ELSEVIER

Computational Geometry 22 (2002) 205–222

Computational Geometry
Theory and Applications

www.elsevier.com/locate/comgeo

Densest translational lattice packing of non-convex polygons ☆

Victor J. Milenkovic [1]

University of Miami, Department of Computer Science, P.O. Box 248154, Coral Gables, FL 33124-4245, USA

Communicated by S. Fortune; received 1 July 2000; accepted 20 February 2001

Abstract

A *translational lattice packing* of k polygons $P_1, P_2, P_3, \ldots, P_k$ is a (non-overlapping) packing of the k polygons which is replicated without overlap at each point of a lattice $i_0 v_0 + i_1 v_1$, where v_0 and v_1 are vectors generating the lattice and i_0 and i_1 range over all integers. A *densest* translational lattice packing is one which minimizes the area $|v_0 \times v_1|$ of the fundamental parallelogram. An algorithm and implementation is given for *densest translational lattice packing.* This algorithm has useful applications in industry, particularly clothing manufacture. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Layout; Packing; Nesting; Lattice packing; Linear programming; Quadratic programming

1. Introduction

A number of industries generate new parts by cutting them from stock material: cloth, leather (hides), sheet metal, glass, etc. These industries need to generate dense non-overlapping layouts of polygonal shapes. Because fabric has a grain, apparel layouts usually permit only a finite set of orientations. Since cloth comes in rolls, the most common layout problem in the apparel industry is *strip packing*: find a layout of polygons $P_1, P_2, \ldots, P_k$ in a rectangular container of fixed width and minimum length. Strip packing is a special case of the more general *minimum enclosure* problem in which both dimensions can vary and the goal is to minimize the area of the rectangle.

The values of k are often in the dozens or even hundreds. Unfortunately, even the purely translational version of strip packing is NP-hard, and therefore one has to expect the running time of a packing algorithm to be exponential in k. In the apparel industry, no layout software has yet replaced a human. The best hope is to develop good minimum enclosure algorithms for small k and then use these algorithms as

☆ Expanded version of paper presented at the 16th Annual ACM Symposium on Computational Geometry (Hong Kong, June 2000).

E-mail address: vjm@cs.miami.edu (V.J. Milenkovic).

URL address: http://www.cs.miami.edu/~vjm (V.J. Milenkovic).

[1] This research was funded by the Alfred P. Sloan Foundation through a subcontract from the Harvard Center for Textile and Apparel Research and by NSF grant NSF-CCR-97-12401.

PII: S0925-7721(01)00051-7

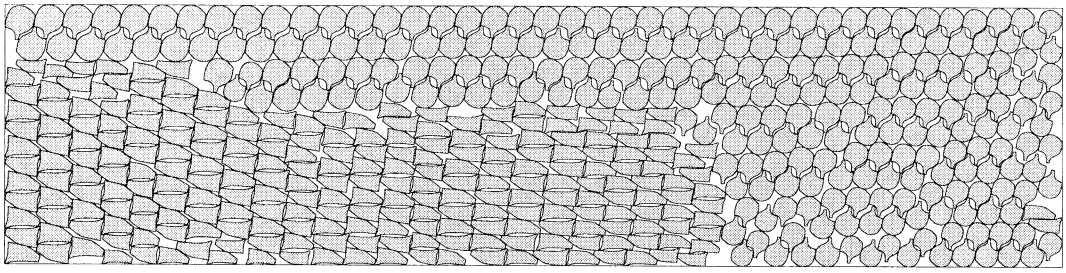

Fig. 1. Human-generated layout of parts for 123 brassieres.

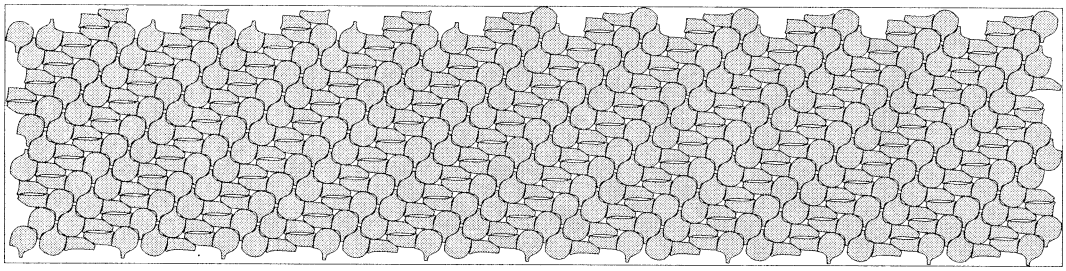

Fig. 2. Densest lattice packing "clipped" to the same rectangle.

part of heuristics for much larger k. Much work has been done in this direction, and this paper follows this philosophy.

Sometimes the large set of polygons consists of a replication of a much smaller set of identical or nearly identical polygons which are much smaller than the container. For example, Fig. 1 shows a layout of 492 parts to make 123 brassieres. There are only two types of part: a cup and a strap. Half of the parts of each type are rotated 180 degrees, creating four different types of parts for the strictly translational problem. As one can see, different regions are approximately *double-lattice packings*: a periodic packing of a part and its 180 degree rotated twin.

Fig. 2 depicts the densest periodic packing of four polygons: a cup, a strap, and the 180 degree rotations of each. An arbitrary translation of the infinite periodic packing is "clipped" to the same rectangle as the human-generated layout. As it happens, this layout contains 7 more cups and 2 more straps than the layout in Fig. 1. This example clearly demonstrates that a solution to lattice packing for small k, such as $k = 4$, might be very useful for generating layouts for very large k, such as $k = 492$.

Why did not the human start with a periodic packing involving all four parts? It is likely that this was beyond the human capability. It is also beyond the ability of current algorithms and heuristics. Indeed, the subject of this paper is how we generated Fig. 2.

Of course, we are ignoring two aspects of the actual layout problem. First, the cloth is not the infinite plane: it has boundaries, and we need to figure out the best way to "clip" the lattice to a strip or rectangle. Second, the parts vary somewhat in size and shape, as they do in this layout of brassieres, because the manufacturers need to generate different sizes for different people. These aspects will have to be dealt with by new algorithms and heuristics. We discuss these possible extensions in the final section of this

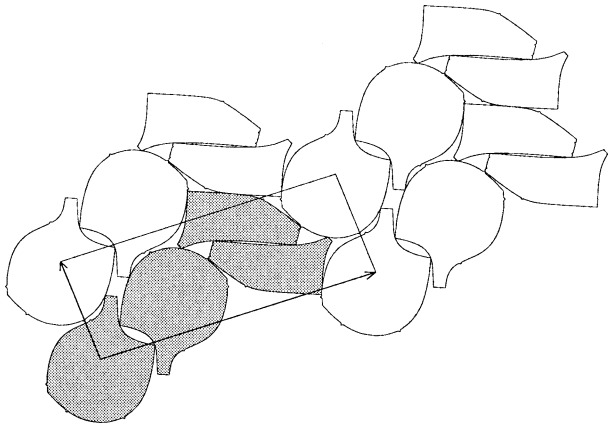

Fig. 3. Four polygons (grey), lattice packing, fundamental parallelogram, and lattice generators (dark arrows).

paper. In the meantime, it is clear that an algorithm for densest lattice packing is a big step toward solving an important industrial problem.

