Icon

Description automatically generated

**AGU IE 461 MANUFACTURUNG SYSTEMS**

**TERM PROJECT FINAL REPORT**

**SUBMITTED TO**

**THE DEPARTMENT OF INDUSTRIAL ENGINEERING OF**

**ABDULLAH GUL UNIVERSITY**

**By**

**Mevlüt Gümüştepe**

**Jan 8, 2024**

**Kayseri / TURKEY**

**TABLE OF CONTENTS**

[**1.** **ASSEMBLY LINE BALANCING WITH HEURISTICS** 1](#_Toc155585873)

[**1.1.** **RANDOM SEARCH ALGORITHM** 1](#_Toc155585874)

[**1.2.** **SHORTEST TASK TIME ALGORITHM** 5](#_Toc155585875)

[**1.3.** **LONGEST TASK TIME ALGORITHM** 8](#_Toc155585876)

[**1.4.** **LOCAL SEARCH ALGORITHM** 11](#_Toc155585877)

[**2.** **RESULTS OF THE ALGORITHMS FOR TEST DATASETS** 14](#_Toc155585878)

[**2.1.** **RANDOM SEARCH ALGORITHM RESULTS** 14](#_Toc155585879)

[**2.2.** **SHORTEST TASK TIME ALGORITHM RESULTS** 16](#_Toc155585880)

[**2.3.** **LONGEST TASK TIME ALGORITHM RESULTS** 18](#_Toc155585881)

[**2.4.** **LOCAL SEARCH ALGORITHM RESULTS** 20](#_Toc155585882)

[**2.5.** **OVERALL RESULTS** 22](#_Toc155585883)

[**3.** **CONCLUSION** 23](#_Toc155585884)

[**4.** **APPENDICES** 24](#_Toc155585885)

**LIST OF TABLES**

[Table 1. Number of Tasks and Cycle Times for Different Datasets 14](#_Toc155586016)

[Table 2. Random Search Algorithm Results 22](#_Toc155586017)

[Table 3. Shortest Task Time Results 22](#_Toc155586018)

[Table 4. Longest Task Time Results 23](#_Toc155586019)

[Table 5. Local Search Algorithm (with Random Search) Results 23](#_Toc155586020)

[Table 6. Local Search Algorithm (with Shortest Task Time) Results 23](#_Toc155586021)

[Table 7. Local Search Algorithm (with Longest Task Time) Results 23](#_Toc155586022)

**LIST OF FIGURES**

[Figure 1. Python Script for Random Search - Part 1 2](#_Toc155586038)

[Figure 2.Python Script for Random Search - Part 2 3](#_Toc155586039)

[Figure 3.Python Script for Random Search - Part 3 4](#_Toc155586040)

[Figure 4.Python Script for Random Search - Part 4 4](#_Toc155586041)

[Figure 5.Python Script for Shortest Task Time - Part 1 5](#_Toc155586042)

[Figure 6.Python Script for Shortest Task Time - Part 2 6](#_Toc155586043)

[Figure 7.Python Script for Shortest Task Time - Part 3 7](#_Toc155586044)

[Figure 8.Python Script for Shortest Task Time - Part 4 7](#_Toc155586045)

[Figure 9.Python Script for Longest Task Time - Part 1 8](#_Toc155586046)

[Figure 10.Python Script for Longest Task Time - Part 2 9](#_Toc155586047)

[Figure 11.Python Script for Longest Task Time - Part 3 10](#_Toc155586048)

[Figure 12.Python Script for Longest Task Time - Part 4 10](#_Toc155586049)

[Figure 13.Python Script for Local Search with Swap Method - Part 1 11](#_Toc155586050)

[Figure 14.Python Script for Local Search with Swap Method - Part 2 12](#_Toc155586051)

[Figure 15.Python Script for Local Search with Swap Method - Part 3 12](#_Toc155586052)

[Figure 16.Python Script for Local Search with Swap Method - Part 4 13](#_Toc155586053)

[Figure 17.Python Script for Local Search with Swap Method - Part 5 13](#_Toc155586054)

[Figure 18. Random Search Result of Jackson Dataset with Cycle Time 10 14](#_Toc155586055)

[Figure 19. Random Search Result of Jackson Dataset with Cycle Time 19 14](#_Toc155586056)

[Figure 20. Random Search Result of Kilbridge Dataset with Cycle Time 56 15](#_Toc155586057)

[Figure 21. Random Search Result of Kilbridge Dataset with Cycle Time 150 15](#_Toc155586058)

