ELSEVIER

Computational Geometry 10 (1998) 305–318

Computational Geometry
Theory and Applications

Rotational polygon overlap minimization and compaction ☆

Victor J. Milenkovic *

Department of Mathematics and Computer Science, University of Miami, P.O. Box 249085, Coral Gables, FL 33146, USA

Communicated by C.M. Hoffmann; submitted 5 August 1997; accepted 13 February 1998

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 Q, translate and rotate the polygons to diminish their overlap to a local minimum. A (local) overlap minimum has the property that any perturbation of the polygons increases the overlap. Overlap minimization is modified to create a practical algorithm for *compaction*: starting with a non-overlapping layout in a rectangular container, plan a non-overlapping motion that diminishes the length or area of the container to a local minimum. Experiments show that both overlap minimization and compaction work well in practice and are likely to be useful in industrial applications. © 1998 Published by Elsevier Science B.V.

Keywords: Layout; Packing or nesting of irregular polygons; Containment; Minimum enclosure; Compaction; Linear programming

1. Introduction

A number of industries generate new parts by cutting them from stock material: cloth, leather (hides), sheet metal, glass, etc. These industries need to generate dense non-overlapping layouts of irregular 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.

Variations of the layout problem have a variety of names: *nesting*, *packing*, and so forth. In the computational geometry literature, *containment* refers to the task of placing k input polygons

☆ 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 97-12401, and by a subcontract of a National Textile Center grant to Auburn University, Department of Consumer Affairs.

* http://www.cs.miami.edu.

PII S0925-7721(98)00012-1

$P_1, P_2, P_3, \ldots, P_k$ into a fixed container Q without overlapping. If the container varies over a class, such as the class of rectangles with a fixed given width and variable length, then *minimum enclosure* refers to the task of finding the minimum area container in the class which allows a layout.

Both in practice and in theory, minimum enclosure is a highly complex non-linear and non-convex global optimization problem. One can consider containment to be special case of another such optimization problem, *global overlap minimization*: find the layout of $P_1, P_2, P_3, \ldots, P_k$ in Q which minimizes the sum of the pairwise overlaps. (Containment has a solution if and only if the global minimum overlap is zero.) Even for purely translational containment and minimum enclosure (polygons cannot rotate), no layout software currently in use can consistently come as close to the optimum as hand layout by expert humans. These humans easily pay their own wages (actually about five times their wages) with the material they save over the best heuristics and algorithms.

This paper addresses *local* optimization versions of containment and minimum enclosure, called *overlap minimization* and *compaction*, respectively. The local optimization problems are simpler to solve than their global counterparts, yet our previous work on translational layout indicates that finding a local optimum is still quite useful. Translational overlap minimization forms an essential part of our algorithms for translational containment and minimum enclosure, and clothing manufacturers currently apply our translation compaction algorithm to human-generated layouts as a post-processing step. We expect rotational algorithms to be at least as beneficial.

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 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 these "natural" methods.

For convex polygons C_i and C_j, this paper defines overlap(C_i, C_j) to be the length of the shortest translation of C_j that moves it off C_i, also known as the *intersection depth* [8]. For non-convex polygons P_i and P_j, we define a different measure for overlap(P_i, P_j) but one which matches the intuitive notion of overlap. The current algorithm minimizes

$$\sum_{0 \leqslant i < j \leqslant k} \text{overlap}(P_i, P_j),$$

where $P_0 = \overline{Q}$ is the complement of the container, although in principle, it can minimize any monotone 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.

This paper also gives a new algorithm for *rotational compaction*. An implementation of the rotational compaction algorithm is shown to reduce material waste for a number of industrial layouts—even *after* translational compaction has been applied. An analysis of the running times shows that this rotational compaction is fast enough for many practical industrial applications.

The following section relates the new algorithms to other work in layout of irregular polygons. Section 2 gives an algorithm for rotational overlap minimization of *convex* polygons. Section 3 gen-

eralizes it to non-convex polygons and compaction. Section 4 gives experimental results and analyzes the running time, particularly for the compaction application.

1.2. Related work

Most of the work on layout has focused on heuristics or meta-heuristics for containment and minimum enclosure, often called *nesting*. Less work has focused on algorithms: techniques that are guaranteed to find the global minimum. Relatively little work has considered the problem of finding local minima, although there is considerable work in the area of physical simulation, which can generate a local minimum as a by-product. This section summarizes recent work in these areas and its relationship to the new algorithms for rotational overlap minimization and compaction.

Heuristics and meta-heuristics are limited in theory because they cannot say "no" if there is no solution. According to our sources in the apparel industry, available layout software is limited in practice to falling about 5% behind humans in cloth utilization. Recent work has used simulated annealing [14], boundary matching [15], grouping of polygons into sub-rectangles [1], genetic algorithms [4], incremental approach [24], database driven layout [16,18], or a hybrid approach [12,13]. See [7,9] for surveys of older work. Many of these approaches either discretize the search space, replace the polygons by approximations, and/or use other accuracy/time trade-offs. Rotational overlap minimization can help speed up these heuristics by moving these approximate layouts to the nearest exact non-overlapping layout, if one exists.

