Introduction to AI
Final Project Report

Bin-Packing Problem Under Multiple-Criterions
(A.I. Multi-Criterion Optimization)

By: Moneer Omar

moneer.omar@mail.huji.ac.il
# Table of Contents

1. Abstract .........................................................5
2. Problem Description .............................................6
   2.1. Overview ..................................................6
   2.2. Applications ............................................6
   2.3. Formal Definition .......................................6
   2.4. Observations ............................................7
   2.5. The Conflicting Nature .................................8
   2.6. Problem Complexity ...................................9
3. Algorithm.........................................................9
   3.1. Setup ....................................................10
   3.2. Heuristics Approximation Approach ....................12
4. Results .........................................................15
5. Conclusions ...................................................18
1. Abstract

In this project, we consider a proposed heuristics approach to the multi-objective bin packing problem (BPP). We mean by multi-objective: meeting the following criterions: 1. minimizing the bins used, and, 2. minimizing the heterogeneousness of the elements in each bin. This leads to a bi-objective formulation of the problem with a tradeoff between the number of bins used and their heterogeneousness. A Best-Fit approximation algorithm is introduced to solve the problem followed by the results of the tests that were carried out.
2. Problem Description

2.1. Overview:
The basic idea (of the single constrain version) is to find the best way to pack a bunch of objects (usually of different sizes) into bins (which are usually of the same size).

There are many variants of this problem, such as 2D packing, linear packing, packing by weight, packing by cost, and so on.

The variant of the problem we are dealing with is as follows:
A given number of n items have to be packed into n bins each of capacity c, where each item j is characterized with \( w_j \) (weight) and \( a_j \) (attribute). The attribute describes the properties of the items on a nominal scale. Thus, two items are compared with respect to their attributes.

2.2. Applications:
From filling up containers, loading trucks with weight capacity to creating file backup in removable media and technology mapping in semiconductor chip design...

It’s also related to many other problems, like: Knapsack problem, cutting stock problem, prefect square dissection, and others...

2.3. Formal definition:
Standard problem statement: What is the minimum number \( k \) of identical bins, each of size \( C \), needed to store a finite collection of indivisible items having weights \( w_1, w_2, \ldots, w_n \), with the attributes \( a_1, a_2, \ldots, a_n \) such that the sum of the weights of the items in each bin does not exceed \( C \), while attempting to reduce the heterogeneousness in each bin?
The mathematical definition of the problem is as follows:

\[ y_i = \begin{cases} 1, & \text{If bin } i \text{ is used} \\ 0, & \text{otherwise} \end{cases} \]

\[ x_{ij} = \begin{cases} 1, & \text{If item } j \text{ is assigned to bin } i \\ 0, & \text{otherwise} \end{cases} \]

\[ u_i = \# \text{ of distinct attributes in bin } i \]

Now, we are asked to minimize the following:

1. \( z_1 = \sum_{i=1}^{n} y_i \)
2. \( z_2 = 1/z_1 \sum_{i=1}^{n} u_i \)

Under the following conditions:

\[
\begin{align*}
\sum_{j=1}^{n} w_j x_{ij} & \leq c y_i \text{ for } i \in \{1,2, \ldots, n\} \\
\sum_{j=1}^{n} x_{ij} & = 1 \text{ for } j \in \{1,2, \ldots, n\} \\
y_i & = 0 \text{ or } 1 \\
x_{ij} & = 0 \text{ or } 1
\end{align*}
\]

Expression (1) minimizes the number of bins. Expression (2) minimizes the average heterogeneousness of the bins.

2.4. **Observations:**

a. Unused bins \((y_i = 0)\) clearly have \(u_i = 0\).

b. Used bins \((y_i = 1)\) have a possible minimal value of \(u_i = 1\). That happens when all the items in the bin have the same attribute.

c. The values of \(a_i\) are bounded by the number of items in the bin \(i\).
2.5. **The Conflicting Nature of the Problem**

The two purposes of the problem are of conflicting nature: while a large number of bins allow minimal heterogeneousness by packing of items which are fully homogenous, yielding to $z_2 = 1$, minimizing $z_1$ requires packing items of different attributes into the same bin.

