Universidad de Córdoba

Escuela Politécnica Superior de Córdoba

Metaheuristics

Assignment 3

Author:
Ventura Lucena Martínez

Teacher:
José María Luna Ariza

UNIVERSIDAD D CÓRDOBA

May 11, 2021

Contents

1 Introduction **3**

2 Code modulation **3**

3 Data structures **4**

4 Procedure **5**
4.1 Graph evaluation 5
4.2 Genetic Algorithm application 7
4.2.1 Mutation operator 1 7
4.2.2 Mutation operator 2 8

5 Evaluation **8**
5.1 Test 1 8
5.2 Test 2 11
5.3 Test 3 14
5.4 Test 4 17
5.5 Test 5 20
5.6 Test 6 23
5.7 Test 7 26
5.8 Test 8 28
5.9 Test 9 31
5.10 Test 10 34

6 Conclusions **37**

List of Figures

1 Easy visualization with data structures. 4
2 Graph evaluation example (1). 5
3 Graph evaluation example (2). 6
4 Graph evaluation example (3). 6
5 Graph evaluation example (4). 7
6 Test 1 - Solution 1. 9
7 Test 1 - Solution 2. 9
8 Test 1 - Solution 3. 10
9 Test 1 - Solution 4. 10
10 Test 1 - Solution 5. 11
11 Test 2 - Solution 1. 12
12 Test 2 - Solution 2. 12
13 Test 2 - Solution 3. 13
14 Test 2 - Solution 4. 13

15 Test 2 - Solution 5. 14
16 Test 3 - Solution 1. 15
17 Test 3 - Solution 2. 15
18 Test 3 - Solution 3. 16
19 Test 3 - Solution 4. 16
20 Test 3 - Solution 5. 17
21 Test 4 - Solution 1. 18
22 Test 4 - Solution 2. 18
23 Test 4 - Solution 3. 19
24 Test 4 - Solution 4. 19
25 Test 4 - Solution 5. 20
26 Test 5 - Solution 1. 21
27 Test 5 - Solution 2. 21
28 Test 5 - Solution 3. 22
29 Test 5 - Solution 4. 22
30 Test 5 - Solution 5. 23
31 Test 6 - Solution 1. 24
32 Test 6 - Solution 2. 24
33 Test 6 - Solution 3. 25
34 Test 6 - Solution 4. 25
35 Test 6 - Solution 5. 26
36 Test 7 - Solution 1. 27
37 Test 7 - Solution 2. 27
38 Test 7 - Solution 3. 28
39 Test 8 - Solution 1. 29
40 Test 8 - Solution 2. 29
41 Test 8 - Solution 3. 30
42 Test 8 - Solution 4. 30
43 Test 8 - Solution 5. 31
44 Test 9 - Solution 1. 32
45 Test 9 - Solution 2. 32
46 Test 9 - Solution 3. 33
47 Test 9 - Solution 4. 33
48 Test 9 - Solution 5. 34
49 Test 10 - Solution 1. 35
50 Test 10 - Solution 2. 35
51 Test 10 - Solution 3. 36
52 Test 10 - Solution 4. 36
53 Test 10 - Solution 5. 37

List of Tables

1 Algorithm parameters. 8
2 Results test 1. 8
3 Results test 2. 11
4 Results test 3. 14
5 Results test 4. 17
6 Results test 5. 20
7 Results test 6. 23
8 Results test 7. 26
9 Results test 8. 28
10 Results test 9. 31
11 Results test 10. 34

1 Introduction

During the course, we have applied different metaheuristics to solve problems, clasical problems such as Traveling Salesman Problem or Knapsack Problem. Hill Climbing, Simulated Annealing, or specific operators frequently used in Genetic Algorithms, such as Crossover-operator or Mutation-operator have provided good solutions to the previous problems. Now, the tools discussed above will be used to solve a real-world problem.

The algorithm needs to deal with a database where each record (rows in the database) has the following form:

$$B0 : A1 : B2 : C3 : D3 : C4 : C7 : E8 : E10 \tag{1}$$

This indicates that there is an event B that occurs at time 0; an event A, which occurs at time 1, etc. Two events can occur at the same instant of time. For example, event C occurs for the first time at time 3 and event D also occurs at that instant of time.

The objective is to extract graphs with temporal restrictions so that the number of records (rows of the database) that this graph fulfills is the maximum possible. Some important considerations are:

- Graphs should be as big as possible. A graph considering just two events, e.g. A and B, would be satisfied by a large number of data records, but it would not make sense. Thus, it is established that a graph must have at least 4 events.
- The graph constraints are important to satisfy the data. Infinite values in a constraint would result in all records being satisfied. Thus, one graph is better than another if the restrictions obtained are as small as possible (ranges).
- If two graphs satisfy the same records, then the graph with the lowest mean restrictions will be better. That is, whose ranges are lower.

2 Code modulation

In order to have well identified the functionalities of the code, 4 different source files have been created:

- **main.py:** contains the main program.
- **data_management.py:** contains functionalities related to data manipulation in certain data structures, such a text files, python dictionaries or data frames.

- **graph.py:** contains functionalities related to graphs, such as it generation, evaluation and plotting.
- **genetic_algorithm.py:** contains functionalities related to the metaheuristic used in the problem. In this case, a genetic algorithm with a double mutation operator. It also contains some codification related to elitism, in order to store the best solutions.

3 Data structures

A very remarkable aspect when dealing with the problem has been the selection of a good data structures, both to store the information and to have easy access and easy understanding when it comes to coding. In this sense, the structures that have facilitated the study have been both **dictionaries** and **pandas data frames** in Python[1]. So that, an example of printed information on the console would be as follows:

```
                        Event                          Time
0                   [A, B, C]                    [0, 10, 21]
1  [B, A, B, C, D, C, C, E, E]  [0, 1, 2, 3, 3, 4, 7, 8, 10]
2        [E, A, C, A, B, C, E]      [0, 4, 8, 9, 10, 10, 12]


1/3 completed.
2/3 completed.
3/3 completed.

       Elite  Profit                                                Graph
0  [0, 1, 2]     1.0        {'AB': [['0', '10']], 'BC': [['10', '21']]}
1  [0, 1, 2]     1.0  {'AB': [['0', '10']], 'BA': [['10', '21']], 'A...

0.14387249946594238 seconds
```

Figure 1: Easy visualization with data structures.

The reason for using Python dictionaries is to have non superfluous nor duplicated data in memory taking the advantage of its functionality with the storage of data in "key:value" pairs.

[1] **Lists** have been also important, due to the fact that we used them in a complementary way with the previous ones commented.

Last data structure, but not least is the **network graph**. We can plot the solution graph by using the library "networkx" that allow us to create multiple sorts of graphs.

4 Procedure

We must split the procedure into two very specific and important parts: the graph evaluation and the application of the genetic algorithm over the graph.

4.1 Graph evaluation

The functionality has to test if every record stored in the data base matches with the constraint or the graph. For instance, lets suppose the following:

```
                      Event                       Time
0                   [A, B, C]                 [0, 10, 21]
1  [B, A, B, C, D, C, C, E, E]  [0, 1, 2, 3, 3, 4, 7, 8, 10]
2        [E, A, C, A, B, C, E]     [0, 4, 8, 9, 10, 10, 12]

Constraints: defaultdict(<class 'list'>, {'AB': [['0', '10']], 'BC': [['10', '21']]})

1/3 completed.
2/3 completed.
3/3 completed.

       Elite  Profit                                              Graph
0  [0, 1, 2]     1.0       {'AB': [['0', '10']], 'BC': [['10', '21']]}
1  [0, 1, 2]     1.0  {'AB': [['0', '10']], 'BC': [['10', '21']], 'C...
2  [0, 1, 2]     1.0  {'AB': [['0', '10']], 'BC': [['10', '21']], 'C...
```

Figure 2: Graph evaluation example (1).

In this execution example we can observe 3 records of a data base, each of them separated in "Event", that represents all the events in each record; and "Time", which represents all the times in each record, all of them ordered. Also, we can see the graph that represents the constraints to satisfy for each record in the data base. Having stated the above, for the constraints to be satisfied, each record will be evaluated, both event and time, and must be kept within the range indicated in the graph:

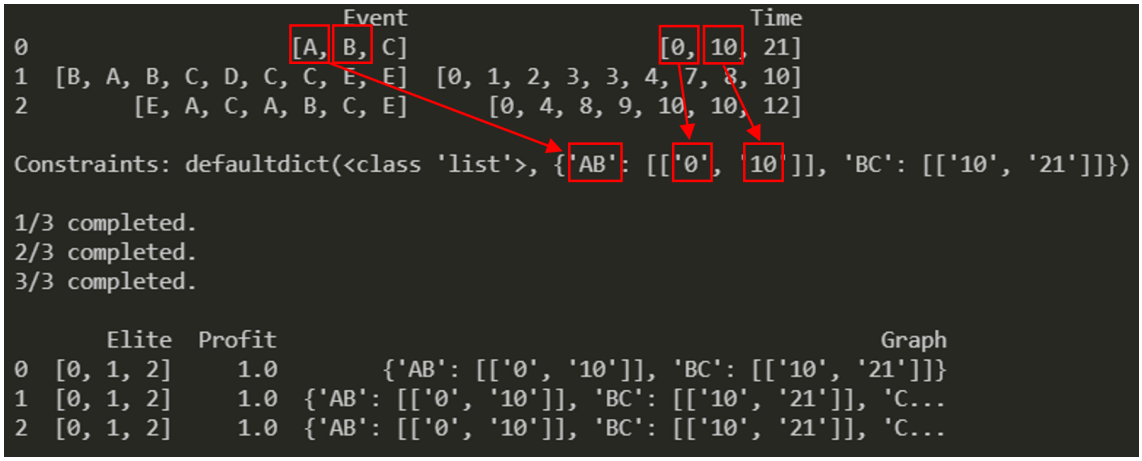

Figure 3: Graph evaluation example (2).

