Proceedings of the 13th Annual Symposium on Computational Geometry (SoCG), June 4-6, 1997.

Rotational Polygon Overlap Minimization

Victor J. Milenkovic
University of Miami
Department of Mathematics and Computer Science*

Abstract

An effective and fast algorithm is given for *rotational overlap minimization*: given an overlapping layout of polygons $P_1, P_2, P_3, \ldots, P_k$ in a container polygon C, translate and rotate the polygons to a layout that minimizes an overlap measure. A (local) overlap minimum has the property that any perturbation of the polygons increases the chosen measure of overlap. Experiments show that the algorithm works well in practice. It is shown how to apply overlap minimization to create algorithms for other layout tasks: *compaction*, *containment*, and *minimal enclosure*. **Compaction**: starting with a non-overlapping layout in a rectangular container, plan a non-overlapping motion that minimizes the length or area of the container. **Containment**: place the polygons into a (possibly non-convex container) without overlapping. **Minimal enclosure**: find a non-overlapping layout inside a minimum-length, fixed-width rectangle or inside a minimum area rectangle. All of these algorithms have important industrial applications.

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. Stripes, plaids, or other patterns on the fabric can further limit the allowed orientations and translations. Nevertheless, apparel manufacturers often allow small rotations (called "tilting"), typically not more than 3 degrees, but still large enough to make a significant difference in cloth utilization. Glass and sheet metal (and sometimes leather) have no grain (or stripes or plaids), and therefore layout on these materials allows arbitrary orientations for the parts.

From the point of view of computational geometry, there are at least four problems related to layout.

Overlap Minimization Given an overlapping layout of polygons P_0, P_1, P_2, ..., P_k, move the polygons to a layout that minimizes some measure of overlap. Usually, P_0 is the unbounded complement $\overline{C}$ of a *container* polygon.

Compaction Given a non-overlapping layout of P_1, P_2, ..., P_k, plan a motion that minimizes some measure. For most textile applications, the polygons are inside a rectangular container R with fixed width and the goal is to minimize the length. Both overlap minimization and compaction generate *local* minima.

Containment Given polygons P_1, P_2, ..., P_k and a fixed container C, place the polygons into C without overlapping. As in the case of overlap minimization, P_0 denotes $\overline{C}$, in which case the goal becomes to find a non-overlapping layout of P_0, P_1, ..., P_k.

Minimum Enclosure Find a non-overlapping layout of P_1, P_2, ..., P_k that minimizes some measure of the container. The most common version in the textile industry is *strip packing*: minimize the length of a rectangular container R of fixed width. It is also useful to minimize the area of R.

These four problems are closely related mathematically and all have important applications in industry. Unfortunately, all are NP-hard or PSPACE-hard if k is not a constant. Section 1.3 discusses our previous work on theory, algorithms, and commercially successful software for *translational* versions of the problems. One of the keys to all of these algorithms is a fast technique for *overlap minimization* using a combination of techniques from computational geometry and mathematical programming. In particular, compaction is basically a special case of overlap minimization, and overlap minimization is essentially "half" of the solution to containment and minimum enclosure.

1.1 New Results.

This paper gives a new algorithm for *rotational* overlap minimization. Like our translational compaction and overlap minimization algorithms, it finds the minimum through the solution of a sequence of linear programs. Our experiments demonstrate that, like our translational algorithms, the new

*P.O. Box 249085, Coral Gables, FL 33124, http://www.cs.miami.edu/~vjm, vjm@cs.miami.edu. This research was funded by the Textile/Clothing Technology Corporation from funds awarded by the Alfred P. Sloan Foundation, by NSF grant CCR-91-157993, and by a subcontract of a National Textile Center grant to Auburn University, Department of Consumer Affairs.

rotational algorithm has "supernatural" convergence. We say "supernatural" because, unlike standard physically based energy minimization techniques, our algorithm anticipates collisions and overlaps before they happen and therefore converges in fewer steps than the "natural" methods.

For convex P_i and P_j, this paper defines $\mathrm{OVERLAP}(P_i, P_j)$ to be the length of the shortest translation of P_j that moves it off P_i. For non-convex P_i and P_j, we define a more complicated measure for $\mathrm{OVERLAP}(P_i, P_j)$ but one which matches the intuitive notion of overlap. In principle, our algorithm can minimize any monotone (see Section 2.2) differentiable function of the pairwise overlaps. Also, if the user desires, the algorithm can (independently) limit the range of motion or range of angle for each polygon P_i. In our experiments, we minimize the overlap measure,

$$\sum_{0 \leq i < j \leq k} \mathrm{OVERLAP}(P_i, P_j).$$

The orientation/position of $P_0 = \overline{C}$, the complement of the container, is fixed, and the other polygons are unconstrained.

This paper also gives new algorithms for rotational compaction and containment which use the new rotational overlap minimization algorithm. The containment algorithm also uses our previous algorithms for translational containment. Previous work [6, 18] has shown that fast algorithms for translational containment and compaction can be used to create algorithms for translational minimum enclosure. The same reasoning works for the rotational case. Implementing rotational versions of these algorithms is a subject for future work. In particular, rotational containment requires the development of robust algorithms for *range intersection*: the intersection of all copies of a polygon as it rotates through a range of angles. The final version of a practical rotational containment algorithm may be different than the one given here, but it will certainly make heavy use of overlap minimization.

The following section discusses the significance of the new work in rotational layout: in theory and in practice. Section 1.3 compares the new algorithms to related work. Finally, Section 1.4 gives a road-map to the rest of the paper.

1.2 Significance.

Overlap minimization and compaction take an existing (overlapping or non-overlapping) layout and move the polygons to a *local* minimum of some measure (overlap or container length or area). Analogously, containment and minimum enclosure seek *global* minima.[1] One might think that finding a global minimum is theoretically more difficult or practically more significant than finding a local minimum, but such is not necessarily the case. Translational containment and minimum enclosure are "merely" NP-complete[2] [6, 19], although it is unknown whether rotational containment is in NP. In contrast, compaction and overlap minimization are likely to be PSPACE-hard. Also, among all the algorithms, compaction is by far the most commercially successful algorithm to date. All four problems are theoretically interesting and practically significant.

1.2.1 Theoretical Significance.

In his Ph.D. thesis [14], Li proved the PSPACE-completeness of *global compaction*: plan a non-overlapping motion to the global minimum length layout reachable from the current layout. Li used a modification of the hardness proof for the Warehouseman problem [13]. It is also likely that many forms of *local* compaction (the type dealt with in this paper) are also PSPACE-hard, depending on the answers to the following questions. 1) Is it possible to implement a totally mechanical 3D Turing machine? 2) What if parts are only allowed to translate? 3) What about in 2D? A positive answer to any of these questions is equivalent to a proof of PSPACE-hardness for compaction in the corresponding domain. The answer to (1) is certainly yes. Question (2) is likely, and (3) is definitely still open. However, we have shown how to build a translational 2D "counter" [15] which is "powered" by (local) compaction. This means that any "reasonable" compaction algorithm has lower bound running time exponential in k. "Reasonable" means the algorithm generates a description of the compaction motion as part of its internal operation. It can easily be shown that all hardness results for compaction also apply to overlap minimization.