1.1. New results

A *translational lattice packing* of k polygons $P_1, P_2, P_3, \ldots, P_k$ is a (non-overlapping) packing of the k polygons which is replicated without overlap at each point of a lattice $i_0 v_0 + i_1 v_1$, where v_0 and v_1 are vectors generating the lattice and i_0 and i_1 range over all integers. Fig. 3 is a close-up of a lattice packing (not the one in Fig. 2). The grey polygons are the four polygons to be packed. Dark arrows depict the two generating vectors v_0 and v_1. Three copies of the grey polygons appear displaced by v_0, v_1, and $v_0 + v_1$. Along with $(0, 0)$, these three displacements form the vertices of the *fundamental parallelogram.*

Note: there are an infinite number of pairs of vectors which generate the same lattice; for example, v_0 and $v_0 + v_1$ also generate the lattice.

The *density* of a lattice packing is the fraction of the area covered by polygons. Copies of the fundamental parallelogram tile the plane. For every pair of integers i_0, i_1, there is one copy of the fundamental parallelogram and one copy of each polygon $P_1, P_2, P_3, \ldots, P_k$. Therefore the density is the total area of the k polygons divided by the area $|v_0 \times v_1|$ of the fundamental parallelogram. For each particular lattice packing problem, the polygons and their areas are fixed, and therefore a *densest* translational lattice packing is one which minimizes the area $|v_0 \times v_1|$ of the fundamental parallelogram. This paper gives an algorithm and implementation for *densest translational polygon lattice packing.*

Note: this algorithm finds a *global* minimum.

The lattice packing algorithm uses the "CG-to-MP" strategy devised by the author for packing and optimization problems: use algorithms of computational geometry to reduce the problem to mathematical programming. It is similar to the author's previous algorithms for minimum enclosure; however, the lattice packing problem presents new difficulties which need to be solved. Also the lattice packing algorithm introduces superior techniques to solve some previous problems.

The first major new difficulty is the absence of a container: as the figure illustrates, the fundamental parallelogram does not contain the four polygons. Without the boundary of a container to "bump up against" the lattice packing algorithm cannot use geometric restriction—a powerful technique developed

for the minimum enclosure problem. It is possible to use LP (linear programming) restriction, but the technique has to be modified.

The second major difficulty is the non-linearity and non-convexity of the objective. The objective to be minimized is the area $|v_0 \times v_1|$ of the fundamental parallelogram of the lattice. For strip packing, the objective is x, the length of the rectangle, which is linear. Even for more general minimum enclosure in an axis-parallel rectangle, the objective is xy, which has a convex boundary $xy = c$. In contrast, the set $v_0 \times v_1 = c$ has a saddle point at every point. [2]

New techniques and algorithms are developed to deal with these major difficulties. The most important development is an approximation to the lattice area. This approximate area is the "trick" that makes the lattice packing algorithm work. In addition, a new algorithm is given for selecting maximal convex subsets of polygonal regions as a part of evaluation and compaction. The new algorithm also introduces a new strategy for subdivision (selecting cutting planes). These new techniques can also be applied to improve the previous algorithms for containment and minimum enclosure.

1.2. Related work

Very little work has been done on lattice packing of polygons, and the only previous results we are aware of apply only to the problem of lattice packing or double-lattice packing a single *convex* polygon [7,13]. We are aware of only some preliminary work on heuristics for lattice packing of multiple non-convex polygons. However, it should be stated that most of the heuristic and meta-heuristic methods for packing are very general in nature and should apply to the lattice packing problem too. There are a number of surveys of packing/nesting heuristics [1,4–6,14,15].

The most closely related algorithms are for containment and minimum enclosure by the author. For multi-polygon *translational* layout, we have had considerable theoretical and practical success using a combination of computational geometry and mathematical programming (CG-to-MP) [2,11,12]. In practice, these algorithms can solve translational containment for up to ten polygons and minimum enclosure for at least five. We have also proved a number of theoretical running time bounds including $O((m^2+mn)^{2k}\log n)$ for placing k non-convex m-gons into a non-convex n-gon and $O(m^{4k-4}\log m)$ for placing them into a minimum area, fixed orientation rectangle [10].

1.3. Outline

Section 2 describes how to transform the densest lattice packing problem first to a *target area problem* and then to an *angle range problem.* It gives a high level description of the algorithm for the angle range problem. Section 3 gives the details of the angle range algorithm and proves its correctness. Section 4 discusses its implementation and experimental results. Overall the program is about 6500 lines of C++, 3100 of which are new and specific to the lattice packing algorithm. All operations are carried out in floating point arithmetic.

[2] Since all vectors are planar, we define $u \times v = u_x v_y - u_y v_x$, the scalar z-component of the three-dimensional cross product. The x-component and y-component are zero.

Table 1
Finding a densest lattice packing using the solution to the target area problem

set A equal to known feasible area
solve target area problem for target area A
repeat
set $A = (1+\varepsilon)^{-1}\lvert v_0 \times v_1\rvert$
solve target area problem for target area A
until problem is infeasible

2. Overview of the lattice packing algorithm

The densest translational polygon lattice packing problem takes as input k planar polygonal regions $P_1, P_2, P_3, \ldots, P_k$. The outputs are k translation vectors $t_1, t_2, t_3, \ldots, t_k$ for the polygons and generating vectors v_0 and v_1 for a lattice. The polygons must not overlap: for $1 \leqslant i, j \leqslant k$ and for any integers i_0, i_1, $P_i + t_i$ does not overlap $P_j + t_j + i_0 v_0 + i_1 v_1$.[3] The area $|v_0 \times v_1|$ of the fundamental polygon is minimal.

This section gives an overview of how we solve the densest lattice packing problem. We first reduce it to the *target area problem* which we in turn reduce to the *angle range problem.* The algorithm for solving the angle range problem is based on four operations: *restriction, evaluation, compaction,* and *subdivision.* These operations use iterated linear programming or iterated quadratic programming. We solve the linear programs and quadratic programs using the CPLEX library.

2.1. Target area problem

Densest lattice packing has a related *feasibility question* which we call the *target area problem:* given $P_1, P_2, P_3, \ldots, P_k$ and a target area A, determine if there exists a lattice packing with lattice area $|v_0 \times v_1| \leqslant A$ and, if so, find one whose area is at a *local minimum.* Table 1 shows how we use a solution to the target area problem to find a solution to the densest lattice packing problem which is at most $1+\varepsilon$ times optimal. For example, if $\varepsilon = 0.0001$, it is within 0.01% of optimal.

Note: a local minimum this close to the global minimum almost certainly is the global minimum.