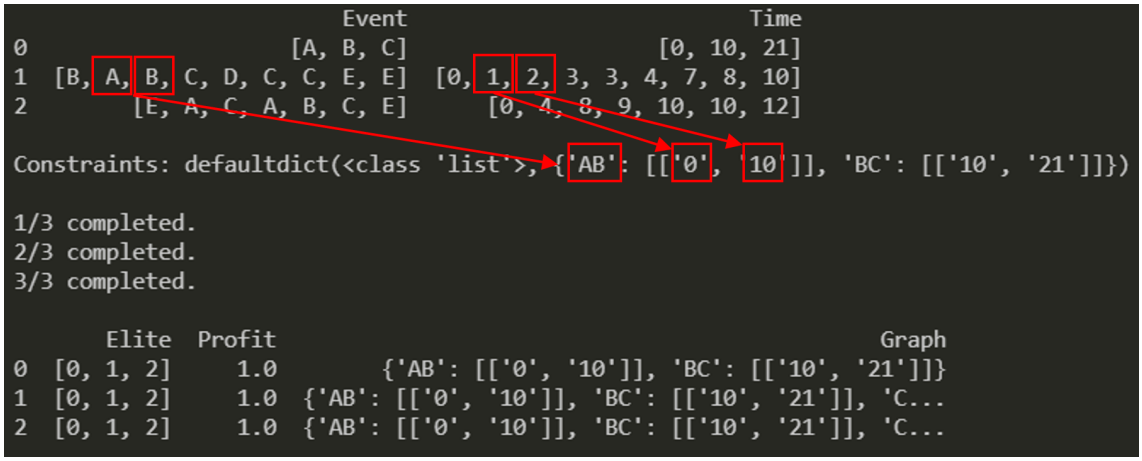

Figure 4: Graph evaluation example (3).

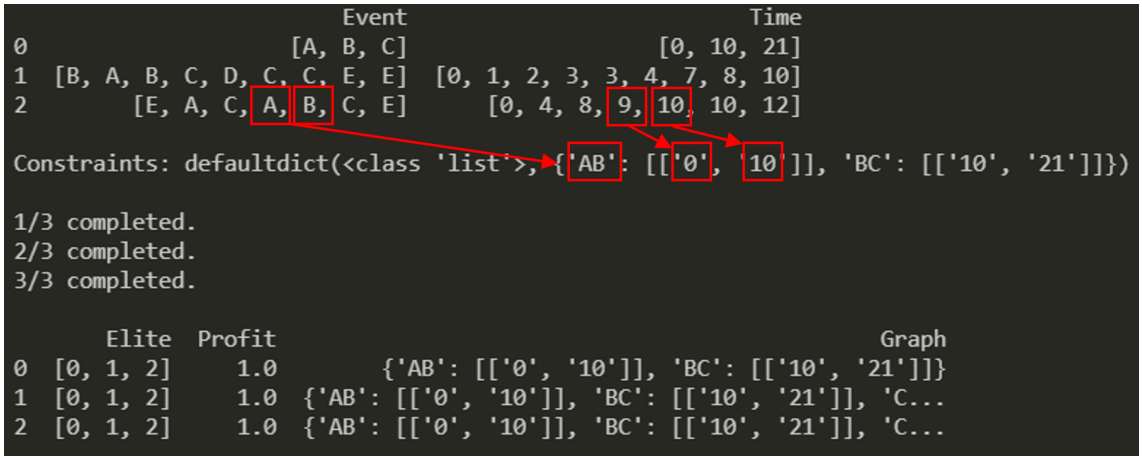

Figure 5: Graph evaluation example (4).

4.2 Genetic Algorithm application

In this case, the method applies two kinds of mutation over the genes of a selected group of a population.

4.2.1 Mutation operator 1

On the first hand, the first mutation operator applies a mutation over a parent gene. This is, a random event on the selected parent is changed for another one existing on the data base. For instance, lets suppose the following:

$$Events = [A, B, C, D, E] \tag{2}$$

These are all the different events in the data base and imagine that **event B** is the one selected to be mutated with another **event E**. The operator would proceed as follows:

$$Original = AB : [[0, 10]], BC : [[10, 21], [21, 22]] \tag{3}$$

$$Mutation = AE : [[0, 10]], EC : [[10, 21], [21, 22]] \tag{4}$$

4.2.2 Mutation operator 2

On the second hand, the second mutation operator applies an extra gene on a selected parent. This is, a random event is generated and appended to the last existing event. Using the previous example, the operator would proceed as follows:

$$Original = AB : [[0, 10]], BC : [[10, 21], [21, 22]] \tag{5}$$

$$Mutation = AB : [[0, 10]], BC : [[10, 21], [21, 22]], CE : [[22, 24]] \tag{6}$$

5 Evaluation

The evaluation of the algorithm has consisted of executing the algorithm 10 times, changing certain parameters:

Parameter	Definition
nSolutions	Population size.
maxGenerations	Maximum iterations.
mProb	Mutation operator probability.

Table 1: Algorithm parameters.

In order to see how the algorithm behaves, these parameters have been changed over the iterations, sometimes, in a way that may not be correct if we are implementing an evolutionary algorithm. For instance, mutation probability should not be more than 0.2, due to the fact that randomness would impact highly on the results, but we have exceed this constraint in some executions, sometime reaching the probability of 1. Also, we have to consider **valid solutions**, which must have, at least, **4 nodes**.

5.1 Test 1

nSolutions	30
maxGenerations	3
mProb	0,2
time (s)	53,966145277
Max. reached	Yes
Valid max.	Yes

Table 2: Results test 1.

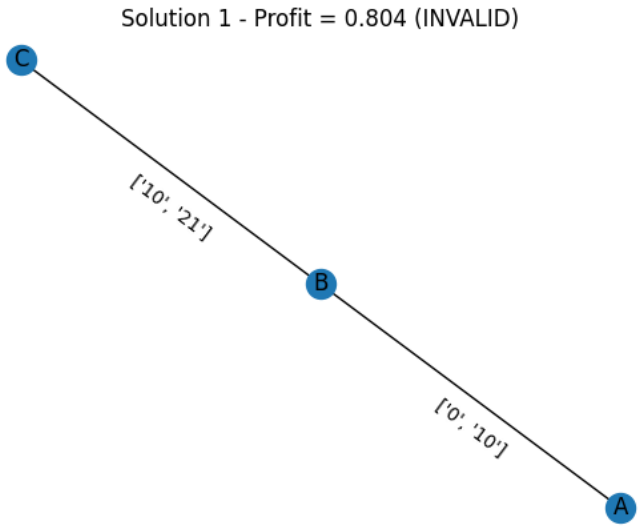

Figure 6: Test 1 - Solution 1.

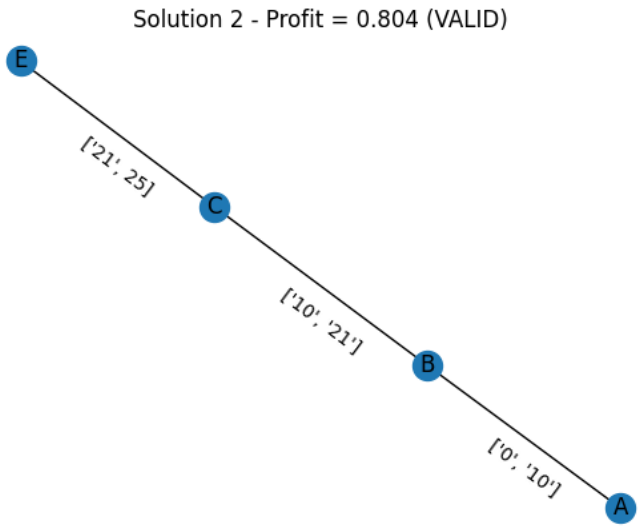

Figure 7: Test 1 - Solution 2.

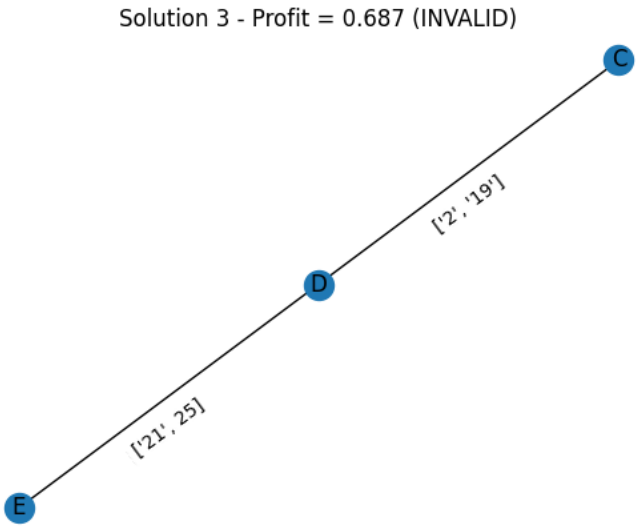

Figure 8: Test 1 - Solution 3.

