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

New Exhaustive, Heuristic, and Interactive Approaches to 2D Rectangular Strip Packing

Neal Lesh, Joe Marks, Adam McMahon, Michael Mitzenmacher

TR2003-05 February 2003

Abstract

In this paper, we consider the two-dimensional rectangular strip packing problem. Computers are especially efficient at producing and evaluating possible packings of a given set of rectangles. However, the characteristics of the search space appear to make it not amenable to standard local search techniques, such as simulated annealing or genetic algorithms. A standard simple heuristic, Bottom-Left-Decreasing, has been shown to handily beat these more sophisticated search techniques. We present several new approaches to the problem. We show that a branch-and-bound-based exhaustive search is much more effective than previous methods for problems with less than 30 rectangles when the rectangles can be tightly packed with little or no unused space. For larger problems, we introduce and demonstrate the effectiveness of a natural improvement to the Bottom-Left-Decreasing heuristic. Furthermore, 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 previous methods on benchmark problems.

Submitted to Computational Geometry 2003

This work may not be copied or reproduced in whole or in part for any commercial purpose. Permission to copy in whole or in part without payment of fee is granted for nonprofit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of Mitsubishi Electric Research Laboratories, Inc.; an acknowledgment of the authors and individual contributions to the work; and all applicable portions of the copyright notice. Copying, reproduction, or republishing for any other purpose shall require a license with payment of fee to Mitsubishi Electric Research Laboratories, Inc. All rights reserved.

Copyright © Mitsubishi Electric Research Laboratories, Inc., 2003
201 Broadway, Cambridge, Massachusetts 02139

Submitted December 2002

New Exhaustive, Heuristic, and Interactive Approaches to 2D Rectangular Strip Packing

N. Lesh* J. Marks† A. McMahon‡ M. Mitzenmacher§

Abstract

In this paper, we consider the two-dimensional rectangular strip packing problem. Computers are especially efficient at producing and evaluating possible packings of a given set of rectangles. However, the characteristics of the search space appear to make it not amenable to standard local search techniques, such as simulated annealing or genetic algorithms. A standard simple heuristic, Bottom-Left-Decreasing, has been shown to handily beat these more sophisticated search techniques.

We present several new approaches to the problem. We show that a branch-and-bound-based exhaustive search is much more effective than previous methods for problems with less than 30 rectangles when the rectangles can be tightly packed with little or no unused space. For larger problems, we introduce and demonstrate the effectiveness of a natural improvement to the Bottom-Left-Decreasing heuristic.

Furthermore, we observe that while the search space for this problem appears difficult for computers to navigate, people seem able to reason about it extremely well. This is unsurprising in that people are known to outperform computers at packing irregular polygons in industrial applications. 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 previous methods on benchmark problems.

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 oriented, orthogonal variation, where rectangles must be placed parallel to the horizontal and vertical axes, and the rectangles cannot be rotated. Further, for our test cases, all dimensions are integers. Like most packing problems, 2D rectangular strip packing (even with these restrictions) is NP-hard.

*Mitsubishi Electric Research Laboratories, 201 Broadway, Cambridge, MA, 02139. lesh@merl.com

†Mitsubishi Electric Research Laboratories, 201 Broadway, Cambridge, MA, 02139. marks@merl.com

‡University of Miami, Florida. This work done while visiting Mitsubishi Electric Research Laboratories. adam@math.miami.edu

§Harvard University, Computer Science Department. michaelm@eecs.harvard.edu. 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.

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) heuristic in this setting 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 packings [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 [8]. 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 proven amenable to such search techniques [8].

In this paper, we present exhaustive, heuristic, and interactive approaches that outperform previous methods. For moderate size problems (fewer than 30 rectangles), we describe an exhaustive search using branch-and-bound techniques. We show it is extremely effective for problems in which the rectangles can be tightly packed with little or no unused space. For example, it solves benchmark problems containing 25 rectangles in an average of 96 seconds.

For larger problems, we present a variation of the BLD heuristic called BLD* that considers successive random perturbations of the original four decreasing orderings. On our benchmark instances, BLD* substantially outperforms BLD as well as applying BL to randomly chosen orderings. For example, after just one minute, BLD* reduces the packing height from an average of 6.4% over optimal by BLD to about 4.6% over optimal, and continues to make improvements given more time. We note that improvements of even 1% can be very valuable for industrial applications of this problem, such as glass and steel cutting.

