2D-Packing Images on a Large Scale: Packing a Billion Rectangles under 10 Minutes

Dominique Thiebaut
Dept. Computer Science
Smith College
Northampton, Ma 01063
Email: dthiebaut@smith.edu

***Abstract*—We present a novel heuristic for 2D-packing of rectangles inside a rectangular area where the aesthetics of the resulting packing is amenable to generating large collages of photographs or images. The heuristic works by maintaining a sorted collection of vertical segments covering the area to be packed. The segments define the leftmost boundaries of rectangular and possibly overlapping areas that are yet to be covered. The use of this data structure allows for easily defining ahead of time arbitrary rectangular areas that the packing must avoid. The 2D-packing heuristic presented does not allow the rectangles to be rotated during the packing, but could easily be modified to implement this feature. The execution time of the present heuristic on various benchmark problems is on par with recently published research in this area, including some that do allow rotation of items while packing. Several examples of image packing are presented. A multithreaded version of our core packing algorithm running on a 32-core 2.8 GHz processor packs a billion rectangles in under 10 minutes.**

Keywords–bin packing; rectangle packing; multi-threaded and parallel algorithms; heuristics; greedy algorithms; image collages.

I. Introduction

We present a new heuristic for placing two-dimensional rectangles in a rectangular surface. The heuristic keeps track of the empty area with a new data structure that allows for the natural packing around predefined rectangular areas where packing is forbidden. The packing flows in a natural way around these "holes" without subdividing the original surface into smaller packing areas. The main application for this heuristic is the creation of collages of large collections of images where some images are disproportionally larger than the others and positioned in key locations of the original surface. This feature could also be applied in domains where the original surface has defects over, which packing is not to take place.

This article is an extended version of a paper presented at Infocomp 2013 [1]. Here we include new results relating to the performance of the core algorithm, and extend our original results by reporting the execution times of a multithreaded version of the core algorithm running on an Amazon EC2 32-core instance, and packing a billion rectangles.

We are especially interested in avoiding packings that place the larger items concentrated on one side of the surface, and keep covering the remainder of the surface using decreasingly smaller items. These are not aesthetically pleasing packings.

This form of *2D-packing* is a special case of the *2D Orthogonal Packing Problem* (OPP-2), which consists in deciding whether a set of rectangular items can be placed, rotated or not, inside a rectangular surface without overlapping, and such that the uncovered surface area is minimized. In this paper we assume that all dimensions are expressed as integers, and that items cannot be rotated during the packing, which is important if the items are images. 2D-packing problems appear in many areas of manufacturing and technology, including lumber processing, glass cutting, sheet metal cutting, VLSI design, typesetting of newspaper pages, Web-page design or data visualization. Efficient solutions to this problem have direct implications for these industries [2].

Our algorithm packs thousands of items with a competitive efficiency, covering in the high 98 to 99% of the original surface for large collections of items. We provide solutions for several benchmark problems from the literature [3]–[5], and show that our heuristic in some cases generates tighter packings with less wasted space than previously published results, although running slower than the currently fastest solution [6].

To improve the aesthetics of the resulting packing, we use Huang and Chen's [7] surprising quasi-human approach borrowed from masons who pack patios by starting with the corners first, then borders, then inside these limits (similarly to the way one solves a jigsaw puzzle). Our algorithm departs from Huang and Chen's in that it implements a greedy localized best-fit first approach and uses a collection of vertical *lines* containing *segments*. Each vertical segment represents the leftmost side of rectangular area of empty space extending to the rightmost edge of the area to cover. The collection keep the lines ordered by their x-coordinate. All the segments in a line have the same x-coordinate and are ordered by their y-coordinate. Representing empty space in this fashion permits the easy and natural definition of rectangular areas that can be excluded from packing, which in turn offers two distinct advantages: the first is that some rectangular areas can be defined ahead of time as containing images positioned at key locations, and therefore should not be packed over. The second is that subsections of the area to pack can easily be delineated and given to other threads/processes to pack in parallel. Simple scheduling and load-balancing agents are required to allow such processes to exchange items as the packing progresses.

II. The Aesthetics of Photo Collages on a Large Scale

The impetus for this algorithm is to pack a large number of images, typically thousand to millions, in a rectangular surface of a given geometry to form large-scale *collages*. In such applications items are not rotated 90 degrees since they represent images. This type of packing is referred to as *nesting* [8].

Large collages of images are challenging, both because of the packing required, and also because of the required aesthetics. Herr et al. [9] present a large collage of images associated with Wikipedia articles organized as a graph that groups together images based on the similarity of the pages on which they appear. In this context, similar pages are pages that have the same number of shared links. The packing is a simple 2D packing where all images are given the same rectangular frame, and is made to occupy a large circle fit for printing on a poster. In this data visualization, articles are represented by green, blue and yellow disks overlapping each to form clusters around text labels identifying concepts, and all overlay the mosaic of packed images. The aim of this visualization is to present a qualitative aesthetics, rather than to use the packing of variously sized images to convey some quantified relationship. Little effort is made to make the images carry any significant information. The number of images displayed is less than a thousand and already illustrates the challenge of displaying a large collection of images.

On a much larger scale is the packing of the over one billion faces of Facebook users attempted by Natalia Rojas [10]. In this visualization, Rojas presents the visitor of the app.thefacesoffacebook.com page with an approximately 1,200 by 1000 pixel image, where each of the 1.2 million randomly colored pixels represent a Facebook profile image. Packing in this case is straightforward: each rectangle is 1x1 and randomly placed. The visitor of the Web site can zoom in on any of the pixels and is presented by a grid of 100 by 100 pixel images, each image representing an actual profile picture of a Facebook user. The uniform size for the images makes for a trivial packing. It is worth noting that Rojas picked a fairly large 100x100 pixel format for each image, allowing visitors to quickly spot the various faces.

In [11], Wattenberg, Viegas and Hollenbach use *chromograms*, colored fixed-height rectangles aligned in a horizontal bar, to show the edits by users on various Wikipedia pages. While their techique is not a collage, it involves using rectangles of various colors to convey some information pertinent to Wikipedia contents. They cite the large-scale historics containing more than 100,000 events or the irregular structure of the edit logs as significant challenges for making the visualization both effective and aesthetically pleasing.

New developments in display technology that covers walls with video screens and displays billion-pixel digital images have been embraced recently by museum, research centers, and corporate offices. The Cleveland Art Museum [12] is an example where museum goers are presented with a packing of images from the museum collection on a wall of large connected touch-screens. The visitors interact with the display, picking images for additional information. The packing is in bands of same-height images. The result is a pleasing collage of images that are allowed to overlap when the user interacts with them. The Texas Advanced Computing Center's Massive Pixel Environment library [13] allows users to display Processing sketches/citeProcessingOrg over multiple large screen displays. Its use is mostly for visualizing simulation results.

