Ostrakov Oleksii

UI task 2

Variant a, c

My task is creating 2 algorithms that practically trying to solve travelling salesman problem. The travelling salesman problem (also called the travelling salesperson problem or TSP) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?" It is an NP-hard problem in combinatorial optimization, important in theoretical computer science and operations research. In the theory of computational complexity, the decision version of the TSP belongs to the class of NP-complete problems. Thus, it is possible that the worst-case running time for any algorithm for the TSP increases superpolynomially (but no more than exponentially) with the number of cities.

The first algorithm is genetic algorithm. Genetic algorithm is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms. Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection. Some examples of genetic algorithm applications include optimizing decision trees for better performance, solving sudoku puzzles, hyperparameter optimization, etc.

The second algorithm is simulated annealing. Simulated annealing is a probabilistic technique for approximating the global optimum of a given function. Specifically, it is a metaheuristic to approximate global optimization in a large search space for an optimization problem. The name of the algorithm comes from annealing in metallurgy, a technique involving heating and controlled cooling of a material to alter its physical properties. Both are attributes of the material that depend on their thermodynamic free energy. Heating and cooling the material affects both the temperature and the thermodynamic free energy or Gibbs energy. Simulated annealing can be used for very hard computational optimization problems where exact algorithms fail; even though it usually achieves an approximate solution to the global minimum, it could be enough for many practical problems.

Implementation

Both algorithms have some things in common. They have **City** class that represents a city that in TSP is visited. This class has attributes x for x coordinate, y for y coordinate and name of the city. Also, both of them have **Route** class that represents a route that TSP is using to visit every city exactly one time. It has array of **Cities** and a length of a full route. It is calculated using the Pythagorean theorem. Also, it has **recalculate_distance()** function to be able to change the length of the route if it changes the order of cities.

The genetic algorithm has a **Generation** class that represents the first algorithm. It has specified amount of a **Route** objects, several attributes to count like level of generation, the length of current best route etc. Also it has **find_best_way()** function that triggers this algorithm. The program works as follows: it gets several important inputs like amount of cities, the list of all cities and their coordinates, generation limit (maximum number of new generation without change), amount of examples to mutate and chosen way of mutation (1 – replace 10% of route with a random cities, 2 – replace 10% of route with next cities). After that algorithm creates a specified amount of examples that have a random route. Then it starts endless cycle where it mutates every route and then checks its length of route. If it finds a new best route, it saves it and refreshes the counter of generations. If generation counter reaches its limit and the new best route wasn't generated before this, program ends. After that it prints best route, order of cities to achieve that length, the number of mutator that achieved this result and a mutation level on what this result was achieved.

The simulated annealing has an **Annealing** class that represents the second algorithm. It has specified amount of a **Route** objects, several attributes to count like temperature, length of annealing, cooling amount etc. Also, program has **forge()** function that triggers this algorithm. The program works as follows: it gets several important inputs like amount of cities, the list of all cities and their coordinates, starting temperature, length of annealing at specific temperature, temperature cooling (amount of degrees to lose every cycle) and amount of possible candidates to be annealed to. Then program starts: it creates a loop where it starts with maximum given temperature and slowly cool downs to 0, for every temperature it creates a given amount of annealing tries. Every try it generates a given amount of candidates that are a little changed from current route. If a new candidate has shorter route, it 100% saves it as best route. If there are no shorter candidate, it chooses a longer candidate with less than 100% chance. This chance is dependent on temperature and lower the temperature is, the lesser chance it will choose longer candidate. If no candidate was chosen for a

whole annealing try, program stops. After that it prints best route, order of cities in that route, the number of temperature it was achieved on and number of proses it was achieved on.

Comparing algorithms

Genetic algorithm is simple and average algorithm for TSP task. It has simple cycle, good result and fast completion. It doesn't have any major flaws in its concept as it doesn't have ani big advantage among other TSP algorithms. From two mutation ways that my program is using the second one (replace 10% of route with next cities) is more preferable as it has higher chances to potentially upgrade current route and not to ruin it.

