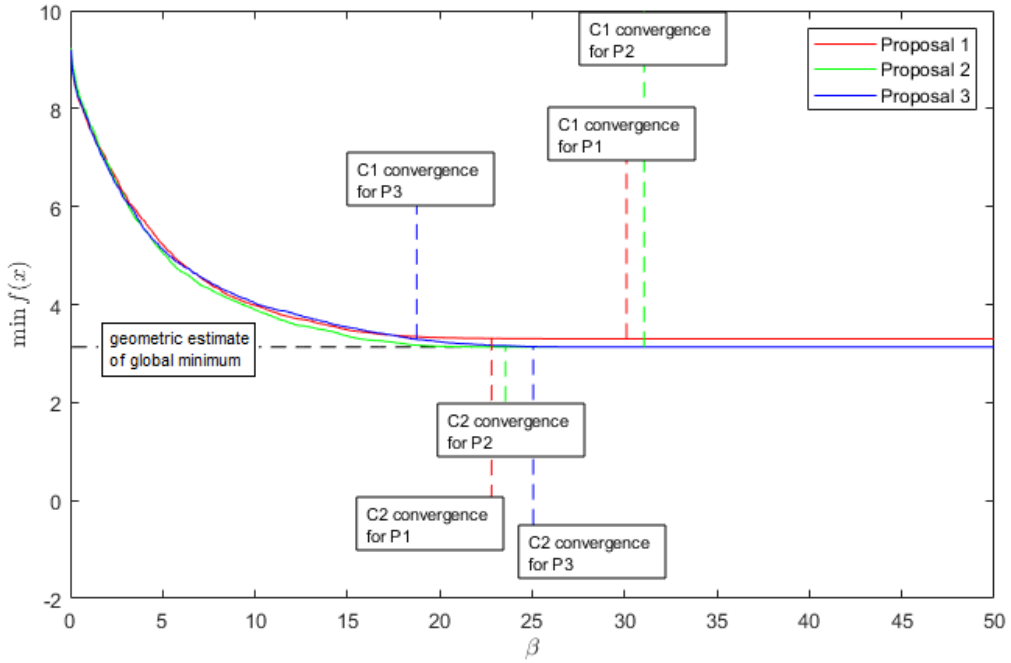

Figure 3: Mean values of the minimum found in dependence of the inverse temperature

Table 1: Summary of optimization results for different proposal methods

$\hat{\theta}$ ... Sample-mean of quantity θ
$\hat{\sigma}(\theta)$ ... Sample-deviation of quantity θ

Proposal Method	Convergence criterion	$\hat{\beta}_{max}$	$\hat{\sigma}(\beta_{max})$	$\hat{f}_{min}$	$\hat{\sigma}(f_{min})$
P1	C1	30.1	3.5	3.302	0.091
P1	C2	22.8	4.0	3.307	0.093
P1	none	50	0	3.300	0.091
P2	C1	30.1	1.3	3.131408622	0
P2	C2	23.6	2.8	3.131408622	0
P2	none	50	0	3.131408622	0
P3	C1	30.1	1.3	3.29	0.99
P3	C2	23.6	2.8	3.142	0.030
P3	none	50	0	3.1319	0.0046

