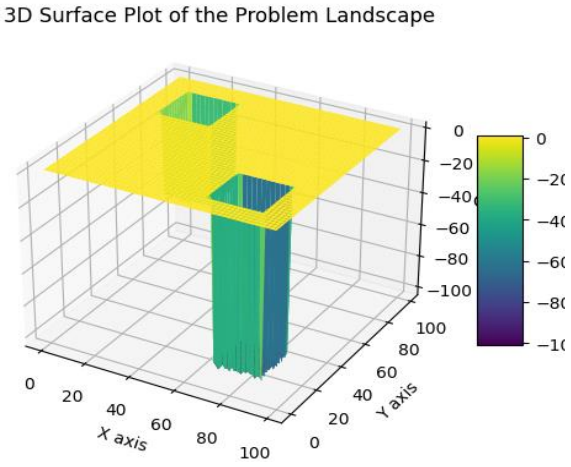
Computer Programming Project – Andrea Vento

First, I generated the data using my ID (3224274) as a seed for the ‘generate_data’ function. Then, i respresented the landscape in 3d to better understand the problem.



TASK 1

I implemented the class for the problem, defined in a (nxn) space. The problem is initialized at a random state (i,j) representing a point in the grid. The move proposal is designed to propose a movement of step 1 along 4 diagonal directions, even allowing border jumping if needed (modulo n operation).

When trying to scale the size of the problem (increasing n, as shown later), I noticed the algorithm struggled to explore the solution space in reasonable time given the standard move proposal (propose_move_standard, one step in the four directions), so also I tried out a different move proposal: propose_move_adaptive, which randomly picks a step everytime it proposes a new move. The purpose of the introduction of an alternative move is to further analyze the problem landscape and to have a comparison term for our standard move.

I constructed the alternative move to have different useful properties:

• no diagoal terms (the proposed step is strictly greater than one);

• connectedness (being randomized, it is possible to get to any configuration from any other, given enough steps, with non-zero probability);

• aperiodicity (the move is pseudorandom);

• symmetry (there is the same probability of proposing a move from i -> j than j -> i).

Furthermore, the compute_delta_cost method has been optimized to reduce the number of operations needed to compute the difference between the new and old cost (without creating a copy of the problem).

After implementing the ‘Probl’ class, I pasted the simulated annealing function we used during class, adopting some changes:

• Implemented a Convergence Timer and Flag (a Boolean value initialized at False), so that, if the Optimization found the desired value (or range, as we will see), the Convergence Timer stops and the Flag turns into True, indicating that the attempt was successful. (Notice that it could've been inserted a break after convergence but I wanted to gather data about the whole process even after convergence for successive tasks)

• Initialized two lists, ‘min_seq’ and ‘acc_rate_seq’, to record the data about the minimum value and acceptance rates found for each annealing step
• Inserted an ‘if statement’ at the beginning of each ‘Mcmc step’ to be able to chose which move to perform at each iteration
• Returned 5 information: • convergence time; • best cost; • min_seq (a list containing the values of the minimum value found at each annealing step); • success (Boolean variable indicating whether the optimization was successful; • beta list (not quite necessary, but useful when plotting and experimenting).
Then, I implemented the ‘accept’ function, returning True or False based on the Metropolis acceptance rule (more on that later in the analysis).
Lastly, I also inserted the ‘greedy’ optimization algorithm for comparison purposes.

TASK 2

The modulo operation defined and implemented in the propose_move_standard method is convenient since it allows our optimisation algorightm to jump over borders. This property semplifies the problem as it makes the 'proposal matrix' (the moves proposed for each instance of the problem) symmetric, which allows as to use the 'simplified Metropolis Rule' for acceptance.
Our proposal matrix (C) is a stochastic matrix, therefore all columns should sum up to 1, we always propose something according to the stated probabilities.
Therefore, if we decided not to allow the border jumping, our proposals wouldn't be symmetric, for example: if the current state (self.i,self.j) is at the right border, we would have a 1/3 probability of moving in any of the three allowed directions but, if the left (Ovest) move is accepted, there will only be a 1/4 probability of proposing the reversed move. In this case, we couldn't use the simplified acceptance rule but the more general one instead. We then would need to change both the 'propose_move' and the 'accept' function accordingly:
• propose_move would be changed from 'np.random.choice([self.i-1, self.i+1]) % n' to 'np.random.choice([max(0,self.i-1), min(self.n-1, self.i+1)])' in order to always stay within the borders of the grid.
• the ‘accept’ function would need to be changed, transitioning from the ‘simplified Metropolis rule’:

$$A\left(x \to x'\right) = \min\left(1, e^{-\beta \Delta c\left(x \to x'\right)}\right)$$