Chazelle [6] gave the first algorithm for the irregular (non-convex) single-polygon translational containment problem, and Avnaim and Boissonat [3] improve the running time. Agarwal et al. [2] give the best running times for single-polygon rotational minimum enclosure. Grinde and Cavalier [10] give a rotational containment algorithm for two convex polygons. Finally, we give an algorithm for k-polygon translational containment and minimum enclosure that is within a log factor of optimal [19]. It is not clear how overlap minimization can improve the theoretical running times of these algorithms. However, we have found overlap minimization to be essential to reducing the running time in practice [22,23].

Local minimization is simpler to achieve than global minimization, yet we have shown an exponential running time lower bound [17] for any "realistic" compaction algorithm: one which generates the motion of the polygons as part of the minimization. In the case of translational compaction, we have not seen this worst case in practice [16,18]. Stoyan et al. [25] have independently arrived at essentially the same approach for translational compaction. Our translational compaction algorithm is also much faster than a physically based approach. In three dimensions, the linear programming approach can be 1000 times faster than a physically based approach [20]. Since the rotational algorithms presented here also use iterated linear programs, we expect them to have the same advantages.

Our translational compaction algorithm [16,18] has been available to industry since 1993. Our feedback from industry indicates that humans indeed do not generate local minima and that compacting their layouts saves material without violating cutting rules. Our translational compaction algorithm can compact an industrial layout of over one hundred clothing parts in 4 or 5 iterations in well under a minute 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[1] is $2,750,000, and the customer base is growing rapidly.

What about rotation? Textile manufacturers do not allow arbitrary rotation of the parts because fabric has a grain. However, they often allow parts to "tilt" (industry term for a small rotation) a few degrees. Li attempted to add rotations to his translational compaction algorithm by rotating polygons one at a time. Even though this technique does not necessarily reach a local minimum, his experiments on 162 layouts [16] showed that adding rotations saved 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 both rotation and translation. Hence a good system for rotational compaction would be even more valuable to industry than the one for translational compaction has been.

2. Overlap minimization for convex polygons

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. Overlap of convex polygons

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: $a \cdot b = a_x b_x + a_y b_y$, $a \times b = a_x b_y - a_y b_x$ and $ab = (a_x b_x - a_y b_y, a_x b_y + a_y b_x)$. The first and second are the two dimensional dot and cross product. The third is the product of points as complex numbers, which is useful for representing rotations: if $u = (\cos\theta, \sin\theta)$, then the product up is point p rotated by θ about the origin. Vector operations are applied pointwise to sets: $uP + t$ is polygon P rotated by θ and translated by t.

Convex polygons A and B overlap if their interiors intersect. Vector t is a *separating translation* for A and B if $A - t/2$ and $B + t/2$ do not overlap, although their boundaries may intersect. Since boundary contact is permitted, the set of separating translations is a closed set, and therefore the set of lengths of separating translations has a minimum. Define the measure of overlap to be this minimum length $o = |t|$. Note that there may be more than one shortest separating translation. We can determine the overlap by solving the following optimization problem on (unit) vector variable u, $|u| = 1$, and scalar variables d and o, $o \geqslant 0$. Minimize o, given

$$u \cdot a \leqslant d + o/2, \ \forall a \in A, \quad \text{and} \quad u \cdot b \geqslant d - o/2, \ \forall b \in B. \tag{1}$$

Lemma 2.1. *The solution to Eq.* (1) *is the overlap* o *of* A *and* B *and a minimum separating translation* $t = o\,u$.

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

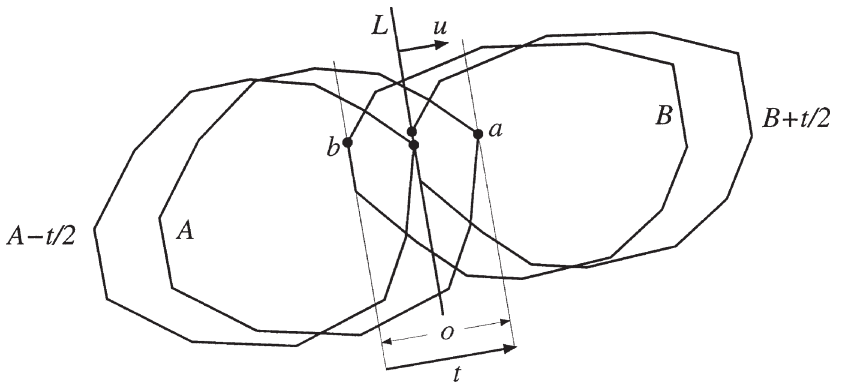

Fig. 1. Illustration of convex polygons A, B, a separating translation t of length o, and a separating line L for $A - t/2$ and $B + t/2$. Vertices a and b are critical (Section 3.2).

