Proceedings of the 16th Annual ACM Symposium on Computational Geometry, June 2000.

Submitted to *Computation Geometry: Theory and Applications*

Densest Translational Lattice Packing of Non-Convex Polygons

Victor J. Milenkovic*
University of Miami, Department of Computer Science
P.O. Box 249085, Coral Gables, FL 33124
vjm@cs.miami.edu

Abstract

A *translation lattice packing* of k polygons $P_1, P_2, P_3, \ldots, P_k$ is a (non-overlapping) packing of the k polygons which can be 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 translation lattice packing.* This algorithm has useful applications in industry, particularly clothing manufacture.

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, ..., 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 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, Figure 1 shows

*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.

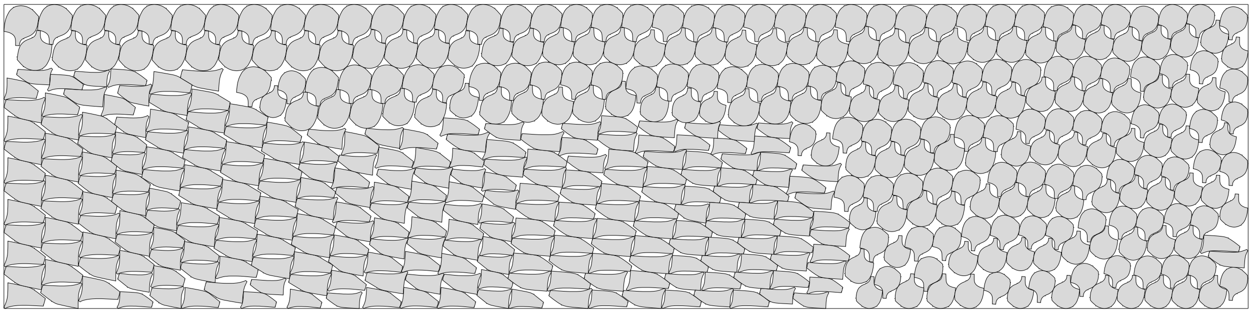

Figure 1: Human-generated layout of parts for 123 brassieres.

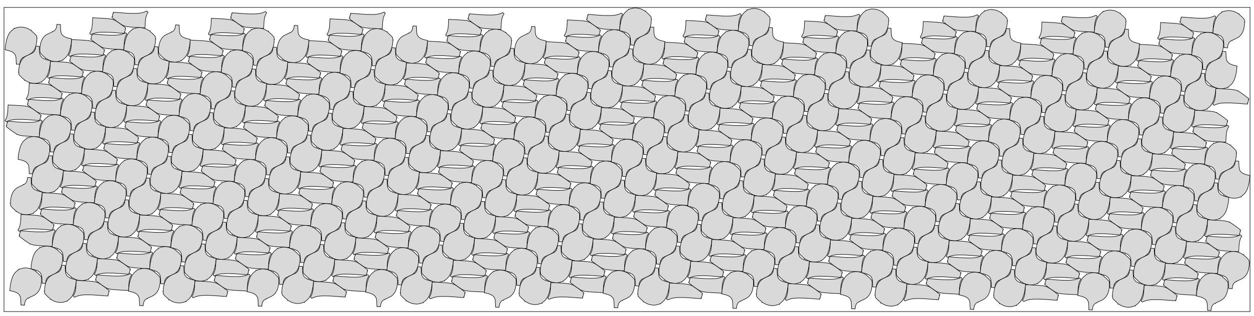

Figure 2: Densest lattice packing "clipped" to the same rectangle.

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.

Figure 2 depicts the densest periodic packing of four polygons: a cup, a side, 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 Figure 1. Clearly, after some fixing by hand and the application of *compaction* [7, 8], this layout could become a much denser solution than the human-generated layout. This example 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 Figure 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 paper. In the meantime, it is clear that an algorithm for densest lattice packing is a big step towards solving an important industrial problem.

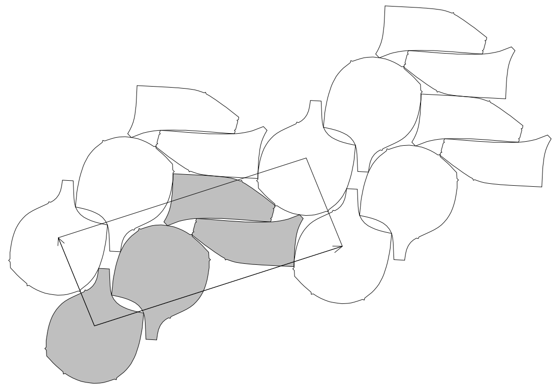

Figure 3: Four polygons (grey), lattice packing, fundamental parallelogram, and lattice generators (dark arrows).

1.1 New Results

A *translation lattice packing* of k polygons $P_1, P_2, P_3, \ldots, P_k$ is a (non-overlapping) packing of the k polygons which can be 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. Figure 3 is a closeup of a lattice packing (not the one in Figure 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 translation 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

$$|v_0 \times v_1| = |v_{0x} v_{1y} - v_{0y} v_{1x}|$$

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.[1]

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 overlap minimization. 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 [14, 6]. 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 [3, 1, 16, 4, 15, 5].

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, 12, 10]. These algorithms can practically 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 $\mathrm{O}((m^2 + mn)^{2k} \log n)$ for placing k non-convex m-gons into a non-convex n-gon and $\mathrm{O}(m^{4k-4} \log m)$ for placing them into a minimum area, fixed orientation rectangle [9] (this paper also surveys other results in multi-polygon translational layout). Of all this work, the one most closely related to the lattice packing algorithm is enclosure in a minimum area axis-parallel rectangle. The following section summarizes this rectangle packing algorithm and explains how the new lattice packing algorithm differs from it.

