Travelling Salesman Problem with Simulated Anealling

Lew Jun Long
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
junlong7368@gmail.com

Ng Zer Dan
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
zerdan6620@gmail.com

Felix Ting Yu Hong
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
felixtingyh@gmail.com

William Chuah Eng Keat
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
williamchuah01@gmail.com

Abstract—**Solving Traveling Salesman Problem (TSP) using the Simulated Annealing (SA) algorithm has been widely used in logistic companies for their delivery systems. It is because the algorithm has better performance regarding its ability to avoid trapping in the local optima. In this paper, we have chosen 10 locations from Kuala Lumpur and Selangor area in Malaysia to find the best route based on the SA algorithm. Besides, we also change the parameters of SA algorithm such as initial temperature, cooling rate, and number of cities. NetBeans will be used as our integrated development environment (IDE) for JAVA programming to implement the SA algorithm. The generated result will be compared with a route mapping application, MyRouteOnline, using the data from Google Map. The experimental results show the difference between the modified SA algorithm and MyRouteOnline when solving the TSP problem.**

Keywords—Traveling Salesman Problem (TSP), Simulated Annealing (SA) algorithm

I. INTRODUCTION

The logistics industry contains a new development trend worldwide as electronic commerce has grown up year after year. Especially during the COVID-19 pandemic, many retail stores have performed digital transformation to overcome the economic repercussions of the COVID-19 crisis. It creates significant pressure on the logistics industry to satisfy the demand for parcel delivery. Finding the best and the shortest route has now become a challenge to increase the effectiveness and efficiency of the logistics industry. As such, with the inconsistent distance and costs among many destinations, the research and analysis on Travelling Salesman Problem (TSP) had to be done to improve the delivery service.

TSP is a well-known problem in the artificial intelligence industry to choose the best way that requires the least distance and cost among various paths. It is classified as an NP-hard problem as it cannot be solved within a short time, although it seems easy to explain, and the degree of difficulty will increase when more destinations are added to the problem.

In this paper, our group will be solving the TSP using the Simulated Annealing (SA) algorithm [1] to find out the best path among the 10 destinations. The primary purpose of implementing this algorithm is its capability to avoid getting stuck in the local optima and provide the optimal solution. Besides, it can come out with a better result even if the assigned time is short in the searching process as the cooling schedule is controllable.

Google Map is known as a navigation application. It uses the A* algorithm, an advanced version of the Breadth-First Search algorithm, to solve the TSP. MyRouteOnline application allows users to input multiple destinations and calculate the best route based on the result of Google Map. Hence, the comparison between the MyRouteOnline application and our final solutions will be conducted to ensure the accuracy of our desired algorithm.

This research paper will be divided into several sections. Section 2 will be the literature review which discusses the similar work with the same problem or the same algorithm. Section 3 will talk about the implemented methodologies in this paper. Material and method which include the used algorithm will be explained in section 4. Section 5 will show the results of our work, while the conclusion will be made in section 6.

II. LITERATURE REVIEW

Simulated annealing was applied to generating itineraries for travelers to reduce the time and effort required to plans trips. In this study, enhanced simulated annealing algorithm, a combination of greedy algorithm and simulated algorithm was used to generate itineraries. It was concluded that the enhanced simulated algorithm has a greater efficiency than conventional simulated annealing algorithm [2].

A study discussed on addressing the slow speed in obtaining the solution for conventional simulated annealing algorithm and proposed using tabu search strategy and two different neighborhood generation approaches. An adaptive cooling schedule was also applied. The study showed significant improvement in the convergence speed comparing to conventional simulated annealing algorithm [3].

Hosam and Ashraf [4] have conducted a research comparing the performance among SA, Genetic Algorithm (GA), and Ant Colony Optimization (ACO) in the TSP problem. All the algorithms were performed using the same platform, JAVA programming. The experimental results were reported as SA performs the fastest execution and ACO produces the shortest distances in solving the TSP problem.

A research paper on maximizing capital growth was done to illustrate the use of Simulated Annealing algorithm to maximize capital growth. It involved importing real financial data and using the algorithm to for the best percentage of investment that is suitable to invest in a trade [5].