These various efforts are all based in part on the availability of new libraries such as Shiffman's most-pixels-ever Processing package [14], which makes it feasible to display very large images or a large collection of small images, making the problem of packing them efficiently a timely one to address.

Algorithm 1 Simplified Packing Heuristic

```
1: $N$ = dimension( $rects$ )
2: $VL = \{L_0, L_\infty\}$
3: while not $VL$.isempty() do
4:    $success$ = false
5:    for all line $vl$ in $VL$ do
6:       $list$ = { } // empty collection
7:       for all segment $sl$ in $vl$ do
8:          $rect$ = rectangle in $Rects$ with height closest to
            but less than $sl$
9:          if $rect$ not null then
10:            $list$.add( Pair( $rect$, $sl$ ) )
11:         end if
12:      end for
13:      if $list$.isempty() then
14:         continue
15:      end if
16:      sort $list$ in decreasing order of ratio of $rect$.length to
         $sl$.length
17:      for all $pair$ in $list$ do
18:         $rect$, $sl$ = $pair$.split()
19:         if $rect$ fits in $VL$ then
20:            pack $rect$ at the top of $sl$
21:            update $VL$
22:            $success$ = true
23:            break
24:         end if
25:      end for
26:      if $success$ == true then
27:         break
28:      end if
29:   end for
30:   if $Rects$.isEmpty() then
31:      break
32:   end if
33: end while
```

III. Review of the Literature

Possibly because of its importance in many fabrication processes [2], different forms of 2D-packing have evolved and been studied quite extensively since Garey and Johnson categorized this class of problems as NP-hard [15]. It is hence

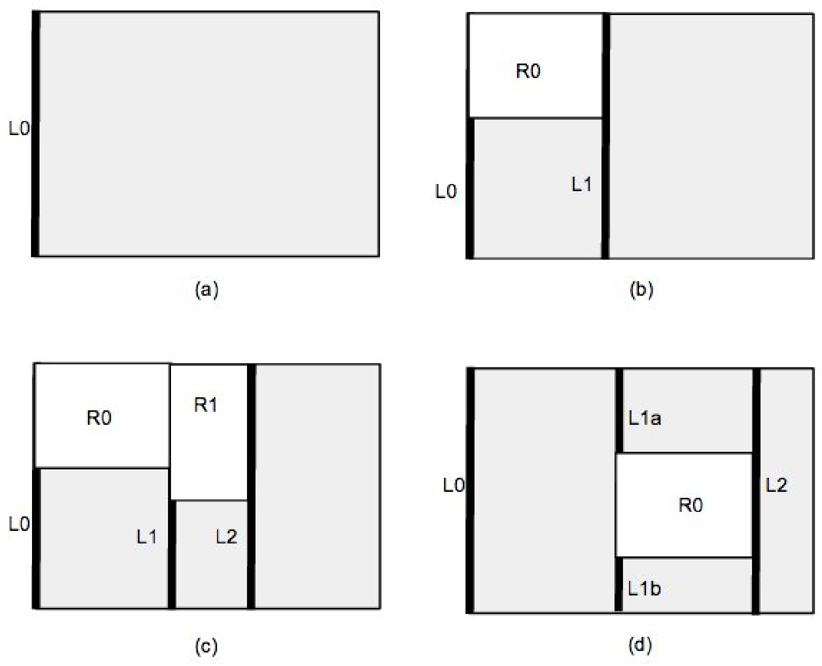

Figure 1. The basic concept of the packing heuristic. (a) The algorithm starts with one vertical line L0 in the left-most position. (b) A rectangle is added, shortening the L0 line and forcing the addition of a second vertical line L1. (c) A second rectangle is added on L1, cutting L1 and forcing the addition of a third line L2. (d) Starting from (a) a rectangle is added in the middle of the available space, creating the addition of a segmented Line L1a, L1b, and a full line L2.

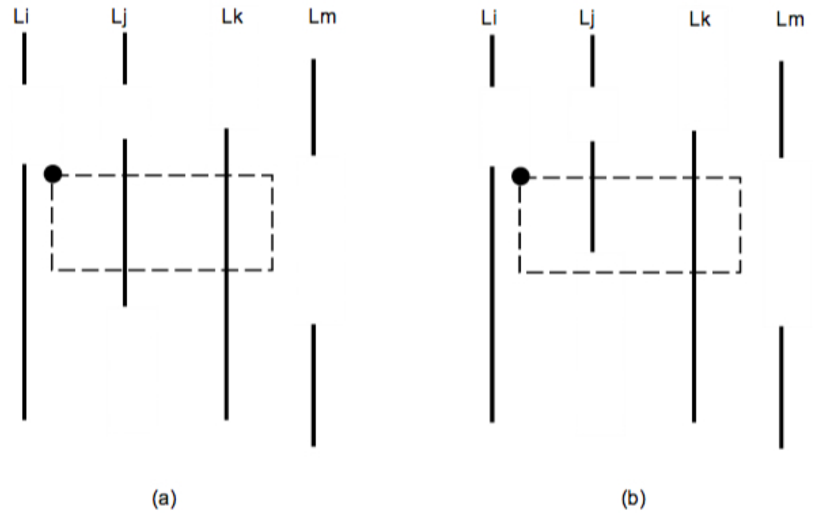

Figure 2. Two examples of potential rectangle placements. In (a) the proposed location for the rectangle (shown in dashed line) is valid and will not intersect with other placed rectangles (not shown) because 1) its horizontal projection on the line L_i directly left of it is fulling included in a segment of L_i, and 2) its intersection with Lines L_j, L_k, and L_m is fully covered by segments of these lines. In (b) the proposed location for the rectangle is not valid, and will result in its overlapping with already placed rectangles since its intersection with Line L_j is not fully included in one of L_j's segments.

a challenge to create a comprehensive review of the literature, as any 2-dimensional arranging of rectangular items in a rectangular surface can be characterized as packing. Burke, Kendall and Whitwell [3] and Verstichel, De Causmaecker, and Vanden Berghe [16] provide among the best encompassing surveys of the literature on 2D-packing and strip-packing research.

While exact solutions are non-polynomial in nature and slow, researchers have achieved optimal solutions for small problem sizes. Baldacci and Boschetti, for example, reports four known approaches to the particular problem of 2D orthogonal non-guillotine cutting problem [17], Beasley's optimal algorithm [18] probably being the one most often cited. Unfortunately such approaches work well on rather small problem sets. Baldacci and Boschetti, for example, report execution times in the order of tens of milliseconds to tens of seconds for problem sets of size less than 100 on a 2GHz Pentium processor.