To the general one that guarantees to satisfy the detailed balance condition even in non-symmetrical proposal cases:

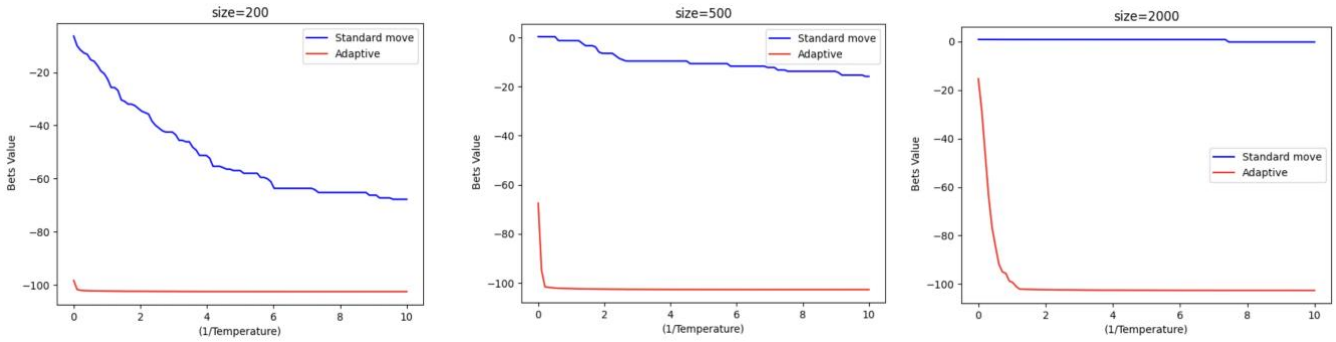
$$A_{ij} = \min\left(1, \frac{C_{ji}\rho_i}{C_{ij}\rho_j}\right)$$

Continue…task 2

To study the performance of the algorithm, I created an auxiliary function ‘Experiment’.
All the following experiments are designed perform 100 trials and compute the average of the results found, using 100 annealing steps, 200 ‘MCMC’ steps, and a range of problem sizes in this set: [100, 200, 500, 1000, 2000, 3500, 5000].
For graphical reasons, I will only include in this document the graphs related to sizes: [200,500,2000].

TASK 3

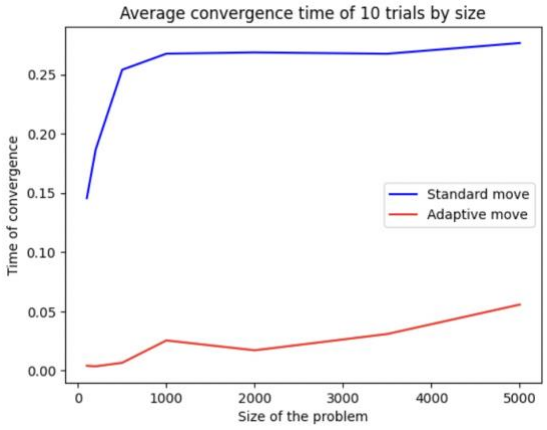
I studied the performance of the algorithm in terms of 'Best Value' for each step of the Annealing process. I then performed this analysis for different sizes of the problem to capture differences related to the scale.