Finally, we observe that while the search space for this problem is difficult for computers to navigate, people seem to be able to reason about it extremely well.[1] 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. We incorporate our new algorithms in an interactive system that combines the advantages of computer speed and human reasoning. Our experiments on large benchmarks show that interactive use of BLD* can produce solutions a 1% closer to optimal in about 15 minutes than BLD* produces on its own in 2 hours.

1.1 Further 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 [8].

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, 7]. One focus has been

[1]This is unsurprising in that people are known to outperform computers at packing irregular polygons in industrial applications [14].

on approximation algorithms. The Bottom-Left heuristic has been shown to be a 3-approximation when the the rectangles are sorted by decreasing width (but the heuristic is not competitive when sorted by decreasing height) [2]. Other early results include algorithms that give an asymptotic 5/4-approximation [3] and an absolute 5/2-approximation [17]. Recently, Kenyon and Remilia have developed a fully polynomial approximation scheme [10].

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 [8, 9].

2 An Exhaustive Branch-and-Bound Algorithm

To begin, we present an exhaustive branch-and-bound algorithm that performs extremely well on problems with fewer than 30 rectangles. It is especially well suited for finding *perfect packings*. A perfect packing is one where the input rectangles fit exactly into a rectangle of the appropriate width with no empty space. One can think of the perfect packing case as being a jigsaw puzzle with oriented rectangular pieces. We also discuss how the scheme generalizes where there is no perfect packing.

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 worst-case $O(n^3)$ algorithms 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 [8].

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 this as BLD.

2.2 Finding Perfect Packings Exhaustively

To begin, we consider the use of BL for finding perfect packings. Although there are examples for which BL cannot produce the optimal packing under *any* ordering [2, 4], this is not the case when the optimal packing is a perfect packing. We have not seen the following fact in the literature, although it may simply be a folklore result.

Theorem 1 *For every perfect packing, there is a permutation of the rectangles that yields that perfect packing using the BL heuristic.*

Proof: Sort the lower left corners of the rectangles in the perfect packing lexicographically. This gives a permutation ordering that will yield that packing using the BL heuristic. □

This theorem indicates that applying BL exhaustively to all possible permutations of the given rectangles will find a perfect packing if one exists. Furthermore, it suggests an important optimization for exhaustive search because it shows that there exists an ordering that yields a perfect packing with the BL heuristic such that every rectangle is placed with the lower left corner in the *first open location* in the lexicographic ordering. (The BL heuristic generally places a rectangle in the first open ordering *in which it fits.*) Thus, an ordering can be rejected as soon as any rectangle does not fit in the first open location. Even though this ordering could possibly yield a perfect packing with the BL heuristic, we are guaranteed to find this perfect packing with some other ordering during our exhaustive search. In the branch-and-bound algorithm, below, we use this idea to dramatically prune the search space.

2.3 Branch-and-Bound with Gap Pruning

To efficiently consider all possible permutations, we use a branch-and-bound framework. Rectangles are placed one at a time, so that at any point in time a *prefix* of some permutation has been placed. The branch is on the next rectangle in the prefix of the permutation. In the case where we have several rectangles with the same dimensions, we can work more efficiently by associating a type with each distinct pair of rectangle dimensions, and branching on the type. The bound is a lower bound on the unused space in any completion of the current prefix.

For perfect packings, we have a trivial bound of zero acceptable empty space. If we can determine that a prefix cannot yield a perfect packing, then we can bypass all completions of that prefix, greatly reducing the time for the exhaustive search. From Theorem 1 we additionally know that if there is a perfect packing, there is an ordering that yields the perfect packing where each rectangle is placed in the first open location in the lexiographic ordering. Thus, if the next rectangle to be placed does not fit in that location, we can immediately prune.

Although we do not report results on branch-and-bound for non-perfect packings (i.e., those which contain unused space) until Subsection 4.3, we describe here how the algorithm can be used to search for them. For non-perfect packings, the bound is defined by the best packing found so far (or an initial bound can be set by a user). We note that, in general, for any packing achievable by BL, there is an ordering that yields that packing in which each rectangle is placed at least as high as all previously placed rectangles. (This is an obvious generalization of Theorem 1 for non-perfect packings.) This justifies including any unused space below a placed rectangle in the lower bound for the unused space associated with the current prefix.