Simulated annealing algorithm is quite more complex in its concept. This algorithm is better used in cases where its need to find more optimal solution rather than exact and only right answer. Algorithm is more precise when in can get close to a local extremum with right answer, which makes its result almost perfect. However, if it's not getting close to a right answer at the start, then it can stuck at wrong extremum and be less precise than other algorithms.

As this algorithm is critical on selecting right starting point, the possible way to upgrade this algo is to have several potential starting point or to have another algorithm to select potentially good starting point.

Examples

In my project it is possible to work with 20 and up to 40 cities with different names in 200 x 200 size, but for these examples 20 cities will be used with simple location and names: 1st city has x coordinate = 1, y coordinate = 1 and name = 1, 2nd city has x coordinate = 2, y coordinate = 2 and name = 2 etc. 20th city has x coordinate = 20, y coordinate = 2 and name = 20. The best route for this city is to go from city 1 to city 2, from city 2 to city 3 etc. from city 19 to city 20 and lastly from city 20 to city 1. This will takes us $19 \times \sqrt{1+1} + \sqrt{2 \times 19^2} = 19\sqrt{2} + 19\sqrt{2} = 38\sqrt{2} = 53{,}7401154$. This is the best achievable route in this example.

Genetic algorithm:

Default: limit – 5000, examples – 500, mutation way – 2

```
result:
best length:  84.85281374238569
city order by name:
3  ( 3 ,  3 )
8  ( 8 ,  8 )
9  ( 9 ,  9 )
1  ( 1 ,  1 )
11  ( 11 ,  11 )
17  ( 17 ,  17 )
4  ( 4 ,  4 )
15  ( 15 ,  15 )
19  ( 19 ,  19 )
14  ( 14 ,  14 )
6  ( 6 ,  6 )
13  ( 13 ,  13 )
5  ( 5 ,  5 )
16  ( 16 ,  16 )
12  ( 12 ,  12 )
10  ( 10 ,  10 )
18  ( 18 ,  18 )
20  ( 20 ,  20 )
2  ( 2 ,  2 )
7  ( 7 ,  7 )
the process was done in  87.78616450005211  seconds.
was achieved by mutator number  234  on mutation level  7615
```

1) Limit

Limit here helps to sharpen the answer. Too little limit will not allow program to make good enough answer while too big limit will take more time. Here are examples with limit 1000, 5000, 10000:

1000 mutation 1:

```
best length:  82.02438661763951
the process was done in  7.62040639994666  seconds.
was achieved by mutator number  428  on mutation level  47


best length:  87.68124086713188
the process was done in  16.529564899974503  seconds.
was achieved by mutator number  201  on mutation level  1352

best length:  87.68124086713188
the process was done in  7.7267888999776915  seconds.
was achieved by mutator number  60  on mutation level  74
```

1000 mutation 2:

```
best length:  82.02438661763951
the process was done in  8.791067600017413  seconds.
was achieved by mutator number  148  on mutation level  264

best length:  73.53910524340094
the process was done in  31.253236000076868  seconds.
was achieved by mutator number  119  on mutation level  3519

best length:  87.68124086713188
the process was done in  7.219698300003074  seconds.
was achieved by mutator number  279  on mutation level  58
```

5000 mutation 1:

```
best length:  84.85281374238566
the process was done in  38.959360099979676  seconds.
was achieved by mutator number  451  on mutation level  60
```

```
best length:  76.36753236814712
the process was done in  59.2378241000697  seconds.
was achieved by mutator number  167  on mutation level  2392

best length:  76.36753236814714
the process was done in  64.99229060008656  seconds.
was achieved by mutator number  62  on mutation level  3258
```

5000 mutation 2:

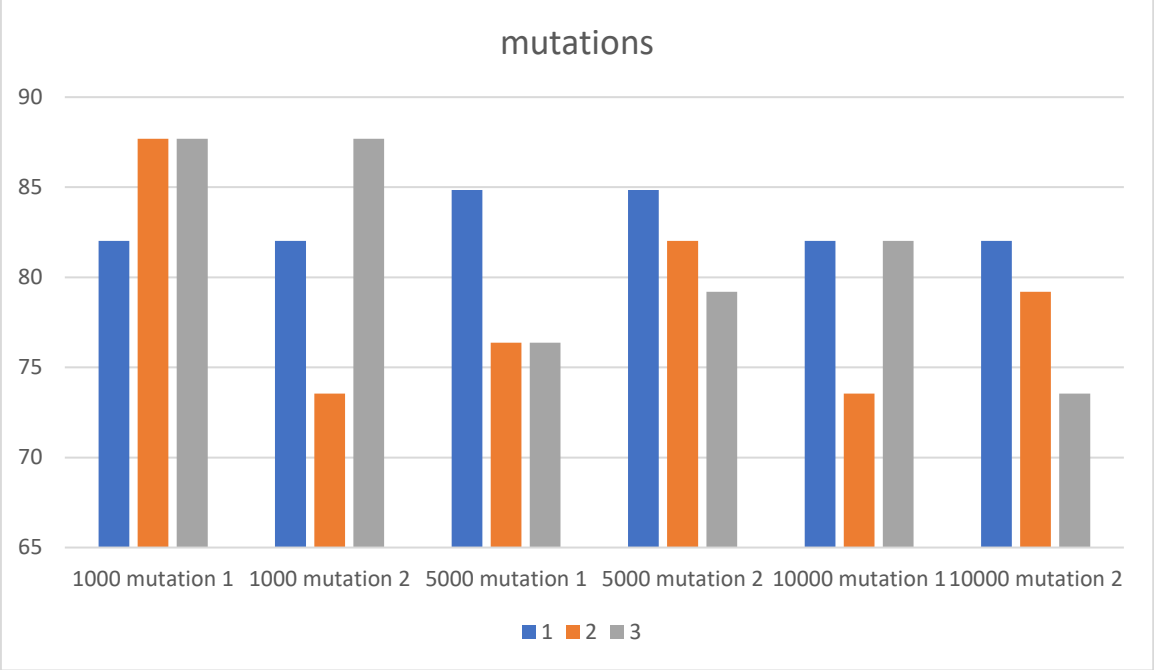
```
best length:  84.85281374238569
the process was done in  53.124261100078  seconds.
was achieved by mutator number  245  on mutation level  2253

best length:  82.0243866176395
the process was done in  54.02578570006881  seconds.
was achieved by mutator number  333  on mutation level  2226

best length:  79.19595949289332
the process was done in  37.06929899996612  seconds.
was achieved by mutator number  195  on mutation level  15
```

10000 mutation 1:

```
best length:  82.0243866176395
the process was done in  76.99635339993984  seconds.
was achieved by mutator number  193  on mutation level  12

best length:  73.53910524340094
the process was done in  98.58904869994149  seconds.
was achieved by mutator number  61  on mutation level  2885

best length:  82.0243866176395
the process was done in  174.01374009996653  seconds.
was achieved by mutator number  145  on mutation level  12960
```

10000 mutation 2:

```
best length:  82.0243866176395
the process was done in  124.37713609996717  seconds.
was achieved by mutator number  318  on mutation level  7623

best length:  79.19595949289332
the process was done in  83.76854840002488  seconds.
was achieved by mutator number  68  on mutation level  1683

best length:  73.53910524340093
the process was done in  195.98606809997  seconds.
was achieved by mutator number  17  on mutation level  17518
```



2) Examples

Examples are objects that are being mutated. Every example is created randomly, so there is a chance that some examples will be closer to right answer. The more examples there are, the more chances that shorter variant will be found, but it would take more time.