1.2.2 Practical Significance.

Companies cannot simply generate a few layouts which they heavily reuse to satisfy their needs. At least three forces act to drive companies to generate as many layouts as possible. 1) Producing a greater variety of styles, sizes, and items increases a company's market share. 2) To keep inventories low, companies need to flexibly generate different mixes of sizes, styles, and items to closely track varying demand. 3) Stock dimensions are not standardized: rolls of cloth vary in width from 55 to 60 inches, and the width available might not be known until the day of delivery from the supplier. We know of a single large clothing manufacturer that generates 1500 layouts of 100 or more parts per week–all by hand![3] Nevertheless, the compensation for the layout workers is less that 1% of the money spent on cloth. Thus, if computers are to replace humans, they must generate layouts with nearly the same density. Unfortunately for computers, trained talented humans generate very dense layouts: 89-91% cloth utilization for men's pants, for example.

Compaction is currently the most practically significant algorithm because it can take human-generated layouts as input and improve them. Our translational compaction algorithm [18, 17] can compact an industrial layout of over one hundred clothing parts in 4 or 5 iterations and 1 or 2 minutes on a PC. For some domains, such as intimate apparel, compaction can reduce cloth waste by over 1%, and the average reduction is about 0.5%. For men's pants, the average reduction is about 0.25%. For one manufacturer with \$100 million in cloth cost, compaction saves them about \$250,000 per year. Anticipated yearly savings for customers already under contract[4] is \$2,750,000, and the customer base is growing

[1] Actually, containment only asks if the global minimum overlap is zero or non-zero. Our containment algorithms could be modified to find the actual global minimum overlap layout if the minimum is non-zero, but that is not such an interesting question in practice.

[2] Containment is the feasibility question and minimum enclosure is the optimality question. From the point of view of complexity theory, they are the same question.

[3] Actually, the hand is moving a mouse and using a CAD system which provides some assistance in single-polygon placement: automatically nestling it up against neighboring polygons.

[4] Customers are under contract to Gerber Garment Technologies, which currently holds exclusive license to our compaction

rapidly. Textile manufacturers do not allow arbitrary rotation of the parts because of the grain of the fabric. However, they often allow parts to "tilt" (industry term for a small rotation) a few degrees. Li attempted to add small rotations to translational compaction (Section 1.3). Even though the tilts are small, our experiments on 162 industrial layouts this early version of rotational compaction have shown that rotational compaction saves at least 49% more cloth than translational compaction alone. For industries cutting sheet metal, glass, and (sometimes) leather, the parts can have arbitrary orientations. For these industries, it is essential that compaction and the other layout algorithms can handle rotation and translation.

Compaction is essentially a special case of overlap minimization, but overlap minimization is useful directly as part of *database driven layout* [14, 16]. Companies with large databases of existing layouts can use them as starting point, substitute parts of different style or different but closely matched size, and generate new layouts. This idea has been in the apparel industry for a number of years, and the main obstacle has been the problem of re-laying the parts without overlaps. Clearly, overlap minimization is an excellent starting point, although other techniques might also be needed. We see this as next commercially significant application after compaction. In apparel applications, *rotational* overlap minimization could take advantage of tilting, and it would be essential for glass and sheet metal applications.

In the apparel industry, no layout software has ever replaced a human. Optimal strip packing (a version of minimum enclosure) for $k > 100$ parts would replace humans, but NP-completeness will probably prevent fast enough solutions for k significantly larger than 10. Nevertheless, as Daniels demonstrates in her Ph.D. thesis [6], and we subsequently confirm [7], fast containment algorithms for modest k are an extremely useful tool for building heuristics or approximate algorithms for large values of k (some of our heuristics match human performance a significant fraction of the time). It is clear that *rotational* containment algorithms will similarly permit more powerful layout heuristics in domains with arbitrarily oriented parts.

1.3 Related Work.

In the literature, there are essentially four approaches to layout of non-convex polygons: 1) heuristics and meta-heuristics (neural nets, simulated annealing, genetic algorithms), 2) physical simulation, 3) "classical" computational/combinatorial geometry, 4) a combination of computational geometry and mathematical programming (which we use here).

Heuristics and meta-heuristics are limited in theory because they cannot say "no" if there is no solution. Currently, they are limited in practice to falling about 5% behind humans in cloth utilization. See [6, 9, 10, 21] for a survey of published results. As stated in the previous section, heuristics will ultimately be *necessary* to solve large layout problems, but it is clear that they are not *sufficient* by themselves.

Another approach one might take to compaction or layout is to treat the polygons as solid physical objects, add some potential energy functions, and then use numerical methods for simulating physics. However, this approach is very slow in practice for many crowded parts. Experiments have shown that physically based techniques are much slower than our translation compaction algorithm [14, 16] for compacting typical layouts of hundreds of polygons. In recent work [20], we show that a generalization of our translational compaction algorithm can realistically animate 1000 spheres in an hourglass at least 1000 times faster than the latest techniques for true physical simulation of translating 3D solids. We expect the same to be true when rotation is added.

Chazelle [5] introduced the single-polygon containment problem: place m-gon P into n-gon container Q. For convex Q, the running time bound is $\mathrm{O}(mn^2)$, and for non-convex P and Q, $\mathrm{O}(m^3n^3(m+n)\log(m+n))$. Avnaim *et al.* [4, 3] improve this to $\mathrm{O}(m^3n^3\log(m+n))$. Most recent work deals with finding the largest copy of a convex P that can be placed, which is equivalent to finding the minimum (scaled) enclosure. For convex Q the best running time is $\mathrm{O}(mn^2\log n)$ [1], and for non-convex Q, $\mathrm{O}(m^2n^2)$ [2]. Grinde and Cavalier use an extensive case analysis and linear programming to place a single convex P [11] or convex P_1 and P_2 [12]. The running time of the first algorithm appears to be $\mathrm{O}(m^2n^3)$, and it is not clear what the running time of the second is, but for one of its cases they are able to use parametric programming and find a solution by solving $\mathrm{O}(m^4n^4)$ linear programs. This is the only previous result on multi-polygon rotational containment we are aware of.

For multi-polygon *translational* layout, we have had considerable theoretical and practical success using a combination of computational geometry and mathematical programming (CG/MP) [8, 19, 18]. As previously stated, these algorithms can practically solve translational compaction for hundreds of polygons and containment for up to ten polygons. These algorithms heavily rely on a model for compaction/overlap minimization described below. Using the same approach, we have 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 [17].[5]

All of our previous translational layout algorithms compute an explicit configuration space, $U_{ij} = \overline{P_i \oplus -Pj}$, for each pair of polygons, where $\oplus$ is the Minkowski sum. Translational compaction/overlap minimization selects a convex subset $C_{ij} \subseteq U_{ij}$ containing or near to $t_j - t_i$, where t_i is the current translation of P_i. (The work defines the overlap of P_i and P_j to be the distance $t_j - t_i$ lies outside U_{ij}.) The overlap minimization algorithm sets up a linear program to diminish the distance of each $t_j - t_i$ from C_{ij}. Compaction sets up a linear program to keep $t_j - t_i$ inside C_{ij} and minimize the length of the container.