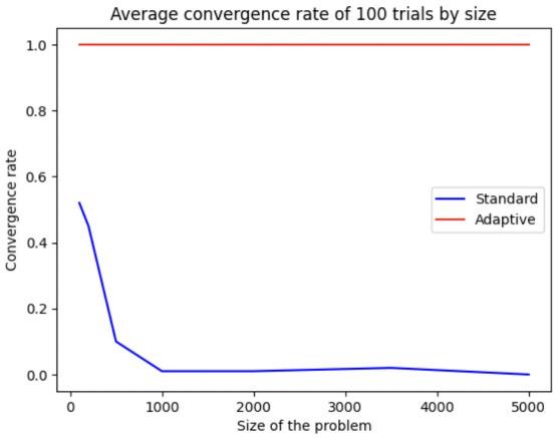


As the graphs show, the higher the value of beta (lower temperature), the lower is the value of the best_cost found by our algorithm(on average). Therefore, to some degree, our annealing procedure is helping us in finding the minimum as it allows to escape local minima by accepting moves that increase the cost with non-zero probability.
Indeed, if we plot the cost found at each iteration for an individual problem, we'll see that it will be allowed to increase according to the Metropolis acceptance rule.
This is the core idea of the Simulated Annealing that helps our Optimization Algorithm to escape local minima allowing to explore areas of the solution space that would otherwise remain unknown using a naive Greedy strategy.
Despite that, I found that our algorithm struggles to consistently find the minimum for problems with greater sizes: this could be indicating that, for the given amount of (Annealing*MCMC) steps, our algorithm is only able to explore a limited area of the solution space using its standard move (the length of the step is not proportional to the size of the problem, therefore the algorithm will explore a progressively lower portion of the solution space as the size of the problem increases). This could explain why the average value of the minimum found keeps increasing as the size of the problem increases.
Furthermore, the adaptive move is more effective in finding the minimum as the length of the step is proportional to 'n', (np.random.randint(1,n)), performing significantly better for greater sizes of the problem in terms of minimum value found.

TASK 4

The following graphs will help to analyze the performance of the algorithm in terms of convergence speed and success.
I noticed that the algorithm struggled to find the actual minimum (around -100 for any ‘n’), therefore, to understand whether the attempt was close to the optimal value, I considered the optimization successful if the best value found was in the neighbors of the optimal one (radius 3).





As expected, the convergence time increases with the size of the problem.
This behavior could be due to two main reasons:

- the increased dimensions of the grid increase the computation needed to perform the same steps.
- the algorithm finds it difficult to find the optimal range, hence the timer doesn’t stop until the complete number of steps (Annealing x MCMC) has been performed.

The graph related to the convergence rate provides key information to clarify this doubt: the decreasing trend exhibited by the convergence rate indicate that, as expected, our algorithm becomes less and less effective at finding the optimal solutions range as the size of the problem increases (using the standard move).
Moreover, our algorithm is optimized to perform (roughly) the same number of operations for every problem, since we only initialize it at the beginning of the optimization.
It follows that, as the size of the problem becomes larger, the computational time increases because progressively more steps are being performed prior to the recording of the final time.
Consequently, for the same reasons, the success rate will decrease as the explored portion of the solution space will became smaller and smaller.
Despite showing similar qualitative behavior, the alternative move proposal performs better both in terms on convergence time and convergence success rate thanks to the possibility to randomly ‘jump’ from a side of the grid to the other.

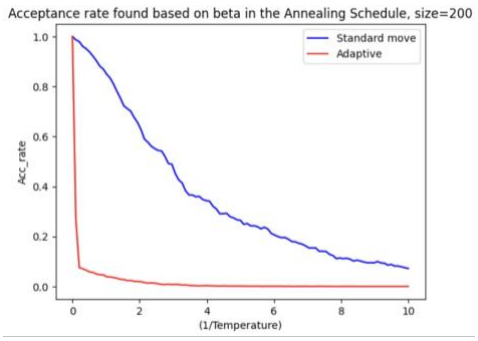
TASK 5

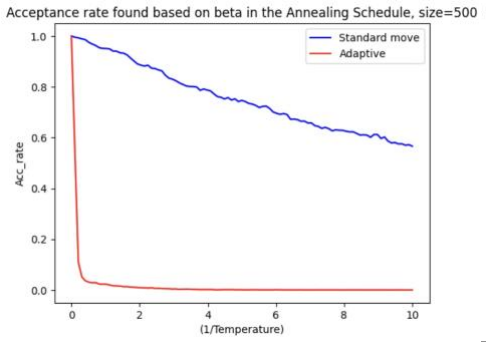
The acceptance probability rate in the Simulated Annealing process is one of the main metrics to analyze to understand the behavior and the effectiveness of our algorithm. There have been designed some rules of thumb that could be interpret the results we get from our experiments:

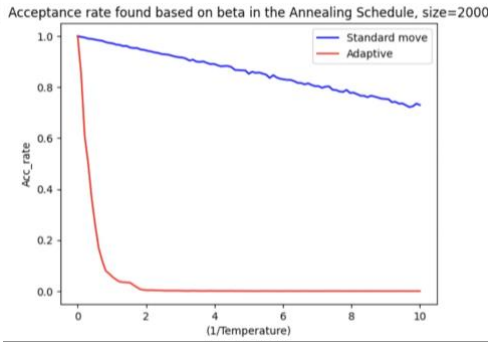
• Acceptance rate too high until the end → time wasted in roaming around instead of optimizing (exception for equal-cost states...)