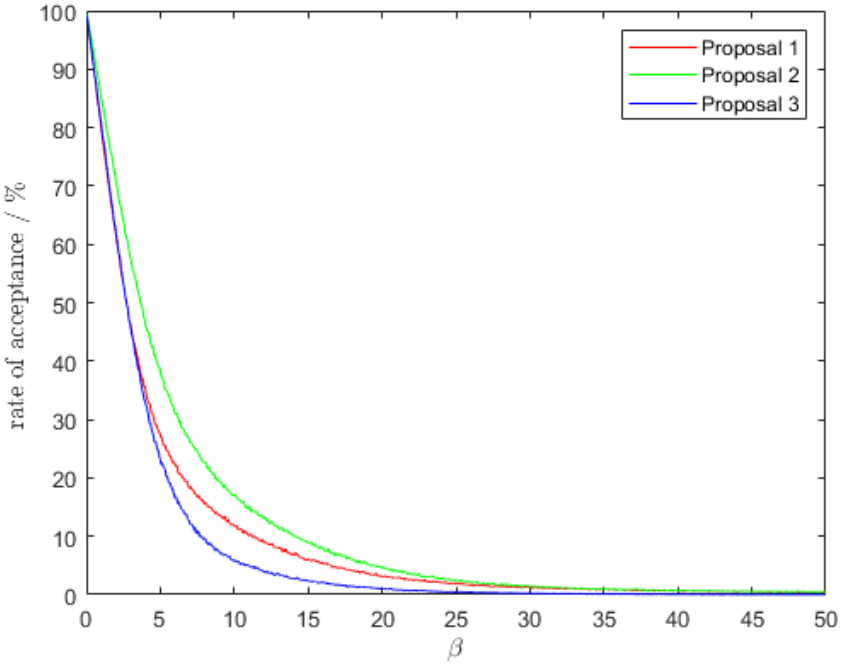

Figure 4: Plot of the mean rate of acceptance for the different proposal methods

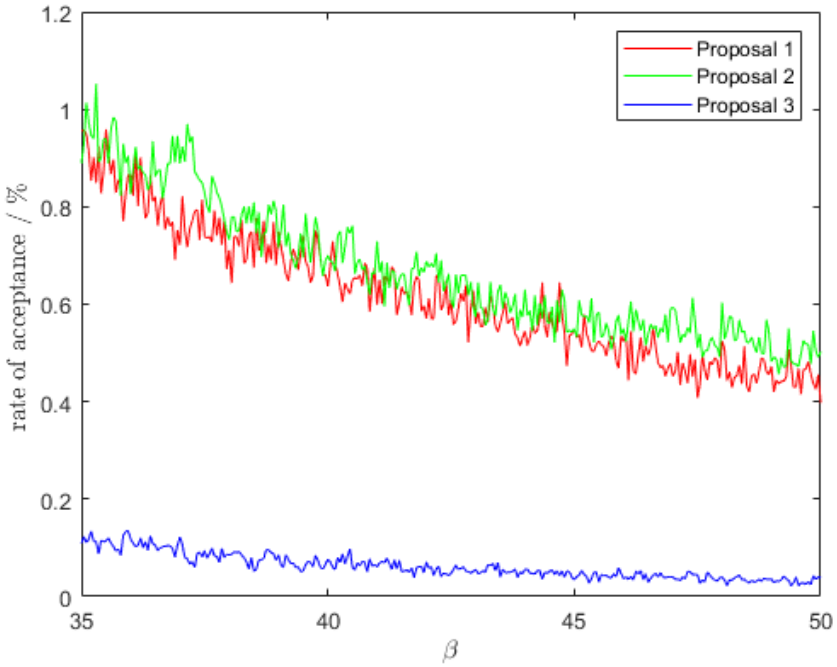

Figure 5: Plot of the mean rate of acceptance for the different proposal methods

3 Experiment: Comparing different cooling strategies

Next we have a look at different cooling strategies. We will test them on the set of points shown in figure 1, which we already used before. First three alternative, nonlinear cooling strategies are compared to a linear, similar to the strategy we already used:

$$\text{B1}: \quad \beta_n = 0.1 + 0.3996\, n \tag{3.1}$$

$$\text{B2}: \quad \beta_n = 0.1 + 33.1140\, n^{0.2} \tag{3.2}$$

$$\text{B3}: \quad \beta_n = 0.1 + 6.31823\, n^{0.5} \tag{3.3}$$

$$\text{B4}: \quad \beta_n = 0.1 + 0.0015984\, n^2 \tag{3.4}$$

Again we perform $N_s = 900$ sweeps per cooling step. Each cooling strategy is used 100 times to have comparable results. In figure 6 the results for the cooling strategies B1 to B4 are summarized.

