AIM coursework report

Ke Liu(20217275)

May 4, 2022

1 Abstract

This program is a C++ program that can be used to solve bin packing problem by using variable neighborhood search method with considerable results. In this report, the algorithm of the program will be introduced in **Section 2**, the statistical results in 5 times will be concluded and demonstrated in **Section 3** and the reflection about the program will be discussed in **Section 4**.

2 Algorithm

This section will describe the representation of candidate solution in **(2.1)**, the initialisation of the solution by using relaxed minimum bin slack algorithm in **(2.2)**, the intensification process including 4 neighborhoods in **(2.3)**, the diversification process(shaking procedure) in **(2.4)** and a fitness function to evaluate the solution in **(2.5)**.

2.1 Solution encoding

In brief, the candidate solutions are represented by this data structure: vector⟨Bin⟩. And the items could be packed into Bin class. The final solution is to print the index of each items of every Bin in the vector⟨Bin⟩.

2.1.1 Class Bin

The Bin class has three attributes: capacity used to record the capacity of each bin, cap_left used to record the residual capacity of each bin and a vector⟨Item⟩used to record the packed items of the bin. And all these details are shown in *Figure.1*.

2.1.2 Class Item

The Item class has two attributes: index used to record the index of each item as a label from the source file and the size of each item. And all these details are shown in *Figure.2*.

2.2 Initialisation

I have tried a lot of different approaches to get the initial solution like first fit and best fit. By comparison with these approaches, relaxed minimum bin slack approach'(relaxed mbs') could result in a more efficient solution. The basis of this is minimum bin slack (mbs) and then minimum bin slack'(mbs') and finally get relaxed mbs' by improving mbs'.

```
/**
 * This class is used to store the information of Bin like the residual capacity,
 * packed items and some functions that deals with the bins like encase.
 */
class Bin{
public:
    Bin(int);                          // Constructor of Bin.
    void set_cap_left(int);            // Calculate the residual capacity of a bin and set it.
    const int get_cap_left();          // Get the residual capacity of the bin.
    void encase(Item);                 // Put an item into a bin.
    vector<Item> packed_items;         // The packed items in the bin.
private:
    int capacity;
    int cap_left;
};
```

Figure 1: Bin class structure.

```
/**
 * This class is used to store the information of each item in each instance, including
 * the index number, the size and the functions to get all these information.
 */
class Item{
public:
    Item(int);                         // Constructor of class Item;
    void setIndex(int);                // Set the index of an item;
    const int getIndex();              // Get the index of an item;
    const int get_size();              // Get the size of an item;
private:
    int index;
    int size;
};
```

Figure 2: Item class structure.

2.2.1 mbs

MBS heuristic is bin-focused and it tries to seek several items that fits the bin capacity as full as possible. First decreasingly sort all the items and start from the biggest item. Then packing and each packing procedure is try all possible subsets of unpacked items that fit the bin capacity.The items that make the least residual capacity could be adopted. And if find a set of items exactly fill with the bin then end the search.

2.2.2 mbs'

MBS' is a improved version from mbs. The only difference between mbs' and mbs is choose an item and permanent fixed in a bin before packing, which could shorten search time by leaving the least space in the bin to fill when searching.

2.2.3 relaxed mbs'

Relaxed mbs' is improved from mbs'. It sets a allowable slack at first. The change is this heuristic would accept a solution if the residual capacity is fewer than or equal to the allowable slack rather than trying its best to find the smallest residual capacity solution in mbs and mbs'. And after my tests, I set the allowable slack for easy pattern, medium pattern, hard pattern 3, 5, 11000 to facilitate the initial results.

2.3 Intensification

There are four heuristics in total. To use the least number of bins, my strategy is to eliminate the bin with the maximum residual capacity and all my heuristics are corresponding with that. So the simplest way is to move the item from this bin into another bin **(2.5.1)**. Secondly, if there is no chances for move directly, then try to use the largest item in this bin to swap a smaller bin from another bin **(2.5.2)**. Then do the same procedure but use the smallest item with the heuristic before **(2.5.3)**. At last, randomly selecting and moving half items from a bin that exceeds the average items per bin into a new bin **(2.5.4)**.

2.3.1 1. Shift

This heuristic is moving the smallest item from the bin with maximum residual capacity into another bin by using best descent, shown in *Figure.3*. This is the most critical heuristic to reduce the use of bins. Every other heuristics are make their contribution to swap the items in the bin with maximum residual capacity with an smaller size item to facilitate to adopt heuristic 1 to reduce item in this bin and try to reduce the bin.