1.3 Comparison to Minimum Rectangle Enclosure

The previous rectangle packing algorithm works as follows. It is given k polygons $P_1, P_2, P_3, \ldots, P_k$ and a target area A. It generates vectors $t_0, t_1, t_2, \ldots, t_{k+1}$, where t_0 and t_{k+1} are diagonal corners of a rectangle with area $\leq A$, and t_i is the translation of P_i into this rectangle, $1 \leq i \leq k$. Or it reports that no solution exists. Since positions are relative, t_0 is always set to $(0, 0)$.

The algorithm introduces auxiliary polygonal regions,

$$P_0 = \{(x, y) \,|\, x \leq 0 \text{ or } y \leq 0\} \qquad \text{and} \qquad P_{k+1} = \{(x, y) \,|\, x \geq 0 \text{ or } y \geq 0\}.$$

For all i, j, $0 \leq i < j \leq k + 1$, except $i = 0, j = k + 1$, it defines regions $U_{ij} = \overline{P_i \oplus -P_j}$, where $-A = \{-a \,|\, a \in A\}$ and where $\oplus$ is the Minkowski sum, $A \oplus B = \{a + b \,|\, a \in A \text{ and } b \in B\}$. Let

[1] 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.

$P+t=\{(p+t)\mid p\in P\}$ denote P translated by vector t. It can be shown that P_i+t_i and P_j+t_j do not overlap if and only if $t_j-t_i\in\overline{P_i\oplus -P_j}$. The algorithm defines $U_{0,k+1}=\{(x,y)\mid xy\leq A\}$. Because positions are relative, it sets $t_0=(0,0)$. The output rectangle has area $t_{k+1x}t_{k+1y}\leq A$ if and only if $t_{k+1}-t_0\in U_{0,k+1}$. To make the computation possible, it approximates the hyperbola $xy=A$ by a polygonal curve. **Difference**: the new lattice packing algorithm must also use regions bounded by hyperbolas. However, it cannot approximate the hyperbolas by polygonal chains and has to deal with them directly.

To find $t_0=(0,0),t_1,t_2,t_3,\ldots,t_{k+1}$ satisfying, $t_j-t_i\in U_{ij}$, $0\leq i<j\leq k+1$, the rectangle packing algorithm applies three operations: *restriction*, *evaluation*, and *subdivision*. It restricts the U_{ij} regions and then uses evaluation to find a local overlap minimum. If the overlap is not zero, it subdivides one of the U_{ij} regions into two parts and then recurses on two subproblems in which all the other U_{ij} polygons stay the same. If it finds a solution, it applies *compaction* to improve it to a local minimum. **Difference**: for lattice packing, compaction becomes an integral part of evaluation.

Restriction: The rectangle packing algorithm uses two restriction algorithms: geometric restriction and linear programming restriction. Geometric restriction performs the following substitution, $U_{ij}\leftarrow(U_{ih}\oplus U_{hj})\cap U_{ij}$, for all i,j,h, over and over. (Here $U_{ih}=-U_{hi}$ if $i>h$.) Linear programming restriction "relaxes" each U_{ij} by replacing it by its convex hull. It computes the range R_{gh} of t_h-t_g for $t_0,t_1,t_2,\ldots,t_k$ satisfying $t_j-t_i\in\mathrm{HULL}(U_{ij})$, $0\leq i<j\leq k$, and then replaces U_{gh} by $R_{gh}\cap U_{gh}$, for all g,h, over and over. **Difference**: lattice packing cannot use geometric restriction and it must modify LP restriction to allow for regions bounded by hyperbolic arcs.

Evaluation: A common element of the new lattice packing algorithm and the older containment/enclosure algorithms is *local overlap minimization*: move an overlapping placement of polygons to a placement which is a local minimum of the total overlap. *Evaluation* iterates a minimization step to reach a local minimum. However, the older algorithms use a minimization step which is only guaranteed to "start well": the output is not guaranteed to diminish the overlap, but some interpolation between the start and finish has smaller overlap. Hence, the older evaluation has to use binary search to find the best interpolant. **Difference**: The new lattice packing algorithm introduces an algorithm for selecting convex subsets of non-convex polygons.[2] This algorithm leads to an overlap minimization step that is guaranteed to diminish the overlap without any interpolation. The new algorithm also uses a quadratic objective and quadratic programming for the first time.

Subdivision: The older subdivision algorithm selects the U_{ij} corresponding to the most overlapped pair of polygons. If U_{ij} has multiple components, the components are segregated into U'_{ij} and U''_{ij}. If U_{ij} is connected, it is cut by a line in a manner to maximize the convexity of the two halves. **Difference**: Lattice packing uses a new subdivision algorithm which, unlike the old algorithm, is guaranteed never to visit the same previous "mistakes".

Compaction: When compacting, the lattice packing algorithm can take advantage of the new algorithm for selecting convex subsets. Unlike previous compaction algorithms [7, 8], it does not need to cover the polygons with star-shaped subsets. **Additional difference**: It also has to deal with the fact that the objective $|v_0\times v_1|$ is not Lorentzian. For this reason, the compaction stage of lattice packing must use interpolation.

[2] In the final revision of a recent journal paper [11], we suggested this new approach as an alternative to our older method. However, the lattice packing algorithm represents the first implementation and test of this new approach.

1.4 Outline

Section 2 describes how to transform the *feasibility question* for lattice packing to two minimization problems: *overlap minimization* and *area minimization*. Section 3 gives an algorithm to solve the minimization problems. 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 Overlap Minimization and Area Minimization

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 $1 \leq i, j \leq k$ 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 of the fundamental polygon $|v_0 \times v_1|$ is minimal.