Scientists from the theory and operations-research communities have also delved on 2D-packing and have generated close to optimal solutions [19], [20]. The *Bottom-Left* heuristic using rectangles sorted by decreasing width has been used in various situations yielding different asymptotic relative performance guarantees [21]–[24]. Other approaches concentrate on local search methods and lead to good solutions in practice, although computationally expensive. *Genetic algorithms*, *tabu search*, *hill-climbing*, and *simulated annealing* [25], [26] are interesting techniques that have been detailed by Hopper and Turton [2], [4]. These meta-heuristics have heavy computational complexities and have been outperformed recently by simpler best-fit based approaches, including those of Hwang and Chen [5], [7], or Burke, Kendall and Whitwell [3]. Huang and Chen show that placement heuristics such as their *quasi-human* approach inspired by Chinese masons outperforms the meta-heuristics in minimizing uncovered surfaces in many cases, although requiring relatively long execution times. Burke et al. propose a best-fit heuristic that is a close competitor in the minimization of the uncovered surface but with faster execution times.

Probably the fastest algorithm to date is that of Imahori and Yagiura [6], which is based on Burke et al.'s best-fit approach. Their algorithm is very efficient and requires linear space and $O(n \log n)$ time, and solves strip-packing problems where the height of the surface to pack can expand infinitely until all items are packed. They report execution times in the order of 10 seconds for problems of size 2^{20} items. Our serial application is slower, as our timing results show below, but provide a better qualitative aesthetic packing in a fixed size surface with similarly small wasted area. A multithreaded version of our heuristic, however, will pack a million rectangles under 4 seconds, and is presented in details in Section VII. Furthermore, the ability to pack around rectangular areas make for easy parallelization of the algorithm, as we illustrate below. Because the time consuming operation of a collage of image is in the resizing and merging of images on the canvas that vastly surpasses our packing time by several orders of magnitude, the added value of the quality of the aesthetics of the packing makes our algorithm none-the-less attractive compared to the above cited faster contenders.

In the next section we present the algorithm, its basic data structure, and an important proposition that controls the packing and ensures the positioning of items without overlap. We follow with an analysis of the time and space complexities of our algorithm, and show that the algorithm uses linear space and requires at most $O(N^3 \log(N)^2)$ time, although experimental results show closer to quadratic evolution of the execution times. This is due to the fact that the algorithm generally finds a rectangle to pack in the first few steps of the process, and the execution time is proportional mostly to the number of rectangles. Only the last few remaining rectangles take the longest amount of time to pack in the left over space. We compare our algorithm to several test cases taken from the literature in the benchmark section, and close with several examples illustrating how the algorithm operates. We then take the core packing loop and show that by subdividing the area to pack into thin horizontal bands, the speed of the packing can be significantly sped up, making the packing of billions of rectangles possible in the space of minutes. The conclusion section presents possible improvements and future research areas.

IV. The Algorithm

A. Basic Data-Structures

The algorithm is a *greedy*, *localized best-fit* algorithm that finds the best fitting rectangles to pack closest to either one of the left side or top side of the surface. Figure 1 captures the essence of the algorithm and how it progresses.

The algorithm maintains ordered collections of vertical *segments* representing rectangular areas of empty space. Segments are vertical but could also be made horizontal without impeding the operation of the algorithm. These vertical segments can be thought of as the left-most height of a rectangle extending to the right-most edge of the surface to pack. Vertical segments with the same x-coordinate relative to the top-left corner of the surface to cover are kept in vertical *lines*. The algorithm's main data structure is thus a *collection* of lines ordered by their x-coordinates, each line itself a collection of segments, also ordered by their y-coordinates. The collections are selected to allow efficient *exact searching*, *approximate searching* returning the closest item to a given coordinate, *inserting* a new item (line or segment) while maintain the sorted order. *Red-black trees* [27] are good implementations for these collections.

The main property on which the algorithm relies to position a new rectangle on the surface without creating an overlap with already positioned rectangles is expressed by the following proposition:

Proposition 1: A new rectangle can be positioned in the surface such that its top-left corner falls on the point of coordinates (x_{tl}, y_{tl}) and such that it will not intersect with already positioned rectangles if it satisfies two properties relative to the set of vertical lines:

1) Let L_{left} be the vertical line whose x-coordinate x_{left} is the floor of x_{tl}, i.e., the largest x such that $x <= x_{tl}$. In other words, L_{left} is the vertical line the closest to or touching the left side of the rectangle. For the rectangle to have a chance to fit at its present location, the horizontal projection of the rectangle on L_{left} must intersect with one of its segments that completely contains this projection.
2) The horizontal projection of the rectangle on *any* vertical line that intersects it must also be completely included in a segment of this line.

Figure 2 illustrates this proposition.