It can be inferred that no single solution $x$ exists in space of feasible solutions $X$ that equally minimizes the two objective functions $z_1$ and $z_2$.

Thus we are going to evaluate a solution $x$ with respect to a vector $Z(x) = (z_1(x), z_2(x))$.

Here, we need to remember some definitions from Game theory we learnt in A.I. course related to the problem:

**Definition 1 (Dominance):** A vector $Z(x)$ is said to dominate $Z(x')$ (in the k-dim space) iff $z_i(x) \leq z_i(x')$ for $i : 1 \ldots k$, and there exists $i$ s.t. $z_i(x) < z_i(x')$. We denote this dominance: $Z(x) \leq Z(x')$.

**Definition 2 (Efficiency, Pareto-optimality):** The vector $Z(x)$ is said to be efficient if no solution $x'$ exists s.t. $Z(x') \leq Z(x)$. $x$ is called Pareto-optimal, and the set of Pareto-optimal alternatives is called Pareto-set $P$.

Hence, the resolution of the problem has to be all the efficient outcomes, or the Pareto-set $P$.

**Illustration:**

![Figure 1: A bunch of items in the right side to be packed in bins 1, 2 and 3. Lengths can be seen as weights $w_i$, and colors can be thought as attributes $a_i$.](image)
2.6. *Problem complexity:*

Bin packing is an NP-hard problem, yet there are many heuristics have been developed: for example the first fit algorithm provides a fast but often non-optimal solution, involving placing each item into the first bin in which it will fit. The algorithm can be made much more effective by first sorting the list of elements into decreasing order (sometimes known as the first-fit decreasing algorithm), although this still does not guarantee an optimal solution, and for longer lists may increase the running time of the algorithm. It is known, however, that there always exists at least one ordering of items that allows first-fit to produce an optimal solution.

3. **Algorithm**

3.1. **Setup**

We think of packing as a two-tier problem:

1. Reordering candidate objects (the top shelf)
2. Packing them (in the bottom shelf)

All different algorithms we are going to describe create another bin whenever the algorithm cannot use the existing ones.

a. **Reorder**

Here we can rearrange the objects in the top shelf in various ways:

1. **As Given:** the objects remain intact
2. **Regular**: maximizes the distances between similarly-sized objects.

![Figure 2: candidates reordered in a regular pattern](image)

3. **Random**: reorders the candidate objects randomly.

![Figure 3: candidates reordered randomly.](image)

4. **Ascending**: reorders the objects from smallest to largest.

![Figure 4: candidates reordered in an ascending order.](image)
5. **Descending**: orders the objects from largest to smallest.

![Figure 5: candidates reordered in a descending order.]

### b. Pack (in the single-objective version)

Now, we take the objects from the top shelf and pack them into the bottom shelf using different algorithms.

1. The **First Fit** algorithm places a new object in the leftmost bin that still has room.

2. The **Last Fit** algorithm places a new object in the rightmost bin that still has room.

3. The **Next Fit** algorithm places a new object in the rightmost bin, starting a new bin if necessary.

4. The **Best Fit** algorithm places a new object in the fullest bin that still has room.

5. The **Worst Fit** algorithm places a new object in the emptiest existing bin.

6. The **Almost Worst Fit** algorithm places a new object in the second-emptiest bin.

7. **Sum of squares fit**: Choose bin such that sum-of-squares heuristic is minimized.
3.2. A Heuristics Approximation Approach

Unfortunately, packing algorithms described above does not take into consideration the heterogeneousness of the items. Therefore, a method controlling the heterogeneousness should be included. The algorithm in the bottom describes such an attempt by allowing successive computation of alternatives with different heterogeneousness levels and this is an idea of how to compute an approximation to the vector optimization problem described above.