Like the translational algorithms, the rotational algorithms presented in this paper use a CG/MP approach, but it differs considerably in detail. With the addition of a rotation parameter, the configuration space U_{ij} becomes a six-dimensional set with curved surface boundaries. At present, this space is much too complex to compute robustly. Li had previously attempted to use the translational configuration spaces as an approximation to the rotational spaces for rotational compaction [14]. On many inputs, this version of compaction only goes part of the way to a local minimum. The algorithms presented here use a different approach that does not require the configuration spaces to be explicitly constructed. (Nor do they use an extensive case analysis as does Grinde and Cavalier.) Some of the methods used here are standard in the (vast) quadratic programming literature, but we are not aware of any previous work on overlap minimization of non-convex polygons.

software.

[5] This paper also surveys other results in multi-polygon translational layout.

1.4 Outline.

Section 2 gives an algorithm for rotational overlap minimization of *convex* polygons. Section 3 generalizes it to non-convex polygons. Section 4 describes how to apply it to accomplish compaction, containment, and minimum enclosure. Section 5 gives experimental results.

2 Overlap Minimization

This section gives an algorithm for rotational overlap minimization of *convex* polygons. It first describes a non-linear optimization problem for overlap minimization of convex polygons. This problem is linearized, and interpolation is applied to find an overlap-diminishing step. This step is repeated until the system converges to a local minimum.

2.1 Notation

In this paper, lower case letters generally denote scalars, points, and vectors. Upper case letters denote sets of points, usually polygonal regions. Scalar multiplication and vector/point addition are defined as usual. This section employs three vector multiplications:

$$
\begin{aligned}
a \cdot b &= a_x b_x + a_y b_y, & (1)\\
a \times b &= a_x b_y - a_y b_x, & (2)\\
ab &= (a_x b_x - a_y b_y, a_x b_y + a_y b_x). & (3)
\end{aligned}
$$

Equation 1 is the standard dot product; 2 is a 2D scalar cross product; and 3 is the product of complex numbers. The last is very useful for representing rotations: if $u = (\cos\theta, \sin\theta)$, then up is p rotated by angle θ about the origin. Any operation on points/vectors can be generalized to point-sets. For instance, $P+a = \{p+a \mid p \in P\}$ and $uP = \{up \mid p \in P\}$.

2.2 Non-linear Formulation.

This section describes a non-linear optimization problem which is equivalent to minimizing the overlap of a collection of convex polygons.

Let $P_0, P_1, P_2, \ldots, P_k$ be convex polygons. Let t_i and $u_i = (\cos\theta_i, \sin\theta_i)$ be vector variables representing the translation and rotation of P_i, $0 \le i \le k$. Under the notation of the previous section, $u_i P_i + t_i$ represents polygon P_i rotated by θ_i and translated by t_i. Let L_{ij} be the line which *best separates* $u_j P_j + t_j$ from $u_i P_i + t_i$, $0 \le i < j \le k$: 1) L_{ij} is tangent to $u_j P_j + t_j$ and 2) $u_i P_i + t_i$ protrudes the minimum distance o_{ij} into the half-plane containing $u_j P_j + t_j$ (Figure 1). Let u_{ij} be the unit normal vector to L_{ij}, pointing into the half-plane containing $u_j P_j + t_j$, and let d_{ij} be the distance from the origin to L_{ij} along the direction u_{ij}.

It is easy to prove that L_{ij} is parallel to some edge of $u_i P_i$ or $u_j P_j$ and that $o_{ij} = \mathrm{OVERLAP}(u_i P_i + t_i, u_j P_j + t_j)$, which this paper defines as the length $|t_{ij}|$ of the shortest translation vector t_{ij} such that $u_j P_j + t_j + t_{ij}$ does not overlap $u_i P_i + t_i$.

For $0 \le i < j \le k$, the separation condition can be expressed as,

$$(u_i p_i + t_i) \cdot u_{ij} \le d_{ij} + o_{ij}, \qquad (4)$$

for all $p_i \in P_i$, and

$$(u_j p_j + t_j) \cdot u_{ij} \ge d_{ij}, \qquad (5)$$

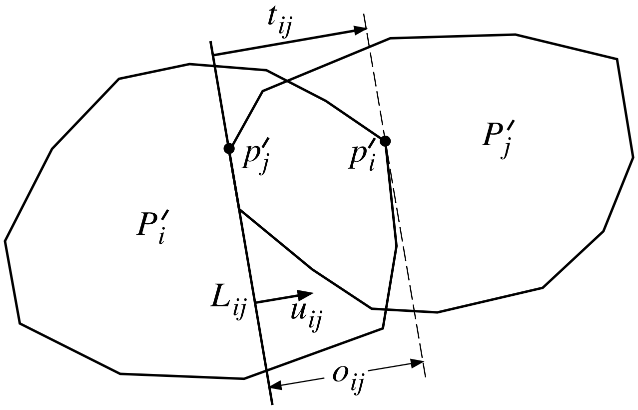

Figure 1: Line L_{ij} best separates overlapping convex polygons $P'_i = u_i P_i + t_i$ and $P'_j = u_j P_j + t_j$. Vertices $p'_i = u_i p_i + t_i$ and $p'_j = u_j p_j + t_j$ are extremal with respect to the direction u_{ij} of minimum overlap o_{ij}. (Extremal vertices are not necessarily unique.) Vector t_{ij} is the shortest translation of P'_j that will move it off of P'_i.

for all $p_j \in P_j$. Since P_i and P_j are convex polygonal regions, it is sufficient to allow p_i and p_j to range over the *vertices* of P_i and P_j, respectively. Since overlaps cannot be negative, we add the constraint $o_{ij} \ge 0$. We refer to these as the *separation constraints.*

Overlap minimization can be expressed as minimizing some *monotone* objective function, $F(o_{ij}, 0 \le i < j \le k)$ on the pairwise overlaps while preserving the separation constraints. A function F is *monotone* in each o_{ij} if increasing any single o_{ij} increases F or leaves it the same. Examples of monotone F are 1) the maximum o_{ij}, 2) the sum of the o_{ij}, and 3) the sum of the squares of the o_{ij}. In our experiments, we use a weighted sum $\sum_{0 \le i < j \le k} w_{ij} o_{ij}$ with non-negative weights.