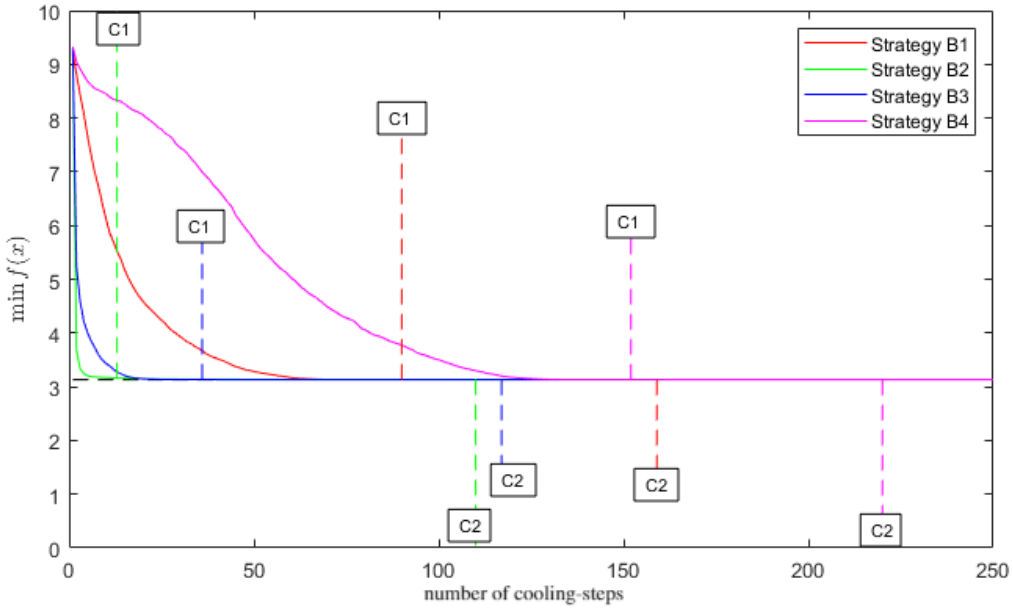

Figure 6: Sample-means of the found minima in dependence of the number of cooling-steps for the nonlinear cooling strategies. C1/C2 mark the points where the different convergence criteria would consider the result to be converged

In table 2 the mean number of cooling-steps and the sample-means and sample-deviations for the two different convergence criteria are also summarized. We see that the quick cooling leads to a bigger sample variation and a worse minimum on average. This is, as explained in section 1.2.5, due to too fast cooling. The strategies B2 and B3 do not seem to be useful, especially because they cool down too quickly at the beginning and then slow down the cooling at the end. Intuitively it would be more meaningful the other way round and indeed the Method B4 shows good results. Nevertheless B4 needs way longer to converge than the linear cooling B1.

Table 2: Summary of optimization results for nonlinear cooling strategies

$\hat{\theta}$ ... Sample-mean of quantity θ
$\hat{\sigma}(\theta)$ ... Sample-deviation of quantity θ

Cooling Strategy	Convergence criterion	$\hat{N}$	$\hat{\beta}_{max}$	$\hat{\sigma}(\beta_{max})$	$\hat{f}_{min}$	$\hat{\sigma}(f_{min})$
B1	C1	90	36.0	2.6	3.131408622	5e-8
B1	C2	158	63.2	3.0	3.131408622	5e-8
B2	C1	12	54.90	0.7	3.160	0.058
B2	C2	109	84.73	1.7	3.132	0.010
B3	C1	35	37.8	2.2	3.132	0.010
B3	C2	116	68.2	1.4	3.131408622	5e-8
B4	C1	151	37.8	2.2	3.131408622	5e-8
B4	C2	219	77.1	4.6	3.131408622	5e-8

In figure 7 the cooling strategies B1-B4 are compared again. There also the course of the inverse temperature over the number of cooling-steps is visualized.