[Figure 22. Random Search Result of Tonge Dataset with Cycle Time 250 15](#_Toc155586059)

[Figure 23. Random Search Result of Tonge Dataset with Cycle Time 600 15](#_Toc155586060)

[Figure 24. Random Search Result of Arcus Dataset with Cycle Time 3786 15](#_Toc155586061)

[Figure 25.Random Search Result of Arcus Dataset with Cycle Time 9000 16](#_Toc155586062)

[Figure 26. Shortest Task Time Result of Jackson Dataset with Cycle Time 10 16](#_Toc155586063)

[Figure 27/ Shortest Task Time Result of Jackson Dataset with Cycle Time 19 16](#_Toc155586064)

[Figure 28. Shortest Task Time Result of Kilbridge Dataset with Cycle Time 56 17](#_Toc155586065)

[Figure 29. Shortest Task Time Result of Kilbridge Dataset with Cycle Time 150 17](#_Toc155586066)

[Figure 30. Shortest Task Time Result of Tonge Dataset with Cycle Time 250 17](#_Toc155586067)

[Figure 31. Shortest Task Time Result of Tonge Dataset with Cycle Time 600 17](#_Toc155586068)

[Figure 32.Shortest Task Time Result of Arcus Dataset with Cycle Time 3786 18](#_Toc155586069)

[Figure 33. Shortest Task Time Result of Arcus Dataset with Cycle Time 9000 18](#_Toc155586070)

[Figure 34. Longest Task Time Result of Jackson Dataset with Cycle Time 10 18](#_Toc155586071)

[Figure 35. Longest Task Time Result of Jackson Dataset with Cycle Time 19 18](#_Toc155586072)

[Figure 36. Longest Task Time Result of Kilbridge Dataset with Cycle Time 56 19](#_Toc155586073)

[Figure 37. Longest Task Time Result of Kilbridge Dataset with Cycle Time 150 19](#_Toc155586074)

[Figure 38. Longest Task Time Result of Tonge Dataset with Cycle Time 250 19](#_Toc155586075)

[Figure 39. Longest Task Time Result of Tonge Dataset with Cycle Time 600 19](#_Toc155586076)

[Figure 40. Longest Task Time Result of Arcus Dataset with Cycle Time 3786 20](#_Toc155586077)

[Figure 41. Longest Task Time Result of Arcus Dataset with Cycle Time 9000 20](#_Toc155586078)

[Figure 42. Local Search Algorithm (with Random Search Results) Example - 1 21](#_Toc155586079)

[Figure 43. Local Search Algorithm (with Shortest Task Time Results) Example – 2 21](#_Toc155586080)

[Figure 44. Local Search Algorithm (with Longest Task Time Results) Example – 3 22](#_Toc155586081)

# **ASSEMBLY LINE BALANCING WITH HEURISTICS**

Assembly line balancing is a production planning problem to assign tasks to workstations to make more efficient and balanced the workflow. There are several types of assembly lines such as multi-manned, single-manned, deterministic task time, and stochastic task time, etc. Additionally, there are two types of assembly lines which are type-1 that considers minimization of the number of stations, and type-2 that considers minimization of the cycle time. In this report, the type-1 assembly line balancing problem is considered. To solve this problem, several heuristic methods which are Random Search Algorithm, Shortest Task Time Algorithm, Longest Task Time Algorithm, and Local Search Algorithm with the results of Random Search, Shortest Task Time, and Longest Task Time algorithms.

## **RANDOM SEARCH ALGORITHM**

Random search algorithm is the method to search the solution space by considering random points within several limitations such as precedence relations, cycle time, number of tasks, etc. The random search algorithm used for assembly line balancing consists of several steps. Firstly, starting point/points of an assembly line are considered as available tasks. Then, one of the available tasks is chosen randomly. At every step, available tasks are updated according to precedence relations. The chosen task will be added to the current station if the cycle time is not exceeded. Otherwise, the task will be added to a new station. The algorithm works until all the tasks are completed.

A computer screen shot of text

Description automatically generated

Figure 1. Python Script for Random Search - Part 1

A computer screen shot of text

Description automatically generated

Figure 2.Python Script for Random Search - Part 2

A computer screen shot of text

Description automatically generated

Figure 3.Python Script for Random Search - Part 3

A black background with text

Description automatically generated

Figure 4.Python Script for Random Search - Part 4

## **SHORTEST TASK TIME ALGORITHM**

One of the methods for balancing the assembly lines is the shortest task time algorithm. Shortest task time algorithm starts with available starting point/points of an assembly line. At every step, the task with the shortest task time is selected and available tasks are updated according to the precedence relations. Then, if the selected task’s task time does not exceed the cycle time, it will be added to the current station, otherwise, it will be added to a new station. The algorithm works until all the tasks are completed. The shortest task time algorithm gives mostly several solutions since at every step there is only one task with the shortest task time.