A journal proposed a metaheuris-tic algorithm that was made with Iterated Local Search. The goal was to create an agile algorithm that is capable of producing great solutions for TSP. While the program was simple, it still performed exceptionally. The results proved that the algorithm was excellent for getting substantial results with little compute duration [6].

III. METHODOLOGY

A. Dataset

Our data consists of 10 locations in Malaysia around Kuala Lumpur and Selangor area, it includes the exact Latitude and Longitude values of each location, which would be converted into kilometers in the Java code itself. Each of these locations were handpicked by our team members. The reason why these locations were picked were because of the high traffic in these areas. The coordinates from the dataset are retrieved from Google Maps directly, which would then be multiplied by one hundred and one to convert into kilometers to be used in the algorithm.

TABLE I. COORDINATES OF THE 10 LOCATIONS

Location name	Coordinates
Stadium Hoki MPK	3.0141512871618423, 101.41415026054891
Tengku Ampuan Rahimah Hospital	3.021587798205863, 101.43551995941914
Supercourts Badminton Centre	3.0358244712586555, 101.59271451562216
Thai Buddhist Chetawan Temple	3.101610029870564, 101.65359638473045
Sunway Lagoon	3.0726692970393037, 101.60489477253626
Merdeka Square	3.1490832343764548, 101.69328107813398
National Stadium Bukit Jalil	3.054864270262906, 101.69129494518741
Zoo Negara	3.2129258941053194, 101.75959200582993
Petronas Twin Towers	3.15775276738902, 101.71188572346719
Leisure Mall	3.0901753889539667, 101.74188439737426

B. Generating neighbour solution

In travelling salesman problem, the neighbour of a state is evaluated to achieve optimal solution where new states are produced through altering the initial state. The state is defined as a permuatation of the visited cities and the way of altering the initial state is by swaping any two of visted cities.

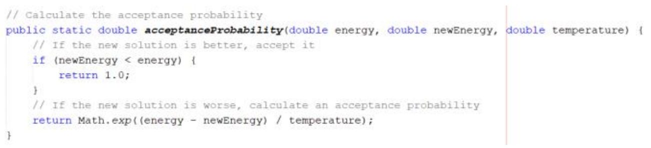
"Move" is the term to define the state altering process and different moves will result in generating different sets of neighbouring states. In the algorithm, the initial state and neighbour state will be compared in terms of the change in energy until the system is cooled. Minimal alterations are usually achieved in last state through these moves to gradually improve the solution.

C. Simulated Annealing

In each iteration of the algorithm, the temperature decreases by a set value and evaluates whether to accept the newly generated route as the next solution. The algorithm calculated the change in energy between the current best solution and the new solution. If the change is negative, meaning the next solution has a shorter path, the algorithm accepts the new solution. On the other hand, if the change is positive, meaning the next solution has a longer path, the algorithm runs an acceptance probability function. The probability of accepting the new solution is calculated with the following formula:

$$acceptanceProbability = e^{\frac{currentEnergy - newEnergy}{temperature}} \quad [1]$$

The calculated probability will be compared to a randomly generated number between 0 to 1. If it is greater than the randomly generated number, the new solution will be accepted even though it has a longer path. The algorithm will be able to avoid getting locked in local optimums. Otherwise, the current solution stays the same. During the early stages of execution, the temperature is high. The function returns a higher probability, which allows the algorithm to search for optimal paths in other areas rather than sticking to the initial direction.



```java
// Calculate the acceptance probability
public static double acceptanceProbability(double energy, double newEnergy, double temperature) {
    // If the new solution is better, accept it
    if (newEnergy < energy) {
        return 1.0;
    }
    // If the new solution is worse, calculate an acceptance probability
    return Math.exp((energy - newEnergy) / temperature);
}
```

Figure 1 Acceptance Probability Function

IV. MATERIALS AND METHODS

A. Algorithm Implementation

This section we illustrate how the algorithm is implemented into the Java code. The following is a summary pseudocode for how the simulated annealing algorithm performs to generate the optimal route.

