Applying Simulated Annealing to the Travelling Salesman Problem

In this part, we turn to the application of one Markov Chain Monte Carlo method, the Simulated Annealing Algorithm to the Travelling Salesman Problem. I have implemented the algorithm provided by the book "Simulated Annealing and the Monte Carlo Method" in Python. Moreover, two interesting alternative approaches have also be proposed.

Problem Introduction

Travelling Salesman Problem (TSP) is a famous problem in combinatorial optimization. In this problem, you are given a list of cities and the distance between any two of them. You are required to find a the shortest path which starts from a city and visits each remaining city exactly once and finally return to the origin.

The problem is equivalent to finding such a permutation with the least cost. Suppose the list consists of $\mathbf{n}$ cities. Then, there should be $\mathbf{n!}$ permutations in total. If by brute force, we have to compute the cost of all the $\mathbf{n!}$ paths and select the least one. There is no doubt for the infeasibility of such method. Even if there are only 20 cities, there will be over 10^{18} paths. Therefore, we need more efficient methods to solve this problem. Up till now, various methods have been proposed. Basically, they can be divided into Exact algorithms and Heuristic and approximation algorithms. In the next part, we wil get a closer look into the method I used in this research, which was Simulated Annealing algorithm.

About Simulated Annealing

For those continuous and differentiable functions, gradient descent is widely used in searching for the global minimum point. However, in the TSP, it is unsuitable to use the gradient-based methods to optimize the function because the search space is discrete rather than continuous and differentiable.

Here we consider the Simulated Annealing (SA) method. The idea of SA comes from annealing in metallurgy, a technique which involves heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. It is a probablistic technique based on MCMC to approximate the global optima of discrete functions with a large search space. The idea is that the Markov Chain should have a unique stationary distribution in which most of the probabilities are placed in states which have small value of cost. **Add some citation here** The most remarkable part is that it involves $f_T(x)$, a set of densities which is proportional to $[f(x)]^{1/T}$. During the process, every time we use MCMC sampling to draw a sample $\mathbf{X^{(k)}}$ based on the probability distribution $f_T(x)$. In order to make the distribution have a sharper peak at the global maxima, we should gradually reduce the temperature as the Markov Chain is running. Therefore,

the design of the Markov Chain and the setting of temperature are of vital importance.

Design of Markov Chain

The MCMC method works if and only if the Markov Chain is irreducible and aperiodic. As for TSP, our objective is to find a permutation of the cities such that the total cost is the lowest. Hence, we treat each permutation of the cities as one state of the Markov Chain. Then this Markov Chain has $\mathbf{n!}$ states, which makes the transition probability matrix really large. However, we do not need to use the whole matrix as we run the Markov Chain. What we need to do is to define "neighboring" states, establish the stationary distribution and determine the transition probability.

Definition of neighborhood

There are several ways to define neighboring states in TSP. Here I have implemented two kinds of neighborhood:

1. $\mathbf{Reverse:}$ For any two states m and n, if there exist two positions i and j such that n can be got by reversing the subsequence of m from i to j while the other positions remain the same, then and m and n are neighbors to each other.
2. $\mathbf{Swap:}$ For any two states m and n, if there exist two positions i and j such that n can be got by swapping the city at position i and the city at position j in m, then m and n are neighbors to each other.

Stationary distribution

Since we wish to put more probability mass on those states with a lower cost, suppose the state space is $\mathbf{S}$ and the cost function is $f: \mathbf{S} \to \mathbf{R}$. The cost function measures the total distance of the path regarding to the permutation. We need to find a distribution in which the smaller the cost, the larger the probability mass. Moreover, as we have stated before, most of the probability should be placed in the optimal state as we reduce the temperature. Thus, the general idea of designing such stationary distribution comes from Bolzmann distribution. The Bolzmann distribution $\pi_{f,T}$ satisfies $\pi_{f,T}(s) = \frac{1}{Z_{f,T}} * exp(\frac{-f(s)}{T})$ for any $s \in \mathbf{S}$ where $Z_{f,T} = \sum_{s \in \mathbf{S}} exp(\frac{-f(s)}{T})$. In this way, the smaller the cost, the larger the probability will be. Moreover, as the temperature tends to zero, we want the probability mass of the global minimum point tends to 1. The proof is as follows:

Without the loss of generality, suppose the state s_1 is the unique global minimum point of the function f and denote $\mathbf{S'}$=$\mathbf{S} \setminus \{s\}$. We also suppose that s_2 satisfies $f(s_2) = min_{s \in \mathbf{S'}} f(s)$. Let $a = f(s_1)$ and $b = f(s_2)$. Then we have:

$$\pi_{f,T}(s_1) = \frac{1}{Z_{f,T}} * exp(\frac{-f(s)}{T})$$

$$= \frac{exp(\frac{-a}{T})}{exp(\frac{-a}{T}) + \sum_{s' \in \mathbf{S'}} exp(\frac{-f(s')}{T})}$$

$$\geq \frac{exp(\frac{-a}{T})}{exp(\frac{-a}{T}) + |\mathbf{S'}| exp(\frac{-b}{T})}$$

$$= \frac{1}{1 + |\mathbf{S'}| exp(\frac{a-b}{T})}$$

Note that the last fraction converges to 1 as T tends to 0. Therefore, as we gradually reduce the temperature, the probability mass of the global minimum point tends to 1. Therefore, in order to find such s_1, we can construct a Markov Chain to simulate the Bolzmann distribution $\pi_{f,T}$ on $\mathbf{S}$.

Transition probability

As I have stated before, the temperature is a parameter in computing the transition probability in the Markov Chain. Hence, the transition probability matrix should also change as the temperature changes. Therefore, the whole algorithm can be viewed as an inhomogeneous Markov Chain in which there are $\mathbf{N}$ Markov Chains running one by one. They have the same set of states but different transition probability matrices. Let the k-th Markov Chain has transition probability matrix $\mathbf{P_k}$ and it runs for N_k iterations. Now that I have defined the neighborhood in two ways, the next problem is the transition probability. According to the idea in the construction of Matropolis chain, given the stationary distribution, the transition probability can be computed as the following:

$$P_{i,j} = \begin{cases} \frac{1}{d_i} * min\{\frac{\pi_j d_i}{\pi_i d_j}, 1\} & if\ s_i\ s_j\ are\ neighbors \\ 0 & if\ s_i\ \neq\ s_j\ and\ they\ are\ not\ neighbors \\ 1 - \sum_{s_l \in I} \frac{1}{d_i} * min\{\frac{\pi_l d_i}{\pi_i d_l}, 1\} & if\ i = j \end{cases}$$

Note that I is the set of the neighbors of s_i and d_i is the cardinality of I. In the third case, we sum over all the neighbors of s_i.

Generally, people construct the Markov Chain by the method of Metropolis chain when implementing SA. Based on this, I propose two alternative ways to define the transition probability. The three methods are listed as the following:

1. We directly construct a Metropolis chain. Here we should note that for any state s, there are $nC2$ ways to choose the two cities for reversing or swapping. Thus, the number of neighbors for any state s should be $nC2$ so we can easily cancel the d_i and d_j which are the number of neighbors of the two states as we have stated previously. Moreover, the term $Z_{f,T}$ can also be cancelled, which frees us from computing $exp(-\frac{f(s)}{T})$ for the whole state space. The following is the formula of the transition probability from state s_i to s_j.

   $$P_{i,j} = \begin{cases} \frac{2}{n(n-1)} * min\{exp(-\frac{f(s_j)-f(s_i)}{T}), 1\} & if\ s_i\ s_j\ are\ neighbors \\ 0 & if\ s_i\ s_j\ are\ not\ neighbors \\ 1 - \sum_{s_l \in I} \frac{2}{n(n-1)} * min\{exp(-\frac{f(s_l)-f(s_i)}{T}), 1\} & if\ i = j \end{cases}$$

   Therefore, we can deduce the transition mechanism. Suppose the current state is s_i. We first uniformly select two distinct positions from 1 to n so that we can generate a neighbor of the current state s_j. Then if $f(s_j) < f(s_i)$, we transit directly. Otherwise, we uniformly generate a random number y between 0 and 1. If $y < exp(-\frac{f(s_j)-f(s_i)}{T})$, we transit to s_j. Otherwise, we stay in state s_i.

2. In the first method, we just randomly generate a neighbor of the current state and determine whether to transit. The probability of getting any neighboring state is the same no matter whether the cost is smaller or not. Therefore, in this method, I give more weights to those neighboring states with a smaller cost, which will increase the probability to transit into a state with a smaller cost. Suppose we assign weight $\boldsymbol{\rho}$ to the states with smaller costs. Empirically, $\boldsymbol{\rho}$ is set to 0.9. It means that the sum of the probabilities to generate states with

smaller costs equals to $\boldsymbol{\rho}$. Thus, the sum of the probabilities to generate states with larger costs equals to $1-\boldsymbol{\rho}$. In practice, we uniformly generate a random number between 0 and 1. If the number is smaller than $\boldsymbol{\rho}$, then we keep on generating neighboring states until we get a state with a smaller cost and transit into this new state. Otherwise, we keep on generating neighboring states until we get a state with a larger cost. The transition probability would then be $exp(-\frac{f(s_j)-f(s_i)}{T})$. Therefore, we again uniformly generate a number between 0 and 1. If the number is smaller than the transition probability, we transit to the new state s_j. Otherwise, we stay in s_i. The transition probability is defined as follows:

$$P_{i,j}=\begin{cases}\frac{\rho}{l_i} & if\ s_i\ s_j\ are\ neighbors\ and\ f(s_j)<f(s_i)\\ \frac{1-\rho}{h_i}exp(-\frac{f(s_j)-f(s_i)}{T}) & if\ s_i\ s_j\ are\ neighbors\ and\ f(s_j)\geq f(s_i)\\ 0 & if\ s_i\ s_j\ are\ not\ neighbors\\ 1-\rho-\sum_{s_h\in\mathbf{H_i}}\frac{1-\rho}{h_i}exp(-\frac{f(s_h)-f(s_i)}{T}) & if\ s_i=s_j\end{cases}$$

Note that l_i represents the number of neighboring states having smaller cost and h_i represents the number of neighboring states having larger cost. $\mathbf{H_i}$ is the set of neighboring states with larger cost.

3. Now that we have came up with the idea to assign larger weights to those neighboring states with smaller cost, we can furtherly consider assigning even more weights to those neighboring states which reduce more cost. In this way, the more cost a state can reduce, the more likely it will be picked. Inspired by the softmax function in Machine Learning, this method modifies the previous one by assigning weights to each neighboring states according to how much it reduces or increases the cost. Therefore, the idea is as the following. We also assign a weight $\boldsymbol{\rho}$ to the neighboring states with smaller cost and $1-\boldsymbol{\rho}$ to the other neighboring states. Then we uniformly generate a random number x between 0 and 1. If $x<\boldsymbol{\rho}$, we generate a number of sample neighbors which should all have smaller cost. Otherwise, we generate a number of sample neighbors which should all have larger cost. Then we assign weights based on their cost. Since there exist some tricks in weights assignment, more details will be provided in the part of experimement design.

Annealing

The temperature is one of the most important hyper-parameters in applying SA algorithm on TSP. It determines the probability of transition to the new state if the new state has a larger cost than the older one. According to the formula, given that the generated state is s_j and it has a larger cost than the old state s_i, the transition probability should be $exp(-\frac{f(s_j)-f(s_i)}{T})$. This positive probability has distinguished the SA algorithm with the Greedy algorithm in which the local optimal is chosen in every step. The Greedy algorithm has a severe drawback. It may get stuck in the local minimum if every time the chain either transits to a state with smaller cost or stays unchanged. There might exist some local optimal states which have the smallest cost among all their neighbors. Greedy algorithm will never get out of such states once getting in and the chain will never converge to the global optimal.

Thus, the probability of "temporarily getting into a worse state" plays an important role in the optimization process. From the formula we can see that the smaller the temperature T, the smaller the transition probability. As the inhomogeneous chain runs, the temperature will gradually decrease. At the beginning, the temperature is usually set high and it enables the chain

to avoid being stuck in the local minimum. After many iterations, the chain converges and the comparatively lower temperature fixes the chain at the global minimum. There is a theorem which states that if T approaches 0 sufficiently slowly, then the probability for the chain to get to the global minimum will tend to 1 as the number of iterations goes to $+\infty$. (Refer to the book "Finite MC..." and cite for this)

Empiracally, the initial temperature is relevant to the difference of cost among different permutation. For example, the initial temperature for the case in which all the cities are within one province must be different from the case in which the cities are from different continents. As for the annealing schedule, although theorems state that sufficiently slow annealing guarantees the convergence to the global minimum, in practice, we have to use faster annealing otherwise the algorithm will take quite long time. In this research, for simplicity, I update the temperature by multiplying it with a discount factor (e.g. 0.9) after certain number of iterations. This number should also be adjusted for each of the three transition mechanism as I have stated above because in practice, the second and third transition method takes far more time for one single iteration than the first one. We will delve deeper into this in the next part about experiment design.