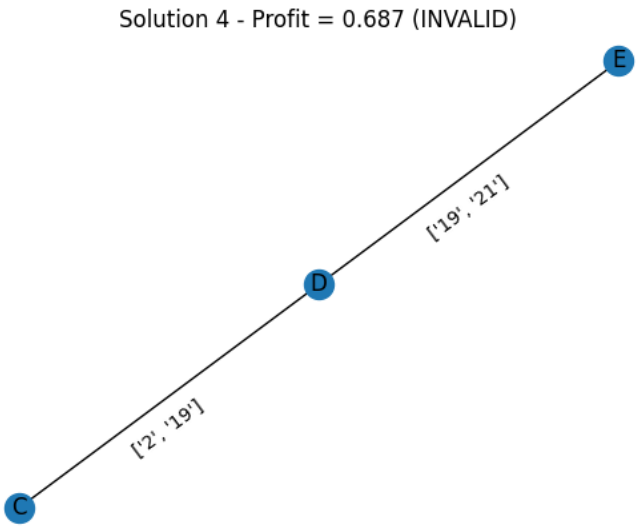

Figure 9: Test 1 - Solution 4.

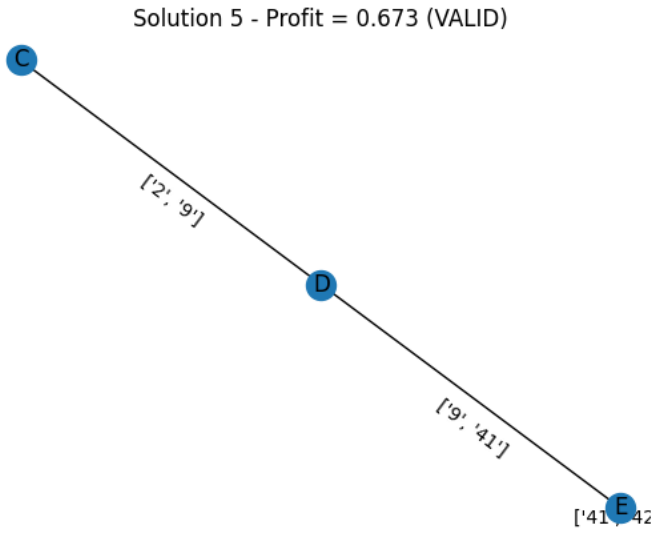

Figure 10: Test 1 - Solution 5.

5.2 Test 2

nSolutions	20
maxGenerations	3
mProb	0,2
time (s)	41.11404061317444
Max. reached	Yes
Valid max.	Yes

Table 3: Results test 2.

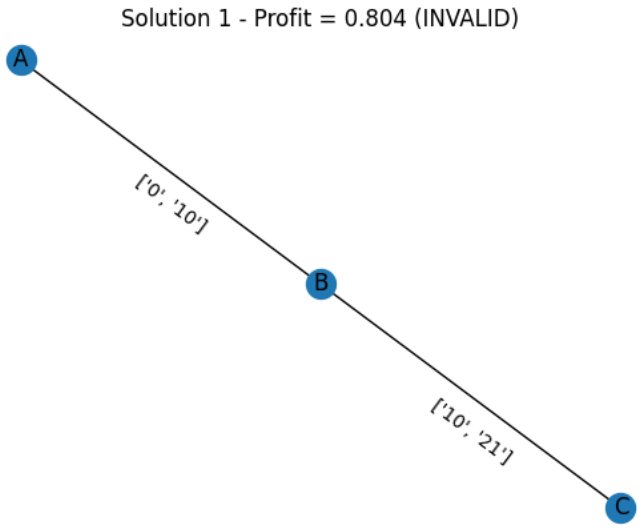

Figure 11: Test 2 - Solution 1.

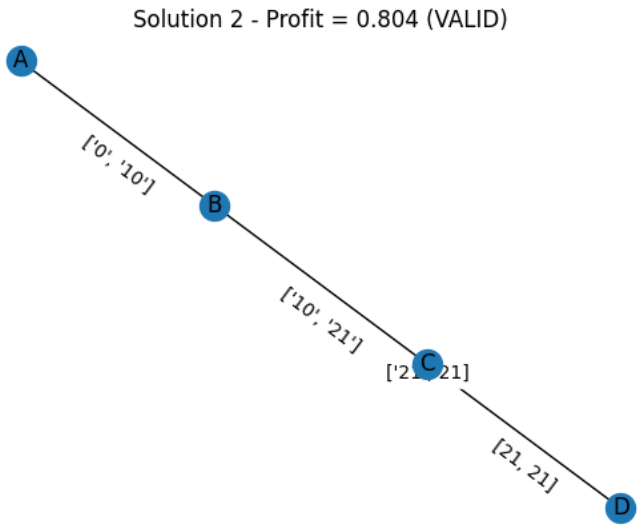

Figure 12: Test 2 - Solution 2.

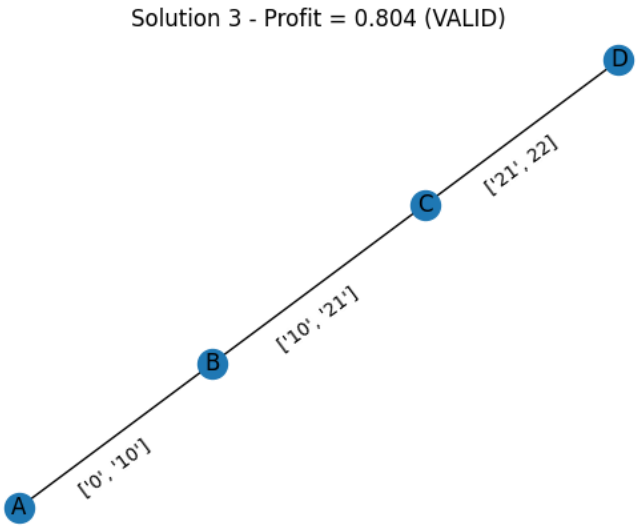

Figure 13: Test 2 - Solution 3.

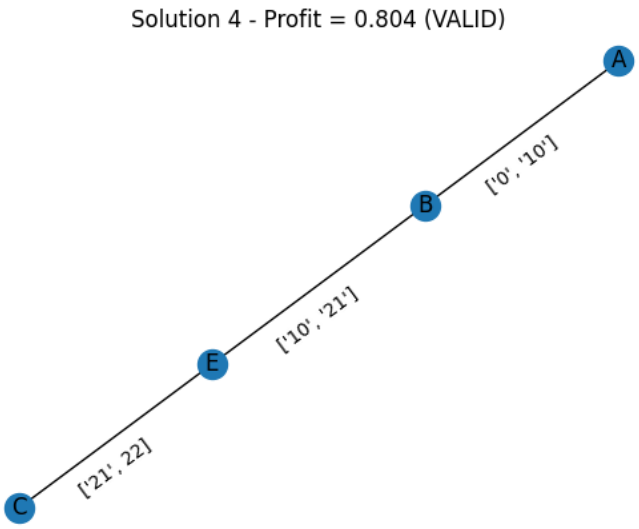

Figure 14: Test 2 - Solution 4.

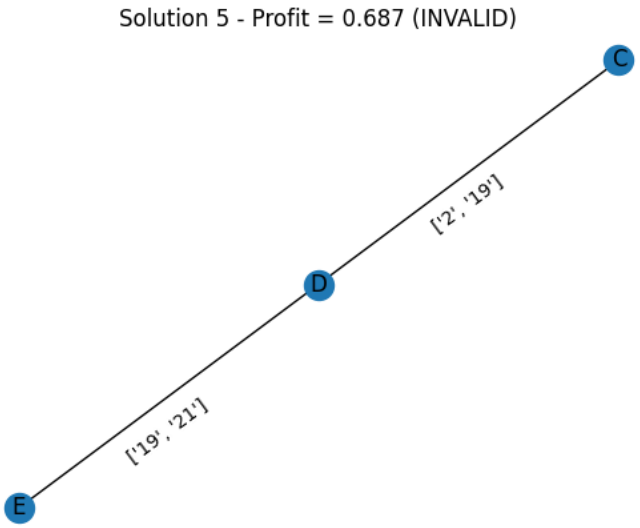

Figure 15: Test 2 - Solution 5.

5.3 Test 3

nSolutions	30
maxGenerations	10
mProb	0,2
time (s)	159.5819399356842
Max. reached	Yes
Valid max.	Yes

Table 4: Results test 3.

