COMP2051 AIM Coursework Report

Junyu Liu
20216355

How to Run the Code

The command line instruction should be typed in the following format:

```
python 20216355.py -s <inputfile> -o <outputfile> -t <maxtime>
```

which strictly follows the similar format for C/C++. If wrong formats are typed, there will be corresponding alert for typing a correct one.

Main Components of the Algorithm

- **Solution Encoding**
  According to the requirement of Bin Packing Problem, the solution is to find a way that packing all items into as few as possible bins. Thus, the main structure of the solution is encoding is designed as follows:

  A list of bins {Bin 1, Bin 2, …, Bin n}, where each bin contains an item list and a number indicates the residual capacity. The bin structure is presented as follows:
  Bin 1 – {item list 1, residual capacity 1}
  Bin 2 – {item list 2, residual capacity 2}
  …
  Bin n – {item list n, residual capacity n}

  Note that the structure presented here only provides high-level abstraction of the solution encoding, which contains all key encoding designs. For more details about how these encodings are implemented in codes, please refer to the code comments (More complex structures are implemented in code to enable convenient programming and calculation).

- **Fitness Function**
  Two level of fitness functions are used to evaluate the difference in objective between two neighboring solutions:
  1. The **first-level** fitness function is straightforward, by comparing the difference in terms of the bin number (objective) in two neighboring solutions. If a neighboring solution has smaller bin number than the current solution, then update the current solution by the neighboring solution.
  2. The **second-level** fitness function is calculated when there is no update in the first-level fitness function. In the second-level fitness function, the uniformity of the residual spaces of each bin in the bin list is calculated by the Euclidian Distance of these residual spaces to the mean residual space. The reason for the second-level fitness function is based on an experimental observation, in which more unevenly distributed (in terms of residual space) bin lists have higher chance to be optimized (e.g., remove a bin) by the following heuristics. Thus, although the bin number (direct objective) of the neighboring solution is not decreased, if it has a more uneven distributed bin list than the current solution, the current solution is still updated.

- **Neighborhoods**
  Three neighborhoods are used in this algorithm and are introduced as follows.
  1. **Neighborhood 1**
     The heuristic of neighborhood 1 is selecting the bin that has the largest residual capacity and try move the items in that bin to a sorted list rest of the bins (sorted from the largest residual capacity to the smallest residual capacity) using best fit descent heuristic. If there are no items left in the chosen bin after moving, remove the bin from the bin list.

2. **Neighborhood 2**
   The heuristic of neighborhood 2 is iteratively choosing two bins from the bin list and swap the largest size item in from the bin that has larger residual capacity with a combination of smaller size items in the other bin (Only valid swaps are accepted to prevent overflow in bins).
3. **Neighborhood 3**
   The heuristic of neighborhood 3 is iteratively choosing two bins from the bin list and move out all items of these two bins and then re-fill these items into the two bins using best fit heuristic.

- **Intensification Mechanisms**
  1. Intensify the elite solution using first-level fitness function
     In first-level fitness solution, the solution that has better objective (i.e., fewer bins) are accepted and updated.
  2. Second-level fitness function
     In the second-level fitness solution, the solutions that have no worse objective but provide more unevenly distributed bin list is updated since such solutions have better optimized bin structures and are more likely to have better objectives (decrease the number of bins) in future operations.
  3. Neighborhood 1 operation
     Neighborhood 1 moves the items from the bin that has largest residual capacity to other bins. This operation continuously creates chances for removing all items from the chosen bin, thus intensifies towards better solution.
  4. Neighborhood 2 operation
     Neighborhood 2 swap the largest size item in from the bin that has larger residual capacity with a combination of smaller size items in another bin, which provides more likely opportunities to have better objectives (decrease the number of bins) in future operations in Neighborhood 1, thus intensifies towards better solution.

- **Diversification Mechanisms**
  1. VNS shaking
     The shaking mechanism is to randomly choose two bins from the bin list and swap the largest item of these two bins if the swap is available. This diversification operation gives more possible space for searching and operating in all three neighborhoods.
  2. Neighborhood 3 operation
     Neighborhood 3 reshuffles the items in two bins, thus generally decreases the unevenness (in terms of residual capacity) of the bin list to diversify the solution. This diversification operation gives more possible space to remove/fill items from/into these two reshuffled bins.

Statistical result

The results for all three problems in 5 runs are provided in the table below (the values shown in the table are the absolute gaps to the given best results):

Problem	Average Result	Best Result	Worst Result
binpack1.txt	0.3	0	1
binpack3.txt	1.6	0	2
binpack11.txt	0.2	0	1

Discussion and Reflection

- **Algorithm Analysis**

This VNS algorithm uses 3 neighbors and 1 shaking mechanism, in which neighbor 1 and neighbor 2 tend to intensify for better objective while neighbor 2 and VNS shaking tend to diversify to create more opportunities for future intensification. The designs of the three neighbors are inspired from [1] without violating the original insights, thus the overall correctness of these neighbors can be high. The neighbor number is set to 3 according to experiments (When neighbor numbers are fewer than 3 or add new neighbors, the results are not as optimal compared to 3 neighbors.). However, it is still possible to add or modify neighbors to further improve the solution objective.

- **Results Analysis**
  From the results in the table above, both the results for "binpack1.txt" and "binpack11.txt" have achieved desired level of objectives (the gap to the best result is less than or equal to 1), while the results for "binpack3.txt" still have an average deviation of 0.6. One possible reason for this issue is that the item number in each problem of "binpack3.txt" are very large and the shaking is not sufficient. However, from my experiments on adjusting shaking parameters and changing the shaking mechanism (e.g., If the item number in a chosen bin is greater than the average number of items in a bin, then create a new bin and move half of the items if the chosen bin to the new bin), there is no steady improvement. Another possible reason is that the neighbors are not enough and may need new neighbors. However, from my experiments on adding new neighbors (e.g., Reshuffle items in three iteratively chosen bins), there is no steady improvement. More research and parameter tuning may be needed for further improvement.

- **Possible Improvements**
  1. Add new neighbors (e.g., Reshuffle items in three iteratively chosen bins) and tune the parameters.
  2. Change VNS shaking mechanisms (e.g., If the item number in a chosen bin is greater than the average number of items in a bin, then create a new bin and move half of the items if the chosen bin to the new bin) and tune parameters.
  3. Change/add more fine-tuned fitness evaluation mechanisms (e.g., Co-evaluate the evenness of residual spaces and objective with hyperparameters.)

Reference

[1] Bai, R., Blazewicz, J., Burke, E. K., Kendall, G., & McCollum, B. (2012). A simulated annealing hyper-heuristic methodology for flexible decision support. *4OR*, *10*(1), 43-66.