Because $1+\varepsilon$ is nearly equal to 1, it might seem that this algorithm will very slowly converge to the optimum. However, each new target area is less than the area of the smallest local minimum seen so far, and therefore each iteration eliminates all local minima which have greater area than the best seen so far. It is reasonable to assume that the solution to the target area problem is a *randomly selected* local minimum which achieves the target. If this were so, then each iteration would eliminate half of the remaining local minima on average, and the number of iterations would be logarithmic in the number of local minima. We have no proof that our algorithm for the target area problem selects a random local minimum, but it does converge quickly in practice.

[3] For polygonal region P and translation vector t, $P + t = \{p + t \mid p \in P\}$ denotes region P translated by t.

2.2. Angle range problem

To solve the target area problem, we reduce it to multiple *angle range* problems. An *angle range problem* is a target area problem plus a pair of intervals $[\theta_{00}, \theta_{01}]$ and $[\theta_{10}, \theta_{11}]$ which constrain the angles of the lattice generators: the generator v_0 must have an angle (with respect to the x-axis) in the range $[\theta_{00}, \theta_{01}]$, and the generator v_1 must have an angle in the range $[\theta_{10}, \theta_{11}]$. The two intervals must not overlap, and they must be shorter than 180 degrees.

It is well known that one can always choose generating vectors v_0 and v_1 for a lattice such that the angle from v_0 to v_1 is between 60 and 120 degrees. A modest but very tedious generalization of this result shows that v_0 and v_1 can always be chosen to lie either in the angle range pair $[-75, -15]$, $[15, 75]$ or the range pair $[-30, 30]$, $[60, 120]$. Therefore, a target area problem can be reduced to two angle range problems.

Note: given these choices of angle ranges, we can assume from now on that $v_0 \times v_1 \geqslant 0$ and hence $|v_0 \times v_1| = v_0 \times v_1$.

2.3. Solving the angle range problem

To solve the angle range problem, we consider the unknown polygon translations $t_1, t_2, t_3, \ldots, t_k$ and the unknown lattice generators v_0 and v_1 to be point-valued ((x, y)-valued) *variables.* To these, we add the following:

- auxiliary point-valued variables w and u;
- linear constraints on the variables;
- a set of *relevant pairs* of polygons;
- (possibly non-convex and non-polygonal) two-dimensional regions U;
- point-inclusion constraints $u \in U$ for each u and corresponding U.

The variables, linear constraints, regions, and point-inclusion constraints are chosen so that, in a solution to the angle range problem, each w equals its corresponding u.

Relevant pairs of polygons. There are an infinite number of polygons in a lattice packing and therefore an infinite number of pairs of polygons. We only assert a non-overlap constraint for a finite set of *relevant pairs* of polygons. By the symmetry of the lattice, it suffices to consider pairs of the form $P_i + t_i$, $P_j + t_j + i_0 v_0 + i_1 v_1$. If some pair of polygons not in this set end up overlapping, then at least one such pair has to be added to the set. Fortunately, this addition only has to happen a finite number of times before all polygons are non-overlapping. This is true because i and j have only k possible values and for $|i_0|$ and $|i_1|$ sufficiently large, $P_j + t_j + i_0 v_0 + i_1 v_1$ is too far away from $P_i + t_i$ to overlap it.

Restriction. The point-inclusion constraints make the problem non-convex and even non-linear. However, we can *relax* each point-inclusion constraint as $u \in \mathrm{CH}(U)$, where $\mathrm{CH}(U)$ is the (convex and polygonal) convex hull of U. We add constraints setting the w's equal to the u's, and the result is a *linear program* (LP). If this LP is infeasible, then the relaxed problem has no solution, and therefore the original problem has no solution. In this case, we say the problem has *restricted to null.* Even if the problem does not restrict to null, we can calculate the range $R(u)$ of a variable u in the relaxed problem. This range must be a superset of the range of u in the unrelaxed problem. Therefore, we can replace U by $U \cap R(u)$ without throwing away any solutions to the original angle range problem. We call this the

restriction of U. To *restrict* a problem, we restrict each U in turn. Calculating the range of a variable u requires iterated linear programming.

Slack objective. The *slack objective* is a (positive-definite quadratic) sum of all terms of the form $(w-u)^2$. For some of these terms, $|w-u|$ corresponds to the *intersection depth* [3] of a relevant pair of polygons: the length of the minimum displacement necessary to un-overlap the two polygons. For other of these terms, $(w-u)^2$ corresponds to an approximate measure of *excess area*: if $v_0 \times v_1 \leqslant A$, then these terms are zero. Hence the slack objective includes a measure of overlap of relevant pairs of polygons plus a measure of *estimated excess area* (EEA).

Evaluation. Unlike restriction, which uses a convex *superset* of each U (its convex hull), *evaluation* calculates a convex polygonal *subset* $I(w,U) \subseteq U$: a maximal convex subset which contains the point of U nearest to w. Evaluation *constricts* (as opposed to *relaxes*) the problem by temporarily replacing the constraint $u \in U$ with $u \in I(w,U)$. The *constricted* problem, along with the slack objective, is a *quadratic program* (QP). Solving this QP gives (possibly) new values for the w's and hence new regions $I(w,U)$ (since the nearest point to w in U may change). Iterating this step yields a local minimum of the slack objective. To *evaluate* a problem, we calculate this local minimum.

Compaction. If the slack objective is zero, the w's equal the corresponding u's. This means that no pair of relevant polygons are overlapping. The EEA is also zero. Unfortunately, the EEA is only an estimate. It is still possible for the true lattice area $v_0 \times v_1$ to be greater than the target A. *Compaction* moves the packing to a local minimum of the lattice area while keeping the w's equal to the u's. Since the cross product $v_0 \times v_1$ is neither linear nor positive-definite, we temporarily replace it by a linearization and use linear programming to solve for the minimum. Since we are using a linear approximation to the objective, we have to use interpolation to find a step that minimizes the true lattice area. To *compact* a problem, we iterate linear programming and interpolation on constricted problems until we reach a local minimum of the lattice area.

Subdivision. There are two kinds of subdivision: *linear* and *angular.* If evaluation results in a non-zero slack objective, then we choose a pair of variables u, w corresponding to a largest term $(w-u)^2$ in the objective. Let L be the perpendicular bisector of the line segment uw. Since $u = w$ in a solution, we can create two sub-problems: one in which u and w are constrained to lie to the left of L and another in which u and w are constrained to lie to the right. This is *linear subdivision.* If evaluation results in a zero slack objective but compaction results in a lattice area $v_0 \times v_1 > A$, then we apply *angular subdivision* to one of the angle ranges $[\theta_{00}, \theta_{01}]$ or $[\theta_{10}, \theta_{11}]$. If we choose to divide the first range, let θ_0 be the angle of v_0. We create two sub-problems: one in which $[\theta_{00}, \theta_{01}]$ is replaced by $[\theta_{00}, \theta_0]$ and another in which it is replaced by $[\theta_0, \theta_{01}]$. This is *angular subdivision.*

2.4. Angle range algorithm

Table 2 gives the algorithm that solves the angle range problem. Keep in mind that restriction requires doubly iterated linear programming; evaluation requires iterated quadratic programming; and compaction requires iterated linear programming and interpolation.