A screen shot of a computer code

Description automatically generated

Figure 5.Python Script for Shortest Task Time - Part 1

A screen shot of a computer program

Description automatically generated

Figure 6.Python Script for Shortest Task Time - Part 2

A computer screen shot of text

Description automatically generated

Figure 7.Python Script for Shortest Task Time - Part 3

A black background with text

Description automatically generated

Figure 8.Python Script for Shortest Task Time - Part 4

## **LONGEST TASK TIME ALGORITHM**

Longest task time is an algorithm that is used to balance assembly lines. Longest task time algorithm chooses tasks with longest task time at every step. The algorithm starts with starting point/points of an assembly line and chooses the task with highest task time. Then, at every step, available tasks are determined according to precedence relations and the task with highest task time is chosen. If the selected task exceeds the cycle time and another task does not exceed the cycle time in the available task list, the task that does not exceed the cycle time is selected. Otherwise, the task with the longest task time is assigned to a new station. The algorithm works until all the tasks are completed. The longest task time algorithm gives mostly several solutions since at every step there is only one task with the longest task time.

A computer screen shot of text

Description automatically generated

Figure 9.Python Script for Longest Task Time - Part 1

A screen shot of a computer

Description automatically generated

Figure 10.Python Script for Longest Task Time - Part 2

A screen shot of a computer code

Description automatically generated

Figure 11.Python Script for Longest Task Time - Part 3

A screenshot of a computer program

Description automatically generated

Figure 12.Python Script for Longest Task Time - Part 4

## **LOCAL SEARCH ALGORITHM**

Local search algorithm is an algorithm that is used to develop the solution obtained from random search algorithm, shortest task time algorithm, and longest task time algorithm in assembly line balancing type-1 problem. In this project, the local search algorithm is used with swap method. Therefore, the best solution obtained from the above algorithms is considered as an initial solution and the random swap values are assigned to each task in the initial solution. Then, randomly chosen two swap values are swapped at every step. While choosing a task, the swap values are considered and the task with highest swap value is selected. The available task list is updated at each step considering precedence relations and the algorithm works until all the tasks are completed.

A screen shot of a computer

Description automatically generated

Figure 13.Python Script for Local Search with Swap Method - Part 1

A screen shot of a computer code

Description automatically generated

Figure 14.Python Script for Local Search with Swap Method - Part 2

A computer screen shot of text

Description automatically generated

Figure 15.Python Script for Local Search with Swap Method - Part 3

A screen shot of a computer code

Description automatically generated

Figure 16.Python Script for Local Search with Swap Method - Part 4

A screen shot of a computer code

Description automatically generated

Figure 17.Python Script for Local Search with Swap Method - Part 5

# **RESULTS OF THE ALGORITHMS FOR TEST DATASETS**

The python scripts written for heuristic methods mentioned above are tested using several datasets which are Jackson, Kilbridge, Tonge, and Arcus datasets for assembly line balancing problems. Each task has a different number of tasks. Each dataset is solved using two different cycle times since the problem considered in this project is a type-1 assembly line balancing problem.

Table 1. Number of Tasks and Cycle Times for Different Datasets

|  |  |  |  |
| --- | --- | --- | --- |
| DATASET | NUMBER OF TASKS | CYCLE TIME 1 | CYCLE TIME 2 |
| JACKSON | 11 | 10 | 19 |
| KILBRIDGE | 45 | 56 | 150 |
| TONGE | 70 | 250 | 600 |
| ARCUS | 83 | 3786 | 9000 |

## **RANDOM SEARCH ALGORITHM RESULTS**

The test datasets are solved for different cycle times using random search algorithm. The random search algorithm runs for the number of tasks times 100 to get better results and the best result obtained from random search algorithm is saved for initial solution of local search algorithm.