100 examples:

```
best length:  93.33809511662426
the process was done in  9.525797500042245  seconds.
```

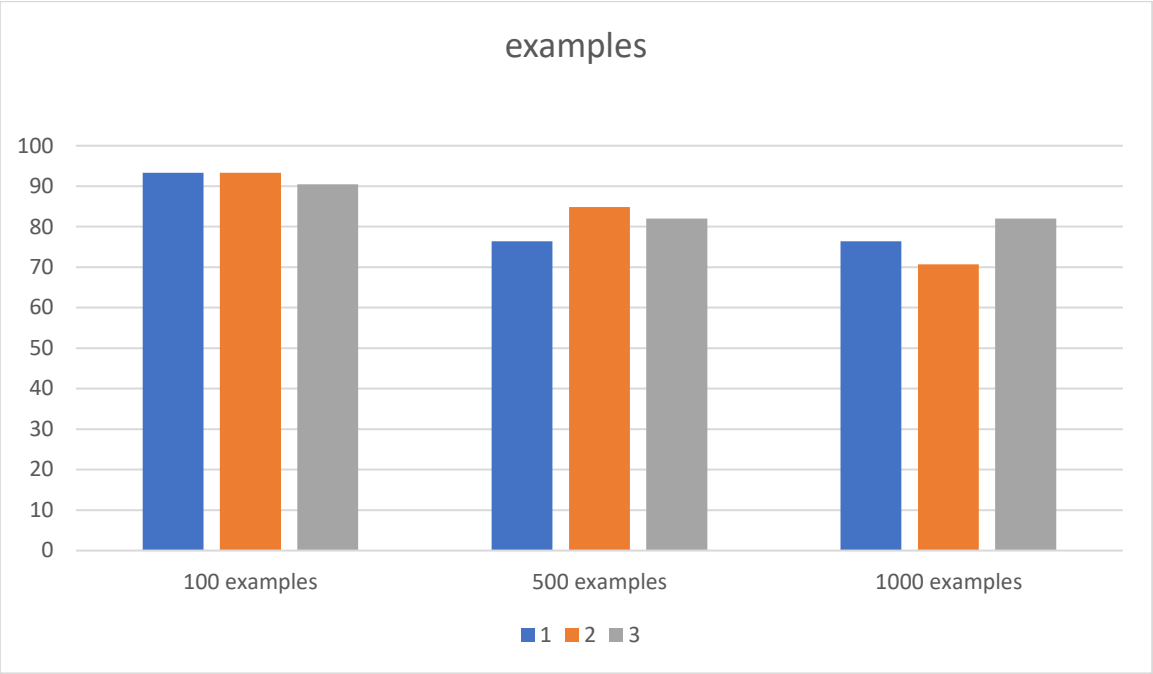
```
was achieved by mutator number  13  on mutation level  2023

best length:  93.33809511662426
the process was done in  12.315672199940309  seconds.
was achieved by mutator number  1  on mutation level  4094

best length:  90.50966799187805
the process was done in  15.668072199914604  seconds.
was achieved by mutator number  73  on mutation level  6771
```

500 examples:

```
best length:  76.36753236814712
the process was done in  37.87379440001678  seconds.
was achieved by mutator number  368  on mutation level  373

best length:  84.85281374238569
the process was done in  47.66849389998242  seconds.
was achieved by mutator number  163  on mutation level  1585

best length:  82.0243866176395
the process was done in  54.21290980000049  seconds.
was achieved by mutator number  200  on mutation level  2733
```

1000 examples:

```
best length:  76.36753236814712
the process was done in  257.27930439997  seconds.
was achieved by mutator number  535  on mutation level  13238

best length:  70.71067811865476
the process was done in  149.7141151999822  seconds.
was achieved by mutator number  103  on mutation level  5468

best length:  82.0243866176395
the process was done in  72.62356470001396  seconds.
was achieved by mutator number  624  on mutation level  60
```



Simulated annealing

Example:

```
result:
best length:  76.36753236814711
the process was done in  53.01391410001088  seconds.
was achieved on temperature  9  at process number  489
city order by name:
8  ( 8 ,  8 )
5  ( 5 ,  5 )
6  ( 6 ,  6 )
7  ( 7 ,  7 )
9  ( 9 ,  9 )
11  ( 11 ,  11 )
10  ( 10 ,  10 )
12  ( 12 ,  12 )
1  ( 1 ,  1 )
2  ( 2 ,  2 )
3  ( 3 ,  3 )
4  ( 4 ,  4 )
14  ( 14 ,  14 )
15  ( 15 ,  15 )
```

```
16  ( 16 ,  16 )
17  ( 17 ,  17 )
19  ( 19 ,  19 )
20  ( 20 ,  20 )
18  ( 18 ,  18 )
13  ( 13 ,  13 )
```

1) Temperature

Temperature is main factor in this algorithm. Temperature cab be from 0 to 100 and it represents the chance for longer node to be selected. The bigger temperature is, the more chances that algorithm won't stuck in wrong extremum but the answer wouldn't be so close to extremum.