Densest lattice packing has a related *feasibility question*: for a given target area A, does there exist a lattice packing with area $|v_0 \times v_1| \leq A$? To solve lattice packing, we transform the feasibility question to two related minimization problems: *overlap minimization* and *area minimization*. If successful, area minimization also generates a *local* minimum of the lattice area. Hence, if we can successfully solve overlap and area minimization, then the result generally is a good candidate for a global minimum area (densest) lattice packing. We set the target area a small fraction (such as 1%) smaller than the best known local minimum and solve the feasibility question again. This process generally requires very few (four or five) iterations. A "no" answer to the feasibility question provides a lower bound on the global minimum area which we can make as close as desired to the upper bound.

Overlap minimization is somewhat a misnomer because it also includes a form of area minimization. The objective consists of a sum of a measure of the total overlap among the polygons plus an *underestimate* of a measure of how much $|v_0 \times v_1|$ exceeds the target area A. A solution with zero objective has no overlaps but may have area greater than A. However, all lattice packings with $|v_0 \times v_1| \leq A$ are zero-objective solutions to overlap minimization. *Area minimization* uses the same variables and constraints as overlap minimization plus the additional constraint that the overlap (plus estimated excess area) objective is zero. The objective of *area minimization* is to minimize $|v_0 \times v_1|$.

Section 2.1 describes the overlap minimization problem, and Section 2.2 describes the area minimization problem. Section 2.3 shows how to detect local minima of either problem. Relevant facts are proved which ensure the correctness of the algorithm for solving overlap and area minimization in Section 3.

2.1 Overlap Minimization

Section 2.1.1 defines the general overlap minimization problem. It consists of a collection of vector-valued variables and planar regions. Section 2.1.2 shows how to choose the regions such that overlap minimization has the properties described above.

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

2.1.1 Definition of Overlap Minimization

The translations $t_1, t_2, t_3, \ldots, t_k$ are the unknown translations of polygons $P_1, P_2, P_3, \ldots, P_k$. Since position is relative, t_1 can be set to $(0,0)$. Vectors v_0 and v_1 are the unknown lattice generators.

Each overlap minimization problem has a fixed pair of *angle ranges* $[\theta_{00}, \theta_{01}]$ and $[\theta_{10}, \theta_{11}]$ for the generators v_0 and v_1. The vectors $b_{00} = (\cos\theta_{00}, \sin\theta_{00})$, *etc.*, are unit vectors pointing along the angle range directions. Hence v_0 lies in the angle formed by b_{00} and b_{01}, and, similarly, v_1 lies between b_{10} and b_{11}. **Note**: these are *linear* constraints on v_0 and v_1.

Four vector-valued variables w_0, w_0', w_1, w_1' are "slaved" to the values of variables v_0 and v_1 in the following manner:

$$\begin{aligned} v_0 &= w_{0x} b_{00} + w'_{0x} b_{10}, & (1)\\ v_1 &= w_{0y} b_{10} - w'_{0y} b_{00}, & (2)\\ v_0 &= w_{1x} b_{01} - w'_{1x} b_{11}, & (3)\\ v_1 &= w_{1y} b_{11} + w'_{1y} b_{01}, & (4) \end{aligned}$$

In other words, v_0 and v_1 are expressed in terms of the vector basis b_{00} and b_{10} and also in terms of the vector basis b_{01} and b_{11}. The components of the w vectors are the coefficients of these expressions with particular signs chosen for technical reasons (to keep the components of the w's positive). Actually, these relations are inverted to express the w's in terms of the v's, but it is mathematically simpler to express v_0 and v_1 in terms of w_0 and w_0' (or w_1 and w_1').

Associated with w_0 and w_1 are regions U_0 and U_1 and two additional vector-valued variables $u_0 \in U_0$ and $u_1 \in U_1$. The set U_0 is H_0 intersected with a convex polygonal set, where

$$H_0 = \{(x,y) \,|\, x, y \geq 0 \text{ and } xy \leq \frac{A}{b_{00} \times b_{10}}\}, \tag{5}$$

and where A is the target area of the lattice feasibility question.[4] The set U_1 is defined similarly with respect to,

$$H_1 = \{(x,y) \,|\, x, y \geq 0 \text{ and } xy \leq \frac{A}{b_{01} \times b_{11}}\}. \tag{6}$$

A *lattice index* $\langle i, j, i_0, i_1 \rangle$, where $1 \leq i, j \leq k$, signifies a choice of two polygons P_i and P_j and a lattice point $i_0 v_0 + i_1 v_1$. L^∞ denotes the set of all lattice indices. For each lattice index $\langle i, j, i_0, i_1 \rangle$, there is a polygonal region $U_{\langle i,j,i_0,i_1 \rangle}$ and a vector-valued variable $u_{\langle i,j,i_0,i_1 \rangle} \in U_{\langle i,j,i_0,i_1 \rangle}$. Finally, there is another vector-valued "slave" variable

$$w_{\langle i,j,i_0,i_1 \rangle} = -t_i + t_j + i_0 v_0 + i_1 v_1. \tag{7}$$

The constraints of overlap minimization are the subset inclusions (u's inside U's) and "slave" relations as previously stated. It also has a set S of linear "cutting plane" constraints which are necessary to implement subdivision. The objective is to minimize the sum of the squares of the distances from the w's to the u's:

$$\sum_{i=0}^{1} (w_i - u_i)^2 + \sum_{\langle i,j,i_0,i_1 \rangle \in L^\infty} (w_{\langle i,j,i_0,i_1 \rangle} - u_{\langle i,j,i_0,i_1 \rangle})^2. \tag{8}$$

[4] Recall that, for purposes of this paper, $u \times v = u_x v_y - u_y v_x$, and so $b_{00} \times b_{10}$ is a scalar.