Verification

Graph Initialization

When it comes to the graph for the cities, all we need is an adjacency matrix which demonstrates the distance between any two cities. Theoretically, we can generate distances randomly rather than using real-world data. However, we cannot just uniformly generate a number between 0 and a positive number (the maximum distance between any two cities) and set it as the distance between two cities because the graph may violate triangular inequality. (i.e. distance between A & B > distance between A & C + distance between B & C) Therefore, I initialize the graph in the following steps:

1. Set a distance d. d is actually the length of a square which covers all the cities.
2. Uniformly generate two independent numbers between 0 and d. Set these two numbers as the $xy-coordinates$ of a city.
3. Repeat the second step for $N-1$ more times.
4. Compute the distance between any two cities.

Experiment Design

Now that we have had two definitions for neighborhood and three ways of transition, there are six combinations and we need experiments to rank the superiority. Basically, experiments should be designed to show from two perspectives: whether and where the Markov chain converges, how fast does the chain converge. The first can be evaluated by plotting the cost of the current permutation after each iteration. Ideally, after a sufficient number of iterations, the temperature T will tend to 0 and the chain will no longer transit. As for the speed of convergence, it can furtherly be evaluated from two perspectives including the number of iterations before convergence and the running time until convergence. This cannot be evaluated by simply regarding to the plot. Thus, here we need one more clarification about the judgement of convergence. In this research, we use the celling of the arithmatic average of the last 10 costs as

the threshold of convergence. The chain has converged at some iteration if and only if the costs of the following 100 permutations are all smaller than the threshold.

Therefore, in general, the evaluation can be divided into four parts. Their contents are shown below. Note that the "Reverse" or "Swap" inside the parenthesis defines the neighborhood.

Part	Content
1	(Reverse) Evaluate the relationship between the cost of permutation and number of iterations
2	(Swap) Evaluate the relationship between the cost of permutation and number of iterations
3	(Reverse) Evaluate the relationship between the cost of permutation and running time
4	(Swap) Evaluate the relationship between the cost of permutation and running time

Before starting the experiments, we still need to clarify some details about the algorithm implementation. In the second and third transitions, the cost of candidate permutations must in accordance with a certain principle. (i.e. all have smaller costs or all have larger costs) Especially in the third one, since we are not able to consider all the permutations having smaller or larger cost than the current permutation, we need to generate a pool of samples. However, as the Markov Chain runs, it will become more and more difficult to find neighboring states with smaller costs. Hence, every time we are trying to search for a neighboring state with a smaller cost, the chain will stay in the same state if the we fail to search such neighboring state after 1000 trials. Otherwise, if the chain has entered into a local minima, then there is no chance to find a neighboring state with a smaller cost and the chain is not able to continue.

Experiments

In every part, we validate by applying the algorithm to graphs with 50, 100, 150, 200 cities respectively. After generating the four graphs, we store them into files and no longer generating new ones in the future. As for the plots, since the second and third transitions take much fewer iterations to converge than the first one, it is necessary to plot the second and third transitions individually.

In order to get the best results for each of the transitions, I tune the hyperparameters including the initial temperature and annealing schedule before comparation.

For reference, since the Markov Chain will converge to the global minima if the annealing is sufficiently slow. Thus, I set a slow annealing schedule and ran for over 100,000 iterations before the experiment for the four graphs in order to get the approximate global minimum cost beforehand. Note that this is just for reference in case that the error is too large during the experiments, not exactly the global minimum. Actually, we have no way other than the brute force method to get the real global minimum.

Since the convergent point may differ even if I repeat the same experiment, I use the **multiprocessing** package in Python to repeat the experiment and use the one with the smallest convergent point.

Part 1

In this part, two states are neighbors if and only if there exists a subsequence such that one can be obtained by **reversing** the subsequence of the other. We wish to find the relationship between the cost of current permutation and the **number of iterations**. The key hyperparameters are listed below:

Vertices	Transition	Initial Temperature	Annealing Schedule	Max Iteration
50	T1	1	300	10000
50	T2	1	30	300
50	T3	1	20	150
100	T1	2	200	30000
100	T2	2	30	600
100	T3	2	20	250
150	T1	5	1000	60000
150	T2	2	50	1000
150	T3	2	40	320
200	T1	2	1000	100000
200	T2	2	50	1500
200	T3	2	40	500

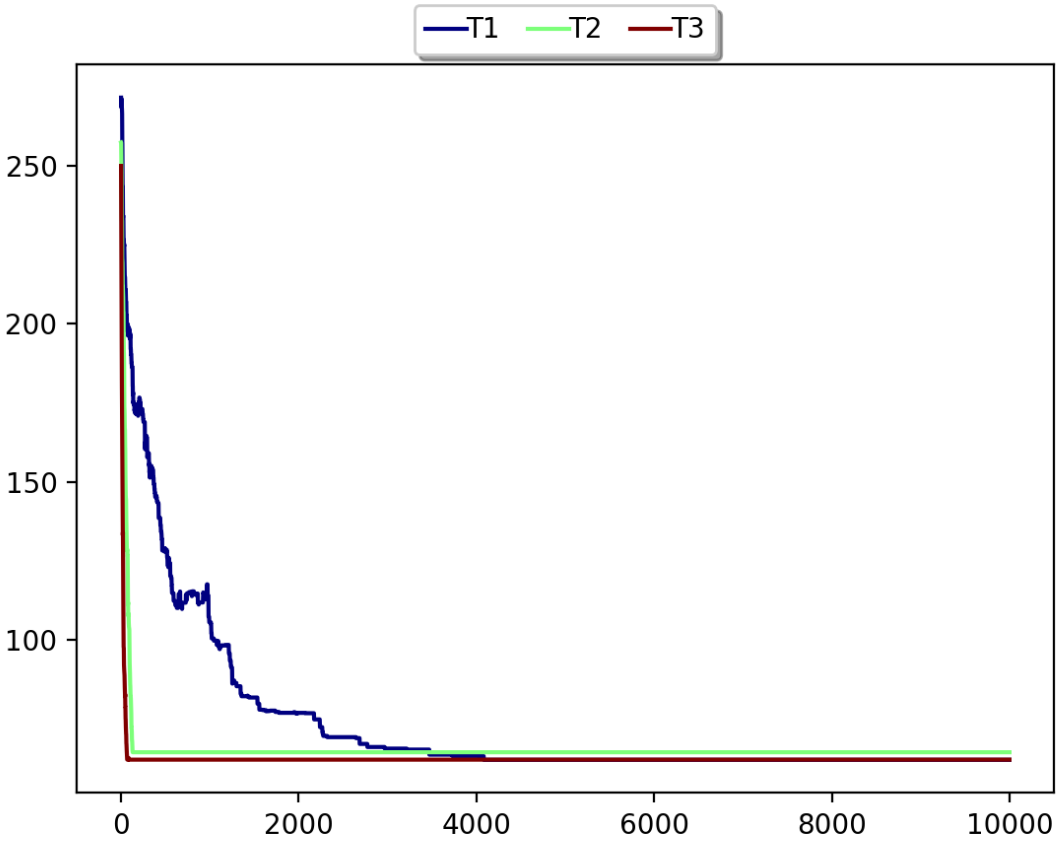
Here are the experiment results I got. Note that for each case, in terms of number of transitions, Transition 1 >> Transition 2 > Transition 3, I plot Transition 2 & 3 individually. The first is the result printed on the terminal. It indicates where the Markov Chain converges and how many iterations it take.

Vertices = 50

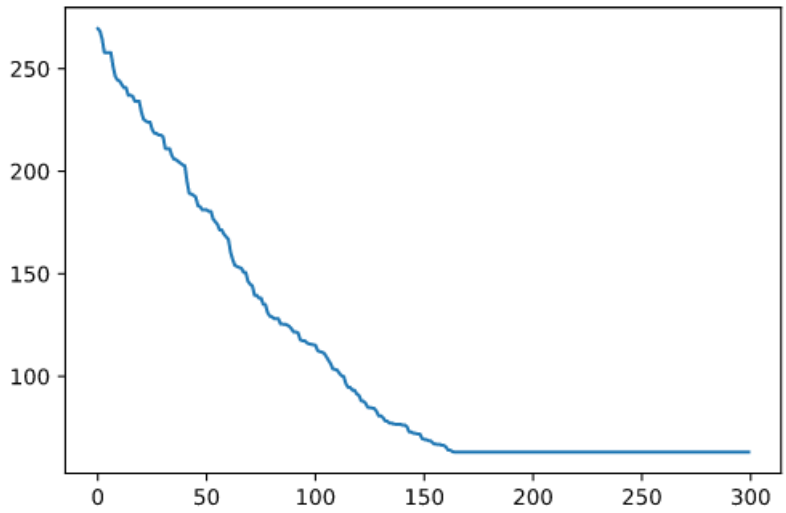
Numerical Result

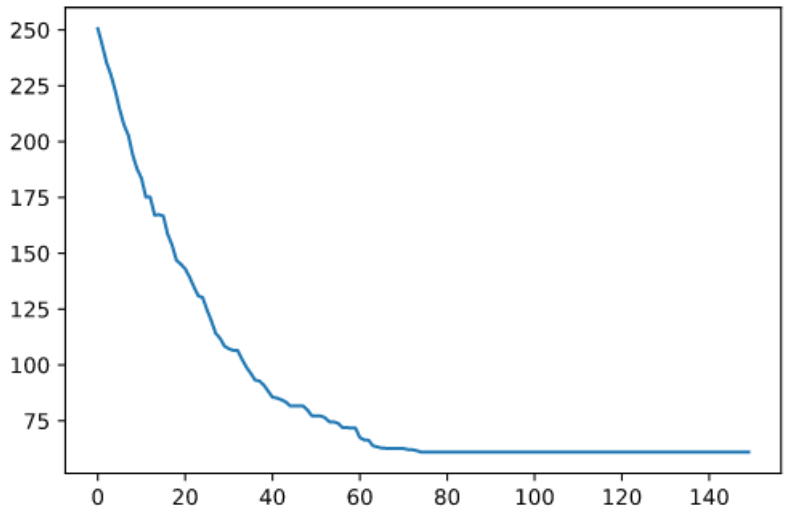
```
Finished transition 1
The sequence converges to 62.16461007996516
Achieved convergent at 4082 iteration

Finished transition 2
The sequence converges to 64.56198417051955
Achieved convergent at 130 iteration

Finished transition 3
The sequence converges to 62.22017848026739
Achieved convergent at 68 iteration
```