Table 2
Algorithm to solve the angle range problem

```
repeat
  repeat
    restrict the problem
    if restricted to null
      return "infeasible"
  while the area of some U diminishes by at least a fixed fraction
  evaluate the problem
  if the slack objective is non-zero
    subdivide using linear subdivision
    solve (recursively) the two sub-problems
    return the solution if either is feasible else return "infeasible"
  compact the layout
  if v0 × v1 > A
    subdivide using angular subdivision
    solve (recursively) the two sub-problems
    return the solution if either is feasible else return "infeasible"
  if any pair of polygons is overlapping
    add one such pair to the set of relevant pairs
until the set of relevant pairs is unchanged
return the solution
```

3. Details of the angle range algorithm

This section gives the details of the algorithm for solving the angle range problem. First it gives a representation for the set of relevant pairs and defines the corresponding w's, u's, U's and constraints. These act to constrain the polygon overlaps. Additional w's, u's, U's, and constraints are introduced to constrain the lattice area. The linear constraints on the variables are described. Details are given on restriction, constriction (constructing the inner convex approximations $I(w, U) \subseteq U$), and the convergence of evaluation and compaction. Finally, this section provides the missing details of angular subdivision and proves that the algorithm of Section 2.4 correctly converges to a solution to the angle range problem or determines that none exists.

3.1. Relevant pairs

A *lattice pair index* $\langle i, j, i_0, i_1\rangle$, where $1 \leqslant i, j \leqslant k$, signifies a choice of two polygons P_i and P_j and a lattice point $i_0 v_0 + i_1 v_1$. The set of *relevant pairs* of polygons is represented as a set of lattice pair indices.

Each lattice pair index $\langle i, j, i_0, i_1\rangle$ corresponds to

two point-valued variables	$w_{\langle i,j,i_0,i_1\rangle}$ and $u_{\langle i,j,i_0,i_1\rangle}$,
a linear equality constraint	$w_{\langle i,j,i_0,i_1\rangle} = -t_i + t_j + i_0 v_0 + i_1 v_1$,
a two-dimensional region	$U_{\langle i,j,i_0,i_1\rangle} = \overline{P_i \oplus -P_j}$,
and a point-inclusion constraint	$u_{\langle i,j,i_0,i_1\rangle} \in U_{\langle i,j,i_0,i_1\rangle}$.

The symbol $\oplus$ represents the *Minkowski sum*, which we use extensively in our work on layout algorithms [11]. The Minkowski sum reduces polygon overlap to point-inclusion. Specifically, for regions P and Q and translations t_p and t_q, $P + t_p$ and $Q + t_q$ overlap if and only if $t_q - t_p$ lies in the interior of the region,

$$P \oplus -Q = \{p - q \mid p \in P \text{ and } g \in Q\}.$$

The variable $u_{\langle i,j,i_0,i_1\rangle}$ is always set to a point of $U_{\langle i,j,i_0,i_1\rangle}$ which minimizes $|w_{\langle i,j,i_0,i_1\rangle} - u_{\langle i,j,i_0,i_1\rangle}|$ with respect to the current value of the variable $w_{\langle i,j,i_0,i_1\rangle}$.

Lemma 1. *The distance* $|w_{\langle i,j,i_0,i_1\rangle} - u_{\langle i,j,i_0,i_1\rangle}|$ *is the intersection depth of polygons* $P_i + t_i$ *and* $P_j + t_j + i_0 v_0 + i_1 v_1$.

Proof. The variable $w_{\langle i,j,i_0,i_1\rangle} = (t_j + i_0 v_0 + i_1 v_1) - t_i$ is the current displacement between the two polygons. In order for the polygons not to overlap, their displacement must lie in $U_{\langle i,j,i_0,i_1\rangle} = \overline{P_i \oplus -P_j}$, the complement of the Minkowski sum. The distance $|w_{\langle i,j,i_0,i_1\rangle} - u_{\langle i,j,i_0,i_1\rangle}|$ is the minimum additional displacement required to accomplish this. □

3.2. Constraints on area

Recall that the angles of the lattice generators v_0 and v_1 are constrained to lie in the ranges $[\theta_{00}, \theta_{01}]$ and $[\theta_{10}, \theta_{11}]$, respectively. Define $b_{00} = (\cos\theta_{00}, \sin\theta_{00})$ to be the unit vector with angle θ_{00}, and define b_{01}, b_{10}, and b_{11} similarly.

We introduce four additional point-valued "w" variables w_0, w_0', w_1, and w_1'. These are related to v_0 and v_1 by the following linear constraints:

$$v_0 = w_{0x} b_{00} + w'_{0x} b_{10}, \qquad v_1 = w_{0y} b_{10} - w'_{0y} b_{00}, \tag{1}$$

$$v_0 = w_{1x} b_{01} - w'_{1x} b_{11}, \qquad v_1 = w_{1y} b_{11} + w'_{1y} b_{01}. \tag{2}$$

In other words, v_0 and v_1 are expressed as a linear combination of b_{00} and b_{10} and also as a linear combination of b_{01} and b_{11}, and the components of these w's are the coefficients of these linear combinations. The signs are chosen to keep the components of the w's positive.

We introduce two additional "u" variables u_0 and u_1 and corresponding planar regions,

$$U_0 = \{(x, y) \mid x, y \geqslant 0 \text{ and } xy \leqslant (b_{00} \times b_{10})^{-1} A\}, \tag{3}$$

$$U_1 = \{(x, y) \mid x, y \geqslant 0 \text{ and } xy \leqslant (b_{01} \times b_{11})^{-1} A\}. \tag{4}$$

The new u's satisfy the point-inclusion constraints $u_0 \in U_0$ and $u_1 \in U_1$ and they are always set to points which minimize $|w_0 - u_0|$ and $|w_1 - u_1|$, respectively.

Note: this might require finding the nearest point to w on a hyperbolic arc $xy = a$. To do this in closed form requires solving a fourth degree equation. However, it suffices to find the point numerically using Newton's method.

Lemma 2. *If $v_0 \times v_1 \leqslant A$, then $w_0 \in U_0$ and $w_1 \in U_1$. If $w_0 \in U_0$ and $w_1 \in U_1$ and either v_0 is collinear with b_{00} or b_{01} or v_1 is collinear with b_{10} or b_{11}, then $v_0 \times v_1 \leqslant A$.*

Proof. Given Eqs. (1) and (2) and the fact that v_0 and v_1 lie in the angle ranges, it follows that the components of w_0, w_0', w_1, and w_1' are all non-negative. Also,

$$v_0 \times v_1 = (w_{0x} b_{00} + w'_{0x} b_{10}) \times (w_{0y} b_{10} - w'_{0y} b_{00}), \tag{5}$$

$$= (w_{0x} w_{0y} + w'_{0x} w'_{0y})(b_{00} \times b_{10}). \tag{6}$$

If the angle ranges are small, then $|w_0'|$ will be small, and $(w_{0x} w_{0y})(b_{00} \times b_{10})$ will be a good approximation to the true area. (Similarly for $(w_{1x} w_{1y})(b_{01} \times b_{11})$.) Furthermore, $w'_{0x} w'_{0y}$ is non-negative, and so the approximation is always an under-estimate. Therefore,

$$(w_{0x} w_{0y})(b_{00} \times b_{10}) \leqslant v_0 \times v_1 \leqslant A,$$

which implies that $w_0 \in U_0$. Similarly for w_1.

