University of Nottingham
UK | CHINA | MALAYSIA

Coursework: Optimizing Inventory Management with 1D Bin Packing

Yuyang ZHANG

20514470

scyyz26@nottingham.edu.cn

April 26, 2025

School of Computer Science
University of Nottingham Ningbo China

1 Introduction and Background

In this report, no artificial intelligence (AI) methods or frameworks were used in the development or execution of the algorithm.

The 1D Bin Packing Problem (1D BPP) is a combinatorial optimization problem that involves determining the most efficient way to pack a collection of items, each with a specified size, into a minimum number of uniform bins. The primary constraint is that the total size of the items assigned to any bin must not exceed the bin's predetermined capacity. The objective is to minimize the number of bins used while adhering to this capacity constraint (Martello & Toth, 1990).

Given that the 1D Bin Packing Problem (1D BPP) is strongly NP-hard, finding an exact solution is computationally intractable for large instances. Consequently, various heuristic and metaheuristic approaches, such as greedy algorithms, genetic algorithms, simulated annealing, and tabu search, are typically employed to obtain near-optimal solutions within a reasonable computational time (Delorme et al., 2016). These methods provide practical solutions by exploring the solution space more efficiently than exhaustive search algorithms.

The 1D Bin Packing Problem (1D BPP) plays a crucial role in optimizing inventory management, storage, and transportation by minimizing space wastage in warehouses and shipping containers. Delorme et al. (2016) underscore its significance in reducing material waste in manufacturing, such as in paper production, while Munien and Ezugwu (2021) highlight its broad industrial applications, including cutting, such as wood and glass, packing, such as transportation and warehousing, and supply chain management,such as vehicle/container loading, cutting stock, trim loss, packaging design, resource allocation, and project scheduling.

2 Methodology

The methodology combines a random search with a heuristic algorithm, primarily an optimized genetic algorithm (GA), to address the one-dimensional Bin Packing Problem (BPP). The random_search_fit function iterates over multiple random permutations of the item sequence, applying either a greedy heuristic or a GA to each permutation and selecting the solution with the fewest bins. Random shuffling introduces diversity, mitigating the local optimality limitations inherent in greedy methods, which are typically deterministic and locally focused in their decision-making (Johnson, 1974). Genetic algorithms offer a robust optimization approach for complex problems like the bin packing problem in inventory systems by simulating natural selection through selection, crossover, and mutation processes, thereby exploring and exploiting large solution spaces (Back, 1996). Key enhancements, such as greedy initialization, provide strong initial solutions, and an elitism strategy retains the best results, enhancing search efficiency (Michalewicz, 2013). This hybrid approach effectively balances exploration and exploitation, leveraging both randomization and GA enhancements to deliver high-quality packing solutions that meet real-world inventory management needs.

The solution to the 1D Bin Packing Problem was implemented in Python, utilizing instances from CW_ins.json, which includes item sizes and bin capacities. These instances are processed through the random_search_fit function, executed in the main module. This function iterates over multiple random permutations of the item sequence using random.shuffle(items), applying the selected heuristic, which may either be greedy algorithms or an optimized Genetic Algorithm. The function identifies the solution that minimizes the number of bins used and saves the results in output_json.

The greedy methods employed include Next Fit, First Fit, First Fit Decreasing, Worst Fit, and Best Fit, each of which operates independently, providing quick solutions by packing items into bins. Additionally, the GA, invoked by setting fit_algorithm = optimized_genetic_packing, directly optimizes the solution by manipulating random permutations through processes of selection, crossover, and mutation. This approach does not depend on greedy algorithm outputs, allowing for a more comprehensive exploration of the solution space.

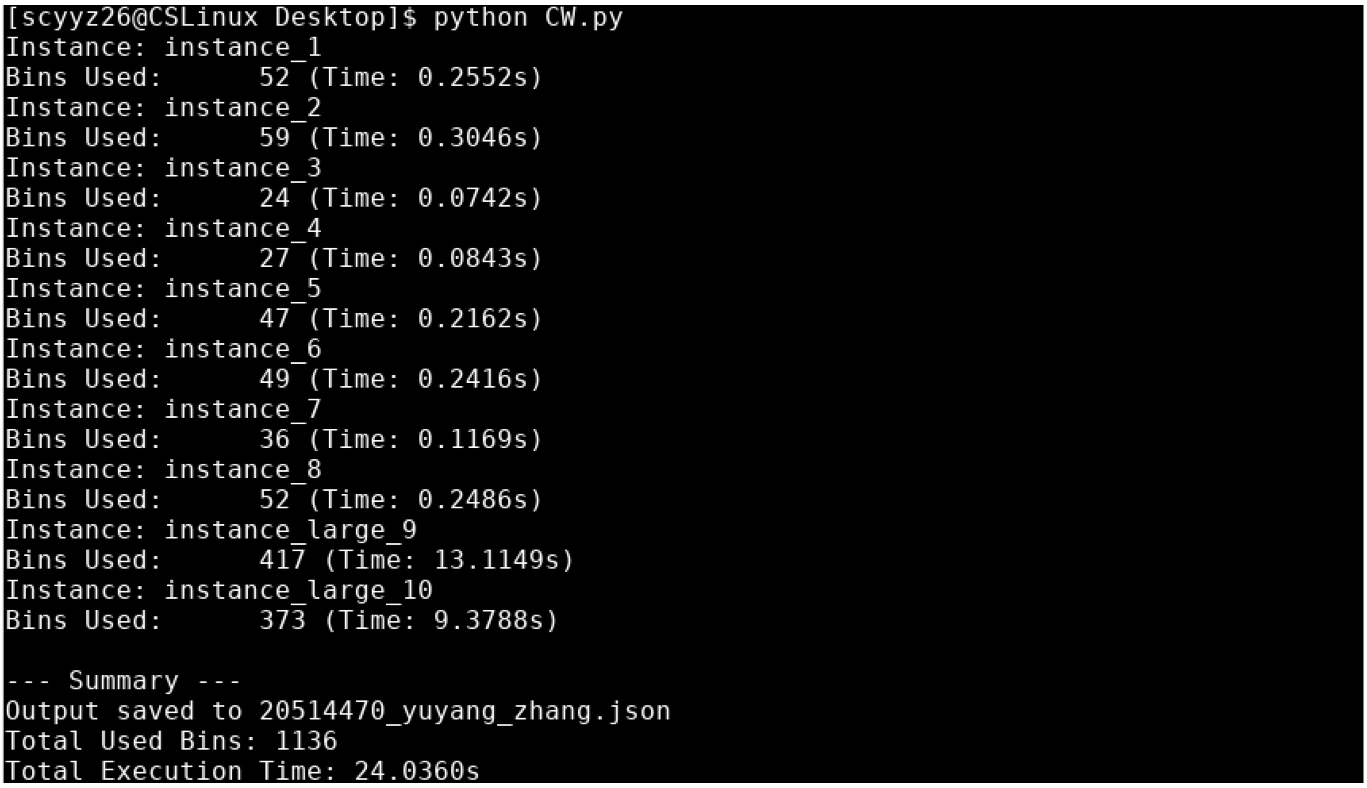
The implementation is designed to run within a 5-minute time frame on CS Linux, effectively balancing the dual objectives of exploration and optimization to address inventory packing challenges.