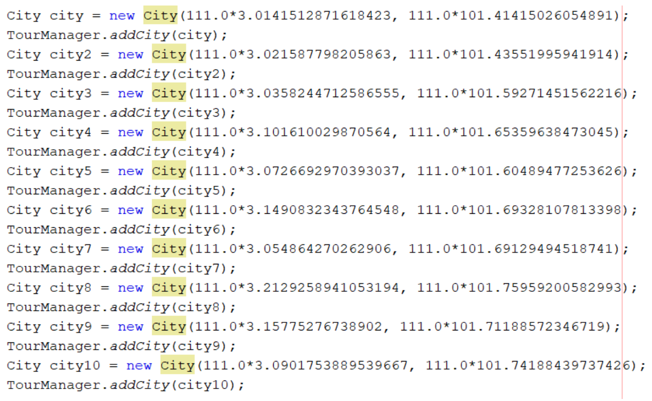
```
START
t = t0
cooling_rate = c
best_solution = Initial Tour
current_solution = Initial Tour
DOWHILE(t>1)
        new_solution = Generate neighbour solution
        current_energy=Length of current_solution
        neighbour_energy = Length of new_solution
        IF(acceptanceProbability>RandomNumber) THEN
                current_solution = new_solution
        IF (current_energy<best_energy) THEN
                best_solution = current_solution
        t = t-cooling_rate
END
```

where,

t0 = the initial temperature
c = the cooling rate set for the algorithm

V. Results and Discussion

A. Changing Intial Temperature

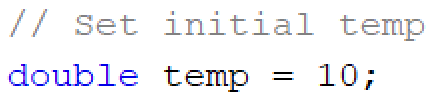


```
City city = new City(111.0*3.0141512871618423, 111.0*101.41415026054891);
TourManager.addCity(city);
City city2 = new City(111.0*3.021587798205863, 111.0*101.43551995941914);
TourManager.addCity(city2);
City city3 = new City(111.0*3.0358244712586555, 111.0*101.59271451562216);
TourManager.addCity(city3);
City city4 = new City(111.0*3.101610029870564, 111.0*101.65359638473045);
TourManager.addCity(city4);
City city5 = new City(111.0*3.0726692970393037, 111.0*101.60489477253626);
TourManager.addCity(city5);
City city6 = new City(111.0*3.1490832343764548, 111.0*101.69328107813398);
TourManager.addCity(city6);
City city7 = new City(111.0*3.05486427026290, 111.0*101.69129494518741);
TourManager.addCity(city7);
City city8 = new City(111.0*3.2129258941053194, 111.0*101.75959200582993);
TourManager.addCity(city8);
City city9 = new City(111.0*3.15775276738902, 111.0*101.71188572346719);
TourManager.addCity(city9);
City city10 = new City(111.0*3.0901753889539667, 111.0*101.74188439737426);
TourManager.addCity(city10);
```

Figure 2 Inserting and converting location coordinates in code

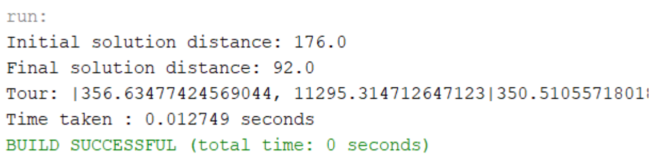
Before starting, we first insert the dataset into the code as shown in the figure above. At the same time, all the coordinate values are multiplied by 111 to convert to kilometers.

In Simulated Annealing, the initial temperature is arguably the most influential parameter for getting the best result, and also responsible for getting out of local optimums. So, we tested the algorithm with different temperatures with a fixed value for cooling rate of 0.0003 and investigate which temperature does the best.



```
// Set initial temp
double temp = 10;
```