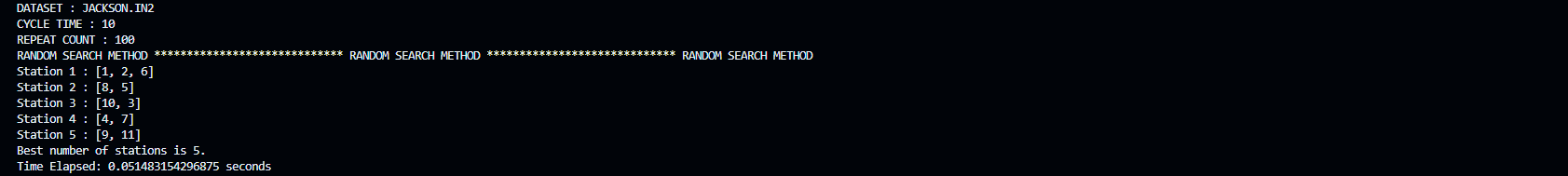


Figure 18. Random Search Result of Jackson Dataset with Cycle Time 10

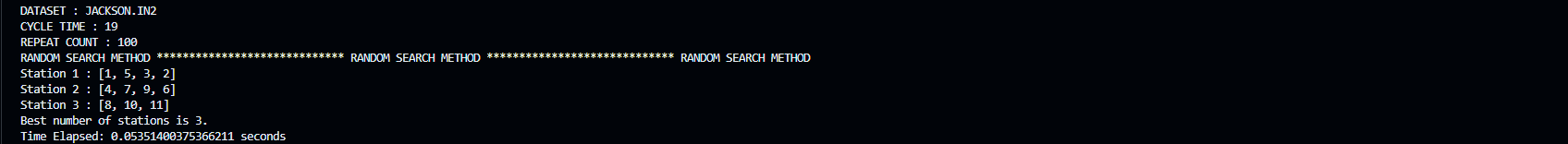


Figure 19. Random Search Result of Jackson Dataset with Cycle Time 19

A black background with white text

Description automatically generated

Figure 20. Random Search Result of Kilbridge Dataset with Cycle Time 56

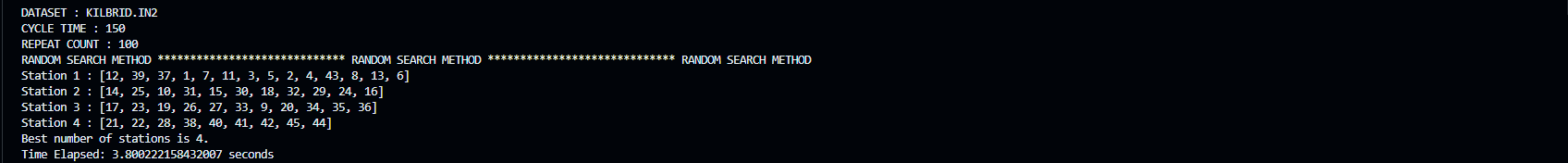


Figure 21. Random Search Result of Kilbridge Dataset with Cycle Time 150

A black background with colorful text

Description automatically generated

Figure 22. Random Search Result of Tonge Dataset with Cycle Time 250

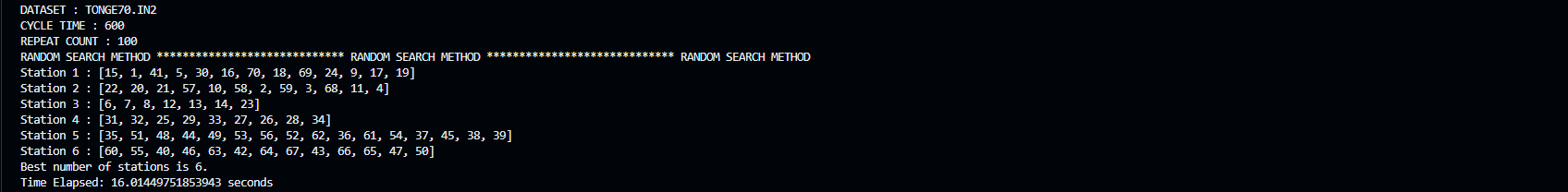


Figure 23. Random Search Result of Tonge Dataset with Cycle Time 600

A black screen with a black background

Description automatically generated

Figure 24. Random Search Result of Arcus Dataset with Cycle Time 3786

A black background with white text

Description automatically generated

Figure 25.Random Search Result of Arcus Dataset with Cycle Time 9000

As seen in the results above, all the tests result with optimal solution except Arcus Dataset with Cycle Time 3786 and Kilbridge Dataset with Cycle Time 56. Reaching optimal results is not guaranteed in random search algorithm, however, the results obtained from the random search algorithm is good and it can be improved if repeat count is increased.