• Acceptance rate too low too early → the system is (nearly) stuck in a local minimum

I computed and plotted the acceptance rate probability registered at each annealing step for 100 trials. I then repeated the experiment for different sizes of the problem to capture differences due to the scale of the problem.



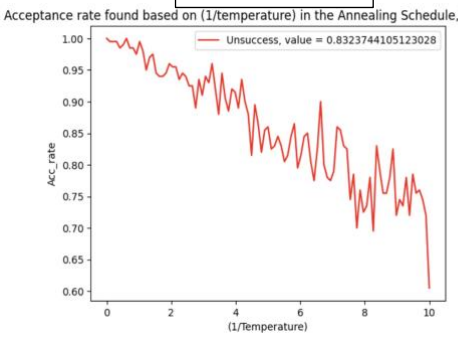




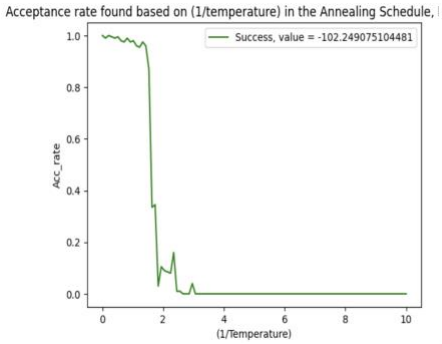
As expected, lower values of the acceptance rate correspond to lower values of the temperature, on average.

Although it seems that the acceptance rate gracefully decreases while increasing beta, what truly happens is slightly different: a different perspective emerges by looking at the graphs representing the acceptance rates based on beta for individual cases.

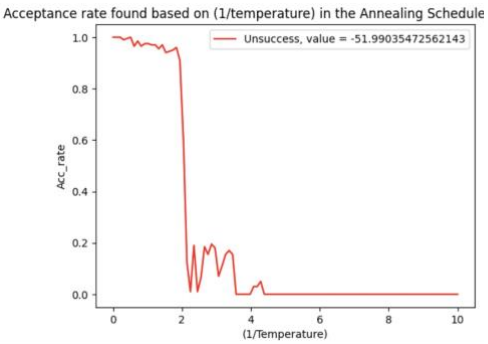
Example 1



Example 2



Example 3



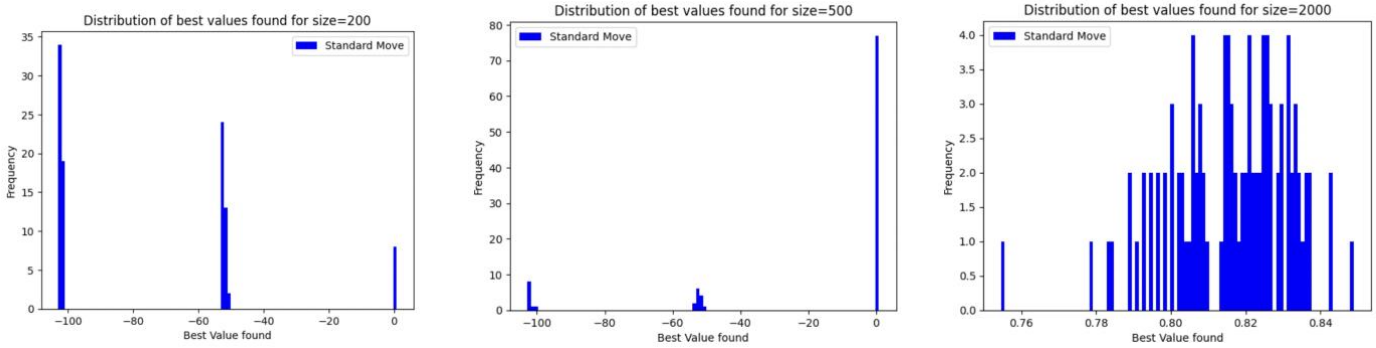
It now becomes clear what is truly happening at each trial of the optimization: the acceptance rate exhibits a deep fall whenever our algorithm is able to find a new configuration with a significantly lower cost.