Proof. Let $\langle u, d\rangle$ denote the line $L = \{(x, y) \mid u \cdot (x, y) = d\}$. By Eq. (1), A lies to the left of line $\langle u, d + o/2\rangle$ and B lies to the right of line $\langle u, d - o/2\rangle$, and therefore $A - t/2$ and $B + t/2$ lie on opposite sides of L and do not overlap (see Fig. 1). Let t' be another separating translation, and let $L' = \langle u', d'\rangle$ ($|u'| = 1$) be the line which separates $A - t'/2$ from $B + t'/2$.[2] For all $a \in A$, $u' \cdot (a - t'/2) \leqslant d'$, which implies $u' \cdot a \leqslant d' + (u' \cdot t')/2$. Similarly, for all $b \in B$, $u' \cdot b \geqslant d' - (u' \cdot t')/2$. Since o is the minimum value satisfying Eq. (1), it follows that $o \leqslant u' \cdot t' \leqslant |t'|$ and thus no separating translation is shorter than $t = o\,u$. □

Lemma 2.1 implies that rotational overlap minimization for polygons $P_0, P_1, P_2, \ldots, P_k$ is the solution to a minimization problem on vector variables u_i, $|u_i| = 1$, and t_i, $0 \leqslant i \leqslant k$, and vector variable $u_{ij} = -u_{ji}$, $|u_{ij}| = 1$, and scalar variables $d_{ij} = -d_{ji}$ and $o_{ij} = o_{ji}$, $o_{ij} \geqslant 0$, $0 \leqslant i < j \leqslant k$. Minimize $\sum_{0 \leqslant i < j \leqslant k} o_{ij}$, given

$$u_{ij} \cdot (u_i p_i + t_i) \leqslant d_{ij} + o_{ij}/2, \quad \forall p_i \in P_i,\ 0 \leqslant i \neq j \leqslant k. \tag{2}$$

In Eq. (2), u_i and t_i represent the rotation and translation of polygon P_i, and o_{ij} is the overlap of polygon $u_j P_i + t_i$ with $u_j P_j + t_j$. This result follows as a corollary to Lemma 2.1.[3] This optimization problem can be modified to minimize the maximum overlap or some other monotonic function on the overlaps instead of the sum. It is also easy to add constraints on the range of rotation or translation of a polygon. In fact, since the container $Q = \overline{P_0}$ is not suppose to move, one would fix $u_0 = (1, 0)$ and $t_0 = (0, 0)$. Finally, since the constraint is linear in p_i, it suffices to allow p_i to range only over the *vertices* of P_i.

Eq. (2) is non-linear, and to solve it we first linearize it. We replace each variable which participates in higher order terms by a perturbed version: $u_{ij} \to u_{ij} + \Delta u_{ij}$, $u_i \to u_i + \Delta u_i$ and $t_i \to t_i + \Delta t_i$. Since $|u_{ij}| = 1$ and $|u_i| = 1$, the linearized approximations to Δu_{ij} and Δu_i are $\Delta\theta_{ij} u_{ij}^{\perp}$ and $\Delta\theta_i u_i^{\perp}$,

[2] Non-overlapping convex polygons can always be separated by a line [11,21].

[3] By setting $u_{ij} = -u_{ji}$, $d_{ij} = -d_{ji}$ and $o_{ij} = o_{ji}$, the two constraints in Eq. (1) are collapsed into one constraint in Eq. (2) for each i, j, $i \neq j$.

respectively, perpendicular to u_{ij} and u_i. Substituting perturbed variables, expanding, dropping higher order terms, and setting $p'_i = u_i p_i + t_i$ yields [4]

$$-\big(u_{ij} \times (u_i p_i)\big)\Delta\theta_i + u_{ij} \cdot \Delta t_i + \big(u_{ij} \times p'_i\big)\Delta\theta_{ij} - d_{ij} - \frac{1}{2}\, o_{ij} \leqslant -u_{ij} \cdot p'_i. \tag{3}$$

Lemma 2.2. *If rotations θ_i (recall that $u_i = (\cos\theta_i, \sin\theta_i)$) and translations t_i, $0 \leqslant i \leqslant k$, do not correspond to a local minimum of the non-linear optimization problem in Eq.* (2)*, then the linear optimization problem in Eq.* (3) *has a solution which diminishes the total overlap in a neighborhood of u_i and t_i: for sufficiently small $s > 0$, rotations $\theta_i + s\Delta\theta_i$ and translations $t_i + s\Delta t_i$ yield a diminished total overlap.*

Proof. If the layout is not at a local minimum for the overlap measure, then some motion (rotation and translation) will diminish the overlap. For a sufficiently small part of this motion, the higher order terms are negligible, and the linearized system approximates the actual system arbitrarily closely. Therefore, the linearized system has a solution which diminishes the overlap. For a sufficiently small part of the motion (small s) towards the linearized solution, the higher order terms are negligible, and therefore the original non-linear system has diminished overlap measure. □

2.2. Algorithm

The linearized optimization problem in Eq. (3) can be solved using linear programming. The overlap minimization algorithm solves the system for the current rotations and translations, determines the value of s which most diminishes the overlap, and replaces θ_i by $\theta_i + s\Delta\theta_i$ and t_i by $t_i + s\Delta t_i$, $1 \leqslant i \leqslant k$. This update step is repeated until the system converges to an equilibrium state of the non-linear optimization problem in Eq. (2). Lemma 2.2 implies that if the algorithm converges, it converges to such an equilibrium, and our experiments in Section 4 indicate that it does indeed converge very fast. All that remains to describe the overlap minimization algorithm for convex polygons is (1) how to determine u_{ij}, d_{ij} and a_{ij} before solving the linear program, and (2) how to determine the best value of s after solving the linear program.