2.3 Linearizing.

This section describes a linearization of the minimization problem of the previous section, resulting in an *overlap minimizing linear program* (OMLP). In this case, $t_i, u_i, u_{ij}, d_{ij}, o_{ij}$ represent *current* (fixed) values and $\Delta t_i, \Delta\theta_i, \Delta\theta_{ij}, \Delta d_{ij}, \Delta o_{ij}$ represent the perturbations of these values. (It is more convenient to use $\Delta\theta_i$ and $\Delta\theta_{ij}$ for the perturbations and u_i and u_{ij} for the unperturbed values.)

Substituting, $t_i + \Delta t_i$ and so forth in Equations 4 and 5 and discarding quadratic terms yields,

$$
\begin{aligned}
(u_i p_i) \times u_{ij} \Delta\theta_i + u_{ij} \cdot \Delta t_i - (u_i p_i + t_i) \times u_{ij} \Delta\theta_{ij} \\
-\Delta d_{ij} - \Delta o_{ij} \quad \le \quad d_{ij} + o_{ij} - (u_i p_i + t_i) \cdot u_{ij},
\end{aligned} \qquad (6)
$$

$$
\begin{aligned}
(u_j p_j) \times u_{ij} \Delta\theta_j + u_{ij} \cdot \Delta t_i - (u_j p_j + t_j) \times u_{ij} \Delta\theta_{ij} \\
-\Delta d_{ij} \quad \ge \quad d_{ij} - (u_j p_j + t_j) \cdot u_{ij}.
\end{aligned} \qquad (7)
$$

To keep the overlaps non-negative, $\Delta o_{ij} \ge -o_{ij}$.

The linearized objective is

$$\sum_{0 \le i < j \le k} F_{ij} \Delta o_{ij}, \qquad (8)$$

where F_{ij} is the partial derivative of F with respect to o_{ij} at the fixed values of the variables. For our experiments, $F_{ij} = w_{ij}$.

The linearized optimization problem is always feasible because one can set all the Δ-variables to zero. In a neighborhood of zero, the difference between the linearized problem and the original problem is second-order. Therefore if the original optimization has a "negative derivative" in some direction, then so will the linearized problem. To be mathematically precise, we have to include all critical points including unstable equilibria in our definition of "local overlap minimum". As long as the system is not at a critical point, the linearized optimization problem will have a non-trivial solution.

2.4 Interpolation.

Because of non-linearities, the solution to the linearized system usually does not necessarily have smaller overlap than the original configuration. However, we can interpolate between the current configuration and the new configuration: for $0 < s < 1$, replace t_i by $t_i + s\Delta t_i$, θ_i by $\theta_i + s\Delta\theta_i$, and so forth. The linearization ensures that for sufficiently small s, the interpolated configuration *will* have smaller overlap.

One can solve exactly for the optimum value of s, but the set of equations is messy and at least fourth degree. Our current implementation uses sampling and interval halving. Abusing notation, let $F(s)$ be the overlap for interpolation value s. We know that $F'(0) < 0$, and we are seeking $s_{\min}$, the first minimum of $F(s)$ for $0 < s \leq 1$. If we have two sample values $s_a < s_b$ such that $F(s_a) < F(s_b)$, then we know that $s_{\min} < s_b$, and we can "throw away" sample intervals to the right of s_b. In our implementation, we alternate between sampling the midpoint of 1) the longest interval and 2) the last (rightmost) interval between samples.

A good value of s yields a new configuration with diminished overlap. The linearization and interpolation are repeated until the system converges to a local overlap minimum. It is highly unlikely that the system would converge to an unstable equilibrium in practice.

3 Non-Convex Polygons

One might think that generalizing overlap minimization to non-convex polygons would involve the convex hull, but in fact the trick is to use *inner* convex approximations instead. This section describes how to define OVERLAP(P, Q) for non-convex polygons based on inner convex approximations and to use this definition to generalize the overlap minimization algorithm. Implementation issues are also addressed.

3.1 Defining an OVERLAP Measure.

Let $\mathcal{C}_\infty(P)$ be the set of all convex subsets of P. Define

$$\text{OVERLAP}_\infty(P,Q) = \max_{C\in\mathcal{C}_\infty(P), D\in\mathcal{C}_\infty(P)} \text{OVERLAP}_\infty(C,D).$$

This measure has two important "intuitive" properties: 1) for convex P, Q, OVERLAP$_\infty(P,Q) =$ OVERLAP(P,Q), and 2) for non-convex P, Q, OVERLAP$_\infty(P,Q) = 0$ if and only if P and Q are not overlapping.

We believe that there is a polynomial-time algorithm for calculating OVERLAP$_\infty$. However, we do not know how to deal with the infinity with respect to the reduction to linear programming. We find it necessary to select a subset $\mathcal{C}(P_i) \subset \mathcal{C}_\infty(P_i)$ for each input polygon P_i, and use an approximation to OVERLAP$_\infty$ which we call "OVERLAP(P_i, P_j)",

$$\text{OVERLAP}(P_i,P_j) = \max_{C_i\in\mathcal{C}(P_i), C_j\in\mathcal{C}(P_j)} \text{OVERLAP}(C_i,C_j).$$

This measure still has the two "intuitive" properties if 1) $P_i \in \mathcal{C}(P_i)$ if P_i is convex, and 2) $\cup_{C_i\in\mathcal{C}(P)} C_i = P_i$.

Our initial experiment showed that is good to choose $\mathcal{C}(P_i)$ whose elements overlap. If instead $\mathcal{C}(P_i)$ is chosen to be a *partition* of P, such as the triangulation, then OVERLAP(P_i, P_j) can have "unintuitive" local minima. Suppose L_i separates $\mathcal{C}(P_i)$ into two subsets, and suppose the same for L_j and $\mathcal{C}(P_j)$. In this case OVERLAP$(u_iP_i + t_i, u_jP_j + t_j)$ has a valley for all u_i, t_i, u_j, t_j which align $u_iL_i + t_i$ with $u_jL_j + t_j$. Section 5.1 discusses our current choice of $\mathcal{C}(P_i)$.

3.2 The Overlap Minimization Algorithm.

Given a choice of $\mathcal{C}(P_i)$ for P_i, $0 \leq i \leq k$, it is easy to generalize the algorithm of Section 2. Simply apply the previous algorithm to the set of convex polygons $\cup_{0\leq i\leq k}\mathcal{C}(P_i)$ with two modifications: 1) all the polygons $C_i \in \mathcal{C}(P_i)$ have the *same* translation/rotation variables t_i, u_i, and 2) each pair of polygons $C_i \in \mathcal{C}(P_i)$ and $C_j \in \mathcal{C}(P_j)$ use the *same* overlap variable o_{ij}. We can also drop any constraints between $C_i, C_i' \in \mathcal{C}(P_i)$.