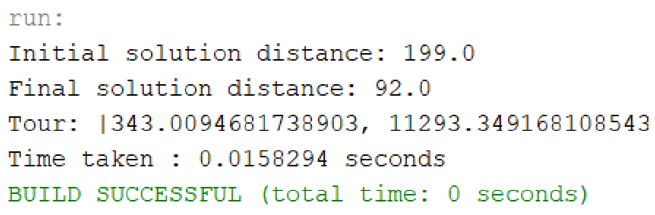
Figure 3 Setting initial temperature



```
run:
Initial solution distance: 176.0
Final solution distance: 92.0
Tour: |356.63477424569044, 11295.314712647123|350.5105571801
Time taken : 0.012749 seconds
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 4 Output for 10 degrees Celsius

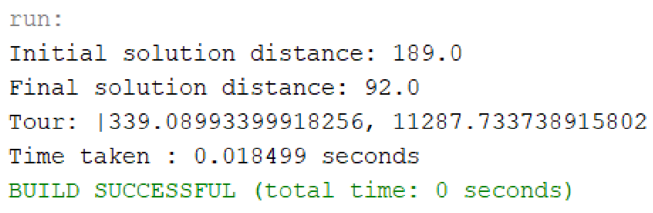
With an initial temperature of 10 degrees Celsius, results were already consistent, reaching the final best solution of 92 kilometers on 20 trials. The time to compute the solution was 0.0127 seconds.



```
run:
Initial solution distance: 199.0
Final solution distance: 92.0
Tour: |343.0094681738903, 11293.349168108543
Time taken : 0.0158294 seconds
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 5 Output for 100 degrees Celsius

Further increasing the temperature to 100 degrees Celsius had no effect on the final solution distance, which is still at 92 kilometers, except that the time to compute the solution took an average 0.0155 seconds, which is 0.0031 seconds longer than with the temperature at 10 degrees Celsius.



```
run:
Initial solution distance: 189.0
Final solution distance: 92.0
Tour: |339.08993399918256, 11287.733738915802
Time taken : 0.018499 seconds
BUILD SUCCESSFUL (total time: 0 seconds)
```

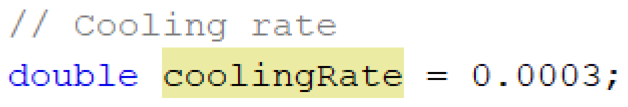
Figure 6 Output for 1000 degrees Celsius

To ensure that any further increase in temperature would be impractical, we tested the algorithm with a temperature of 1000 degrees Celsius. Figure 6 shows that the final solution distance is still the same as with 100 degrees Celsius and 10 degrees Celsius.

Upon experimenting with different temperatures, we have concluded that the temperature parameter has little influence on the results and any temperature above 10 would be sufficient to calculate the best solution for the given data set. In addition, using a lower temperature value also keeps the runtime of the algorithm lower than running with higher temperatures.

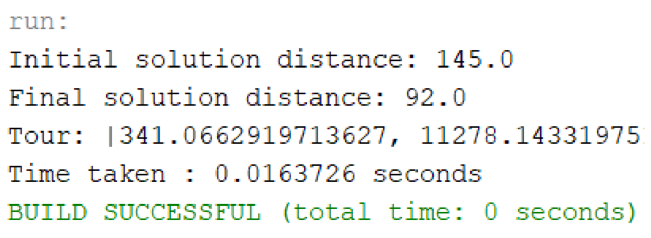
B. Changing Cooling Rate

Following the changing of temperature, the other required parameter is the Cooling Rate. Initially, we did the change of temperature investigation with a cooling rate of 0.0003, but here we will investigate if changing the cooling will affect the final solution. In this part, we will be using an initial temperature value of 100 degrees, as it was shown to be consistent to get the best final solution in the previous part.



```
// Cooling rate
double coolingRate = 0.0003;
```