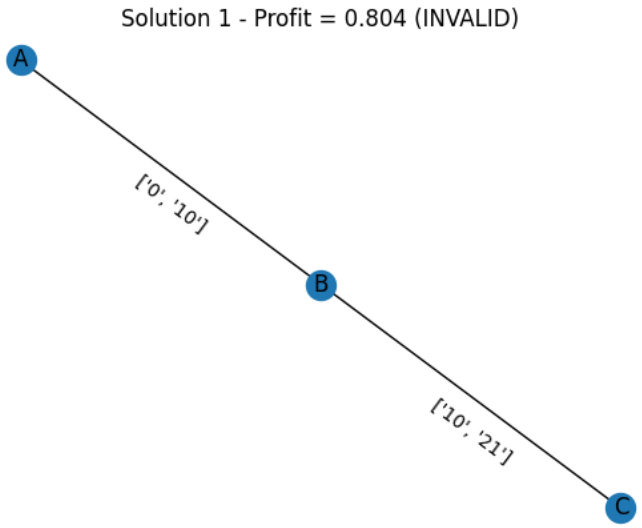

Figure 16: Test 3 - Solution 1.

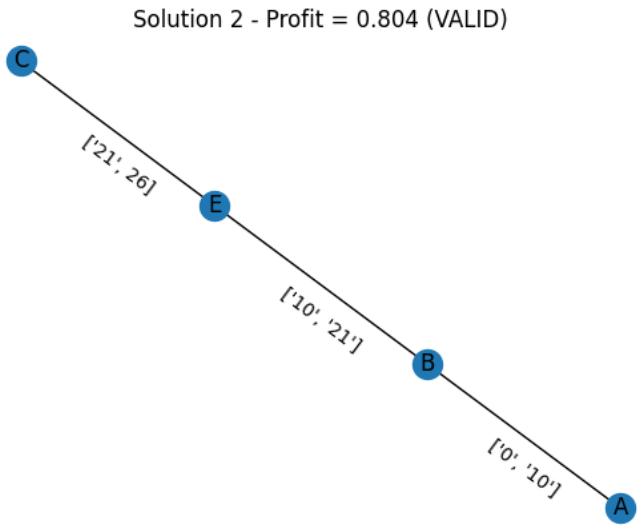

Figure 17: Test 3 - Solution 2.

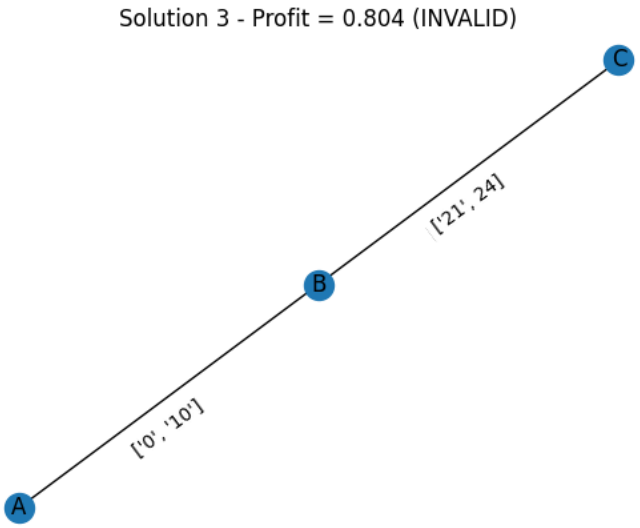

Figure 18: Test 3 - Solution 3.

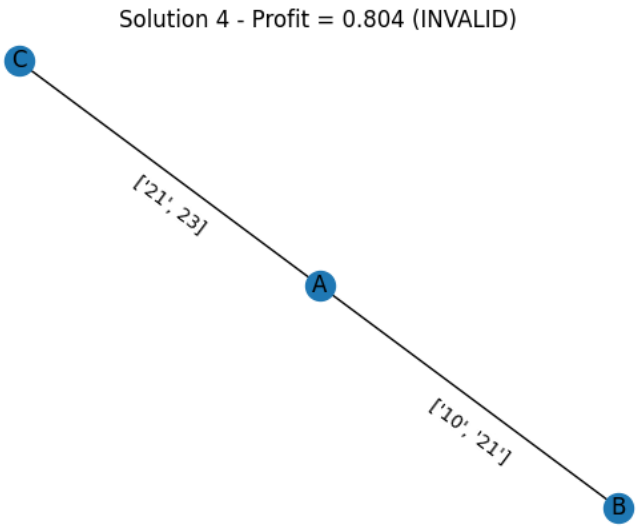

Figure 19: Test 3 - Solution 4.

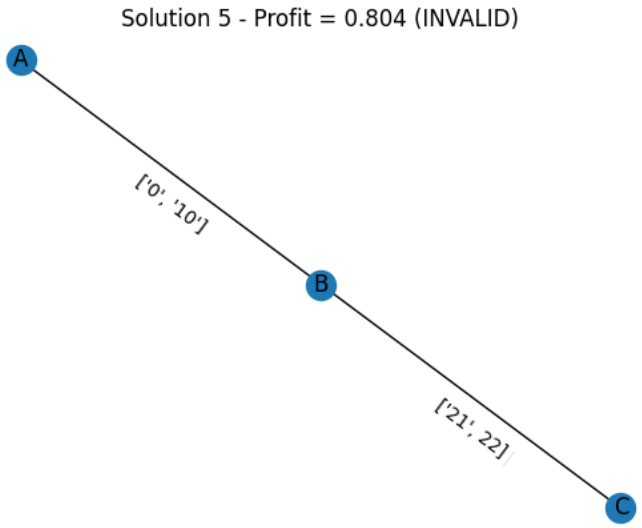

Figure 20: Test 3 - Solution 5.

5.4 Test 4

nSolutions	50
maxGenerations	5
mProb	0,1
time (s)	143.60365080833435
Max. reached	Yes
Valid max.	Yes

Table 5: Results test 4.

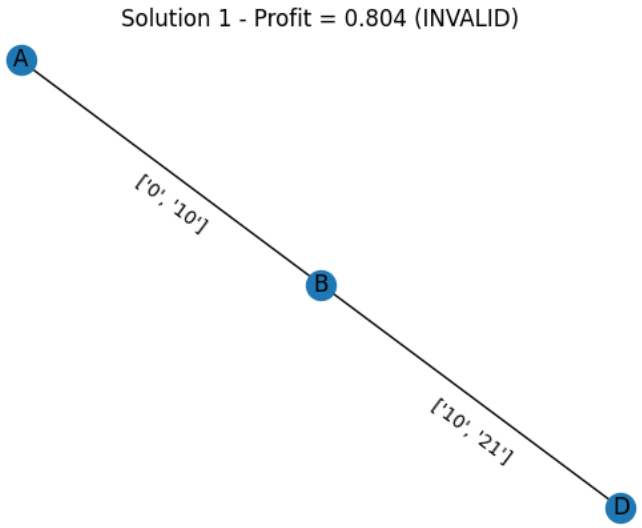

Figure 21: Test 4 - Solution 1.

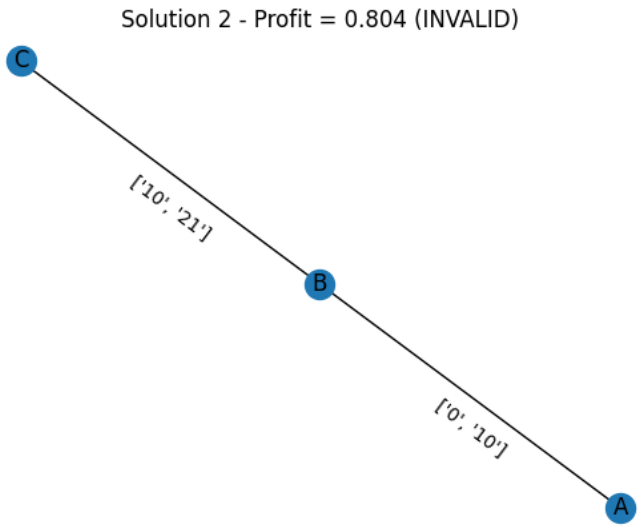

Figure 22: Test 4 - Solution 2.

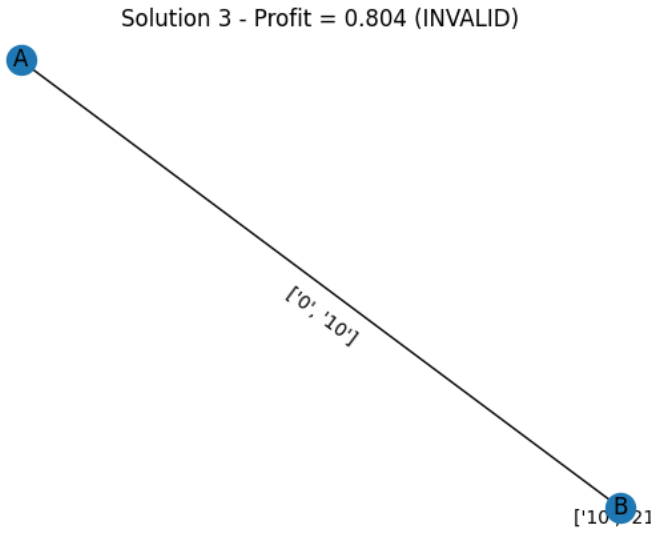

Figure 23: Test 4 - Solution 3.