For a pair of non-overlapping convex polygons, at least one separating line must be parallel to an edge of one of the polygons [11,21]. Hence, one can quickly solve the optimization in Eq. (1) by checking each unit vector u perpendicular to an edge of A or B. Again, it suffices to let a and b range over the *vertices* of A and B. For fixed rotations and translations, this technique determines the value of u_{ij} for $u_i P_i + t_i$ and $u_j P_j + t_j$, and hence d_{ij} for these rotated/translated polygons.

After solving the linear program, 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 $\Sigma(s)$ be the total overlap for interpolation value s. We know that $\Sigma'(0) < 0$, and we are seeking $s_{\min}$, the first minimum of $\Sigma(s)$ for $0 < s \leqslant 1$. If we have two sample values $s_l < s_r$ such that $\Sigma(s_l) < \Sigma(s_r)$, then we know $s_{\min} < s_r$, and we can discard sample intervals to the right of s_r. In our implementation, we alternate between sampling the midpoint of (1) the longest interval, and (2) the last (rightmost) interval between samples.

[4] The appearance of the cross product arises from the identity $a \times b = a^{\perp} \cdot b$.

3. Non-convex polygons

The overlap of two non-convex polygons can also be defined as the length of the shortest separating translation (also known as the *intersection-depth* [8]), and this is the overlap measure we use in our translational algorithm. Unfortunately, this measure is not conducive to mathematical programming in the rotational case. Instead, the rotational algorithm minimizes a measure based on an *inner convex cover* (ICC). Let P be a polygon and let $\text{icc}(P)$ be a set of (possibly overlapping) convex subsets of P whose union is P. Define

$$\text{overlap}(P_i, P_j) = \max_{C_i \in \text{icc}(P_i), C_j \in \text{icc}(P_j)} \text{overlap}(C_i, C_j), \tag{4}$$

where overlap of convex polygons is the intersection depth. This overlap measure has two intuitive properties: (1) it is zero only for non-overlapping polygons, and (2) for convex inputs, it is equal to the intersection depth.

Section 3.1 gives two ways of constructing inner convex covers of non-convex polygons, one geared towards overlap minimization and the other geared towards compaction. Section 3.2 gives an algorithm for overlap minimization, and Section 3.3 gives an algorithm for compaction.

3.1. Constructing inner convex covers

In our first experiments, we used an overlap measure based on a convex *decomposition* of each P_i, such as a triangulation. However, if $\text{icc}(P)$ is a partition of P, then $\text{overlap}(P_i, P_j)$ can have "unintuitive" local minima. Suppose L_i separates $\text{icc}(P_i)$ into two subsets, and suppose the same for L_j and $\text{icc}(P_j)$. In this case $\text{overlap}(u_i P_i + t_i, u_j P_j + t_j)$ has a constant-depth valley for all u_i, t_i, u_j, t_j which align $u_i L_i + t_i$ with $u_j L_j + t_j$. This valley traps the optimization in a highly overlapped state because the overlap measure gives no information on which way to move along the valley. Based on this experience, we developed highly inner convex covers whose elements overlap as much as possible: "fat" ICC and "potato" ICC. See Fig. 2.

To generate a "fat" ICC, our system generates a convex partition of P_i in a greedy fashion, and then greedily "fattens" each convex 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

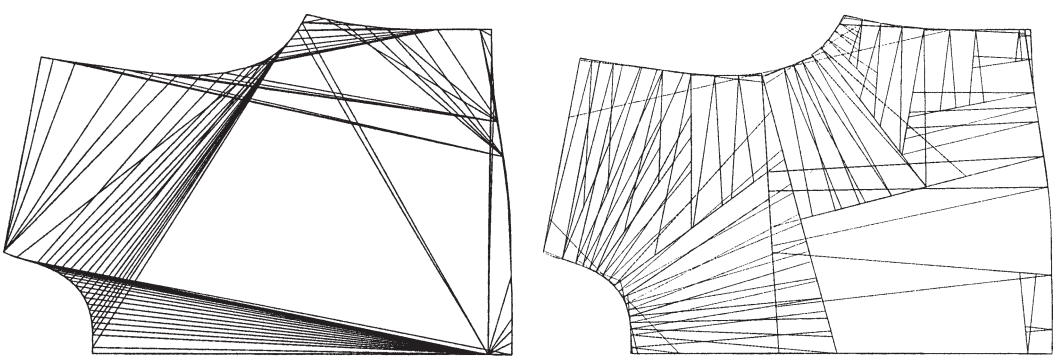

Fig. 2. Inner convex covers (ICC) of non-convex apparel polygon with 71 vertices. "Fat" ICC (left) has 39 polygons but covers each point 24 times on average. "Potato" ICC (right) has 101 polygons but only covers each point 8 times on average.

