COMP 2051 Artificial Intelligence Methods Coursework Report

Zhuoran BI
20217231

1. Introduction

The bin packing problem(BPP) is an optimization problem, in which items of different sizes must be packed into a finite number of bins or containers, each of a fixed given capacity, in a way that minimizes the number of bins used. The problem is NP-hard, and the corresponding decision problem - deciding if items can fit into a specified number of bins is NP-complete.

A possible integer linear programming formulation of the problem is[1]

$$minimize \quad K = \sum_{j=1}^{n} y_j$$

$$subject\,to \quad K \geq 1,$$

$$\sum_{i \in I}^{n} s(i) x_{ij} \leq B y_j, \forall j \in \{1, \ldots, n\}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \forall i \in I$$

$$y_j \in \{0, 1\}, \forall j \in \{1, \ldots, n\}$$

$$x_{ij} \in \{0, 1\}, \forall i \in I \, \forall j \in \{1, \ldots, n\}$$

where $y_j = 1$ if bin j is used and $x_{ij} = 1$ if item i is put into bin.

2. Initial solution

There are a group of basic algorithms that can solve the 1D bin packing problem. For example, some basic heuristic algorithms and their performance are shown below:

Algorithm	Approximation guarantee	Worst case list L	Time-complexity
Next Fit	$NF(L) \leq 2 \cdot \mathrm{OPT}(L) - 1$	$NF(L) = 2 \cdot \mathrm{OPT}(L) - 2$	$\mathcal{O}(\mid L \mid)$
First Fit	$FF(L) \leq \lfloor 1.7\mathrm{OPT}(L) \rfloor$	$FF(L) = \lfloor 1.7\mathrm{OPT}(L) \rfloor$	$\mathcal{O}(\mid L \mid \log(\mid L \mid))$
Best Fit	$BF(L) \leq \lfloor 1.7\mathrm{OPT}(L) \rfloor$	$BF(L) = \lfloor 1.7\mathrm{OPT}(L) \rfloor$	$\mathcal{O}(\mid L \mid \log(\mid L \mid))$
Worst Fit	$WF(L) \leq 2 \cdot \mathrm{OPT}(L) - 1$	$WF(L) = 2 \cdot \mathrm{OPT}(L) - 2$	$\mathcal{O}(\mid L \mid \log(\mid L \mid))$

Comparison table

I implemented two initial solutions which are First Fit Decreasing (FFD) and Best Fit Decreasing (BFD). This means that, if I can know all the items in advance, I could first sort this problem by a decreasing order and then apply it to First Fit and Best Fit. For this problem, I choose Best Fit Decreasing (BFD) as my initial solution, as it provides a solution with a smaller bin slack than First Fit Decreasing (FFD).

3. Variable neighbourhood search (VNS) for BPP

The variable neighbourhood search algorithm for the BPP is based on moves. A move is the transfer of an item from its current bin to another bin or the swap of a pair of items across their respective current bins. And only valid moves, which do not violate the bin capacity constraint, are taken into consideration[2].

3.1. Solution encoding

My coursework is done by using C++.

A solution of this BPP is consist of a problem instance, an objective value, a feasibility value a Vector that stores the bins of the solution.

In this vector, every element is a bin, and the total number of bins that are used can be represented by the size of the vector.

In addition, both bins and items are structures. Bin has the components that are capacity left and a vector which stores all the items in

this bin. Item has weight and index, which is the weight of the item and their ID in their problem instance.

3.2. Neighbourhood Methods

3.2.1. Heuristic 1 (Transfer)

This heuristic selects each item from the bin with the largest residual capacity and tries to move the items to the rest of the bins using the best fit descent heuristic[3]. This step is vital, as it is possible to move all the items of the largest residual capacity bin to become empty, then the number of bins could reduce.

3.2.2. Heuristic 2 (Swap)

Exchange largestBin_largestItem: Selects the largest item from the bin with the largest residual capacity and exchanges this item with another smaller item (or several items whose capacity sum is smaller) from another randomly selected non-fully-filled bin[3].

Above is the original heuristic algorithm, but I enhanced it to swap with the best bin which has the biggest weight gap between these two items.

This is a step working for Heuristic 1, if a smaller bin is placed into the largest residual capacity, it is more likely to be transferred to another non-fully filled bin.

3.2.3. Heuristic 3 (Swap)

This heuristic is very similar to the heuristic 2, the only difference is that the target bin of this heuristic is the second-largest bin, which is the largest bin in heuristic 2.

From my observation, in many cases, the algorithm with heuristic 1 and heuristic 2 will easily get into a trap if none of the items could be transferred to another bin and no smaller item could be exchanged.

So, heuristic 3 was designed by me to solve this problem, the second-largest bin is more like to hold the item from the bin with the largest residual capacity. The more the capacity in the second bin left, the more the transfer could happen. As a result, this heuristic also work for Heuristic 1.

3.3. Objective function

The most convincing objective function is the number of bins used, as a result; I take this as my main objective to decide if my algorithm would like to take this move.

But the BPP is special, as the number of bins is always the same, although the distribution of the items inside each bin is different. So I defined that, if there is a move that satisfies my design will be accepted, which goes different with a unique neighbourhood.

Heuristic 1: If the transfer happened, the result will be accepted. No matter how many items in the target bin are transferred, it is good for reducing the number of bins.

For, Heuristic 2 and Heuristic3 I first design a fitness fuction, this is:
$|After[1].capacityLeft - After[2].capacityLeft| - |Before[1].capacityLeft - Before[2].capacityLeft|$

note: Before[1] and Before[2] are the two bins before swap, After[1] and After[2] are the two bins after swap.

But then I find that, using if the items are swapped as the objective is more straightforward.