2.3.2 2. Exchange_Largest

This heuristic aims to exchange the largest item of the item in the largest residual capacity bin with a smaller item from another not full bin, shown in *Figure.4*. Because there is no free space for the items in the bin with the maximum residual capacity in other bins,this procedure is to make it best to swap the bigger items to smaller items in this bin so that there will be more possibility to shift straightly in heuristic 1.

2.3.3 3. Exchange_Smallest

Similarly to heuristic 2, this heuristic aims to exchange the Smallest item of the item in the largest residual capacity bin with a smaller item from another not full bin, shown in *Figure.5*. And the design of this heuristic is also swapping the bigger items to smaller items in the bin with maximum residual capacity. The special point is this search is from the smallest item because there could be a situation that bigger item is not suitable but smaller item is acceptable.

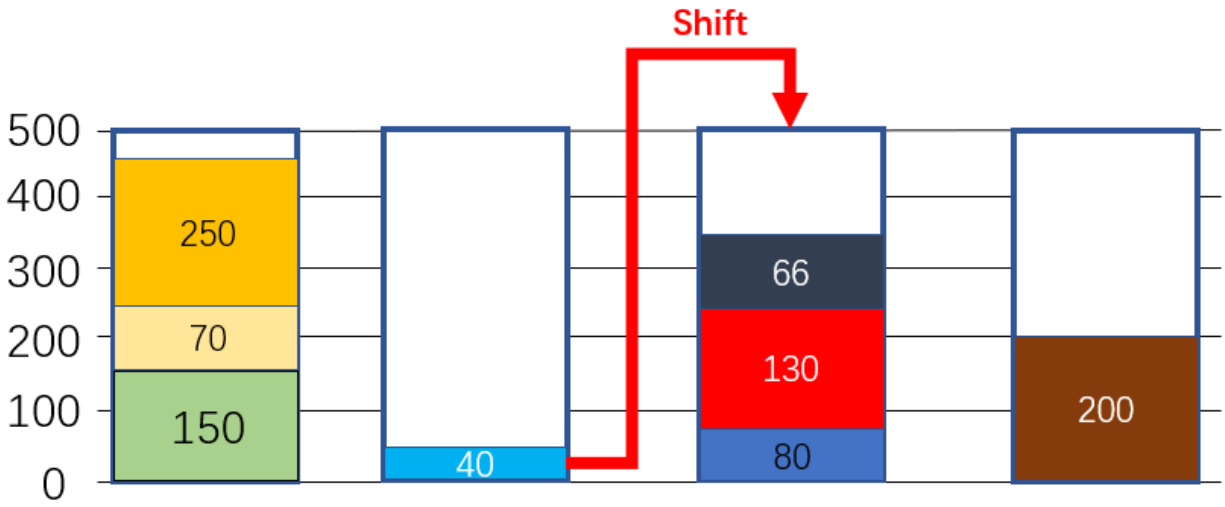

Figure 3: Heuristic1: Shift

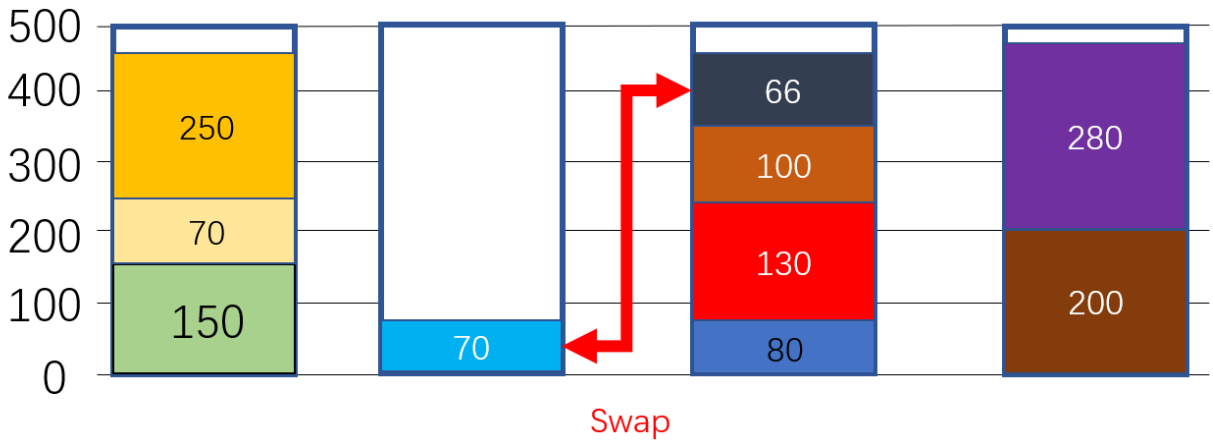

Figure 4: Heuristic2: Exchange_largest

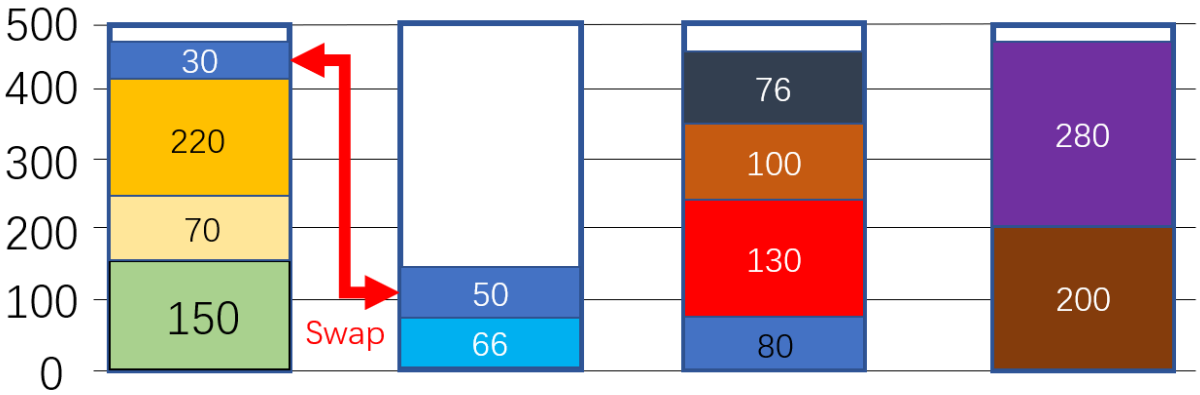

Figure 5: Heuristic3: Exchange_smallest

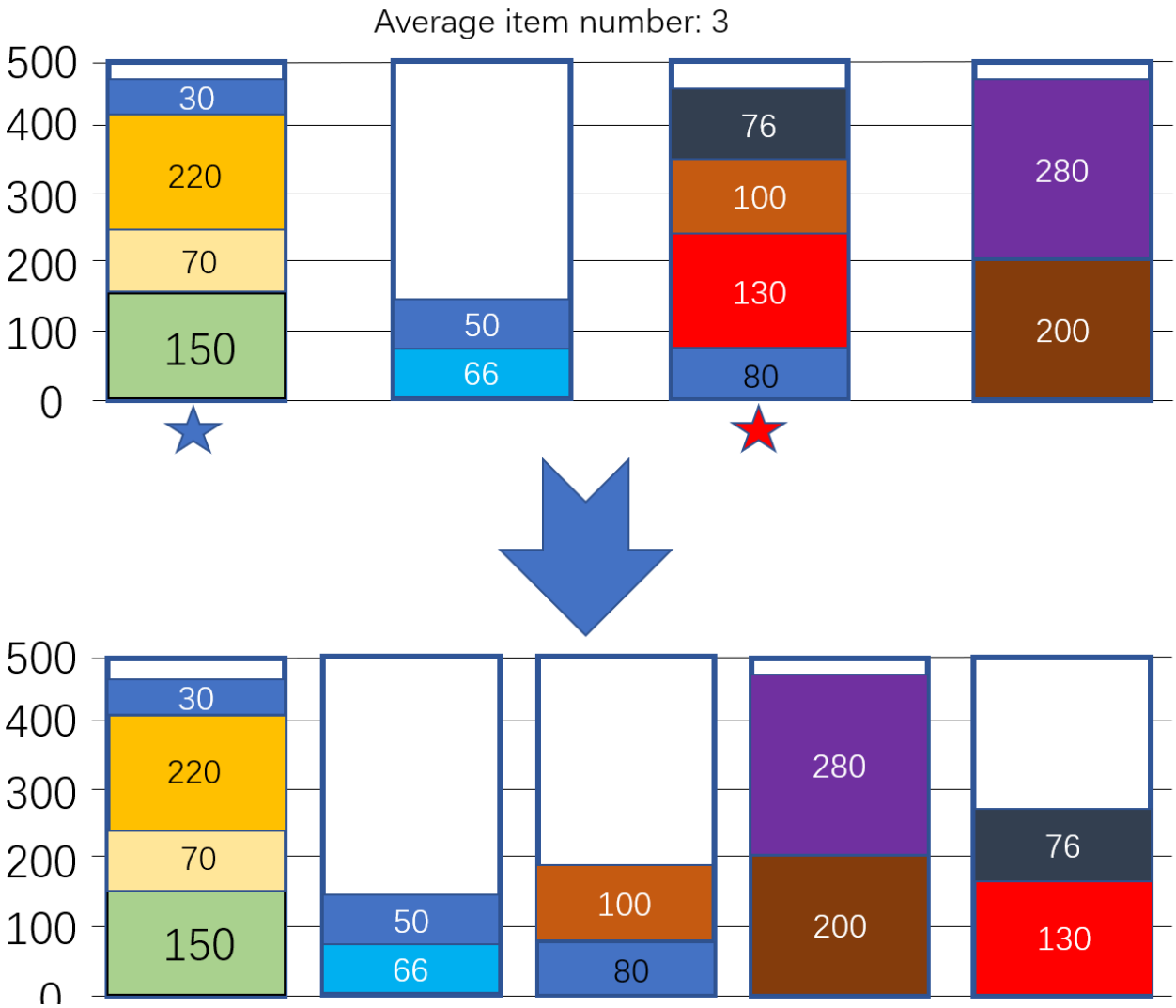

Figure 6: Heuristic4: Split

2.3.4 4. Split