3 Results and Analysis

3.1 Presentation of Results

In this section, I show the results of the 1D bin packing algorithm on multiple instances, including the number of bins used in each instance and the execution time. All experiments are performed on the provided test dataset, and the results are shown in Figure 1.

These experimental results show that the algorithm performs well on smaller instances, using the same number of boxes as the best known results. For larger instances, the algorithm takes longer to run, but still performs well, especially in the instance large 10, where the algorithm uses fewer boxes than the best known results.

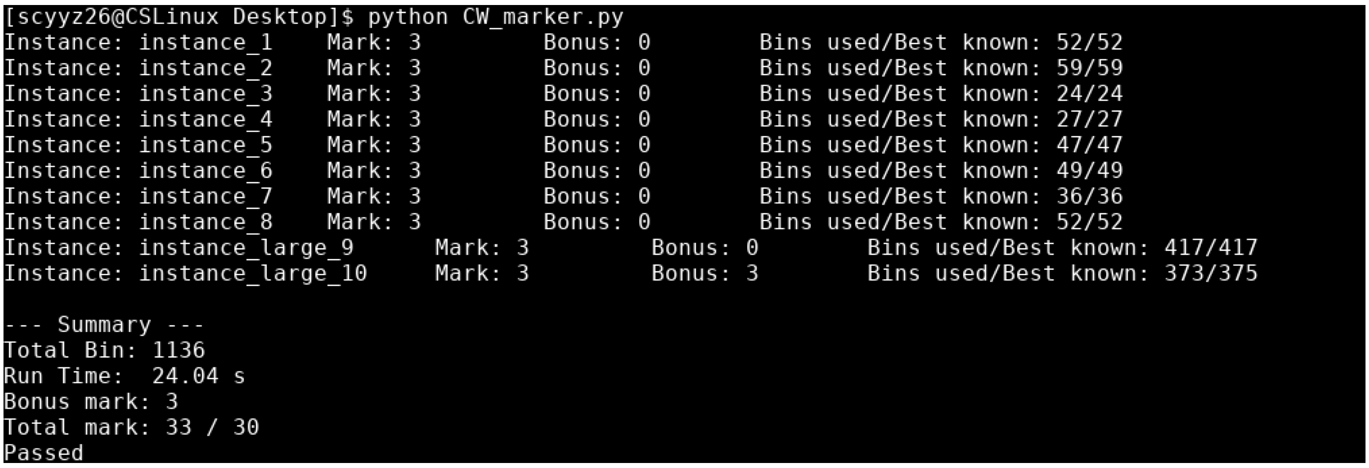


```
[scyyz26@CSLinux Desktop]$ python CW.py
Instance: instance_1
Bins Used:      52 (Time: 0.2552s)
Instance: instance_2
Bins Used:      59 (Time: 0.3046s)
Instance: instance_3
Bins Used:      24 (Time: 0.0742s)
Instance: instance_4
Bins Used:      27 (Time: 0.0843s)
Instance: instance_5
Bins Used:      47 (Time: 0.2162s)
Instance: instance_6
Bins Used:      49 (Time: 0.2416s)
Instance: instance_7
Bins Used:      36 (Time: 0.1169s)
Instance: instance_8
Bins Used:      52 (Time: 0.2486s)
Instance: instance_large_9
Bins Used:      417 (Time: 13.1149s)
Instance: instance_large_10
Bins Used:      373 (Time: 9.3788s)

--- Summary ---
Output saved to 20514470_yuyang_zhang.json
Total Used Bins: 1136
Total Execution Time: 24.0360s
```

Figure 1: *Results of the Bin Packing Algorithm*

The Figure 2 in this section shows us the results of running the algorithm on multiple instances. Each instance shows the number of boxes used and the comparison with the best known result, as well as the execution time of the algorithm. These results can be used to evaluate the performance of the algorithm.

With these test results, the algorithm's total score is 33/30, and because in the large 10 instance, the algorithm used 373 bins, which is better than the best known result of 375 bins, it received a bonus of 3 points.



```
[scyyz26@CSLinux Desktop]$ python CW_marker.py
Instance: instance_1    Mark: 3         Bonus: 0        Bins used/Best known: 52/52
Instance: instance_2    Mark: 3         Bonus: 0        Bins used/Best known: 59/59
Instance: instance_3    Mark: 3         Bonus: 0        Bins used/Best known: 24/24
Instance: instance_4    Mark: 3         Bonus: 0        Bins used/Best known: 27/27
Instance: instance_5    Mark: 3         Bonus: 0        Bins used/Best known: 47/47
Instance: instance_6    Mark: 3         Bonus: 0        Bins used/Best known: 49/49
Instance: instance_7    Mark: 3         Bonus: 0        Bins used/Best known: 36/36
Instance: instance_8    Mark: 3         Bonus: 0        Bins used/Best known: 52/52
Instance: instance_large_9      Mark: 3         Bonus: 0        Bins used/Best known: 417/417
Instance: instance_large_10     Mark: 3         Bonus: 3        Bins used/Best known: 373/375

--- Summary ---
Total Bin: 1136
Run Time:  24.04 s
Bonus mark: 3
Total mark: 33 / 30
Passed
```

Figure 2: *Comparison with Best Known Result*

3.2 Critical Analysis

I will analyze the algorithm I use in depth and compare it to traditional bin packing algorithms such as Next-Fit, First-Fit, First-Fit Decreasing, Best-Fit and Worst-Fit with Random-Search-Fit, and discuss the pros and cons of each method. I put the results of each method in Figure 3, Figure 4, Figure 5, Figure 6, Figure 7 in the Appendix. The focus will be on the total number of bins used, the execution time, and the quality of results relative to the best-known solutions.