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. Fig. 2 shows the result for a typical non-convex apparel part. The "fat" ICC yields an excellent overlap measure. Unfortunately, it can have so many convex polygons covering some portions of P_i that the linear programs have many redundant constraints, and the overlap calculation takes a long time.

The "potato" ICC is an attempt to reduce the multiple coverage to a number roughly logarithmic in the number of vertices in P_i. The "potato" algorithm generates two overlapping polygons whose union is P_i. It does so by selecting a concave vertex and making two alternative cuts, each skewed 0.1 radians from the angle bisector. It greedily chooses the concave vertex such that the area of largest of the two polygons is minimized. It then recurses on the two polygons. This yields a binary tree of overlapping polygons which cover P_i. The leaves of the tree are convex polygons. For each polygon in the tree, the algorithm generates (an approximation to) the largest area convex subset. These convex subsets form the set $\text{icc}(P_i)$.

The largest convex subset problem is called the "potato cutting problem", and the fastest theoretical algorithm runs in time $O(n^7)$ [5]. Since this algorithm is very complicated and may be impractical to implement, we use the following approximate algorithm for the potato cutting problem. The largest area convex subset can be generated as follows: for each concave chain of the input polygon, construct a cut which is an internal tangent segment to that chain. The trick is to select the correct cuts! To generate an approximation, simply select an arbitrary cut for each concave chain and then visit each one in turn. When a cut is visited, sample nearby cutting angles to find one that maximizes the area. Repeatedly visit cuts until no more improvement is possible. We make no claims about the quality of the approximation, but it appears to work well in practice.

3.2. The overlap minimization algorithm

Given a choice of $\text{icc}(P_i)$ for P_i, $0 \leqslant i \leqslant k$, it is easy to generalize the algorithm of Section 2.2. Simply apply it to the set of convex polygons $\bigcup_{0\leqslant i\leqslant k} \text{icc}(P_i)$ with two modifications: (1) all the polygons $C_i \in \text{icc}(P_i)$ share the *same* translation/rotation variables t_i, u_i, and (2) each pair of polygons $C_i \in \text{icc}(P_i)$ and $C_j \in \text{icc}(P_j)$ use the *same* overlap variable o_{ij}. We can also drop any constraints between convex polygons belonging to the same ICC.

This rotational overlap minimization algorithm is very inefficient. It creates the constraints of Eq. (3) for each choice $\langle C_i, p_i, C_j, p_j\rangle$, for $0 \leqslant i < j \leqslant k$, for all $C_i \in \text{icc}(P_i)$, for all vertices $p_i \in C_i$, for all $C_j \in \text{icc}(P_j)$, and for all vertices $p_j \in C_j$. 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* algorithm. In it, the minimization step "guesses" a much smaller set S of constraint choices. The dynamic algorithm then takes a step, solving the linear program and calculating the best value of s, thus generating tentative new angles θ_i and translations t_i. If it determines that S is missing critical constraints for these new angles and positions, it adds the missing constraints to the set S and starts over at the previous angles and positions. The algorithm terminates when this update adds no new constraints, at which point it accepts the new values of θ_i and t_i.

To generate the initial set S of constraints, the dynamic algorithm considers pairs P_i and P_j which are overlapping at their current rotations and translations. For each such pair, it selects the pair C_i, C_j,

where $C_i \in \text{icc}(P_i)$ and $C_j \in \text{icc}(P_j)$, which determine the value of $\text{overlap}(u_iP_i + t_i, u_jP_j + t_j)$. It then determines the critical vertex $p_i \in C_i$: p_i has the maximum value of $u_{ij} \cdot (u_ip_i + t_i)$. It adds to S the constraint of Eq. (3) corresponding to the vertex p_i to S. Similarly, it adds the constraint corresponding to the critical vertex p_j of C_j. To update the set S of constraints, the algorithm similarly adds to S the critical constraints for the tentative new angles and positions.

3.3. Compaction

This section gives an algorithm that modifies the overlap minimization technique to do compaction: given a non-overlapping set of polygons in a container with fixed width and variable length, plan a motion for the polygon which minimizes the length. This is a local minimum. It also discusses how to generalize this algorithm to minimize the area of a rectangle with variable width and length.

Let P_0 be the complement $\overline{R}$ of the rectangular container, and introduce an additional "piston" rectangle P_{k+1} on top of all the polygons as shown on the left of Fig. 4. Add the additional constraints $o_{ij} = 0$, $0 \leqslant i, j \leqslant k+1$, $i \neq j$, to the linear program of Eq. (3). Set the objective to minimize the y-coordinate y_{k+1} of the "piston". Using a standard mathematical programming technique, we also add terms equivalent to $\varepsilon \sum_{1 \leqslant i \leqslant k} (|\Delta\theta_i| + |\Delta x_i| + |\Delta y_i|)$, where $\Delta t_i = (\Delta x_i, \Delta y_i)$. We use $\varepsilon = 0.001$. These additional terms serve to stabilize the motion and, surprisingly, even speed up the solution to the linear program.