Combined Result



Results for Transition 2 & Transition 3



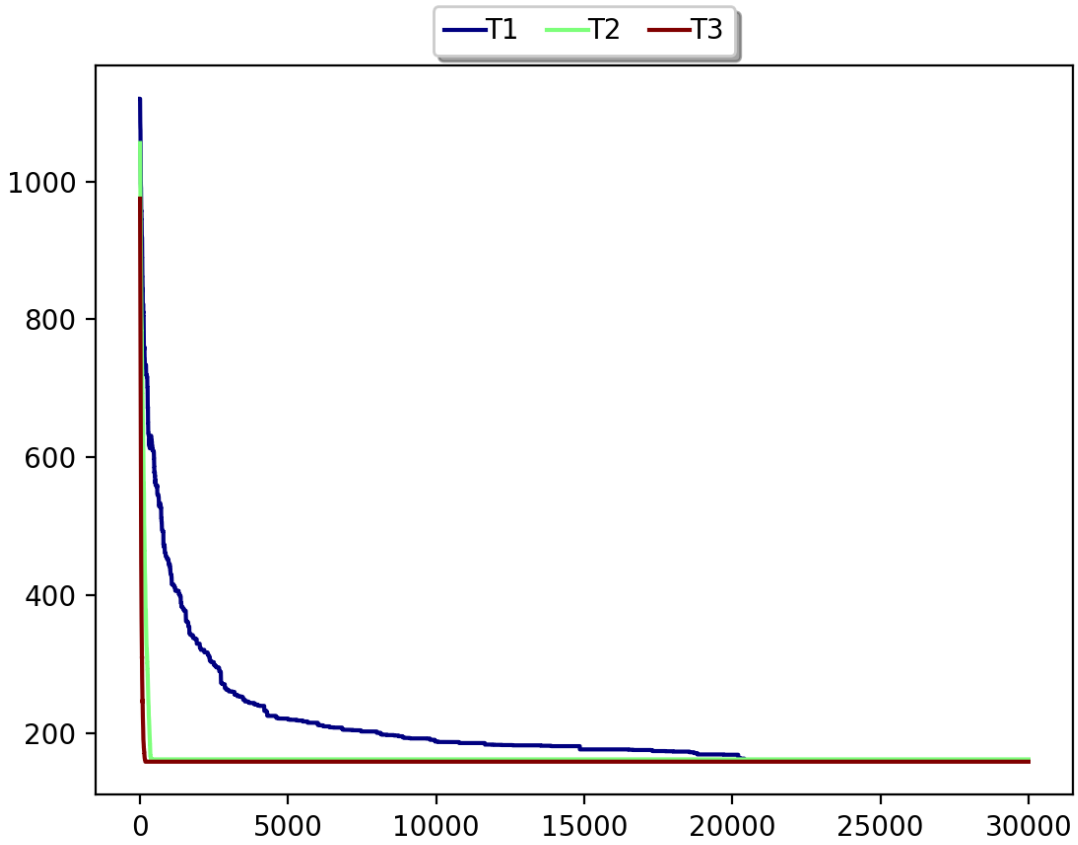


Vertices = 100

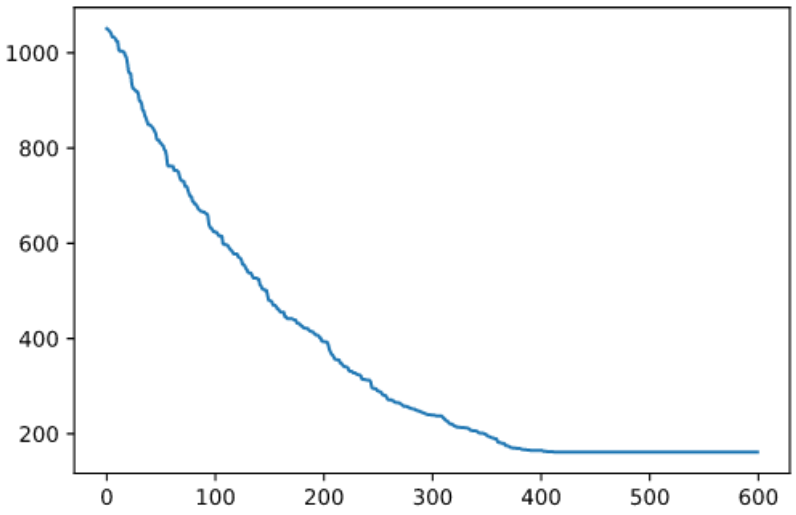
Numerical Result

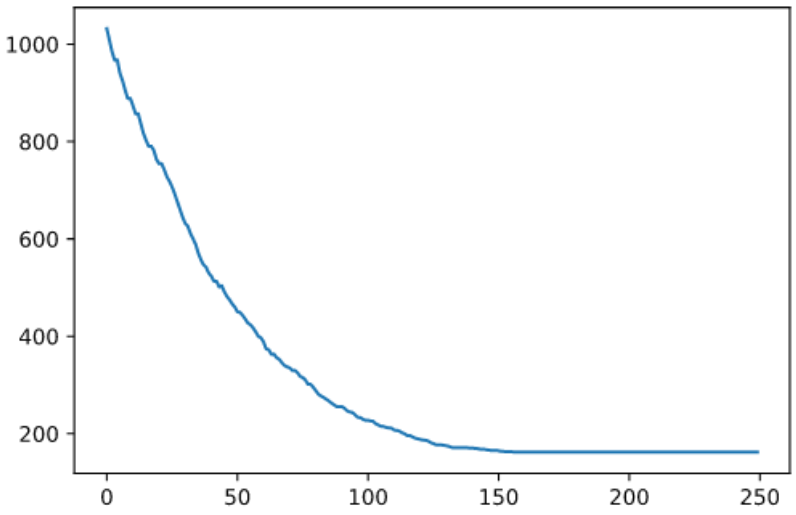
```
Finished transition 1
The sequence converges to 160.68886421544337
Achieved convergent at 27031 iteration

Finished transition 2
The sequence converges to 162.08027085354541
Achieved convergent at 378 iteration

Finished transition 3
The sequence converges to 158.73844704513414
Achieved convergent at 178 iteration
```

Combined Result



Results for Transition 2 & Transition 3



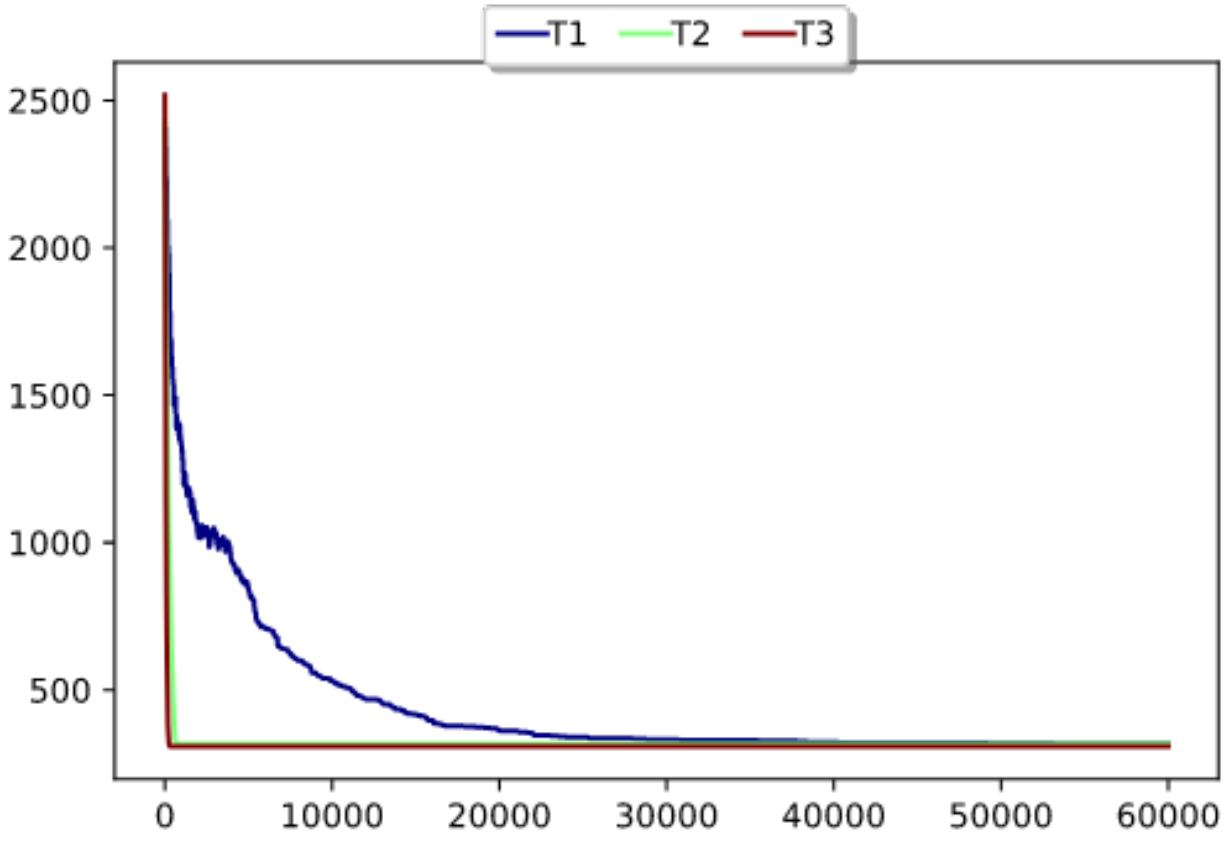


Vertices = 150

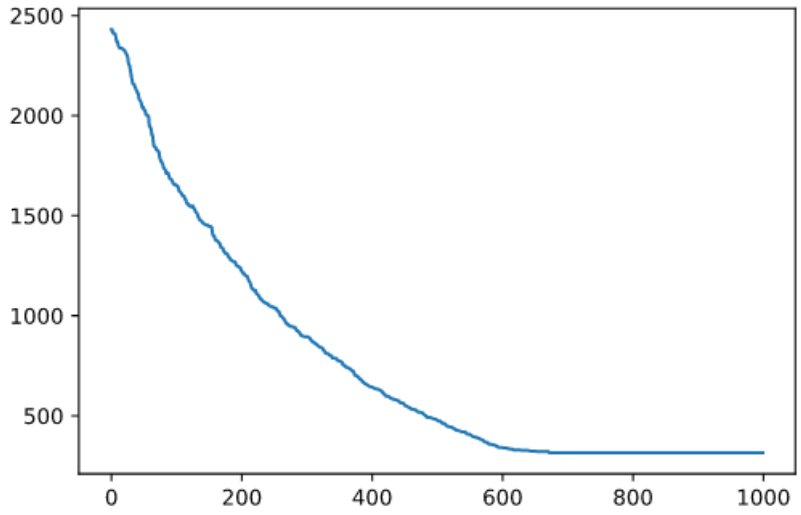
Numerical Result

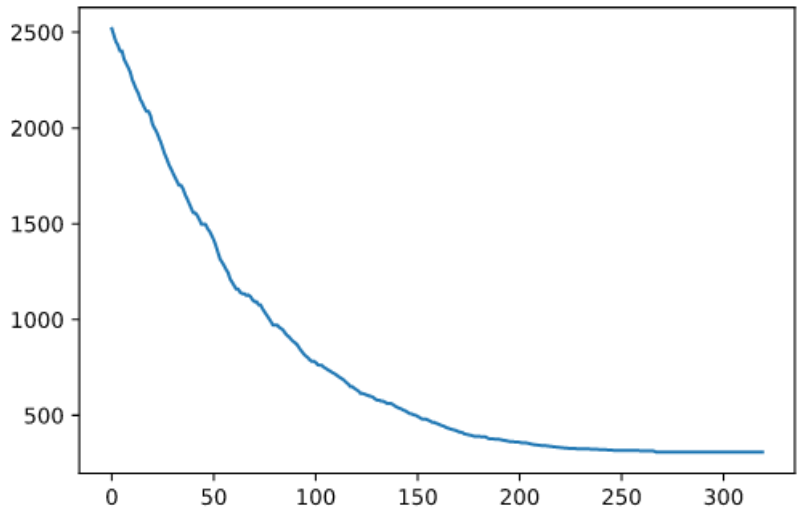
```
Finished transition 1
The sequence converges to 316.46740787075936
Achieved convergent at 53088 iteration

Finished transition 2
The sequence converges to 315.69932369786005
Achieved convergent at 675 iteration

Finished transition 3
The sequence converges to 307.39801588360785
Achieved convergent at 267 iteration
```