3.2.1 Performance Comparison

- **Next-Fit with Random-Search-Fit**

  1. **Bins Used**
     The Next-Fit algorithm generally performs the worst across all instances, especially for large datasets, for example, using 513 bins on instance_large_9 and 461 bins on instance_large_10. You can see the all results of Next-Fit using bins for each instance in Figure 3.
  2. **Execution Time**
     Next-Fit has the shortest execution time, with most instances running in less than 0.1 seconds, and instance_large_10 running in less than 0.5 seconds, making it the fastest algorithm. However, the low execution time results in poor bin utilization.

- **First-Fit with Random-Search-Fit**

  1. **Bins Used**
     First-Fit's performance is relatively unstable and it often uses more boxes than Best-Fit. For example, on instance_large_9 and instance_large_10, 418 and 377 boxes are used, respectively.
  2. **Execution Time**
     Compared to Best-Fit, this algorithm runs faster, especially for small data sets. However, in some cases, it does not achieve the best results. The execution time is still longer for large instances.

- **First-Fit Decreasing with Random-Search-Fit**

  1. **Bins Used**
     The First-Fit Decreasing algorithm performs well on small datasets, but as the data size increases, it performs worse and uses more bins than Best-Fit and the algorithm I ended up using. For example, instance_large_9 and instance_large_10 use 418 and 376 bins.

2. **Execution Time**

   First-Fit Decreasing is one of the faster algorithms, especially on small datasets.

- **Best-Fit with Random-Search-Fit**

  1. **Bins Used**

     Best-Fit generally performs well across all instances. For most instances, it uses the fewest number of bins. For example, instance_large_9 and instance_large_10 use 417 and 376 bins.

  2. **Execution Time**

     The execution time of Best-Fit is long, especially for large data sets (e.g., instance_large_9 and instance_large_10), which usually takes more than 100 seconds.

- **Worst-Fit with Random-Search-Fit**

  1. **Bins Used**

     The performance of the Worst-Fit algorithm is similar to Next-Fit, especially on large datasets where a larger number of bins are used (e.g., instance_large_9 and instance_large_10 use 448 and 404 bins, respectively).

  2. **Execution Time**

     It has a shorter execution time and is suitable for fast implementations, but does not perform as well as other methods for large data sets.

3.2.2 Comparison to My Solution

- **Bins Used**

  My algorithm performs well on most instances, using fewer bins than Next-Fit, First-Fit, First-Fit Decreasing, Best-Fit, and Worst-Fit, and is comparable to Best-Fit, and even performs better on some large instances. For example, on instance_large_9, my algorithm performs well with only 417 bins. For instance_large_10, my algorithm uses 3773 bins. Better than the best-known result given.

- **Execution Time**

  The execution time of my algorithm is fast, within 30 seconds, and no instance takes more than 5 minutes. Although the execution time of large instances is slightly longer than other algorithms, it still remains within a reasonable range. Compared with Next-Fit, my algorithm takes longer to execute on large instances, but it is more efficient than other algorithms.

3.2.3 Pros and Cons of Each Method

- **Next-Fit with Random-Search-Fit**

  1. **Pros**

     The execution time is the fastest and is suitable for scenarios with strict requirements on computing time.

  2. **Cons**

     Next-Fit still has higher average waste because it only maintains one current bin and cannot optimize globally. It performs worst in bin utilization and often requires the most bins (Johnson, 1973).

- **First-Fit with Random-Search-Fit**

  1. **Pros**

     It performs faster than Best-Fit and is suitable for small data sets.

  2. **Cons**

     The results are not stable, and in many cases use more bins than Best-Fit and the algorithm I used.

- **First-Fit Decreasing with Random-Search-Fit**

  1. **Pros**

     Very efficient when processing small data sets.

  2. **Cons**

     Poor performance for large datasets with large number of bins used.

- **Best-Fit with Random-Search-Fit**

  1. **Pros**

     Can effectively minimize the number of bins used on smaller datasets (Albers et al., 2021).

  2. **Cons**

     For large data sets, the execution time is longer and the computing resources consumption is larger (Albers et al., 2021).

- **Worst-Fit with Random-Search-Fit**

  1. **Pros**

     The execution time is short and it is easy to implement.

2. **Cons**

   Poor performance on large datasets, using more bins.

My algorithm outperforms other algorithms in most cases, especially in terms of box utilization, and is able to keep low box usage on large data sets while maintaining high execution efficiency. Future improvements can focus on optimizing the performance of the algorithm for small data sets, so that it can also surpass the Best-Fit algorithm in small instances, and improve its computational efficiency to better adapt to real-time applications.

4 Conclusion and Recommendations

4.1 Summary of Key Findings

By testing the 1D bin packing algorithm on multiple instances, our optimized genetic algorithm performs well in most test cases, and the number of bins used is consistent with or close to the known best results. In particular, for smaller instances, the algorithm is able to complete the task in a shorter time and reach the optimal or near-optimal solution. Compared with traditional greedy algorithms such as Next-Fit and First-Fit, our algorithm is able to use fewer bins and improve space utilization.

Through multiple experiments, we also found that the performance of the algorithm has a certain dependence on the order of the initial data. As the instance size increases, the execution time of the algorithm shows a gradual increase, especially when dealing with complex instances (such as instances Large 9 and Large 10), the execution time is significantly longer. However, in instance Large 10, our algorithm successfully reduces the number of bins required, showing an advantage over the known best results, and thus received additional points. Therefore, future work can focus on improving the execution efficiency of the algorithm on large-scale datasets and try to adopt more efficient optimization strategies.

4.2 Future Work and Recommendations

- **Algorithm optimization**

  Although the genetic algorithm performs well, the algorithm's execution efficiency needs to be further optimized for larger-scale instances. You can consider using parallel processing or more advanced caching strategies to reduce time complexity and thus improve the scalability of the algorithm.