Since our 'accept' is highly depending on the delta cost, once we accept the new best cost (which will be significanly lower that the previous one), we will observe a drastic decrease in the acceptance rate. This is because the algorithm is performing an MCMC 'walk' in a region with an extremely high variance of costs (the border of two regions), many more proposals will be rejected with respect to the regular areas of the landscape, as highly positive delta costs will be found.

Following this drastic drop, the acceptance rate recorded at the successive values of beta will be lower for two main factors:
• the regular ‘Beta effect’: increasing the value of beta, the ‘accept’ function will naturally reject more proposals (this happens in every case);
• a possible increase in the variance of the delta cost in the ‘lower cost’ region (the two (sub)optimal regions), with respect to the regular cost region (around 0).

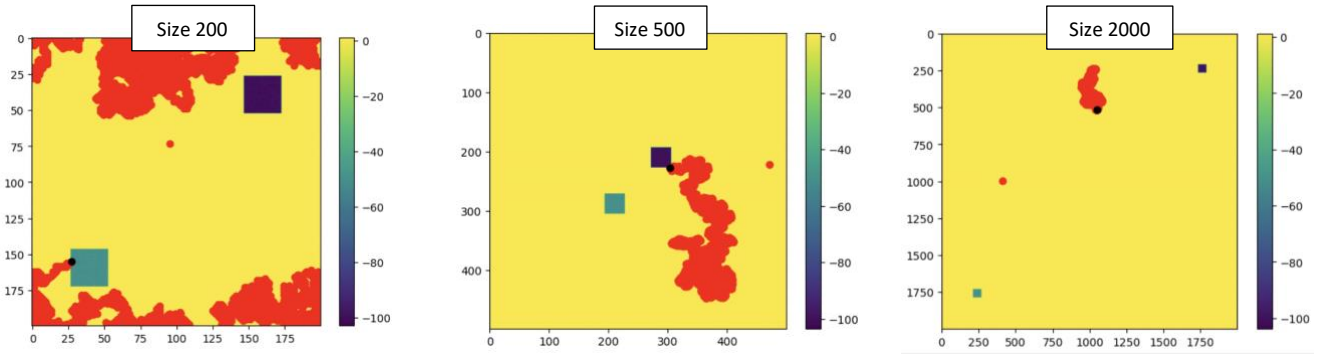
In particular, running the experiment multiple times, I becomes noticeable that this behavior is shown in 2 different cases: • the optimal range is found (around -100)
• a suboptimal range is found (around -50).

OBSERVATION



The distribution of the Best Values found by the ‘Simann Function’ (100 trials) follows a trimodal distribution, which is coherent with the three examples above.
Therefore, we vertical fall in the
In particular, as discussed earlier, our algorithm will struggle to find a (sub)obptimal range for problems of greater sizes, indicating that the acceptance rate will smoothly decrease without drastic falls (simply due to the increasing value beta which will progressively reduce the probability of accepting a move). On the other hand, if the optimization is able to find a (sub)optimal solution, the acceptance rate will have the ‘falling’ behavior described above.
All the gathered information suggests a specific landscape for our problem, with several peculiarities.
• The trimodal distribution suggests 3 different outcomes for our optimization: each exhibiting a considerable probability of happening for smaller instances of the problem. However, increasing the size of the landscape, the Best Value is progressively more probably found around 0, indicating that the three ‘areas’ of the solution space don’t always increase proportionally to the size of the problem.
• The drastic drop in the acceptance rate in case of (sub)successful attempts suggests that, given our standard move, it is difficult to keep exploring other portions of the solution space once our algorithm finds a (sub)optimal area (this effect in highlighted even more for greater sizes of the problem, in which it is more probable to find a (sub)optimal region later in the annealing process, resulting in a (nearly) zero probability of escaping due to the high value of beta).