Combined Result



Results for Transition 2 & Transition 3



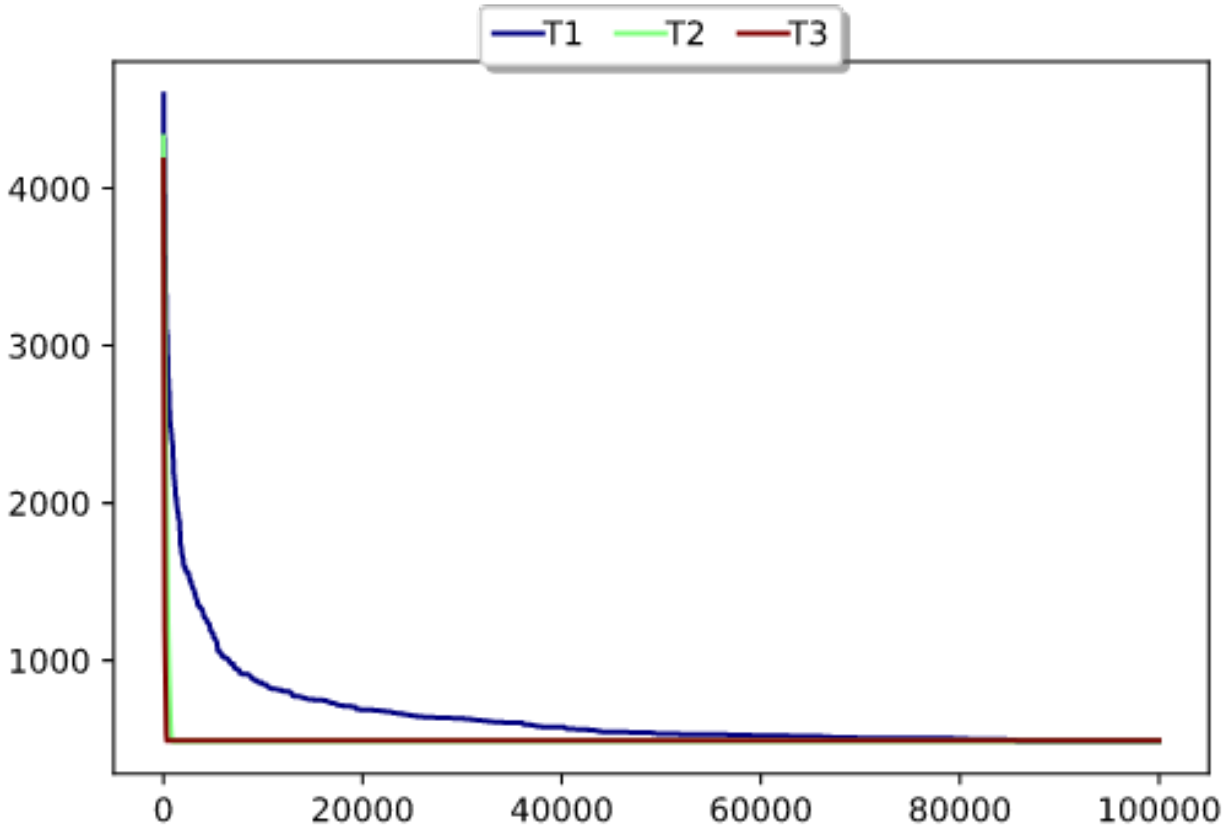


Vertices = 200

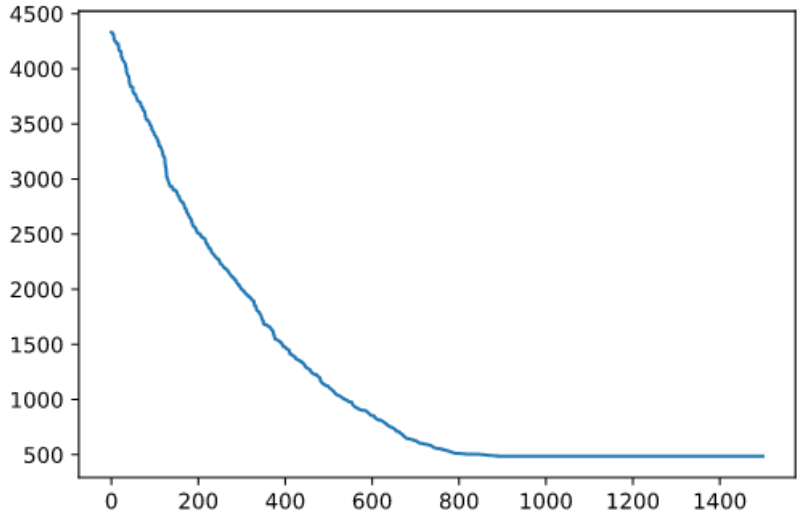
Numerical Result

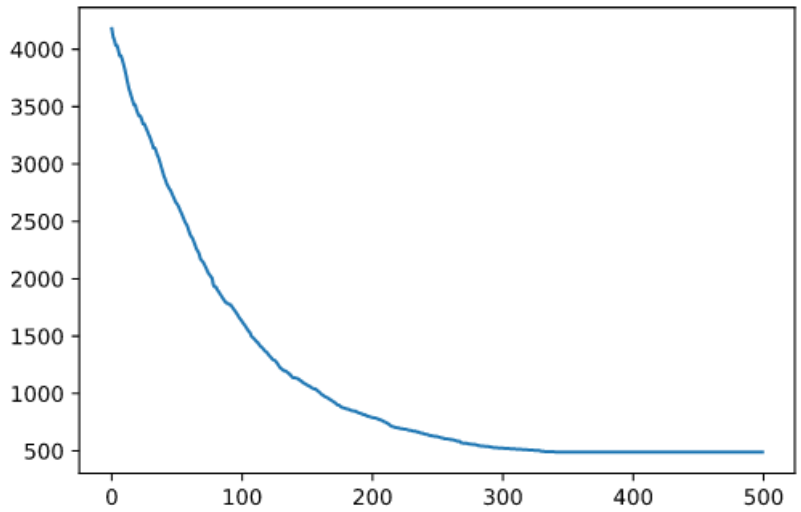
```
Finished transition 1
The sequence converges to 485.74905750753106
Achieved convergent at 92873 iteration

Finished transition 2
The sequence converges to 485.8727524561855
Achieved convergent at 904 iteration

Finished transition 3
The sequence converges to 488.6980835782184
Achieved convergent at 375 iteration
```

Combined Result



Results for Transition 2 & Transition 3





Conclusion for Part 1

The results above illustrate the followings:

1. The number of iterations required before convergence: Transition 1 >> Transition 2 > Transition 3. Time required for one iteration: Transition 3 >> Transition 2 > Transition 1.
2. With proper hyperparameter adjustment, all of the three transitions converge to approximately the same cost.
3. Transition 2 & Transition 3 are more "hyperparameter neutral". It means that when changing the number of vertices, we do not need to change the initial temperature and annealing schedule too much. However, for Transition 1, we have to make more changes. This illustrates that Transition 2 & Transition 3 might be more useful because it takes less time to tune the hyperparameters.

Part 2

In this part, two states are neighbors if and only if one can be obtained by **swapping** the positions of two cities in the other. We wish to find the relationship between the cost of current permutation and the **number of iterations**. The key hyperparameters are listed below:

Vertices	Transition	Initial Temperature	Annealing Schedule	Max Iteration
50	T1	2	1000	20000
50	T2	1	50	600
50	T3	1	10	150
100	T1	5	1500	50000
100	T2	4	40	1200
100	T3	1	10	600
150	T1	5	3000	13000
150	T2	2	100	2500
150	T3	1	20	500
200	T1	5	3500	140000
200	T2	2	150	3000
200	T3	1	30	550

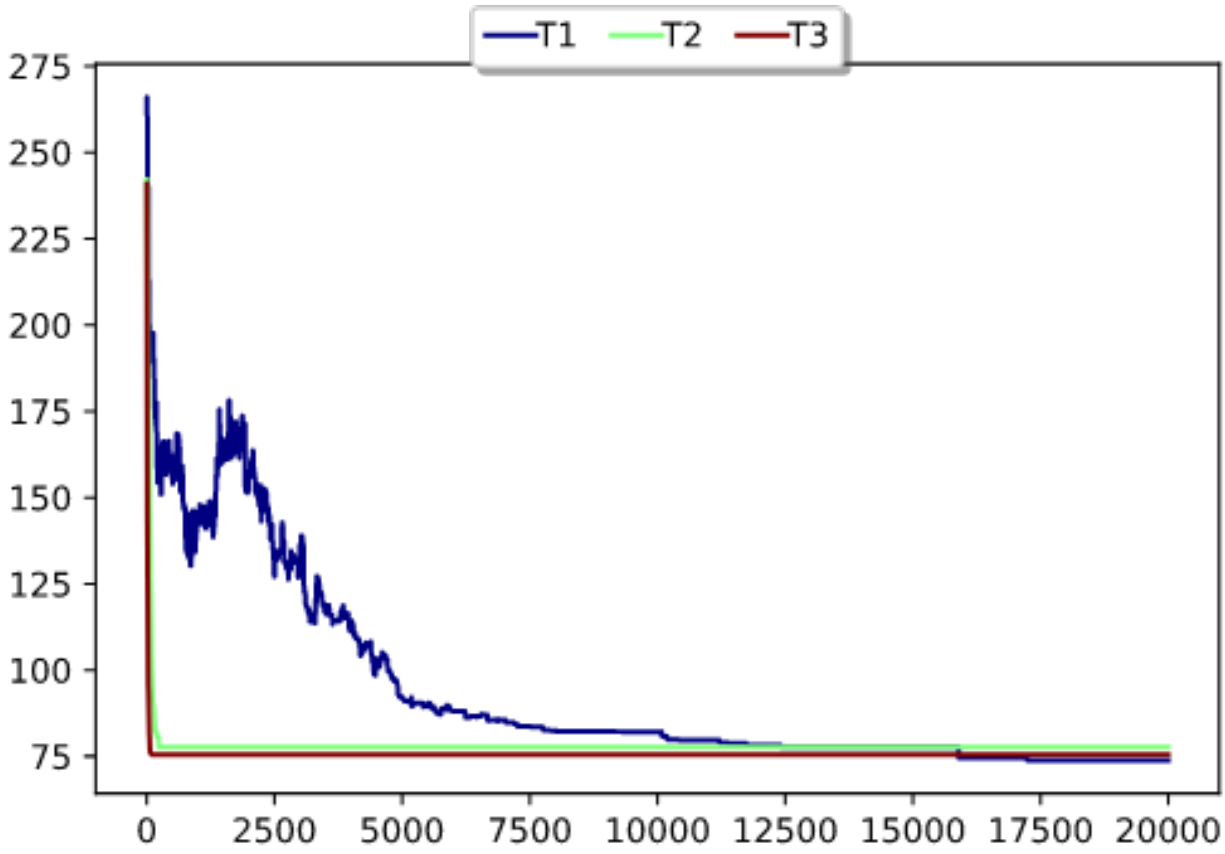
Here are the experiments results.

Vertices = 50

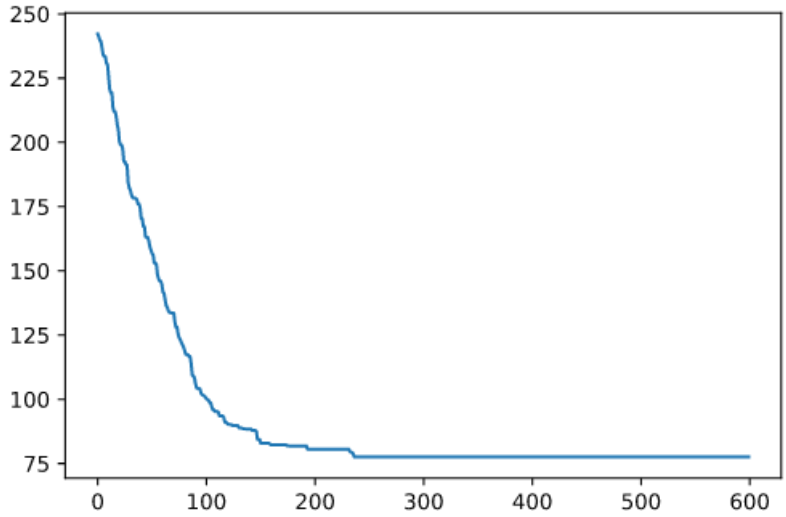
Numerical Result

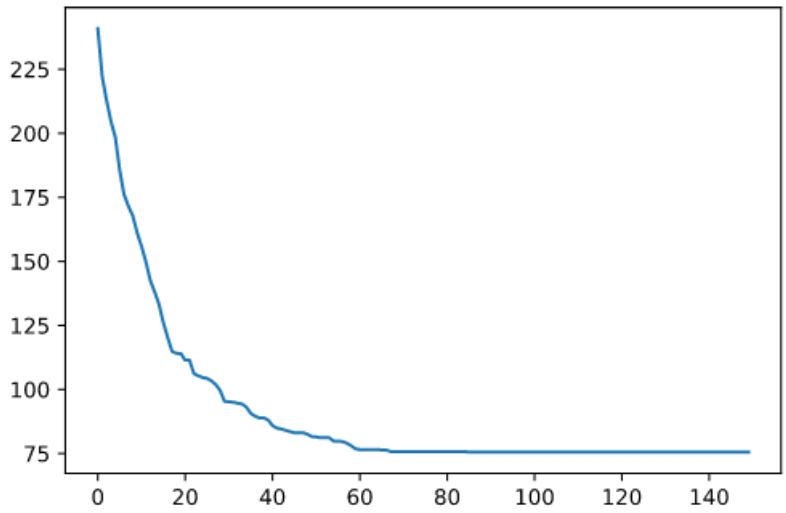
```
Finished transition 1
The sequence converges to 73.85886989674202
Achieved convergent at 17234 iteration

Finished transition 2
The sequence converges to 77.58108961482705
Achieved convergent at 236 iteration

Finished transition 3
The sequence converges to 75.49441005636082
Achieved convergent at 67 iteration
```

