MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

New Heuristic and Interactive Approaches to 2D Rectangular Strip Packing

N. Lesh, J. Marks, A. McMahon, M. Mitzenmacher

TR2005-113   September 2005

Abstract

In this paper, we consider the two-dimensional rectangular strip packing problem. A standard simple heutistic, Bottom-Left-Decreasing (BLD), has been shown to perform quite well in practice. We introduce and demonstrate the effectiveness of BLD, a stochastic search variation of BLD. While BLD places the rectangles in decreasing order of heigh, width, area and perimeter, BLD successively tries random orderings, chosen from a distribution determined by their Kendall-tau distance from one of these fixed orderings. Our experiments on benchmark problems show that BLD produces significantly better packings than BLD after only 1 minute of computation. Furthermore, we show that BLD outperforms recently reported methaheuristics. Furthermore, we observe that people seem able to reason about packing problems extremely well. We incorporate our new algorithms in an interactive system that combines the advantages of computer speed and human reasoning. Using the interactive system, we are able to quickly produce signiticantly better solutions than BLD by itself.

Journal of Experimental Algorithmics, JEA 2005

New Heuristic and Interactive Approaches to 2D Rectangular Strip Packing

N. Lesh[1], J. Marks[1], A. McMahon[2], M. Mitzenmacher[2],

[1] Mitsubishi Electric Research Laboratories, 201 Broadway, Cambridge, MA, 02139

lesh@merl.com, marks@merl.com

[2] University of Miami, Florida.*adam@math.miami.edu

[3] Harvard University, Computer Science Department

michaelm@eecs.harvard.edu †

Abstract

In this paper, we consider the two-dimensional rectangular strip packing problem. A standard simple heuristic, Bottom-Left-Decreasing (BLD), has been shown to perform quite well in practice. We introduce and demonstrate the effectiveness of BLD*, a stochastic search variation of BLD. While BLD places the rectangles in decreasing order of height, width, area, and perimeter, BLD* successively tries random orderings, chosen from a distribution determined by their Kendall-tau distance from one of these fixed orderings. Our experiments on benchmark problems show that BLD* produces significantly better packings than BLD after only 1 minute of computation. Furthermore, we show that BLD* outperforms recently reported metaheuristics.

Furthermore, we observe that people seem able to reason about packing problems extremely well. We incorporate our new algorithms in an interactive system that combines the advantages of computer speed and human reasoning. Using the interactive system, we are able to quickly produce significantly better solutions than BLD* by itself.

Subject classification: Cutting stock/trim: 2D rectangular strip packing. Artificial intelligence: interactive methods.

*This work done while visiting Mitsubishi Electric Research Laboratories.

†Supported in part by NSF CAREER Grant CCR-9983832 and an Alfred P. Sloan Research Fellowship. This work was done while visiting Mitsubishi Electric Research Laboratories.

1 Introduction

Packing problems involve constructing an arrangement of items that minimizes the total space required by the arrangement. In this paper, we specifically consider the two-dimensional (2D) rectangular strip packing problem. The input is a list of n rectangles with their dimensions and a target width W. The goal is to pack the rectangles without overlap into a single rectangle of width W and minimum height H. We further restrict ourselves to the orthogonal variation, where rectangles must be placed parallel to the horizontal and vertical axes. We consider two variations: *fixed orientation* in which the rectangles cannot be rotated, and *variable orientation* in which they can be rotated by 90 degrees. Further, for all our test cases, all dimensions are integers. Like most packing problems, 2D rectangular strip packing (even with these restrictions) is NP-hard.

A common method for packing rectangles is to take an ordered list of rectangles and greedily place them one by one. Perhaps the best studied and most effective such heuristic for the fixed-orientation variation is the Bottom-Left (BL) heuristic, where rectangles are sequentially placed first as close to the bottom and then as far to the left as they can fit. For some problems, BL cannot find the optimal packing under any ordering of the rectangles [2, 4], nor does it perform well in practice when applied to random orderings. However, a very successful approach is to apply BL to the rectangles ordered by decreasing height, width, perimeter, and area and return the best of the four packings that result [11]. We refer to this scheme as Bottom-Left-Decreasing (BLD).

A natural alternative approach would be to find good orderings of the rectangles for BL or other similar heuristics, using standard search techniques such as simulated annealing, genetic algorithms, or tabu search. Despite significant efforts in this area, the search space has not generally proven amenable to such search techniques; for more details see the thesis of Hopper [11]. Recently, genetic algorithms and tabu search, using a different search space, have proven more successful [14].

In this paper, we present a variation of the BLD heuristic called BLD* that considers successive random perturbations of the original four decreasing orderings. We also present an apparently novel generalization of BL, and consequently of BLD and BLD*, for the variable orientation case. Our experiments on both benchmark and randomly generated problems show that BLD* substantially outperforms BLD, as well as BL applied to randomly chosen orderings. For example, for the benchmarks taken from Hopper [11] in the case of fixed orientation, BLD* reduces the packing height from an average of 9.4% over optimal by BLD to about

4.9% over optimal after just one minute. These are the best published results for these benchmarks that we are aware of. We note that improvements of even 1% can be very valuable for industrial applications of this problem, such as glass and steel cutting.

This work was done as part of Human-Guided Search (HuGS) project, an ongoing effort to develop interactive optimization systems [16]. Determining how people can effectively interact with powerful stochastic search algorithms is important because it leverages people's abilities in areas in which they currently outperform computers, such as visual perception and strategic assessment. Furthermore, involving people in the process of optimization can help them understand and trust the produced solutions, as well as modify them on the fly if the need arises. Human-guided search makes particular sense in the context of cutting and packing problems, where humans can often find better solutions than the best current algorithms. Our interactive optimization system is designed to allow people to effectively use the computer's speed and preprogrammed algorithms to more quickly find good solutions than they could themselves find alone, especially for larger problems.

For the 2D packing problem, we explored people's ability to guide our BLD* heuristic. We found that people can reason about this problem to make use of the computer's power extremely well. People can identify particularly well-packed subregions of a given packing and then focus a search algorithm on improving the other parts. People can also devise multi-step repairs to a packing problem to reduce unused space, often producing packings that could not be found by the BL heuristic for any ordering of rectangles. Our experiments on large benchmarks show that interactive use of BLD* can produce solutions 1% closer to optimal in about 20 minutes than BLD* produces on its own in 2 hours. Thus, 2D packing seems to be a problem for which people and computers can currently produce better results together than either can alone.

