Variable Neighbourhood Search for Bin Packing Problem Report

ID number: 20217124

Instructors: Prof.Ruibin Bai, Dr.Huan Jin (Joyce)

Module code: COMP2051

05/2022

1 Introduction of the Algorithm

The variable neighborhood algorithm, referred to as VNS, uses neighborhood structures composed of different actions to search alternately and achieves a good balance between concentration and dispersity. The VNS has two main components: variable neighbourhood descent(VND) and shaking phase. Furthermore, the neighborhood is the set of all the solutions obtained by operating on the current solution. This VNS algorithm is designed for solving the Bin Pack Problem. Furthermore, this algorithm is designed and implemented according to the logic of the following pseudo-code.[Talbi, 2009]

Algorithm 1 Template of the general variable neighborhood search algorithm.

```
Input: a set of neighborhood structures $N_k$ for $k = 1, ..., k_max$ for shaking.
a set of neighborhood structures $N_l$ for $k = 1, ..., l_max$ for local search.
$x = x_0$;/*Generate the initial solution*/
Output: Best found solution.
  repeat
    for $k = 1$ to $k_{max}$ do
      Shaking: pick a random solution $x^{'}$ from the $k_{th}$ neighborhood $N_k(x)$ of $x$;
      Local search by VND ;
      for $l = 1$to $l_{max}$ do
        Find the best neighbor $x^{''}$ of $x^{'}$ in $N_l(x^{'})$ ;
        if $f(x^{''}) < f(x^{'})$ then $x^{'} = x^{''}; l = 1$;
        else
          $l = l + 1$
          Move or not;
        if local optimum is better than $x$ then
          $x = x^{''}$;
          Continue to search with $N_1$ $(k = 1)$ ;
        else
          $k = k + 1$
  until Stopping criteria
```

2 Main Components of the Algorithm

This section illustrates all components of the VNS algorithm and some reasons for choosing the encoding methods.

2.1 Solution Encoding

The value encoding method is used for encoding the solution. The solution is encoded as a string of Bin objects representing all Bin objects, and the Bin object has a *cap_ left* attribute representing the remaining capacity of Bin. Each Bin object contains a list of Item objects representing the items loaded in the Bin. Item object has index and volume attributes. The item index is used to represent the solution result or check the solution's feasibility. The volume attribute of an item is used to determine whether the Bin's capacity is enough or to check the feasibility of the solution. Furthermore, the concrete implementation of encoding is demonstrated in the code below:

```
class Solution:
def __init__(self, num_of_bins, problem):
    self.problem = problem
    self.num_of_bins = num_of_bins
bins = [Bin]

class Bin:
cap_left = 0
```

```
    def __init__(self, cap_left: int):
        self.cap_left = cap_left
    items = [Item]

    class Item:
    def __init__(self, index: int, volume: int):
        self.index = index
        self.volume = volume
```

Listing 1: Code Implementation of Fitness Function

2.2 Fitness Function

The fitness function is referred to as the evaluation function. This evaluation compares the number of two bins, and the solution containing fewer bins is more optimal than another one. However, the two solutions could have the same number of bins. The evaluation function compares the number of bins which has zero left capacity. The solution which has more zero capacity bins is the better one. Otherwise, if the number of the full bin of two solutions is the same, the current solution is still retained.

2.3 Initial Solution

The algorithm initializes the solution by the greedy search. Firstly, it packs the most oversized item into a bin until the bin's capacity is zero or the capacity is not enough for another item. Then, a new bin is opened for packing the rest items. The procedure does not stop until all items are packed. The following code is the concrete implementation of initializing solution.

```
def init_solution(prob):
    sln = Solution(0, prob)
    bins = []
    while len(sln.problem.items) > 0:
        items = []
        cap_left = sln.problem.cap_of_bin
        # find the item which has the biggest volume
        sln.problem.items.sort(key=functools.cmp_to_key(cmp2))
        items.append(sln.problem.items[0])
        cap_left -= sln.problem.items[0].weight
        sln.problem.items.pop(0)
        # sort by descending
        sln.problem.items.sort(key=functools.cmp_to_key(cmp1))
        while len(sln.problem.items) > 0 and sln.problem.items[0].weight <= cap_left:
            # sort by ascending
            sln.problem.items.sort(key=functools.cmp_to_key(cmp2))
            for it in sln.problem.items:
                if it.weight <= cap_left:
                    items.append((it))
                    cap_left -= it.weight
                    sln.problem.items.remove(it)
            # sort by descending
            sln.problem.items.sort(key=functools.cmp_to_key(cmp1))
        new_bin = Bin(cap_left)
        new_bin.items = items
        bins.append(new_bin)
        sln.num_of_bins += 1
    sln.bins = bins
    return sln
```

Listing 2: Code Implementation of Initialization

2.4 Neighborhood

2.4.1 Set of Neighbors

Neighbourhood refers to the set of solutions that can be obtained by performing neighbourhood actions on the current solution. The algorithm used two different approaches to find the neighbours.

2.4.2 Variable Neighborhood Descent

The Variable Neighborhood Descent(VND) procedure searches for the objective solution in one neighbourhood set. The neighbourhood selection strategy references Prof.Bai's research, with minor changes.[Bai et al., 2012] Changing the neighbours is finding two bins and combining the items in these two bins according to the slight capacity deviation. If there is more than one recombination method, select the one with the larger number of items. This action may reduce the number of Bins in the solution. Meanwhile, the bin selection is according to a probability calculated by $\Psi = \frac{resCap_i}{\Sigma resCap_i}$, where the $resCap_i$ is the residual capacity of bin i.