## **SHORTEST TASK TIME ALGORITHM RESULTS**

The assembly line balancing problems mentioned above are solved using shortest task time algorithm with different cycle times. In most of the datasets, optimal results or results really close to optimal results are obtained.

A black background with many small colored lines

Description automatically generated with medium confidence

Figure 26. Shortest Task Time Result of Jackson Dataset with Cycle Time 10

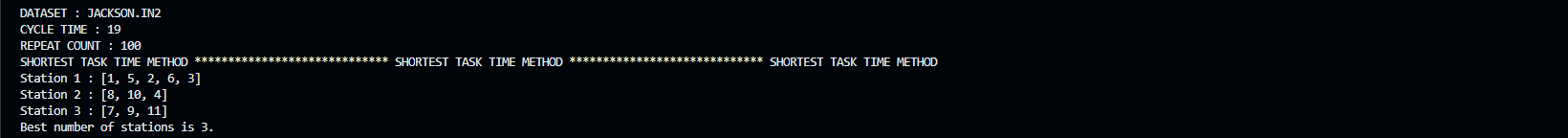


Figure 27/ Shortest Task Time Result of Jackson Dataset with Cycle Time 19

A black background with a black border

Description automatically generated with medium confidence

Figure 28. Shortest Task Time Result of Kilbridge Dataset with Cycle Time 56

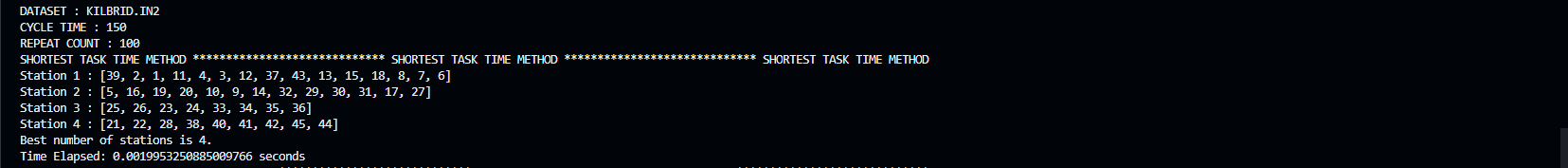


Figure 29. Shortest Task Time Result of Kilbridge Dataset with Cycle Time 150

A black background with a black border

Description automatically generated with medium confidence

Figure 30. Shortest Task Time Result of Tonge Dataset with Cycle Time 250

A black background with colorful lines

Description automatically generated

Figure 31. Shortest Task Time Result of Tonge Dataset with Cycle Time 600

A black screen with white text

Description automatically generated

Figure 32.Shortest Task Time Result of Arcus Dataset with Cycle Time 3786

A black background with blue and white text

Description automatically generated

Figure 33. Shortest Task Time Result of Arcus Dataset with Cycle Time 9000

## **LONGEST TASK TIME ALGORITHM RESULTS**

The assembly line balancing problems mentioned above are solved using shortest task time algorithm with different cycle times. In most of the datasets, optimal results or results really close to optimal results are obtained.