Temperature 10, cooling 1:

```
best length:  110.3086578651014
the process was done in  2.5312362998956814  seconds.
was achieved on temperature  10  at process number  352

best length:  110.3086578651014
the process was done in  0.6885221999837086  seconds.
was achieved on temperature  10  at process number  100

best length:  79.1959594928933
the process was done in  8.38124020001851  seconds.
was achieved on temperature  9  at process number  196
```

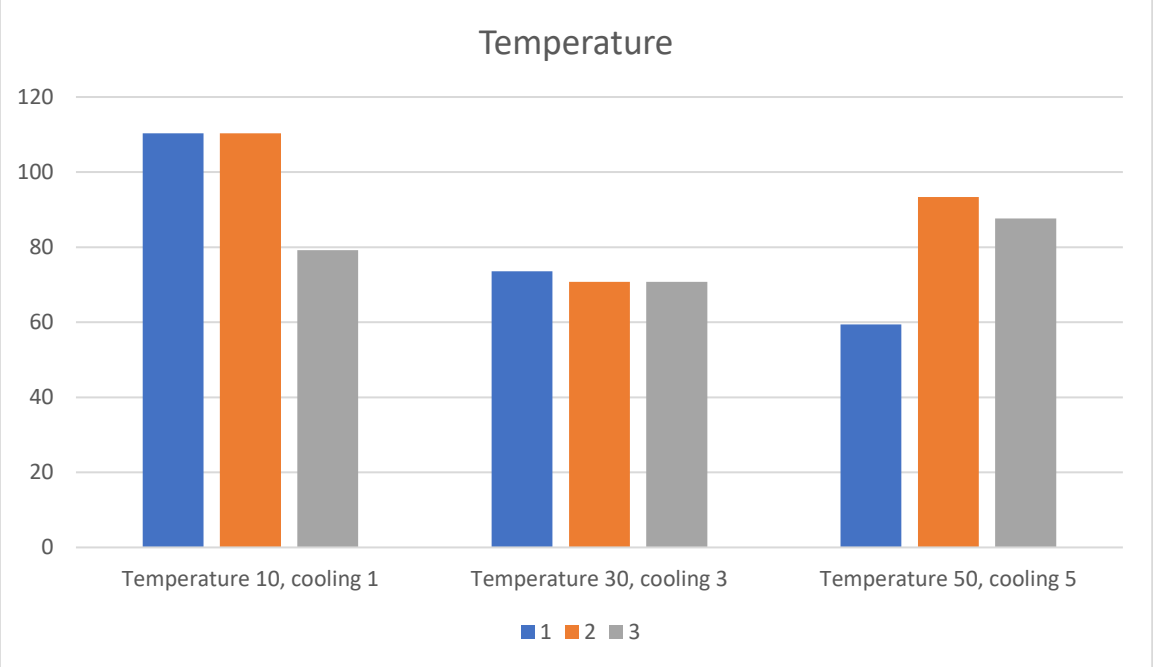
Temperature 30, cooling 3:

```
best length:  73.53910524340094
the process was done in  56.504194600041956  seconds.
was achieved on temperature  9  at process number  793

best length:  70.71067811865474
the process was done in  53.949032299919054  seconds.
was achieved on temperature  9  at process number  590
```

```
best length:  70.71067811865473
the process was done in  56.89982340007555  seconds.
was achieved on temperature  9  at process number  990
```

Temperature 50, cooling 5:

```
best length:  59.39696961966998
the process was done in  65.6988317000214  seconds.
was achieved on temperature  5  at process number  45

best length:  93.33809511662427
the process was done in  57.02903520001564  seconds.
was achieved on temperature  10  at process number  84

best length:  87.68124086713188
the process was done in  59.92233149998356  seconds.
was achieved on temperature  10  at process number  464
```



2) Length of annealing

Length is amount of times annealing happens on current temperature. Increasing length is potentially makes route more accurate, but it increases amount of operations needed.

