Solving 0/1 knapsack problem using champions league optimization algorithm

Fadhluddin bin Sahlan (1817445)

Muhammad Hassan Bin M Ashhuri (1812637)

Click here for video presentation

CSC 2301: Intelligent Systems

Instructor: Dr. Amir 'Aatieff Bin Amir Hussin

Date of submission: 5th August 2020

Solving 0/1 knapsack problem using Champions League Algorithm

There are many mutants with their own power value and weight of stones they can carry. The tasks asking to program a subroutine for the sentinels to recognize which mutant need to capture in order to catch the highest value group of mutants. We need to program the sentinels so it can determine the best combination of mutants the sentinel can carry on maximizing the efficiency and value of its bounty. This problem is the same as the 0/1 knapsack problem. The knapsack problem is a complex problem with the aim to find maximum value with the weight needed to be less than knapsack capacity. This problem has been used in many sectors such as industry, transportation, resource management and many more. Solving knapsack problems using old traditional methods can be tiring and take long computation time as the number of objects increases.

This paper will be focusing on solving the problem by using a new innovative algorithm which is the Champions League Algorithm. The result of this research shows that this algorithm is able to solve Sentinel-mutant problem (0/1 knapsack problem) which it achieves to obtain optimal value in an impressively short amount of time similar to the one that obtained by using traditional brute force algorithm which take about 14 minutes for one complete run to obtain optimal solution. The algorithm is implementing heuristics search and not exhaustive search which searches throughout the search space. The algorithm only searches part of the search space and improves the searched solutions repeatedly.

Background: Brief description about algorithm and its inspiration

Figure 1: Champions League

The Champions League Algorithm is basically an inspiration from the UEFA Champions League Cup (UCL). Why Champions? Because, we want to take the winning solution as an optimum value. The solution must face several stages (based on the parameters) and fight to win as the best solution. It is the same as the Champions League Cup where there are several groups. Then, the team will fight each other and the best one will head to the next stage which are quarter final, semi final and final. This algorithm also has an implementation of a genetic algorithm by using the crossover and mutation features. However, the way this algorithm uses the genetic algorithm is not the same as the usual one.

Methodology of champions league algorithm

The algorithm basically contains several parameters that need to be tuned in order to achieve a good solution at the end of the execution. The algorithm is taking the idea of mutation and crossover in genetic algorithms to enhance the quality of solutions but the way it is implemented is different from genetic algorithms. It starts by initializing grouping parameters which are, number of groups, number of solutions generated per group which the number of these parameters will be the power of two ($n = 2^k$). Then a set of groups will be produced with a number of solutions inside the group being generated randomly.

Just like in champions league which has quarter final, semi final, and final this algorithm also implements this idea by pairing the groups and letting them fight between them in every stage. The term fight here is to perform crossover and mutation on the solutions of those pairs of groups. Before the execution of the algorithm there will be several parameters that need to be initialized which are, number of crossover points, number of mutation points, mutation iteration, crossover iteration and final iteration. Each solution in the group will be sorted from high to low according to the power before the crossover. Each element (solution) in the group will be crossover with the element of its respective pair group at the same index. The crossover will occur at 8 points (as set in the number of crossover points) where the points of the crossover are randomly generated and it is not necessary for the crossover to occur in successive indexes but the index of the crossover points can occur in separate indexes (figure 2 explains this matter).

1	1	0	1	0	0	0	0	1	1	1	0	1	0	1	0
⇅		⇅	⇅	⇅	⇅			⇅			⇅		⇅		
1	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0

Figure 2: The implementation of crossover

The mutation occurs at a number of random points (number of mutation points) as set during the initialization of genetic operator parameters. Once the genetic operators are

performed, the new solutions produced will be checked first whether its weight is less than knapsack max capacity and also check whether the new solution is better than the previous one. If these two conditions are not satisfied, the old solutions will not be replaced by the new one. The main purpose of these implementations is to improve the solutions and avoid getting stuck in local optima. The crossover and mutation will be repeated several times according to mutation iteration and crossover parameter that is initialized during the initialization stage. After this process, the groups will be ranked according to their total power of solutions and half of the groups will be eliminated from the game. All the above processes continue until there is only one single group left in the league.

The winning group then will perform genetic operators again between the solutions in the group repeatedly as set during the initialization stage its number of iteration (final iterator). The solutions then will be rank and half of the solutions will be eliminated from the winning group. The process continues until there is only one single solution left in the winning group and this solution will be chosen as the optimal solution produced from the algorithm. The algorithm explained in the flow chart in figure 3.

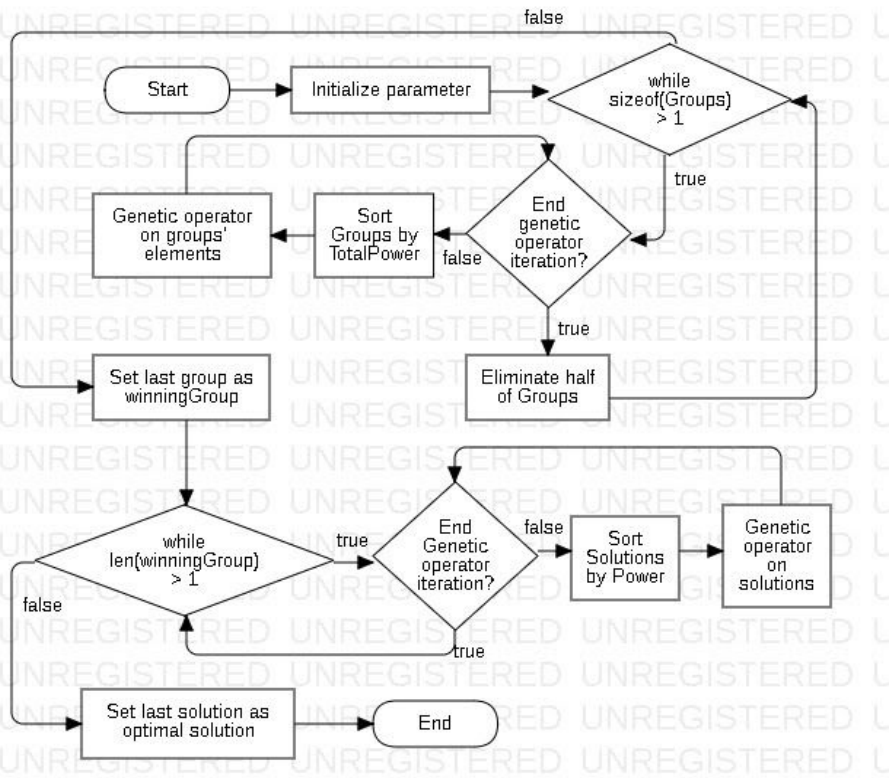


Figure 3: Flowchart of the algorithm

The coding also implements object oriented programming with there are class of Solution and class of Group created where the relationship between these classes are a Group can has one to multiple Solutions while every Solution can only have one single group. The class diagram is provided below in figure 4.