2.1.2 The Root Problems

The *root* overlap minimization problems are the ones which correspond to the lattice feasibility question for area A. The first root problem has the pair of angle ranges $[-75,-15],[15,75]$, and the second has range pair $[-30,30],[60,120]$.

For each root problem, $U_0 = H_0$ and $U_1 = H_1$, and for each lattice index $\langle i,j,i_0,i_1\rangle \in L^\infty$,

$$U_{\langle i,j,i_0,i_1\rangle} = \overline{P_i \oplus -P_j}, \tag{9}$$

where $A \oplus B = \{a + b \,|\, a \in A \text{ and } b \in B\}$ denotes the Minkowski sum. It is sometimes useful to specify that two polygons are touching. In that case we select a small, disk-like polygon B, and chose only those point which lie "on the rim" defined by this disk:

$$U_{\langle i,j,i_0,i_1\rangle} = (P_i \oplus -P_j \oplus B) \cap \overline{P_i \oplus -P_j}. \tag{10}$$

The set S is null.

Theorem 2.1 *A zero-objective solution to a root overlap minimization problem corresponds to a non-overlapping lattice packing of $P_1, P_2, P_3, \ldots, P_k$.*

Proof. The Minkowski sum has the property [11] that $P_i + t_i$ and $P_j + t_j$ overlap if and only if $-t_i + t_j$ is interior to $P_i \oplus -P_j$. The layout consists of copies of P_i at t_i plus all lattice vectors. If two copies of polygons intersect, one can assume w.l.o.g. that one of the two lattice vectors is $(0,0)$. Hence, for some t_i and t_j and lattice vector $i_0v_0 + i_1v_1$, $P_i + t_i$ will overlap $P_j + t_j + i_0v_0 + i_1v_1$. But, $\langle i,j,i_0,i_1\rangle \in L^\infty$, and therefore

$$-t_i + t_j + i_0v_0 + i_1v_1 = w_{\langle i,j,i_0,i_1\rangle} = u_{\langle i,j,i_0,i_1\rangle} \in U_{\langle i,j,i_0,i_1\rangle} = \overline{P_i \oplus -P_j}.$$

Hence, these two copies of P_i and P_j do not intersect, contradicting the assumption that some pair intersects. □

Theorem 2.2 *Every non-overlapping lattice packing with area $|v_0 \times v_1| \leq A$ corresponds to a zero-objective solution to one of the root overlap minimization problems.*

Proof. 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 $[-30,30],[60,120]$.

A non-overlapping lattice packing has no overlap, and therefore for each $\langle i,j,i_0,i_1\rangle \in L^\infty$,

$$-t_i + t_j + i_0v_0 + i_1v_1 \in \overline{P_i \oplus -P_j} = U_{\langle i,j,i_0,i_1\rangle}.$$

Hence we can set,

$$w_{\langle i,j,i_0,i_1\rangle} = u_{\langle i,j,i_0,i_1\rangle} = -t_i + t_j + i_0v_0 + i_1v_1.$$

Given Equations 1 to 4 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,

$$\begin{aligned} v_0 \times v_1 &= (w_{0x}b_{00} + w'_{0x}b_{10}) \times (w_{0y}b_{10} - w'_{0y}b_{00}), \\ &= (w_{0x}w_{0y} + w'_{0x}w'_{0y})b_{00} \times b_{10}. \end{aligned}$$

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,

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

which implies that $w_0 \in H_0 = U_0$ and we can set $u_0 = w_0$. Similarly for w_1.

Note: if *either* w'_{0x} or w'_{0y} (or w'_{1x} or w'_{1y}) is zero, then the approximation *equals* the true area. □

2.2 Area Minimization

A solution to overlap minimization is not necessarily a solution to lattice packing, since its area may still be greater than A. The *area minimization* problem is to minimize the true area function $|v_0 \times v_1|$ under the constraint that the overlap plus the difference between estimated area and target area equals zero. This constraint is equivalent to setting the w's equal to the u's: $w_0 = u_0$, $w_1 = u_1$, and $w_{\langle i,j,i_0,i_1\rangle} = u_{\langle i,j,i_0,i_1\rangle}$ for all $\langle i,j,i_0,i_1\rangle \in L^\infty$. The *root* area minimization problems are the two area minimization problems that correspond to the root overlap minimization problems. It follows from Theorem 2.2 that solutions with area $|v_0 \times v_1| \leq A$ to lattice packing correspond to solutions with area $|v_0 \times v_1| \leq A$ to area minimization.

2.3 Recognizing Local Minima

The algorithm of the next section solves overlap minimization and area minimization. An important part of this algorithm is the generation of *local* minima. A local minimum of either problem is a setting for the variables such that any infinitesimal change which respects the constraints will also increase the objective. This section describes how to detect local minima.

Let U be a polygonal region and let $u \in U$ be a point inside it. A *convex approximation* to U *near* u is a convex polygonal subset $I \in U$ such that $u \in I$ and such that for some open set $O \subseteq U$, $u \in O$ and $O \cap I = O \cap U$. In other words, I "looks like" U in the vicinity of u. If w lies outside U, then a *convex approximation* I to U *near* w is a convex approximation to U near u for some point $u \in U$ which minimizes $|w - u|$.

Let $U = C \cap H$, where C is a convex polygonal region and $H = \{(x,y) \,|\, xy \leq a\}$ for some constant a. If $u_x u_y < a$ and hence u does not lie on the curve $xy \leq a$, set $U' = U$. Else set $U' = U \cap \{(x,y) \,|\, u_y x + u_x y \leq 2a\}$: U intersected with the half-plane tangent to $xy \leq a$ at u. A *convex approximation* $I \subseteq U$ to U near u is defined to be a convex approximation to U' near u, under the rules of the previous paragraph.

