# Simulated Annealing

## Introduction

This algorithm was used in seeking for the best path from a node to any of the base stations. This was done by iteratively checking the absolute cost of the random paths generated up to a point when the following 2 conditions have been met:

1. The temperature cools to the lowest set temperature for the algorithm
2. There has been no considerable improvement in the absolute cost over the course of several iterations.

For purposes of exploration of the search space, the probability of the algorithm picking a worse off solution is high at the initial phases (when the temperature is high) but in later iterations, better solutions are highly considered.

## Process Flow

Routing Table Creation: The algorithm starts with the creation of a connectivity graph. This is done by calculating the distances and the bandwidth connectivity between the nodes. This graph will be essential in creating random routes that have end-to-end connectivity at a fast rate. This process is implemented by the `calculate\_route\_costs ()` function which is fed in the x\_y\_data that contains the x, y positions of the sensors. A 2D array with the calculated bandwidth information is returned.

Initial Route Generation: A random route in created and to ensure validity, the routing table created in the previous step is utilized. The path generated could be of varying length and could be leading to any of the base stations. This is done concurrently with the route absolute cost calculation. This is done in the `generate\_route ()` function which takes in the origin node and the routing table data. The absolute cost calculation is done in the `bandwidth\_latency\_calculation ()` function with takes in the generated route. The route with its corresponding cost is returned from the whole process.

Variable Initialization: This is a stage where variables like the ‘starting temperature’, ‘final temperature’, ‘cooling rate’, and ‘improvement checker count’ are initialized. These are the variables that control the functioning of the main function called `simulated\_annealing ()`. Their functions are defined below:

1. starting temperature: the start temperature defined for the simulated annealing algorithm
2. final temperature’: the final temperature defined for the simulated annealing algorithm
3. cooling rate: the temperature cooling rate for the algorithm for the algorithm
4. improvement checker count: the number of iterations to go through before checking if there is any sensible improvement in the path quality

Recursive Exploration: The algorithm then gets into the core functionality of generating other possible solutions and checking if they are better than the current best solution. Initially, the first solution is set as the best solution and thereafter, any solution that is better than that is used as the best solution for forthcoming iterations. Meanwhile, the cooling of the temperature happens at the set rate. During the initial iterations, the probability of a worse of solution being used as an acceptable solution is high since the high temperature is used in calculating the probability of a path being picked. As the temperature goes down, the influence of temperature in determining whether a solution will be accepted is low. The formulae used in determining whether a solution is picked is as follows:

delta = current path cost – best path cost

probability = exponential (-delta / current temperature)

Just like in a roulette wheel, a random number is generated and the side in which is falls in determines the solution’s acceptance. In this case, if the number is less than the calculated probability, the worse off solution is accepted.

This loop goes on until the lowest temperature is reached. However, I a situation where several iterations have been done but there is no significant improvement, the loop is broken and the solution accepted for purposes of saving time and computation resources.

The main function returns the generated solutions that is categorized as ‘best routes’ and ‘possible routes’ as is a standard with all the other algorithm implementations for visualization purposes. The solutions variable containing the best path in the format required by the solutions file is also returned.

The algorithm has a good performance for two unique implementations:

1. The means of constructing the new paths without having to test invalid paths
2. The performance improvement checker which checks the significance of the improvement in the course of several iterations.

In the future, we would like to add of some extra improvement to the algorithm. This is variation of the cooling rate over time. Since the algorithm is more exploratory during the initial phases (can accept worse off solutions), we will have a slow cooling rate for proper exploration to occur but progressively, increase the cooling rate as we settle to a certain range of solutions just to save of time since at that point the algorithm rarely accepts anything that is worse off.