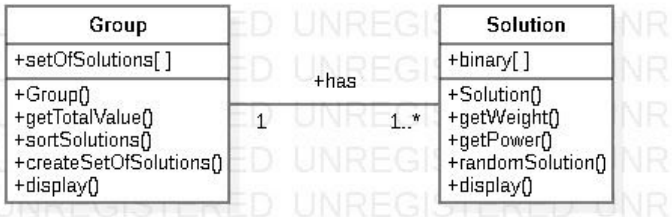
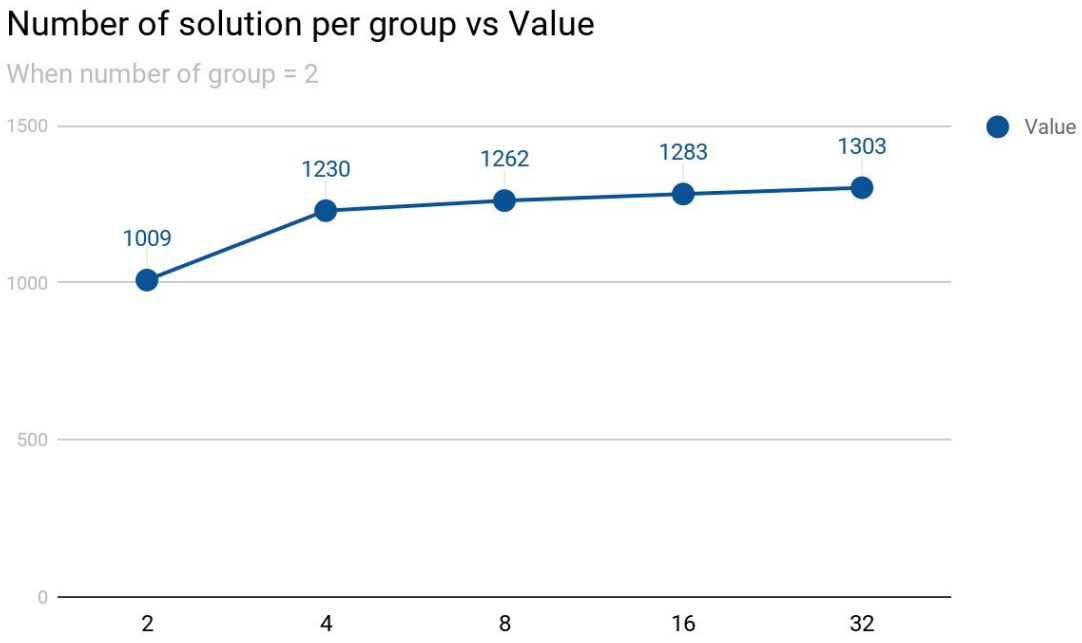


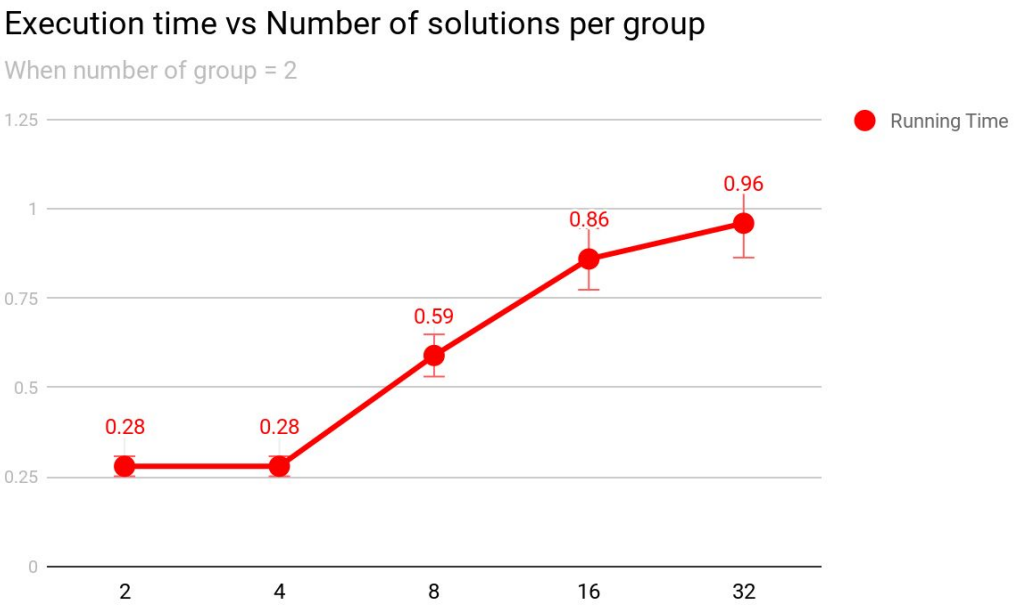
Figure 4: Class diagram of the algorithm