2 Background

Packing problems in general are important in manufacturing settings; for example, one might need n specific rectangular pieces of glass to put together a certain piece of furniture, and the goal is to cut those pieces from the minimum-height fixed-width piece of glass. The more general version of the problem allows for irregular shapes, which is required for certain manufacturing problems such as clothing production. However, the rectangular case has many industrial applications [11].

The 2D rectangular strip packing problem has been the subject of a great deal of research, both by the theory community and the operations-research community [6, 8, 21]. One focus has been on approximation algorithms. The Bottom-Left heuristic has been shown to be a 3-approximation when the rectangles are sorted by decreasing width; that is, the resulting height is *always* within a factor of 3 of optimal [2]. The Bottom-Left heuristic is not within a factor of k of optimal for any fixed constant k when the rectangles are sorted by decreasing height. Other approximation results include algorithms that give an asymptotic 5/4-approximation [3], an absolute 5/2-approximation [26], and an absolute 2-approximation algorithm [27]. Recently, Kenyon and Remilia have developed an asymptotic fully polynomial approximation scheme [15].

Another focus has been on heuristics that lead to good solutions in practice. There are two main lines of research in this area. One line considers simple heuristics such as BLD. Another line focuses on local search methods that take substantially more time but have the potential for better solutions: genetic algorithms, tabu search, hill-climbing, and simulated annealing. The recent thesis of Hopper provides substantial detail of the work in this area [11, 12]. We compare BLD* with more recent work by Iori *et. al.*, who provide results for their novel tabu algorithm, genetic algorithm, and hybrid algorithm on a wide range of instances from the literature [14].

Exact algorithms have received relatively little consideration. We have developed an exhaustive branch-and-bound algorithm which generally solves problem instances with fewer than 30 rectangles for which a perfect packing, i.e., one with no empty space, exists [19, 20]. Other recent work includes that of Fekete and Schepers, who suggest branch-and-bound techniques for bin and strip packing problems [9, 10]. They test their general approach on the knapsack problem, and not strip packing problems, and hence we are unable to provide a direct comparison Other similar work has also been done simultaneously by Korf [18] and by Martello, Monaci, and Vigo [23], who use branch-and-bound techniques to determine optimal packings.

The fixed-orientation problem has received much more attention than the variable-orientation problem, although some genetic-algorithm approaches have allowed reorientation as one of the mutation operations (e.g., [13, 7]). We are unaware of any previous work on adapting the BL algorithm for variable orientations (as we describe below).

2.1 The Bottom-Left Heuristic

The *Bottom-Left* (BL) heuristic, introduced in [2], is perhaps the most widely used heuristic for placing rectangles. We think of the points in the strip to be packed as being ordered lexicographically, so that point A lies before point B if A is below B or, if A and B have the same height and A is to the left of B. Given a permutation of the rectangles, the Bottom-Left heuristic places the rectangles one by one, with the lower left corner of each being placed at the first point in the lexicographic ordering where it will fit. There are natural algorithms that require $O(n^3)$ time in the worst case for the problem; Chazelle devised an algorithm that requires $O(n^2)$ time and $O(n)$ space in the worst case [5]. In practice the algorithm runs much more quickly, since a rectangle can usually be placed in one of the first open spots available. When all rectangle dimensions are integers, this can be efficiently exploited. Hopper discusses efficient implementations of this heuristic in her thesis work [11].

Perhaps the most natural permutation to choose for the Bottom-Left heuristic is to order the rectangles by decreasing height. This ensures that at the end of the process rectangles of small height, which therefore affect the upper boundary less, are being placed. It has long been known that this heuristic performs very well in practice [6]. It is also natural to try sorting by decreasing width, area, and perimeter, and take the best of the four solutions. While usually decreasing height is best, in some instances these other heuristics perform better. We refer to the algorithm that takes the best packing produced by these four orderings as BLD.

2.2 Benchmarks

In this paper, we evaluate our algorithm and interactive system on both a set of structured benchmarks with known optimal packings and on randomly generated test instances without known optimal packings. The former is a set of benchmarks recently developed by Hopper. All instances in this benchmark have perfect packings of dimension 200 by 200. The instances are derived by recursively splitting the initial large rectangle randomly into smaller rectangles; for more details, see [11]. This benchmark set contains problems with size ranging from 17 to 197 rectangles. We use the non-guillotinable instances from this set, collections N1 (17 rectangles) through N7 (197 rectangles), each containing 5 problem instances.

The strengths of this benchmark are that a wide range of algorithms have been tested against it, providing meaningful comparisons; problem sizes vary from the small to the very large; and the optimal solution is known by construction. The benchmark problems, however, are highly structured, and because all instances have perfect packings, they yield limited insight on the performance of algorithms when perfect packings are not available. We note that we have developed an exhaustive branch-and-bound algorithm which can quickly solve the N1-N3 problem instances [19, 20]; therefore we tend to focus on the N4-N7 collections to evaluate our heuristic methods.

After developing our algorithms, we ran them on other instances available in the literature to compare it to the recently reported results of [14]. These instances include the "ht" benchmarks by Hopper and Turton and the "gcut" examples available at the Operations Research Library (http://mscmga.ms.ic.ac.uk/info.html). We also ran BLD* on 10 classes of randomly-generated problems from the literature, described in [14]. The target width and the range of width and heights for the rectangles varies by class. The specific instances are available for download at http://www.or.deis.unibo.it/ORinstances/2BP/. Each class has problems of five sizes, ranging from 20 to 100 rectangles, and 10 instances per size.

3 Orienting rectangles

We modified the BL and BLD heuristics for the variable orientation problem. The modified heuristic again places rectangles one at a time according to some permutation, but now it considers both orientations when placing each rectangle. For each orientation, the placement is determined by the first point in the bottom-left lexicographic ordering where the rectangle will fit, following the Bottom-Left paradigm. Given these two possible placements, the algorithm must decide between them. We experimented with three decision rules. The first rule computes where the bottom-left corner would be positioned by both orientations, and chooses the orientation in which the bottom-left corner is earliest in the lexicographic ordering. The second and third rules are the same, except that they compare where the center and top-right corner of the rectangle is positioned, respectively. In the case of ties (which turn out to be very rare), we choose randomly between the two orientations.

Because the rectangles can be reoriented, it does not make sense to order them by decreasing width or height. Instead, we consider ordering the rectangles in decreasing order of the length of their minimum or maximum dimension, as well as in decreasing order by area and perimeter.