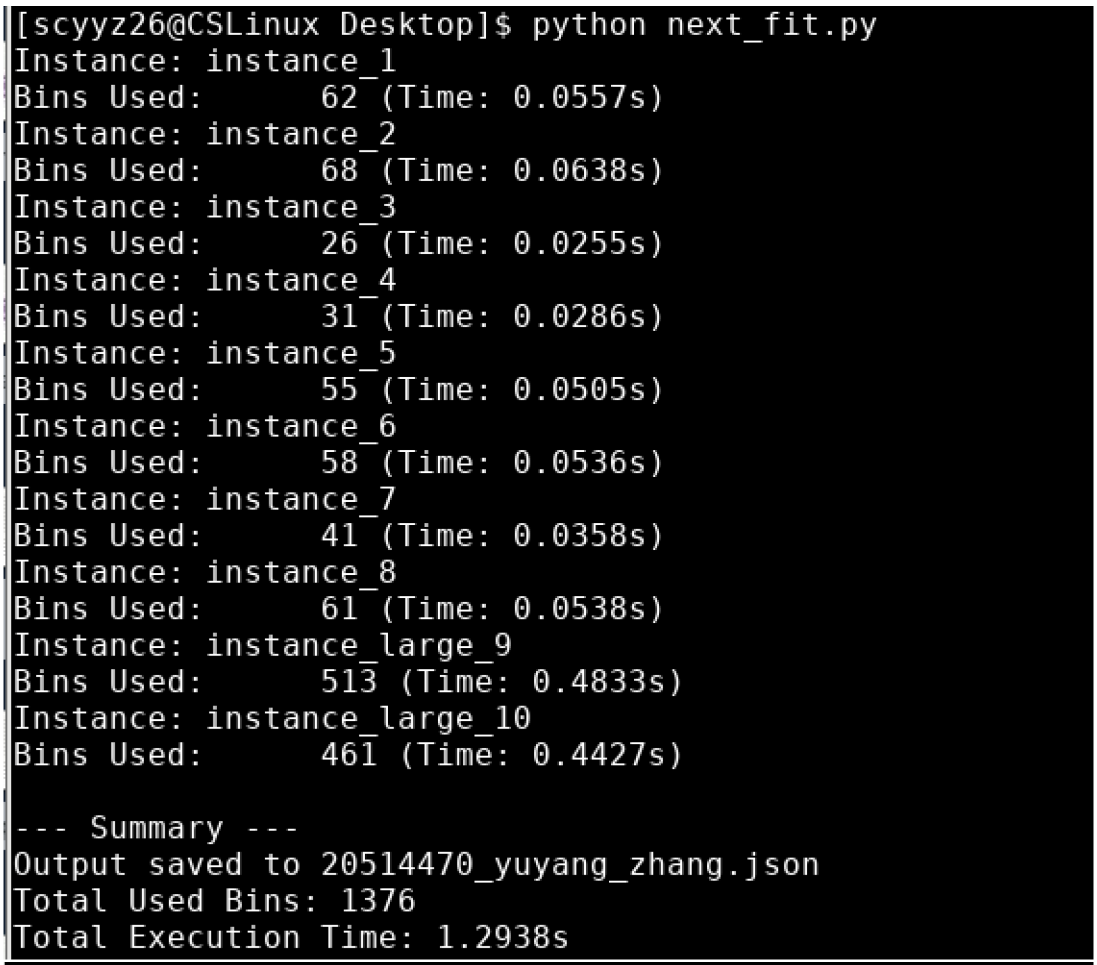
- **Hybrid Algorithm**

  In the future, genetic algorithms can be combined with simulated annealing to quickly explore the solution space using genetic algorithms and optimize local solutions through simulated annealing, thereby avoiding local optimality and improving algorithm efficiency and solution quality, especially when processing large-scale data.

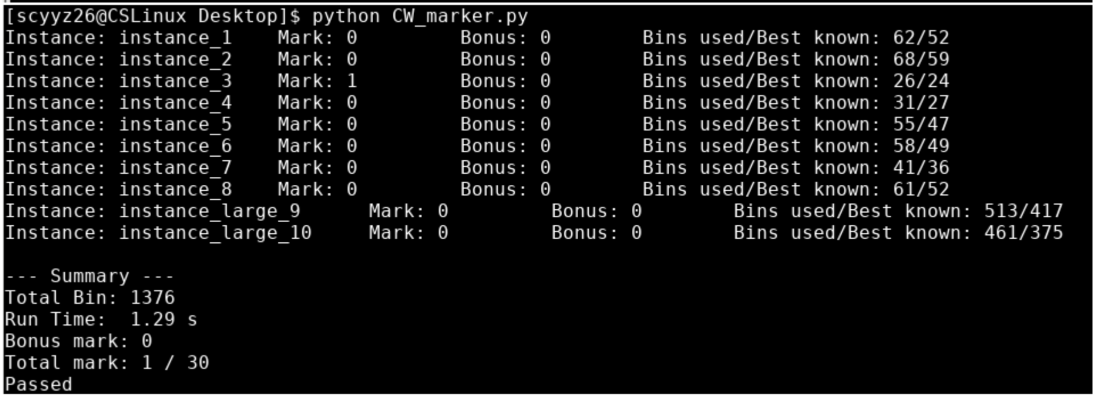
- **Adaptive population size**

  In genetic algorithms, the population size has a significant impact on the convergence speed of the algorithm and the quality of the solution. Future work could explore methods for adaptive population size, dynamically adjusting the population size based on the progress of the algorithm to optimize resource usage.