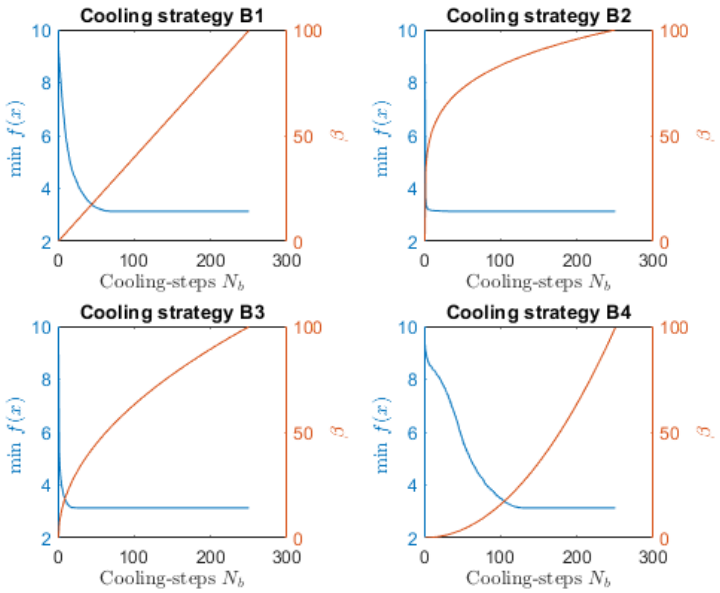

Figure 7: Comparison of nonlinear cooling strategies. The sample-mean of the found minima are plotted in dependence of the number of cooling-steps. Also the values of the inverse temperature β are plotted for each cooling step

Next we therefore consider only linear cooling strategies but with different values of $\Delta\beta$.

$$\text{B5}: \quad \beta_n = 0.1 + 0.1\, n \tag{3.5}$$
$$\text{B6}: \quad \beta_n = 0.1 + 0.5\, n \tag{3.6}$$
$$\text{B7}: \quad \beta_n = 0.1 + 1\, n \tag{3.7}$$
$$\text{B8}: \quad \beta_n = 0.1 + 3\, n \tag{3.8}$$

In figure 8 the results for the cooling strategies B5 to B6 are summarized. In table 3 the mean number of cooling-steps and the sample-means and sample-deviations for the two different convergence criteria are again summarized. Although all results are not bad, the cooling strategy B6 appears to work best, considering the quality of the results and the number of cooling-steps until converged.

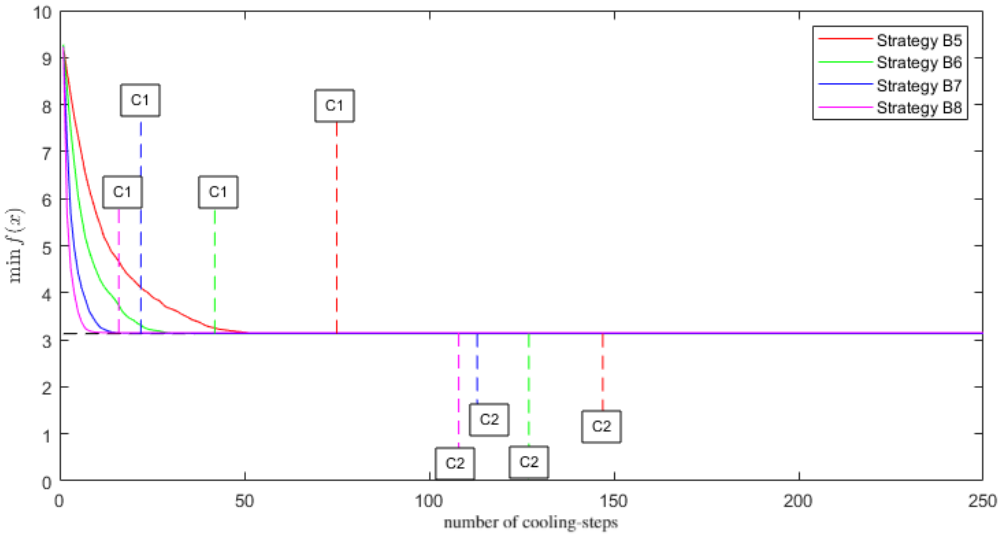

Figure 8: Mean values of the found minima in dependece of the number of cooling-steps for the linear cooling strategies

4 Experiment: Convergence in dependence of the number of points L

We now want to evaluate a rule of thumb on how many cooling-steps are required until the minimum can be considerd as good. Since it appears from fig 8 that the criterion C2 is to strict, we will weaken it and use the following criterion

C3 : The minimum is assumed to be good if the minimum is the same for the last 50 cooling-steps.