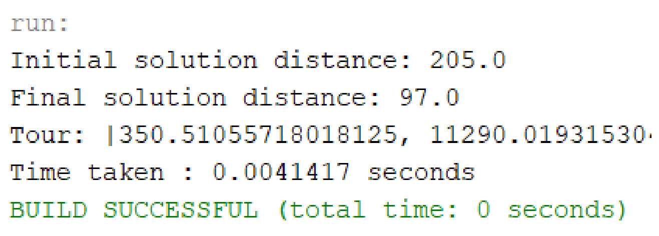
Figure 7 Cooling rate parameter



```
run:
Initial solution distance: 145.0
Final solution distance: 92.0
Tour: |341.0662919713627, 11278.14331975
Time taken : 0.0163726 seconds
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 8

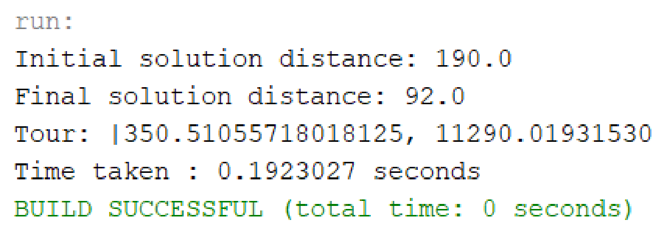
Over 20 trials, with the cooling rate set at 0.003, we get consistent final solution distance of 92 kilometers. With an average execution time of 0.016 seconds.



```
run:
Initial solution distance: 205.0
Final solution distance: 97.0
Tour: |350.51055718018125, 11290.01931530
Time taken : 0.0041417 seconds
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 9 Output with cooling rate of 0.03

Experimenting with higher cooling rates, we changed the cooling rate from 0.0003 to 0.03. The final solution distances averaged about 94 to 108 kilometers over 20 trials, but with a significantly lower time taken for each trial, taking about 0.012 seconds faster than with cooling rate of 0.0003. This shows inconsistency in getting the final solution distance and producing worse solutions for the most part.



```
run:
Initial solution distance: 190.0
Final solution distance: 92.0
Tour: |350.51055718018125, 11290.01931530
Time taken : 0.1923027 seconds
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 10 Output with cooling rate of 0.000003

Following the test with a higher cooling rate, we also tested with a lower cooling rate of 0.000003. Figure 10 shows the output for one of the trials. Over 20 trials, the final solution distance was consistently 92 kilometers, with an average compute duration of around 0.2 seconds, largely slower than with cooling rate of 0.0003.

With these results, it is concluded that the cooling rate plays a significant factor in consistently producing the best final solution distance. We found out that the optimal cooling rate for getting consistent results and having low compute time is 0.0003.

C. *Comparing Results with Google Maps algorithm*

To compare the results of the SA algorithm against Google Maps algorithm, we imported the exact same dataset in the MyRouteOnline application to get the best solution from there. To reiterate, we are using MyRouteOnline to access Google Maps algorithm because MyRouteOnline directly syncs its results from Google Maps, therefore results will be the same.

TABLE II. BEST ROUTE FROM MYROUTEONLINE

Stop #	Location name
1	Stadium Hoki MPK
2	Tengku Ampuan Rahimah Hospital
3	Supercourts Badminton Centre
4	Sunway Lagoon
5	Thai Buddhist Chetawan Temple
6	Merdeka Square
7	Petronas Twin Towers
8	Zoo Negara
9	Leisure Mall
10	National Stadium Bukit Jalil
11	Stadium Hoki MPK

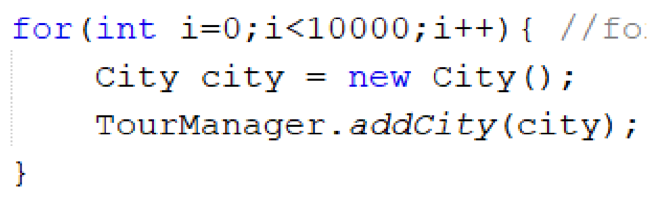
TABLE III. BEST ROUTE FROM SA ALGORITHM

Stop #	Location name
1	Stadium Hoki MPK
2	Tengku Ampuan Rahimah Hospital
3	Sunway Lagoon
4	Thai Buddhist Chetawan Temple
5	Merdeka Square
6	Petronas Twin Towers
7	Zoo Negara
8	Leisure Mall
9	National Stadium Bukit Jalil
10	Supercourts Badminton Centre
11	Stadium Hoki MPK

In terms of the distance travelled, the final solution distance for Google Maps is 146.37 kilometers, which is considerably larger than that of the SA algorithm, with a significant difference of 54.37. To explain the reason for why Google Maps distance is significantly higher than the SA algorithm boils down to the limitation of our code, as it only draws straight lines between locations, while Google Maps follows along the roads.

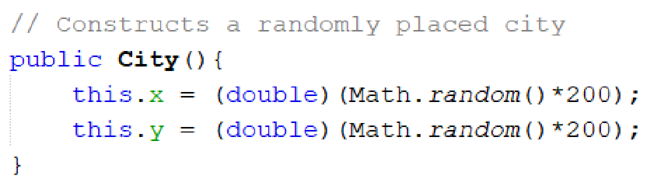
Though an interesting observation made here is that the route for SA and Google Maps differ at one stop, right after Tengku Ampuan Rahimah Hospital. For Google Maps, it leads to Supercourts Badminton Centre, while the SA algorithm leads to Sunway Lagoon instead. This would align with the fact that the limitation for the SA algorithm only draws straight lines, and ignores all road paths.

D. *Handling High Numbers of Cities*