We ran experiments to evaluate the possible combinations of ordering methods and decision rules for the rectangles of the 20 instances in the N4 to N7 collections

sort by				choose by		
min	max	area	perim.	center	bottom left	top right
yes	no	no	no	5.62	15.38	4.43
no	yes	no	no	5.58	6.40	5.98
no	no	yes	no	5.15	8.08	4.60
no	no	no	yes	5.23	5.58	4.70
no	no	yes	yes	5.00	6.28	4.40
no	yes	no	yes	4.83	5.30	4.70
no	yes	yes	no	4.85	5.82	4.53
no	yes	yes	yes	4.83	5.83	4.33
yes	no	no	yes	4.68	5.58	4.33
yes	no	yes	no	4.85	8.08	4.43
yes	no	yes	yes	4.78	6.28	4.23
yes	yes	no	no	4.73	6.47	4.38
yes	yes	no	yes	4.45	5.50	4.33
yes	yes	yes	no	4.68	6.50	4.43
yes	yes	yes	yes	4.60	6.08	4.23

Table 1: Results of BLD modified for variable orientation. Each number is an average of the 20 problem instances in the N4-N7 benchmark collections.

using this variation of BLD. Table 1 shows the average percent over optimal from the various combinations. If more than one ordering is used, then we took the best packing produced from all of the relevant orderings. The results indicate that the most effective decision rule is to chose the orientation that places the top-right corner as early as possible in the lexicographic ordering. The most effective ordering is to sort the rectangles by their minimum dimension.

Our current understanding of why sorting by minimum dimension is better than by maximum dimension when the rectangles are reorientable is best expressed by an example: a 50×1 rectangle can be oriented so as to only add at most 1 to the height, and so it is reasonable to place this rectangle toward the end. Similarly, using the top-right corner to decide orientation most closely approximates the objective function being used to evaluate an entire packing.

4 Improving the BLD Heuristic

A natural way to improve the BLD heuristic is to apply BL to other permutation orders. At the expense of more time, more orders besides the four suggested can be tried to attempt to improve the best solution found. One standard technique would be *random-repeat*: permutations are repeatedly chosen uniformly at random, and the best solution found within the desired time bound is used. Random permutations, however, are known to perform poorly [11]. We tried BL on random permutations on the N4 through N7 benchmark collections. After 20 minutes, the average height of the best solution found was 9.6% over the optimal compared to the 6.4% over optimal generated by the BLD heuristic in less than a second. (All times reported in this paper are for experiments run on a Linux machine with a 2000 MhZ Pentium processor running Java code.)

Instead, we suggest the following stochastic variation of BLD, which we call BLD*. Our intuition for why BLD performs so much better than BL with random-repeat is that the decreasing sorted orders save smaller rectangles for the end. Therefore, BLD* chooses random permutations that are "near" the decreasing sorted orders used by BLD, as they will also have this property. There are many possible ways of doing this; indeed, there is a deep theory of distance metrics for rank orderings [22]. BLD* uses the following simple approach: start with a fixed order (say decreasing height), and generate random permutations from this order as follows. Items are selected in order one at a time. For each selection, BLD* goes down the list of previously unaccepted items in order, accepting each item with probability p, until an item is accepted. If the last item is reached and not selected, then we restart at the beginning of the list, again taking an item with probability p. After an item is accepted, the next item is selected, starting again from the beginning of the list of unaccepted items. (See Figure 1.) More formally, choose the ith item as follows. Let q initially be 0. Repeat the following: with probability p, terminate and output the $(q+1)$st unselected item from the original sorted list; otherwise increment q by 1 modulo $n-i+1$. This approach generates permutations that are near decreasing sorted order, preserving the intuition behind the heuristic, while allowing a large number of variations to be tried.

The probability starting from some fixed ordering x of obtaining some other ordering y is proportional to $(1-p)^{Ken(x,y)}$, where $Ken(x,y)$ is the Kendall-tau distance between the two permutations. This is also known as bubble-sort distance, because it counts the number of swaps bubble-sort would make transforming x to y.

Our current version of BLD* first tries the four orders used by BLD and then

```
Algorithm BLD*:

1. Let $p \leftarrow 0.5$.
2. Repeat until halted:
   (a) Let $R_1, R_2, \ldots, R_n$ be the $n$ rectangles in order of decreasing height.
   (b) For $j = 1$ to $n$ do:
       i. Let $q \leftarrow 0$.
       ii. Repeat until a rectangle is selected:
           A. Choose $x$ uniformly from $[0, 1]$.
           B. If $x < p$, select rectangle $R_{q+1}$.
           C. If $x > p$, $q \leftarrow q + 1 \bmod n - j + 1$.
       iii. Place the selected rectangle according to the Bottom Left rule.
       iv. Remove the selected rectangle from the list, leaving the list
           $R_1, R_2, \ldots, R_{n-j}$ of remaining rectangles in sorted order.
   (c) Save if the solution is the best seen so far.
3. Return the best solution.
```

Figure 1: A pseudocode description of BLD*. Different orderings, different p values, and different placement rules could be used.

	Fixed orientation BLD* score after t seconds							
problem	0(BLD)	30	60	120	300	600	1800	3600
Hopper N1, size=17	16.4	6.0	6.0	6.0	5.6	5.1	5.0	4.5
Hopper N2, size=25	12.2	6.6	6.4	5.8	5.7	5.4	4.8	4.7
Hopper N3, size=29	12.4	6.1	6.0	6.0	5.6	5.1	5.0	4.6
Hopper N4, size=49	9.0	5.3	5.1	4.9	4.4	4.4	4.0	3.9
Hopper N5, size=73	7.6	5.0	4.6	4.4	4.4	4.3	4.0	4.0
Hopper N6, size=97	5.4	4.3	4.0	3.9	3.8	3.5	3.4	3.0
Hopper N7, size=197	3.0	2.8	2.3	2.3	2.2	1.9	1.8	1.8
Hopper N1-N7	9.4	5.3	4.9	4.8	4.5	4.2	4.0	3.8

Table 2: Average results of BLD* on Hopper benchmarks with fixed orientation.

permutes each of these orders in round-robin fashion.