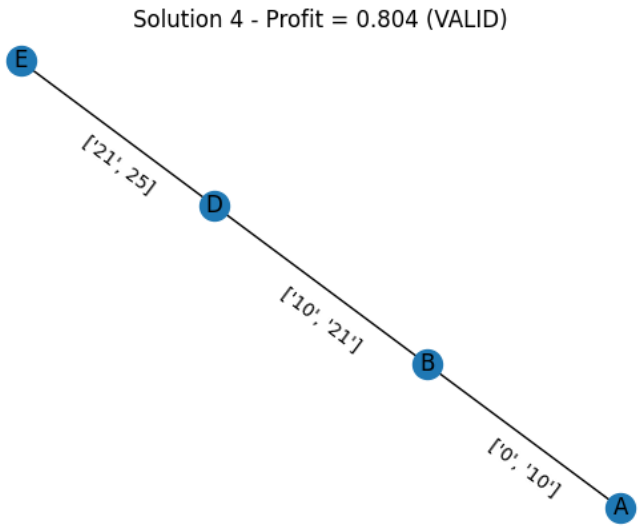

Figure 24: Test 4 - Solution 4.

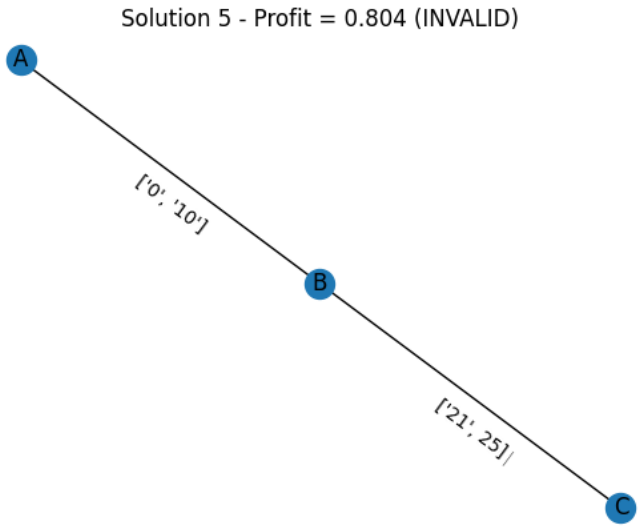

Figure 25: Test 4 - Solution 5.

5.5 Test 5

nSolutions	50
maxGenerations	10
mProb	0,2
time (s)	303.5257692337036
Max. reached	Yes
Valid max.	Yes

Table 6: Results test 5.

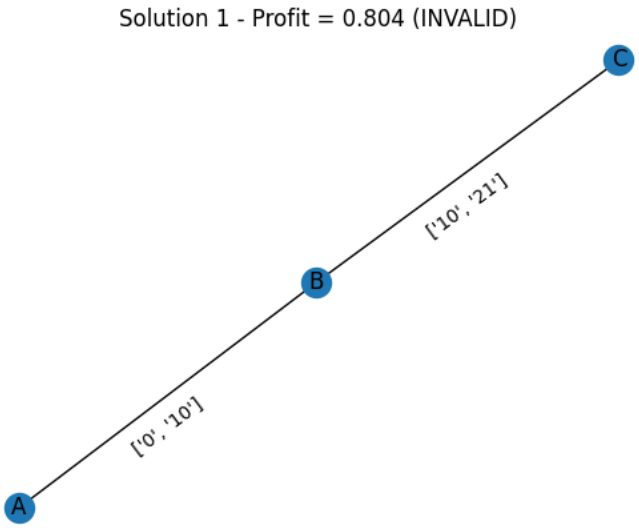

Figure 26: Test 5 - Solution 1.

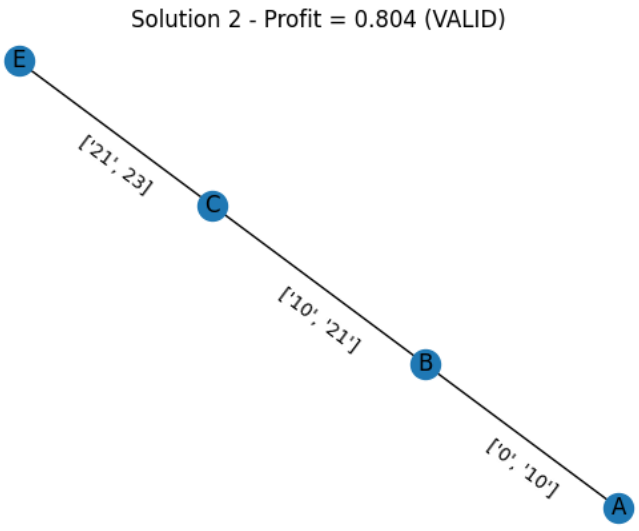

Figure 27: Test 5 - Solution 2.

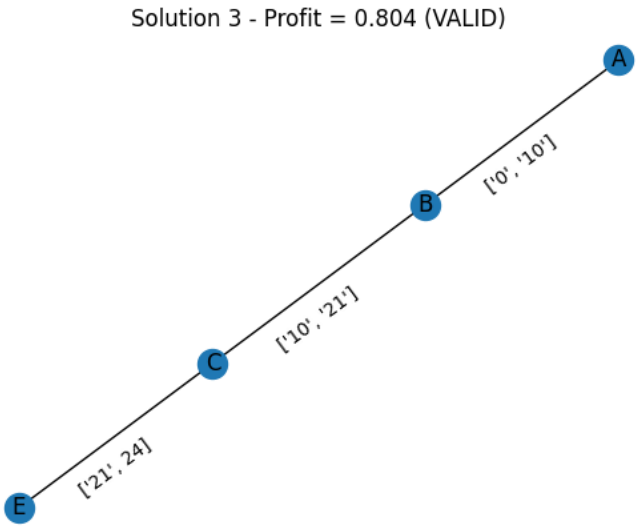

Figure 28: Test 5 - Solution 3.

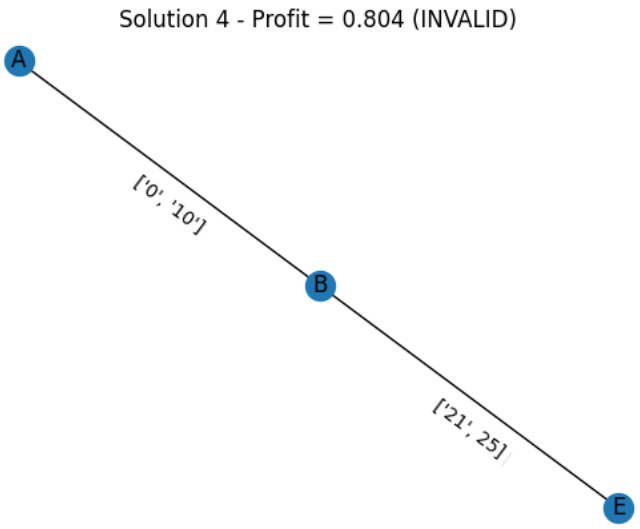

Figure 29: Test 5 - Solution 4.

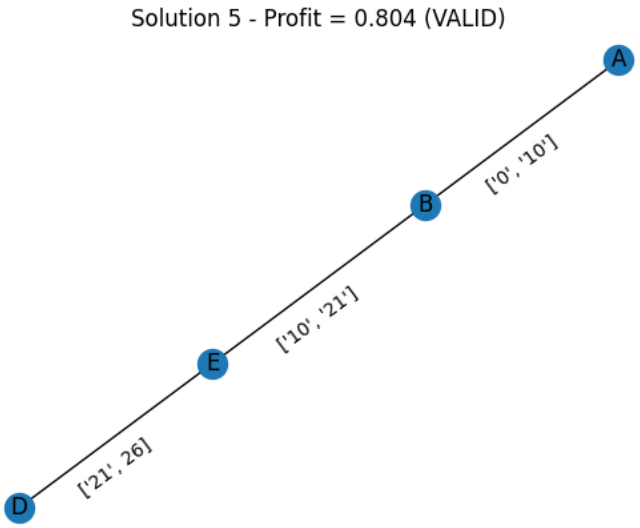

Figure 30: Test 5 - Solution 5.

5.6 Test 6

nSolutions	5
maxGenerations	10
mProb	0,1
time (s)	35.61323642730713
Max. reached	No
Valid max.	-

Table 7: Results test 6.

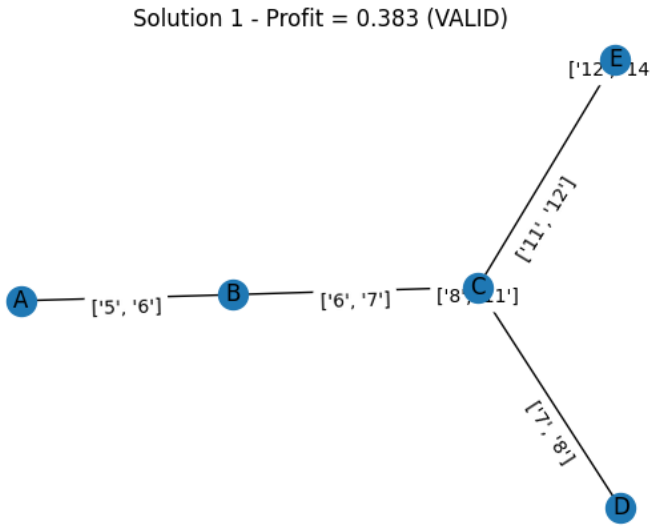

Figure 31: Test 6 - Solution 1.