If $w_0 \in U_0$, then $(w_{0x} w_{0y})(b_{00} \times b_{10}) \leqslant A$. If v_0 is collinear with b_{00}, then Eq. (1) implies that w'_{0x} must be zero. By Eq. (6),

$$v_0 \times v_1 = (w_{0x} w_{0y})(b_{00} \times b_{10}) \leqslant A.$$

The other cases are analogous. □

Lemma 1 implies that $P_i + t_i$ and $P_j + t_j + i_0 v_0 + i_1 v_1$ do not overlap if and only if there exists values for variables $w_{\langle i,j,i_0,i_1\rangle}$ and $u_{\langle i,j,i_0,i_1\rangle}$ such that $w_{\langle i,j,i_0,i_1\rangle} = u_{\langle i,j,i_0,i_1\rangle}$ and $u_{\langle i,j,i_0,i_1\rangle} \in U_{\langle i,j,i_0,i_1\rangle}$. In other words, setting the w's equal to the u's is equivalent to eliminating overlap.

Lemma 2 is not quite as strong with respect to the target area. Setting $w_0 = u_0 \in U_0$ and $w_1 = u_1 \in U_1$ only implies that the target area is met if (in addition) v_0 or v_1 is at an extreme of its angle range.

3.3. Additional linear constraints for all LPs and QPs

Constraining v_0 to the appropriate angle range is equivalent to the two linear constraints $b_{00} \times v_0 \geqslant 0$ and $b_{01} \times v_0 \leqslant 0$. Similarly for v_1.

Bounds are put on the lengths of v's in the direction of the b's: $v_0 \cdot b_{00} \leqslant D$, $v_0 \cdot b_{01} \leqslant D$, $v_1 \cdot b_{10} \leqslant D$, and $v_1 \cdot b_{11} \leqslant D$. The "diameter" D is chosen sufficiently large to ensure that no solutions are missed.

Without loss of generality, we can set $t_1 = (0, 0)$. Since we can arbitrarily add or subtract v_0 and v_1 from the other translations $t_2, t_3, \ldots, t_k$, we would like to constrain them to lie in the parallelogram with vertices $(0, 0)$, v_0, $v_0 + v_1$, and v_1: the primary fundamental parallelogram. However, this would require nonlinear constraints. Instead, we construct the smallest octagon which contains this parallelogram and which has sides parallel to b_{00}, b_{01}, b_{10}, or b_{11}. Due to the angle range constraints, the combinatorial structure of this octagon is fixed, and the constraint that each t_i lies in this octagon is convex and linear.

3.4. Restriction

Restriction is as described in Section 2.3. First, set the w's equal to the u's and replace each point-inclusion constraint $u \in U$ by a relaxed constraint $u \in \mathrm{CH}(U)$. Assuming each convex hull is a polygon, the result is a linear program. We have shown in previous work [11] how to trace out the range $R(u)$ of a point-valued variable u in a linear program.

Initially, the convex hull of U_0 (and U_1) is the first quadrant $\{(x, y)|x, y \geqslant 0\}$. Restriction intersects U_0 with a convex polygonal region $R(u_0)$. It is simplest to always compute this intersection with the *original* U_0 (Eq. (3)). If $R(u_0)$ has n edges, $U_0 \cap R(u_0)$ can have up to n connected components, each with a single concave hyperbolic edge. Using the quadratic formula, it is simple to determine if each edge of $R(u_0)$ has zero, one, or two intersections with the boundary of U_0. Some care has to be taken when carrying out this construction in floating point. Obviously, if the computed values of $p_x p_y$ and $q_x q_y$ indicate that vertices p and q of $R(U_0)$ both lie inside U_0, then the segment pq must cross the boundary of U_0 an *even* (zero or two) number of times, no matter what the numerics say! The same method constructs U_1.

3.5. Constructing inner convex approximations

Evaluation and compaction both depend on *constricting* the problem by constructing a convex subset $I(w, U) \subseteq U$ which is near to w. This section gives the algorithm for constructing $I(w, U)$ and proves that it has a visibility property essential for the correct convergence of both evaluation and compaction.

Algorithm. Given a point w, a region U, and a point $u \in U$ which minimizes $|w - u|$, the algorithm sets I equal to the component of U which contains u. The algorithm repeatedly clips I by the concave element of I which lies nearest to u. When no concave elements remain, the result is $I(w, U)$. Fig. 4 shows two examples. The next two paragraphs define *clipping* and *concave elements.*

Clipping. Clipping is defined as follows. To clip I by a line L, replace I with $I \cap H$, where H is the half-plane bounded by L and containing u. If $u \in L$, H is chosen so that the inward pointing normal vector at or near u points into the interior of I. To clip I by a straight edge e, clip I by the line L which contains e. To clip I by a concave curved edge e, calculate the point p on e closest to u and clip I by the line L tangent to e at p. To clip I by a (concave) vertex $v \neq u$, clip I by the line L which contains v and

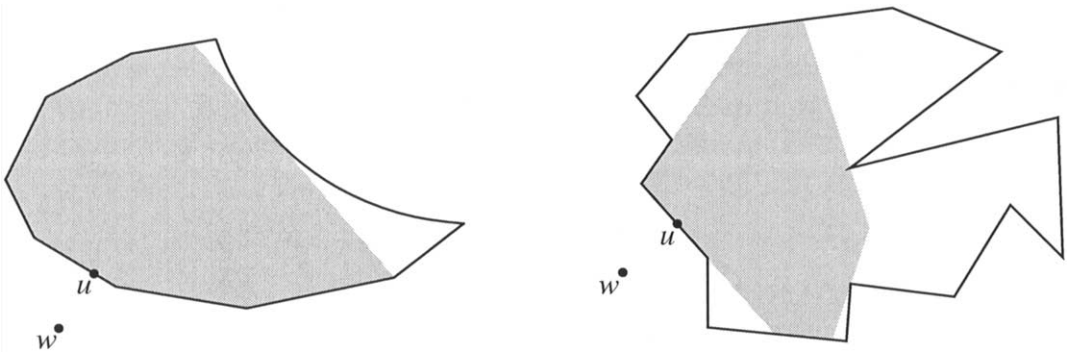

Fig. 4. Selecting convex subset near to w.

is perpendicular to the line segment uv. To clip I by a (concave) vertex $v = u$, clip I by the line L which contains v and is perpendicular to the angle bisector of the two edges coming into vertex v.