Theorem 2.3 *A setting of the variables for either overlap minimization or area minimization is a local minimum if they are a local minimum when each U (U_0, U_1, $U_{\langle i,j,i_0,i_1\rangle}$) is replaced by a convex approximation near the corresponding u (u_0, u_1, $u_{\langle i,j,i_0,i_1\rangle}$).*

Proof. Clearly, replacing a polygonal U by a convex approximation I makes no difference with respect to infinitesimal changes in u. However, if U is U_0 or U_1 and has a segment of a concave hyperbola $xy = a$ as part of its boundary, and if $u_x u_y = a$, then I can only be tangent to $xy = a$ at u, but it cannot contain an open subset $O \subseteq U$ about u because I is convex. Nevertheless, if the system were not at a local minimum, then for some trajectory $u \to u + s\Delta u$, the objective has negative derivative with respect to s at $s = 0$. This trajectory has to point into (or along the boundary of) the half-plane tangent to u at $xy = a$. Therefore, the trajectory would also diminish the objective for the approximated system, contradicting the assumption that the approximated system is at a local minimum. □

This theorem has an important *technological* implication because it means we can verify local minima using a linear or quadratic program solver. If we replace each U by a convex approximation, then overlap minimization becomes an instance of *quadratic programming*, and we can verify that the system is at a local minimum using a quadratic program solver. For area minimization, the objective $v_0 \times v_1$ is quadratic but not positive definite and therefore we cannot use quadratic programming to verify a minimum. However, we can linearize the objective about its current value $v_0 \times v_1$,

$$(v_0 + \Delta v_0) \times (v_1 + \Delta v_1) \approx v_0 \times v_1 - v_1 \times \Delta v_0 + v_0 \times \Delta v_1, \tag{11}$$

and solve the resulting linear program. If this linear program is at a (global) minimum, then area minimization is at a local minimum.

3 The Overlap/Area Minimization Algorithm

This section describes an algorithm that takes an overlap minimization problem as input. It either 1) finds a solution to the corresponding area minimization problem with area $\leq$ the target A or 2) reports that no solution exists. If it can find a solution, it returns a solution whose lattice area is at a *local minimum*. To solve the lattice feasibility problem, we run this algorithm on the two root overlap minimization problems and a decreasing sequence of target areas. Since the algorithm returns a local minimum area (when successful), only a few such calls are required to bracket the global minimum area in a small interval.

The overlap/area minimization algorithm uses *restriction*, *evaluation*, *compaction*, and two kinds of *subdivision*. *Restriction* replaces one or more U regions by a subset. *Evaluation* moves the system to a local minimum of the overlap (plus approximate excess area) measure in Equation 8. If that minimum is not zero, the algorithm *subdivides* the problem into two subproblems and recurses on each. If the minimum is zero, then *compaction* moves the system to a local minimum of the area minimization problem. If the lattice area $|v_0 \times v_1| \leq A$, the target area, then it reports success. If the area is greater than the target, the algorithm employs a different type of *subdivision* to generate two new overlap minimization problems and recurses on each.

Section 3.1 describes how the algorithm avoids dealing with an infinite set L^∞ of lattice indices. Section 3.2 gives the restriction algorithm. Section 3.3 gives the evaluation algorithm and proves that it reaches a local overlap minimum. Section 3.4 gives the compaction algorithm and proves that it reaches a local area minimum. Section 3.5 gives the subdivision algorithms and proves that each makes it impossible for the algorithm to revisit the current local minimum in either of the subproblems. This ensures that the algorithm does not go into an infinite loop.

Note: it may be possible to prove a stronger convergence property for the algorithm. However, since packing is NP-hard, one cannot expect to prove anything other than an exponential running time bound, which has little use in practice.

3.1 Finite Sets of Lattice Indices

Obviously, we cannot compute with an infinite lattice index set L^∞. What we do is select a small finite set L, as described below. If the algorithm "thinks" it is successful, it then checks an appropriately large set of lattice indices for a possible overlap. If a pair of polygons is overlapping, the algorithm adds the corresponding lattice index to L and calls itself recursively with the *same* overlap minimization problem (and augmented L).

The initial L has only lattice indices $\langle i, j, i_0, i_1 \rangle$ such that $|i_0|, |i_1| \leq 1$. W.L.O.G., it suffices that each index in L satisfies $i < j$ and $-1 \leq i_0 \leq 1$ and $-1 \leq i_1 \leq 1$ or it satisfies $i = j$ and $0 < i_0$ or it satisfies $i = j$ and $i_0 = 0$ and $0 \leq j_0 \leq 1$.

3.2 Restriction

In our previous work, we employed both *geometric* restriction and *LP (linear programming)* restriction. The former only works well if the packing problem has a fixed container, which lattice packing does not. In any case, we do not see how to generalize geometric restriction to lattice packing.

LP restriction [11] starts with a convex linear *relaxation* of the minimization problem. Using iterated linear programming, it generates a convex polygonal *range* R for each variable in the relaxed problem. The range for the relaxed problem is a superset of the range of the original unrelaxed minimization problem. Restriction then intersects each range R of the relaxed problem with the corresponding non-convex region-valued variable U of the original unrelaxed problem. If $R \cap U$ is empty, the original problem has no solution; otherwise, restriction sets the new U equal to $R \cap U$. LP restriction repeats this operation for all ranges R until the fraction of area removed from some U falls below some threshold.

Section 3.2.1 gives a relaxation of the overlap minimization problem. The technique for computing the ranges from the relaxed problem appears in previous work [11]. Section 3.2.2 describes how to intersect the resulting ranges with the planar U regions.