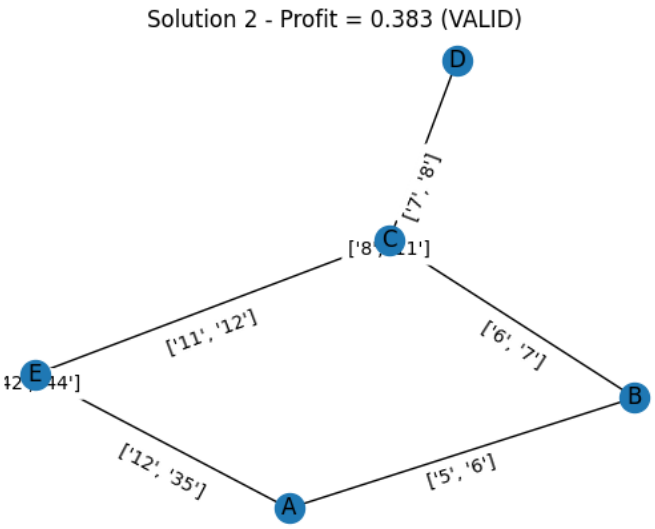

Figure 32: Test 6 - Solution 2.

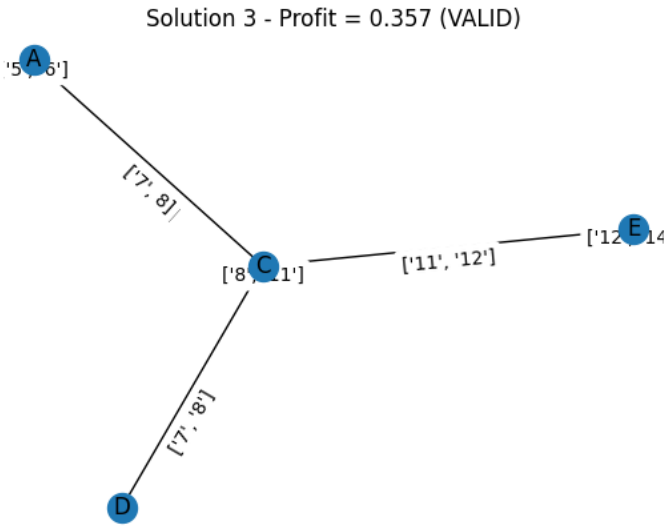

Figure 33: Test 6 - Solution 3.

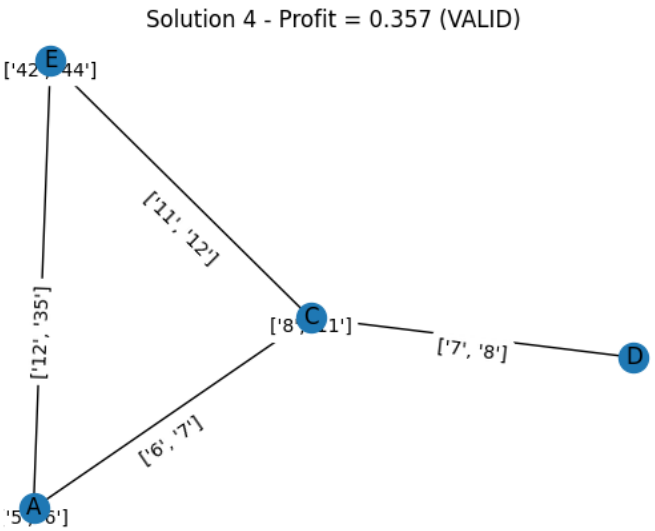

Figure 34: Test 6 - Solution 4.

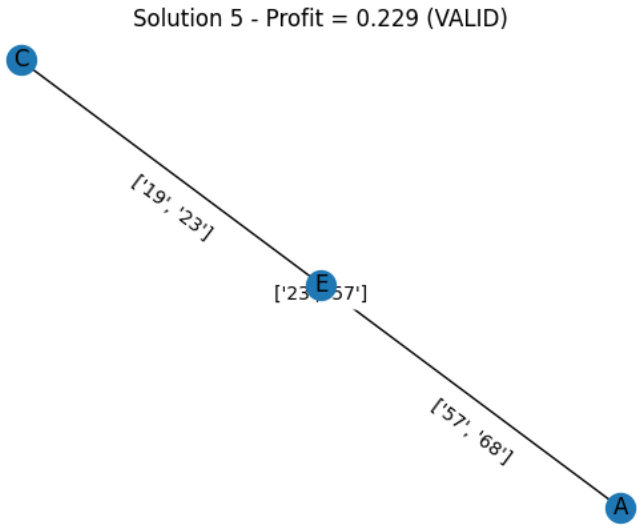

Figure 35: Test 6 - Solution 5.

5.7 Test 7

nSolutions	100
maxGenerations	2
mProb	0,1
time (s)	155.47787427902222
Max. reached	Yes
Valid max.	Yes

Table 8: Results test 7.

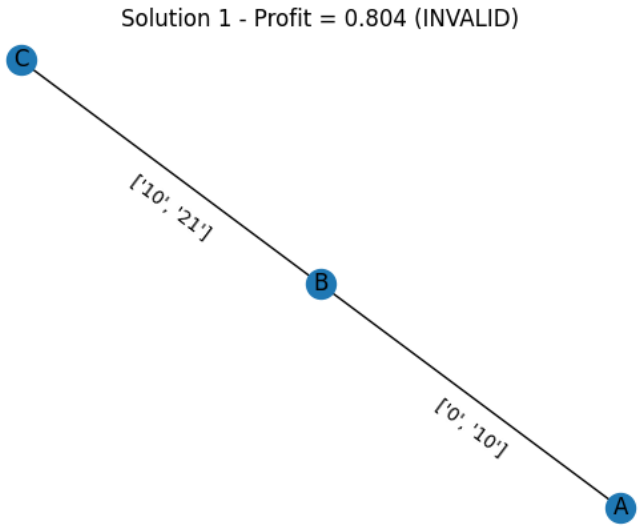

Figure 36: Test 7 - Solution 1.

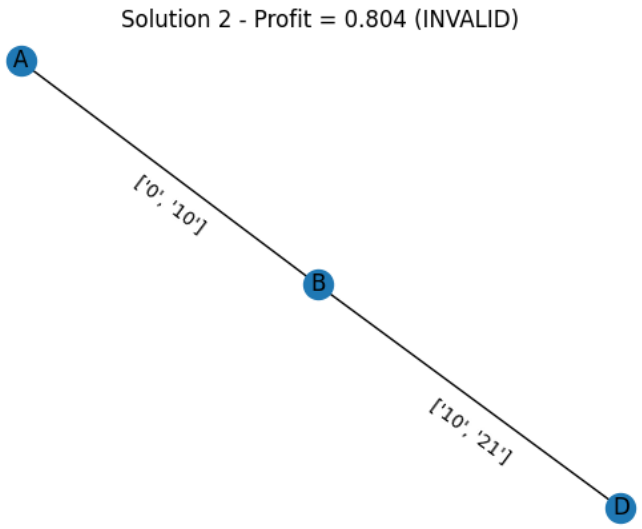

Figure 37: Test 7 - Solution 2.

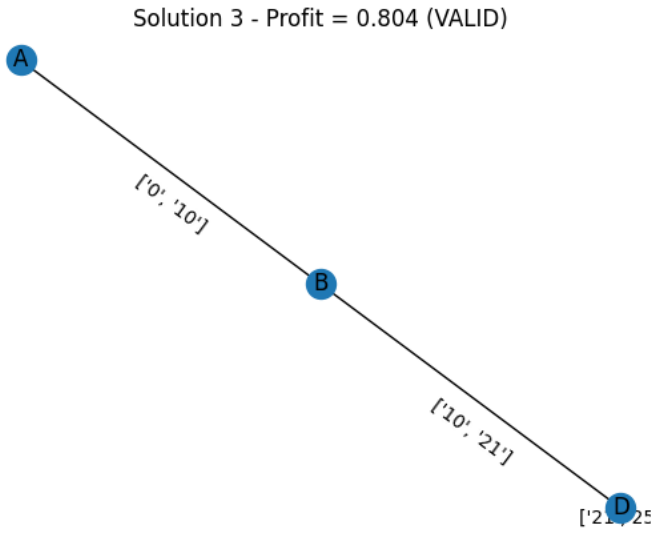

Figure 38: Test 7 - Solution 3.

5.8 Test 8

nSolutions	30
maxGenerations	5
mProb	0,5
time (s)	80.96481418609619
Max. reached	Yes
Valid max.	Yes

Table 9: Results test 8.