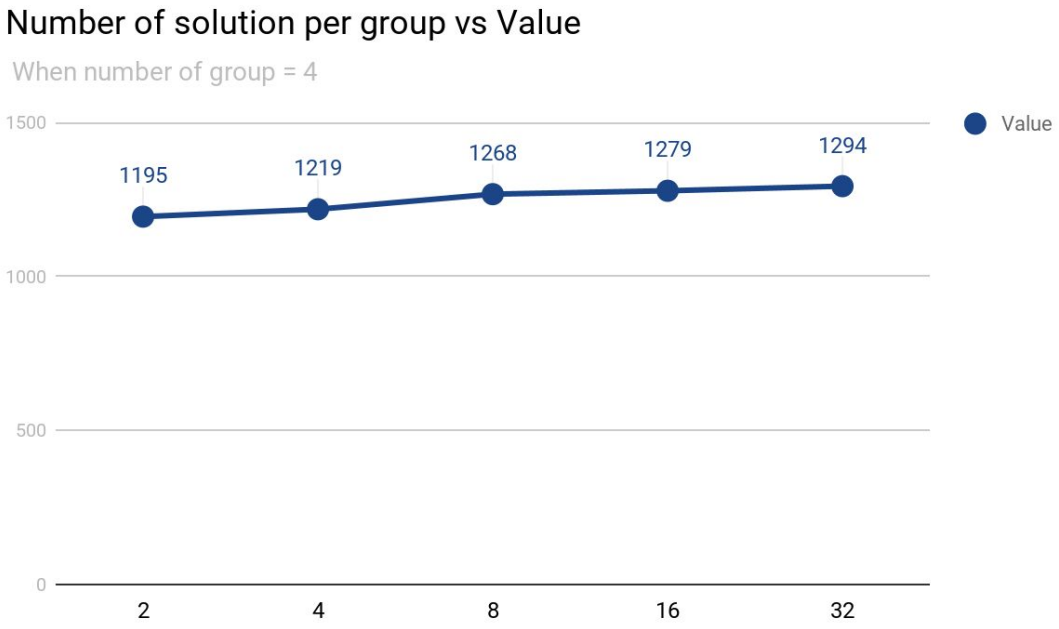
Evaluation of the algorithm

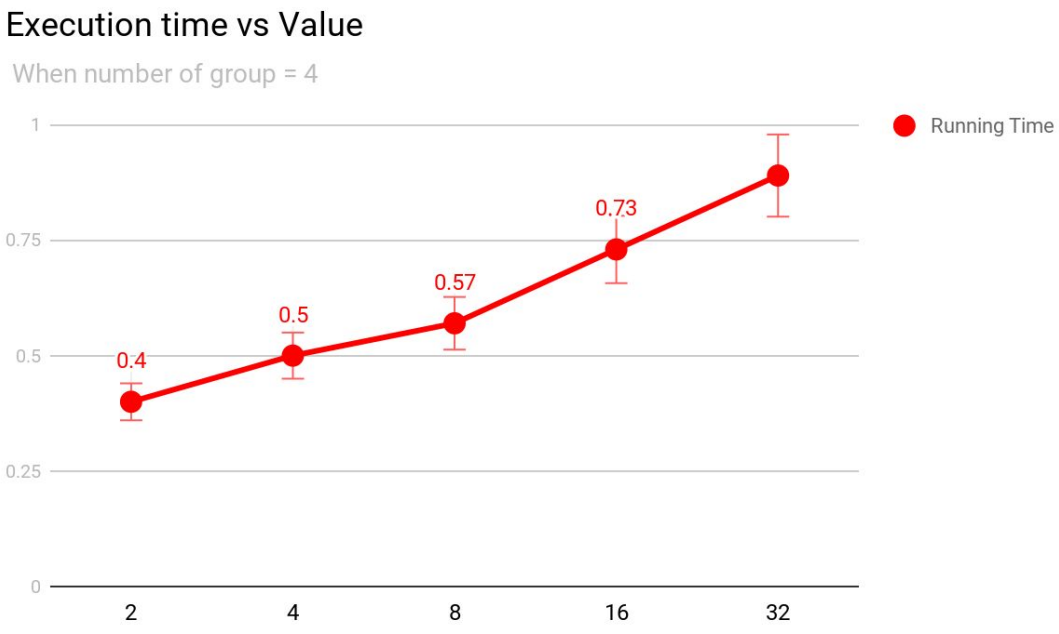
In this algorithm evaluation, we try to execute several testing on our algorithm by changing its parameters and try to observe the changes on the value. In this testing, we try to examine what factor that can affect the value on reaching the optimum value which is 1313. We will change the parameter number of the group and number solution per group to reach the optimum value.

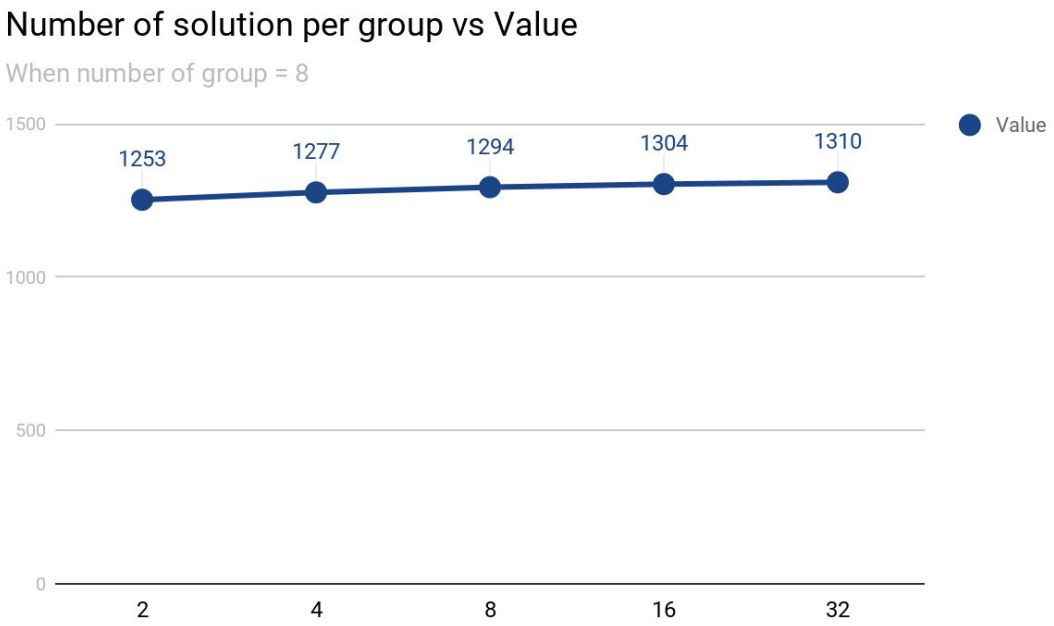
Test 1 - Here is the testing where the number of solutions per group as an independent variable while changing by 2^n and the number of groups as a fixed parameter. For the value of crossover iteration and mutation iteration both parameters are fixed to 10.