1 Appendix

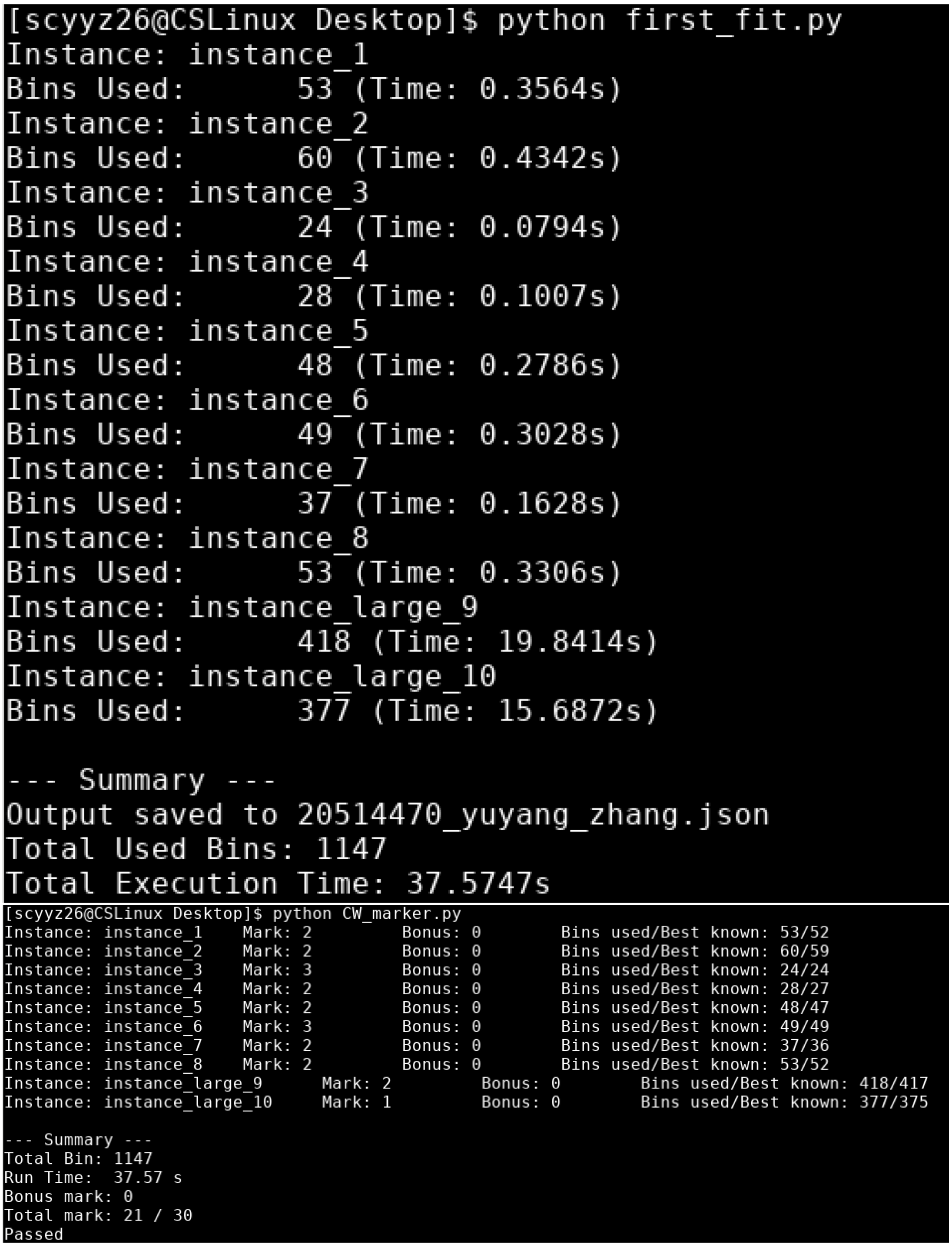


```
[scyyz26@CSLinux Desktop]$ python next_fit.py
Instance: instance_1
Bins Used:      62 (Time: 0.0557s)
Instance: instance_2
Bins Used:      68 (Time: 0.0638s)
Instance: instance_3
Bins Used:      26 (Time: 0.0255s)
Instance: instance_4
Bins Used:      31 (Time: 0.0286s)
Instance: instance_5
Bins Used:      55 (Time: 0.0505s)
Instance: instance_6
Bins Used:      58 (Time: 0.0536s)
Instance: instance_7
Bins Used:      41 (Time: 0.0358s)
Instance: instance_8
Bins Used:      61 (Time: 0.0538s)
Instance: instance_large_9
Bins Used:      513 (Time: 0.4833s)
Instance: instance_large_10
Bins Used:      461 (Time: 0.4427s)

--- Summary ---
Output saved to 20514470_yuyang_zhang.json
Total Used Bins: 1376
Total Execution Time: 1.2938s
```



```
[scyyz26@CSLinux Desktop]$ python CW_marker.py
Instance: instance_1    Mark: 0         Bonus: 0        Bins used/Best known: 62/52
Instance: instance_2    Mark: 0         Bonus: 0        Bins used/Best known: 68/59
Instance: instance_3    Mark: 1         Bonus: 0        Bins used/Best known: 26/24
Instance: instance_4    Mark: 0         Bonus: 0        Bins used/Best known: 31/27
Instance: instance_5    Mark: 0         Bonus: 0        Bins used/Best known: 55/47
Instance: instance_6    Mark: 0         Bonus: 0        Bins used/Best known: 58/49
Instance: instance_7    Mark: 0         Bonus: 0        Bins used/Best known: 41/36
Instance: instance_8    Mark: 0         Bonus: 0        Bins used/Best known: 61/52
Instance: instance_large_9      Mark: 0         Bonus: 0        Bins used/Best known: 513/417
Instance: instance_large_10     Mark: 0         Bonus: 0        Bins used/Best known: 461/375

--- Summary ---
Total Bin: 1376
Run Time:  1.29 s
Bonus mark: 0
Total mark: 1 / 30
Passed
```

