COMP2051 Artificial Intelligence Methods Course Work Report

Information

-
-

COMP2051 Artificial Intelligence Methods Course Work Report

- Information
- Algorithm Structure
  - Solution Encoding
  - Initial Solution
  - Heuristics:
    1. Shift
    2. Split
    3. Exchange LargestBin_LargestItem
    4. Exchange RandomBin_Reshuffle
  - Variable Neighbourhoods
    1. RBR + LBLI + Shift
    2. Split + LBLI + Shift
    3. Hybrid Heuristics
  - Shaking
- Statistical Result
- Discussion

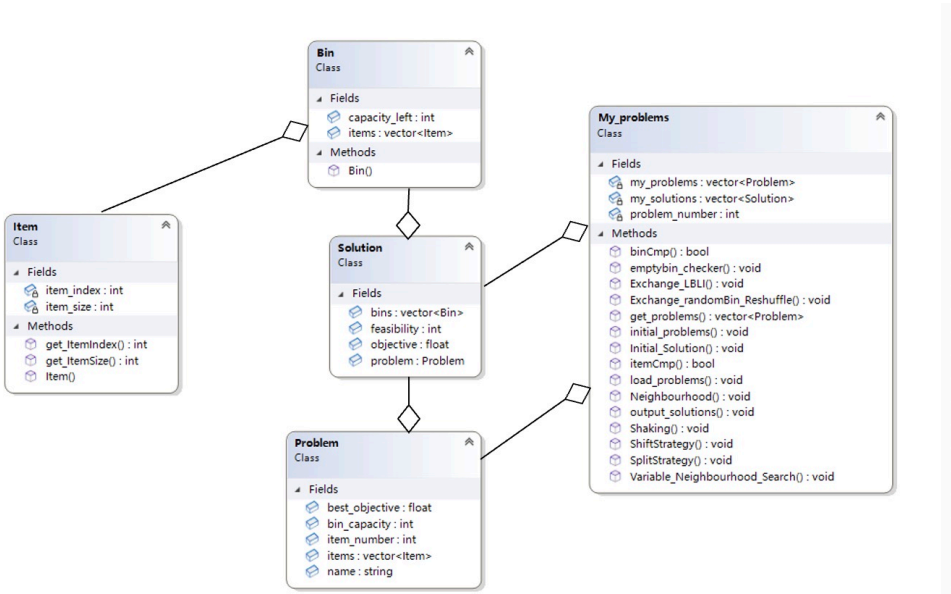
Algorithm Structure

Solution Encoding

There are mainly five classes in this program to load problems and generate solutions. They are "Item", "Bin", "Problem", "Solution" and "My_problems" classes.

- The "Item" class is used for storing the information of each item which is in the bin of a problem and solution.
- The "Bin" class is used for storing the information of each bin which will store items in the bin packing problem.
- The "Problem" class is used for storing the information of the loaded problems with items.
- The "Solution" class is used for storing the information of the solution of the corresponding problem with the problem object, objective and bins.
- The "My_problems" class is used for processing VNS with these problems and solutions including loading problems, outputting solutions, generating initial solutions, variable neighbourhood descents, heuristic methods and shaking.

The class diagram is shown as follows:



Initial Solution

The initial solution first uses an item vector to store all the items from the problem. This method then **sorts the items** from small size items to big size one. After the sort operation, this program uses the **first-fit** descent heuristic to load these items into bins and generate the initial solution.

Heuristics:

This program uses four heuristics to solve the bin packing problem.

1. Shift

This heuristic mainly selects each item from the bin with the largest left capacity or residual capacity and moves these items using the best-fit descent heuristic. This heuristic shows a convergency ability to reduce the number of bins. At the end of the heuristic, the empty bin checker detects empty bins and deletes them.

2. Split

This heuristic randomly selects a bin whose item number exceeds the average item numbers per bin. This heuristic shows the shuffle's ability to mix the items and other heuristics can make the items as an optimal way and maximum filling way to pack these items. This heuristic also takes advantage of the small size bins to fill in the residual capacity that the large size of items causes.

3. Exchange LargestBin_LargestItem

This heuristic selects the largest size item in the largest residual capacity bin and randomly selects another bin which is not full, to exchange the item in the largest residual capacity bin with the items whose sum is smaller than the largest residual capacity one and the residual capacity which are in the random bin. This heuristic improves the residual capacity of one bin and shows the previous steps to reduce the number of bins of the problem. Then this largest residual capacity bin can be emptied by other heuristics.