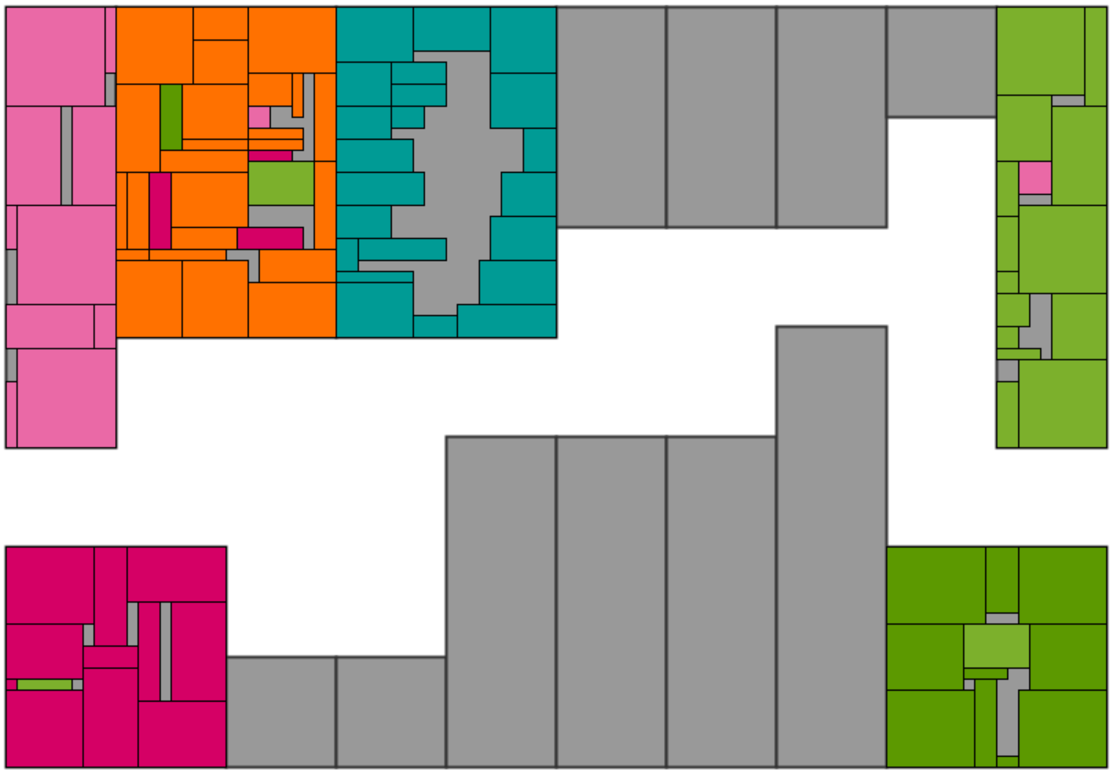

Figure 3. A solution generated by our algorithm for the packing of 100 items in 16 objects as proposed by Hopper as the “M1a” case.

B. Basic Operation

The algorithm starts with two vertical lines, L_0 and L_∞. The first line originates at the top-left corner of the surface to cover, and contains a single segment whose length defines the full *height* of the surface to pack. L_∞ is a vertical line located at an x-coordinate equal to the width of the surface to pack. L_∞ contains no segments. It identifies the end of the area to pack. Any rectangle that extends past the end of the area to cover will cross L_∞, and because this one does not contain segment, the second part of the proposition above will reject the rectangle.

To simplify the description of the algorithm, we use the generic term *line* to refer to lines and segments. The algorithm packs from left to right, and favors top rather than down locations. Starting with the vertical line L_0 it finds the item R_0 with the largest height less than L_0. If several items have identical largest height, the algorithm picks the one with the largest perimeter and tests whether it can be positioned without overlapping any other already placed items. The algorithm tries three different locations: at the top of L_0, at the bottom of L_0, or at the centre of L_0. The item is positioned at the first location that offers no overlap, otherwise the next best-fitting item is tested, and so on.

The positioning of R_0 shortens L_0, as shown in Figure 1(b). A new line L_1 is added to the right of R_0 to indicate a new band of space to its right that is free for packing.

The goal is to place all larger items first and automatically the smaller ones find places in between the larger ones.

In Figure 1(c), the algorithm finds R_1 as the rectangle whose width is the largest less than L_1 and positions it against the left most part of L_1. Adding R_1 shortens L_1, indicating that all the space right of the now shorter L_1 is free for packing. Again, a new line L_2 is added to delineate a band of empty space to the right of R_1.

We implement the data-structures for the lines as trees sorted on the line position relative to the top-left corner of the initial surface, so that a line or a group of lines perpendicular to particular length along the width or height of the original surface can be quickly found.

Note that in our context these line-based data-structures allow for the easy random positioning of rectangles in the surface before the packing starts, as illustrated in Figure 1(d) where a rectangle R_0 is placed first in the middle of the surface before the packing starts.

C. The Code and its Time and Space Complexities

We now proceed to evaluate the time complexity of our heuristic, whose algorithmic description is given in Algorithm 1. In it, N is the number of items to pack, $rects$ is the list of items to pack, VL the collection of vertical lines, and vl one such individual line.

Since N is the original number of items to pack, then clearly the size of VL is $O(N)$. Given a line vl of VL, we argue that the average number of segments it contains (exemplified by

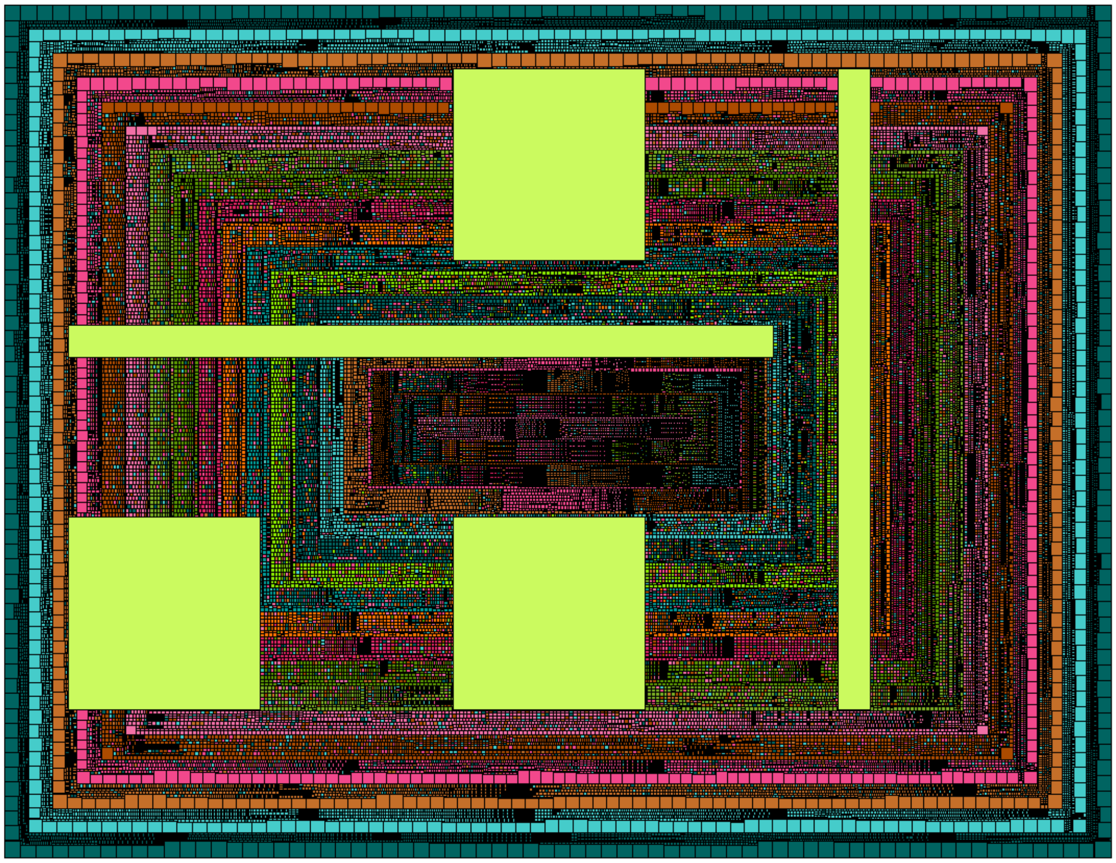

Figure 4. The packing of 97,272 randomly generated items in a a rectangular surface. The application is multithreaded, each thread associated with a rectangular border. 5 large lime-green rectangles with different geometries are placed in various locations before the computation starts.