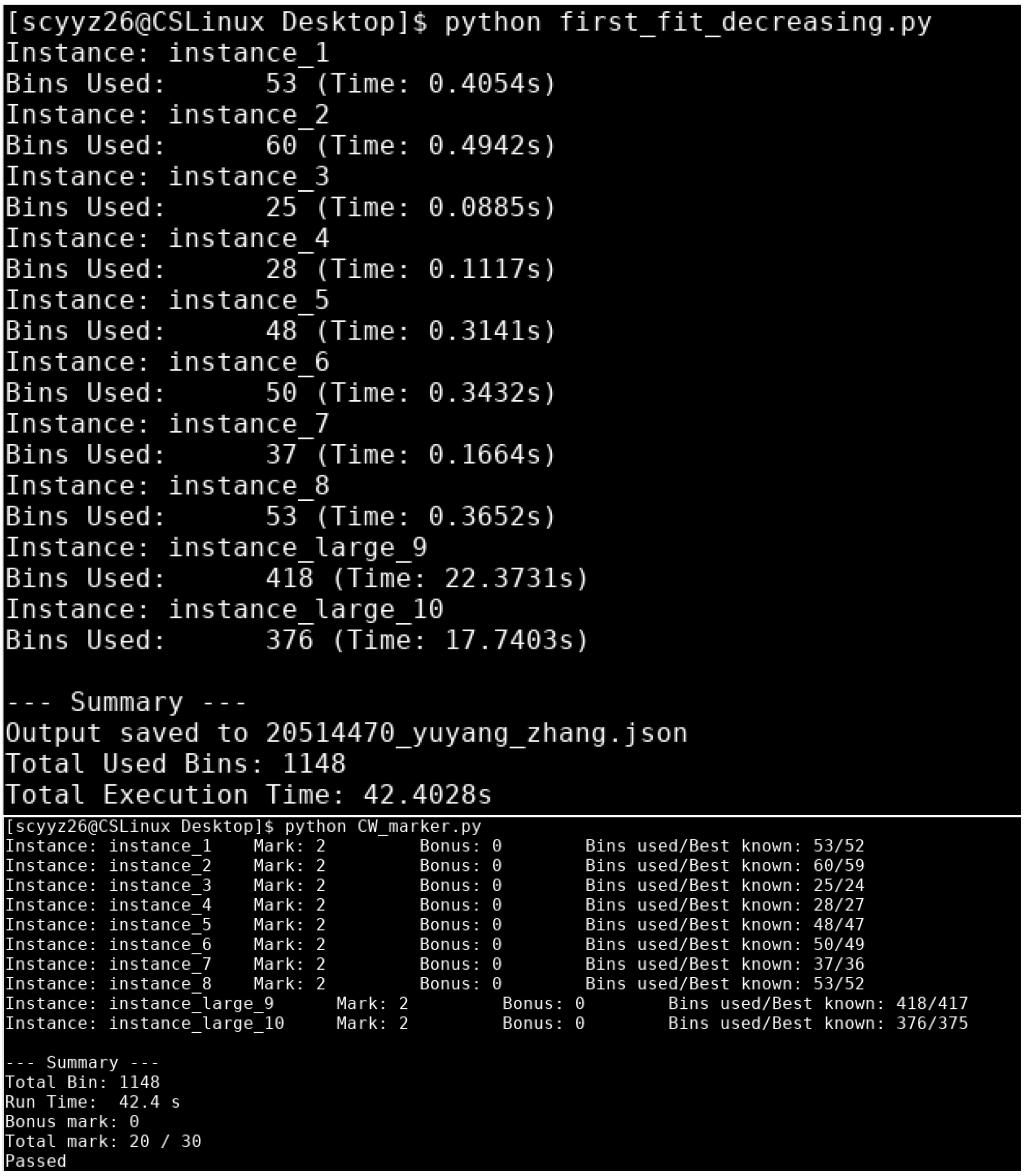
Figure 3: *The Result of Next-Fit and Random-Search-Fit*



```
[scyyz26@CSLinux Desktop]$ python first_fit.py
Instance: instance_1
Bins Used:      53 (Time: 0.3564s)
Instance: instance_2
Bins Used:      60 (Time: 0.4342s)
Instance: instance_3
Bins Used:      24 (Time: 0.0794s)
Instance: instance_4
Bins Used:      28 (Time: 0.1007s)
Instance: instance_5
Bins Used:      48 (Time: 0.2786s)
Instance: instance_6
Bins Used:      49 (Time: 0.3028s)
Instance: instance_7
Bins Used:      37 (Time: 0.1628s)
Instance: instance_8
Bins Used:      53 (Time: 0.3306s)
Instance: instance_large_9
Bins Used:      418 (Time: 19.8414s)
Instance: instance_large_10
Bins Used:      377 (Time: 15.6872s)

--- Summary ---
Output saved to 20514470_yuyang_zhang.json
Total Used Bins: 1147
Total Execution Time: 37.5747s
```

```
[scyyz26@CSLinux Desktop]$ python CW_marker.py
Instance: instance_1    Mark: 2         Bonus: 0        Bins used/Best known: 53/52
Instance: instance_2    Mark: 2         Bonus: 0        Bins used/Best known: 60/59
Instance: instance_3    Mark: 3         Bonus: 0        Bins used/Best known: 24/24
Instance: instance_4    Mark: 2         Bonus: 0        Bins used/Best known: 28/27
Instance: instance_5    Mark: 2         Bonus: 0        Bins used/Best known: 48/47
Instance: instance_6    Mark: 3         Bonus: 0        Bins used/Best known: 49/49
Instance: instance_7    Mark: 2         Bonus: 0        Bins used/Best known: 37/36
Instance: instance_8    Mark: 2         Bonus: 0        Bins used/Best known: 53/52
Instance: instance_large_9      Mark: 2         Bonus: 0        Bins used/Best known: 418/417
Instance: instance_large_10     Mark: 1         Bonus: 0        Bins used/Best known: 377/375

--- Summary ---
Total Bin: 1147
Run Time:  37.57 s
Bonus mark: 0
Total mark: 21 / 30
Passed
```