Table 3: Summary of optimization results for nonlinear cooling strategies

$\hat{\theta}$ ... Sample-mean of quantity θ
$\hat{\sigma}(\theta)$ ... Sample-deviation of quantity θ

Cooling Strategy	Convergence criterion	$\hat{N}$	$\hat{\beta}_{max}$	$\hat{\sigma}(\beta_{max})$	$\hat{f}_{min}$	$\hat{\sigma}(f_{min})$
B5	C1	74	37.1	2.4	3.131408622	0
B5	C2	147	73.4	3.1	3.131408622	0
B6	C1	42	41.90	3.7	3.1322	0.0082
B6	C2	126	126.4	4.3	3.131408622	0
B7	C1	21	54.9	3.7	3.136	0.022
B7	C2	112	282.1	9.1	3.135	0.020
B8	C1	15	79.1	4.4	3.150	0.046
B8	C2	107	535.8	8.8	3.150	0.046

We now generate random pointsets of different size and try to find a minimum with the Simulated Annealing algorithm. For each number of points L we draw 100 random sets. The number N_S of sweeps per cooling step will be chosen in such a way that the rule of thumb $N_S \approx L^2$ is fulfilled for the largest pointset. For pointsets with lower L, this might lead ta an unnecessarily large number of total sweeps, but it should not have a great influence on the rate of convergence in terms of the number of cooling-steps. At all, it should actually improved the convergence, because there is more time to move in configuration space for each β value. We fix

$$N_s = 50^2 = 2500 \tag{4.1}$$

As proposal method we use the method P2 and we consider the linear cooling strategy B6. The result is shown in figure 9
One recognizes a linear behavior for the increase of the number of cooling-steps required to find a good minimum. A linear fit gives

$$N_{C3} = 2.43L + 35.48 \tag{4.2}$$

as the line of linear regression. This can be formulated as a rule of thumb

$$N_{C3} \approx 2.5L + 35 \tag{4.3}$$

for the number of cooling-steps that have to be provided at least in order to find a good minimum. Note that we provided a sufficiently high value for N_s. to obtain this rule of thumb.

5 Experiment: Solving a Problem for L = 100 points

At the end we now apply the algorithm at a problem with $L = 100$ points. As proposal method we use the method P2 and we consider the linear cooling strategy B6. As criterion of

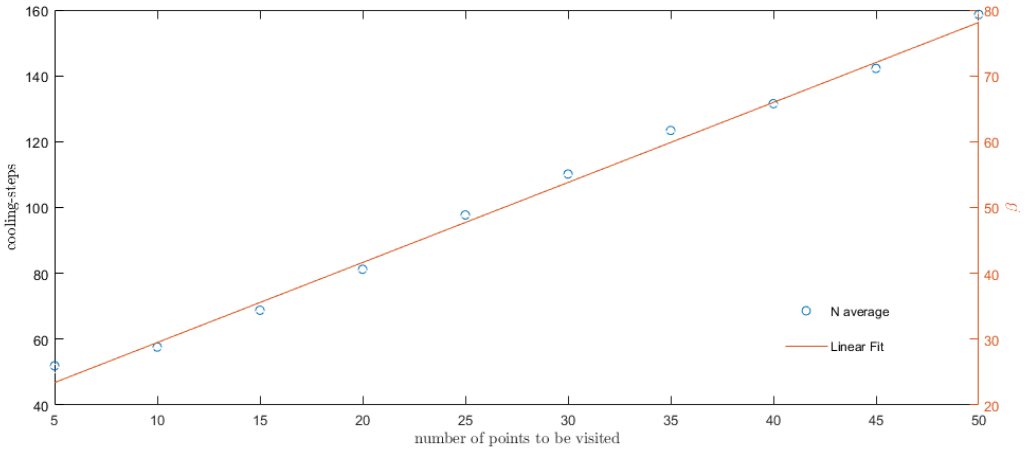

Figure 9: Dependence of the convergence point on the number of points L. The left y axis shows the average number of required cooling-steps while the right axis points out average inverse temperature β_{max} where convergence is achieved