3.3 Reducing Running Time.

The rotational overlap minimization algorithm as described in Sections 2 and 3.2 is very inefficient. It creates the constraints of Equations 6 and 7 for each choice $\langle C_i, p_i, C_j, p_j\rangle$, for $0 \leq i < j \leq k$, for $C_i \in \mathcal{C}(P_i)$, for vertex $p_i \in C_i$, for $C_j \in \mathcal{C}(P_j)$, and for vertex $p_j \in C_j$. To reduce avoid considering this huge set of constraint choices ($\Omega(k^2n^4)$ if each P_i has n vertices), we use a *dynamic* approach. The minimization step "guesses" a much smaller set S of constraint choices. If after taking a step it determines that critical constraint choices were missing, it adds them to the set S and starts over.

```
S ← ∅
CreateConstraints()
TakeAStep()
while (UpdateConstraints() > 0)
      Restart()
      TakeAStep()
```

For each $0 \leq i < j \leq k$, the function CreateConstraints() computes a maximally overlapping pair $C_i \in \mathcal{C}(P_i)$ and $C_j \in \mathcal{C}(P_j)$ and adds constraint choices to S of the form $\langle C_i, p_i, C_j, p_j\rangle$, where $p_i' = u_ip_i + t_i \in C_i' = u_iC_i + t_i$ lies on L_{ij} (Section 2.2) and $p_j' = u_jp_j + t_j \in C_j' = u_jC_j + t_j$ lies the maximal distance from L_{ij} on the same side as C_i. The function TakeAStep() sets up and solves the OMLP (Section 2.3) and performs interpolation (Section 2.4), using *only* the elements of S. The function UpdateConstraints() does the same as CreateConstraints() with the new positions of the polygons and returns the number of *new* constraints it added to S. Finally, Restart() moves the polygons back to their starting positions. **NOTE**: this section describes a single overlap minimization step. When UpdateConstraints() returns 0, the set S is verified, and the algorithm has taken a successful step.

4 Compaction, Containment, and Minimum Enclosure using Overlap Minimization

This section gives algorithms that use overlap minimization to do compaction and to solve containment and minimum enclosure problems.

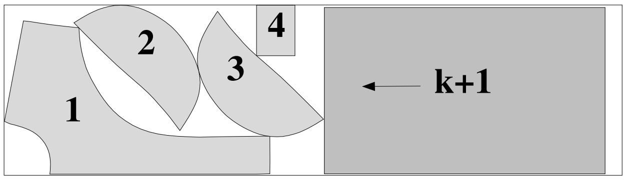

Figure 2: LMLP: Additional term in objective function acts as leftward "force".

4.1 Compaction.

To perform compaction using overlap minimization, let P_0 be the complement $\overline{R}$ of the rectangular container, and introduce an additional infinite "piston" rectangle P_{k+1} to the right of all the polygons as shown in Figure 2. Start with the overlap minimization linear program (OMLP) for this configuration. Add a term $c \cdot x_{k+1}$ to the objective function, where c is a constant and x_{k+1} is the x-coordinate of t_{k+1}, the translation of P_{k+1}. Finally add constraints so that P_0 is fixed and P_{k+1} cannot rotate and can translate only in the x-direction. We call this the *length minimizing* linear program (LMLP). Finally, adding non-overlap constraints, $o_{ij} = 0$, $0 \leq i < j \leq k+1$, yields the non-overlap length minimizing linear program (NOLMLP). To compact, iterate the LMLP with a small value of c: $c = 0.001$ and $w_{ij} = 1$ $0 \leq i < j \leq k$ (Section 2.3) until convergence. Iterate the NOLMLP to get rid of overlaps.

To minimize the area of the container rather than its length, we use the same strategy with an *area minimizing* linear program (AMLP). In this case, P_0 is the complement of the first quadrant, $P_0 = \{(x,y) \,|\, x \leq 0 \text{ or } y \leq 0\}$, and P_{k+1} is the complement of the third quadrant, $P_{k+1} = \{(x,y) \,|\, x \geq 0 \text{ or } y \geq 0\}$. We remove any penalty for overlap between P_0 and P_{k+1}: in the objective, set the weight $w_{0,k+1}$ of $o_{0,k+1}$ to zero. Translation t_0 is fixed at $(0,0)$ and angles θ_0 and θ_{k+1} are fixed at zero. The goal is to minimize the area of the rectangle with diagonal $t_0 t_{k+1}$. This rectangle has area $x_{k+1} y_{k+1}$, where $(x_{k+1}, y_{k+1}) = t_{k+1}$. It can be shown [17], that this area can be diminished if and only if its gradient $y_{k+1} \Delta x_{k+1} + x_{k+1} \Delta y_{k+1}$ can be diminished. To finish the creation of the AMLP, we add the terms $c\, y_{k+1} \Delta x_{k+1} + c\, x_{k+1} \Delta y_{k+1}$ to the objective.

4.2 Containment.

This section gives an algorithm for rotational containment which uses overlap minimization plus our successful algorithm for translational containment [18].

4.2.1 Range Intersection.

Let $P(\theta)$ denote polygon P rotated by angle θ about its local origin. Under our previous notation, $P(\theta) = uP$ for $u = (\cos\theta, \sin\theta)$. Let $P[\alpha, \beta]$ denote the *range intersection* of P,

$$P[\alpha, \beta] = \bigcap_{\alpha \leq \theta \leq \beta} P(\theta).$$

The range intersection $P[\alpha, \beta]$ has the property that if it cannot be placed in a container, then no rotated copy $P(\theta)$ of P can be placed in the container for any θ in $[\alpha, \beta]$.

The output of range intersection is a *polycircle*: a planar region bounded by circular arcs and straight line segments. Furthermore, each arc is concentric with the center of rotation. For purposes of applying our current translational containment algorithm, we would have to take an inner polygonal approximation. (Actually, since polycircles are closed under set operations, Minkowski sum, and convex hull, it should be in principle possible to generalize our translational containment algorithm to polycircles because these are the only operations that this algorithm uses.) In the rest of the section, $P[\alpha, \beta]$ denotes an polygonal approximation to P. The full paper discusses the issue of selecting an appropriate accuracy for the approximation. The resulting rotational containment algorithm will *not* be approximate because it uses overlap minimization to find the actual solution.

4.2.2 Containment Algorithm.

As stated in Section 1, the goal of containment is to place polygons $P_1, P_2, \ldots, P_k$ into a container C. The rotational containment algorithm first establishes an initial *range hypothesis*, $\mathcal{R}^{\text{init}} = \langle [\alpha_i^{\text{init}}, \beta_i^{\text{init}}],\, 1 \leq i \leq k \rangle$, representing the possible angles which a containment solution might have. If P_i has limits on its rotation, α_i^{init} and β_i^{init} are set to the bounds of this limit. In particular, if P_i is not allowed to rotate, $\alpha_i^{\text{init}} = \beta_i^{\text{init}} = 0$. If P_i^{init} can rotate arbitrarily, $[\alpha_i^{\text{init}}, \beta_i^{\text{init}}] = [0, 2\pi]$.

The rotational containment algorithm applies two types of operations to a range hypothesis $\mathcal{R} = \langle [\alpha_i, \beta_i],\, 1 \leq i \leq k \rangle$.