For inputs with many polygons, even the dynamic algorithm of Section 3.2 can take many iterations to settle to a stable set of constraints. We found the following "reckless" algorithm to work best in practice. *Always accept* the tentative new angles and positions, even if S is missing critical constraints. Add in these missing constraints. Push the previous values of u_i and t_i, $0 \leqslant i < j \leqslant k+1$, on a stack. If the linear program is infeasible, pop the stack to return to the previous angles and positions, but do not remove constraints from S. In this way, the algorithm acts recklessly, not waiting for a stable set of constraints but learning from its mistakes. The original angles and positions at the bottom of the stack correspond to a non-overlapping configuration, and therefore its linear program is always feasible. Thus the compaction algorithm never pops an empty stack.

To minimize the area of the container rather than its length, use the same strategy with an *area minimizing* linear program. In this case, P_0 is the complement of the first quadrant, $P_0 = \{(x, y) \mid x \leqslant 0 \text{ or } y \leqslant 0\}$, and P_{k+1} is the complement of the third quadrant, $P_{k+1} = \{(x, y) \mid x \geqslant 0 \text{ or } y \geqslant 0\}$. We remove any penalty for overlap between P_0 and P_{k+1}: remove constraints for $(i, j) = (0, k+1)$. 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_0t_{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 [19], 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. The new objective is therefore to minimize $y_{k+1}\Delta x_{k+1} + x_{k+1}\Delta y_{k+1}$ (plus ε times the absolute value terms).

4. 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 future, we plan to 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. In fact, they are often more complex than glass or sheet metal parts because humans have more curves than windows or washing machines. Overlap minimization experiments were run on a DEC Alpha 3000/700 (Digital Equipment Corporation) using the simplex method of CPLEX 3.0 (CPLEX Optimization, Inc.), the "fat" ICC, and dynamic constraint calculation. Compaction experiments were run on an SGI PowerChallenge L (Silicon Graphics, Inc.) using the dual simplex option of CPLEX 4.0, the "potato" ICC, and reckless constraint calculation.

4.1. Overlap minimization

Fig. 3 shows two examples of overlap minimization for the polygon in Fig. 2. In the first example, the container is the bounding box for the polygon before a large rotation and translation is applied. The polygon minimized its overlap with the exterior of the container in 3 iterations. Running times on the DEC Alpha: 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. The second example is overlap minimization for two copies of the same polygon in a container 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; LP, 0.93 seconds. Iteration 1 required 2 auxiliary LP calls, and iteration 2 required 4 (included in 0.93 seconds).

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

Fig. 3. Overlap minimization for one polygon and for two polygons.

4.2. Compaction

We focused most of our experiments on the industrially significant problem of compaction. All inputs are true, human-generated layouts. Translational compaction [16,18] has been applied to each, and therefore any further benefit is the result of rotation. Iteration ceased when the objective diminished by less that one part in a million. At each step $|\Delta x_i|$ and $|\Delta y_i|$ were limited to 0.25 inch and $|\Delta\theta_i|$ was limited to 0.1 radians. These values were chosen after several trials to determine bounds that would keep the number of infeasible steps to a minimum. (A development version of this system would have to dynamically adjust these bounds automatically.) Rotational compaction was applied to layouts with 23, 54, 108, 180 and 360 polygons. Fig. 4 shows the input and output for the first experiment. Table 1 summarizes the results, and an analysis appears in Table 2.

The most remarkable property of the rotational compaction algorithm is that the number of iterations is small and does not appear to grow significantly with the size of the problem. The input with 23 polygons and 1,120 vertices requires 22 iterations. The input with 360 polygons and 21,967 vertices requires 31 iterations. This is a testimony to the power of linear programming. It can optimize everywhere in the layout simultaneously, and unlike physically based methods, it can rapidly transmit