We now describe a more powerful bounding method. While observing our algorithm run interactively, we determined that much time was wasted in the following type of scenario, demonstrated in Figure 1. Suppose the current placement of rectangles requires a *gap* of width w and height h to be filled for a perfect packing. If there are no unplaced rectangles of width w, nor any way to combine unplaced rectangles to obtain rectangles of width w, then there is no way to obtain a perfect packing. Note that gaps arise between placed rectangles and between rectangles and the outer boundary.

To handle this situation, we have found it worthwhile to implement a simple procedure based on dynamic programming that provides a loose upper bound on the tallest possible rectangle of width

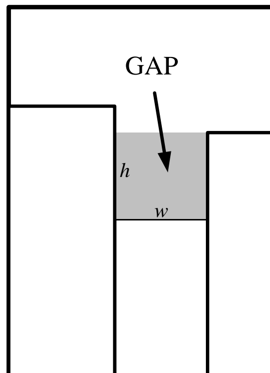


Figure 1: Mind the gap!

w that can be constructed with the unplaced rectangles. Note that for the Bottom-Left heuristic, bounding in this fashion is more useful than bounding the widest possible rectangles of height h, because we create more gaps of small width than small height early in the prefix ordering. Although both can be used, our experience is that the best performance is achieved by using only bounding on the width of the gaps.

Our approach is easily described as follows. Consider a list of the unplaced rectangles $R_1, R_2, \ldots, R_n$ in some order. Let $w(R_i)$ and $h(R_i)$ be the width and height of R_i. We find values $B_{j,k}$ that are upper bounds on the maximum height rectangle of width $j \geq 1$ that can be constructed using the first $k \geq 1$ rectangles. Hence $B_{w(R_1),1} = h(R_1)$ and $B_{j,1} = 0$ if $j \neq w(R_1)$. For $k > 1$, we choose:

$$B_{j,k+1} = B_{j,k} \text{ if } j < w(R_{k+1});$$
$$B_{j,k+1} = B_{j,k} + h(R_{k+1}) \text{ if } j = w(R_{k+1});$$
$$B_{j,k+1} = B_{j,k} + \min(B_{j-w(R_{k+1}),k}, h(R_{k+1})) \text{ if } j > w(R_{k+1}).$$

Theorem 2 follows from an obvious induction:

Theorem 2 *For all $j, k \geq 1$, $B_{j,k}$ is an upper bound on the maximum height rectangle of width j that can be constructed using $R_1, R_2, \ldots, R_k$.*

The bound above is loose, because in the case where $j > w(R_{k+1})$, a rectangle R_i with $i \leq k$ may be contributing to both terms in the summation. However, note that in the case where there is no way to place the remaining rectangles to obtain a width w, then in fact $B_{w,n}$ will equal 0. Further, the bounds can depend on the order in which the remaining rectangles are considered following the procedure above.

Calculating $B_{j,n}$ for every j up to the biggest gap after each placement and checking that all gaps can at least potentially be filled allows the algorithm to avoid prefixes that cannot yield perfect packings. The bound above can be improved slightly in various ways; for example, taking the best bound from different orderings of the unplaced rectangles, and adding a bit more sophistication to avoid overcounting caused by many rectangles with small width. We have found that the technique above applied once to a random ordering is effective in our experiments.

This technique generalizes to non-perfect packings. For example, if there is a gap of width j and $B_{j,n}$ is 0, then the height of the gap is a lower bound on the unused space inside the gap.

2.4 Solution-richness

Our experience is that problems that have at least one perfect packing typically have a great number of them. Informally, we say that a class of problems is *solution-rich* if it has this property. Solution-rich problems are more amenable to exhaustive searches, since there are many good solutions to find. We believe that in many cases perfect packing problems are solution-rich, since often rectangles combine into a larger rectangle that can be symmetrically reconfigured in various ways to obtain a different perfect packing. Even the small problem instances we consider below have hundreds of solutions.

One class of problems that is provably solution-rich is those with *guillotinable* solutions. A guillotinable solution has the property that it can be obtained by a sequence of cuts parallel to the axes, each of which crosses either the entire length or width or the remaining connected rectangular piece. Guillotinable solutions are important for some manufacturing settings [8]. A problem with one guillotinable perfect packing must have many.

Theorem 3 *Any guillotinable problem on n rectangles with a perfect packing has at least 2^{n-1} perfect packings.*

Proof: The proof is a simple induction. Consider the first cut of the guillotinable solution. This divides the problem into two subproblems of size k and ℓ with $k + \ell = n$. These subproblems have 2^{k-1} and $2^{\ell-1}$ perfect packings respectively by the induction hypothesis, and there are 2 ways to put the two subproblems together. □

We note that the non-guillotinable problems of Hopper that we use as benchmarks are constructed in such a way that they are also solution-rich. A simple induction shows that these benchmarks have at least $2^{(n-1)/2}$ perfect packings. We omit details for brevity.