- **Evaluate**: attempt to find a solution to containment whose angles fall within these ranges.
- **Subdivide**: for some i, split $[\alpha_i, \beta_i]$ into $[\alpha_i, \mu_i]$ and $[\mu_i, \beta_i]$. Create two sub-hypotheses $\mathcal{R}'$ and $\mathcal{R}''$ by replacing $[\alpha_i, \beta_i]$ by either $[\alpha_i, \mu_i]$ or $[\mu_i, \beta_i]$ in $\mathcal{R}$.

4.2.3 Evaluation.

To evaluate a hypothesis $\mathcal{R} = \langle [\alpha_i, \beta_i],\, 1 \leq i \leq k \rangle$, the algorithm runs translational containment on the range intersections $P_1[\alpha_1, \beta_1], P_2[\alpha_2, \beta_2], \ldots, P_k[\alpha_k, \beta_k]$. If there is no solution, then by the property of range intersection, the algorithm can discard the entire hypothesis. If translational containment returns a solution $t_1, t_2, \ldots, t_k$, then the rotational algorithm uses this solution as a starting point for overlap minimization. For $1 \leq i \leq k$, it gives P_i translation t_i and angle $\theta_i = (\alpha_i + \beta_i)/2$. It then runs overlap minimization on $P_1, P_2, \ldots, P_k$. If the minimum overlap is zero, it has found a solution! Otherwise, it subdivides the hypothesis.

4.2.4 Subdivision.

To subdivide the hypothesis, the algorithm selects the polygon P_i which has the largest overlap with its neighbors. Let θ_i now denote the angle of P_i returned by overlap minimization. If θ_i is in the interior of the interval $[\alpha_i, \beta_i]$, it subdivides this interval at $\mu_i = \theta_i$. If θ_i equals α_i or β_i, it splits the interval at its midpoint. The algorithm then recurses on the two sub-hypotheses.

4.3 Minimum Enclosure.

In her Ph.D. thesis [6], Daniels describes how to find a fixed width, minimum length rectangular enclosure for polygons $P_1, P_2, \ldots, P_k$. This method uses containment and compaction. She observes that containment plus binary search can be used to determine the minimum feasible length. Compaction can speed this search by shrinking any feasible length to a local minimum before continuing the binary search.

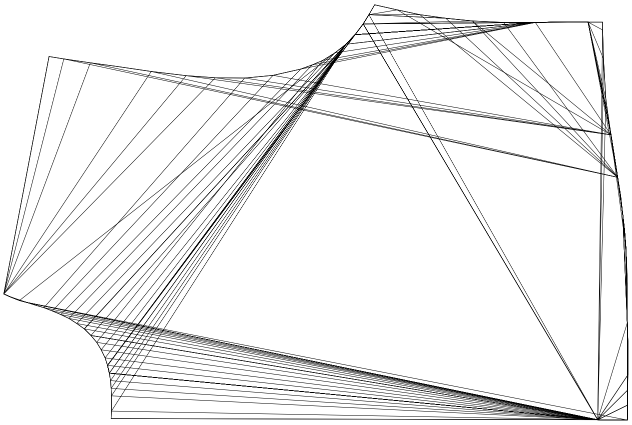

Figure 3: Convex cover of non-convex apparel polygon: 71 vertices, 39 polygons in cover.

Daniels also suggested that the minimum area rectangle can at least be approximated by sampling different widths. In more recent work [18], we show how to use containment directly to solving the following problem without sampling: given area A, is there a layout whose enclosing rectangle R has area $\mathrm{AREA}(R) \leq A$? This algorithm can be combined with area compaction to rapidly find a minimum area enclosing rectangle. *All* of these methods work equally well with *rotational* compaction and containment algorithms.

5 Experimental Results

We ran our experiments on polygonal parts from apparel manufacturing. Of course, the domains most in need of rotational algorithms are glass and sheet metal. In the full paper, we will also obtain data from manufacturers in these domains and run experiments on these parts. However, the apparel parts are representative of non-convex shapes which arise in manufacturing. All experiments were run on a DEC Alpha 3000/700 (name trademark of Digital Equipment Corporation) using CPLEX 3.0 (product of CPLEX Optimization, Inc.) to solve the linear programs. The standard "optimize" option (simplex method) was used.

5.1 Generating Convex Covers

As stated in Section 3.1, it is necessary to generate a subset $\mathcal{C}(P_i) \subset \mathcal{C}_\infty(P_i)$ for each input polygon P_i. Our initial experiments showed us that it is good for elements of $\mathcal{C}(P_i)$ to overlap as much as possible. Currently, our system generates a convex partition of P_i in a greedy fashion, and then greedily "fattens" each element of the partition. The partition algorithm repeatedly cuts P_i with a chord. A cut which eliminates two concave vertices is always better than a cut which only eliminates one. A cut which yields two boundary chains with roughly equal number of vertices is better than one which yields chains with very unequal numbers of vertices. The algorithm selects the best cut and recurses on the two halves. To fatten a convex partition element C_i, the algorithm greedily adds vertices from P_i to the boundary of C_i as long as the result is a convex subset of P_i. Figure 3 shows the result for a typical non-convex apparel part.

5.2 Experiments.

5.2.1 Overlap Minimization: One Polygon.

Figure 4 shows overlap minimization for the polygon in Figure 3. The container is the bounding box for the polygon. We applied a fairly large initial rotation and translation. The polygon minimized its overlap with the exterior of the container in 3 iterations. Running times: decomposition, 2.1 seconds; overlap calculations (OC), 1.5 seconds; linear programming (LP), 0.41 seconds. Iterations 1 and 2 required one auxiliary LP each (included in the 0.41 seconds) to dynamically determine constraints (Section 3.3).

5.2.2 Overlap Minimization: Two Polygons.

Figure 5 shows overlap minimization for two copies of the same polygon. The container is chosen to give a tight fit in the y-direction. Again, only 3 iterations are required. (If the y-dimension is even 1% larger, only 2 iterations are required.) Running times: decomposition, 4.2 seconds; OC, 23.7 seconds; linear programming (LP), 0.93 seconds. Iteration 1 required 2 auxiliary LP calls, and iteration 2 required 4 (included in 0.93 seconds) for dynamic constraint calculation.

Time for decomposition and LP grow linearly, but OC jumps by almost a factor of 10. This happens because we are (naively) calculating the overlap of every $C_1 \in \mathcal{C}(P_1)$ with every $C_2 \in \mathcal{C}(P_2)$ in order to calculate $\mathrm{OVERLAP}(u_1P_1 + t_1, u_2P_2 + t_2)$.

