Task 1

I have chosen to use Two change as my neighbourhood. To generate a neighbour of R1 the algorithm first generates two non-equal random numbers between 0 and the number of participants that need to be ranked minus 1. The participants in positions specified by these two numbers are then swapped in the ranking. So, a neighbour R2 is a neighbour of R1 when two participants have swapped places in the ranking, whilst the rest of the ranking stays the same.

For example:

Below is the initial solution I'm using for this problem. It's a list of integers where an integer corresponds to a racer's number.

R1
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35]

Say the two random numbers chosen where 4 and 9 As the list is indexed from 0, that would mean racers ranked in 5th and 10th in the ranking will be swapped shown below in R2.

R2
[1, 2, 3, 4, **10**, 6, 7, 8, 9, **5**, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35]

Now to compute the new cost of the ranking, R2, we only need to consider the cost of the two swapped participants and the participants in between.

So, we first calculate the cost of the sub ranking [5, 6, 7, 8, 9, 10] from R1, say the cost of this is X.

Then we need to calculate the cost of the sub ranking [10, 6, 7, 8, 9, 5] from R1, say the cost of this is Y.

The cost of R2 is then just:

R2Cost = R1Cost - X + Y

This method results in far fewer computations when running the Simulated Annealing algorithm compared to if the entire cost of the ranking was to be calculated each time a neighbour was generated.

Task3

```
Lowest Kemeny score found: 64
timeTaken: 0.327
Position Candidate
```

Position	Candidate
1	Alain Prost
2	Niki Lauda
3	Rene Arnoux
4	Elio de Angelis
5	Corrado Fabi
6	Derek Warwick
7	Michele Alboreto
8	Nelson Piquet
9	Patrick Tambay
10	Andrea de Cesaris
11	Mauro Baldi
12	Teo Fabi
13	Thierry Boutsen
14	Riccardo Patrese
15	Gerhard Berger
16	Nigel Mansell
17	Jo Gartner
18	Keke Rosberg
19	Ayrton Senna
20	Eddie Cheever
21	Jonathan Palmer
22	Marc Surer
23	Martin Brundle
24	Huub Rothengatter
25	Jacques Laffite
26	Stefan Bellof
27	Francois Hesnault
28	Stefan Johansson
29	Piercarlo Ghinzani
30	Manfred Winkelhock
31	Johnny Cecotto
32	Philippe Alliot
33	Mike Thackwell
34	Pierluigi Martini
35	Philippe Streiff

```
Lowest Kemeny score found: 63
timeTaken: 0.494
Position Candidate
```

Position	Candidate
1	Alain Prost
2	Niki Lauda
3	Elio de Angelis
4	Rene Arnoux
5	Corrado Fabi
6	Derek Warwick
7	Michele Alboreto
8	Nelson Piquet
9	Patrick Tambay
10	Andrea de Cesaris
11	Mauro Baldi
12	Teo Fabi
13	Thierry Boutsen
14	Riccardo Patrese
15	Gerhard Berger
16	Nigel Mansell
17	Jo Gartner
18	Keke Rosberg
19	Ayrton Senna
20	Eddie Cheever
21	Marc Surer
22	Jonathan Palmer
23	Martin Brundle
24	Huub Rothengatter
25	Jacques Laffite
26	Stefan Bellof
27	Francois Hesnault
28	Stefan Johansson
29	Piercarlo Ghinzani
30	Manfred Winkelhock
31	Johnny Cecotto
32	Philippe Alliot
33	Mike Thackwell
34	Pierluigi Martini
35	Philippe Streiff

```
Lowest Kemeny score found: 63
timeTaken: 0.822
Position Candidate
```

Position	Candidate
1	Alain Prost
2	Niki Lauda
3	Elio de Angelis
4	Rene Arnoux
5	Corrado Fabi
6	Michele Alboreto
7	Derek Warwick
8	Nelson Piquet
9	Patrick Tambay
10	Andrea de Cesaris
11	Mauro Baldi
12	Teo Fabi
13	Thierry Boutsen
14	Riccardo Patrese
15	Gerhard Berger
16	Nigel Mansell
17	Jo Gartner
18	Keke Rosberg
19	Ayrton Senna
20	Eddie Cheever
21	Marc Surer
22	Jonathan Palmer
23	Martin Brundle
24	Huub Rothengatter
25	Jacques Laffite
26	Stefan Bellof
27	Francois Hesnault
28	Stefan Johansson
29	Piercarlo Ghinzani
30	Manfred Winkelhock
31	Johnny Cecotto
32	Philippe Alliot
33	Pierluigi Martini
34	Mike Thackwell
35	Philippe Streiff

First, I ran the simulated annealing algorithm 10000 times with random values for the parameters, with the ranges shown below.