```
for(int i=0;i<10000;i++){ //fo
    City city = new City();
    TourManager.addCity(city);
}
```

Figure 11 Set number of randomized Cities

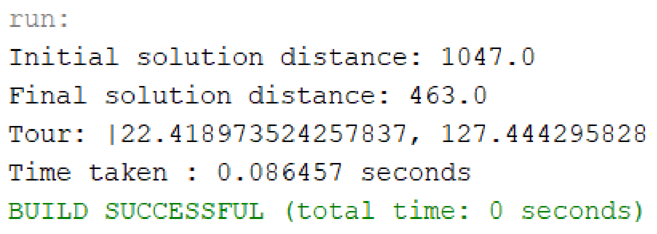


```
// Constructs a randomly placed city
public City(){
    this.x = (double)(Math.random()*200);
    this.y = (double)(Math.random()*200);
}
```

Figure 12 Function to randomize coordinates for each City

In addition to accepting our own coordinates, the SA algorithm code also allows us to set a specified number of cities, while the program gives each city a randomized location. This investigation would be useful to see how well it can take a high number of locations, and at what point does it take too long to compute.

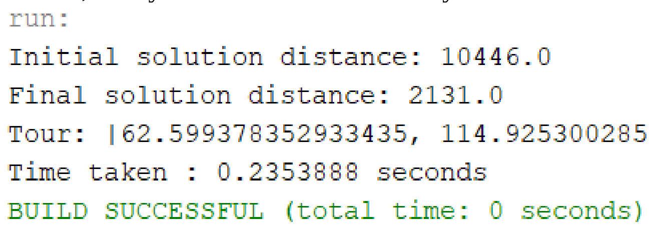
As for parameters, since we are dealing with a much larger quantity of data, we will use a higher initial temperature of 1000 degrees Celsius and a lower cooling rate of 0.00003.



```
run:
Initial solution distance: 1047.0
Final solution distance: 463.0
Tour: |22.418973524257837, 127.444295828
Time taken : 0.086457 seconds
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 13 Results for 10 cities

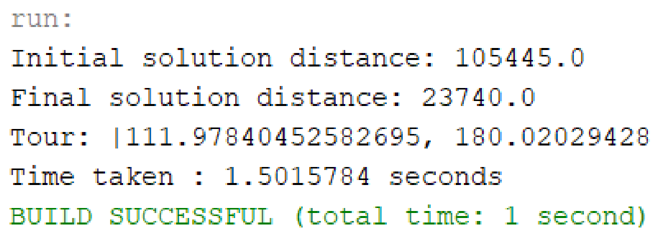
With only 10 cities to handle, the algorithm computes the solution with relative ease, taking only 0.086 seconds, barely noticeable to the human eye.



```
run:
Initial solution distance: 10446.0
Final solution distance: 2131.0
Tour: |62.599378352933435, 114.925300285
Time taken : 0.2353888 seconds
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 14 Results for 100 cities

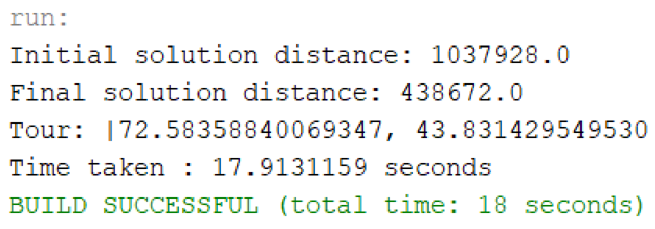
Bumping up the number of cities to 100 sees a notable increase in computational time, ending up with 0.235 seconds, which is more than double of that with 10 cities.