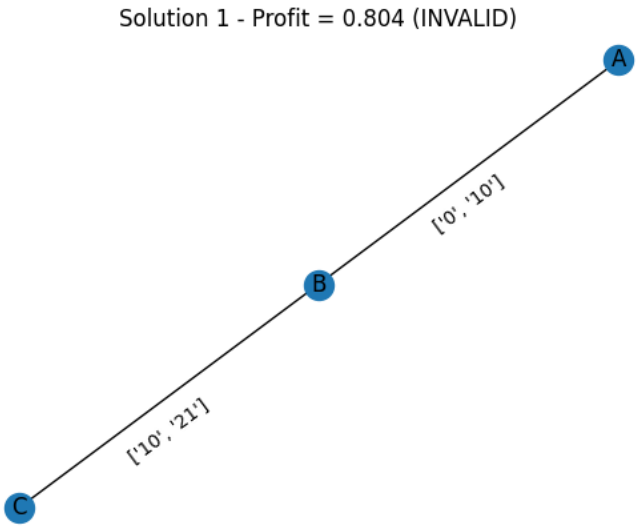

Figure 39: Test 8 - Solution 1.

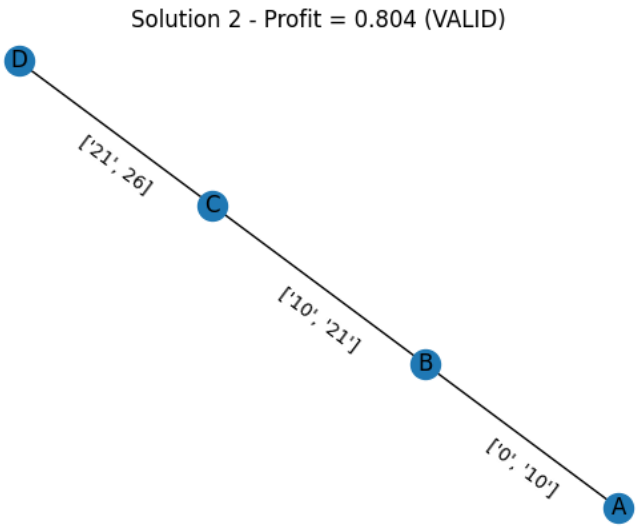

Figure 40: Test 8 - Solution 2.

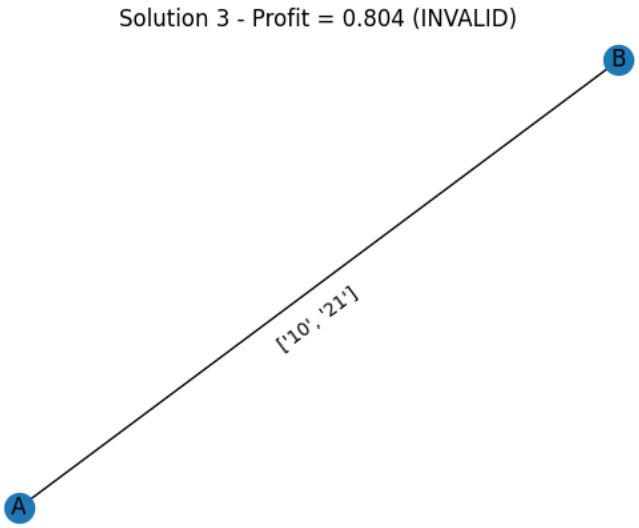

Figure 41: Test 8 - Solution 3.

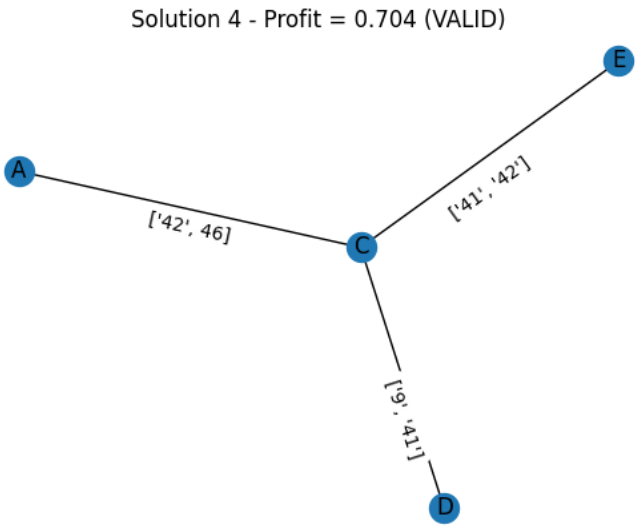

Figure 42: Test 8 - Solution 4.

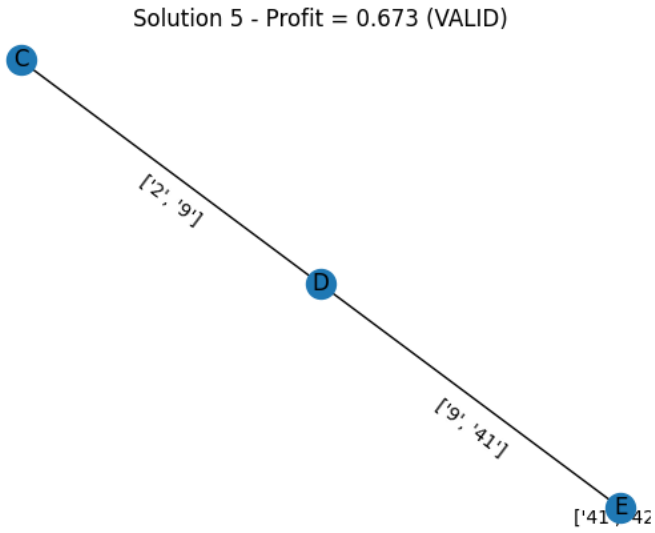

Figure 43: Test 8 - Solution 5.

5.9 Test 9

nSolutions	10
maxGenerations	5
mProb	1
time (s)	20.765729904174805
Max. reached	Yes
Valid max.	Yes

Table 10: Results test 9.

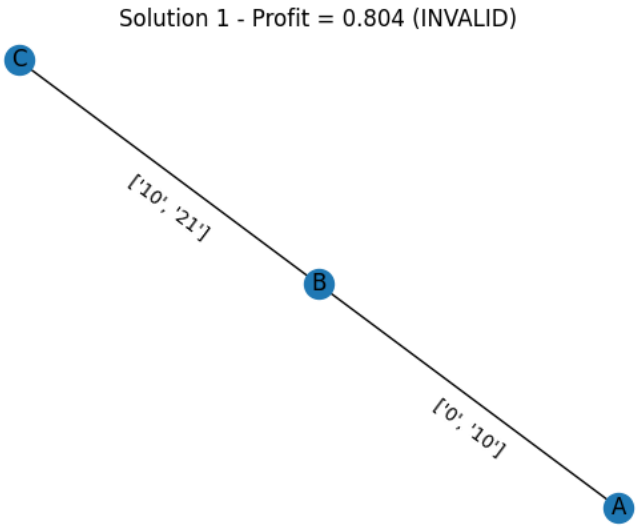

Figure 44: Test 9 - Solution 1.

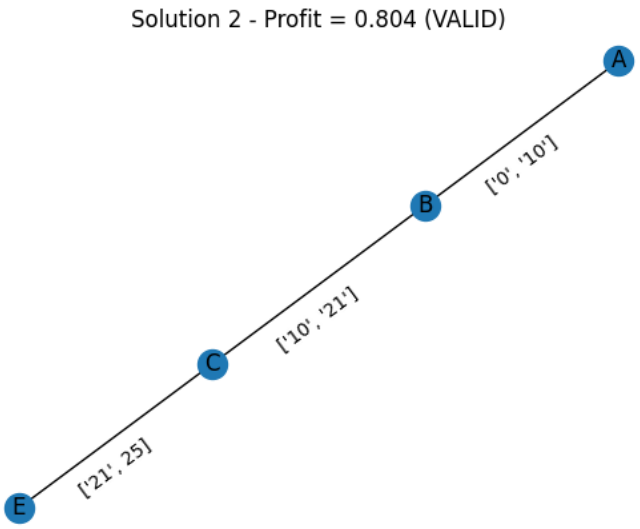

Figure 45: Test 9 - Solution 2.

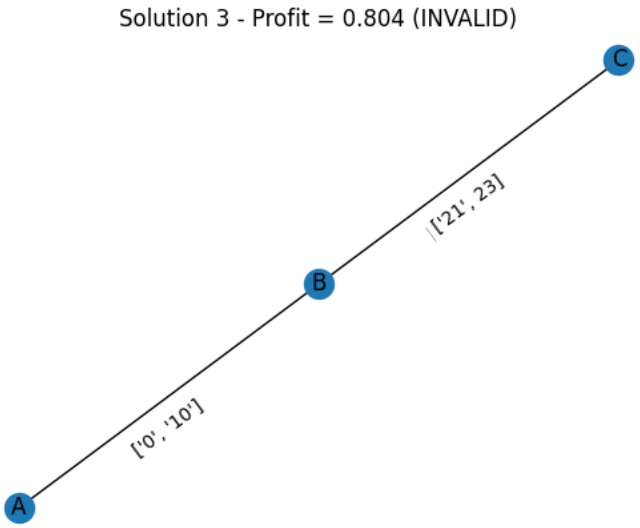

Figure 46: Test 9 - Solution 3.