The algorithm is based on the conventional method. However, we added a control parameter $u_{\text{max}}$, initialized with a value of $u_{\text{max}} = u = 1$. This means that only fully homogenous items are allowed. So if we want to assign an item, it will be assigned to a best-fit bin containing other items of identical attributes.
3.2. Algorithm: Multi-criterion Best-Fit algorithm

Input: \( s, \alpha \)

1. Calculate the maximum possible heterogeneousness of a bin, \( \tilde{u} \)
2. Initialize: \( u = 1 \)
3. \( P^{\text{approx}} = \emptyset \)
4. Repeat:
   For \( m = 1 \) to \( \alpha \) do:
      i. Construct a new solution \( x \):
      ii. For all \( n \) items do:
         1. Compute the maximally allowed heterogeneousness \( u_{\max} \) of the best-fit bin:
            \[ u_{\max} = \left\lfloor u \right\rfloor \text{ with probability } q_u \]
            \[ u_{\max} = \left\lceil u \right\rceil \text{ with probability } 1 - q_u \]
      5. Compute the best-fit bin with respect to \( u_{\max} \)
      6. Assign \( i \) to the best-fit bin
5. \( P^{\text{approx}} \) is updated with \( x \):
   Remove all elements in \( P^{\text{approx}} \) witch are dominated by \( Z(x) \);
   Add \( x \) to \( P^{\text{approx}} \) if \( Z(x) \) is not dominated by any element in \( P^{\text{approx}} \)
6. End for
7. Update \( P^{\text{approx}} \) with \( x \):
8. \( u = u + s \)
9. Return \( P^{\text{approx}} \)

* \( q_u = 1 - (u - \left\lfloor u \right\rfloor) \)
Discussion:
Since $u$ is incremented by $s$, it becomes increasingly high, leading to best-fit bins become possible that have a higher heterogeneousness. This concept is randomized throughout the generation of solutions, allowing a gradual transition from $u_{\text{max}} = u = 1$ to the maximum possible heterogeneousness $u_{\text{max}} = \bar{u}$.

Throughout the algorithm, an archive for best solutions $P_{\text{approx}}$ is kept and updated. This set of solutions represents an approximation to the true Pareto-set $P$.

Setup of inputs:
- The tests done were different in size: $n = 100$, $n = 200$, $n = 500$, $n = 1000$.
- The number of bins is $n/5$, each of capacity $C = 1000$.
- Five different attributes were assigned to items in an equal probability of $1/5$.
- $s = 0.1$ and $\alpha = 100$.
- The trivial lower bound $\left[ \frac{\sum_{j=1}^{n} w_j}{C} \right]$ is met by randomly splitting the capacity into five items $j$, … $j+4$ that their weights add up to 1000.
- Different orders of the weights have been tested: Ascending, Descending, regular, random.
- For comparison purposes, Random-fit, first-fit and next-fit algorithms have been tested. Apart from the way they choose the bin with respect to $u_{\text{max}}$, they are identical.
4. Results

The results plotted are the best found vectors $Z(x) = (z_1(x), z_2(x))$.

Dominating results are in bold.

- In Table 1, we can see that for $n = 100$, Best-fit and Random-fit and Next-fit performed comparably well in random and descending order of $w_j$.

