ELSEVIER

European Journal of Operational Research 84 (1995) 539–561

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Compaction and separation algorithms for non-convex polygons and their applications *

Zhenyu Li [1], Victor Milenkovic *

Harvard University, Center for Research in Computing Technology, Aiken Computational Laboratory, 33 Oxford Street Cambridge, MA 02138, USA

Abstract

Given a two-dimensional, non-overlapping layout of convex and non-convex polygons, compaction can be thought of as simulating the motion of the polygons resulting from applied 'forces'. By moving many polygons simultaneously, compaction can improve the material utilization of even tightly packed layouts. Compaction is hard: finding the tightest layout that a valid motion can reach is PSPACE-hard, and even compacting to a local optimum can require an exponential number of moves. Our first compaction algorithm uses a new velocity-based optimization model based on existing physical simulation approaches. The performance of this algorithm illustrates that these approaches cannot quickly compact tightly packed layouts. We then present a new position-based optimization model. This model represents the forces as a linear objective function, and it permits direct calculation, via linear programming, of new non-overlapping polygon positions at a local minimum of the objective. The new model yields a translational compaction algorithm that runs two orders of magnitude faster than physical simulation methods. We also consider the problem of separating overlapping polygons using the least amount of motion, prove optimal separation to be NP-complete, and show that our position-based model also yields an efficient algorithm for finding a locally optimal separation. The compaction algorithm has improved cloth utilization of human-generated layouts of trousers and other garments. Given a database of human-generated markers and with an efficient technique for matching, the separation algorithm can automatically generate layouts that approach human performance.

Keywords: Packing; Compaction; Polygon nesting; Irregular shapes; Minkowski sums; Physically based modeling

1. Introduction

1.1. Definitions

In this paper, a two-dimensional *layout* is defined to be the placement of a set of polygonal pieces on a rectangular sheet of material. The layout acts as a plan for cutting the pieces out of the material, and therefore the pieces must not overlap each other or cross the boundary of the

* A previous version of this paper was presented in [19].

* Corresponding author. Current address: University of Miami, Department of Mathematics and Computer Science, P.O. Box 249085, Coral Gables, FL 33124, USA. Research funded by the Textile/Clothing Technology Corporation from funds awarded to them by the Alfred P. Sloan Foundation and by NSF grants CCR-91-157993 and CCR-90-09272.

[1] Current address: GTE Laboratories Incorporated, 40 Sylvan Road, Waltham, MA02254. Research funded by the Textile/Clothing Technology Corporation from funds awarded to them by the Alfred P. Sloan Foundation.

SSDI 0377-2217(95)00021-6

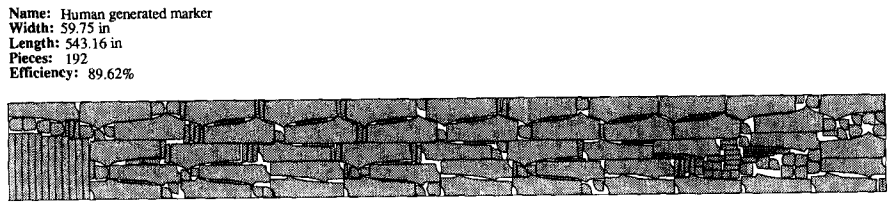

Fig. 1. A human-generated trouser marker.

material. A 'faulty' or overlapping layout violates this rule; a non-overlapping layout does not. The efficiency (utilization) of a non-overlapping layout is the ratio of the area covered by the pieces to the total area of the rectangular sheet. In the apparel industry, the layout sheet has fixed width (corresponding to the width of a bolt of cloth) and arbitrary length, and a non-overlapping layout is called a *marker*. Fig. 1 shows a human-generated trouser marker with efficiency of 89.62%. Optimal marker making, finding a placement of a given set of pieces with highest efficiency, is a hard problem. According to Dyckhoff's typology [11] of cutting and packing problems, marker making can be classified as a 2/V/O/M type problem.