convergence we use C3. For the value N_s of sweeps per cooling step we fix

$$N_s = 100^2 = 10000 \tag{5.1}$$

and from our rule of thumb we assume, that $N_b = N_{C3}(100) \approx 285 \approx 300$ cooling-steps should be sufficient to get a good minimum. Figure 10 shows the minimization process. The found minimum is reached after 226 cooling-steps, which corresponds to $\beta = 112.6$. Figure 11 shows some of the configuration found during the optimization process.

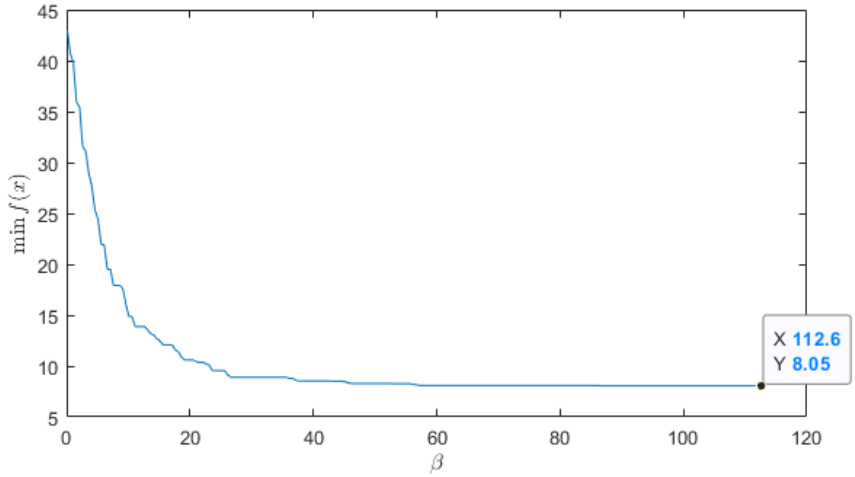

Figure 10: Course of the best minimum found over the inverse temperature

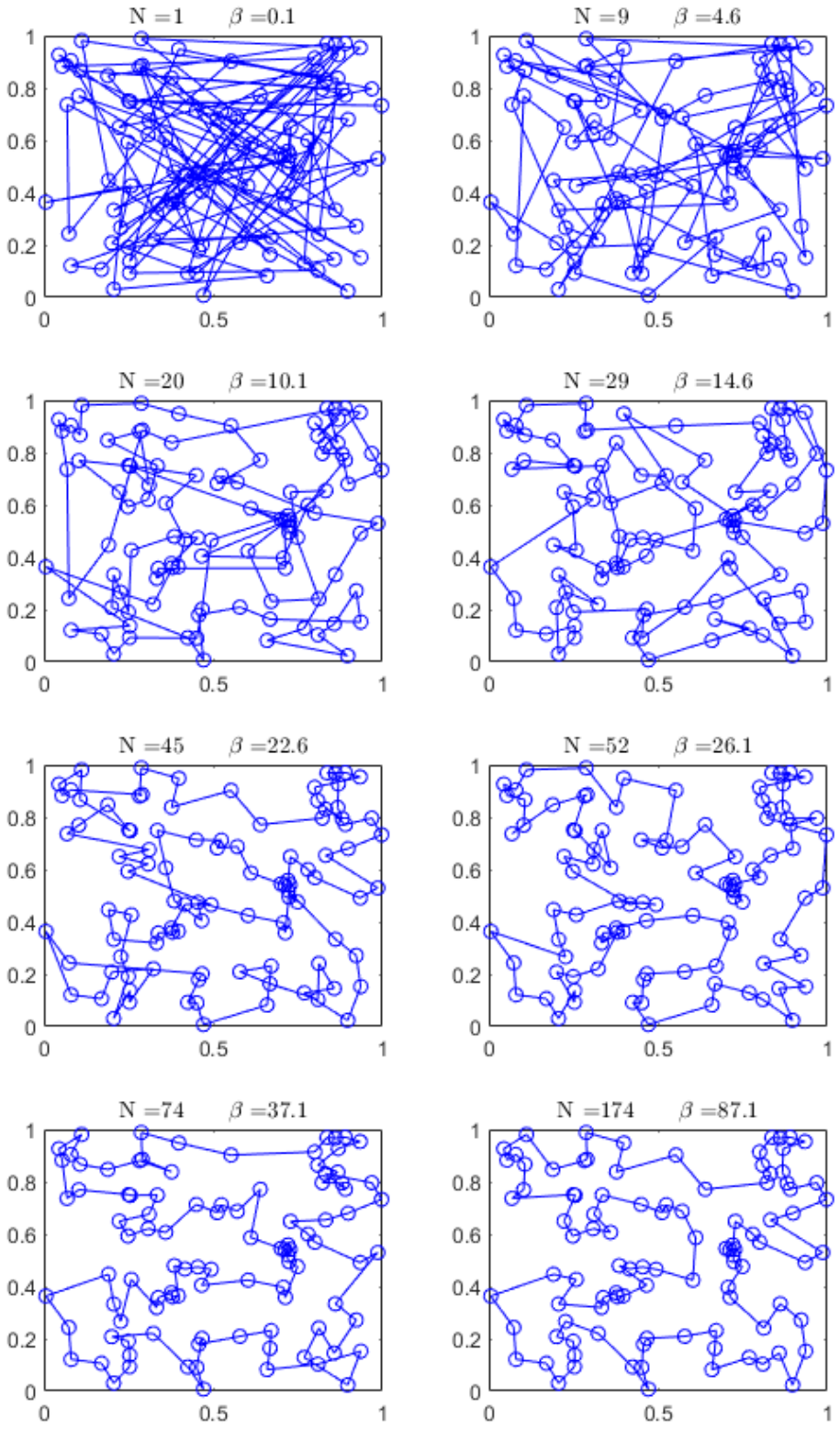

Figure 11: Different intermediate configurations during the optimization prozess

6 Summary

We have investigated different proposal methods and cooling strategies for the Classical Simulated Annealing algorithm. Proposal method P2 with a linear cooling strategy worked out to give the best results. Different criteria for evaluating the quality of the minimum have been discussed. There are still a lot more things that could be tested, such as different probability distributions for the acceptance probability, tempering or adaptive strategies for cooling, etc. We found that $0.1 < \beta_{min} < 1$ gives a good value for the minimal value of the inverse temperature. We derived a rule of thumb to have an approximate apriory estimate on how to choose the number of cooling-steps and number of sweeps per cooling step in order to obtain good results. However for the rule to hold the use of proposal method P2 and a linear cooling strategy are required.

7 Code

Here the most important code snippets are shown. The code is written in the rather young programming language JULIA.