<table>
<thead>
<tr>
<th>Item order</th>
<th>Best-fit</th>
<th>Random-fit</th>
<th>First-fit</th>
<th>Next-fit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Decreasing $w_j$</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(22, 1.000)</td>
<td>(22, 1.000)</td>
<td>(23, 2.013)</td>
<td>(23, 1.003)</td>
<td></td>
</tr>
<tr>
<td>(21, 2.058)</td>
<td>(21, 1.962)</td>
<td>(22, 2.112)</td>
<td>(22, 1.721)</td>
<td></td>
</tr>
<tr>
<td><strong>Increasing $w_j$</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(25, 1.000)</td>
<td>(25, 1.003)</td>
<td>(26, 1.822)</td>
<td>(25, 1.342)</td>
<td></td>
</tr>
<tr>
<td>(24, 1.118)</td>
<td>(24, 1.132)</td>
<td>(25, 1.987)</td>
<td>(25, 1.830)</td>
<td></td>
</tr>
<tr>
<td><strong>Random</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(22, 1.000)</td>
<td>(22, 1.000)</td>
<td>(22, 1.000)</td>
<td>(22, 1.000)</td>
<td></td>
</tr>
<tr>
<td>(21, 1.184)</td>
<td>(21, 1.962)</td>
<td>(21, 1.545)</td>
<td>(21, 1.400)</td>
<td></td>
</tr>
<tr>
<td><strong>Regular</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(24, 2.130)</td>
<td>(24, 2.000)</td>
<td>(24, 2.430)</td>
<td>(24, 2.510)</td>
<td></td>
</tr>
<tr>
<td>(23, 1.917)</td>
<td>(23, 2.102)</td>
<td>(23, 2.590)</td>
<td>(23, 2.870)</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Vectors $(z_1, z_2)$ for $n = 100$. Dominating results are highlighted in bold.
• Similar results with the instance of \( n = 200 \), as Best-fit and Random-fit and Next-fit gave the best outcomes.

<table>
<thead>
<tr>
<th>Item order</th>
<th>Best-fit</th>
<th>Random-fit</th>
<th>First-fit</th>
<th>Next-fit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Decreasing ( w_j )</strong></td>
<td>(43, 1.000)</td>
<td>(43, 1.000)</td>
<td>(43, 1.003)</td>
<td>(43, 1.030)</td>
</tr>
<tr>
<td></td>
<td>(42, 1.221)</td>
<td>(42, 1.862)</td>
<td>(42, 2.102)</td>
<td>(42, 1.922)</td>
</tr>
<tr>
<td><strong>Increasing ( w_j )</strong></td>
<td>(50, 1.000)</td>
<td>(50, 1.000)</td>
<td>(50, 1.022)</td>
<td>(50, 1.042)</td>
</tr>
<tr>
<td></td>
<td>(49, 1.178)</td>
<td>(49, 1.531)</td>
<td>(49, 1.687)</td>
<td>(49, 1.690)</td>
</tr>
<tr>
<td><strong>Random</strong></td>
<td>(43, 1.000)</td>
<td>(43, 1.000)</td>
<td>(43, 1.000)</td>
<td>(43, 1.000)</td>
</tr>
<tr>
<td></td>
<td>(42, 1.379)</td>
<td>(42, 1.360)</td>
<td>(42, 1.440)</td>
<td>(42, 1.500)</td>
</tr>
<tr>
<td><strong>Regular</strong></td>
<td>(44, 1.300)</td>
<td>(43, 1.504)</td>
<td>(43, 1.430)</td>
<td>(43, 1.320)</td>
</tr>
<tr>
<td></td>
<td>(43, 2.050)</td>
<td>(42, 2.310)</td>
<td>(42, 2.100)</td>
<td>(42, 2.510)</td>
</tr>
</tbody>
</table>

Table 2: Vectors \((z_1, z_2)\) for \( n = 200 \). Dominating results are highlighted in bold

• With \( n = 500 \), Random-fit and Best-fit lead to very close solutions, but Random-fit is superior. Increasing order of weights is obviously not favorable, and random order becomes more favorable than decreasing.