2.5 Near-Perfect Packings

Our methods for efficiently handling perfect packings can be applied when the optimal packing contains only a small amount of unused space. This can be achieved by simply introducing a small number of 1×1 rectangles, corresponding to the amount of acceptable unused space. This increases the branching factor, although note that all 1×1 rectangles can be treated as of the same type, so the branching increase for k 1×1 rectangles is not as large as for k rectangles with distinct sizes.

2.6 Experimental Results

Our primary set of benchmarks is the recently developed set of 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 [8]. 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. Furthermore, we were unable to find any other substantial set of benchmarks in the literature. The benchmark problems, however, are highly structured,

dataset	size	num solved	seconds to solve	iterations to solve
N1	17	5/5	less than 1	259.0
N2	25	5/5	96	3,658,186.8
N3	29	4/5	496.5	17,648,907.75

Table 1: Exhaustive branch-and-bound for perfect packings with gap pruning. The best-performing previous methods produce solutions at best 5% above optimal [8].

and because all instances have perfect packings, they yield limited insight on the performance of algorithms when perfect packings are not available.

We evaluate our branch-and-bound algorithm on collections N1, N2, and N3, which have 17, 25, and 29 rectangles, respectively. The best-performing previous methods produce solutions at best 5% above optimal [8]. As shown in Table 1, our branch-and-bound algorithm quickly finds perfect packings for all benchmark instances with 17 and 25 rectangles and 4 out of 5 instances with 29 rectangles. [2] (It did not solve the last 29-rectangle problem even when run for several hours.) The table also shows the number of iterations, or placed rectangles, required to find the perfect packings.

We were significantly aided by the solution-richness of the instances. Our algorithm found a solution after exploring, on average, about 1% of the search space.

The gap-pruning is also extremely effective. Our algorithm requires 11,988.6 iterations to solve the N1 cases with these pruning methods turned off, compared to only 259.0 iterations on average with the pruning. Without the pruning, it was only able to solve one of the N2 cases within half an hour.

3 Improving the BLD Heuristic

While our exhaustive branch-and-bound algorithm can solve problems with less than 30 rectangles quickly, for a larger number of rectangles, exhaustive search becomes prohibitive. Previous results show that the BLD heuristic appears to be the most effective algorithm [8].

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 [8]. 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 a matter of seconds.

Instead we suggest the following variation, which we call BLD*. Our intuition for why BLD* performs so much better than random BL 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 [13]. BLD* uses the following

[2] All experiments reported in this paper were run on a Linux machine with a 2000 MhZ Pentium processor running unoptimized Java code.

time (minutes)	average height over optimal
0 (BLD)	6.4%
1	4.6%
2	4.4%
5	4.0%
10	3.7%
20	3.6%
120	3.4%

Table 2: Average results of BLD* on N4-N7 at various time intervals. The first row indicates the result of BLD on these problems.

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 either an item is accepted or the last item is reached (in which case it is accepted). After an item is accepted, the next item is selected, starting again from the beginning of the list. 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.[3] BLD* first tries the four orders used by BLD and then permutes each of these orders in round-robin fashion.

3.1 Experimental Results

We ran BLD* on collections N4 through N7, using $p = 0.5$ after some preliminary investigation. As shown in Table 2, BLD* dramatically improves solutions over BLD even with just one minute of computation. It continues to improve steadily, though improvements clearly taper off with time. We also measured 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.

4 Interactive Packing

Human guidance has been shown to improve the performance of optimization algorithms for a variety of problems (e.g., [1, 11, 12] 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.

[3] A slightly more elegant approach, which we have not yet implemented, would not simply take the last rectangle if we reach the end of the list, but instead restart at the beginning of the list, again taking an item with probability p. In this case, 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 (also known as bubble-sort distance) between the two permutations. We do not expect this to affect performance.

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.