$L1a$ and $L1b$ in Figure 1) is $O(N)$. The goal of the Loop starting at Line 3 is to pack all rectangles, and it will repeat N times, hence $O(N)$. The combined time complexity of the loops at Lines 5 and 7 is $O(N)$ because they touch at most all segments in all the lines, which is bounded by $O(N)$. The time complexity of Line 16 is clearly $O(N \log N)$, although on the average the number of pairs to sort will be $O(\sqrt{N})$ rather than $O(N \log N)$. The loop starting at Line 17 processes at most $O(N)$ pairs, and for each rectangle in it, must compare it to at most $O(N)$ line vl. So it contributes $O(N^2)$, which overpowers the sorting of the list. Therefore, the combined complexity of the whole loop starting on Line 3 is $O(N^3)$.

Empirically, however, the algorithm evolves in quasi quadratic fashion as illustrated in Figure 6, where various selections of rectangles with randomly set dimension are packed in a rectangular surface that is selected ahead of time to be of a given aspect ratio, and whose total area is 1% larger than the sum of all the items to pack. We found this approach the best for packing quickly. The dimensions of the randomly-sized rectangles for all the experiments reported here are computed by the following equations:

$$\begin{aligned} width &= max(20, RandInt(500)) \\ height &= max(20, RandInt(500)) \end{aligned}$$

where $RandInt()$ returns a random integer between 0 and 500, excluded. This translate in 230,400 uniformly distributed possible geometries. Remember that we do not allow for rectangles to rotate, so all geometries are unique.

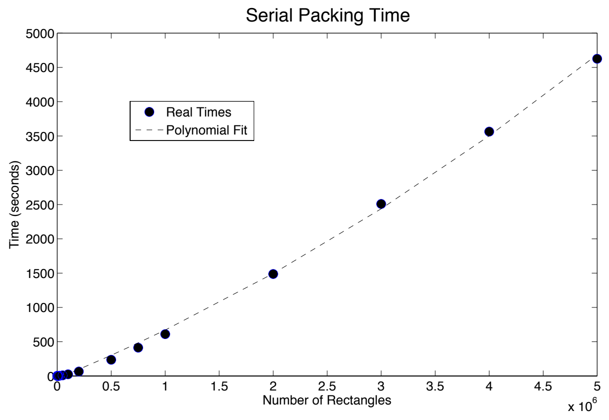

Figure 6. Running times and regression fits for packings of 100 to 5,000,000 random rectangles on one core of a 3.5 GHz 64-bit AMD 8-core processor.

Note that the times reported are user times, and that

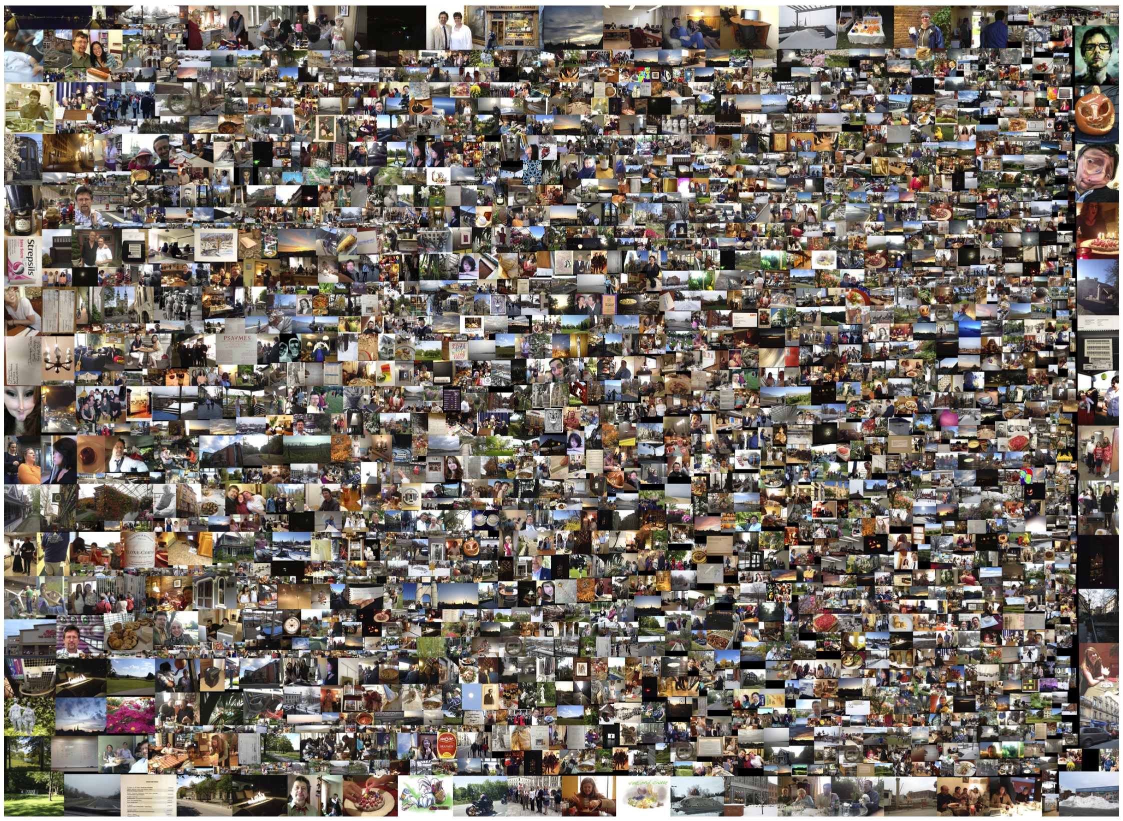

Figure 5. The packing of 2,200 photos of various sizes and aspect-ratios, as many are cropped for artistic quality. The size of the photos is randomly picked by the algorithm. All the photos belong to this author.

only one core of the processor is used, corresponding to a totally serial execution time. A second-degree polynomial fit of the measured times is shown. The fit has equation $y = 5.985\ 10^{-11}\ x^2 + 6.447\ 10^{-4}\ x - 3.514\ 10^1$, with a correlation coefficient r^2 of 0.99898, and a standard error of 49.263.

The space complexity is clearly $O(N^2)$, since the addition of a rectangle to a group of R already packed rectangles will cut at most R lines and introduce at most R new segments. The cumulative effect results in a quadratic variation of the number of segments.