Concave elements. Concave elements are defined as follows. A vertex is concave according to the standard definition. A straight edge is concave if at least one of its endpoints is a concave vertex. A curved concave edge is always concave.

Lemma 3. *If w does not lie at a concave vertex of U, then $I(w, U)$ contains a point $u \in U$ nearest to w, and for some open set O containing u, $I(w, U)$ contains all points of $O \cap U$ which are visible to u.*

Proof. This lemma uses the standard definition that $a, b \in U$ are *visible* to each other if the line segment $ab \subset U$. The algorithm clips away portions of U that lie farther and farther away from u. We only need to check that the visibility condition is satisfied when the algorithms clips I by elements which contain u.

If u lies at a convex vertex, the algorithm might clip I by one or both of the two edges of that vertex, but that does not change the neighborhood. If u lies on a straight edge, the algorithm might clip I by that edge, again not changing the neighborhood. If u lies in the interior, then no elements touch it. Finally, if u lies on the concave hyperbolic edge e (of U_0 or U_1), then in a vicinity of u, only points in the half-plane bounded by the tangent line to the curve at u are visible to u. Again clipping I by e does not remove any points visible to u in a neighborhood about u. □

3.6. Evaluation and compaction

Both evaluation and compaction *constrict* the problem by replacing each point-inclusion condition $u \in U$ by $u \in I(w, u)$. Evaluation sets each u to a point of U that minimizes $|w - u|$, constructs the $I(w, U)$ regions, and minimizes the slack objective using quadratic programming. It repeats these steps until the slack objective converges to a minimum (numerically).

In each of its minimization steps, compaction solves an LP instead of a QP and performs an interpolation. Specifically, it constricts the problem, sets the w's equal the u's, and minimizes a linearization of the lattice area $v_0 \times v_1$ about its current value:

$$(v_0 + \Delta v_0) \times (v_1 + \Delta v_1) \approx v_0 \times v_1 - v_1 \times \Delta v_0 + v_0 \times \Delta v_1. \tag{7}$$

The variables of this LP are perturbations (deltas) of the current values of the variables of the lattice area problem. This LP has a non-trivial solution if and only if there is some perturbation which diminishes the lattice area. For some s, $0 < s \leqslant 1$, the compaction algorithm adds s times the calculated deltas to the current values of the variables. The interpolation of the area $(v_0 + s\Delta v_0) \times (v_1 + s\Delta v_1)$ equals

$$v_0 \times v_1 - s(v_1 \times \Delta v_0 - v_0 \times \Delta v_1) + s^2 \Delta v_0 \times \Delta v_1.$$

Its value is minimized when the derivative with respect to s is zero,

$$s_{\min} = (v_1 \times \Delta v_0 - v_0 \times \Delta v_1)(2\Delta v_0 \times \Delta v_1)^{-1}.$$

Since for sufficiently small s, the area must be smaller than the current area, $s_{\min} > 0$. If $s_{\min} < 1$, compaction uses $s = s_{\min}$; otherwise, it simply adds the full deltas to the variables. This update step is iterated until the lattice area numerically converges to a minimum.

The following theorem implies that both evaluation and compaction converge to a true local minimum of the original point-inclusion constraints.

Theorem 4. *Except for cases in which some u is equal to a concave vertex of the corresponding U, any minimization problem with respect to the point-inclusion constraints $u \in U$ has the same equilibria when these constraints are constricted to $u \in I(w, U)$.*

Proof. Since $I(w, U) \subseteq U$, the constricted problem is more constrained than the original problem, and therefore any equilibrium of the original is an equilibrium of the constricted problem.

Suppose the minimization with respect to the original problem is not at an equilibrium. That means there is some path from the current values of the variables, parameterized by s, such that the derivative of the objective with respect to s at $s = 0$ is negative.

Let $u(s) \subset U$ be the curve traced out by variable u in the objective-diminishing path. Let u' be the derivative of this curve at $s = 0$. The linear path $u + su'$ has the same derivative. Since U has no convex curved boundary edges, for sufficiently small s, $u + su' \in U$. Since this is also true for smaller s, $u + su'$ is visible to u in U. By Lemma 3, for sufficiently small s, $u + su' \in I(w, U)$. Therefore, for sufficiently small s, the interpolant for s is a objective-diminishing solution to the constricted problem. Hence, that problem is not at an equilibrium either. □

Evaluation or compaction should only extremely rarely if at all converge to a state in which some u is at a concave vertex of the corresponding U. First, if $w \notin U$, then a point $u \in U$ which minimized $|w - u|$ *cannot* lie at a convex vertex. Even if $w \in U$, the concave vertices of U are not vertices of $I(w, U)$. The point w would have no "reason" to converge to this particular point. Even if evaluation or compaction do so converge, it does not ruin the overall correctness of the angle range algorithm. We could prove convergence of the algorithm even considering this possibility, but it makes the proof twice as long and tedious to read. To keep the proofs simple, we disregard this special case.

3.7. Subdivision

Section 2.3 gave a complete description of *linear subdivision.* Since the cutting line L is midway between a particular w and u, they lie on opposite sides of L. Therefore, neither sub-problem allows this particular minimum of the slack objective to occur again. This section gives the details of *angular subdivision,* and proves that it prevents this particular minimum of the lattice area from occurring again.

Angular subdivision is only necessary if the EEA is zero but the lattice area is greater than the target. Lemma 2 implies that neither v_0 nor v_1 can lie at an extreme of its angle range, and therefore they both lie in the interior of their angle ranges. The subdivision algorithm chooses the cut that minimizes the maximum angle range in the subproblems. Suppose the range $[\theta_{00}, \theta_{01}]$ of v_0 is cut at the current angle θ_0 of v_0, and therefore the current v_0 is at an extreme of the angle range in each sub-problem: $[\theta_{00}, \theta_0]$ or $[\theta_0, \theta_{01}]$. If v_0 is at an extreme, by Lemma 2, EEA equals zero implies target area is met. Therefore, the current local minimum cannot be the output of compaction in either sub-problem. Similarly if the range of v_1 is cut.

Theorem 5. *The algorithm in Table 2 for solving the angle range problem is correct and runs in finite time.*

Proof. Restriction eliminates no solutions. Lemma 3 implies that evaluation and compaction work correctly. By Lemma 1, zero slack objective implies no overlaps. Therefore, if the algorithm terminates, it terminates correctly.

The discussion of this section, based partly on Lemma 2, proves that the algorithm visits each local minimum of the slack objective no more than once and each local minimum of the lattice area no more than once. Therefore the algorithm terminates. □

4. Results and conclusions

Section 4.1 gives implementation details, particularly techniques used to speed up the algorithm in practice. Section 4.2 provides examples of lattice packings generated by the algorithm, and Section 4.3 examines the performance of the algorithm for infeasible target areas. Finally, Section 4.4 gives conclusions and directions for future research.