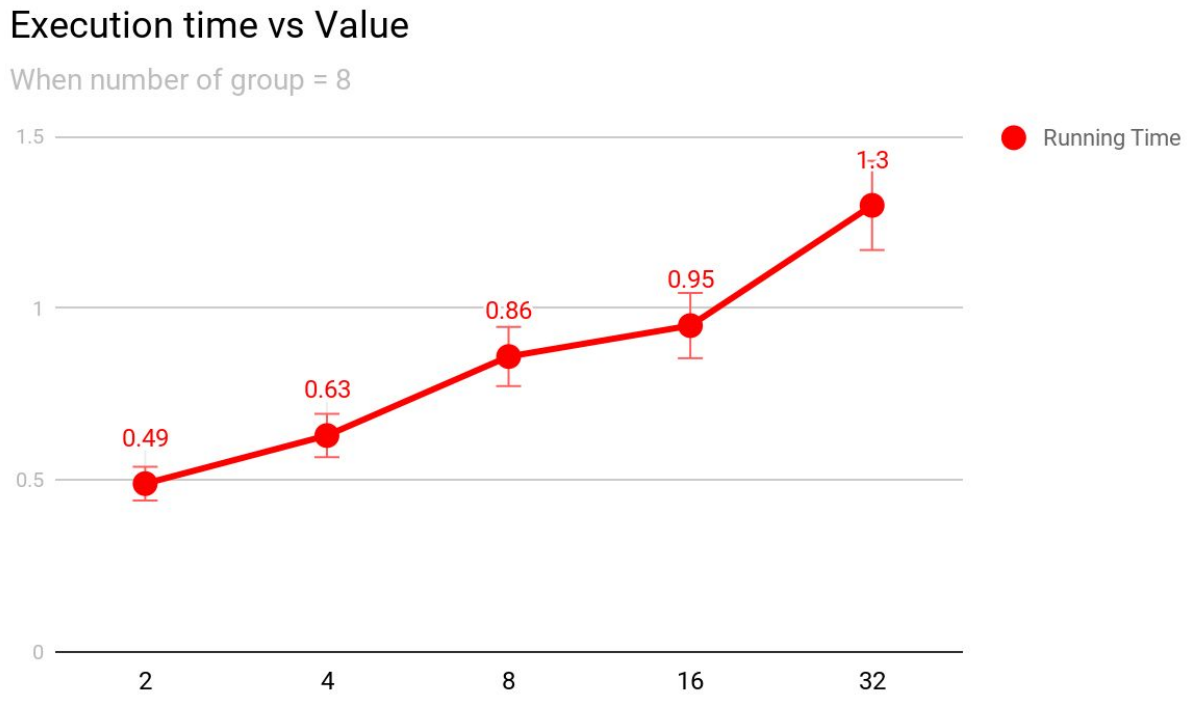


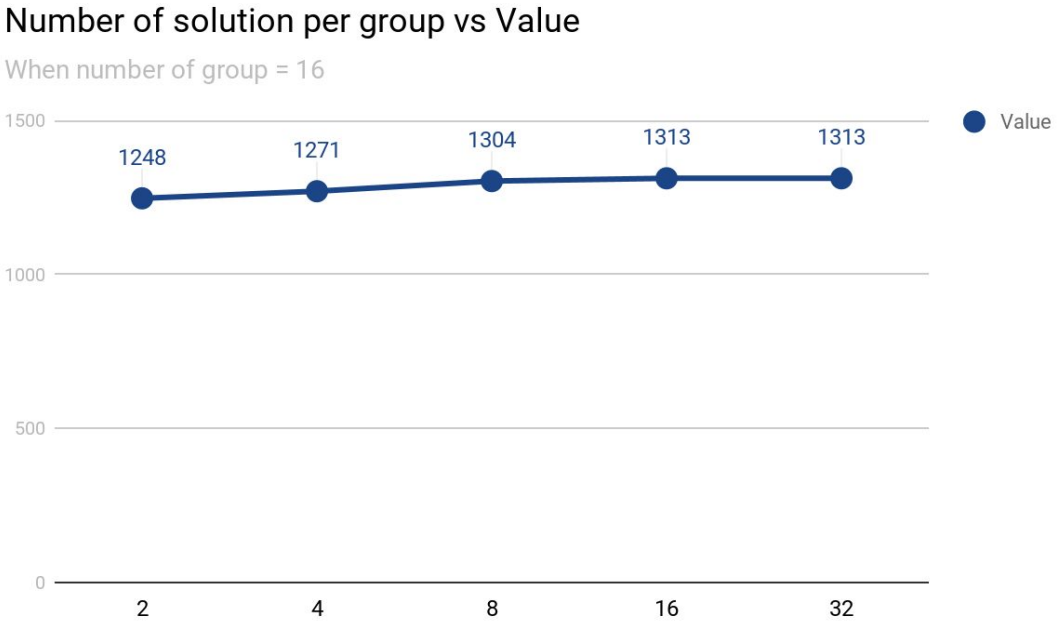

Execution time vs Number of solutions per group
When number of group = 2
Running Time
1.25
1
0.75
0.5
0.25
0
0.28
0.28
0.59
0.86
0.96
2
4
8
16
32