3.2.1 Relaxation of Overlap Minimization

The relaxed problem must permit every setting of the variables with the objective in Equation 8 equal to zero; otherwise, it would not be a valid relaxation! If the zero objective is feasible then it is possible to set $w_0 = u_0 \in U_0$ and similarly for the other w's, u's and U's. In the relaxed problem, w_0 is constrained to lie in the convex hull $\mathrm{HULL}(U_0)$ (instead of non-convex and non-polygonal U_0), and similarly, $w_1 \in \mathrm{HULL}(U_1)$ and $w_{\langle i,j,i_0,i_1 \rangle} \in \mathrm{HULL}(U_{\langle i,j,i_0,i_1 \rangle})$, for each $\langle i, j, i_0, i_1 \rangle \in L$.

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

We would like to constrain $t_1, t_2, t_3, \ldots, t_k$ to lie in the parallelogram with vertices $(0, 0)$, v_0, $v_0 + v_1$, and v_1: the primary fundamental parallelogram.[5] However, this would require non-linear 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

[5] As previously stated, the polygons themselves do not necessarily lie in the fundamental parallelogram. However, it is possible to choose their translations $t_1, t_2, t_3, \ldots, t_k$ to lie in this parallelogram.

structure of this octagon is fixed, and the constraint that t_i lies in this octagon is convex and linear.

Finally, the relaxed problem has all the linear constraints which are present in the unrelaxed overlap minimization problem: "slave" relations (Equations 1 through 4 and 7) and the constraints in the set S. However, neither the u variables nor any constraints on them appear in the relaxed problem.

3.2.2 Intersecting Ranges with Regions

In the case of overlap minimization, LP restriction intersects the range of w_0 with the set U_0 and sets U_0 to the result. Similarly, it intersects U_1 with the range of w_1 and $U_{\langle i,j,i_0,i_1\rangle}$ with the range of $w_{\langle i,j,i_0,i_1\rangle}$. This last case involves only the intersection of convex polygonal regions with non-convex polygonal regions. We implement this operation in floating point using *nearest pair rounding* [13].

Let R_0 be the range of w_0 in the relaxed problem. Since U_0 is already the intersection of a convex set with H_0 (Equation 5), it suffices to replace U_0 with $R_0 \cap H_0$ (instead of $R_0 \cap U_0$). The intersection of a convex polygonal region with H_0 can at worst have multiple connected components, each with a single concave hyperbolic arc. Using the quadratic formula, it is simple to determine if each edge of R_0 has zero, one, or two intersections with the boundary of H_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_0 both lie inside H_0, then the segment pq must cross the boundary of H_0 an *even* (zero or two) number of times, no matter what the numerics say!

3.3 Evaluation

Evaluation starts with the same set of constraints as (the linear relaxation used by) restriction and adds additional constraints to force the u's to lie inside the U's. In particular, for w_0, it computes a point $u_0 \in U_0$ which minimizes $|w_0 - u_0|$, selects a convex approximation $I(w_0, U_0)$ to U_0 near u_0, and adds the constraint $u_0 \in I(w_0, U_0)$. It acts similarly for w_1 and each $w_{\langle i,j,i_0,i_1\rangle}$. This approach improves on our previous work [10] which uses a convex set C which "looks like" U in the vicinity of w, but may or may not be a subset.

Given the restriction constraints plus the additional convex approximation constraints, evaluation minimizes the overlap minimization objective of Equation 8. Since the constraints are all linear and the objective is a sum of squares, a quadratic program solver can find the minimum. Evaluation then computes new u's and convex approximations and minimizes again. It repeats until the quadratic program cannot improve the objective. By Theorem 2.3, evaluation terminates on a local minimum of the objective.

To finish the description of evaluation, it suffices to provide algorithms for computing the convex approximations. Section 3.3.1 addresses the hyperbolic regions U_0 and U_1, and Section 3.3.2 deals with the other, purely polygonal, U's.

3.3.1 The Hyperbolic Regions

A hyperbolic region U (U_0 or U_1) is at worst a union of disjoint "almost-convex" sub-regions: each sub-region has a convex chain on one side and a concave hyperbolic arc on the other. For a given w (w_0 or w_1), evaluation first sets u (u_0 or u_1) equal to the point of U which lies closest to w, and it selects the sub-region which contains it. It calculates the point u' on the hyperbolic arc which

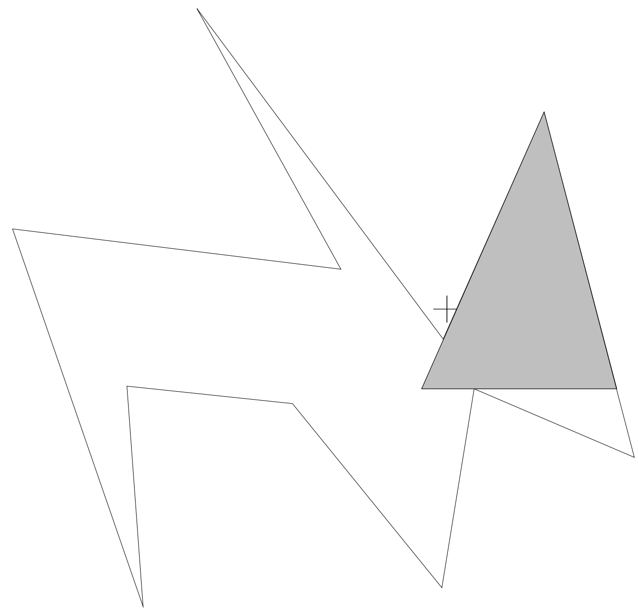

Figure 4: Selecting convex subset near to point (cross).

lies closest to u. The convex approximation $I(w, U)$ is the intersection of the subregion with the half-plane $\{(x, y) \mid u'_y x + u'_x y \leq 2u'_x u'_y\}$ tangent to the hyperbolic arc at u'.