TASK 6



Plotting the optimization for a fixed value of n, the aforehand described properties become more intuitive.
The (sub) optimal regions don't increase their total area proportionally to the problem, as expected. This explains why the convergence success rate decreases as the size of the problem increases.
Moreover, given our Standard Move, the proportion of the area of the solution space our algorithm is able to cover gets smaller and smaller.
These observations are coherent with the struggles found to effectively optimize our problem for greater instances of the data.
In addition, the path towards the minimum is not smooth, the (sub) optimal regions represent a vertical fall with respect to the 'yellow' area.
These properties not only explain the drastic drop in the acceptance rate once we find a minimum, but also the subsequent lower acceptance rates found after the drop: if the delta cost was similar between the (sub)optimal regions and the 'yellow' ones, we would observe the acceptance rate to get back to the its regular trend (slightly downwards sloping) once the algorithm gets over the 'borders' of the regions.
However, this doesn't happen, indicating that the variance of the delta cost il greater in the (sub)optimal regions.

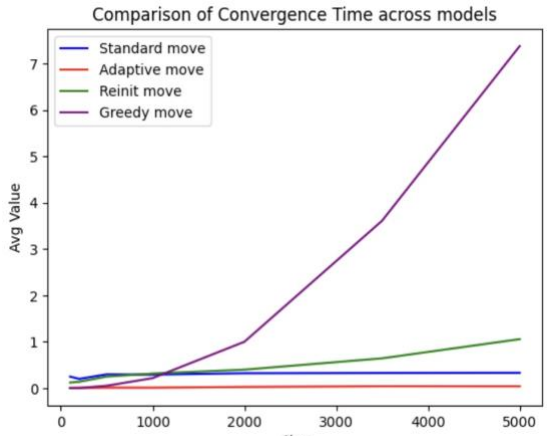
A suggested solution would be to increase the number of MCMC (or Annealing) steps to give our function more opportunities to further explore the solution space. Despite being theoretically correct, this solution would be too demanding from a computational perspective (we might as well check every point in the grid performing O(n^2) checks).
An alternative idea would consist in choosing lower values of beta, increasing the overall acceptance rate, and resulting in a greater covered area of the solution space. This approach will show too high acceptance rates even at the end of the optimization, basically performing a random walk on the landscape.

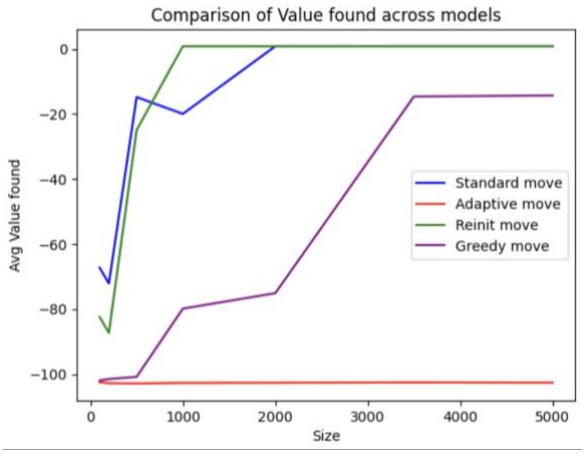
TASK 7

Having considered all the information above, we can conclude our Simulated Annealing algorithm could be improved in many areas, in order to adapt to the specific problem, we are trying to tackle.
For sure it is better than a basic greedy strategy, which would be easily trapped in a local minimum if not reinitialized.
Running our algorithm for greater sizes of the problem we encountered some major problems:
• the limited portion of the solution space we managed to cover using our Standard Move of 1 step
• the high correlation of our Best Value found to the initial point (initial points closer to the (sub)optimal regions result in higher probability of finding a (sub)optimal region)
• the higher variance in the delta costs in the (sub)optimal regions (we need to find a way to escape even close local minima if we intend to find the global minimum)

In order to solve those issues there can be adopted different strategies:
• setting the step of the algorithm to be a function of the size of the problem (ensuring that all necessary Simulated Annealing requirements are met)
• initializing the algorithm multiple times in order to further explore the solution space
• given the prior knowledge we have on the algorithm, adopted a naïve greedy approach consisting in choosing random points on the grid and returning the best one (although it may seem silly, it could be a good trade-off between computational complexity and accuracy in this case.

I implemented all these 4 different approaches, and tested them to compare them





Overall, the Adaptive Move method performs better both from an optimization success rate and from a convergence time perspective.
Surprisingly, the Naïve greedy method outperforms both the Standard and Reinitialization Methods in finding the minimum value, on average.
On the other hand, this additional precision comes with a price, the algorithm will need to check more points, on average, to find an optimal range, resulting in an higher convergence time.

In conclusion, despite its drawbacks, the simulated annealing problem, combined with an appropriate step choice (adaptive move), will result in the best trade off between precision and convergence time.