<table>
<thead>
<tr>
<th>Item order</th>
<th>Best-fit</th>
<th>Random-fit</th>
<th>First-fit</th>
<th>Next-fit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Decreasing ( w_j )</strong></td>
<td>(102, 1.000)</td>
<td>(102, 1.000)</td>
<td>(102, 1.000)</td>
<td>(102, 1.050)</td>
</tr>
<tr>
<td></td>
<td>(101, 2.020)</td>
<td>(101, 1.918)</td>
<td>(101, 2.200)</td>
<td>(101, 2.402)</td>
</tr>
<tr>
<td><strong>Increasing ( w_j )</strong></td>
<td>(127, 1.000)</td>
<td>(127, 1.000)</td>
<td>(127, 1.000)</td>
<td>(128, 1.000)</td>
</tr>
<tr>
<td></td>
<td>(126, 1.114)</td>
<td>(126, 1.173)</td>
<td>(126, 1.020)</td>
<td>(127, 1.300)</td>
</tr>
<tr>
<td><strong>Random</strong></td>
<td>(102, 1.000)</td>
<td>(104, 1.000)</td>
<td>(105, 1.000)</td>
<td>(105, 1.000)</td>
</tr>
<tr>
<td></td>
<td>(101, 1.985)</td>
<td>(103, 1.951)</td>
<td>(103, 1.842)</td>
<td>(104, 1.660)</td>
</tr>
<tr>
<td><strong>Regular</strong></td>
<td>(105, 1.000)</td>
<td>(105, 1.000)</td>
<td>(106, 1.000)</td>
<td>(106, 1.010)</td>
</tr>
<tr>
<td></td>
<td>(104, 1.340)</td>
<td>(104, 1.231)</td>
<td>(105, 1.893)</td>
<td>(105, 1.705)</td>
</tr>
</tbody>
</table>

Table 3: Vectors \((z_1, z_2)\) for \( n = 500 \). Dominating results are highlighted in bold
With $n = 1000$, we can obviously see that the order of the items in the input plays an important role in minimizing the number of bins required as the input size becomes greater. Best results are obtained in applying a decreasing order of weights. The random order turned to be weak with such size.

<table>
<thead>
<tr>
<th>Item order</th>
<th>Best-fit</th>
<th>Random-fit</th>
<th>First-fit</th>
<th>Next-fit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Decreasing $w_j$</strong></td>
<td>(203, 1.000) (202, 1.286)</td>
<td>(205, 1.000) (202, 1.614) (201, 1.910)</td>
<td>(205, 1.000) (203, 1.798)</td>
<td>(205, 1.000) (204, 1.400)</td>
</tr>
<tr>
<td><strong>Increasing $w_j$</strong></td>
<td>(250, 1.000) (249, 1.204)</td>
<td>(250, 1.000) (249, 1.642)</td>
<td>(250, 1.000) (249, 1.892)</td>
<td>(250, 1.000) (249, 1.790)</td>
</tr>
<tr>
<td><strong>Random</strong></td>
<td>(205, 1.000) (204, 1.140)</td>
<td>(205, 1.000) (204, 1.970)</td>
<td>(206, 1.000) (205, 1.890)</td>
<td>(205, 1.020) (204, 1.660)</td>
</tr>
<tr>
<td><strong>Regular</strong></td>
<td>(208, 1.000)</td>
<td>(207, 1.020)</td>
<td>(209, 1.200)</td>
<td>(210, 1.000)</td>
</tr>
</tbody>
</table>

*Table 4:* Vectors $(z_1, z_2)$ for $n = 1000$. Dominating results are highlighted in bold
5. Conclusions

In this project, I consider the problem of bin packing under multi-criterion environment: minimizing the bin as well as minimizing the average heterogeneousness of the bins, based on nominal attributes of the items to be packed. The two conflicting goals were formulated as a vector optimization problem. For this scenario, the modified best-fit procedure computed the set of efficient solutions. In a controlled consideration of the heterogeneousness of the bins by integrating a parameter, the selection process advanced in a randomized fashion.

All approximation algorithms tested have not been able to identify the minimal solution for $z_1$ (the number of bins); as best solutions used one more bin than the minimal possible value.

In addition, while random order of weights for inputs of small sizes leads to good results, pre-processing the items in a decreasing order becomes vital to get good results. In conclusion, the results are satisfying, as very close approximations to the set of efficient outcomes have been seen.

6. References:

-Note: these papers were major contributions to this project, but handful other websites were used.