7.1 Optimization function

```
function simulatedAnnealing!(
    data::Dict{Symbol, Array{Float64,1}},
    inital_state::T,
    beta_fnc::Function,
    cost_fnc::Function,
    proposal_fnc::Function,
    params::Dict{Symbol, Real},
    ) where T

    # parse data and params
    parseParams!(params)
    parseData!(data)

    # init required variables
    best_state = inital_state
    minimal_cost = cost_fnc(inital_state)

    current_state = inital_state
    current_cost = minimal_cost

    N = 0
    for i_rm = 1:params[:N_rm]          # push NaN where running mean
        push!(data[:rm_a_rate], NaN)    # can not be calculated
```

```
end

# simulate annealing
stagnation_count = 0
while N < params[:N_max]

    beta = beta_fnc(N)
    a_rate = 0
    stagnation_count +=1
    for k in 1:params[:N_s]
        trial_state = proposal_fnc(current_state)
        trial_cost = cost_fnc(trial_state)

        delta_cost = trial_cost - current_cost
        p = min(1. , exp(-beta*delta_cost))
        if p == 1.
            current_state = trial_state
            current_cost = trial_cost
            a_rate +=1

            if current_cost < minimal_cost
                best_state = current_state
                minimal_cost = current_cost
                stagnation_count = 0

            end

        elseif rand() <= p
            current_state = trial_state
            current_cost = trial_cost
            a_rate +=1
        end

    end

    # evaluate rate of acceptance
    a_rate = a_rate/params[:N_s]
    if length(data[:a_rate])>params[:N_rm]
        rm_a = sum(data[:a_rate][end:-1:(end-params[:N_rm]+1)])
        rm_a = rm_a / params[:N_rm]
        push!(data[:rm_a_rate], rm_a )
    end
    # store data
    push!(data[:beta], beta)
```

```
        push!(data[:a_rate], a_rate)
        push!(data[:c_min], minimal_cost)

        # check criteria of convergence
        N = N + 1
        if data[:rm_a_rate][end] < params[:rm_a_min]
            break
        end

        if stagnation_count > params[:max_stagnation]
            break
        end
    end

    return best_state
end
```

7.2 Proposal functions

```
import StatsBase

function genProposal1(state::Array{Float64,2})

    proposal = copy(state)
    n, = size(state)

    #swap states
    idx1, idx2 = StatsBase.sample(2:(n-1), 2, replace=false)
    proposal[[idx1, idx2],:] .= proposal[[idx2, idx1],:]

    return proposal
end

function genProposal2(state::Array{Float64,2})

    proposal = copy(state)
    n, = size(state)

    #reverse states
    idx1, idx2 = sort(StatsBase.sample(2:(n-1), 2, replace=false))
    proposal[idx1 : idx2, :] .= reverse(proposal[idx1:idx2,:], dims = 1)

    return proposal
end
```

```
function genProposal3(state::Array{Float64,2})

    proposal = copy(state)
    n, = size(state)

    idx1, idx2, idx3 = sort(StatsBase.sample(2:n, 3, replace=false))
    proposal = vcat(
        proposal[1:idx1-1,:],
        proposal[idx2:(idx3-1),:],
        proposal[idx1:(idx2-1),:],
        proposal[idx3:end,:]
        )

    return proposal
end

function genProposal4(state::Array{Float64,2})

    proposal = copy(state[1:end-1,:])
    n, = size(state)

    idx1 = rand(1:n-1)
    if idx1 == n-1
        proposal[[1, idx1],:] .= proposal[[idx1, 1],:]
    else
        proposal[[idx1, idx1+1],:] .= proposal[[idx1+1, idx1],:]
    end
    proposal = vcat(proposal, transpose(proposal[1,:]))

    return proposal
end
```

7.3 Cost function

```
function costFunction(state::Array{Float64,2})

    cost = diff(state, dims = 1).^2
    cost = sqrt.(sum(cost, dims = 2))
    cost = sum(cost)

    return cost
end
```

7.4 Diverse functions

```
function parseData!(data::Dict{Symbol, Array{Float64,1}})

    if !haskey(data, :a_rate)
        data[:a_rate] = Array{Float64,1}()
    end
    if !haskey(data, :rm_a_rate)
        data[:rm_a_rate] = Array{Float64,1}()
    end
    if !haskey(data, :E_min)
        data[:c_min] = Array{Float64,1}()
    end
    if !haskey(data, :beta)
        data[:beta] = Array{Float64,1}()
    end

    return
end

function parseParams!(params::Dict{Symbol, Real})

    if haskey(params, :N_s)
        if params[:N_s] < 100
            error("N_s must be at least 100")
        end
    else
        params[:N_s] = 100
    end
    if !haskey(params, :N_max)
        params[:N_max] = 1e6
    end
    if !haskey(params, :N_rm)
        params[:N_rm] = 10
    end
    if !haskey(params, :rm_a_min)
        params[:rm_a_min] = 0.001
    end
    if !haskey(params, :max_stagnation)
        params[:max_stagnation] = 100
    end

    return
end
```