```
run:
Initial solution distance: 105445.0
Final solution distance: 23740.0
Tour: |111.97840452582695, 180.02029428
Time taken : 1.5015784 seconds
BUILD SUCCESSFUL (total time: 1 second)
```

Figure 15 Results for 1000 cities

Further elevating the number of cities to 1000 exponentially increased the time taken to 1.502 seconds, which is 1.267 seconds longer than with 100 cities.



```
run:
Initial solution distance: 1037928.0
Final solution distance: 438672.0
Tour: |72.58358840069347, 43.831429549530
Time taken : 17.9131159 seconds
BUILD SUCCESSFUL (total time: 18 seconds)
```

Figure 16 Results for 10000 cities

Lastly, we tested with 10000 cities. At this point, the wait time was too long to be convenient, as the time it took to compute was 17.91 seconds.

If we were using an app that was utilizing the SA Algorithm, we would be comfortable with waiting for 3 seconds at most. But any more than that, like with 10000 cities having a wait time of 17 seconds, it would be inconvenient. Overall, we would say that the algorithm is more than capable of handling a tremendous number of cities, but no more than 1000 cities.

VI. CONCLUSION

This paper discusses on using a simulated algorithm to solve the travelling salesman problem. We modified the parameters of a simple simulated algorithm to see what changes it brings to the system and the results. From the results, we concluded that changing the initial temperature between 10 and 1000 degrees Celcius did not display any change in the resulting distance of the path. However, the time taken for the algorithm to finish executing becomes longer with higher temperatures. Results also show that at faster cooling rates, the algorithm obtains a solution faster, but solutions are inconsistent and not optimal. At slower cooling rates, the algorithm was able to obtain the optimal solution, but it takes significantly more time to do so. The optimal cooling rate we concluded was 0.0003. We have also modified the number of cities to experiment on the time taken. The results show that below 1000 cities, the algorithm performed within an acceptable duration. With numbers more than 1000, the duration becomes too long to be convenient to users. Lastly, we compared the solution obtained from the algorithm with MyRouteOnlines's online route planner, which syncs with google maps. We entered the same locations for both the algorithm and the website. We have found that the optimal distance generated from MyRouteOnline was significantly higher than the algorithm. It is believed that the massive difference between our results and MyRouteOnline is that the simulated algorithm gets to the following location on a straight line and does not consider the real-world obstacles and roads that are possible to reach the following location. This is the limitation of solving the travelling salesman problem using the simulated algorithm.

REFERENCES

[1] L. Jacobson, "Simulated Annealing for beginners", *Theprojectspot.com*, 2013. [Online]. Available: https://www.theprojectspot.com/tutorial-post/simulated-annealing-algorithm-for-beginners/6. [Accessed: 28- Apr- 2021]

[2] N. Hanafiah, I. Wijaya, S. Xavier, C. Young, D. Adrianto and M. Shodiq, "Itinerary Recommendation Generation using Enhanced

Simulated Annealing Algorithm", *Procedia Computer Science*, vol. 157, pp. 605-612, 2019.

[3] Y. Liu, S. Xiong and H. Liu, "Hybrid simulated annealing algorithm based on adaptive cooling schedule for TSP", Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation - GEC '09, 2009.

[4] H. Mukhairez and A. Maghari, "Performance Comparison of Simulated Annealing, GA and ACO Applied to TSP"", *International Journal of Intelligent Computing Research*, vol. 6, no. 4, pp. 647-654, 2015.

[5] L, Yong, Zhu, B and T, Yong, "Simulated annealing algorithm for optimal capital growth", Physica A: Statistical Mechanics and its Applications, Volume 408, Pages 10-18, ISSN 0378-4371, , 2014.

[6] Subramanian, A, Battarra, M. "The Journal of the Operational Research Society", Abingdon Vol. 64, Iss. 3, Pages 402-409. 2013.