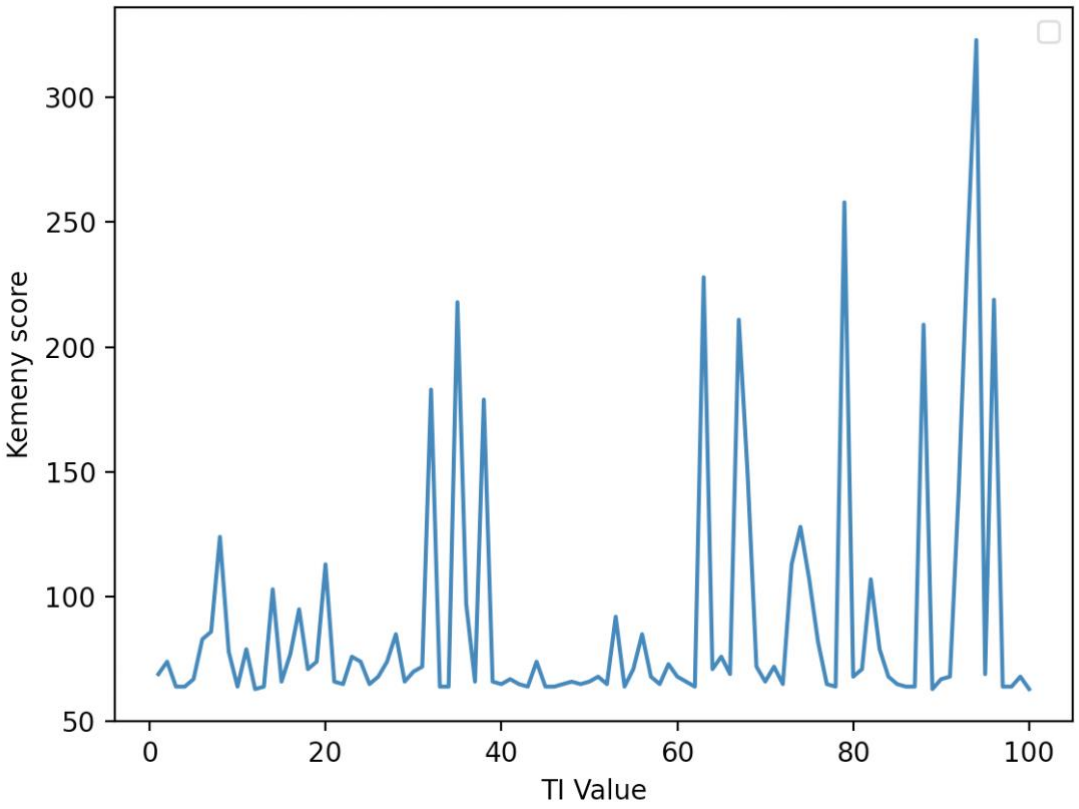
```
TI = random.randint(1, 100)
TL = random.randint(1, 10000)
a = random.uniform(0.8, 0.99)
num_non_improve = random.randint(1, 10000)
```

I sorted all the results by the Kemeny score each ranking achieved. I took 1000 of the rankings which achieved the lowest Kemeny score.

I took the average of each of the parameters which returned:

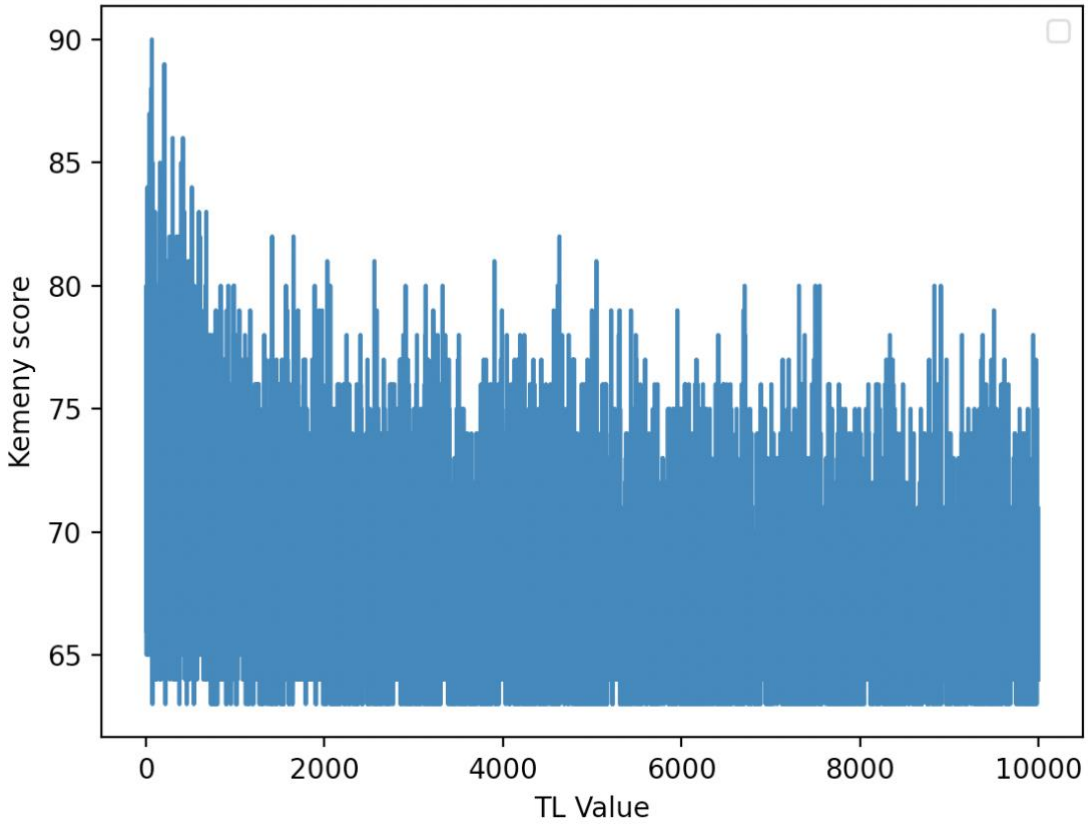
TI = 43
TL = 1781
a = 0.866
num_non_improve = 6919

using values from 1 to 100 for TI and keeping the other variables the same I found lower values tend to have lower Kemeny scores shown in the graph below.

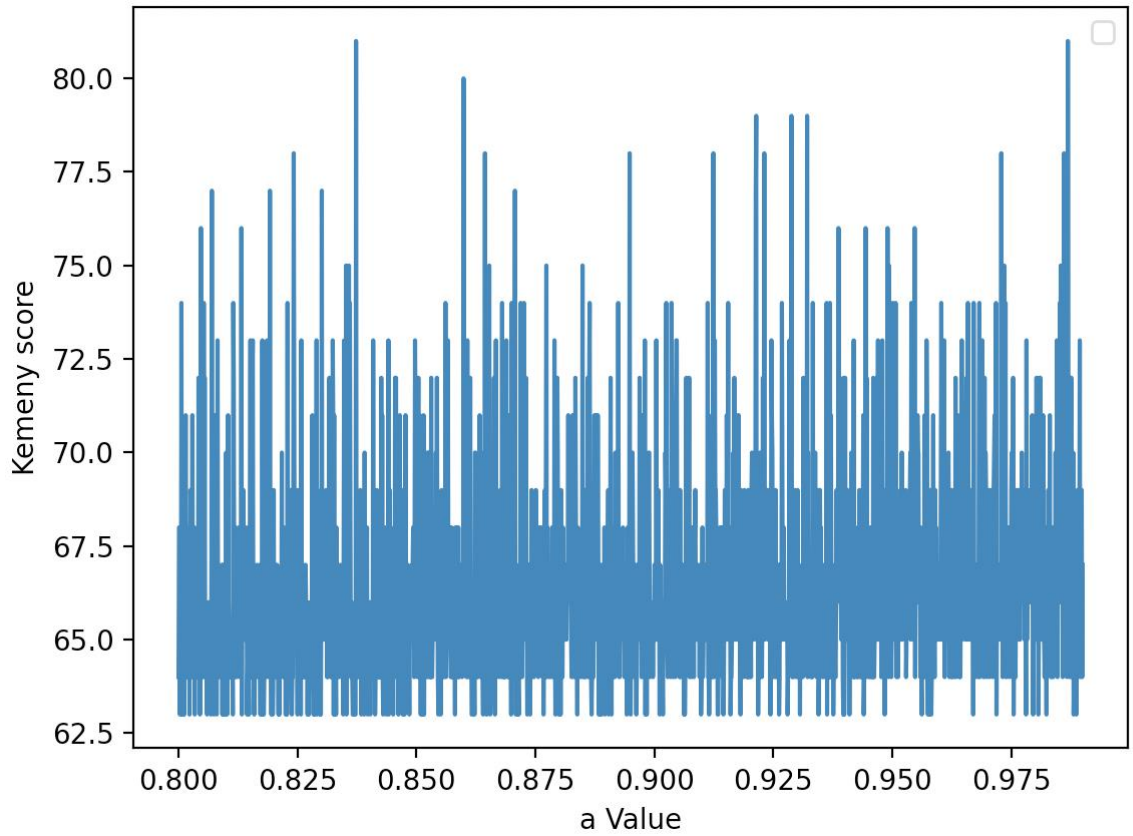


As can be seen in the graph higher values of TI can achieve low scores but have a high chance of drastically higher Kemeny scores.