Shown in *Figure.6*. The split procedure could be divided into two part. The first procedure is to randomly select a bin from those bins that whose quantity of items is more than the average number of item of per bin. The second procedure is to move these items to a new bin. This heuristic could restructure a certain bin with relatively more items so that these heuristics above could have more probability to reduce the use of bins.

2.4 Diversification

The shaking procedure is randomly pick two different items from two different bins that are not full and swap them. To avoid spending too much time on searching two items that can be swapped, I limit every search should be no longer than 1 seconds. This diversification process has the probability to regroup the structure of the items in every bin so that facilitating to search for a better solution rather than search locally.

2.5 Fitness function

The fitness function will judge the quantity of the bins after heuristic by comparison with the quantity of bins before heuristic. And three situations occur:

1. If the quantity of bins after is fewer than the quantity of bins before, then the new solution could be considered as a better solution.

2. If the quantity of bins of two cases are the same, then picking the two bins that involved with the move of items in the heuristic. Calculate the residual_gap by using the absolute value of the difference of the residual capacity of two bins after heuristic minus the absolute value of the difference of the residual capacity of the two bins before heuristic.

$$residual\_gap = |RC\_A_{after} - RC\_B_{after}| - |RC\_A_{before} - RC\_B_{before}|$$

RC: residual capacity, before: before heuristic, after:after heuristic
A,B: bins that involved with the move of items in the heuristic

If the residual_gap is bigger than 0 then the new solution is considered as a better solution. Otherwise the initial solution is better.

3. If the quantity of bins after is more than the quantity of bins before, then checking the items in the new bin. If any item in the new bin has fewer size than any item in the bin with the maximum residual, then the new solution is better. Otherwise the initial solution is better.

3 Statistical results

In this section, the statistical results will be shown in **(3.1)**.

3.1 Data

There are 20 instances in easy pattern, 20 instances in medium pattern and 10 instances in hard pattern. The result from every instance in 5 times are recorded and the best gap, worst gap are labeled and the absolute gap is calculated.