4.1. Implementation details

We implemented the algorithm in C++ on a 600 MHz Pentium II PC running RedHat Linux 5.2. We used CPLEX 6.5 to solve the linear and quadratic programs.

A general "rule of thumb" in hand layout is that it is good to pair polygons with their 180 degrees rotated twins. We constrained these pairs to be "nearly touching" as follows. Select a small, disk-like polygon B, and chose only those point which lie "on the rim" of the Minkowski sum as defined by this disk:

$$U_{\langle i,j,i_0,i_1\rangle} = (P_i \oplus -P_j \oplus B) \cap \overline{P_i \oplus -P_j}. \tag{8}$$

The selected polygon B is a square 1% of the diameter of the polygons.

We observe that it is not necessary to restrict a U if it is already convex. In fact, one can prove that this optimization does not affect the restriction of non-convex U.

For $U_{\langle i,j,i_0,i_1\rangle}$ not corresponding to the pair w, u just subdivided, we employed a faster but weaker form of restriction. Specifically, we only calculated the bounding, axis-parallel, rectangle of the range $R(u_{\langle i,j,i_0,i_1\rangle})$ instead of $R(u_{\langle i,j,i_0,i_1\rangle})$ itself. As a result of this modification, the algorithm visits more subproblems, but it requires much less time to restrict each subproblem. Our experiments showed that this tradeoff reduced the overall running time. Whether we use the actual range $R(u_{\langle i,j,i_0,i_1\rangle})$ or its bounding box, we repeat restriction until no U diminishes more than 50% in area.

Finally, to generate positive examples, we limited the depth of subdivision to a specific bound. This prevented the algorithm from spending too much time in a barren region of the search space. For negative examples, it is (unfortunately), necessary to search the entire space.

4.2. Positive examples

To illustrate the positive examples, we ran the program on four inputs, two with two polygons, two with four polygons: (1) two "free-hand" pentagons, (2) two bra parts, (3) the two parts of (1) plus 180 degrees rotated copies, and (4) the same with respect to (2). Figs. 5–7 illustrate examples (1)–(3). Figs. 2 and 3 illustrate example (4).

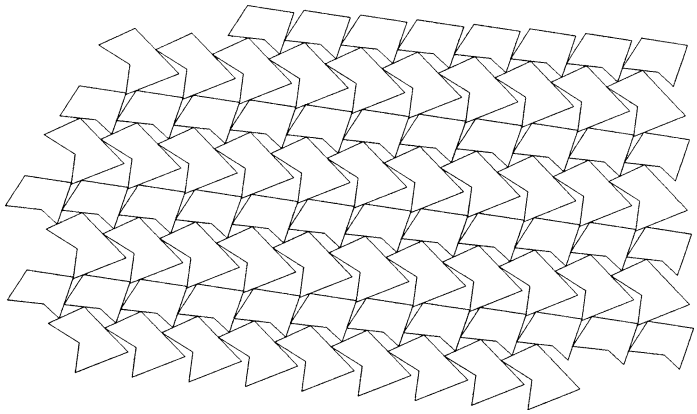

Fig. 5. Input (1): lattice packing of two pentagons.

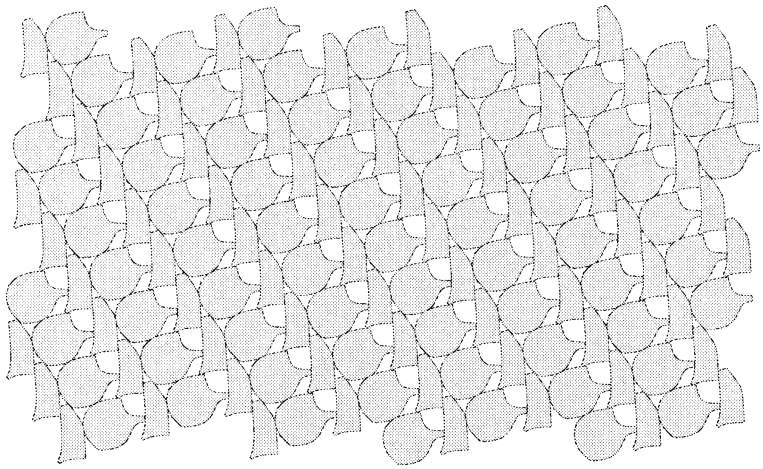

Fig. 6. Input (2): lattice packing of two bra parts.

Obviously, example (4) is the most difficult problem, the most time-consuming, and the one which is typical of the clothing industry. The cup polygons have 73 vertices, and the strap polygons have 51 vertices. We employed the following strategy to bound the optimal area. We first set the search depth to 3 and used a reasonable estimate for the lattice area. If we found a solution, we set the target area 1% smaller than the solution's lattice area. We repeated this step until we found no solution at depth 3. Then we tried diminishing the target area by 0.01%. We repeated this step until we found no solution. At this point we ran the same target with greater and greater depths. If a solution was found at a greater depth, we set the next target to the solution's area less 0.01%. We stopped when the running time become prohibitive.

Using this strategy, our first four target areas were 1200000, 1083314, 1072803, 1081202. The second generated a solution with area 1081310 and the third and fourth found no solution (above depth 3). This solution is depicted in Fig. 3. The total time was half an hour (1869 seconds). This solution has density

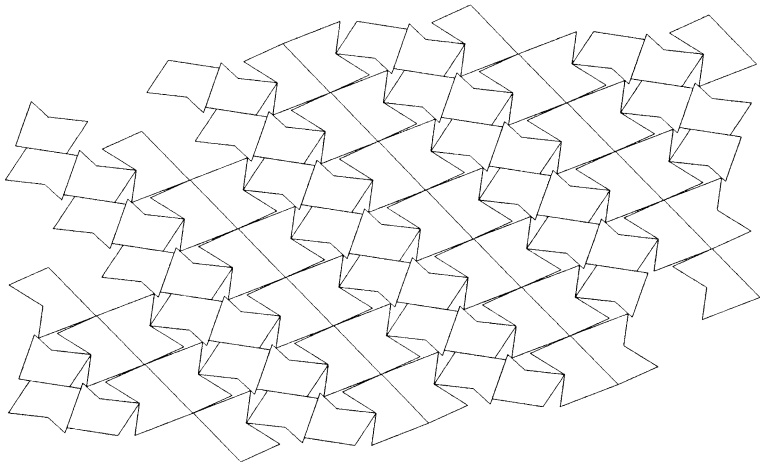

Fig. 7. Input (3): lattice packing of four pentagons.

90.2%, and when arbitrarily clipped to the rectangle in Fig. 1, it contains one more cup and 22 more straps than the human-generated layout.

We next started increasing the depth of search with the same target area 1081202. Searching to depths 4 and 5 found no solution and ran in 1131 second and 2038 seconds. Searching to depth 6 found a solution with area 1078220 and density 90.4% (in the plane). Fig. 2 depicts this packing clipped to a rectangle. We set the new target area to 1078112. Even a depth 8 search, which required just over 3 hours (11475 seconds), could find no better solution. A depth 10 search required just over 7 hours (25796 seconds). However, the depth bound cut off search on only one-quarter (584 out of 2376) of the subproblems, rather than one-half, which give us hope that a complete search would only take a few days.