Combined Result



Results for Transition 2 & Transition 3



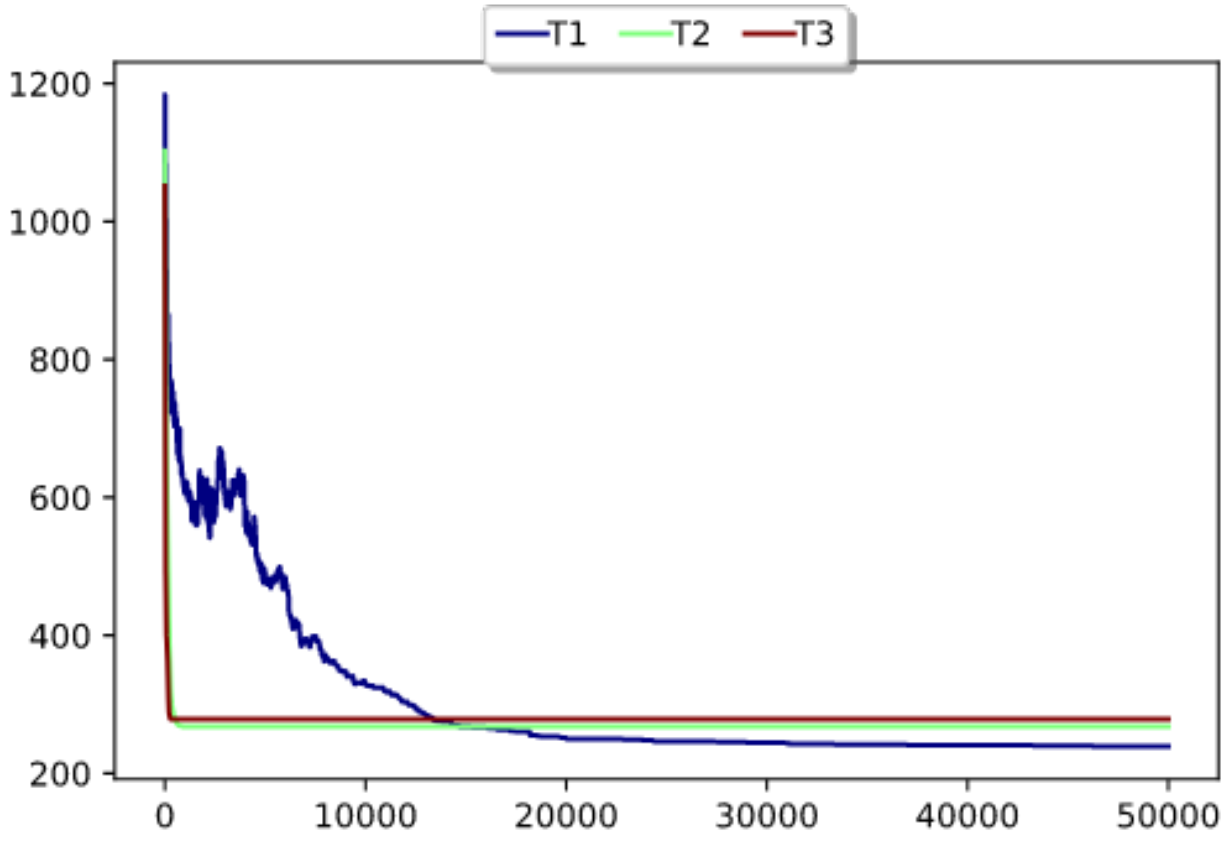


Vertices = 100

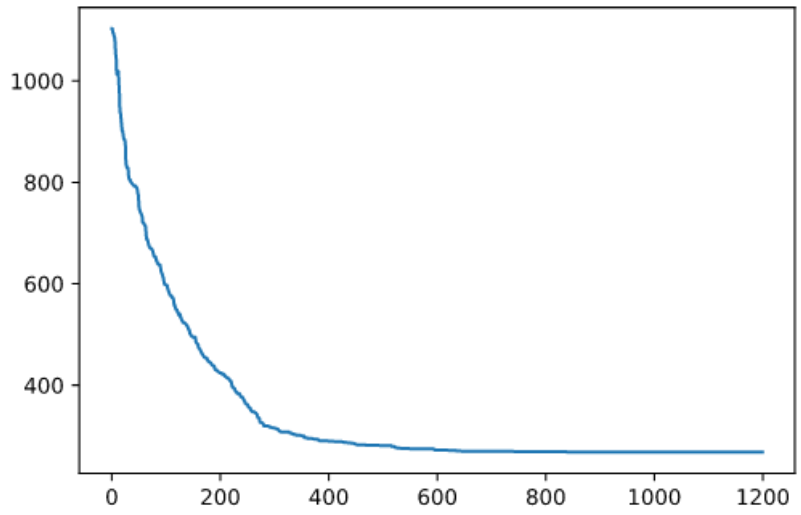
Numerical Result

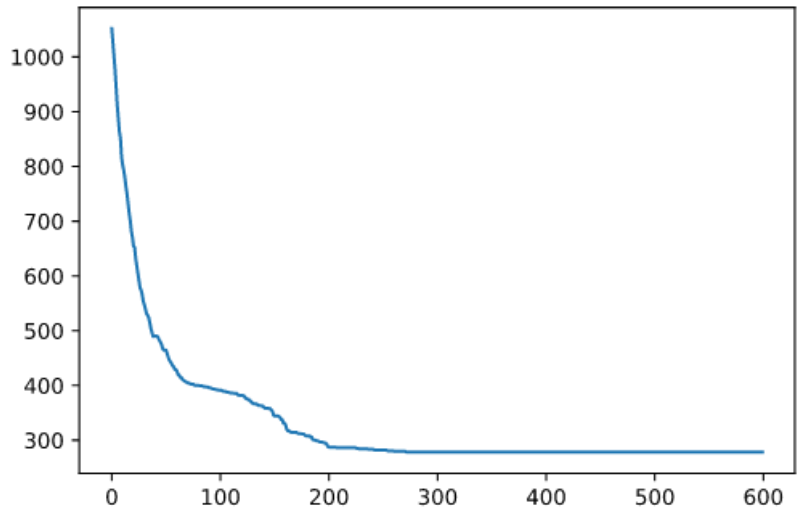
```
Finished transition 1
The sequence converges to 238.6920394841241
Achieved convergent at 46249 iteration

Finished transition 2
The sequence converges to 267.8322424631352
Achieved convergent at 897 iteration

Finished transition 3
The sequence converges to 278.0844680154938
Achieved convergent at 271 iteration
```

Combined Result



Results for Transition 2 & Transition 3



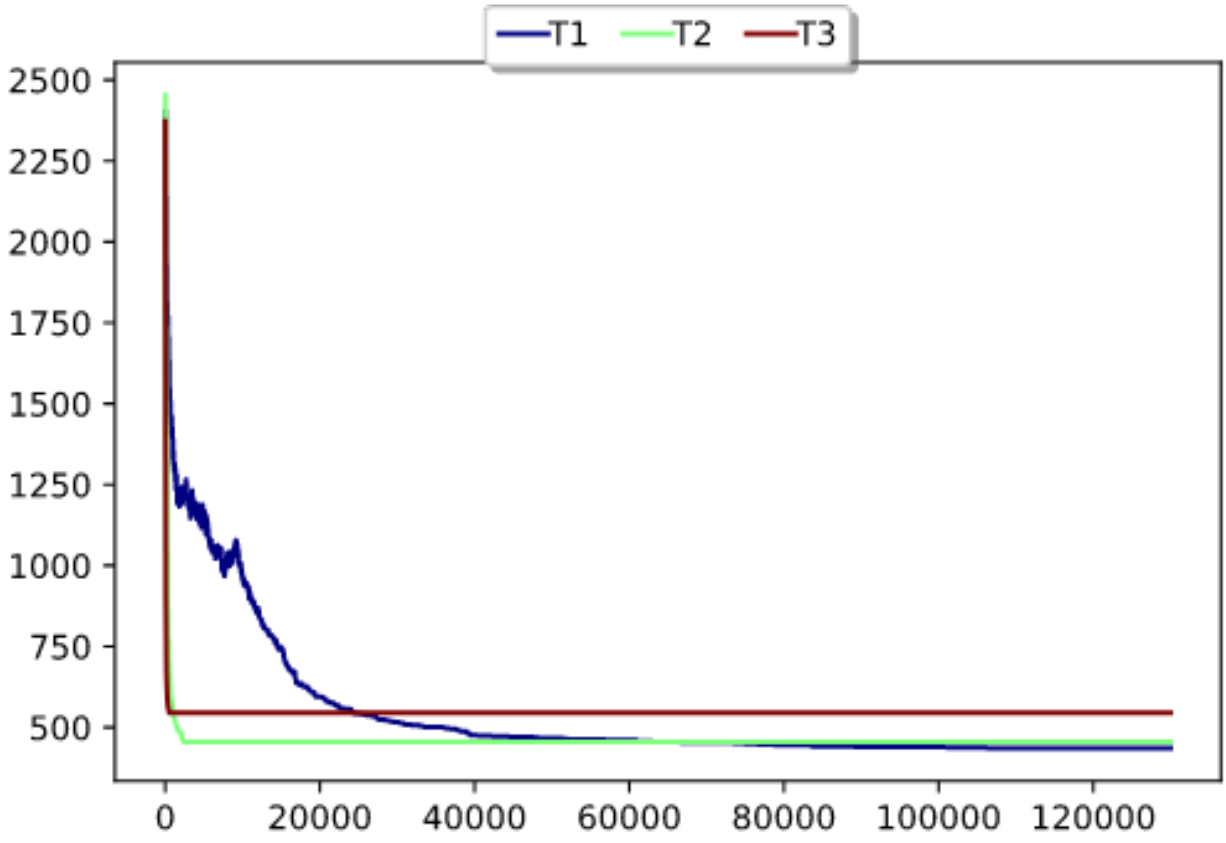


Vertices = 150

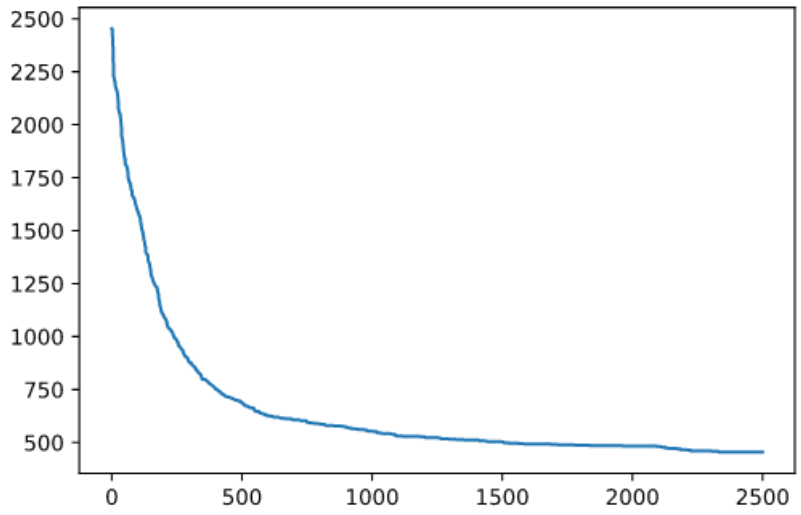
Numerical Result

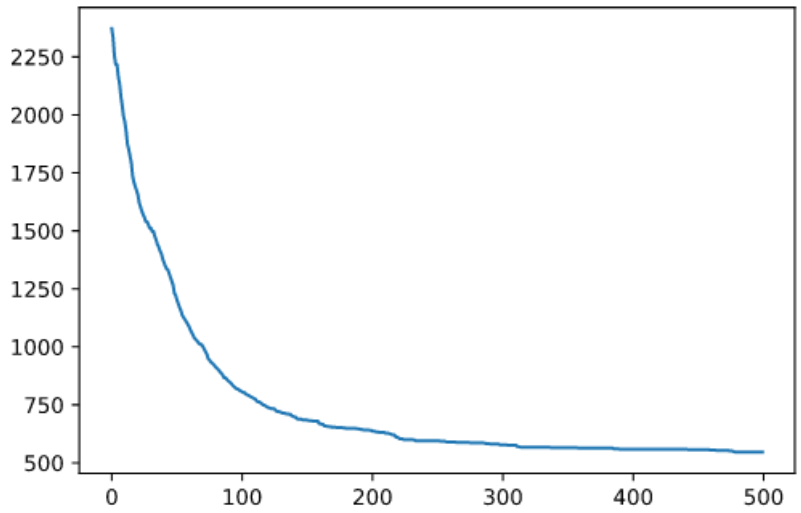
```
Finished transition 1
The sequence converges to 435.1697740643882
Achieved convergent at 118997 iteration

Finished transition 2
The sequence converges to 454.13587864981116
Achieved convergent at 2486 iteration

Finished transition 3
The sequence converges to 545.6245235419655
Achieved convergent at 489 iteration
```