	Variable orientation BLD* score after t seconds							
problem	0(BLD)	30	60	120	300	600	1800	3600
Hopper N1, size=17	12.5	4.6	4.6	3.8	3.8	3.7	3.7	3.5
Hopper N2, size=25	9.7	4.7	4.6	4.1	4.0	3.9	3.6	3.3
Hopper N3, size=29	10.0	4.4	4.0	3.9	3.7	3.5	3.3	3.2
Hopper N4, size=49	6.1	3.4	3.2	3.1	3.0	2.9	2.9	2.6
Hopper N5, size=73	6.6	3.0	3.0	2.8	2.8	2.6	2.1	2.1
Hopper N6, size=97	3.6	2.3	2.3	2.2	1.9	1.9	1.9	1.9
Hopper N7, size=197	1.4	1.0	1.0	1.0	1.0	1.0	0.9	0.9
Hopper N1-N7	7.1	3.6	3.2	3.0	2.9	2.8	2.6	2.5

Table 3: Average results of BLD* on Hopper benchmarks with variable orientation.

4.1 Experimental Results

We first ran BLD* on the Hopper benchmarks N1-N7 to quantify how much improvement BLD* provides over BLD. We used $p = 0.5$ based on a small amount of preliminary investigation of different values. For the fixed orientation problem, we used all four orderings (height, width, area, and perimeter). The Hopper instances are given in a format that specifies the dimension of each rectangle when the orientation is fixed in such a way that a perfect packing is possible. For the variable orientation problem, we used our modified version of BLD* with using only the minimum-dimension ordering and the top-right decision rule.

The results are shown in Tables 2 and 3. The table shows the results of running BLD and the results of running BLD* at various time increments. The numbers represent the percentage over the optimal width of 200. For all cases, BLD* dramatically improves solutions over BLD even with just one minute of computation. It continues to improve steadily, though improvements taper off with time. Note that BLD performs poorly on small instances and so the improvements for BLD* are more substantial.

We also ran experiments on the N4-N7 collections to measure how many permutations BLD* considered before improving upon the best solution by BLD; it considered an average of only 15.25 permutations to do so.

Tables 4 and 5 compare the performance of BLD* against those reported by [14] for the fixed orientation case (the only case they consider). Their work presents an algorithm for computing lower bounds for these problems, as well as the results of running three algorithms (a tabu search, a genetic algorithm, and

Problem			Iori *et. al.*		BLD*					
				est. best	best result after t seconds computation					
C	n	W	LB	score	60	120	300	600	1800	3600
1	20	10	60.3	**61.1**	61.5	61.4	61.3	61.3	61.3	61.3
1	40	10	121.6	**121.8**	122.1	122.1	122.0	122.0	122.0	122.0
1	60	10	187.4	189.0	189.1	189.0	189.0	189.0	**188.9**	188.9
1	80	10	262.2	262.8	262.9	262.9	262.9	262.9	262.9	**262.8**
1	100	10	304.4	305.5	305.9	305.8	305.8	305.7	305.6	**305.5**
2	20	30	19.7	19.9	20.0	19.9	19.9	19.9	**19.8**	19.8
2	40	30	39.1	39.9	**39.5**	39.5	39.3	39.1	39.1	39.1
2	60	30	60.1	61.6	**61.0**	61.0	60.9	60.9	60.9	60.6
2	80	30	83.2	84.6	**84.0**	83.9	83.6	83.6	83.6	83.6
2	100	30	100.5	101.8	**101.1**	101.1	101.0	101.0	100.8	100.8
3	20	40	157.4	164.7	**164.6**	164.6	164.3	164.3	164.2	164.2
3	40	40	328.8	337.9	**336.6**	335.4	335.1	335.1	334.8	334.8
3	60	40	500.0	515.9	**513.0**	512.4	511.3	511.0	510.4	510.1
3	80	40	701.7	717.4	**716.9**	716.5	715.8	713.6	713.5	713.0
3	100	40	832.7	847.7	847.8	**846.7**	845.9	845.1	844.4	844.1
4	20	100	61.4	65.6	**64.5**	64.4	64.3	64.2	64.1	63.9
4	40	100	123.9	131.2	**129.6**	129.3	128.8	128.6	128.2	128.1
4	60	100	193.0	202.1	**201.0**	201.0	200.8	200.6	200.1	199.9
4	80	100	267.2	278.6	278.8	**278.4**	277.6	277.3	277.1	276.6
4	100	100	322.0	332.2	334.6	334.2	333.8	333.6	332.4	**332.1**

Table 4: Comparison of BLD* on randomly generated problems, with fixed orientation, proposed by Martello and Vigo. Results averaged over 10 instances.

a hybrid algorithm) for five minutes of CPU time on a Pentium III 800 MHz machine. The hybrid algorithm generally performed the best, but not in all cases. In [14], the scores are reported as a *percent gap*, defined as $(s - LB)/s$ where s is the score obtained by the algorithm on the problem and LB is the lower bound computed by their algorithm. We felt that reporting absolute scores would simplify future comparisons, especially since improved lower bounds would decrease the percent gap for a given packing score achieved by a packing algorithm. We thus estimated the absolute scores of the algorithms in [14] by finding the score that would produce the reported percent gap. The number is approximate because the reported lower bounds and percent gaps are averaged over 10 instances.

The first three columns of the tables indicate the problem class, the number of rectangles, and the target width. The next two columns give the lower bound