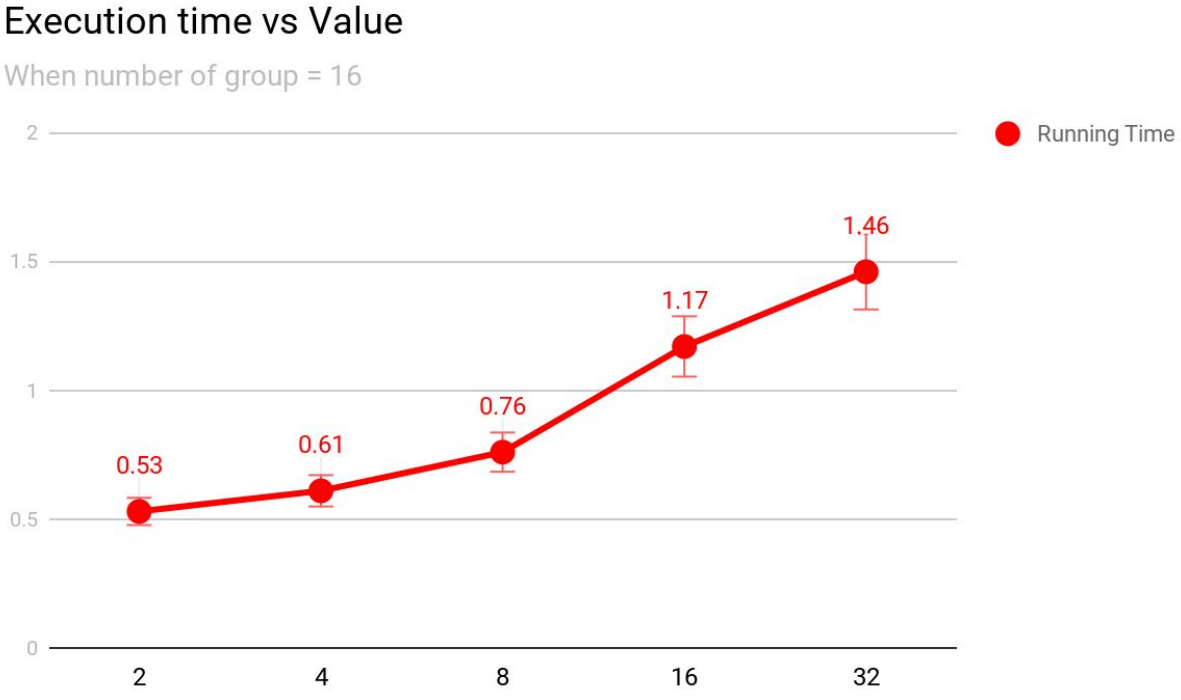

Number of solution per group vs Value
When number of group = 4
1500
1000
500
0
1195
1219
1268
1279
1294
2
4
8
16
32
Value

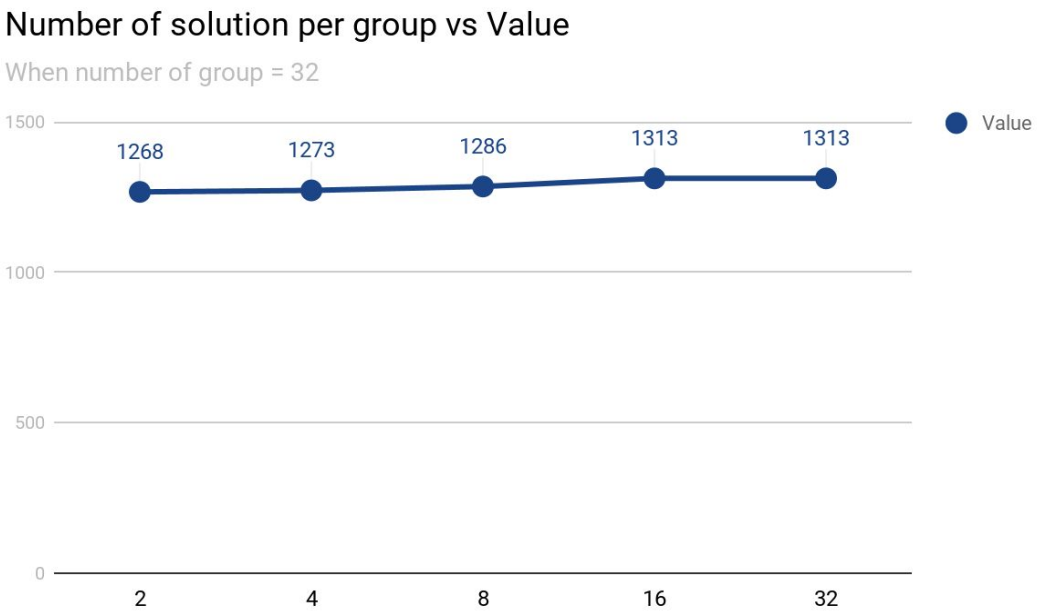

Execution time vs Value
When number of group = 4
1
0.75
0.5
0.25
0
0.4
0.5
0.57
0.73
2
4
8
16
32
Running Time

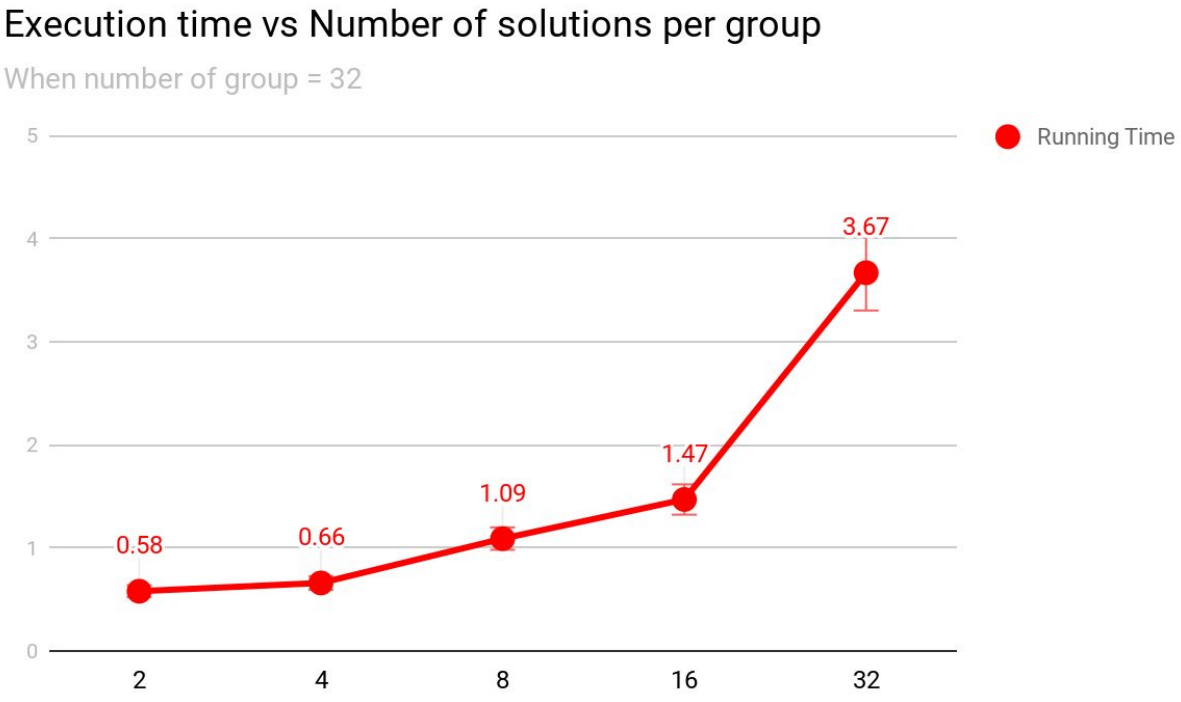

Number of solution per group vs Value
When number of group = 8
1500
1000
500
0
1253
1277
1294
1304
1310
2
4
8
16
32
Value