Combined Result



Results for Transition 2 & Transition 3



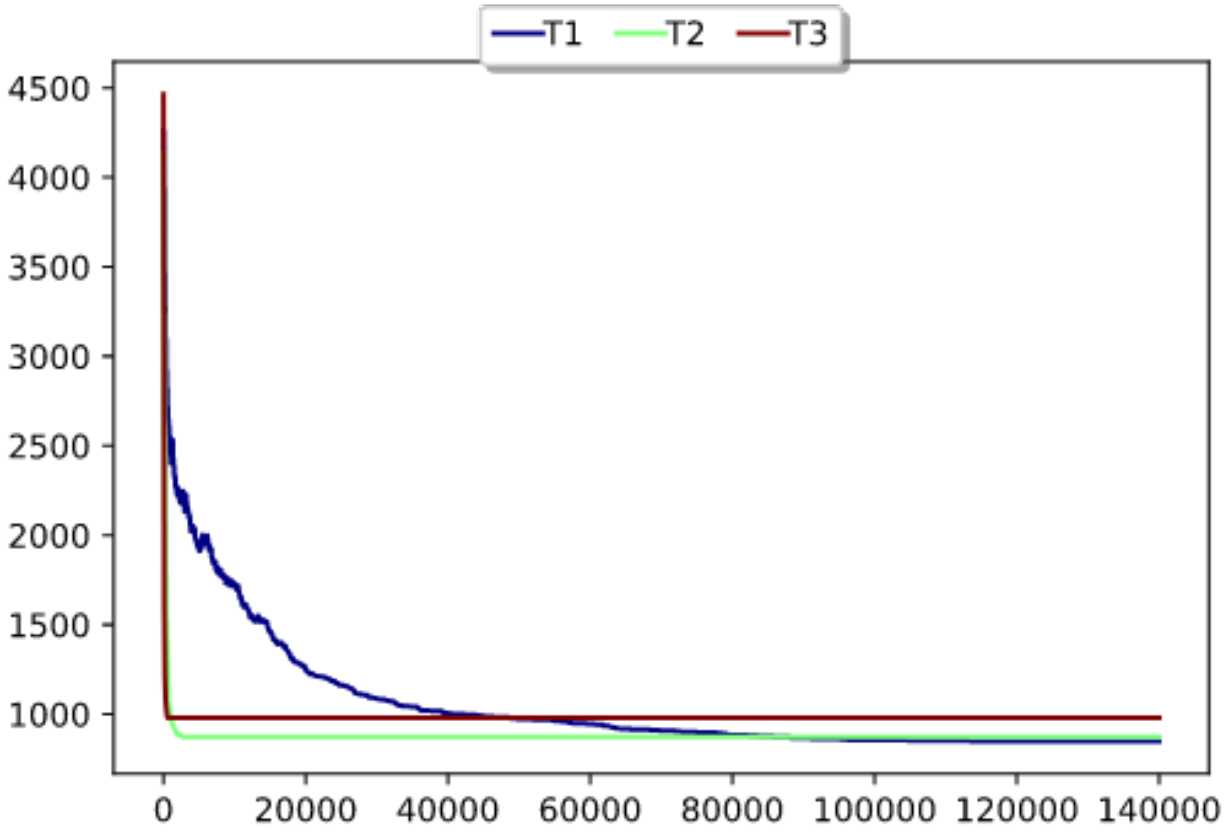


Vertices = 200

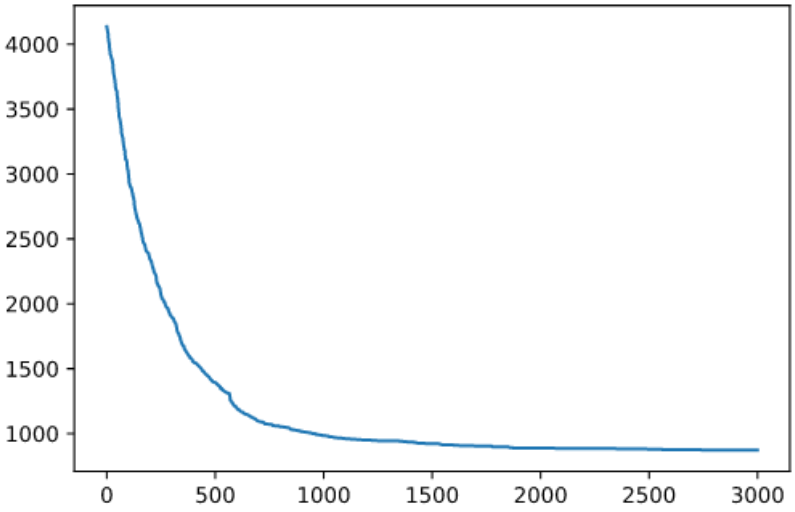
Numerical Result

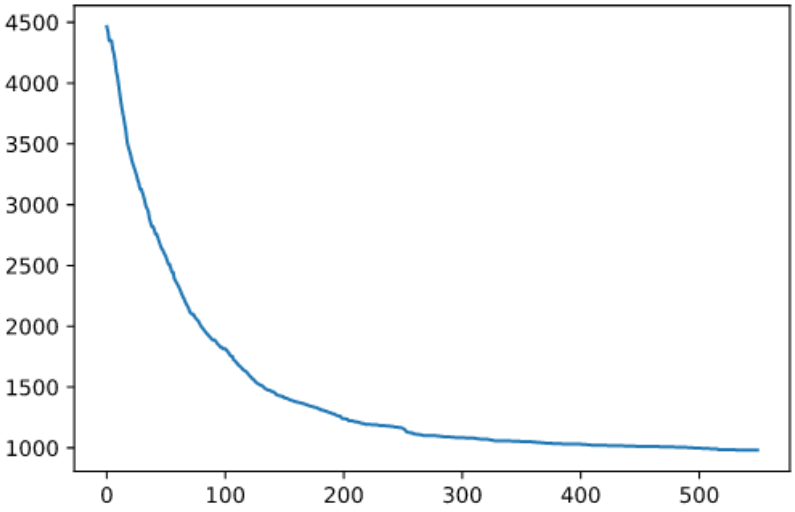
```
Finished transition 1
The sequence converges to 848.5888219401288
Achieved convergent at 131744 iteration

Finished transition 2
The sequence converges to 871.6472426358516
Achieved convergent at 2791 iteration

Finished transition 3
The sequence converges to 980.8444326040242
Achieved convergent at 540 iteration
```

Combined Result



Results for Transition 2 & Transition 3





Conclusion for Part 2

The number of iterations needed before convergence satisfiy: Transition 1 >> Transition 2 > Transition 3. Through arduous adjustment to the hyperparameters, I find that the annealing schedule should be more "mild" in swapping than reversing. In other words, swapping takes more iterations and a slower annealing process before convergence. Meanwhile, the convergence point of swapping is larger than reversing. Hence, reversing is more eligible to define neighborhood.

Part 3

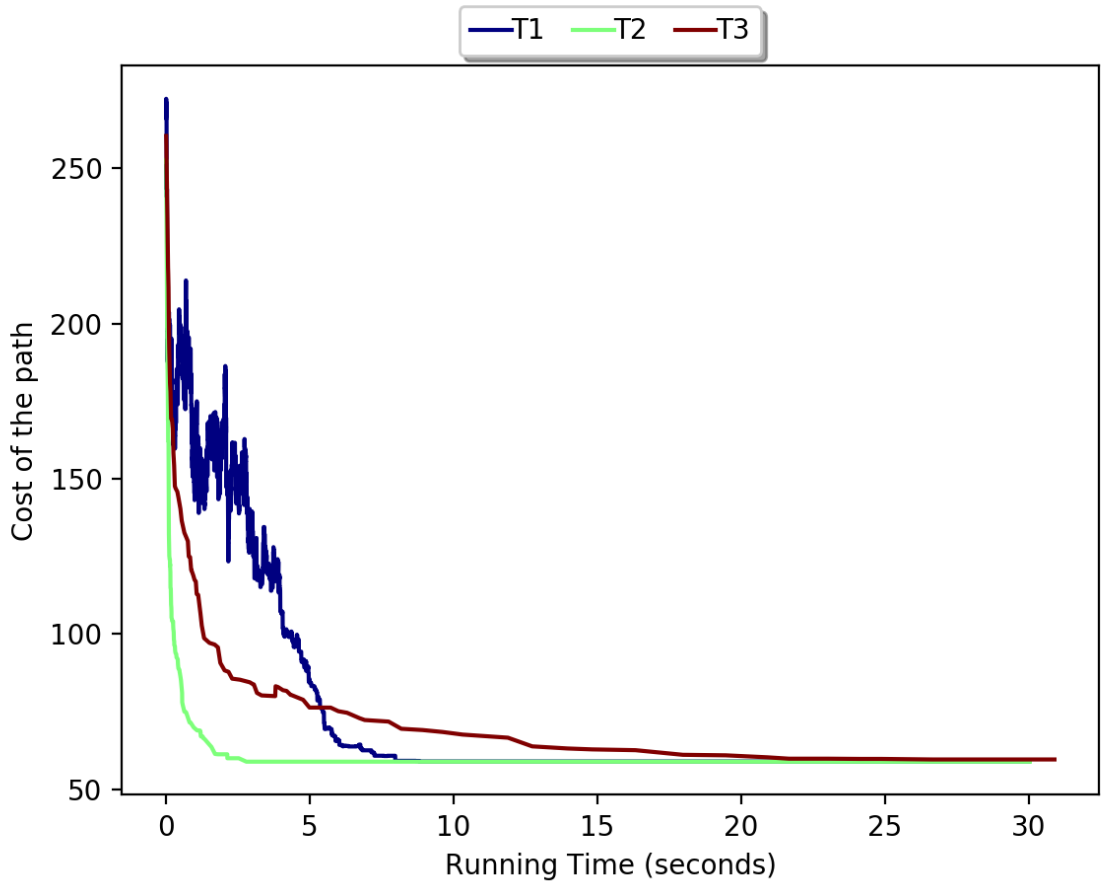
Now that the former experiments have shown the superiority of Transitions 2 & 3 in terms of iterations, we need further validation on the running time since the time required for one iteration varies a lot. In this part, we return to the **reversing** neighborhood. But this time, the x-axis will be changed from Iterations to Running Time. Literally, running time is the most crucial indicator of a good algorithm. We still apply the algorithms on the same set of graphs with vertices 50, 100, 150, 200. Here are the results.

Vertices = 50

Numerical Result

```
Finished transition 1. The number of iteration is: 36894
The sequence converges to 58.99115705942636
Achieved convergent at 8.812406063079834s

Finished transition 2. The number of iteration is: 206
The sequence converges to 58.921850444886516
Achieved convergent at 2.8095459938049316s

Finished transition 3. The number of iteration is: 87
The sequence converges to 60.37472418083472
Achieved convergent at 19.46391201019287s
```

Plot

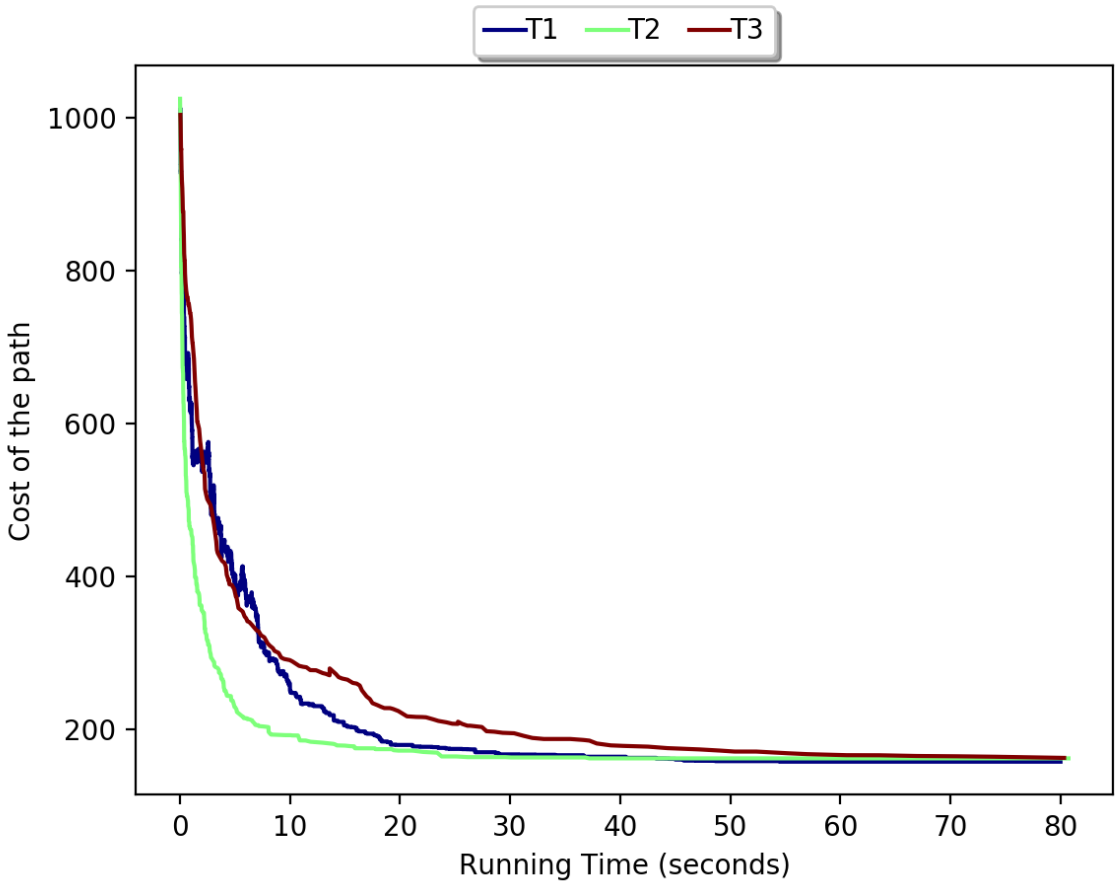


Vertices = 100

Numerical Result

```
Finished transition 1. The number of iteration is: 40084
The sequence converges to 157.72167779138036
Achieved convergent at 54.49112391471863s

Finished transition 2. The number of iteration is: 430
The sequence converges to 162.06955272959348
Achieved convergent at 36.942704916000366s

Finished transition 3. The number of iteration is: 174
The sequence converges to 166.75372020925028
Achieved convergent at 60.57168984413147s
```