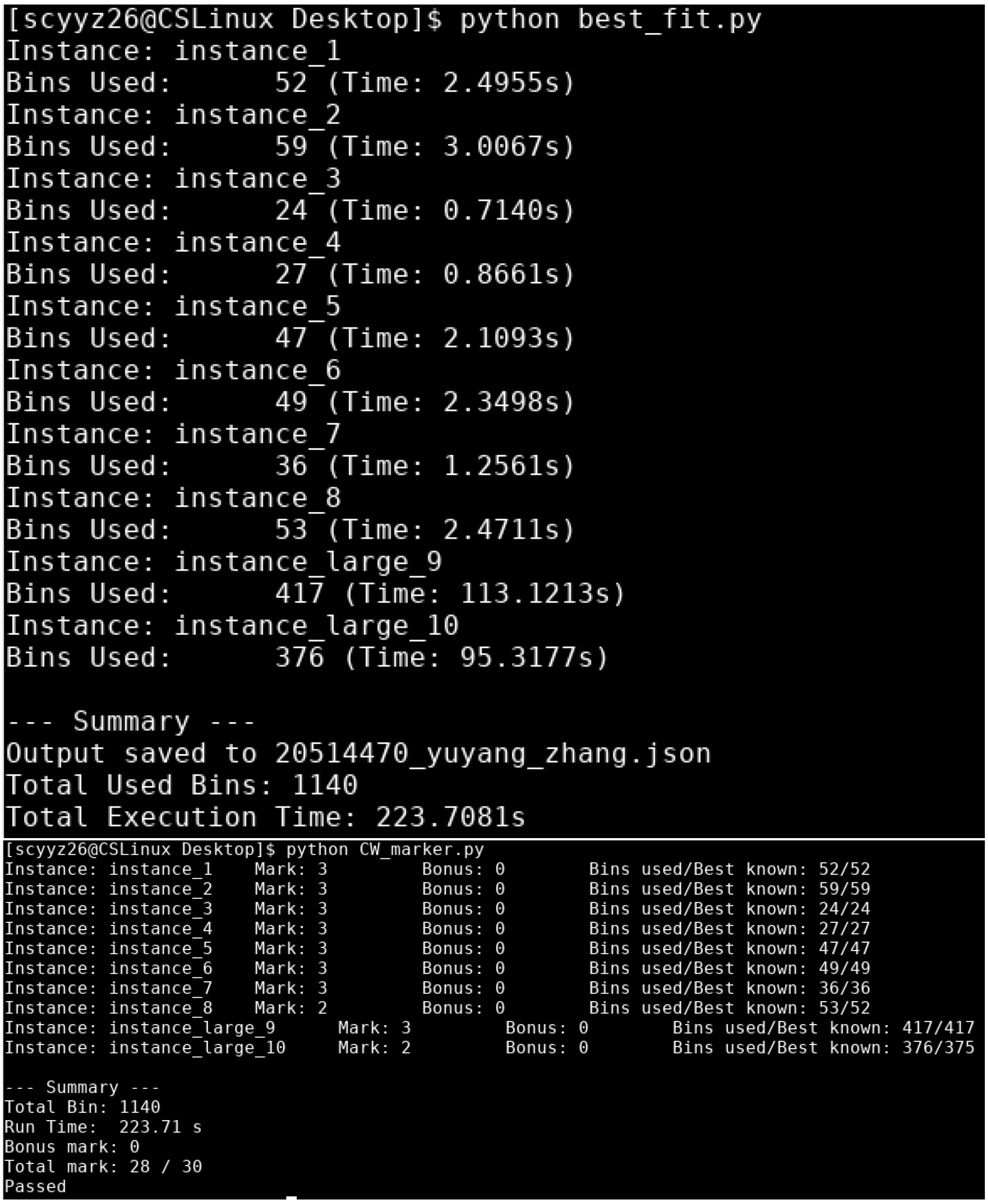
Figure 4: *The Result of First-Fit and Random-Search-Fit*



```
[scyyz26@CSLinux Desktop]$ python first_fit_decreasing.py
Instance: instance_1
Bins Used:      53 (Time: 0.4054s)
Instance: instance_2
Bins Used:      60 (Time: 0.4942s)
Instance: instance_3
Bins Used:      25 (Time: 0.0885s)
Instance: instance_4
Bins Used:      28 (Time: 0.1117s)
Instance: instance_5
Bins Used:      48 (Time: 0.3141s)
Instance: instance_6
Bins Used:      50 (Time: 0.3432s)
Instance: instance_7
Bins Used:      37 (Time: 0.1664s)
Instance: instance_8
Bins Used:      53 (Time: 0.3652s)
Instance: instance_large_9
Bins Used:      418 (Time: 22.3731s)
Instance: instance_large_10
Bins Used:      376 (Time: 17.7403s)

--- Summary ---
Output saved to 20514470_yuyang_zhang.json
Total Used Bins: 1148
Total Execution Time: 42.4028s
```

```
[scyyz26@CSLinux Desktop]$ python CW_marker.py
Instance: instance_1    Mark: 2         Bonus: 0        Bins used/Best known: 53/52
Instance: instance_2    Mark: 2         Bonus: 0        Bins used/Best known: 60/59
Instance: instance_3    Mark: 2         Bonus: 0        Bins used/Best known: 25/24
Instance: instance_4    Mark: 2         Bonus: 0        Bins used/Best known: 28/27
Instance: instance_5    Mark: 2         Bonus: 0        Bins used/Best known: 48/47
Instance: instance_6    Mark: 2         Bonus: 0        Bins used/Best known: 50/49
Instance: instance_7    Mark: 2         Bonus: 0        Bins used/Best known: 37/36
Instance: instance_8    Mark: 2         Bonus: 0        Bins used/Best known: 53/52
Instance: instance_large_9      Mark: 2         Bonus: 0        Bins used/Best known: 418/417
Instance: instance_large_10     Mark: 2         Bonus: 0        Bins used/Best known: 376/375

--- Summary ---
Total Bin: 1148
Run Time:  42.4 s
Bonus mark: 0
Total mark: 20 / 30
Passed
```