D. Algorithmic Features

Our heuristic sports one feature that is key for our image-collage application: Rectangular areas in which packing is forbidden can easily be identified inside the main surface to be packed, either statically before starting the packing or even dynamically during run time. We refer to these areas as *empty zones*. This feature offers the user the option of positioning interesting images at key positions on the surface to be packed ahead of time. In other domains of application these could be areas with defects. Additionally, it allows parallel packing approaches where rectangular empty zones can be given out to new processes to pack in parallel, possibly shortening the execution time.

V. BENCHMARKS

A set of benchmark cases used frequently in the literature are those of Hopper and Turton [4], and of Burke, Kendall and Whitwell [3]. For the sake of brevity we select a sample of representative cases and run our heuristic on each one. The computer used to run the test is one core of a 64-bit Ubuntu machine driven by a 3.5GHz AMD 8-core processor, with 16GB of ram. The heuristic is coded in Java. Note that all published results do not always provide a derivation of the time complexity of the heuristic presented, and the goodness of the algorithm is measured by its execution time on various benchmark cases. Unfortunately, all experiments are run are on different types of computers, ranging from ageing memory-limited laptops to supped up desktops, all with different processor speed and memory capacities. To provide a more objective comparison, we make the following assumptions: *a*) all results reported in the literature corresponded to compiled applications that are all memory residents, *b*) they are the only workload running on the system, *c*) MIPS are linearly related to CPU frequency, and thus we scale the execution times of already published data reported by the ratio of their operating CPU frequencies to that of our processor (3.5GHz).

We follow the same procedure used by the researchers whose algorithms we compare ours to, and we run our application multiple times (in our case 30 times) on the same

TABLE I. PERFORMANCE COMPARISON TABLE

			Burke		GRASP		3-way		DT	
Case	Number items	optimal height	diff.	time (s)	diff.	time (s)	diff.	time (s)	diff.	time (s)
N1	10	40	0	∼14.571	0	∼34.286	5	<0.009	0	0.05
N2	20	50	0	∼14.571	0	∼34.286	3	<0.009	6	<0.01
N3	30	50	1	∼14.571	1	∼34.286	4	<0.009	10	<0.01
N4	40	80	2	∼14.571	1	∼34.286	6	<0.009	49	<0.01
N5	50	100	3	∼14.571	2	∼34.286	4	<0.009	5	0.03
N6	60	100	2	∼14.571	1	∼34.286	2	<0.009	22	0.01
N7	70	100	4	∼14.571	1	∼34.286	7	<0.009	14	<0.01
N8	80	80	2	∼14.571	1	∼34.286	3	<0.009	23	<0.01
N9	100	150	2	∼14.571	1	∼34.286	13	<0.009	5	0.04
N10	200	150	2	∼14.571	1	∼34.286	2	0.01	10	0.03
N11	300	150	3	∼14.571	1	∼34.286	2	0.01	2	0.49
N12	500	300	6	∼14.571	3	∼34.286	5	0.02	7	0.07
N13	3152	960	4	∼14.571	3	∼34.286	4	0.20	5	0.927
C7-P1	196	240	4	∼14.571	4	∼34.286	6	<0.009	17	0.02
C7-P2	197	240	4	∼14.571	3	∼34.286	4	<0.009	41	0.02
C7-P3	196	240	5	∼14.571	3	∼34.286	5	<0.009	24	0.01

problem set and keep the best result.

Table I shows the scaled execution times of the various heuristics for problem sets taken from the literature. Column 1 identifies the various cases from Burke et al. [3], with the number of items packed in Column 2, and the optimal height of the packing in Column 3. The difference between the resulting height of algorithm's packing and optimal along with the execution time in seconds are shown for each of 4 algorithms, in Columns 4-5, 6-7, 8-9, and 10-11. Our heuristic's data covers the last two columns. The times are those reported in the literature multiplied by a scaling factor equal to the 3.5GHz/*speed of processor*, where the processor is the one used by the researchers. For the Burke column, the speed of the processor is 850MHz. For the GRASP column, 2GHz, and for the 3-way column, 3GHz.

We observe that, as previously discovered [6] our packing efficiency improves as the number of items gets larger (in the thousand of items), which is the size of our domain of interest. The execution times of our heuristic are faster than those of Burke's best-fit, or of GRASP, and at most five times slower than the fast running 3-way best-fit of Imahori and Yagiur [6]. This difference might be attributed to either the choice of language used to code the algorithm, Java in our case, versus C for theirs.

VI. PERFORMANCE FOR LARGE SCALE PACKING

A. Subdividing the space into horizontal bands

In this section, we report on experiments conducted on a modified version of our heuristic where we skip the packing of the corners and borders first, and divide the rectangular packing area in small non overlapping horizontal bands of the same length as the large area to pack. We have found that limiting the packing to smaller bands significantly decreases the packing time by reducing the size of the Red-Black tree data-structures holding all the lines and segments. Typically, the area to pack is divided in 256, 512, 1024 or more horizontal bands as long as all the rectangles can be packed in the given area (small-height bands decrease the packing efficiency). For packing 1,000,000 rectangles without dividing the space into band requires 603 seconds on one core of our 3.5 GHz processor. Dividing the space into 256 bands and packing each band one after the other, serially, on one core, brings the user execution time to 4.03 seconds and maintains a packing efficiency greater than 99%. Figure 7 shows the user-times in seconds for packing 1,000,000 randomly selected rectangles when the area to pack is divided in 1, 2, 4, 8, ... 256 bands. Note that when we increase the number of bands by two, the execution time is almost halved. Therefore, applications that require high-

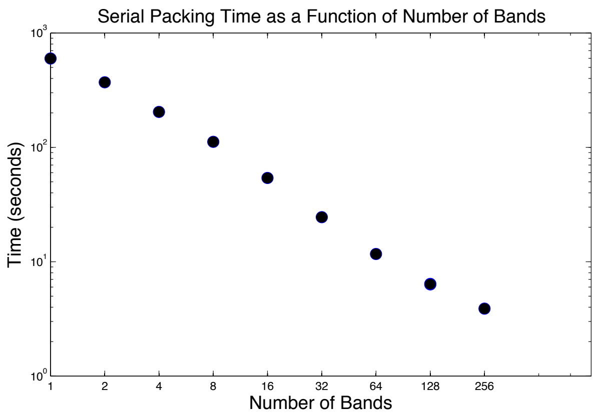

Figure 7. Running times for packing 1,000,000 rectangles with 1 to 256 bands dividing the packing area. User times measured on one core of a 3.5 GHz 64-bit AMD 8-core processor.

speed 2D-packing should organize the area to pack in as many narrow horizontal bands as possible while maintaining a target efficiency.

In the next section, we show several packings generated by our heuristic.