Plot

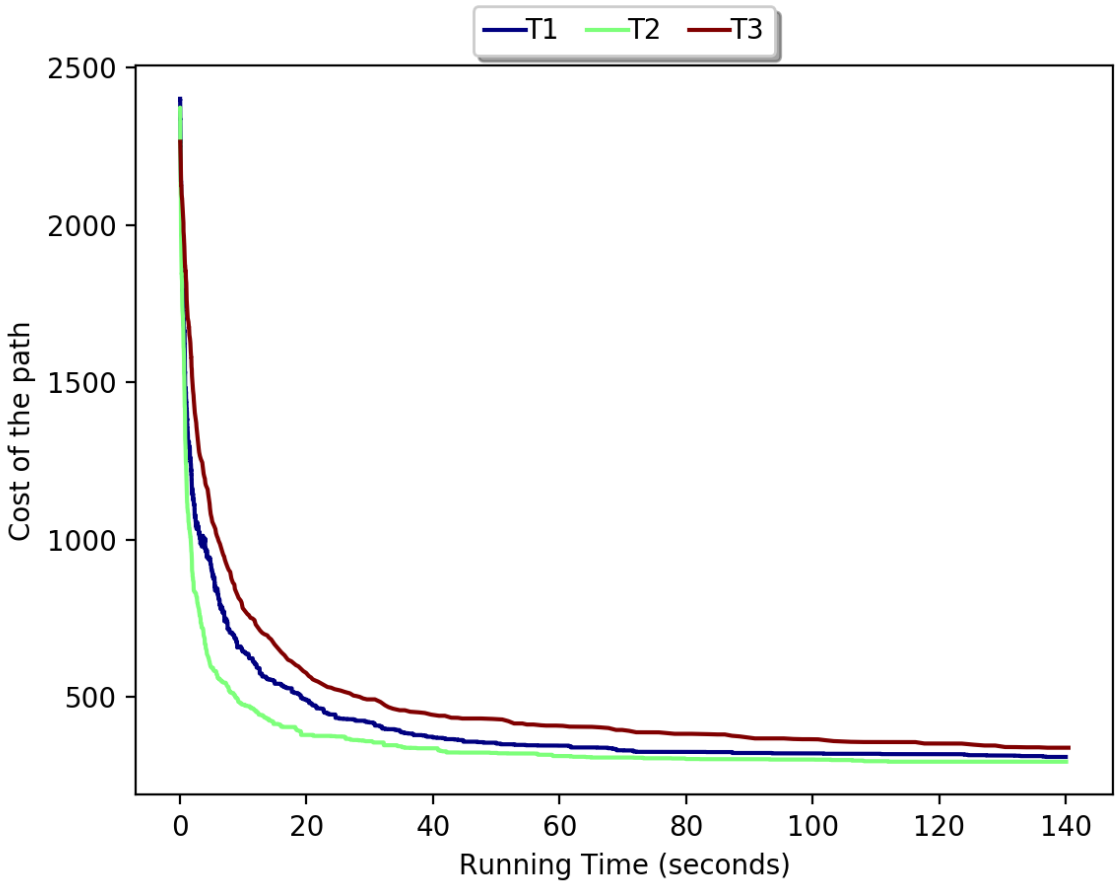


Vertices = 150

Numerical Result

```
Finished transition 1. The number of iteration is: 56187
The sequence converges to 309.55979693711566
Achieved convergent at 136.88966012001038s

Finished transition 2. The number of iteration is: 1501
The sequence converges to 294.49155765446795
Achieved convergent at 111.96345901489258s

Finished transition 3. The number of iteration is: 270
The sequence converges to 341.63962620888464
Achieved convergent at 130.40936398506165s
```

Plot

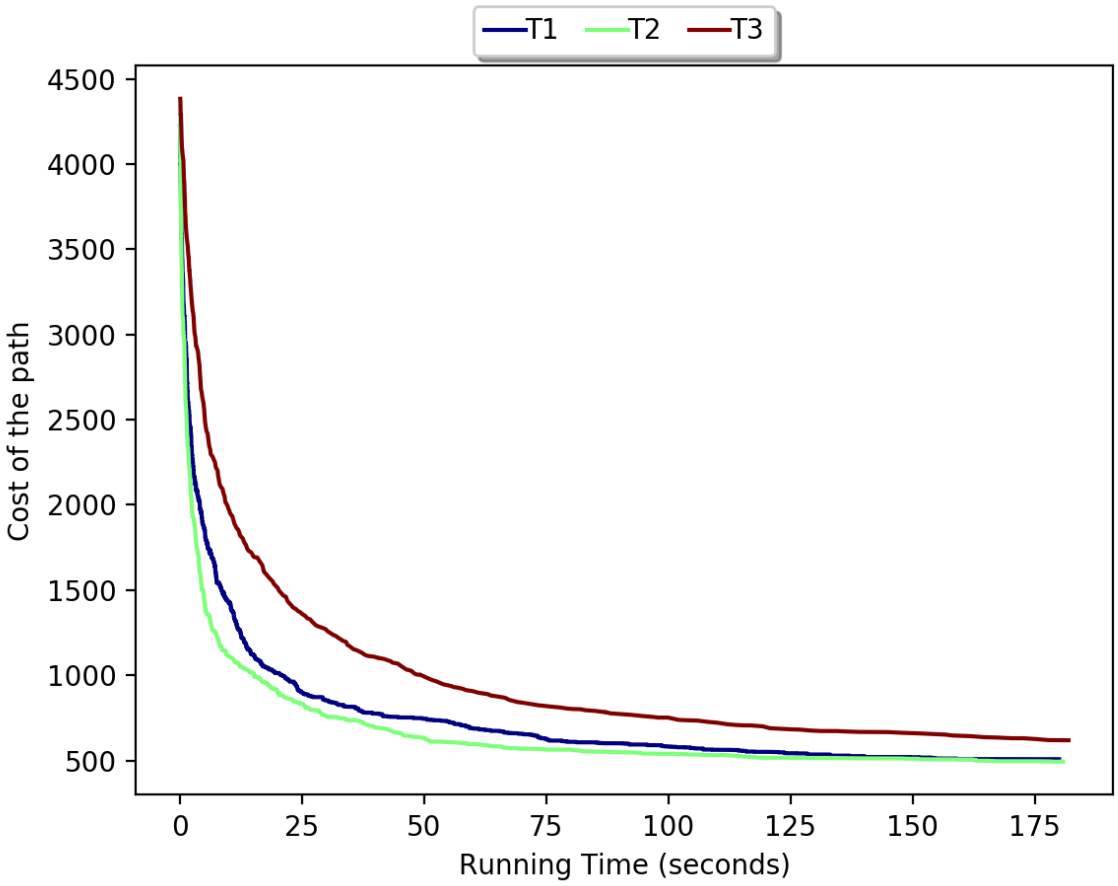


Vertices = 200

Numerical Result

```
Finished transition 1. The number of iteration is: 61597
The sequence converges to 507.7037073014054
Achieved convergent at 174.12559008598328s

Finished transition 2. The number of iteration is: 888
The sequence converges to 495.1247414696413
Achieved convergent at 167.2721071243286s

Finished transition 3. The number of iteration is: 280
The sequence converges to 630.7562056685595
Achieved convergent at 170.30010294914246s
```

Plot



Conclusion for Part 3

Using the same set of hyperparameters as Part 2, the Markov Chains have approximated the global minimums successfully. Referring to the plots, we can see that Transition 3 has weak competitiveness because it converges the slowest in terms of running time even if it takes the fewest number of iterations. But for Transition 2, the green curve (for Transition 2) in every plot decreases faster than the blue curve (for Transition 1), which indicates the efficiency in Transition 2. By checking the sequence of cost, the evident superiority of Transition 2 has been shown by both the smaller convergent point and the tremendous advantage in running time. In general, Transition 2 is better than Transition 1.

Part 4

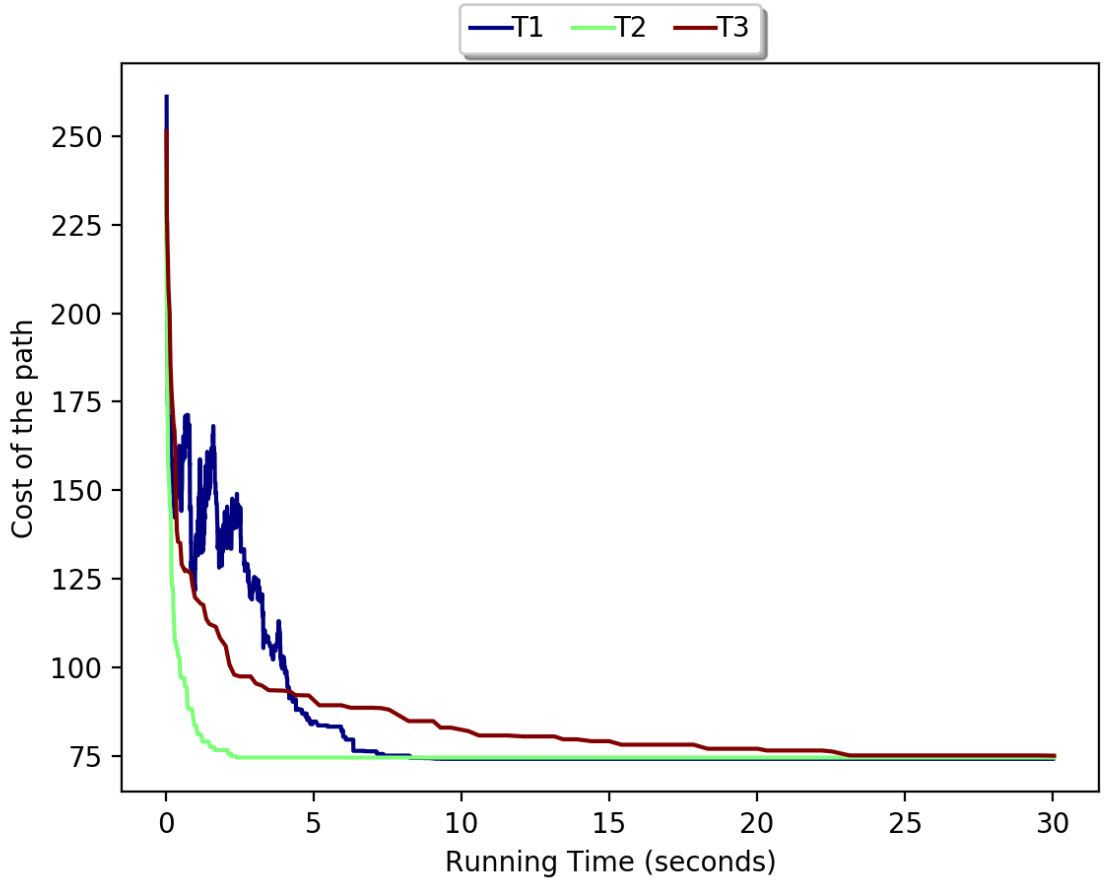
Formerly, I have shown that no matter how I adjust the hyperparameters, the performance of the Markov Chain is unsatisfactory. Thus, it is meaningless to use **swapping** as the definition of neighborhood. However, I still do experiments for this part in order to compare the superiority of the three transition mechanisms when the neighborhood definition is no longer "reversing". The objective of this part is to check whether the comparative advantage of different transition mechanisms varies among different neighborhood definitions. Here are the results.