A black background with colorful lines

Description automatically generated

Figure 34. Longest Task Time Result of Jackson Dataset with Cycle Time 10

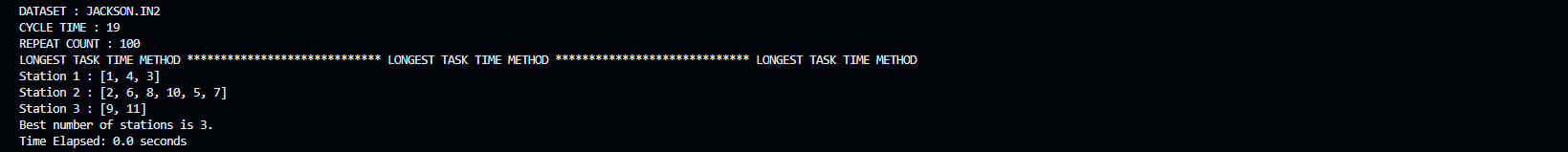


Figure 35. Longest Task Time Result of Jackson Dataset with Cycle Time 19

A black background with a black border

Description automatically generated with medium confidence

Figure 36. Longest Task Time Result of Kilbridge Dataset with Cycle Time 56

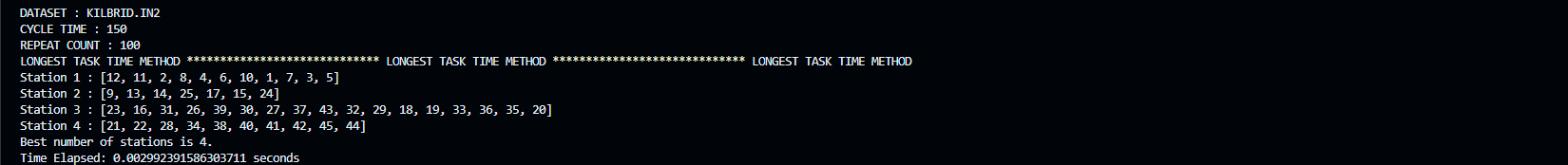


Figure 37. Longest Task Time Result of Kilbridge Dataset with Cycle Time 150

A black screen with a black background

Description automatically generated

Figure 38. Longest Task Time Result of Tonge Dataset with Cycle Time 250

A black background with colorful lights

Description automatically generated with medium confidence

Figure 39. Longest Task Time Result of Tonge Dataset with Cycle Time 600

A black screen with white text

Description automatically generated

Figure 40. Longest Task Time Result of Arcus Dataset with Cycle Time 3786

A black background with many small colored lines

Description automatically generated with medium confidence

Figure 41. Longest Task Time Result of Arcus Dataset with Cycle Time 9000

## **LOCAL SEARCH ALGORITHM RESULTS**

Local search algorithm is applied to results of random search, shortest task time and longest task time algorithms. In most cases, the initial solution is developed or remains the same as the initial station number. The reason that the local search algorithm does not develop the initial solution is random search algorithm works very well. In the results obtained for shortest task time and longest task time initial solutions are developed most of time since these algorithms gives results that are not optimal.

A black background with white lines

Description automatically generated

Figure 42. Local Search Algorithm (with Random Search Results) Example - 1

A black background with white lines

Description automatically generated

Figure 43. Local Search Algorithm (with Shortest Task Time Results) Example – 2

A black background with white lines

Description automatically generated

Figure 44. Local Search Algorithm (with Longest Task Time Results) Example – 3

## **OVERALL RESULTS**

The results obtained from random search algorithm, shortest task time algorithm, longest task time algorithm, and local search algorithm are entered to tables and calculation times and gaps from optimal solution are determined. When all the results are compared, random search algorithm and local search algorithm with random search result gives best station numbers as optimal station numbers except Kilbridge problem with a cycle time of 56 and Arcus problem with a cycle time of 3786 gives non-optimal solutions. When the local search algorithm runs with shortest task time result or longest task time result, it performs worse than the local search algorithm with random search results because it starts with worse solutions than the random search algorithm.

Table 2. Random Search Algorithm Results



Table 3. Shortest Task Time Results



Table 4. Longest Task Time Results



Table 5. Local Search Algorithm (with Random Search) Results



Table 6. Local Search Algorithm (with Shortest Task Time) Results



Table 7. Local Search Algorithm (with Longest Task Time) Results



# **CONCLUSION**

Assembly line balancing problems are one of the important problems in the production industry. There are several types of assembly lines in the sector such as deterministic task time, stochastic task time, automated, etc. Balancing the assembly lines is important for balanced and continuous production. The assembly line balancing method can be solved using mathematical models or several heuristics like random search, local search, shortest task time, longest task time, and simulated annealing, etc.

Developing mathematical models can be complex since assembly line balancing is an np-hard problem. Additionally, optimal results cannot be obtained due to complexity within a reasonable time. Therefore, heuristic methods would be life-saving solution methods because assembly line balancing problems can be solved, and optimal solutions can be obtained within shorter times than the mathematical models.

As seen from the results of test datasets above, complex problems such as Tonge and Arcus can be solved with heuristics in less than a minute and results really close to optimal station numbers can be obtained. For smaller datasets like Jackson and Kilbridge, optimal results can be obtained with smaller repeat counts. Moreover, the solutions can be improved by increasing the repeat count of the algorithms. Finally, random search algorithm gives better results when it is compared with shortest task time and longest task time algorithms.

# **APPENDICES**

Random search algorithm, shortest task time algorithm, longest task time algorithm are coded with three separate files. Also, local search algorithm for each method is included at each file that runs with initial solution of above algorithms. All the files can be found on zip.

* Random Search and Local Search Python Script: mevlut\_random\_search.py
* Shortest Task Time and Local Search Python Script: mevlut\_random\_search.py
* Longest Task Time and Local Search Python Script: mevlut\_random\_search.py