3.3.2 The Polygonal Regions

Given a w and a polygonal region U, $I(w, U)$ is a convex subset of U which lies near to w. To construct I, the algorithm first sets $I = U$, which is non-convex, and then performs a series of half-plane "cuts" on I: replace I by $I \cap H$, where H is a half-plane. We say that we "cut I by an edge e of U" if H is the natural half-plane $H(e)$ defined by e: $H(e)$ has e as part of its bounding line and it lies to the "interior side" of e. We say that we "cut I by a vertex v (with respect to w)" if H is $H(v, t)$: the half-plane which contains w and whose boundary contains v and is perpendicular to the segment wv.

The convex subset selection has two stages, each of which performs a series of half-plane cuts on I. After both stages have completed, I is a convex approximation to U near w. Stage I ensures that I contains a nearest point $u \in U$ to w, and Stage II ensures that it is convex and "as large as possible".

Stage I: If w is not interior to I (which equals U at this point), then cut I by the half-planes of all nearest edges of I to w (the nearest edge may not be unique).

Stage II: Repeat the following step. Select the nearest element, vertex or edge, of I to w that is not a subset of the boundary of one of the previous cutting half-planes. Cut I by this edge or vertex. Stop when I is convex.

3.4 Compaction

Let us suppose that evaluation succeeds in driving the overlap measure to zero. This will place each w (w_0, w_1, $w_{\langle i,j,i_0,i_1 \rangle}$) inside its corresponding U. All pairs of polygons (or at least the pairs represented by L) will be non-overlapping, but the true lattice area $v_0 \times v_1$ may not be less than the target.

Compaction drives the true area to a local minimum. First, it applies all the previous constraints. To these it adds the constraint that each w equals its corresponding u. Finally, using Equation 11, it linearizes the true area near its current value and minimizes the linearized objective using linear programming. Since the true area is not a convex function (otherwise we would just use quadratic programming!), this compaction step may increase the true area. However, some interpolation of the compaction step does decrease the true area. It suffices to solve a quadratic equation to determine the best interpolant. This overall step (LP plus interpolation) is repeated until a local minimum is reached.

3.5 Subdivision

If either evaluation fails to drive the overlap to zero or if compaction fails to drive the true area below the target area, then subdivision divides the problem into two subproblems. Each subproblem has the property that evaluation/compaction cannot reach the same local minimum that evaluation/compaction reached for their parent.

If evaluation fails, then subdivision selects the w, u pair which is farthest apart. It introduces a cutting plane midway between them and perpendicular to uw. In one subproblem u and w must lie to one side, and in the other subproblem they must lie on the other side. Hence the current values of w and u cannot occur again.

If compaction fails, then subdivision cuts one of the angle ranges at v_0 or v_1. The properties of the approximate area function ensure that v_0 and v_1 lie *interior* to the angle ranges because if either lay at an extreme, then one of the two approximate areas $((w_{0x}w_{0y})(b_{00} \times b_{10})$ or $(w_{1x}w_{1y})(b_{01} \times b_{11}))$ would equal the true area. (See the note after Theorem 2.2.) Since evaluation succeeded, both approximate areas are $\leq$ the target area. Since compaction failed, the true area is greater than the target. Therefore, neither approximate area equals the target area. The subdivision algorithm chooses the cut that minimizes the maximum angle range in the subproblems.

Each subproblem inherits S (Section 2.1.1) from its parent. Subdivision adds an additional linear constraint to each S corresponding to the positive or negative side of the cutting plane. Finally, constraints on w_0 resulting from $w_0 \in \text{HULL}(U_0)$ are converted to linear constraints on v_0, v_1 and added to S for each child, and similarly for w_1.

Theorem 3.1 *The local minimum resulting from an unsuccessful evaluation or unsuccessful compaction cannot reappear in either of the two subproblems resulting from subdivision.*

Proof. If evaluation is unsuccessful, it is clear that a cut halfway between w and u will prevent both of these points for ever appearing *simultaneously* in any subproblem.

If evaluation is successful but compaction is unsuccessful, suppose that subdivision cuts the range b_{00}, b_{01} at the generator v_0 of the resulting local minimum. This particular value of v_0 can only appear at an extreme of an angle range of a subproblem. By the note of Theorem 2.2, the estimated area for this local minimum equals the true area because one of its generators lies at the extreme of a range. Since the true area for this local minimum is greater than the target area, so is its estimated area (in each subproblem). However, evaluation/compaction cannot yield a local minimum whose estimated area is greater than the target. $\square$

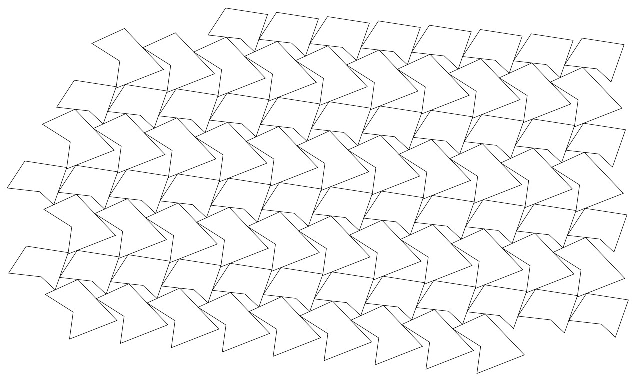

Figure 5: Input 1: lattice packing of two pentagons.

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 Linux. We use 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-degree rotated twins. Using the method described in Section 2.1.2, we constrained such pairs to be "nearly touching". The selected polygon B is a square about 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 $U_{\langle i,j,i_0,i_1\rangle}$ other than the one just subdivided, we employed a faster but weaker form of restriction. Specifically, we only calculated the bounding, axis-parallel, rectangle of the range R instead of R 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 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 degree rotated copies, and 4) the same with respect to (2). Figures 5 through 7 illustrate Examples 1-3. Figures 2 and 3 illustrate Example 4.

