Simulated Annealing Based Algorithm for the 2D Bin Packing Problem with Impurities

B. Beisiegel[1], J. Kallrath[2], Y. Kochetov[3], A. Rudnev[4]

[1]B2 Software-Technik GmbH
45472 Mülheim an der Ruhr Germany

[2]Am Mahlstein 8
67273 Weisenheim am Berg Germany

[3,4]Sobolev Institute of Mathematics
630090 Novosibirsk Russia

1 Introduction

In the classical 2D rectangular bin packing problem [2] we are given a set of two dimensional rectangular items and an unlimited number of identical large rectangular bins. We need to place the items into a minimal number of bins. The orientation of the items is parallel to the bounds of the bins. Overlaps of items are not allowed.

In this paper we consider a more complicated real-world problem originating in the steel industry. The bins are inhomogeneous sheets with impurities. We assume that each impure area is rectangle. For each bin we are given a set of impurities, size, and location of each impurity into the bin. As a consequence now the bins are not identical anymore and the number of bins is finite. Moreover, we introduce the linear order on the set of bins. First of all, we have to use the first bin. If we need additional bins we use the second bin and so on. The items have the attribute whether they can be located in the area with impurities. The goal is to find solutions with a minimal number of bins.

For solving this NP-hard problem we have developed a tailored Simulated Annealing algorithm (SA). Feasible solutions are presented by the directed root tree

encoding scheme. It has linear decoding time if the maximal number of impurities per bin is a constant. The initial solution is built by a greedy algorithm. It is a polynomial time heuristic which allows us to start SA with low temperature. The SA algorithm packs the items separately in each bin. It uses two types of neighborhoods. The first one changes the structure of the directed root tree. The second one swaps two items in the vertices of the tree. Computational experiments show that the algorithm produces feasible solutions with small deviations from the lower bound within a few minutes.

2 Representation of solutions

There are many encoding schemes for the 2D Rectangular Packing problems [3]. In this paper we use the oriented tree representation.

2.1 Encoding

Let us consider a feasible solution and show how to generate an oriented tree for each bin.

Definition 1. A feasible solution is called compacted if there is no item that can be shift left or bottom from its original position with other items fixed.

In *Figure* 1 we can see compacted and non-compacted feasible solutions. Further we consider the compacted solutions only.

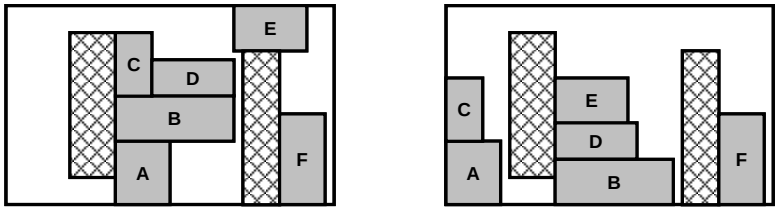

Fig. 1. An example of non-compacted (left) and compacted (right) placements for a bin

Definition 2. The item *B* is in the horizontal relation to the item *A* if

1. *B* is to the right of *A.*
2. The projections of *A* and *B* on the vertical axis are overlapped.
3. *A* and *B* are either adjunct or are divided by impurities only.

The oriented tree is built as follows. The set of nodes is the set of items in the bin with an additional node representing the root of the tree. The root corresponds to a dummy item placed on the left bound of the bin. The height of this item is the height of the bin. Node *A* is the parent of node *B* if item *B* is in horizontal relation to item *A*. If *B* is in horizontal relation with several items then the lowest one is the parent for *B*. In *Figure* 2 an example of oriented tree is presented.

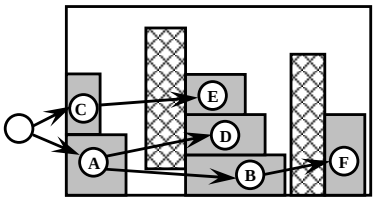

Fig. 2. The oriented root tree for the packing

2.2 Decoding

For a given oriented tree we generate packing by the following way. The root dummy rectangle is placed on the left side of the bin. According to the depth-first rule for the tree, we place items one by one in such a way that the left side of each item and the right side of its parent are on the same vertical line. The *y*-coordinate is defined by the previous packed items. Roughly speaking, we "drop" the current item on the right of its parent. If the item overlaps an impurity and cannot use it we consider two new positions for the item: above and right of the impurity. In the first case the item is shifted upwards and put on the impurity. In the second case the item is shifted to the right and put after the impurity. In the last case the new vertical position is defined by the previous packed items again. If the new position is overlapped with other impurities we put the item on the impurities. So we have two positions for the item. The lowest one is selected for packing. In *Figure* 3 we illustrate the idea of this algorithm. The time complexity of the algorithm is linear if the number of impurities per bin is a constant.

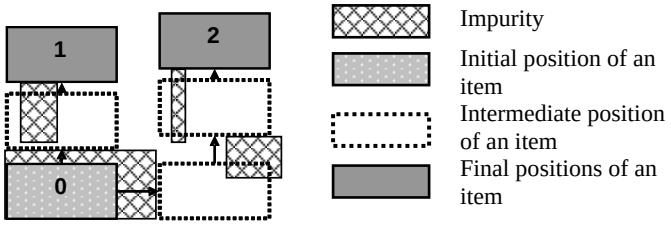

Fig. 3. Packing of the item

3 Neighborhood

Let us consider an oriented root tree. Each node of the tree corresponds to an item. The set of neighbors for the tree consists of two parts. In the first part we include all trees, which can be obtained by moving a leaf to another position. It gives us $O(n^2)$ neighbors, n is number of nodes. The second part of this set contains $(n-1)(n-2)/2$ neighbors. They are obtained by swapping items for the non root nodes. This neighborhood has the following property. We can reach an arbitrary oriented root tree from an arbitrary starting tree in a finite number of steps by moving from a tree to a neighboring one.