Problem			Iori *et. al.*		BLD*					
				est. best	best result after t seconds computation					
C	n	W	LB	score	60	120	300	600	1800	3600
5	20	100	512.2	536.2	536.6	536.3	**535.7**	535.5	534.9	534.8
5	40	100	1053.8	1085.8	**1084.2**	1083.2	1081.5	1079.9	1078.5	1076.7
5	60	100	1614.0	1667.9	**1656.9**	1655.2	1654.1	1651.5	1651.2	1651.1
5	80	100	2268.4	2304.7	**2303.2**	2302.6	2300.0	2298.6	2297.6	2297.2
5	100	100	2617.4	2695.9	**2680.8**	2675.6	2672.4	2670.5	2670.1	2669.5
6	20	300	159.9	174.9	**172.5**	172.3	171.6	171.3	170.5	170.3
6	40	300	323.5	346.0	**343.8**	342.7	342.2	340.5	339.4	339.2
6	60	300	505.1	530.9	536.6	535.3	533.8	531.8	531.0	**529.7**
6	80	300	699.7	**732.2**	743.9	740.1	737.4	736.3	734.8	733.9
6	100	100	843.8	**874.9**	890.6	888.4	884.9	883.7	882.7	882.1
7	20	100	490.4	502.7	**501.9**	501.9	501.9	501.9	501.9	501.9
7	40	100	1049.7	1060.3	**1059.4**	1059.4	1059.0	1059.0	1059.0	1059.0
7	60	100	1515.9	**1529.5**	1530.4	1530.4	1530.4	1529.8	1529.7	1529.7
7	80	100	2206.1	2224.4	**2223.7**	2223.6	2223.5	2223.0	2222.4	2222.2
7	100	100	2627.0	**2646.4**	2648.4	2647.1	2646.7	2646.6	2646.5	2646.5
8	20	100	434.6	467.6	**466.0**	465.8	463.6	462.8	461.9	461.6
8	40	100	922.0	979.3	**978.6**	977.1	973.4	971.2	968.7	967.8
8	60	100	1360.9	1436.0	1437.5	**1433.3**	1432.8	1429.9	1426.7	1425.1
8	80	100	1909.3	2007.2	2014.9	2010.9	**2005.9**	1997.8	1994.0	1992.0
8	100	100	2362.8	2477.2	2491.4	2486.5	2477.4	**2473.0**	2468.5	2466.7
9	20	100	1106.8	1119.2	**1106.8**	1106.8	1106.8	1106.8	1106.8	1106.8
9	40	100	2189.2	2231.2	**2190.9**	2190.7	2190.7	2190.7	2190.6	2190.6
9	60	100	3410.4	3410.4	$\underline{3410.4}$	3410.4	3410.4	3410.4	3410.4	3410.4
9	80	100	4578.6	4873.9	**4588.1**	4588.1	4588.1	4588.1	4588.1	4588.1
9	100	100	5430.5	5718.9	**5434.9**	5434.9	5434.9	5434.9	5434.9	5434.9
10	20	100	337.8	355.1	**352.0**	351.7	351.5	351.3	351.3	351.1
10	40	100	642.8	674.2	**670.1**	669.0	667.9	667.1	666.0	665.7
10	60	100	911.1	953.6	**946.9**	945.0	943.0	941.5	940.7	940.1
10	80	100	1177.6	1229.6	**1226.1**	1223.6	1221.4	1221.1	1218.7	1217.8
10	100	100	1476.5	1537.5	**1536.0**	1532.8	1529.6	1528.4	1526.6	1525.3

Table 5: Comparison of BLD* on randomly generated problems, with fixed orientation, proposed by Berkey and Wang. Results averaged over 10 instances.

and the best of the three results from [14]. The next five columns show the result of running BLD* for 60, 120, 300, 600, 1800, and 3600 seconds of wall-clock

Problem			Iori *et. al.*		BLD*					
				best	best result after t seconds computation					
name	n	W	LB	score	60	120	300	600	1800	3600
ht13	73	60	90.0	94.0	**92.0**	92.0	92.0	92.0	92.0	92.0
ht14	73	60	90.0	93.0	**92.0**	92.0	92.0	92.0	92.0	92.0
ht15	73	60	90.0	94.0	**92.0**	92.0	92.0	92.0	92.0	92.0
ht16	97	80	120.0	126.0	**123.0**	123.0	123.0	123.0	122.0	122.0
ht17	97	80	120.0	124.0	**123.0**	123.0	122.0	122.0	122.0	122.0
ht18	97	80	120.0	124.0	**123.0**	123.0	123.0	123.0	122.0	122.0
gcut1	10	250	1016.0	1016.0	1016.0	1016.0	1016.0	1016.0	1016.0	1016.0
gcut2	20	250	1133.0	1207.0	1211.0	1211.0	1207.0	**1205.0**	1204.0	1195.0
gcut3	30	250	1803.0	1803.0	1803.0	1803.0	1803.0	1803.0	1803.0	1803.0
gcut4	50	250	2934.0	3130.0	**3072.0**	3072.0	3072.0	3072.0	3063.0	3054.0
gcut5	10	500	1172.0	1273.0	1273.0	1273.0	1273.0	1273.0	1273.0	1273.0
gcut6	20	500	2514.0	2675.0	2682.0	2682.0	2682.0	2682.0	2675.0	**2656.0**
gcut7	30	500	4641.0	4758.0	4795.0	4788.0	4783.0	4774.0	4774.0	**4754.0**
gcut8	50	500	5703.0	6197.0	**6181.0**	6155.0	6089.0	6089.0	6081.0	6081.0

Table 6: Comparison of BLD* on problems from the literature, with fixed orientation.

time on a 1000 MHz Alpha processor. Each result is an average over 10 instances. If the result from [14] is the best result, it appears in bold; otherwise the earliest result for BLD* that beats the best result reported by [14] is in bold. An underlined result indicates a tie.

From these results, BLD* clearly performs better than all three of the algorithms in [14]. In 31 of the 50 comparisons in these tables, BLD* after 1 minute produces a better result than all three of the other algorithms do in five minutes (on a slightly slower processor), and ties in one other case. In three additional cases, BLD* outperforms the other algorithms after two minutes and in two cases, BLD* ties the other algorithms after two minutes.

Additionally, [14] also provides the time at which their algorithms found the best result during the five minutes of running time. Quite often, the best result is found within 20 or 30 seconds. Thus, for very short running times, it is possible that their algorithms outperform BLD*. As shown in the tables, however, BLD* generally continues to improve steadily over time.

Finally, we also compare BLD* using some of the other benchmarks from the literature. Table 6 shows results for the larger (in terms of number of rectangles) “ht” problems and the “gcut” problems reported on in [14]. BLD* after 1 minute performs better on all the “ht” problems and the two largest “gcut” problems. Even on the smaller “gcut” problems, BLD* eventually beats or matches the best score from the other algorithms. Of course, BLD* is designed for larger problems.

5 Interactive Packing

Human guidance has been shown to improve the performance of stochastic optimization algorithms for a variety of problems (e.g., [1, 16, 17] and the papers cited therein). In order for human interaction to be justified for an optimization problem, improvements in solution quality must have high enough value to warrant investing human effort. This is the case for packing problems in which manufacturing costs, and thus potential savings, are high. In order for interaction to be applicable to an optimization problem, there must exist effective visualizations for its problems and solutions. Fortunately, the obvious geometric visualization for packing problems (e.g., see Figure 2) is simple and effective.

In order for human interaction to be beneficial, human reasoning must offer some advantages over the best automatic methods. We have found that people can help overcome many of the limitations of the BLD* heuristic. People can identify particularly well-packed subregions of solutions, and focus BLD* on improving the other parts. Furthermore, people can readily envision multi-step repairs to a packing problem to reduce unused space. These repairs often involve producing solutions that could not be produced by the BLD heuristic.