Execution time vs Value
When number of group = 8
1.5
1
0.5
0
0.49
0.63
0.86
0.95
1.3
2
4
8
16
32
Running Time

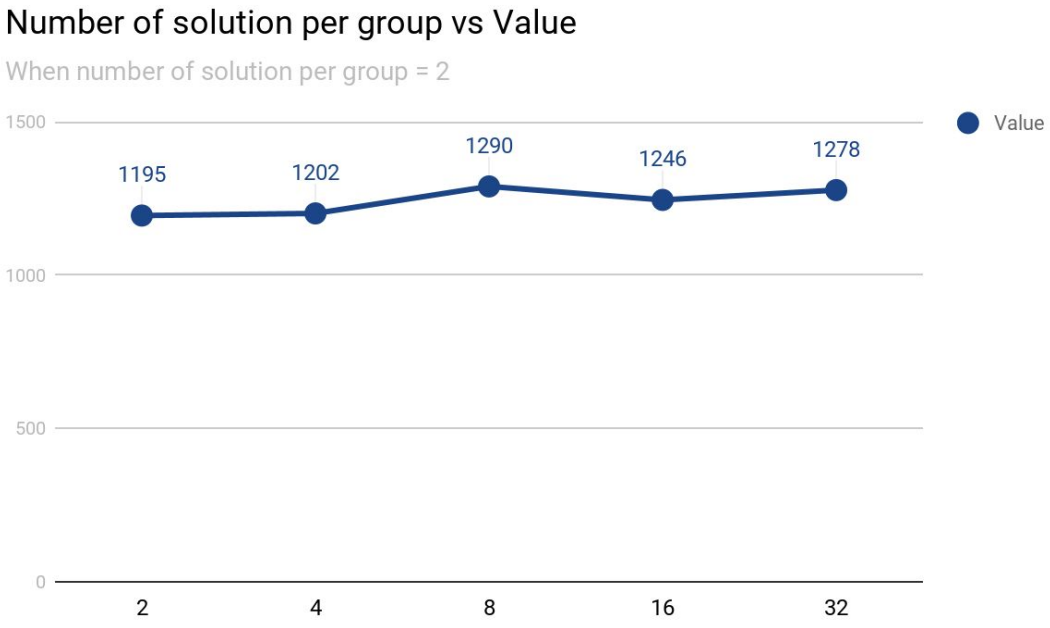

Number of solution per group vs Value
When number of group = 16
1500
1000
500
0
1248
1271
1304
1313
1313
2
4
8
16
32
Value

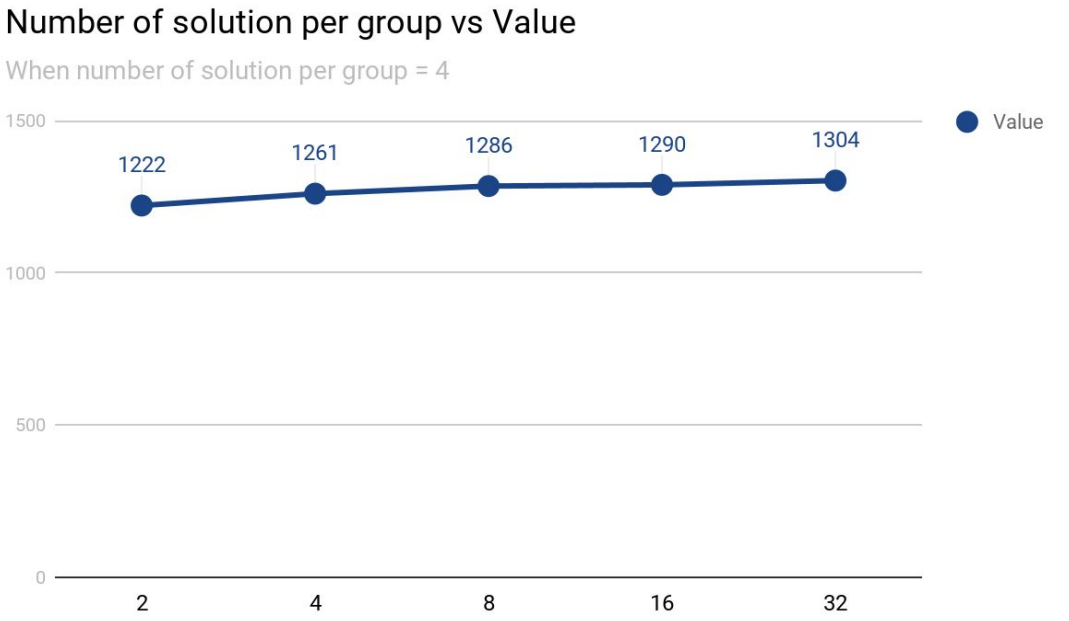

Execution time vs Value
When number of group = 16
2
1.5
1
0.5
0
0.53
0.61
0.76
1.17
1.46
2
4
8
16
32
Running Time