4.1 Interactive System

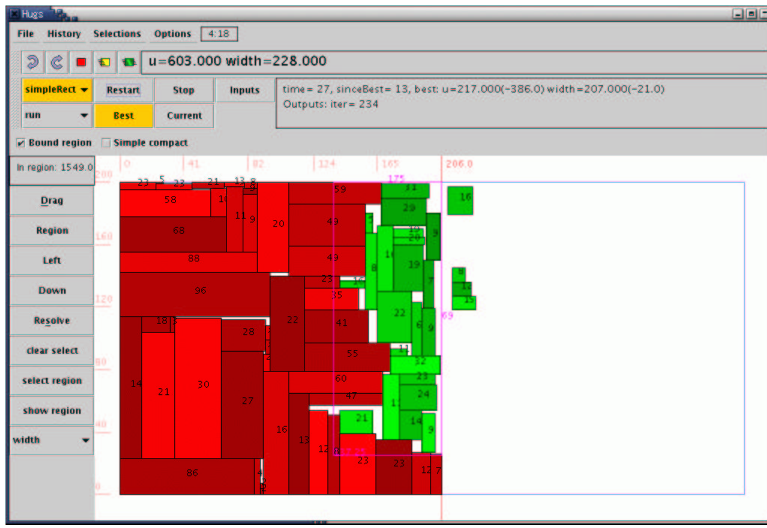
We have developed an interactive rectangle-packing system in Java using the Human-Guided Search (HuGS) Toolkit [12]. The toolkit provides a conceptual framework for interactive optimization as well as software for interacting with a search algorithm, logging user behavior, providing history functions 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 contains buttons which allow 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.

The user can also invoke, monitor, and halt either our branch-and-bound algorithm or the BLD* heuristic. The user specifies a target region in which to pack rectangles, denoted by a purple rectangular outline. The user can then choose a search algorithm and invoke it by pressing a Start button. Any frozen rectangles within the region are left where they are. The search algorithm 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 re-invoke 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



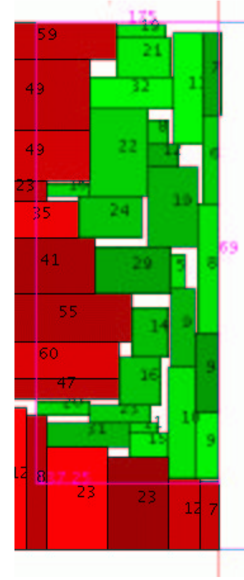


Figure 2: Interactive system: The image on the left is a screen shot of our system in use. The user has selected a region to apply BLD* to and has frozen most of the rectangles (frozen rectangles shown in red/dark gray, unfrozen in green/light gray). The image on the right shows a blowup of the selected 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.

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.

4.2 Experiments on Medium to Large Problems

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 experiments on the 20 problem instances in the Hopper benchmark suite that contain between 49 and 97 rectangles. We ran BLD* for 2 hours on on each instance. 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. BLD* was the only search algorithm available for these experiments. 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 3, 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. Note that because BLD* is a random restart strategy, it is unlikely that running it for another 2 hours would improve the solution much.

We also tested our interactive system on the few other benchmarks we could find in the literature, including in particular ones without known optimal solutions, referred to by Hopper as D1 and D3.

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 3: Interaction experiment results: 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.

[8, 15, 16]. 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.

4.3 Divide-and-Conquer for Very Large Problems

The instances in the N7 collection, with 197 rectangles, have a qualitatively different feel than the others in the Hopper dataset. Automatic methods are quite successful at producing dense solutions for these instances. BLD* produces solutions within 1.8% of the optimal height, on average, for the five instances within 20 minutes. It did not find better solutions even with up to 2 hours of computation.

In our practice trials, we also found it difficult to improve upon these solutions, interactively, using only BLD*. The problem is that the unused space is distributed into a great number of tiny gaps throughout the packing. This makes it harder to pack the remaining rectangles into the target space (e.g., 1% over optimal). We were able to make steady progress, but it seemed like it would take hours to fit them all in.

However, we found that an automatic divide-and-conquer algorithm that we have been working on is very useful for these large problems. As an automatic algorithm, this algorithm does not outperform BLD*. However, we found that it produces solutions that are more amenable to human repair because the unused space amalgamates in larger regions.