5.2.3 Compaction: 23 Polygons.

Figures 6 and 7 illustrate the even iterations for the LMLP (Section 4.1) applied to compact an actual industrial layout of 23 apparel part polygons. (These figures are rotated 90 degrees to put the piston on the "top".) Iteration 6 is within one part in 10^6 of optimal. Since it has no overlaps, it is not necessary to apply the NOLMLP. The length (and area) of the layout is diminished by 2.3%. Since the "geometric" overlap calculation (OC) dominates the cost of linear programming, we modified the program to ignore the results of interpolation. This increases the number of constraints added to each linear program, but greatly decreases the number of auxiliary linear programs. To make the cost estimate fair, we ran interpolation anyway, but discarded its output, setting $s = 1$ each time. Running times: 6 essential LP calls (one per iteration), 1.7 minutes; 52 auxiliary LP calls (8.7 per iteration on average), 15 minutes; geometric calculations (decomposition and OC), 333 minutes.

6 Conclusions

The full paper will include more experiments, but these three clearly demonstrate that the number of iterations is surprisingly few and that the cost of the *essential* linear programming is very small. The actual translations and orientations of the polygons in each iteration of the 23-polygon example are the output of just *six* linear programs which require less than a total of two minutes to solve. We will also do experiments to compare our method with methods based on physical modeling instead of linear programming. However, it is clear that these physical methods will require many more than six iterations.

Without the use of dynamic constraint constraint calculations, the two-polygon example takes almost 200 seconds, and linear programming dominates the running time. The

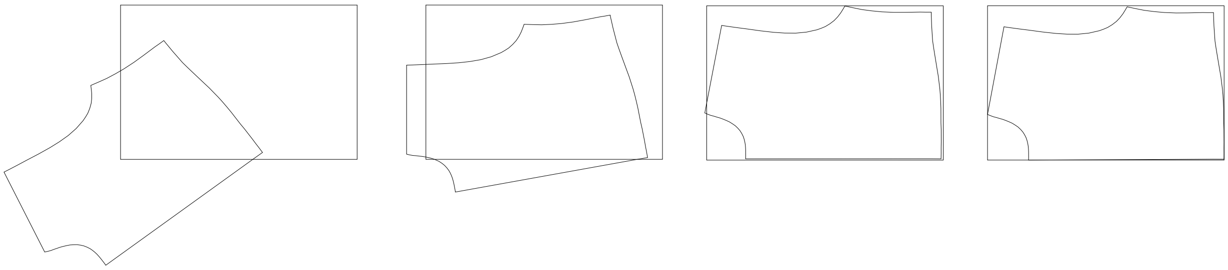

Figure 4: Overlap minimization for one polygon.

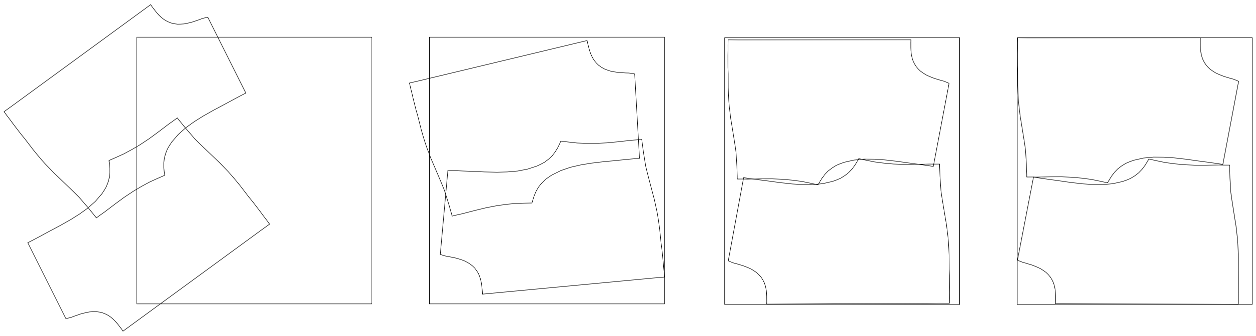

Figure 5: Overlap minimization for two polygons.

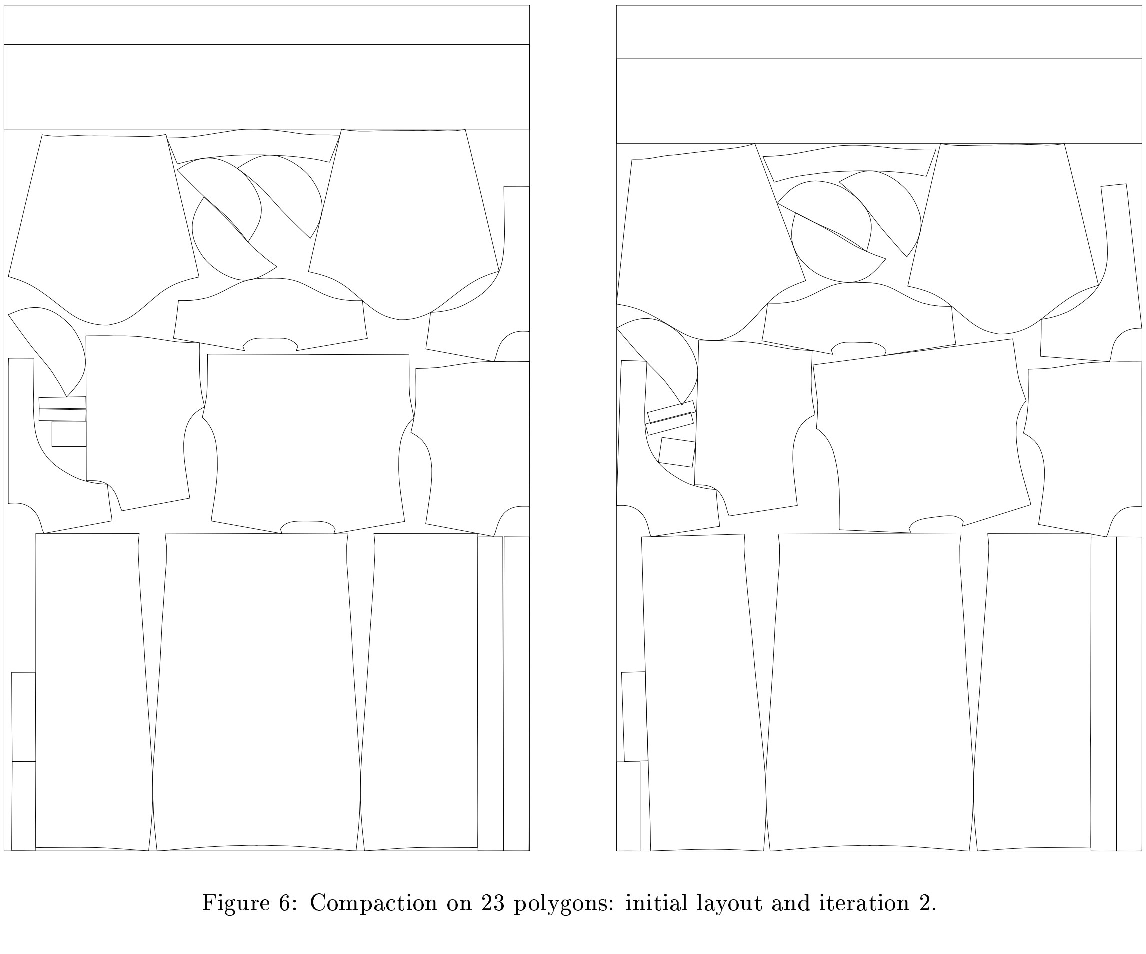

Figure 6: Compaction on 23 polygons: initial layout and iteration 2.