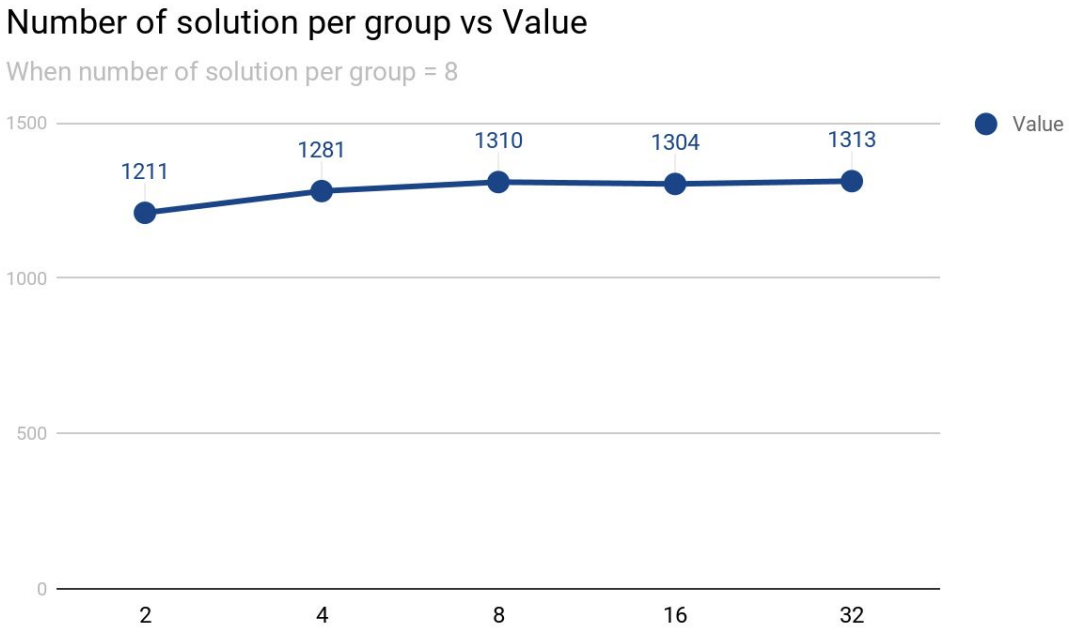

Number of solution per group vs Value
When number of group = 32
1500
1000
500
0
1268
1273
1286
1313
1313
Value
2
4
8
16
32

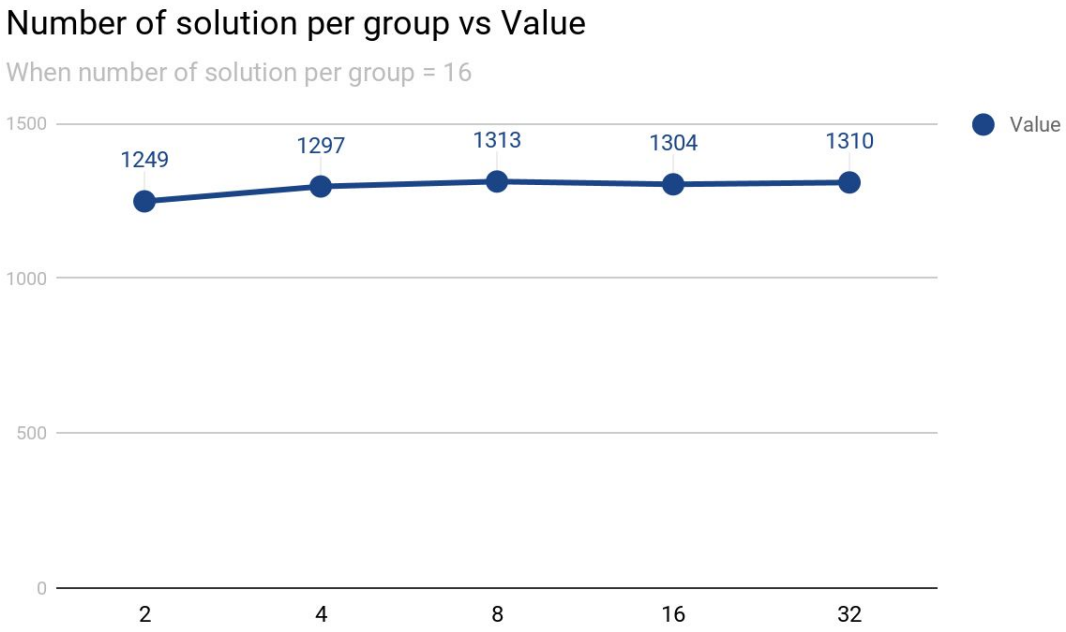

Execution time vs Number of solutions per group
When number of group = 32
5
4
3
2
1
0
0.58
0.66
1.09
1.47
3.67
Running Time
2
4
8
16
32

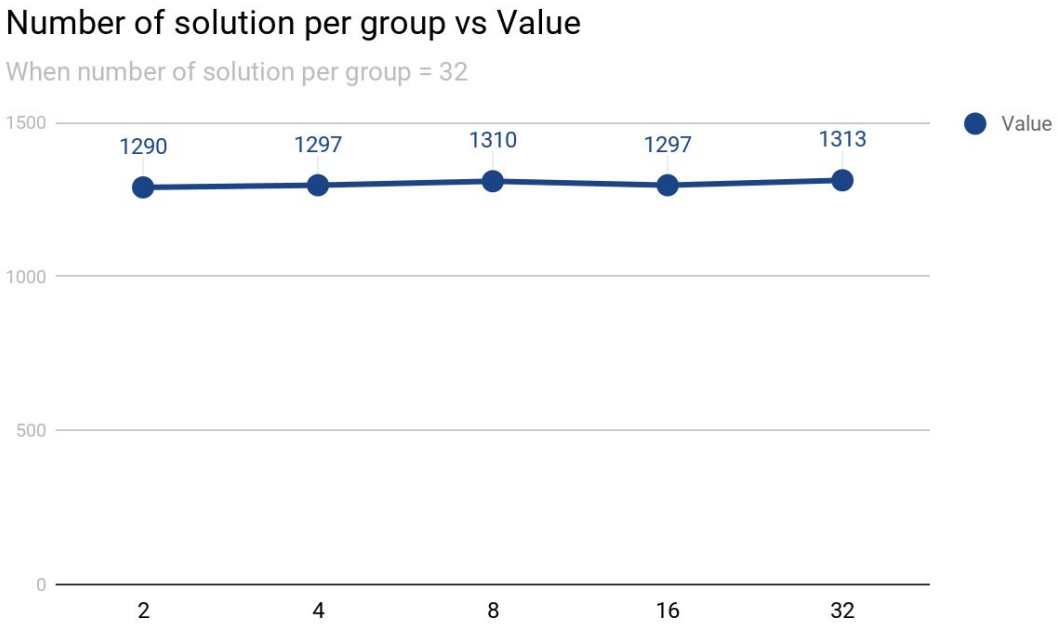
Test 2 - Here is the testing where the number of groups as an independent variable and it is changing by 2^n while the number of solutions per group as a fixed parameter. For the value of crossover iteration and mutation iteration both parameters are fixed to 10.