4. Exchange RandomBin_Reshuffle

This heuristic randomly selects two non-full-filled bins with the probability of the residual capacity of bins. All items from these two bins are considered and choose the best items combination, which means maximally filling one bin and the remaining items are filled in another bin. This heuristic shows the same idea as the LargestBin_LargestItem heuristic. This heuristic generates a full used bin and a remaining items bin, which can be emptied by other heuristics.

Implementation of the best item's combination:
This implementation first gets items from these two bins, **reshuffle** them and uses **first-fit** heuristic decent to pack these items. This process runs many times (200 in this case), to search for the best combination after filtering out invalid combinations, which means the combination is valid for these two bins and one of them is maximally filled in items.

Variable Neighbourhoods

There are three neighbourhood spaces in this program.

1. RBR + LBLI + Shift

This neighbourhood movement is combined with RandomBin_Reshuffle, LargestBin_LargestItem and Shift heuristics. The first two heuristics are used to generate a bin with a larger residual capacity and finally Shift heuristic is used to empty this bin. This moves many times (200 in this case) to generate a neighbourhood space with solutions.

2. Split + LBLI + Shift

This neighbourhood movement is combined with Split, LargestBin_LargestItem and Shift heuristics. The first two heuristics are used to generate two bins with larger residual capacity and finally Shift heuristic is used to empty this bin. This moves many times (200 in this case) to generate a neighbourhood space with solutions.

3. Hybrid Heuristics

This neighbourhood movement is combined with all four heuristics which are mentioned before. Split, RandomBin_Reshuffle, and LargestBin_largestITem are used to fully generate the maximally filled bins and bins with larger residual capacity and the Shift heuristic finally empties bins which are not full. This moves many times (200 in this case) to generate a neighbourhood space with solutions.

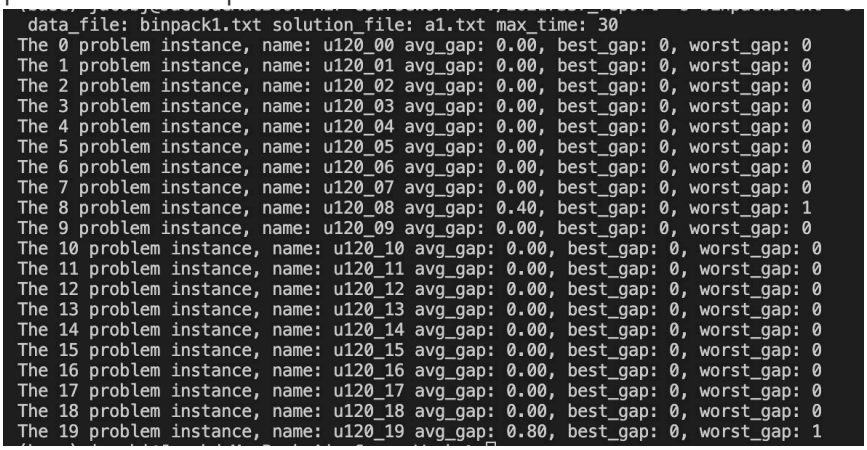
Shaking

The shaking contains three components.
First, this shaking movement **reshuffles all the bins which are not full**. In the reshuffling, items from these bins are reshuffled and packed in bins with the **first-fit** descent heuristic. **Second**, after reshuffling, this shaking movement randomly selects two bins and packs the items in these bins randomly. **Third**, this shaking movement randomly selects two bins and **exchanges** the largest size item of each bin.

Statistical Result

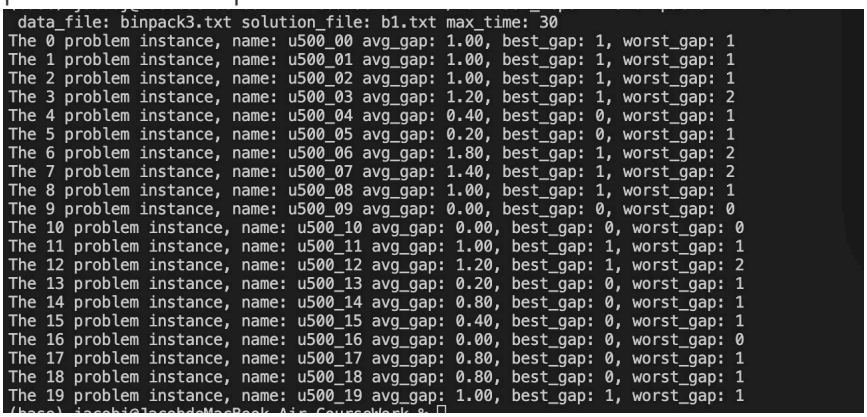
The program is tested five times for each problem. These are the result:

1. problem file: "binpack1.txt"



Problem Name	Avg Gap	Best Gap	Worst Gap		Problem Name	Avg Gap	Best Gap	Worst Gap
u120_00	0.00	0	0		u120_10	0.00	0	0
u120_01	0.00	0	0		u120_11	0.00	0	0
u120_02	0.00	0	0		u120_12	0.00	0	0
u120_03	0.00	0	0		u120_13	0.00	0	0
u120_04	0.00	0	0		u120_14	0.00	0	0
u120_05	0.00	0	0		u120_15	0.00	0	0
u120_06	0.00	0	0		u120_16	0.00	0	0
u120_07	0.00	0	0		u120_17	0.00	0	0
u120_08	0.40	0	1		u120_18	0.00	0	0
u120_09	0.00	0	0		u120_19	0.80	0	1

2. problem file : "binpack3.txt"



Problem Name	Avg Gap	Best Gap	Worst Gap		Problem Name	Avg Gap	Best Gap	Worst Gap
u500_00	1.00	1	1		u500_10	0.00	0	0
u500_01	1.00	1	1		u500_11	1.00	1	1
u500_02	1.00	1	1		u500_12	1.20	1	2
u500_03	1.20	1	2		u500_13	0.20	0	1
u500_04	0.40	0	1		u500_14	0.80	0	1
u500_05	0.20	0	1		u500_15	0.40	0	1
u500_06	1.80	1	2		u500_16	0.00	0	0
u500_07	1.40	1	2		u500_17	0.80	0	1
u500_08	1.00	1	1		u500_18	0.80	0	1
u500_09	0.00	0	0		u500_19	1.00	1	1

3. problem file : "binipack11.txt"

```
 data_file: binpack11.txt solution_file: c1.txt max_time: 30
The 0 problem instance, name: HARD0 avg_gap: 0.00, best_gap: 0, worst_gap: 0
The 1 problem instance, name: HARD1 avg_gap: 0.00, best_gap: 0, worst_gap: 0
The 2 problem instance, name: HARD2 avg_gap: 1.00, best_gap: 1, worst_gap: 1
The 3 problem instance, name: HARD3 avg_gap: 1.00, best_gap: 1, worst_gap: 1
The 4 problem instance, name: HARD4 avg_gap: 0.20, best_gap: 0, worst_gap: 1
The 5 problem instance, name: HARD5 avg_gap: 0.40, best_gap: 0, worst_gap: 1
The 6 problem instance, name: HARD6 avg_gap: 0.00, best_gap: 0, worst_gap: 0
The 7 problem instance, name: HARD7 avg_gap: 0.00, best_gap: 0, worst_gap: 0
The 8 problem instance, name: HARD8 avg_gap: 0.00, best_gap: 0, worst_gap: 0
The 9 problem instance, name: HARD9 avg_gap: 0.20, best_gap: 0, worst_gap: 1
```

Problem Name	Avg Gap	Best Gap	Worst Gap
HARD0	0.00	0	0
HARD1	0.00	0	0
HARD2	1.00	1	1
HARD3	1.00	1	1
HARD4	0.20	0	1
HARD5	0.40	0	1
HARD6	0.00	0	0
HARD7	0.00	0	0
HARD8	0.00	0	0
HARD9	0.20	0	1

Discussion

1. Initial Solution
   When designing the initial solution. There is another implementation. In this program, the initial solution is generated by sorting items from small item size to large item size and first-fit descent heuristic. Another implementation will sort these items from large item size to small item size and use the first-fit descent heuristic. In my test, I found that the initial solution by sorting items from large item size to small item size will generate many bins with fewer items and residual capacity which can not be filled by any items. This also causes less operation by the four heuristics and there are still capacities that can not be removed. Then more heuristics can be created to solve this problem.
   Although this implementation will generate a much better solution for problems than the used implementation. The used implementation can generate a large number of bins and the bins can have a tendency to be filled up with the VNS.
2. Number of Items
   It is obvious and tested that the VNS with the combination of the four heuristics can solve the easy problems and hard problems better. This VNS will show the disadvantage of the middle-level problems which means the performance on the middle-level problems is not better than the easy level and hard level problems. There is one difference between them, which is the number of items of each problem. The number of items of the middle-level problems is over two times more than the number of items of easy level problems and hard level problems. Then this VNS does not work very well on the problems with a large number of items.