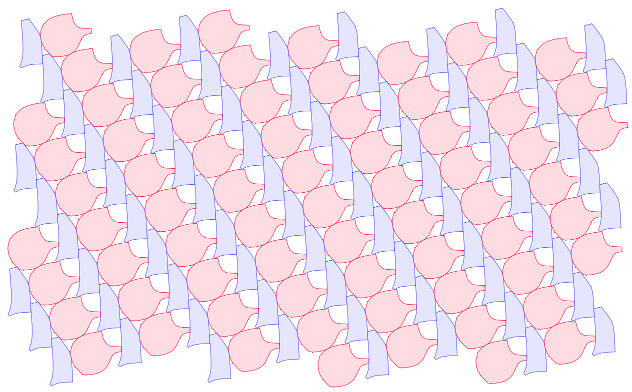

Figure 6: Input 2: lattice packing of two bra parts.

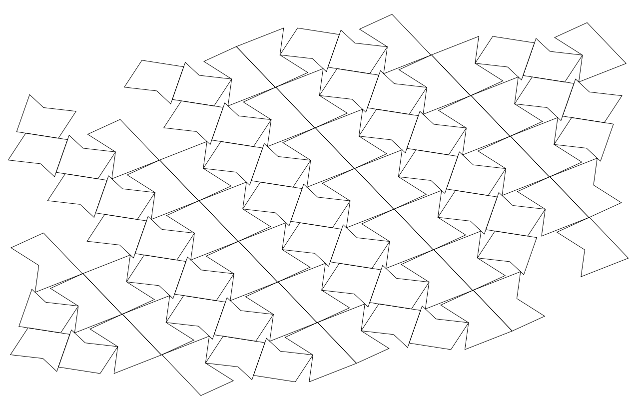

Figure 7: Input 3: lattice packing of four pentagons.

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 these 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 0.01% times the solution's area. 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 Figure 3. The total time was half an hour (1869 seconds). This solution has density 90.2%, and when arbitrarily clipped to the rectangle in Figure 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). Figure 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 Figure 7!

Fortunately, for those layouts for which verification is possible, it appears that we can obtain tight lower bounds on the optimal area. Table 4.3 shows the dependence of the running time on ϵ, where the target area is $(1-\epsilon)A_{\text{optimal}}$, 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".

4.4 Conclusions and Future Work

The lattice packing in Figure 2 has density 90.4% (in the plane). By contrast, the human-generated layout in Figure 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

Table 1: Experiments on negative examples $(1-\epsilon)$ times the optimal area.

$e = -\log_2 \epsilon$	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
$e = -\log_2 \epsilon$	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	1.1	1.11	1.08	1.14

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 main problem is that of "clipping" the lattice to a strip or rectangle and, if possible, generating a lattice that clips well. Heuristic tools already exist for the other remaining problems. For example, after packing a set of replicated parts of "average" size using lattice packing and clipping, one can "randomly" replace the parts with the actual sizes. Our existing compaction/overlap minimization tools [7, 8], or extensions, should be able to remove the overlaps and generate a dense strip packing. 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.

5 Acknowledgements

The author thanks Harald Schmidl for help in proofreading.

References

[1] K. Daniels. *Containment algorithms for nonconvex polygons with applications to layout.* Ph.D. thesis, Harvard University, Cambridge, MA, 1995.

[2] K. Daniels and V. J. Milenkovic. Multiple Translational Containment, Part I: An Approximate Algorithm. *Algorithmica*, 19:148–182, 1997.

[3] K. Dowsland and W. Dowsland. Solution approaches to irregular nesting problems. *European Journal of Operational Research*, 84(3):506–21, August 1995.

[4] K. A. Dowsland and W. B. Dowsland. Packing Problems. *European Journal of Operational Research*, 56:2 – 14, 1992.

[5] H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operations Research*, 44:145–159, 1990.

[6] G. Kuperberg and W. Kuperberg. Double-lattice packings of convex bodies in the plane. *Discrete Comput. Geom.*, 5:389–397, 1990.

[7] Z. Li. *Compaction algorithms for nonconvex polygons and their applications.* Ph.D. thesis, Harvard University, Cambridge, MA, 1994.

[8] Z. Li and V. Milenkovic. Compaction and separation algorithms for nonconvex polygons and their applications. *European Journal of Operations Research*, 84:539–561, 1995.

[9] V. Milenkovic. Translational polygon containment and minimal enclosure using linear programming based restriction. In *Proc. 28th Annu. ACM Sympos. Theory Comput. (STOC 96)*, pages 109–118, 1996.

[10] V. Milenkovic and K. Daniels. Translational polygon containment and minimal enclosure using geometric algorithms and mathematical programming. Technical Report 25-95, Center for Research in Computing Technology, Division of Applied Sciences, Harvard University, Cambridge, MA, 1995.

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

[12] V. J. Milenkovic. Multiple Translational Containment, Part II: Exact Algorithms. *Algorithmica*, 19:183–218, 1997.

[13] V. J. Milenkovic. Rotational polygon containment and minimum enclosure using only robust 2d constructions. *Computational Geometry: Theory and Applications*, 13:3–19, 1999.

[14] D. M. Mount. The densest double-lattice packing of a convex polygon. Report 2584, Dept. Comput. Sci., Univ. Maryland, College Park, MD, 1991.

[15] P. E. Sweeney and E. R. Paternoster. Cutting and Packing Problems: A Categorized, Application-Oriented Research Bibliography. *Journal of the Operational Research Society*, 43(7):691–706, 1992.

[16] P. Whelan and B. Batchelor. Automated packing systems: review of industrial implementations. In *Proceedings of the SPIE - The International Society for Optical Engineering, vol.2064*, pages 358–69. SPIE, September 1993.