4.3. Negative examples

As in many domains, it takes much more time to prove infeasibility than to generate a positive answer. We can be pretty certain that we have found the densest layout for example (4), but the running time for an arbitrary depth search would be prohibitive. Similarly, it required two hours just to verify (to arbitrary depth) that there is no layout twice as dense as the one in Fig. 7!

Fortunately, for those layouts for which verification is possible, it appears that we can obtain tight lower bounds on the optimal area. Table 3 shows the dependence of the running time on ε, where the target area is $(1-\varepsilon)A_{\text{optimal}}$, for example (2) (two bra parts). The experiments examine values of ε which are negative powers of 2 from 0.5 to 2^{-18}. As the table indicates, the running time reaches a plateau. Any addition running time required for greater accuracy is swamped by the overall running time for the verification.

Note: the conference version of this paper seemed to find the opposite result, but it turns out that the previous experiment did not examine enough ε's to find the "plateau".

Table 3
Experiments on negative examples $(1-\varepsilon)$ times the optimal area

$e = -\log_2 \varepsilon$	1	2	3	4	5	6	7	8	9
Seconds	11	86	146	231	302	368	423	415	473
$n =$ # subproblems	6	72	145	202	261	344	391	365	416
Seconds/subproblem	1.83	1.19	1.01	1.14	1.16	1.07	1.08	1.14	1.14
$e = -\log_2 \varepsilon$	10	11	12	13	14	15	16	17	18
Seconds	452	476	492	462	466	517	526	536	444
$n =$ # subproblems	386	429	435	397	400	472	474	495	389
Seconds/subproblem	1.17	1.11	1.13	1.16	1.17	1.10	1.11	1.08	1.14

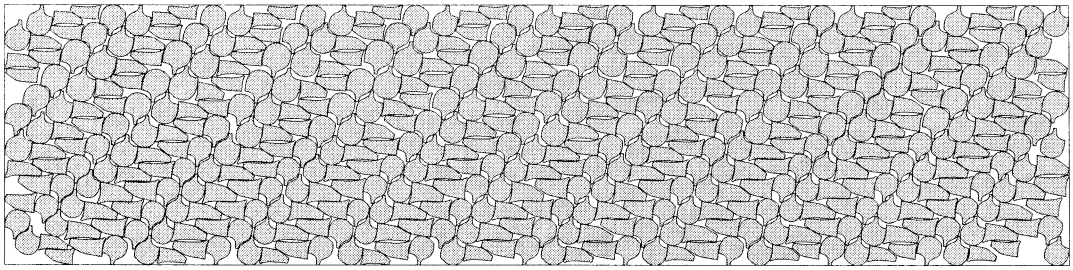

Fig. 8. Layout automatically generated from lattice packing.

4.4. Conclusions and future work

The lattice packing in Fig. 2 has density 90.4% (in the plane). By contrast, the human-generated layout in Fig. 1 covers only 80.0% of the rectangle. Obviously, we cannot hope to cover 90.4% of a rectangle, but it seems very likely that additional heuristics and algorithms will yield a much denser layout for the original strip packing problem than humans can generate, and the computer running time will be less than the human layout time.

The running times for positive examples were quite reasonable, especially if we are willing to accept a solution just short of optimal. The running times for verification of the densest lattice are not practical yet, but from an industrial point of view, this does not matter. With a limited depth search, the algorithm can quickly find layouts that are much denser than those used currently in practice. In the course of commercializing this algorithm, we expect to obtain more examples on which to run experiments and verify that the running times in example (4) are indeed typical. It is also likely that relatively minor modifications of the current algorithm will yield much faster running times for both positive and negative examples.

There are still additional steps to be taken to generate good industrial layouts from a lattice packing. The clipped lattice packing in Fig. 2 has excess parts. It also is a packing of only one *representative* cup and strap from the original layout in Fig. 1. We naively discarded excess parts and naively substituted the original parts, and then we applied overlap minimization and compaction [8,9]. The resulting layout (Fig. 8) matches the human layout in density, but it is a tight fit and cannot be compacted any smaller.

However, if we fully exploit the freedom we have selecting the appropriate subset of the lattice and in substituting the original shapes, it is likely we can beat the human density. This is a subject of future work. It is our belief that lattice packing will very shortly lead to the first almost-direct application of a global minimization algorithm to an industrially significant non-convex layout problem.

References

[1] K. Daniels, Containment algorithms for nonconvex polygons with applications to layout, Ph.D. Thesis, Harvard University, Cambridge, MA, 1995.
[2] K. Daniels, V.J. Milenkovic, Multiple translational containment, Part I: An approximate algorithm, Algorithmica 19 (1997) 148–182.
[3] D. Dobkin, J. Hershberger, D. Kirkpatrick, S. Suri, Computing the intersection-depth of polyhedra, Algorithmica 9 (1993) 518–533.
[4] K. Dowsland, W. Dowsland, Solution approaches to irregular nesting problems, European J. Oper. Res. 84 (3) (1995) 506–521.
[5] K.A. Dowsland, W.B. Dowsland, Packing problems, European J. Oper. Res. 56 (1992) 2–14.
[6] H. Dyckhoff, A typology of cutting and packing problems, European J. Oper. Res. 44 (1990) 145–159.
[7] G. Kuperberg, W. Kuperberg, Double-lattice packings of convex bodies in the plane, Discrete Comput. Geom. 5 (1990) 389–397.
[8] Z. Li, Compaction algorithms for nonconvex polygons and their applications, Ph.D. Thesis, Harvard University, Cambridge, MA, 1994.
[9] Z. Li, V. Milenkovic, Compaction and separation algorithms for nonconvex polygons and their applications, European J. Oper. Res. 84 (1995) 539–561.
[10] V. Milenkovic, Translational polygon containment and minimal enclosure using linear programming based restriction, in: Proc. 28th Annual ACM Symp. Theory Comput. (STOC-96), 1996, pp. 109–118.
[11] V. Milenkovic, K. Daniels, Translational polygon containment and minimal enclosure using mathematical programming, Internat. Trans. Oper. Res. 6 (1999) 525–554.
[12] V.J. Milenkovic, Multiple translational containment, Part II: Exact algorithms, Algorithmica 19 (1997) 183–218.
[13] D.M. Mount, The densest double-lattice packing of a convex polygon, Report 2584, Dept. Comput. Sci., Univ. Maryland, College Park, MD, 1991.
[14] P.E. Sweeney, E.R. Paternoster, Cutting and packing problems: A categorized, application-oriented research bibliography, J. Oper. Res. Soc. 43 (7) (1992) 691–706.
[15] P. Whelan, B. Batchelor, Automated packing systems: Review of industrial implementations, in: Proceedings of the SPIE—The International Society for Optical Engineering, Vol. 2064, SPIE, 1993, pp. 358–369.