VII. Packing Examples

In this section, we provide several examples of packing under various conditions and constraints, some of them taken from the literature.

In Figure 3, we apply our heuristic to Hopper's *M1a case* [2] where 100 items must be packed into 16 different objects. Our algorithm also packs the objects, although this is not a requirement of the test. In this experiment, our heuristic is multithreaded and several threads pack the different objects. A scheduler simply distributes the objects to separate threads, picking the largest object first and assigning it to a new thread implementing our packing heuristic. Then the scheduler picks the next largest object (in terms of its area) and assigns it to a new thread, and so on. The earliest starting threads are given a random sample of the items to pack. Threads that start last have to wait until earlier threads finish packing and return items that could not be packed. This automatically packs objects in such a way that as few objects as possible are packed, and some left empty, which may be desirable.

In Figure 4, the original surface is divided at run time into smaller surfaces, or *borders* one inside the other as the packing progresses, and individual threads are running the packing on individual borders. Here again the threads are given random samples of the original population of items and a load balancing scheme allows for the exchange of items between threads. This is represented by items with different colors. For example, the items associated with the first thread are all dark green, and some can be found in the light green, orange or pink borders as they are rejected by the first thread once it has packed the dark green band. Note that the utilization of the surface is 99.30%.

In Figure 4, we have placed five large items (yellow-green rectangles) on the surface before launching the packing algorithm. Notice how the heuristic naturally packs around these areas. Note also that as in Figures 3 and 4, we follow Huang and Chen's quasi-human approach [7] and pack corners and borders first before proceeding with the inside areas. Note that this modification of the algorithm fits completely with the natural properties of the heuristic, and enhances the visual aspect of the final packing.

VIII. 2D-Packing a Billion Random Rectangles

The ability to divide the rectangular area to pack into thin horizontal bands leads us to consider the challenge of packing a billion rectangles with random dimensions. The main obstacle to solve this problem is not in a long execution time, but rests in the ability to keep a billion objects in random-access memory (RAM). Given that a rectangle is defined by a geometry requiring a minimum of two *longs* for the coordinates, and two *ints* for the dimensions, at least 24 to 32 Gigabytes of storage are required to store a billion such objects, depending on whether the application runs on a 32 or 64-bit system. If packing speed is of the essence, then the objects must reside in memory.

To keep the computation CPU-bound as well as RAM-bound, and measure the best possible packing performance, we multi-thread our packing heuristic and run it on an Amazon *c3.8xlarge* instance, which, at the time of this writing, sports the following characteristics:

- **CPU Architecture:** 64 bits.
- **Cores:** 32 hyperthreaded 2.8 GHz Intel Xeon E5-2680v2 cores (Ivy Bridge).
- **Performance:** Combined CPU speed equivalent to 108 *m1.small* Amazon instances. An m1.small typically has the same performance as a 1.0-1.2 GHz 2007 Intell Opteron or 2007 Xeon processor.
- **RAM:** 60 Gigabytes.
- **Disk Storage:** two 320-GB Solid-Stated Device disks.
- **Network Speed:** 10 GBits.

Note that the c3.8xlarge processor frequency can be turbo-boosted to 3.6 GHz if enough thermal room is available.

A. Multithreaded Algorithm

We adopt a simple scheduling and load-balancing of the different threads. Assuming that there are N rectangles to pack and that the area to cover is divided into B bands, we assign each one to a single thread, and we run in parallel the packing of the first $B-1$ bands. We perform a *join* operation on all the running threads. We allocate $alpha\ N/B$ rectangles ($alpha > 1$) to each thread so that the packing can benefit from a greater collection of rectangles than what can fit during the packing. We have found that $alpha$ = 1.01 yields good packing efficiencies. When the first $B-1$ bands have been packed, the threads return the rectangles that could not be packed and these are returned to the pool of unpacked items. This collection, along with all remaining unallocated rectangles, is given to a final thread that packs the last band.

The algorithm is detailed in Algorithm 2 below.

B. Execution Time

Dividing the total area to be covered in 4096 bands, and launching the multithreaded algorithm to pack a billion rectangles on an Amazon c3-8xlarge instance takes **8 minutes and 56 seconds of user time**. For comparison, dividing the space in 2048 bands, instead, results in a user time of 11 minutes and 52 seconds. Packing 1 million rectangles in 256 bands now takes only 3.2 seconds. We keep the RAM usage low by observing that since we need randomly sized rectangles, they do not need to be stored ahead of time, but instead they can be generated on the fly as they are passed to each thread. Only the rectangles that have been packed and assigned coordinates relative to each band's top-left corner are kept.

Algorithm 2 Multithreaded Packing Scheduling and Load-Balancing

```
1: $rects$ = list of all rectangles to pack
2: $N$ = dimension( $rects$ )
3: $B$ = dimension( $bands$ )
4: $noBandsPacked$ = 0
5: $\alpha$ = 1.01
6: while $noBandsPacked < B$-1 do
7:     create Thread $t_i$ for Band $b_i$
8:     allocate ($N/B * \alpha$) rectangles to $t_i$
9:     $rects$.remove( all allocated rectangles )
10:    start $t_i$
11:    $noBandsPacked \leftarrow noBandsPacked + 1$
12: end while
13: join on all threads $t_i$
14: for all joined thread $t_i$ do
15:    $rects$.append( rectangles unpacked by $t_i$ )
16: end for
17: create Thread $t_{B-1}$
18: allocate $rects$ to $t_{B-1}$
19: start $t_{B-1}$
20: join on $t_{B-1}$
```

C. New Application Domains

The ability to quickly pack large collections of rectangles opens applications based on 2D-packing to the realm of *real-time* and *interactive* implementations.

It is now conceivable that a user may interact with a large display, say in a museum showing its entire collection as a packing of images, pick one or a group of images and have the application automatically remove, reposition, or resize them, quickly refreshing the space around or underneath them with a new 2D-packing. This new feature requires the implementation of fast data structures for quickly locating packed rectangles inside an area or overlapping a given point. *R-trees* [28], which maintain groupings of objects in space by geographical closeness, offer interesting possibilities, and are the subject of ongoing research.

IX. Conclusions

We have presented a new heuristic for packing or nesting two-dimensional images in a rectangular surface. The heuristic packs the items by creating a collection of segments that are maintained in two data structures, one for horizontal segments, and one for vertical segments. The segments represent the leftmost and topmost side of rectangular surfaces that extend to the edges of the original surface to pack. These data structures permit to test quickly whether a new item can be positioned in the surface without overlapping a previously placed item.