This paper considers the simpler problem of planning motions of the pieces in an already existing layout to improve the layout in some fashion: increase efficiency by shortening the length, open space for new pieces without increasing the length, or eliminate overlap among pieces in an overlapping layout. We use the term *compaction* for any motion planning task that stays entirely within the set of feasible (non-overlapping) configurations, such as the first two tasks. We use the term *separation* for a task that moves the configuration from an overlapping to a non-overlapping configuration. Ultimately, we introduce a compaction/separation optimization model that encompasses both types of motions and other interesting combinations.

1.2. Applications

Properly applied, compaction and separation can do a variety of marker layout tasks. Leftward or downward compaction reduces the area and increases the efficiency of a layout. This process involves moving the right boundary leftward to shorten the length of the layout in the x-direction, or moving the top boundary down to shorten the length in the y-direction, or both. Fig. 2 shows the leftward compaction of the marker in Fig. 1, which improves the efficiency of the marker by 0.94%. Another application is opening gaps (free space between pieces) to fit new pieces into a layout of fixed area. One repeatedly finds gaps [9], and in case a gap is not large enough to hold a new piece, one opens the gap by pushing away the surrounding pieces. Of course, compaction is only a tool: it does not say which gap to choose or even what forces to apply. Fig. 3 shows compaction opening a gap by applying the forces shown by the arrows. After this compaction, two new polygons (which were not part of the motion

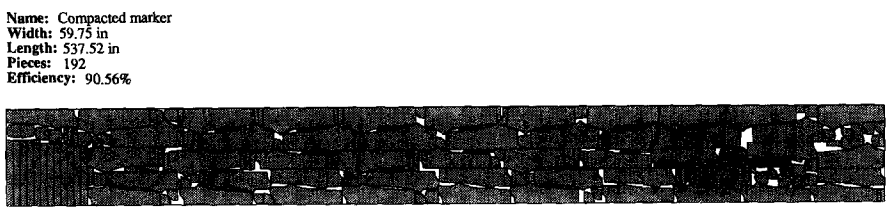

Fig. 2. The human-generated marker after compaction.

planning but are shown for illustration purposes) can be placed without overlap. If one chooses to place the two polygons in the overlapping positions without first opening the gap, then *separation* can achieve the same result. Separation does not require specific forces as inputs, but the user has to choose the gap. Although we call them different functions, our algorithm does both separation and compaction: it will first eliminate overlaps and then optimize motion in the direction of all specified forces. Applications of compaction/separation are very important when the pieces in the layout are to be cut from expensive materials, as in aircraft, automobile and apparel manufacturing.

This research grew from our current project [25] to generate trouser markers automatically with an efficiency competitive with those generated manually by experienced humans. Humans with many years of experience can reliably generate trouser markers whose efficiency is in the range 89–91% and probably within 1% of optimal. Our algorithm for panel (large piece) placement [26] can exceed human performance at least 60% of the time, and the average efficiency after automatic panel placement and compaction is slightly better than the human average. We plan to use the compaction techniques described here to improve the efficiency of our machine generated markers and to open gaps during trim (small piece) placement. Currently, compaction is being put to use in industry to improve human-generated layouts.

A marker acts as a cutting plan for a human or robotic cloth cutter. Usually, apparel manufacturers cut many layers of cloth simultaneously, generating many copies of each garment. Because of the high labor cost of creating an efficient marker, they also use the same marker as a cutting plan as many times as possible. It follows that even a small percentage increase in efficiency for a marker results in a large savings in material. Trouser manufacturers cut sixty layers of cloth simultaneously each time they use a marker, and a 1% increase in efficiency represents a savings of about $25 per cutting. A representative of a large trouser manufacturer estimates that a 0.1% average improvement in efficiency would save his company about two million dollars per year.