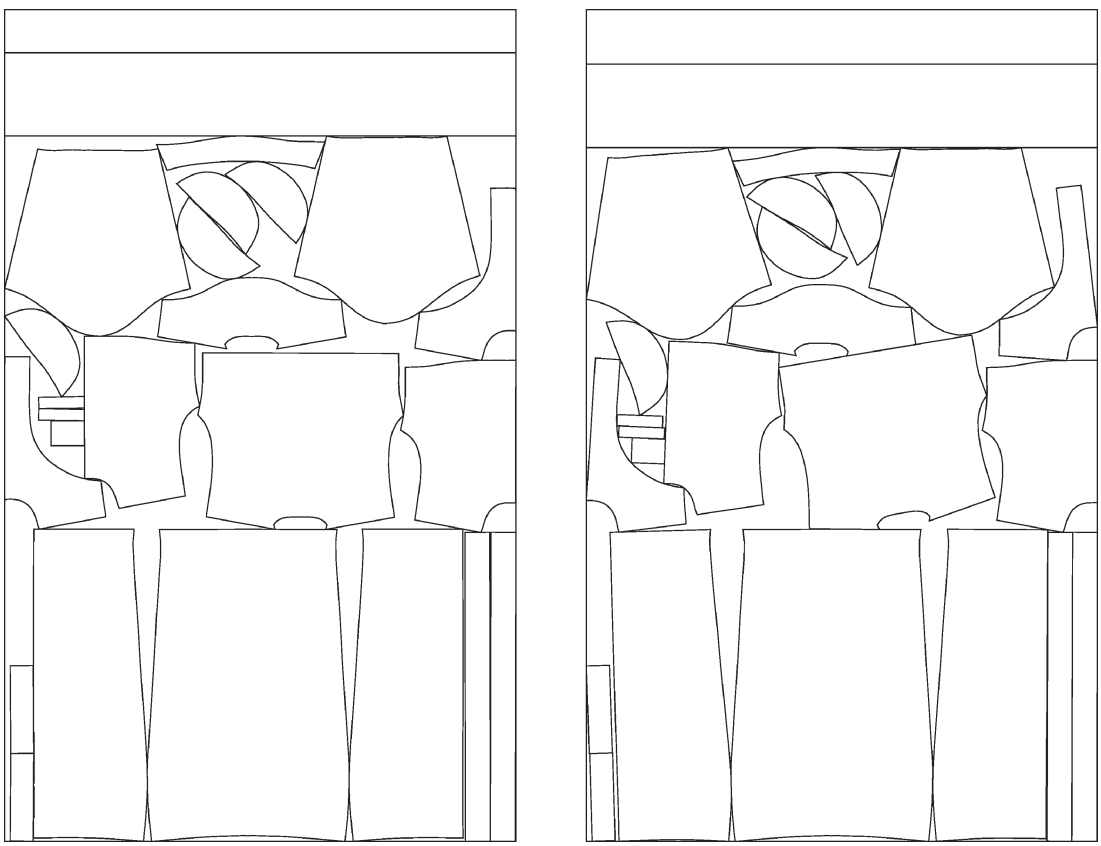

Fig. 4. Compaction experiment 1: 23 polygons.

Table 1
Rotation compaction on five industrial apparel layouts (after translational compaction)

Experiment	1	2	3	4	5
#poly	23	54	108	180	360
Length (inch)	85.34	123.04	269.04	220.46	146.29
Translational	84.81	121.34	268.82	218.86	143.82
% change	−0.62	−1.38	−0.08	−0.73	−1.69
Rotational	83.41	119.98	268.49	217.91	141.63
% change	−1.65	−1.12	−0.12	−0.43	−1.52
#steps	22	22	25	23	31
#LP	22	26	27	33	33
Time (sec.)	41	210	128	1752	15890
LP time	6	41	56	1358	12930
Time/LP	0.27	1.6	2.1	41	392
#vert	1120	4163	2040	9183	21967
#verts/poly	49	77	19	51	61
#pairs	105	281	376	1108	3012
#cols	540	1054	1789	3817	9203
#rows	558	1277	1673	5593	16581
#non-zeros	3900	9334	11276	42463	128428

Table 2
Analysis of rotational compaction experiments

Experiment	1	2	3	4	5
#rows/col	1.0	1.2	0.9	1.5	1.8
#non-zeros/row	7.0	7.3	6.7	7.6	7.7
LP time/rows^2	0.86	0.98	0.75	1.3	1.4
#rows/vert	0.5	0.3	0.8	0.6	0.8
#rows/poly	24	24	15	31	46

information from one end of the layout to the other. The results also confirm Li's finding (Section 1.2) that allowing rotations can save material even after translational compaction. [5]

The linear programs are nearly square: as Table 2 indicates, the number of columns per row rises slowly. The number of non-zero coefficients per row also rises slowly. The conventional wisdom about the simplex method is that it should have a running time quadratic in the number of rows in these

[5] Tilt limits make reaching a local minimum *easier*. Since we are measuring running time, we did not impose them. With tilt limits, our algorithm would run faster than shown here, and it would save more material than Li's algorithm.

circumstances. Indeed, the ratio of linear programming time (per program) to number of rows squared is nearly constant, rising very slowly from 0.86 to 1.4. The number of rows is a little more difficult to predict. It appears to be roughly proportional to the number of vertices, or alternatively the number of polygons, in the input. This is what one would expect since each row corresponds to a polygon-polygon constraint and since in a planar packing, each polygon can be expected to have about six neighbors. As the input size grows, the linear programming time dominates the running time. Overall, these experiments show that the running time grows somewhat faster that quadratically in the input complexity.

5. Conclusions

The rotational overlap minimization algorithm requires remarkably few iterations to locate a local minimum. Its running time is quite reasonable. It clearly will be useful for future applications in global algorithms for containment and minimum enclosure, which we intend to address in the near future. The experiments with the rotational compaction algorithm show that it will be both useful and practical for industrial applications. It saves at least as much material over translational compaction as Li's earlier rotational algorithm (Section 1.2). Its running time is clearly within the practical realm, especially given the fast pace in hardware development. (For comparison, translational compaction of the 360-polygon layout requires only about five minutes.)