Our packing heuristic does not rotate items, but none-the-less compares favourably with other heuristics published in the literature that solve 2D-strip packing with rotation of items allowed. If rotation of items is required, a possible modification of the algorithm is to give a packing thread two versions of the same rectangle, one the 90-degree rotated version of the other, both linked to each other. Whenever one of the versions is packed, the algorithm quickly searches its list of unpacked items and removes the item linked to the one just packed.

The data structure used to maintain the empty areas lends itself well to positioning items in key places ahead of time, or in subdividing the original surface into multiple holes that can be either left empty, reserved for large size items, or assigned to separate processes that will pack in parallel. Such holes may contain defects (for example in a sheet of metal, or glass) that need to be avoided by the packing process.

It is possible to significantly speed the core packing algorithm up by slicing the area to cover into individual thin horizontal rectangular bands. This limits the amount of searching for the best fitting place for the next rectangle, and the execution time drops inversely proportionally with the width of the bands.

If the slicing of the packing area creates undesirable horizontal dividing lines on which rectangles align themselves during the packing, one can easily pre-pack small rectangles over the boundaries of bands, hiding in effect the dividing line.

Because our domain of application is that of image collages, we have found that the the quasi-human approach of Huang and Chen, along with subdividing the surface into nested rectangular area significantly improves the aesthetic quality of the packing compared to most heuristic that privilege one side or corner and put all largest items there and finish packing with the smaller items at the opposite end.

References

[1] D. Thiebaut, “2d-packing images on a large scale,” in Proceedings of INFOCOMP 2013, Nov 17-22 2013, pp. 19–26.

[2] E. Hopper, “Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods,” Ph.D. dissertation, Cardiff University, United Kingdom, 2000.

[3] E. K. Burke, G. Kendall, and G. Whitwell, “A new placement heuristic for the orthogonal stock-cutting problem,” Oper. Res., vol. 52, no. 4, Aug. 2004, pp. 655–671.

[4] 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, vol. 1, no. 128, 2000, pp. 34–57.

[5] W. Huang, D. Chen, and R. Xu, “A new heuristic algorithm for rectangle packing,” Computers and Operations Research, vol. 34, no. 11, November 2007, pp. 3270–3280.

[6] S. Imahori and M. Yagiur, “The best-fit heuristic for the rectangular strip packing problem: An efficient implementation and the worst-case approximation ratio,” Comput. Oper. Res., vol. 37, no. 2, Feb. 2010, pp. 325–333.

[7] W. Huang and D. Chen. Simulated annealing. Accessed 05/12/2014. [Online]. Available: http://cdn.intechopen.com/pdfs/4629/InTech-An_efficient_quasi_human_heuristic_algorithm_for_solving_the_rectangle_packing_problem.pdf [retrieved: July, 2008]

[8] R. D. Dietrich and S. J. Yakowitz, “A rule-based approach to the trim-loss problem,” International Journal of Production Research, vol. 29, 1991, pp. 401–415.

[9] B. W. Herr, T. Holloway, E. F. Hardy, K. W. Boyack, and K. Brner, “Science-related wikipedian activity,” 3rd Iteration (2007): The Power of Forecasts: Places and Spaces: Mapping Science, 2007.

[10] N. Rojas. The faces of facebook. Accessed 5/12/2014. [Online]. Available: http://app.thefacesoffacebook.com/ (2013)

[11] M. Wattenberg, F. Vigas, and K. Hollenbach, "Visualizing activity on wikipedia with chromograms," in Human-Computer Interaction INTERACT 2007, ser. Lecture Notes in Computer Science, C. Baranauskas, P. Palanque, J. Abascal, and S. Barbosa, Eds. Springer Berlin Heidelberg, 2007, vol. 4663, pp. 272–287. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74800-7_23

[12] Collection wall. Accessed 5/12/2014. [Online]. Available: http://www.clevelandart.org/gallery-one/collection-wall (2013)

[13] B. Westing, H. Nieto, H. Nieto, and K. Gaither, "Massivepixelenvironment: A tool for rapid development with distributed displays," in CHI 2013, 2013.

[14] D. Shiffman. Most pixels ever. [Online]. Available: https://github.com/shiffman/Most-Pixels-Ever-Processing/wiki (2008)

[15] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. New York: W. H. Freeman and Company, 1979.

[16] J. Verstichel, P. D. Causmaecker, and G. V. Berghe, "An improved best fit heuristic for the orthogonal strip packing problem," International Transactions in Operational Research, no. 20, June 2013, pp. 711–730.

[17] R. Baldacci and M. A. Boschetti, "A cutting-plane approach for the two-dimensional orthogonal non-guillotine cutting problem," European Journal of Operational Research, vol. 183, no. 3, 2007, pp. 1136–1149, accessed 5/12/2014.

[18] J. Beasley, "An exact two-dimensional non-guillotine cutting tree search procedure," Operations Research, vol. 33, no. 1, January 1985, pp. 49–64.

[19] E. G. Coffman, M. R. Gazey, and D. S. Johnson, Approximation algorithms for bin-packing an updated survey. Springer-Verlag, 1984.

[20] H. Dyckhoff, "Typology of cutting and packing problems," European Journal of Operational Research, vol. 44, 1990, pp. 145–159.

[21] B. S. Baker, E. G. C. Jr., and R. L. Rivest, "Orthogonal packings in two dimensions," SIAM Journal on Computing, vol. 9, 1980, pp. 846–855.

[22] B. S. Baker, D. J. Brown, and H. P. Katseff, "A 5/4 algorithm for two-dimensional packing," Journal of Algorithms, vol. 2, 1981, pp. 348–368.

[23] D. Sleator, "A 2.5 times optimal algorithm for packing in two dimensions," Information Processing Letter, vol. 10, 1980, pp. 37–40.

[24] C. Kenyon and E. Remilia, "Approximate strip-packing," in Proceedings of the 37th Annual Symposium on Foundations of Computer Science, 1996, pp. 31–35.

[25] D. Liu and H. Teng, "An improved bl-algorithm for genetic algorithm of the orthogonal packing of rectangles," European Journal of Operational Research, vol. 112, no. 2, January 1999, pp. 413–420.

[26] T. W. Leung, C. K. Chan, and M. Troutt, "Application of a mixed simulated annealing genetic algorithm heuristic for the two-dimensional orthogonal packing problem," European Journal of Operational Research, vol. 145, no. 3, March 2003, pp. 530–542.

[27] L. J. Guibas and R. Sedgewick, "A dichromatic framework for balanced trees," Foundations of Computer Science, IEEE Annual Symposium on, vol. 0, 1978, pp. 8–21, accessed 5/12/2014.

[28] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, ser. SIGMOD '84. New York, NY, USA: ACM, 1984, pp. 47–57.