We informally describe the algorithm as follows. The algorithm is given a target size that defines the enclosing rectangle within which to fit all the given rectangles. The algorithm divides the target region into a set of rectangular sub-regions, and works on each sub-region separately. The first sub-region is either a wide rectangle that has the width of the entire target region with a random height (within some given bounds) or a tall rectangle that has the height of the entire target region with a random width. It decides whether to make a wide or tall sub-region probabilistically based on whether there are more wide or tall unplaced rectangles. To fill a sub-region it first runs branch-and-bound on that region in the mode designed for perfect packings. This fills some portion of the sub-region without introducing any unused space. It then freezes the rectangles within the sub-region and runs a second branch-and-bound algorithm, without the perfect-packing constraint. It then freezes the rectangles placed by the second invocation of branch-and-bound. At this point, it essentially repeats the entire process on the remaining unused area and the remaining unplaced

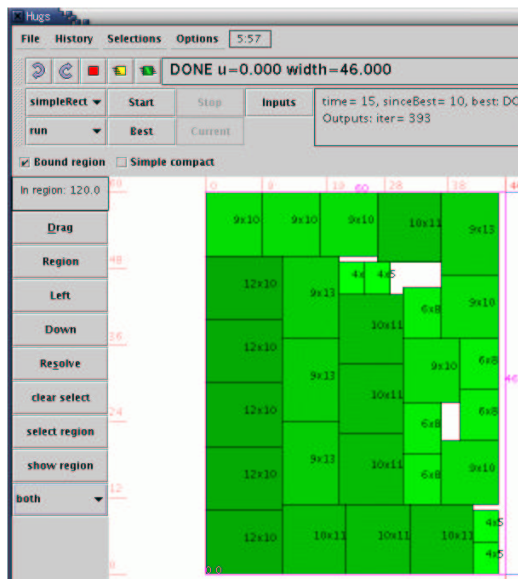


Figure 3: Our solution to the D1 dataset, which is one unit better than the best previously published solution. The solution has width 46 in our interface, or height 46 in the standard formulation.

pieces. When the unused area becomes small enough, it runs BLD* on the sub-problem that remains (we found this more effective than treating the last area as its own sub-region.) An optimization, which we use in the experiments, is that for each phase, it tries several (e.g., five) sub-regions and selects the one with minimum unused space.

This approach works well in that the branch-and-bound algorithm is often able to fill the first few sub-regions perfectly, or near perfectly, and thus all the unused space is concentrated in the last sub-regions. This is an advantage for user interaction because it allows people to focus their repair efforts on a much smaller subproblem.

We ran experiments on the N7 collection similar to those described above, with the exception that the users began the session by invoking the divide-and-conquer routine and letting it run for about 10 to 15 minutes. The user then worked to improve the best solution by divide-and-conquer using manual repair and applying BLD* to sub-regions of the space. For each problem, we set a target of 1% over optimal (or about 0.8% closer than BLD* could achieve on average). We were able to reach this target for all 5 of the N7 problem instances, requiring from 12.5 to 36 minutes.

5 Conclusion

We have developed several new approaches for 2D rectangular strip packing problems, substantially improving the state of the art and providing new insights into the problem. Our branch-and-bound algorithm and improved heuristic algorithm BLD* raise the bar for completely automatic systems. There is certainly further work that can be done for both approaches. For example, improving the upper bounding method used for gaps or finding other ways to lower bound wasted space may allow larger problems to be handled with branch-and-bound techniques. Finding ways to tune the

parameter p for the BLD* algorithm or determining if other ways of generating orders for the BL heuristic are better would be useful.

We believe our most significant contribution, however, may be the demonstration of the utility of interactive systems in this context. On our benchmark problems, we come within 1%-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 provide a way of tapping that insight while still taking advantage of the computer's substantially greater 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. Even if it is possible, however, it 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 the spirit of ongoing HuGS project.

As an example of a more specific problem to be tackled, the variation where rectangles are allowed to be rotated 90 degrees seems worth studying in the context of interactive algorithms. The additional choices this variation allows dramatically increases the state space and potential complexity for purely automatic algorithms. We suggest instead the following possible approach. The automatic algorithms do not use the ability to rotate rectangles; instead, this ability is reserved for the user, who can use it to gain flexibility in trying to repair solutions generated automatically. We suspect this approach may be quite successful for this class of problems.

6 Acknowledgments

We would like to thank Victor Milenkovic for useful discussions and for sending us an excellent summer intern!

References

[1] D. Anderson, E. Anderson, N. Lesh, J. Marks, B. Mirtich, D. Ratajczak, and K. Ryall, 2000. Human-guided simple search. In *Proceedings of AAAI 2000*, 209–216.

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

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

[9] 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.

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

[11] 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.

[12] 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.

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

[14] V. J. Milenkovic and K. M. Daniels. Translational Polygon Containment and Minimal Enclosure using Mathematical Programming. *International Transactions in Operational Research*, 6:525-554, 1999.

[15] 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.

[16] 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.

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