5.1 Interactive System

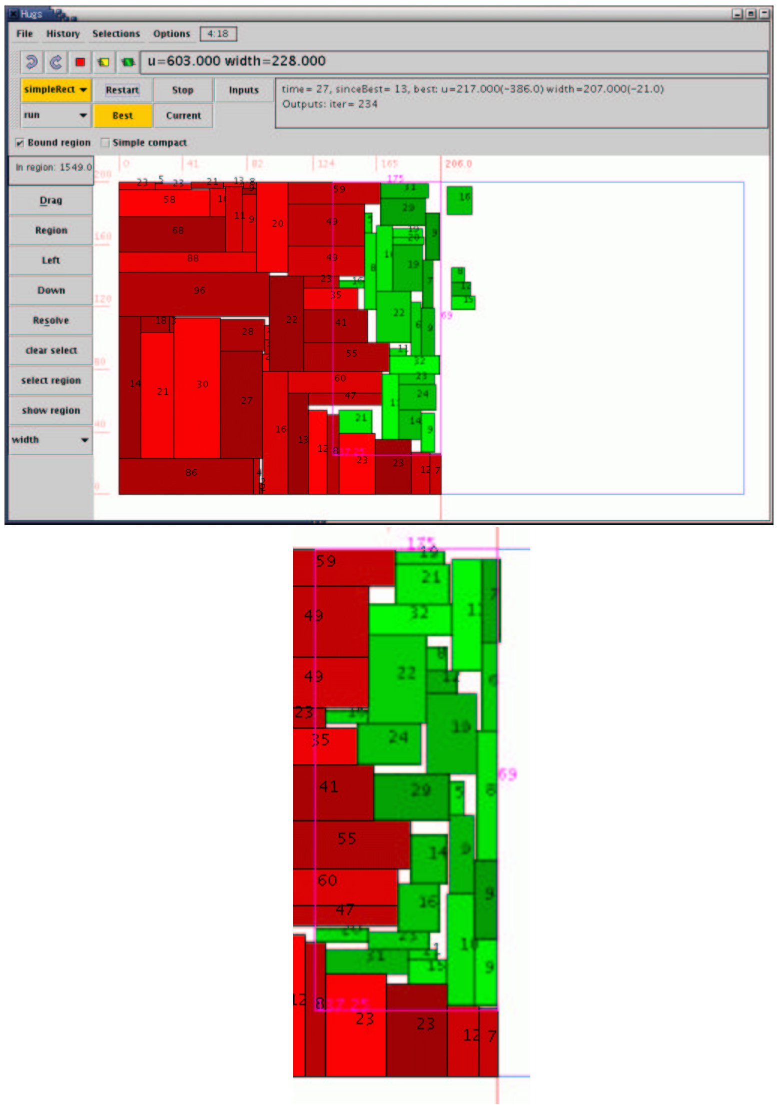
We have developed an interactive rectangle-packing system in Java using the Human-Guided Search (HuGS) Toolkit [17]. The toolkit provides a conceptual framework for interactive optimization as well as software for interacting with a search algorithm, logging user behavior, providing history functions including undo and redo, file I/O, and some other GUI functions. We did not however utilize the human-guidable tabu or hill-climbing search algorithms provided in HuGS, as we did not find them effective for this problem in our initial explorations.

In our system, the user is always visualizing a current solution as shown in Figure 2. Given the aspect ratio of a computer monitor, we found it more natural to rotate the problem by 90 degrees, so that there is a fixed height and the goal is to minimize the width of the enclosing rectangle.

The user can manually adjust the current solution by dragging one or more rectangles to a new location. The interface allows the user to cause all the rectangles to be shifted downward or leftward. This basically has the effect of pulling all of the rectangles in one direction until each touches its neighbor or an edge of the possible packing area. These functions also resolve overlaps among rectangles. Additionally, the user can freeze particular rectangles. Frozen rectangles appear in red and will not be moved by the computer. Rectangles that are not frozen appear in green. For the variable orientation problem, the user has the option of reorienting rectangles, manually.

The user can also invoke, monitor, and halt the BLD* heuristic. The user specifies a target region in which to pack rectangles, denoted by a purple rectangular outline. The user can then invoke BLD* by pressing a Start button. Any frozen rectangles within the region are left where they are. BLD* then tries to fill the region using any rectangles that are not currently frozen. The system works in the background, and uses a text display to indicate the value of the best, i.e., most tightly packed, solution it has found so far. The user can retrieve this solution by pressing the Best button. The user can retrieve the current solution the engine is working on by pressing the Current button. The user can manually modify the currently visualized solution without disturbing the current search. When the search algorithm finds a new best solution, the Best button changes color to alert the user. The user can halt the search algorithm by pressing the Stop button, or reinvoke it by pressing the Start button again.

The user can optionally set a target for the solution she is trying to reach. For example, the user can indicate that the enclosing rectangle should be 200×204. The system provides some visual cues for how to meet this goal. More importantly, the target solution size affects how solutions are ranked. Rather than using the true objective function (i.e., the size of the enclosing rectangle), the system ranks solutions based on the total area of the rectangles that fall within the target solution size. We found this feature to be extremely useful. For example, the user typically begins a session by having BLD* try to pack the entire target region. Because of our modification, the search algorithm might return, for example, a packing with one rectangle that sticks out of the target region by several units rather than a packing in which many rectangles stick out of the target region by one unit. We usually found the former packings much easier to repair.



```
u=603.000 width=228.000
time= 27, sinceBest= 13, best: u=217.000(-386.0) width=207.000(-21.0)
Outputs: iter= 234
```

Figure 2: Interactive system: The top image is a screen shot of our system in use. The user has selected a region to apply BLD* to and has frozen most of the rectangles (frozen rectangles shown in red/dark gray, unfrozen in green/light gray). The image on the right shows a blowup of the selected portion on the packing, after BLD* has run for a few seconds and the user has pressed the Best button to see the best solution found.

dataset	number of rectangles	percent over optimal by BLD* in two hours	time for users to find packing 1% closer to optimal
N4	49	4.3%	3.3% in 14 min., 21 sec.
N5	73	4.1%	3.1% in 13 min., 52 sec.
N6	97	3.3%	2.3% in 17 min., 12 sec.