The analysis shows that the running time of rotational compaction is roughly quadratic in the input complexity. This suggests possible strategies for faster compaction. If the layout is divided into k equal regions, then the running time for compaction in each would be less the $1/k^2$ times the running time for the entire layout. The total running time would be less than $1/k$ times the running time for the entire layout. Polygons in one region would have to be permitted to intrude into neighboring regions and overlap the polygons by a small amount. The overlap would indicate the location of the greatest "pressure". The question would be how many iterations would be required to bring the layout to a final, non-overlapping, compacted layout. If the number of iterations is not k times as large, then this method would be of benefit. Of course, this approach could also be used for compaction in *parallel* on several processors or computers.

Finally, as stated in Section 1.2 many current and future heuristics for nesting/layout might benefit from the addition of overlap minimization and compaction. We would be happy to assist in any way we can such an application of our algorithms.

References

[1] R.M.S. Abd El-Aal, A new technique for nesting irregular shapes based on rectangular modules, Current Adv. Mech. Design Production 6 (1996) 533–540.

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

[3] F. Avnaim, 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, Springer, Berlin, 1988, pp. 322–333.

[4] C. Bounsaythip, S. Maouche, Irregular shape nesting and placing with evolutionary approach, in: 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation (Cat. No. 97CH36088-5), IEEE, New York, 12–15 October 1997, p. 5.
[5] J.S. Chang, C.K. Yap, A polynomial solution for the potato-peeling problem, Discrete Comput. Geom. 1 (1986) 155–182.
[6] B. Chazelle, The polygon containment problem, in: Preparata (Ed.), Advances in Computing Research, Vol. 1: Computational Geometry, JAI Press, Greenwich, CT, 1983, pp. 1–33.
[7] K. Daniels, Containment algorithms for nonconvex polygons with applications to layout, Ph.D. Thesis, Harvard University, Cambridge, MA, 1995.
[8] D. Dobkin, J. Hershberger, D. Kirkpatrick, S. Suri, Computing the intersection-depth of polyhedra, Algorithmica 9 (1993) 518–533.
[9] K.A. Dowsland, W.B. Dowsland, Solution approaches to irregular nesting problems, European J. Oper. Res. 84 (3) (1995) 506–521.
[10] R.B. Grinde, T.M. Cavalier, A new algorithm for the two-polygon containment problem, Comput. Oper. Res., to appear, 1996.
[11] L. Guibas, L. Ramshaw, J. Stolfi, A kinetic framework for computational geometry, in: Proceedings of the 24th IEEE Symposium on Foundations of Computer Science, 1983, pp. 100–111.
[12] G.C. Han, S.J. Na, Two-stage approach for nesting in two-dimensional cutting problems using neural network and simulated annealing, J. Engrg. Manufacture 210 (B6) (1996) 509–519.
[13] R. Heckmann, T. Lengauer, Computing upper and lower bounds on textile nesting problems, Lecture Notes in Computer Science 1136, 1996, pp. 392–405.
[14] P. Jain, P. Fenyes, R. Richter, Optimal blank nesting using simulated annealing, J. Mech. Design 114 (1) (1992) 160–165.
[15] H.J. Lamousin, W.N. Waggenspack, G.T. Dobson, Nesting of complex 2-D parts within irregular boundaries, J. Mech. Design 118 (4) (1996) 615.
[16] Z. Li, Compaction algorithms for nonconvex polygons and their applications, Ph.D. Thesis, Harvard University, Cambridge, MA, 1994.
[17] Z. Li, V. Milenkovic, The complexity of compaction problem, in: Proc. 5th Canad. Conf. Comput. Geom., Waterloo, Canada, 1993, pp. 7–11.
[18] Z. Li, V. Milenkovic, Compaction and separation algorithms for nonconvex polygons and their applications, European J. Oper. Res. 84 (1995) 539–561.
[19] V. Milenkovic, Translation polygon containment and minimal enclosure using linear programming based restriction, in: Proc. 28th Annu. ACM Sympos. Theory Comput. (STOC 96), 1996, pp. 109–118.
[20] V. Milenkovic, Position-based physics: simulating the motion of many highly interacting spheres and polyhedra, in: Computer Graphics, Proc. SIGGRAPH '96, 1996, pp. 129–136.
[21] V.J. Milenkovic, Multiple translation containment, Part II: Exact algorithms, Algorithmica 19 (1997) 183–218.
[22] V. Milenkovic, K. Daniels, Translation 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.
[23] V. Milenkovic, K.M. Daniels, Translational polygon containment and minimal enclosure using mathematical programming, Internat. Trans. Oper. Res., 1997, to appear.
[24] Y.K.D.V. Prasad, S. Somasundaram, K.P. Rao, A sliding algorithm for optimal nesting of arbitrarily shaped sheet metal blanks, Internat. J. Production Res. 33 (6) (1995) 1505–1520.
[25] Yu.G. Stoyan, M.V. Novozhilova, A.V. Kartashov, Mathematical model and method of searching for a local extremum for the non-convex oriented polygons allocation problem, European J. Oper. Res. 92 (1996) 193–210.