$$gap = Objective - Best\ known\ objective$$

$$Best\ gap = \{gap\}_{MIN}$$

$$Worst\ gap = \{gap\}_{MAX}$$

$$Abs\_gap = \frac{1}{n}\sum_{i}^{n} gap_i - Best\ known\ objective$$

3.1.1 Binpack1 (easy)

Identifier	1st gap	2nd gap	3rd gap	4th gaph	5th gap	Best gap	Worst gap	Abs_gap
u120_00	0	0	0	0	0	0	0	0
u120_01	0	0	0	0	0	0	0	0
u120_02	0	0	0	0	0	0	0	0
u120_03	0	0	0	0	0	0	0	0
u120_04	0	0	0	0	0	0	0	0
u120_05	0	0	0	0	0	0	0	0
u120_06	0	0	0	0	0	0	0	0
u120_07	0	0	0	0	0	0	0	0
u120_08	1	1	1	1	1	1	1	1
u120_09	0	0	0	0	0	0	0	0
u120_10	0	0	0	0	0	0	0	0
u120_11	0	0	0	0	0	0	0	0
u120_12	0	0	0	0	0	0	0	0
u120_13	0	0	0	0	0	0	0	0
u120_14	0	0	0	0	0	0	0	0
u120_15	0	0	0	0	0	0	0	0
u120_16	0	0	0	0	0	0	0	0
u120_17	0	0	0	0	0	0	0	0
u120_18	0	0	0	0	0	0	0	0
u120_19	0	0	0	0	0	0	0	0

3.1.2 Binpack3 (medium)

Identifier	1st gap	2nd gap	3rd gap	4th gaph	5th gap	Best gap	Worst gap	Abs_gap
u500_00	1	1	1	1	1	1	1	1
u500_01	1	1	1	1	1	1	1	1
u500_02	0	0	0	0	0	0	0	0
u500_03	1	1	1	1	1	1	1	1
u500_04	0	0	0	0	0	0	0	0
u500_05	0	0	0	0	0	0	0	0
u500_06	1	1	1	1	1	1	1	1
u500_07	1	1	1	1	1	1	1	1
u500_08	1	1	1	1	1	1	1	1
u500_09	0	0	0	0	0	0	0	0
u500_10	0	0	0	0	0	0	0	0
u500_11	0	0	0	0	0	0	0	0
u500_12	1	1	1	1	1	1	1	1
u500_13	1	1	1	1	1	1	1	1
u500_14	0	0	0	0	0	0	0	0
u500_15	1	1	1	1	1	1	1	1
u500_16	0	0	0	0	0	0	0	0
u500_17	1	1	1	1	1	1	1	1
u500_18	0	0	0	0	0	0	0	0
u500_19	1	1	1	1	1	1	1	1

3.1.3 Binpack11 (hard)

Identifier	1st gap	2nd gap	3rd gap	4th gaph	5th gap	Best gap	Worst gap	Abs_gap
HARD0	0	0	0	0	0	0	0	0
HARD1	0	0	0	0	0	0	0	0
HARD2	1	1	1	1	1	1	1	1
HARD3	1	1	1	1	1	1	1	1
HARD4	1	1	1	1	1	1	1	1
HARD5	0	0	0	0	0	0	0	0
HARD6	0	0	0	0	0	0	0	0
HARD7	1	1	1	1	1	1	1	1
HARD8	0	0	0	0	0	0	0	0
HARD9	0	0	0	0	0	0	0	0

4 Reflection

In this section, the performance of the algorithm will be discussed in *4.1* and the analysis of the result will be shown in *4.2*.

4.1 Performance discussion

The biggest drawback of this algorithm I think is the shaking procedure. The shaking procedure is a 1-1 swap at a time, which makes the changes of the solution are relatively limited and it may usually happen that the heuristic 1-4 can make no change after shaking, so it sometimes spends a lot of time but does no improvements.
Another drawback is the swaps in heuristics are merely 1-1 swap, which definitely limits the program to get a better solution. If there will be other heuristics with 2-1 swap or 3-1 swap, the solution could be better and there could more possible to use fewer bins than current situation.
The fitness function could also be better. In the current situation, for example: a new solution with the same bins quantity as the initial solution but negative residual_gap, which could reduce its use of bins in the process of VNS. But the fitness function will simply consider it as a worse solution because the same bins currently but negative residual_gap. The fitness function is short-sighted and it may reject some better solution. And I think this is the main reason why the program can't make the abs_gao to be all 0.

4.2 Result analysis

According to the three tables from 5 times running, the algorithm has made a considerable result. In general, every gap from every run of each instance is 0 or 1, which shows that the algorithm is relatively powerful because it can use bins whose quantity is the same as the best solution in most instances and the other cases use one extra bin only. This algorithm is extremely stable and it is obvious that every run of each instance is the same and the best gap and worst gap of every instance are the same, which leads to the same gap so that the abs_gap is also the same as any certain of gap. From the other side, the algorithm is not strong enough because it seems that it doesn't have a better solution to use less bins in the instances with gap 1.