Table 7: Interaction experiment results with fixed orientation: The second column shows the average percentage over optimal achieved by BLD* in two hours. These results are at least 2%-3% closer to optimal than the best previously published results. The third column shows the average time it took interactive use of BLD* to achieve a solution another 1% closer to optimal. The values are averaged over the five problem instances in the corresponding collection.

5.2 Interaction Experiments

The primary goal of these experiments was to evaluate the hypothesis that interactive use of BLD* can produce superior solutions than BLD* can on its own.

We ran our first set of experiments on the fixed-orientation problem, using the 15 problem instances in the N4-N6 collections in the Hopper benchmark suite. We ran BLD* for 2 hours on on each instance. (We optimized the code slightly since we ran the user experiments and thus BLD* was slower here than shown in Tables 2 and 3.) We then performed one trial for each instance in which a user attempted to find a solution 1% closer to optimal than the best solution found by BLD* within 2 hours, e.g., if BLD* found a solution of width 206, we would give the users a target of 204. The users were two authors of this paper. We were careful that a user had never before seen the particular instances on which they were tested. We logged the users' actions, but the primary measure was how long it took the user to reach their target.

As shown in Table 7, the users were able to reach these targets in about 15 minutes on average. In every case, the target was reached within 30 minutes. While this is not exactly a "head-to-head" comparison, since the users had the target scores to reach, the fact that people were able to improve on the solutions so quickly confirms our hypothesis.

The N7 problem instances presented a significant challenge because BLD* was able to produce extremely tight packings, only 1.8% over optimal on average, even for the fixed-orientation problem. In our practice trials, we found it difficult to improve upon these solutions, interactively, using only BLD*. The difficulty is that the unused space is distributed into a great number of tiny gaps throughout

dataset	number of rectangles	percent over optimal by BLD* in two hours	time for users to find packing 1% closer to optimal
N4	49	2.9%	1.9% in 26 min., 21 sec.
N5	73	2.6%	1.6% in 19 min., 59 sec.

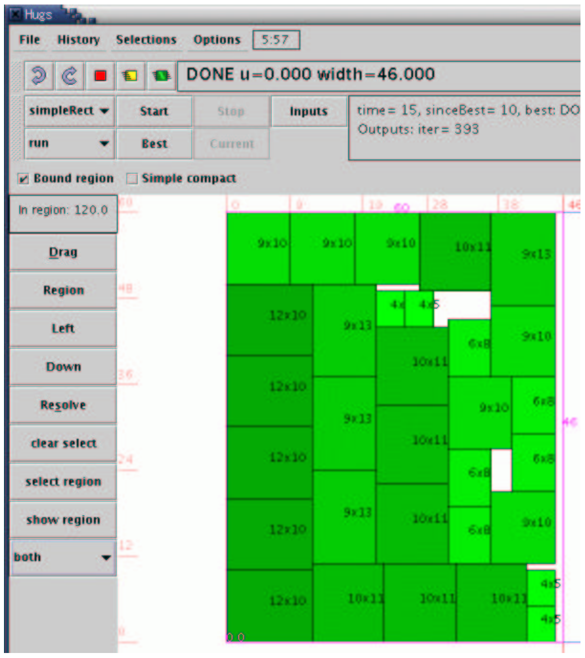
Table 8: Interaction experiment results with variable orientation: The second column shows the average percentage over optimal achieved by BLD* in two hours. The third column shows the average time it took interactive use of BLD* to achieve a solution another 1% closer to optimal. The values are averaged over the 2 trials each of five problem instances in the corresponding collection.

the packing. This makes it harder to pack the remaining rectangles into the target space. We were able to make steady progress, but it seemed like it would take hours to get a better solution. Instead, we devised a divide-and-conquer algorithm which produced solutions in which unused space is more concentrated (described more fully in [19]). Using the divide-and-conquer algorithm as well as BLD*, our test subjects were able to produce solutions 1% over optimal (or about 0.8% closer than BLD* could achieve on average) in 12.5 to 36 minutes of interactive use.

We also ran a set of experiments for the variable-orientation problem. In these experiments, the users employed our variation of BLD* that orients the rectangle and could also manually orient them. As in the first experiments, we measured how long it took the users to find solution 1% closer to optimal than the best solution found by BLD* within 2 hours. We ran these experiments on the N4 and N5 collections, with the same users as the first experiments. Thus, in this experiment, the users had previously worked on the problem instances in the fixed-orientation variation. However, we believe both that there is little transfer between the problem variations, and that it is extremely hard to remember anything about a given problem instance. For these problems, we ran both users on each problem instance.

We thought this task might be too difficult since the targets were so much closer to optimal. However, as shown in Table 8, the users were able to reach the targets almost as quickly as in the first experiments, requiring an average of 23 minutes and 10 seconds.

To verify that our results were not dependent on the Hopper dataset, we ran interaction experiments on random test instances (designed before we were aware of the random problems in the literature.) We used four problems with 50 rectan-



DONE u=0.000 width=46.000

time= 15, sinceBest= 10, best: DO
Outputs: iter= 393

In region: 120.0

Figure 3: Our solution to the D1 dataset, which is one unit better than the best previously published solution. The solution has width 46 in our interface, or height 46 in the standard formulation.

percent over ideal by BLD* in two hours	percentage successful trials	time for users to find packing approx. 1% closer to ideal
3.85%	93.75%	2.90% in 15 minutes, 35 seconds

Table 9: Interaction experiment results on random data sets with variable orientation. The values are averaged over the 2 trials each of eight problem instances (excluding one that was not solved within an hour).

gles, and four with 100 rectangles. For half of each size, we chose the dimensions of the rectangles uniformly from 1 to 50. For the other half, we chose the width x uniformly at random from 1 to 50, and the height by choosing a number y uniformly at random from 1 to 50 and fixing the dimension to be either y or $50 - y$, whichever is further from x; this skews the rectangles making them less square. Since we do not know the optimal answer for the randomly generated benchmarks, we evaluate a packing with a given height in terms of its percentage *over ideal*, where the ideal is the nearest integer rounding up from the total area divided by the target width.

Two users each tried all eight problem instances. As in the second set of experiments, variable orientation was allowed. We measured how long it took the users to find solution 1% closer to ideal than the best solution found by BLD* within 2 hours, rounding to the nearest integer. Notice that the ideal width is not always evenly divisible by 100, and so when aiming for 1% closer to ideal, we were forced to round. Furthermore, since we do not know the optimal width, we did not know a priori that the targets were achievable.