Length 100:

```
best length:  56.568542494923804
the process was done in  5.502496900036931  seconds.
was achieved on temperature  6  at process number  13

best length:  67.88225099390854
the process was done in  5.736875499947928  seconds.
was achieved on temperature  6  at process number  34

best length:  65.05382386916236
the process was done in  6.116977900033817  seconds.
was achieved on temperature  6  at process number  91
```

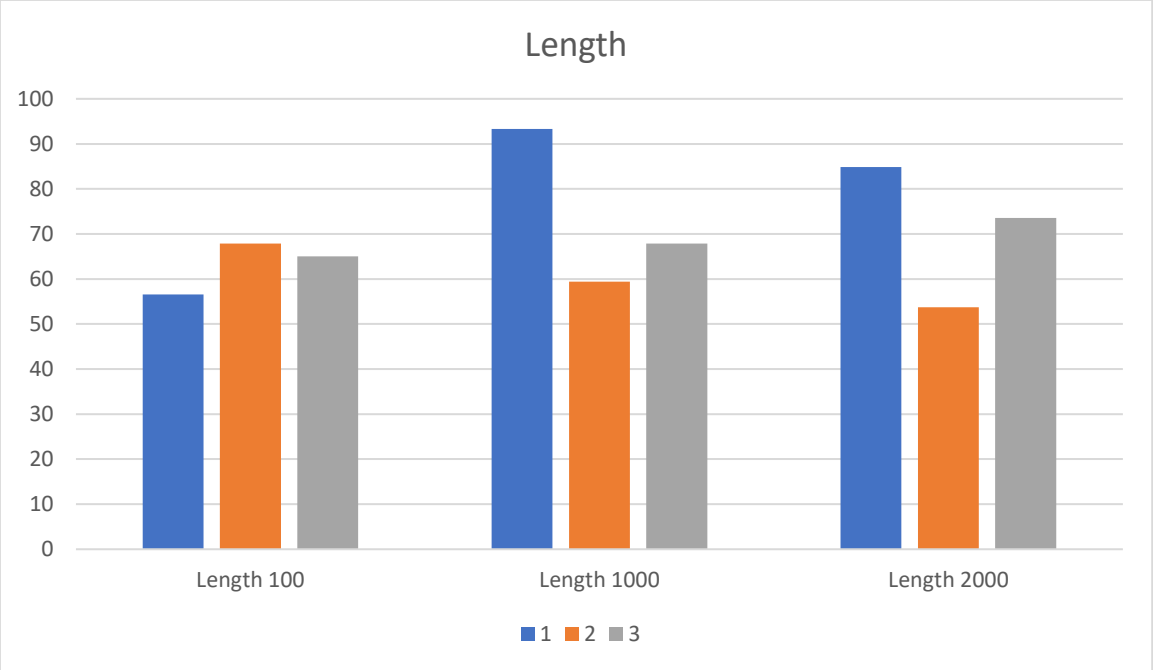
Length 1000:

```
best length:  93.33809511662426
the process was done in  52.06139259994961  seconds.
was achieved on temperature  9  at process number  308

best length:  59.39696961966998
the process was done in  57.878991000005044  seconds.
was achieved on temperature  6  at process number  321

best length:  67.88225099390854
the process was done in  57.18244889995549  seconds.
was achieved on temperature  6  at process number  92
```

Length 2000:

```
best length:  84.85281374238568
the process was done in  108.39314950001426  seconds.
was achieved on temperature  9  at process number  1533
```

```
best length:  53.74011537017759
the process was done in  114.50532709993422  seconds.
was achieved on temperature  6  at process number  35

best length:  73.53910524340093
the process was done in  107.56945149996318  seconds.
was achieved on temperature  9  at process number  692
```



3) Candidates

Candidates represent possible variants of current route. Increasing amount of created candidates creates more possibilities to create better route, but increases amount of operations and has bigger chance to stop cycle as none possible variants were found.