So, Heuristic 2 and Heuristic 3: If the swap happened, the result will be accepted.

3.4. Shaking

Shaking is defined as a random swap of a pair of items, which are not in the same bin and have different weights. As a result, the shaking could make sure that every shaking could make sense.

This process executes when Heuristic 1, Heuristic 2 and Heuristic 3, all did not find an acceptable solution.

At this moment, some random swap should be tackled to break the structure of the whole solution, which Shaking does.

3.5. Intensification and diversification

For intensification, the core of my design is the Heuristic 1, as it is the only step for my algorithm to have a chance to delete an empty bin. The

other two steps are both swapping, so the total number of bins will not change. Because of this, my design is using other Heuristic methods to enhance the chances for Heuristic 1 to transfer out the item. The Heuristic 2 is used to make the items in the largest bin to be smaller and the Heuristic 2 is used to create larger space in other bins for items to transfer in. As time goes by, the transfer condition will be created, again and again, so the solution will be intensified.

For diversification, shaking is a method to keep the diversity. In some conditions, if all the three Heuristic methods can not get a batter solution, the shaking will randomly swap a pair of items from non-full bins. Because of this, the structure of each bin will be changed, then some Heuristic methods could work again.

4. Result

Instance name	**Number of runs**	**1**	**2**	**3**	**4**	**5**	**Best**	**Worst**	**Avg**
u12_00		0	1	0	1	0	1	1	**0.4**
u12_01		0	0	0	0	0	0	0	**0**
u12_02		0	0	0	0	0	0	0	**0**
u12_03		0	0	0	0	0	0	0	**0**
u12_04		0	0	0	0	0	0	0	**0**
u12_05		1	1	1	1	1	1	1	**1**
u12_06		0	1	1	0	1	0	1	**0.6**
u12_07		1	1	1	1	1	1	1	**1**
u12_08		1	1	1	1	1	1	1	**1**
u12_09		0	0	0	0	0	0	0	**0**
u12_10		1	1	1	1	1	1	1	**1**
u12_11		1	1	1	1	1	1	1	**1**
u12_12		0	0	0	0	0	0	0	**0**
u12_13		0	0	0	0	0	0	0	**0**
u12_14		0	0	0	0	0	0	0	**0**
u12_15		1	0	0	0	1	0	1	**0.4**
u12_16		0	0	0	0	0	0	0	**0**
u12_17		0	0	0	0	0	0	0	**0**
u12_18		0	1	0	1	1	0	1	**0.6**
u12_19		1	1	1	1	1	1	1	**1**

Instance name	Number of runs	1	2	3	4	5	Best	Worst	Avg
u500_00		2	1	2	1	2	1	2	**1.6**
u500_01		1	1	1	1	1	1	1	**1**
u500_02		1	1	1	1	1	1	1	**1**
u500_03		2	2	2	1	2	1	2	**1.8**
u500_04		2	2	2	1	1	1	2	**1.6**
u500_05		0	0	0	0	0	0	0	**0**
u500_06		2	2	2	1	2	1	2	**1.8**
u500_07		2	2	2	2	2	2	2	**2**
u500_08		1	1	1	1	1	1	1	**1**
u500_09		1	1	1	1	1	1	1	**1**
u500_10		1	1	1	1	1	1	1	**1**
u500_11		2	1	2	2	2	1	2	**1.8**
u500_12		2	2	1	2	2	1	2	**1.8**
u500_13		1	1	1	1	1	1	1	**1**
u500_14		1	1	1	1	1	1	1	**1**
u500_15		1	1	1	1	1	1	1	**1**
u500_16		1	1	1	1	1	1	1	**1**
u500_17		2	2	2	2	2	2	2	**2**
u500_18		2	2	2	2	2	2	2	**2**
u500_19		2	2	2	2	2	2	2	**2**

Instance name	Number of runs	1	2	3	4	5	Best	Worst	Avg
HARD0		0	0	0	0	0	0	0	**0**
HARD1		0	0	0	0	0	0	0	**0**
HARD2		1	1	0	0	1	0	1	**0.6**
HARD3		1	1	1	0	0	0	1	**0.6**
HARD4		0	0	0	0	0	0	0	**0**
HARD5		0	0	0	0	0	0	0	**0**
HARD6		0	0	0	0	0	0	0	**0**
HARD7		0	0	0	0	0	0	0	**0**
HARD8		0	0	0	0	0	0	0	**0**
HARD9		0	0	0	0	0	0	0	**0**

5. Reflection and conclusion

As you can see in the result part, my algorithm does great when dealing with the instances with the small input item number(for example, HARD), but perform weaker when dealing with some large amount of input items, which can indicate that my algorithm has some drawbacks.

My solution will get better as the running time goes. If the number of items is too large, it will take more time for my algorithm to random shake a suitable solution for my Heuristic methods to operate. As a result, maybe it will take a longer time to get the best solution for some large amount of items input.

Future work could explore the possibility of designing more effective Heuristic methods and a better initial solution.

In all, the success of the VNS algorithm used in the Bin Packing Problem indicates that it is a flexible framework, which may tackle many combinatorial optimization problems.

References

[1] Martello, Silvano; Toth, Paolo (1990), "Bin-packing problem" (PDF), *Knapsack Problems: Algorithms and Computer Implementations*, Chichester, UK: John Wiley and Sons, ISBN 0471924202

[2] Fleszar, Krzysztof, and Khalil S. Hindi. "New heuristics for one-dimensional bin-packing." Computers & operations research 29.7 (2002): 821-839.

[3] Bai, Ruibin, et al. "A simulated annealing hyper-heuristic methodology for flexible decision support." 4OR 10.1 (2012): 43-66.