As shown in Table 9, the results were similar to the previous experiments. In one case, a user was not able to achieve the target within an hour, but for the other 15 cases, the average time to reach the target was only 15 minutes and 35 seconds.

Finally, we also tested our interactive system on the few other (fixed-orientation) benchmarks we could find in the literature, including in particular ones without known optimal solutions, referred to by Hopper as D1 and D3. [11, 24, 25]. The best solutions for D1 and D3 in the literature appear to have height 47 and 114. We were able to find a solution with height 46 (or width 46 in our interface) in about 15 minutes, as shown in Figure 3. We were able to match the 114 for D3 in about 20 minutes.

6 Conclusion

We have developed several new approaches for 2D rectangular strip packing problems, improving the state of the art and providing new insights into the problem. Specifically, we have shown that our BLD* algorithm outperforms previous automatic methods such as those described by Iori, Martello, and Monaci [14].

Equally significant is the demonstration of the utility of interaction for packing problems. On the larger Hopper benchmark problems, we come within 1.6%-3.3% of optimal in about 15 minutes of interactive use: this is a significant improvement over all previously reported results. We believe that for many similar problems, humans have significant geometric insight that is currently difficult to capture in a computer algorithm. Interactive systems can tap into that insight while still taking advantage of the computer's superior computational power.

There are two clear broad directions that could be pursued based on results for interactive systems. One tack would be to attempt to classify how human users obtain improved results for this problem, and design an algorithm that encodes this approach well enough to match or exceed human performance. We believe that this could be a difficult task; indeed, our two users seemed to pursue very different strategies in their use of the system. This approach highlights the utility in developing interactive systems to inspire and refine new algorithms. A second tack would be to design interactive systems for other geometric problems, in order to gain insight into how to best design systems that allow beneficial interaction to occur. This is in the spirit of the ongoing HuGS project.

References

[1] D. Anderson, E. Anderson, N. Lesh, J. Marks, B. Mirtich, D. Ratajczak, and K. Ryall, Human-guided simple search. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pp. 209–216, 2000.

[2] B. S. Baker, E. G. Coffman, Jr., and R. L. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9:846-855, 1980.

[3] B. S. Baker, D. J. Brown, and H. P. Katseff. A $5/4$ algorithm for two-dimensional packing. *Journal of Algorithms*, 2:348-368, 1981.

[4] D. J. Brown. An improved BL lower bound. *Information Processing Letters*, 11:37-39, 1980.

[5] B. Chazelle. The Bottom-Left Bin-Packing Heuristic: An Efficient Implementation. *IEEE Transactions on Computers*, 32(8):697-707, 1983.

[6] E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin-packing: an updated survey. In: G. Ausiello, M. Lucertini, and P. Serafini, editors, *Algorithm Design for Computer Systems Design*, pages 49-106, Springer-Verlag, 1984.

[7] C. H. Dagli and P. Poshyanonda. New approaches to nesting rectangular patterns. *Journal of Intelligent Manufacturing* 8, pp. 177-190, 1997

[8] H. Dyckhoff. Typology of cutting and packing problems. *European Journal of Operational Research*, 44, 145-159, 1990.

[9] S. P. Fekete and J. Schepers. On more-dimensional packing III: Exact algorithms. Available as a preprint at Mathematisches Institut, Universität zu Köln, preprint key zpr97-290.

[10] S. P. Fekete and J. Schepers. A New Exact Algorithm for General Orthogonal D-Dimensional Knapsack Problems. In *Proceedings of the 5th Annual European Symposium on Algorithms*, pp. 144-156, 1997.

[11] E. Hopper. Two-Dimensional Packing Utilising Evolutionary Algorithms and other Meta-Heuristic Methods, PhD Thesis, Cardiff University, UK. 2000.

[12] E. Hopper and B. C. H. Turton. An Empirical Investigation of Meta-heuristic and Heuristic Algorithms for a 2D Packing Problem. *European Journal of Operational Research*, 128(1):34-57,2000.

[13] Hwang S.M., Cheng Y.K., and Horng J. T., On solving rectangle bin packing problems using genetic algorithms. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, Part 2 (of 3), pp. 1583-1590.

[14] M. Iori, S. Martello, and M. Monaci. Metaheuristic Algorithms for the Strip Packing Problem, in P. M. Pardalos, V. Korotkikh, Eds., *Optimization and Industry: New Frontiers*, Kluwer Academic Publishers, pp. 159-179, 2003.

[15] C. Kenyon and E. Remilia. Approximate Strip-Packing. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 31-36, 1996.

[16] G. Klau, N. Lesh, J. Marks, and M. Mitzenmacher. Human-Guided Tabu Search. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pp. 41-47, 2002.

[17] G. Klau, N. Lesh, J. Marks, M. Mitzenmacher, and G.T. Schafer. The HuGS platform: A toolkit for interactive optimization. In *Proceedings of Advanced Visual Interfaces*, pp. 324-330, 2002.

[18] Korf, R. E. Optimal rectangle packing: Initial results. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-03)*, Trento, Italy, 2003.

[19] N. Lesh, J. Marks, A. McMahon, and M. Mitzenmacher. New Exhaustive, Heuristic, and Interactive Approaches to 2D Rectangular Strip Packing. MERL Technical Report TR2003-05, 2003.

[20] N. Lesh, J. Marks, A. McMahon, and M. Mitzenmacher. Exhaustive approaches to 2D rectangular perfect packings. *Information Processing Letters*, 90, pp. 7-14, 2004.

[21] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2);241-252, 2003.

[22] J. I. Marden. *Analyzing and Modeling Rank Data*, Chapman & Hall, New York, New York, 1995.

[23] S. Martello, M. Monaci, and D. Vigo. An exact approach to the strip packing problem. *INFORMS Journal on Computing*, 15(3):310-319, 2003.

[24] K. Ratanapan and C. H. Dagli. An object-based evolutionary algorithm for solving rectangular piece nesting problems. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pp. 989-994, 1997.

[25] K. Ratanapan and C. H. Dagli. An object-based evolutionary algorithm: the nesting solution. In *Proceedings of the International Conference on Evolutionary Computation*, pp. 581-586, 1998.

[26] D. Sleator. A 2.5 times optimal algorithm for packing in two dimensions. *Information Processing Letters*, 10:37-40, 1980.

[27] A. Steinberg. A Strip-Packing Algorithm with Absolute Performance Bound 2. *SIAM Journal on Computing*, vol 26, number 2, pp. 401-409, 1997.