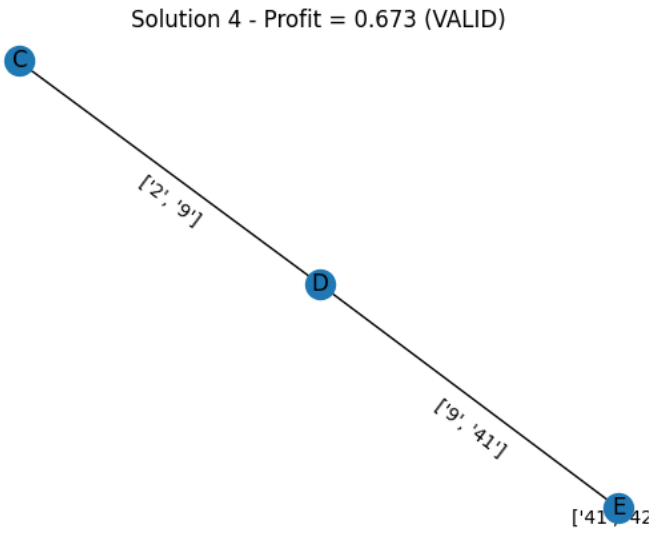

Figure 47: Test 9 - Solution 4.

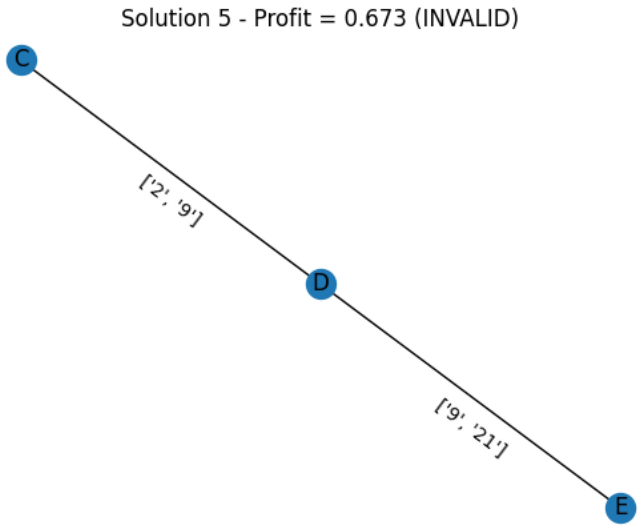

Figure 48: Test 9 - Solution 5.

5.10 Test 10

nSolutions	10
maxGenerations	5
mProb	0
time (s)	26.90307378768921
Max. reached	Yes
Valid max.	No

Table 11: Results test 10.

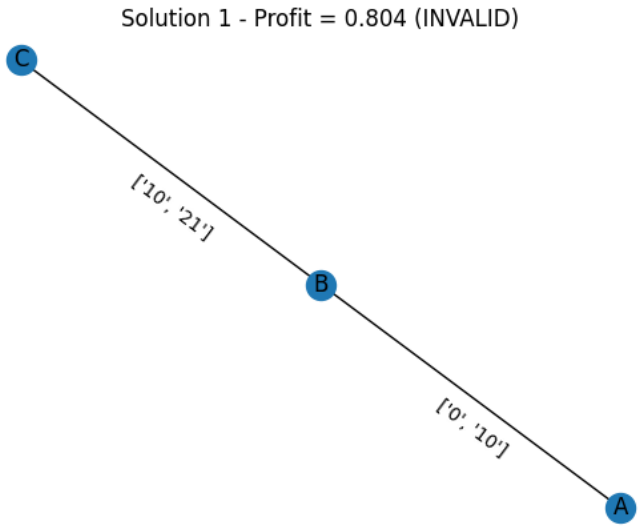

Figure 49: Test 10 - Solution 1.

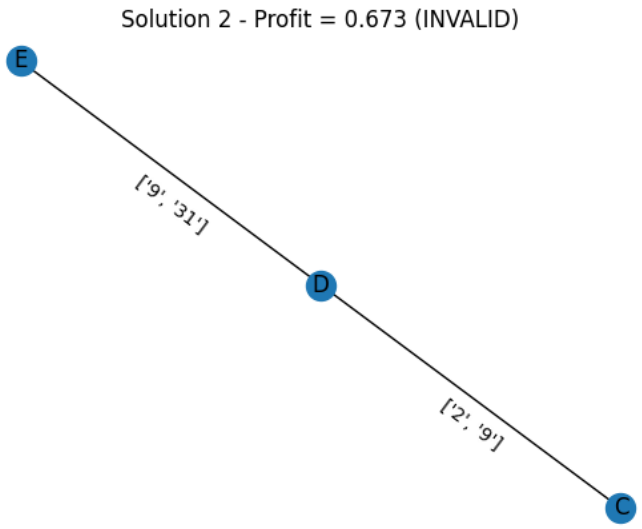

Figure 50: Test 10 - Solution 2.

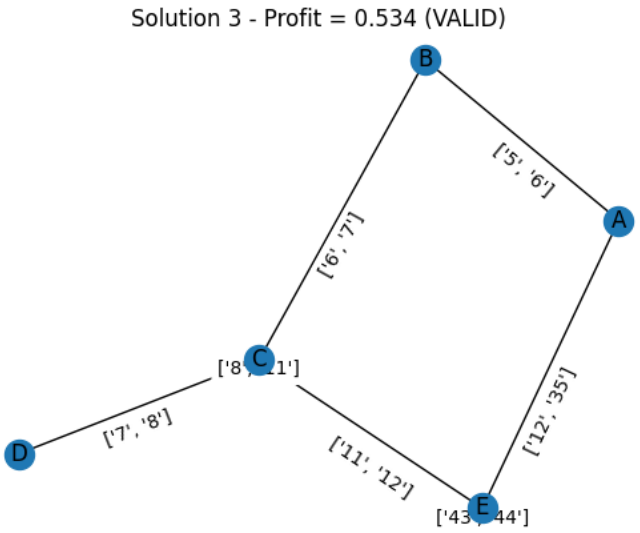

Figure 51: Test 10 - Solution 3.

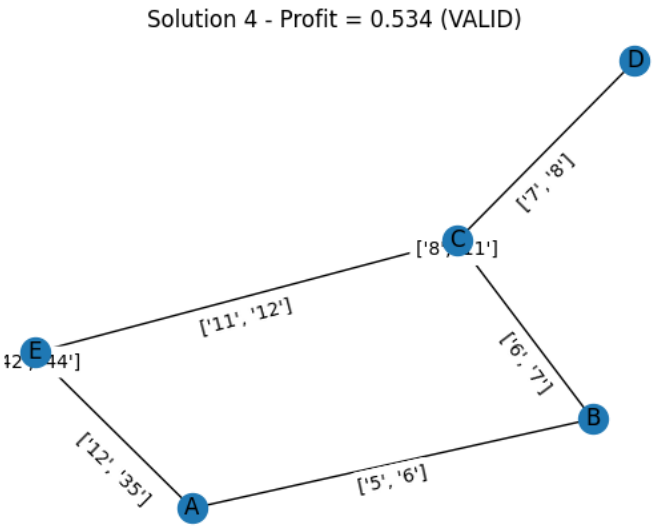

Figure 52: Test 10 - Solution 4.

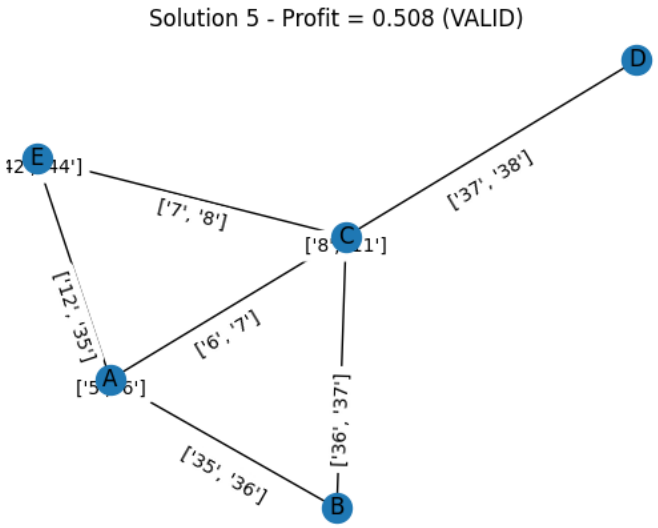

Figure 53: Test 10 - Solution 5.

6 Conclusions

Overall, in the tests carried out, the maximum profit (maximum number of records satisfied, which has been 804) has been reached with probability 0.9. Within this 0.9, the probability of reaching a valid maximum profit solution has been 0.8888.

It is also remarkable that the algorithm behaves better in term of computing time increasing the population size instead of increasing the number of iterations. This could be explained due to the fact that the graphs (or constraints) generated are not build randomly, but taking some records as it and then, they could be mutated.

In addition, even though the result are quite good (considering that the algorithm has no bugs and computes the results with no mistakes) we could add some extra functionalities that could help to reach better solutions, if they exist:

- **Crossover operator:** to combine genetic information of selected parents.
- **Heuristics:** for more refined solutions. Even though the solutions satisfy the constraints, we could refine them reducing the ranges in the graphs.
- **Metaheuristics combination:** including, for instance, Simulated Annealing with mutation probability.

No one of these functionalities were added due to the fact that good results were obtained.

References

[1] Assignment 3 statement - Moodle Universidad de Córdoba.

[2] Assignment 3 data - Moodle Universidad de Córdoba.

[3] NetworkX documentation.