10 candidates:

```
best length:  152.73506473629422
the process was done in  0.1792905000038445  seconds.
was achieved on temperature  30  at process number  133

best length:  118.79393923933998
the process was done in  0.6078224999364465  seconds.
was achieved on temperature  30  at process number  428
```

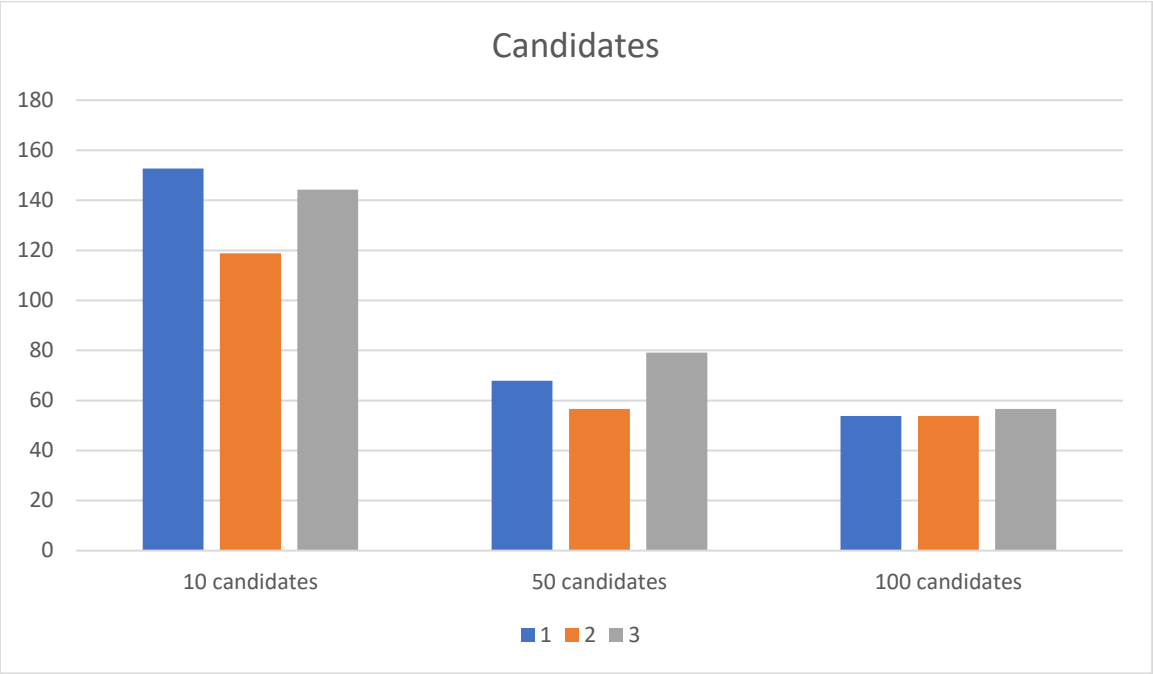
```
best length:  144.2497833620557
the process was done in  0.30749969999305904  seconds.
was achieved on temperature  30  at process number  228
```

50 candidates:

```
best length:  67.88225099390854
the process was done in  52.176176900044084  seconds.
was achieved on temperature  9  at process number  342

best length:  56.56854249492379
the process was done in  53.561696799937636  seconds.
was achieved on temperature  9  at process number  488

best length:  79.19595949289332
the process was done in  52.71162459999323  seconds.
was achieved on temperature  9  at process number  514
```

100 candidates:

```
best length:  53.74011537017759
the process was done in  129.2849604000803  seconds.
was achieved on temperature  3  at process number  1

best length:  53.740115370177605
the process was done in  134.60196790006012  seconds.
was achieved on temperature  3  at process number  5

best length:  56.56854249492379
the process was done in  130.338645300013  seconds.
was achieved on temperature  3  at process number  20
```


Candidates
180
160
140
120
100
80
60
40
20
0
10 candidates
50 candidates
100 candidates
1
2
3