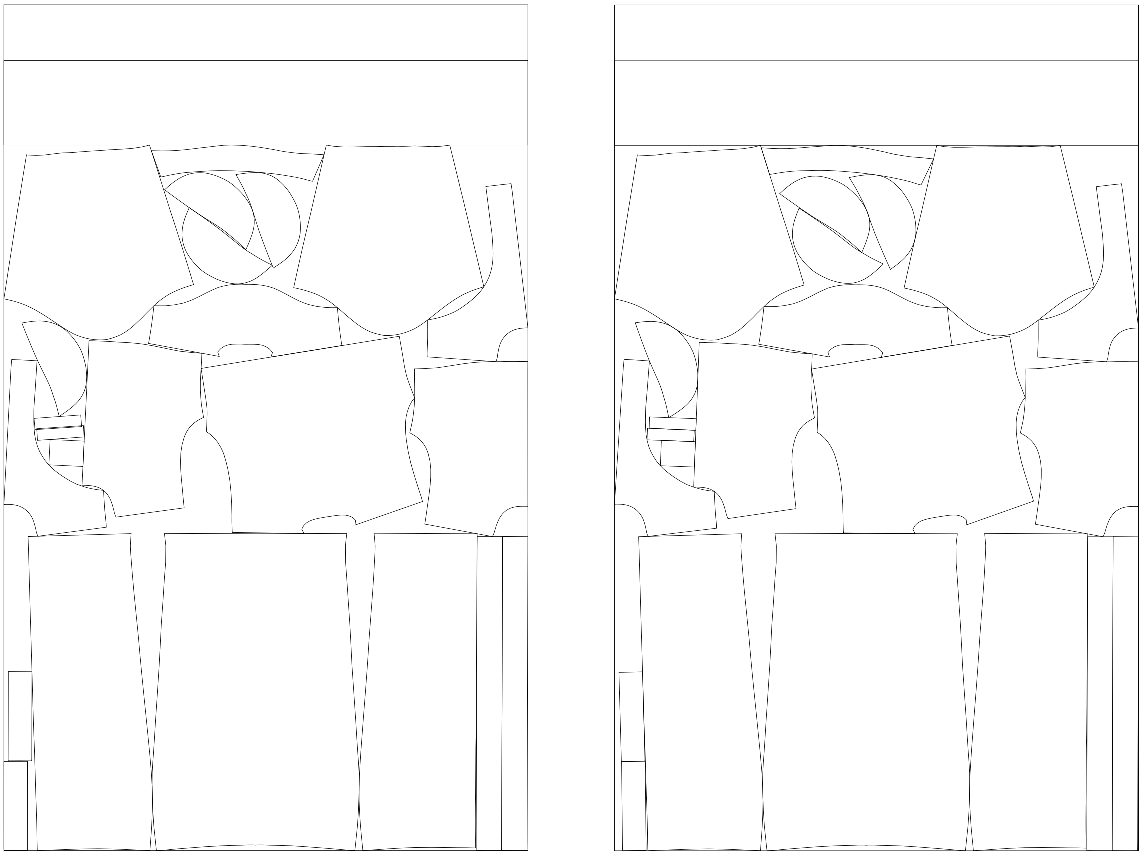

Figure 7: Compaction on 23 polygons: iterations 4 and 6.

23-polygon example would be practically impossible. With dynamic constraint calculations, the cost of linear programming, even including the auxiliary calls, is quite reasonable. Geometric calculations dominate the running time. Currently these are done quite naively, and it is likely they can be sped up by one or two orders of magnitude.

OPEN QUESTIONS: Is there a polynomial-time algorithm for $\mathrm{OVERLAP}_\infty$ (Section 3.1)? Is there a way to use it instead of the approximation OVERLAP? Alternatively, what choice of $\mathcal{C}(P_i) \subset \mathcal{C}_\infty(P_i)$ plus what algorithm for overlap calculation gives the fastest algorithm for OVERLAP?

References

[1] P. Agarwal, N. Amenta, and M. Sharir. Largest placement of one convex polygon inside another. In *Proceedings of the Second Workshop on Algorithmic Foundations of Robotics*, July 1996.

[2] P.K. Agarwal, B. Aronov, and M. Sharir. Motion planning for a convex polygon in a polygonal environment. In preparation., 1992.

[3] F. Avnaim. *Placement et déplacement de formes rigides ou articulées.* PhD thesis, Université de Franche-Comté, France, 1989.

[4] F. Avnaim and J. Boissonnat. Polygon placement under translation and rotation. In *Proceedings of the 5th Annual Symposium on Theoretical Aspects of Computer Science; Lecture Notes in Computer Science (294)*, pages 322–333. Springer-Verlag, 1988.

[5] B. Chazelle. The Polygon Containment Problem. In Preparata, editor, *Advances in Computing Research, Volume 1: Computational Geometry*, pages 1–33. JAI Press, Inc., Greenwich, Connecticut, 1983.

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

[7] K. Daniels and V. Milenkovic. Column-based strip packing using ordered and compliant containment. In *Proc. 1st ACM Workshop on Appl. Comput. Geom.*, pages 33–38, 1996.

[8] K. Daniels and V. J. Milenkovic. Multiple Translational Containment, Part I: An Approximate Algorithm. *Algorithmica, special issue on Computational Geometry in Manufacturing*, In press.

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

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

[11] R. Grinde and T. Cavalier. Containment of a Single Polygon Using Mathematical Programming. Technical Report IMSE Working Paper 92-164, The Pennsylvania State University, Department of Industrial and Management Systems Engineering, 1993.

[12] R. B. Grinde and T. M. Cavalier. A new algorithm for the two-polygon containment problem. *Computers and Operations Research*, page to appear, 1996.

[13] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects: P-space hardness of the "Warehouseman's Problem". *Internat. J. Robot. Res.*, 3(4):76–88, 1984.

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

[15] Z. Li and V. Milenkovic. The complexity of compaction problem. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 7–11, Waterloo, Canada, 1993.

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

[17] 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.

[18] 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.

[19] V. J. Milenkovic. Multiple Translational Containment, Part II: Exact Algorithms. *Algorithmica, special issue on Computational Geometry in Manufacturing*, In press.

[20] Victor Milenkovic. Position-based physics: simulating the motion of many highly interacting spheres and polyhedra. In *Computer Graphics*, pages 129–136, 1996. Proc. SIGGRAPH '96.

[21] 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.