Vertices = 50

Numerical Result

```
Finished transition 1. The number of iteration is: 46883
The sequence converges to 74.27872774135002
Achieved convergent at 8.25687313079834s

Finished transition 2. The number of iteration is: 684
The sequence converges to 74.64004169172019
Achieved convergent at 2.394709825515747s

Finished transition 3. The number of iteration is: 108
The sequence converges to 75.22993763189143
Achieved convergent at 23.12556505203247s
```

Plot

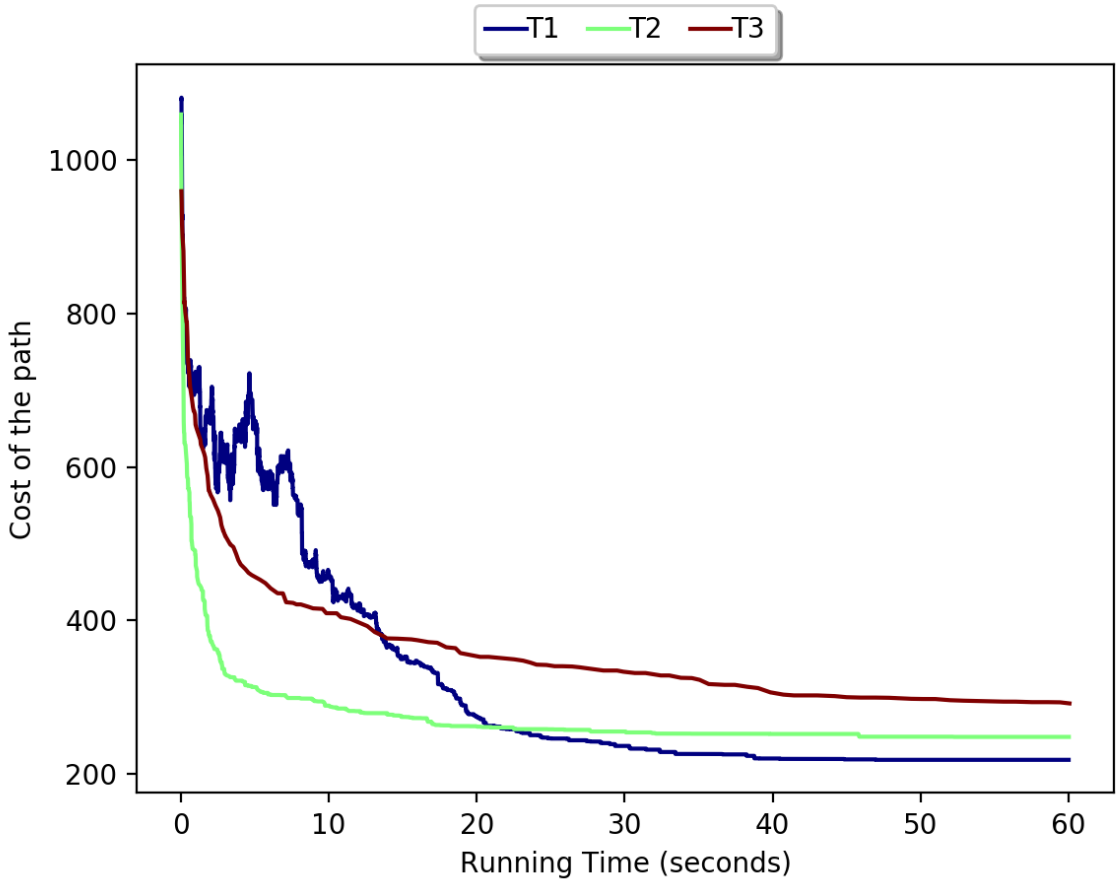


Vertices = 100

Numerical Result

```
Finished transition 1. The number of iteration is: 43941
The sequence converges to 218.52037102696062
Achieved convergent at 46.960736036300665s

Finished transition 2. The number of iteration is: 904
The sequence converges to 248.38172462998608
Achieved convergent at 45.850069999694824s

Finished transition 3. The number of iteration is: 148
The sequence converges to 294.3078109782663
Achieved convergent at 54.72653913497925s
```

Plot

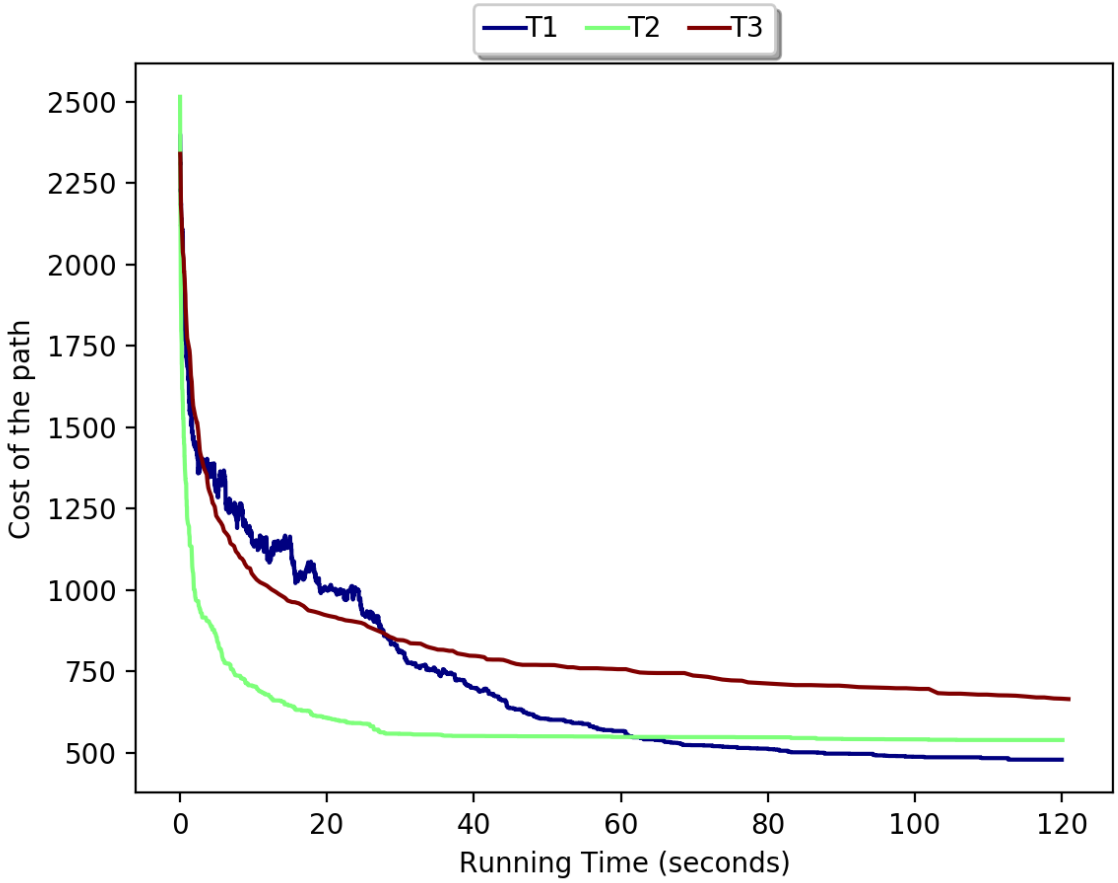


Vertices = 150

Numerical Result

```
Finished transition 1. The number of iteration is: 50688
The sequence converges to 478.76293164238984
Achieved convergent at 118.87909030914307s

Finished transition 2. The number of iteration is: 1302
The sequence converges to 539.1764067223468
Achieved convergent at 101.86787986755371s

Finished transition 3. The number of iteration is: 210
The sequence converges to 671.7748223519801
Achieved convergent at 116.52697587013245s
```

Plot

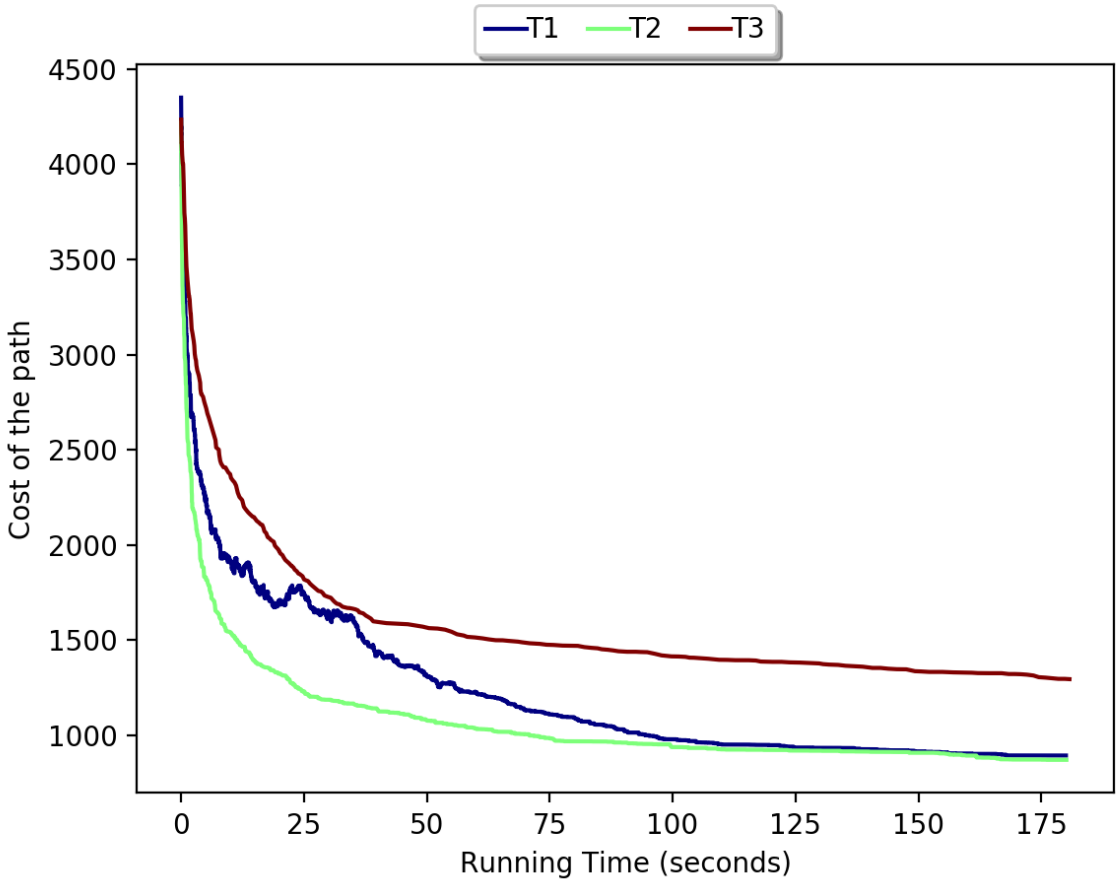


Vertices = 200

Numerical Result

```
Finished transition 1. The number of iteration is: 61588
The sequence converges to 894.4463167245215
Achieved convergent at 167.80365800857544s

Finished transition 2. The number of iteration is: 1546
The sequence converges to 871.8336636330816
Achieved convergent at 176.06053400039673s

Finished transition 3. The number of iteration is: 251
The sequence converges to 1305.0469329729808
Achieved convergent at 174.47721767425537s
```

Plot



Conclusion for Part 4

Even though the green curve (Transition 2) decreases faster than the blue curve (Transition 1), the numerical results show that Transition 1 performs better than Transition 2 in terms of both convergent point and running time under the "swapping" neighborhood. Therefore, we can conclude that the comparative advantage of different transition mechanisms varies among different neighborhood definitions.

Conclusions

According to the experiment results, we can summarize the following conclusions. I also provide my intuitive interpretation for each conclusion.

1. Number of iterations required before convergence: Transition 1 >> Transition 2 > Transition 3;

   Running time per iteration: Transition 3 > Transition 2 >> Transition 1.

   Interpretation: Probability measures have been added to Transitions 2 & 3, especially Transition 3 in which we allocate the weights to candidate states based on their costs. Thus, it takes much longer time for one single iteration in Transitions 2 & 3. However, since "more preference" is given to the neighboring states with smaller costs, the cost of current state descents faster. Thus, the efficiency of the two alternative transitions is higher.

2. In terms of neighborhood definition, "reverse" is better than "swap".

   Interpretation: In a permutation, we denote the connection between two cities as a "bond". When reversing a subsequence, the cost of the entire path will be changed by new arragements of 2 bonds. But for swapping, the cost will be changed by new arrangement of 4 bonds. Thus, we make a guesstimation here. Changing more bonds in a permutation makes the Markov Chain harder to get to a new state with a smaller cost because of the hedging effect. (i.e. The length of some bonds may decrease while the others increase.)

3. When using the "reverse" neighborhood, Transition 3 performs the worst while Transition 2 performs the best.

   Interpretation: Transition 3 is trapped by its huge computing complexity caused by the candidate permutation generation and negative softmax weight allocation. As for Transition 2, since we put more weight on those neighboring states with smaller costs, the cost decreases faster than Transition 1. Meanwhile, the positive probability of switching into a state with larger cost keeps the chain from falling into the local minima.