Number of solution per group vs Value
When number of solution per group = 4
1500
1000
500
0
1222
1261
1286
1290
1304
2
4
8
16
32
Value

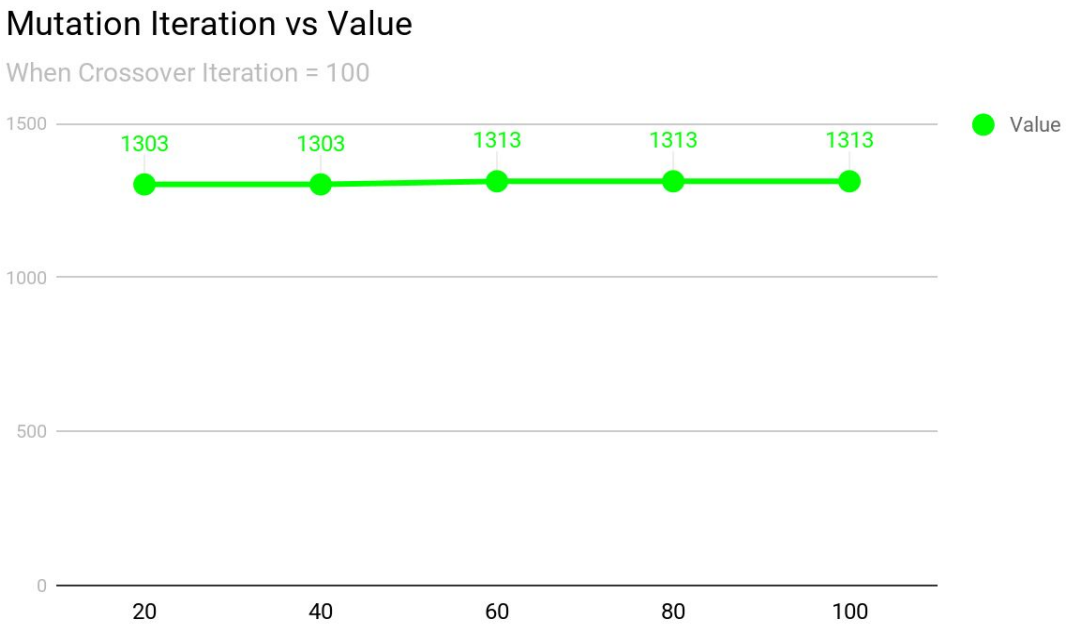

Number of solution per group vs Value
When number of solution per group = 8
1500
1000
500
0
1211
1281
1310
1304
1313
2
4
8
16
32
Value

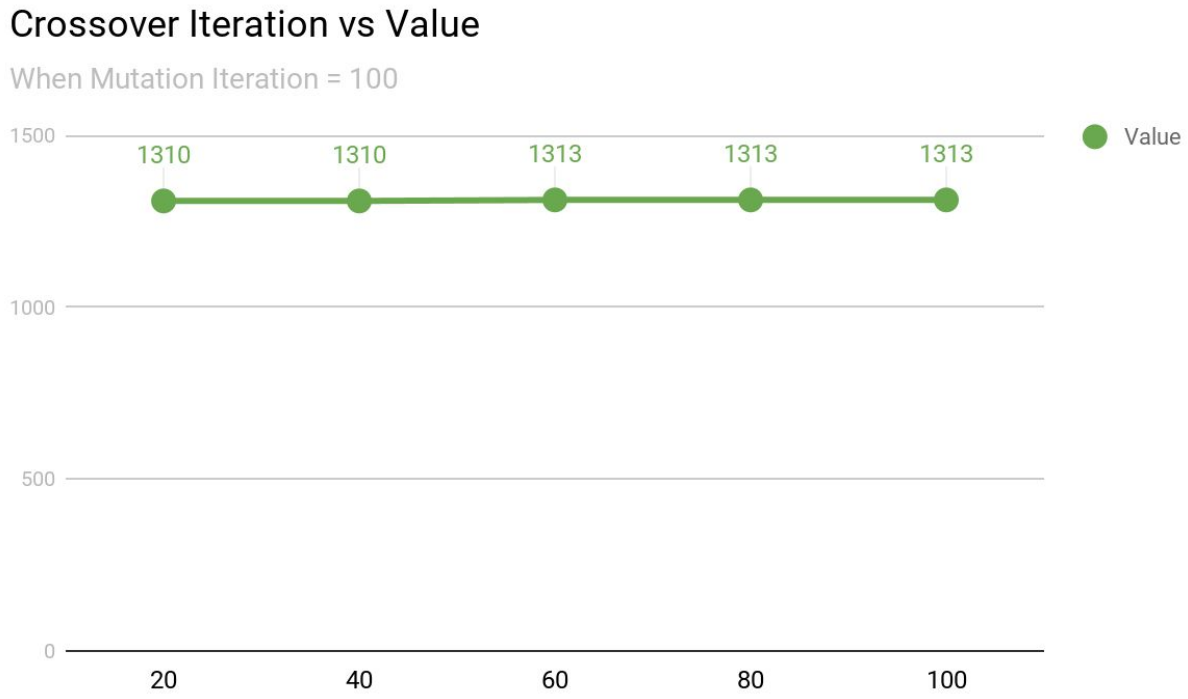




Test 3 - We fixed the number of solutions per group and number of groups to 8 while changing the value of mutation iteration and crossover iteration parameter for discovering is there any

effect on reaching the optimum value.





Conclusion

In conclusion, based on the evaluation of this algorithm, we are able to get the required optimum value which is 1313. Apart from that we manage to get the optimum value by time average equal to 1.9425 seconds which is approximately 420 times faster than brute force method. Also, from the result of this algorithm testing, we can see that the value was affected by changing the parameters value. For the test 1 and test 2, we can see that the optimal solution achieved by algorithm is increased when the value of the number of groups or number of solutions per group are increasing too, however, the running time also slightly increases. Moreover, on the test 3, we try to change the crossover and mutation iteration parameters value and result also as same as test 2 and test 3 which the optimal solution achieved by algorithm will increase when the parameters value are increasing too. Apart from that, when the parameters are increasing, the running time to execute will also increase too but it did not take very much time to reach the optimum value compared to the brute force method. From the result of our findings, we can conclude that this algorithm is successive to solve this problem on finding the optimum solution within a short amount of time. We hope that this idea of algorithms can be beneficial in solving problems in the most efficient way. We also might be improving this algorithm to make it more efficient to find the optimum value since the problem in the future will become more complex and take time to solve.