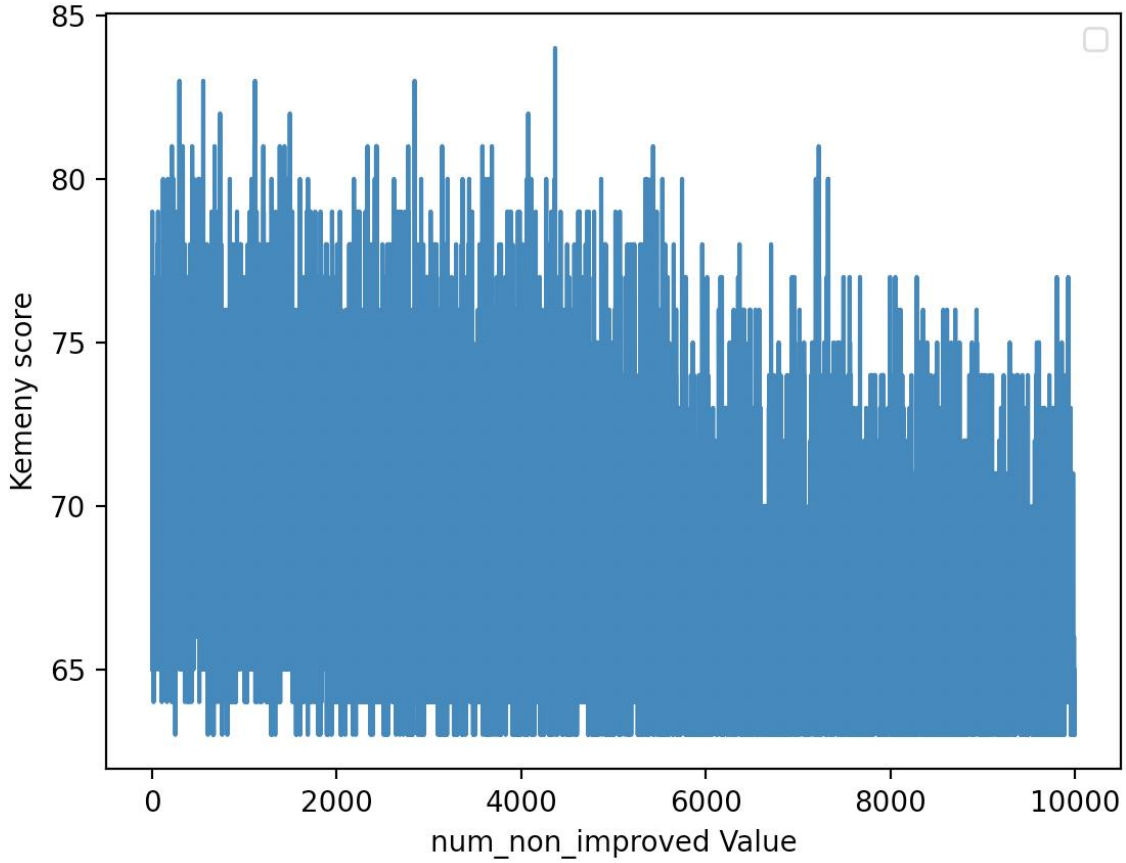
I then repeated this same process of fixing other variables while generating a range of one.



From this graph it seems TL doesn't affect the Kemeny score a lot but tends to work slightly better for higher values.



Kemeny scores are quite consistent in this range of a. Low scores are slightly denser between 0.800 and 0.825



The num non improved value tends to result in better scores the larger it is given the range shown above.

From the graphs above it can be seen that TI effected the performance of the algorithm the most. From these experiments I have decided to use the values show below.

```
TI = 1
TL = 9000
a = 0.825
num_non_improve = 9000
```

To get out of a local optimum the algorithm needs to accept a new solution even if it's worse than the current. When I count these moves there are about 150 per run. This suggests a few local optima.