2.4.3 Shaking Phase

The shaking phase avoids getting a local objective solution for only one neighbourhood instead of an optimal global solution. The method of finding neighbours is changing items in two bins randomly. There are two different strategies to change items. Firstly, the heuristic exchanges the largest weight item from the largest residual capacity with another item or other items from a randomly selected bin. Necessarily, this bin is not full of items. However, if the heuristic cannot find satisfying bins, it randomly selects two different bins and recomposes their items. The recomposition principle is similar to the VND procedure.

2.4.4 Stopping Criteria

When the number of bins of the current solution is less than or equal to the optimal solution, the algorithm stops and outputs the current solution. Otherwise, the algorithm chooses another neighbourhood set by the shaking phase to find the global objective solution.

2.5 Intensification and Diversification Mechanisms

The intensification is to find the optimal neighborhood solution in the shaking phase neighborhood set. The exploitation does not stop until the optimal neighborhood solution is found or it reaches the maximum iteration. Once the algorithm finds the optimal solution in the current neighborhood set, it finds the neighbors of the other solution generated by the shaking phase, which avoids converging to poor local optima. In addition, the exploration also does not stop if the heuristic finds the global objective or the exploitation reaches the maximum iteration. However, the algorithm only accepts the improving solution, which the fitness function evaluates.

3 Statistical Result

This section illustrates the statistical table of all instance. The table1 shows the result of binpack problem1, the table 2 shows the result of binpack problem3 and the table3 shows the result of binpack problem11.

Instance_Name	AVG	BEST	WORST	BEST_GAP	WORST_GAP
u120_00	48	48	48	0	0
u120_01	49	49	49	0	0
u120_02	46	46	46	0	0
u120_03	49	49	49	0	0
u120_04	50	50	50	0	0
u120_05	48	48	48	0	0
u120_06	48.4	48	49	0	1
u120_07	49	49	49	0	0
u120_08	51	51	51	1	1
u120_09	47	47	47	1	1
u120_10	52	52	52	0	0
u120_11	49	49	49	0	0
u120_12	49	49	49	1	1
u120_13	49	49	49	0	0
u120_14	50	50	50	0	0
u120_15	48	48	48	0	0
u120_16	52	52	52	0	0
u120_17	52	52	52	0	0
u120_18	49	49	49	0	0
u120_19	50	50	50	1	1

Table 1: The Statistics Result of Problem 1

Instance_Name	AVG	BEST	WORST	BEST_GAP	WORST_GAP
u500_00	199	199	199	1	1
u500_01	202	202	202	1	1
u500_02	202.2	202	203	0	1
u500_03	205	205	205	1	1
u500_04	206	206	206	0	0
u500_05	206	206	206	0	0
u500_06	208	208	208	1	1
u500_07	205.2	205	206	1	2
u500_08	197	197	197	1	1
u500_09	202	202	202	0	0
u500_10	200.2	200	201	0	1
u500_11	201	201	201	1	1
u500_12	200	200	200	1	1
u500_13	196	196	196	0	0
u500_14	204	204	204	0	0
u500_15	201.6	201	202	0	1
u500_16	202	202	202	0	0
u500_17	198.2	198	199	0	1
u500_18	202.2	202	203	0	1
u500_19	197	197	197	1	1

Table 2: The Statistics Result of Problem 3

Instance_Name	AVG	BEST	WORST	BEST_GAP	WORST_GAP
HARD0	56	56	56	0	0
HARD1	57	57	57	0	0
HARD2	57	57	57	1	1
HARD3	56	56	56	1	1
HARD4	57	57	57	0	0
HARD5	56	56	56	0	0
HARD6	57	57	57	0	0
HARD7	55	55	55	0	0
HARD8	57	57	57	0	0
HARD9	56	56	56	0	0

Table 3: The Statistics Result of Problem 11

All the tables shown above indicate that all the statistics results(average, best and the worst result of algorithm, and the best, worst gap between best known solution) are not very volatile. Furthermore, the correctness of the algorithm has been verified by the binchecker executable file. Hence, this VNS algorithm is stable.

4 Discussion and Reflection

This section discusses the result of the Bin Pack Problem solved by the VNS algorithm.
The result of the algorithm approaches the current best-known solution or equals the current best-known solution. For the simple, medium, and hard problems, the average absolute gaps between the VNS result and the optima are greater than zero but less than one when the algorithm only runs once. Furthermore, the simple and the hard problems are solved in less than half the maximum running time, and the medium problems are solved in a long time but do not exceed the maximum running time. However, the algorithm cannot find all the neighborhoods due to a few neighbors' explorations strategies. In the original implementation of the algorithm, a split strategy was used to explore more neighbors. Nevertheless, this decision was discarded due to the poor final results of the combined convergence strategy.
Considering the performance and results of the algorithm, this VNS has decent feasibility.

5 How To Run the Python Code

Please follow the instructions to use this VNS to solve the Bin Pack Problem. Firstly, please ensure that all problem files and solution files are in the same folder before running this program. Furthermore, please input the ***"python 20217124.py -s data_file -o solution_file -t max_ time"*** on the terminal to run the program.

References

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

E.-G. Talbi. Variable neighborhood search. In *Metaheuristics: from design to implementation*, volume 74, pages 150–152. John Wiley & Sons, 2009.