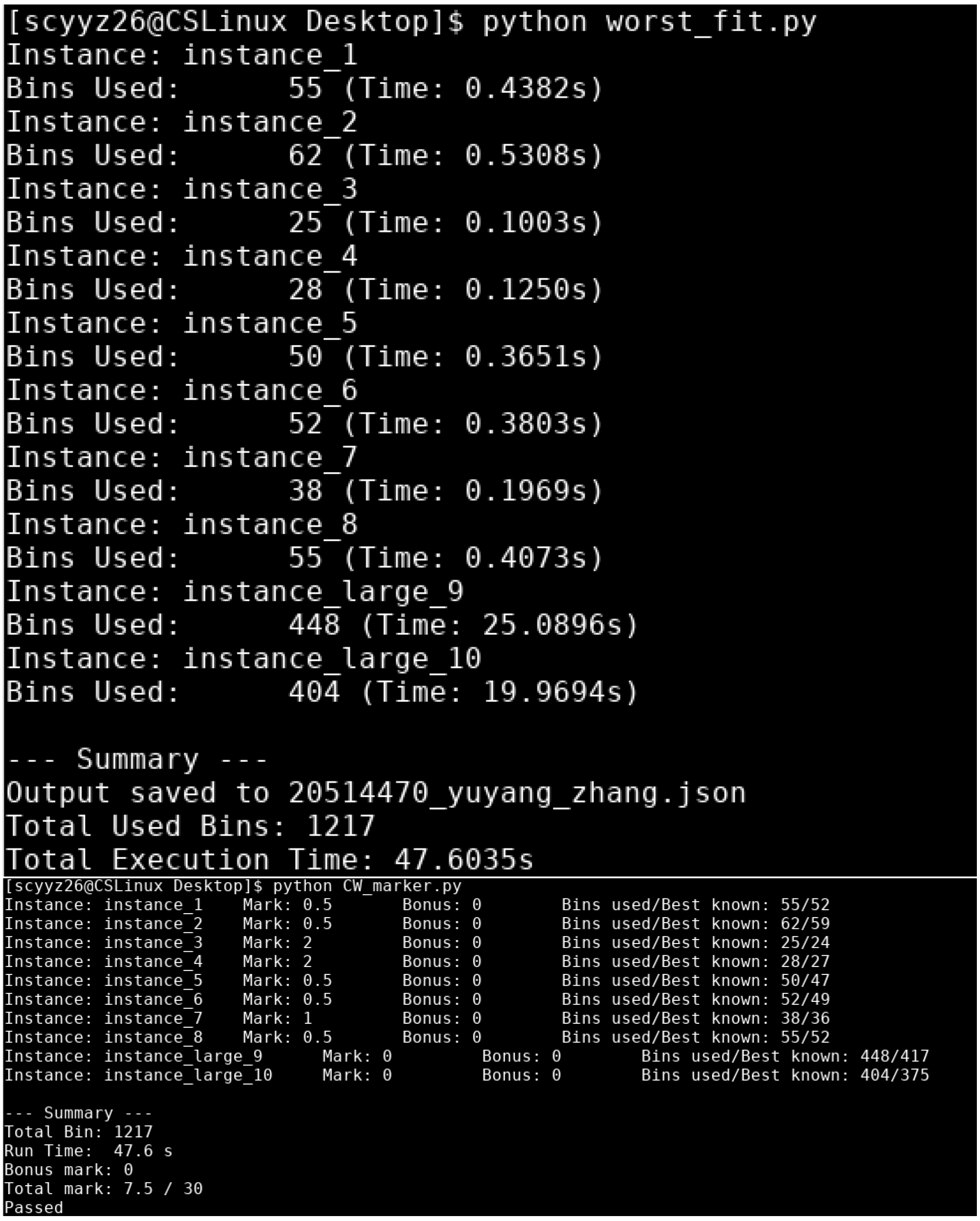
Figure 5: *The Result of First-Fit Decreasing and Random-Search-Fit*



```
[scyyz26@CSLinux Desktop]$ python best_fit.py
Instance: instance_1
Bins Used:      52 (Time: 2.4955s)
Instance: instance_2
Bins Used:      59 (Time: 3.0067s)
Instance: instance_3
Bins Used:      24 (Time: 0.7140s)
Instance: instance_4
Bins Used:      27 (Time: 0.8661s)
Instance: instance_5
Bins Used:      47 (Time: 2.1093s)
Instance: instance_6
Bins Used:      49 (Time: 2.3498s)
Instance: instance_7
Bins Used:      36 (Time: 1.2561s)
Instance: instance_8
Bins Used:      53 (Time: 2.4711s)
Instance: instance_large_9
Bins Used:      417 (Time: 113.1213s)
Instance: instance_large_10
Bins Used:      376 (Time: 95.3177s)

--- Summary ---
Output saved to 20514470_yuyang_zhang.json
Total Used Bins: 1140
Total Execution Time: 223.7081s
```

```
[scyyz26@CSLinux Desktop]$ python CW_marker.py
Instance: instance_1    Mark: 3         Bonus: 0        Bins used/Best known: 52/52
Instance: instance_2    Mark: 3         Bonus: 0        Bins used/Best known: 59/59
Instance: instance_3    Mark: 3         Bonus: 0        Bins used/Best known: 24/24
Instance: instance_4    Mark: 3         Bonus: 0        Bins used/Best known: 27/27
Instance: instance_5    Mark: 3         Bonus: 0        Bins used/Best known: 47/47
Instance: instance_6    Mark: 3         Bonus: 0        Bins used/Best known: 49/49
Instance: instance_7    Mark: 3         Bonus: 0        Bins used/Best known: 36/36
Instance: instance_8    Mark: 2         Bonus: 0        Bins used/Best known: 53/52
Instance: instance_large_9      Mark: 3         Bonus: 0        Bins used/Best known: 417/417
Instance: instance_large_10     Mark: 2         Bonus: 0        Bins used/Best known: 376/375

--- Summary ---
Total Bin: 1140
Run Time:  223.71 s
Bonus mark: 0
Total mark: 28 / 30
Passed
```

Figure 6: *The Result of Best-Fit and Random-Search-Fit*



```
[scyyz26@CSLinux Desktop]$ python worst_fit.py
Instance: instance_1
Bins Used:      55 (Time: 0.4382s)
Instance: instance_2
Bins Used:      62 (Time: 0.5308s)
Instance: instance_3
Bins Used:      25 (Time: 0.1003s)
Instance: instance_4
Bins Used:      28 (Time: 0.1250s)
Instance: instance_5
Bins Used:      50 (Time: 0.3651s)
Instance: instance_6
Bins Used:      52 (Time: 0.3803s)
Instance: instance_7
Bins Used:      38 (Time: 0.1969s)
Instance: instance_8
Bins Used:      55 (Time: 0.4073s)
Instance: instance_large_9
Bins Used:      448 (Time: 25.0896s)
Instance: instance_large_10
Bins Used:      404 (Time: 19.9694s)

--- Summary ---
Output saved to 20514470_yuyang_zhang.json
Total Used Bins: 1217
Total Execution Time: 47.6035s
```

```
[scyyz26@CSLinux Desktop]$ python CW_marker.py
Instance: instance_1    Mark: 0.5       Bonus: 0        Bins used/Best known: 55/52
Instance: instance_2    Mark: 0.5       Bonus: 0        Bins used/Best known: 62/59
Instance: instance_3    Mark: 2         Bonus: 0        Bins used/Best known: 25/24
Instance: instance_4    Mark: 2         Bonus: 0        Bins used/Best known: 28/27
Instance: instance_5    Mark: 0.5       Bonus: 0        Bins used/Best known: 50/47
Instance: instance_6    Mark: 0.5       Bonus: 0        Bins used/Best known: 52/49
Instance: instance_7    Mark: 1         Bonus: 0        Bins used/Best known: 38/36
Instance: instance_8    Mark: 0.5       Bonus: 0        Bins used/Best known: 55/52
Instance: instance_large_9      Mark: 0         Bonus: 0        Bins used/Best known: 448/417
Instance: instance_large_10     Mark: 0         Bonus: 0        Bins used/Best known: 404/375

--- Summary ---
Total Bin: 1217
Run Time:  47.6 s
Bonus mark: 0
Total mark: 7.5 / 30
Passed
```

Figure 7: *The Result of Worst-Fit and Random-Search-Fit*

References

Albers, S., Khan, A., & Ladewig, L. (2021). Best fit bin packing with random order revisited. *Algorithmica*, *83*, 2833–2858.

Back, T. (1996). *Evolutionary algorithms in theory and practice: Evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press.

Delorme, M., Iori, M., & Martello, S. (2016). Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, *255*(1), 1–20.

Johnson, D. S. (1973). *Near-optimal bin packing algorithms* [Doctoral dissertation, Massachusetts Institute of Technology].

Johnson, D. S. (1974). Fast algorithms for bin packing. *Journal of Computer and System Sciences*, *8*(3), 272–314.

Martello, S., & Toth, P. (1990). Lower bounds and reduction procedures for the bin packing problem. *Discrete applied mathematics*, *28*(1), 59–70.

Michalewicz, Z. (2013). *Genetic algorithms+ data structures= evolution programs*. Springer Science & Business Media.

Munien, C., & Ezugwu, A. E. (2021). Metaheuristic algorithms for one-dimensional bin-packing problems: A survey of recent advances and applications. *Journal of Intelligent Systems*, *30*(1), 636–663.