4 Simulated annealing

We apply the Simulated Annealing algorithm [4] for each bin in parallel. The algorithm is shown in *Figure* 4. The initial solution, i.e. initial set of trees, can be selected at random but we develop a polynomial time heuristic to get near optimal solutions. It allows us to start SA with low temperature. The objective function $F(T)$ which we wish to minimize is the used part of the bin with penalties. We apply the penalties for infeasible solutions when some items go out from the bin. For fixed temperature $t > 0$ we apply the random search during prescribed number of iterations for all bins (*Step* 3.1) and decrease the temperature (*Step* 3.2, $r < 1$). If the current solution is feasible we apply the Unload algorithm to change the set of items into the bins. This algorithm tries to unload the last bins by moving or swapping "large" items.

1. Find initial tree for every bin
2. Set initial temperature $t > 0$
3. Repeat until stopping criteria is true
   - 3.1. Repeat loop given number of times for every bin
     - 3.1.1. Chose random tree T' from neighborhood $N(T)$
     - 3.1.2. Set $\Delta = F(T') - F(T)$
     - 3.1.3. If $\Delta \le 0$, then $T = T'$
     - 3.1.4. If $\Delta > 0$, then $T = T'$ with probability $e^{-\Delta/t}$
   - 3.2. Set $t = rt$
   - 3.3. Execute *Unload* algorithm
4. Return set of trees

Fig. 4. Simulated annealing

5 The initial solution

Let us remove all impurities of the bins and compute the lower bound N_{LB} for the minimal number of bins [1]. Put the items in decreasing order of their areas and declare all bins are closed. Open the first N_{LB} bins. We place the items into the bins one by one: the first item into the first bin, the second item into the second bin and so on. At the $(N_{LB} + 1)$ step we put the current item into the N_{LB} bin, the next one we put into the $(N_{LB} – 1)$ bin and so on. We try to distribute "large" items through the N_{LB} bins. If we spend all items then we have an optimal solution. Otherwise we repeat this approach for the unpacked items. In *Figure* 5 we show the idea of this algorithm. It is polynomial time heuristic where we generate oriented root trees for all bins step by step.

1. Sort items list L by non-increasing area
2. Close all bins to use
3. Find the lower bound N_{LB} of optimal bins number for list L
4. Open N_{LB} empty bins to use
5. Place items from L into the open bins in prescribed order
6. Remove placed items from list L
7. If L is empty then stop
8. If set of closed bins is not empty, then go to *step* 3, else place remained items in additional infinitely large bin

Fig. 5. Heuristic for initial solution

6 Unload algorithm

This algorithm is used in our SA after decreasing the temperature in *Step* 3.3. It tries to select small items for the last bins and next to unload them. The algorithm uses two operations:

1. Swap operation. It swaps "small" items in the current bin by "large" items in the last bins without violation of feasibility.

2. Move operation. It moves the items from the current bin into the previous bins.

The algorithm consists of two stages. At the first stage we use swap operation for every bin starting from the end of the bin list. At the second stage we use move operation starting from the beginning of the list. The time complexity of the algorithm is $O(n^3)$.

7 Experimental results

The developed algorithm is coded in DELPHI environment and tested on real world and random generated instances. Our real world instances have small dimension, $n \leq 30$. The algorithm finds optimal solutions for all of them. To study the algorithm for large dimension we generate identical bins 200×300 and cut them to get the set of items. So, we have optimal solution if the impurities are absent, otherwise we have a lower bound only. The impurities are generated at random for every bin. The total number of impurities is $n/5$. The sizes of impurities are generated at random as items. Number of items, which can use the impurities, is selected as $n/5$. Computational results for three classes of instances are presented in *Table* 1. In class *Dl* every bin contains *l* items in average. For $n = 100$ the lower bound N_{LB} is quite far from the optimal solution and relative errors are large. For large scale instances the heuristic solutions is not too far from the lower bound and the relative errors are small.

Table 1. Deviation from the lower bound, %

	$n = 100$	$n = 150$	$n = 200$	$n = 300$	$n = 500$	$n = 700$	$n = 1000$
*D*3	**24**	**24**	**25**	**12**	**8.4**	**6.8**	**4**
*D*5	**35**	**30**	**27**	**13**	**8**	**6.4**	**6**
*D*10	**30**	**26**	**30**	**13**	**10**	**5.7**	**8**

References

[1] Boschetti M. A., Mingozzi A. (2003) The two-dimensional finite bin packing problem. Part I: New lower bounds for the oriented case. 4OR, vol 1, 1, pp 27–42.

[2] Dyckhoff H. (1990) A typology of cutting and packing problems. European J. Oper. Res. vol 44, 2, pp 145–159.

[3] Guo P.–N., Cheng C.–K., Yoshimura T. (1999) An O-tree representation of non-slicing floorplan and its applications. Proc. DAC, pp. 268–273.

[4] Johnson D. S., Aragon C. R., McGeoch L. A., Schevon C. (1989) Optimization by simulated annealing: An experimental evaluation, part I (graph partitioning). Operations Research, vol 37, 6, pp 865–892.