Experienced human marker makers can generate high efficiency markers. However, our experiments show that our compaction algorithm can improve the efficiency of almost all production quality markers, trousers or other garments. Humans cannot achieve this improvement by hand because they have already packed the production quality marker very tightly. Every piece is touching its surrounding pieces, and so there is no room to move any one piece. Increasing the marker efficiency requires a coordinated motion of all the pieces, which a human cannot achieve.

Another difficult task is to move pieces to open a gap somewhere in a marker without requiring a larger sheet of material. Human marker makers need to open gaps to place trim pieces. Even experienced marker makers find the task of opening gaps in tightly packed markers time-consuming and awkward. Since they can only move one piece at a time, it is unlikely that they are opening gaps the maximum amount. A com-

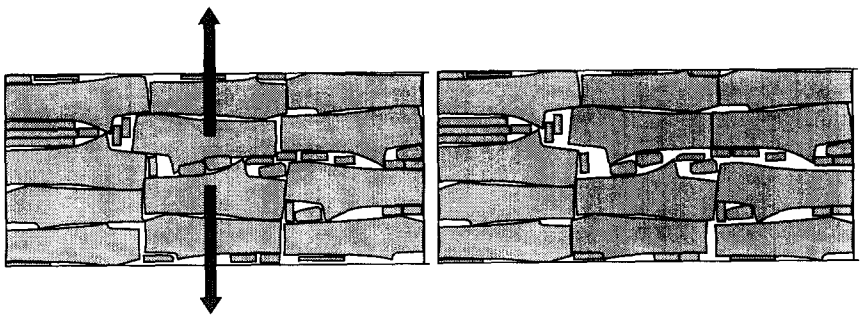

Fig. 3. Opening a gap in a marker.

paction algorithm that can efficiently open a gap can be a valuable tool for human marker makers.

These applications of compaction tasks all address tightening of a layout. In marker making, one encounters the need for *separation*, and there are two main applications of this operation. The first application involves removing overlaps from markers. Some markers may have overlapping pieces, and we want to eliminate the overlaps by moving pieces away from each other. If necessary, we would even increase the length of the material to allow space for resolving overlaps. The second application involves distributing available free space uniformly among the pieces. Frequently, the length of a marker is determined by a critical 'path' of pieces connecting the left boundary of the material to the right boundary. It is difficult or sometimes impossible to reduce the length of the critical path. However, pieces not on the path can have large free spaces around them. An appropriate motion of these pieces can equally distribute these spaces and thus increase the minimum distance between adjacent non-critical pieces in the marker. We refer to such a motion as *floating*. Close contact among pieces can lead to errors in cutting, and thus floating makes cutting easier and less prone to errors. Fig. 4 shows an example of floating.

Separation for overlap resolution in turn has an application in *database-driven* automated marker making. The underlying idea is to collect high quality markers generated by experienced human marker makers into a database. When a new marker making order comes in, we look at the database for a similar marker. Then we match the pieces and do a piecewise substitution of the corresponding pieces. The substitution process will inevitably introduce overlaps. By applying separation and leftward compaction, we can generate a marker of sufficient quality automatically.

1.3. Related work

As far as we know, no efficient algorithm has been previously published for compaction or separation of many non-convex polygons (see [39] for a survey of current results). In industry, several CAD firms have tried 'one piece at a time' methods, which are not successful in practice.

Previous research efforts in compaction are largely concentrated in the field of VLSI design [4,18,23,29,36,40]. This research focuses on compacting a layout consisting of rectangles and sometimes variable-length rectilinear (orthogonal) wires. We found these techniques not readily extendible to non-rectangular polygons or not applicable in our case in which the layouts are already tightly packed.

Physically based simulation methods provide another approach to compaction. They simulate the pieces as rigid bodies and apply a set of forces to move the bodies in the desired directions until they reach a local minimum of some potential function. These simulation methods fall into two types: *spring* model and *contact force* model. The difficulty we have found with physi-

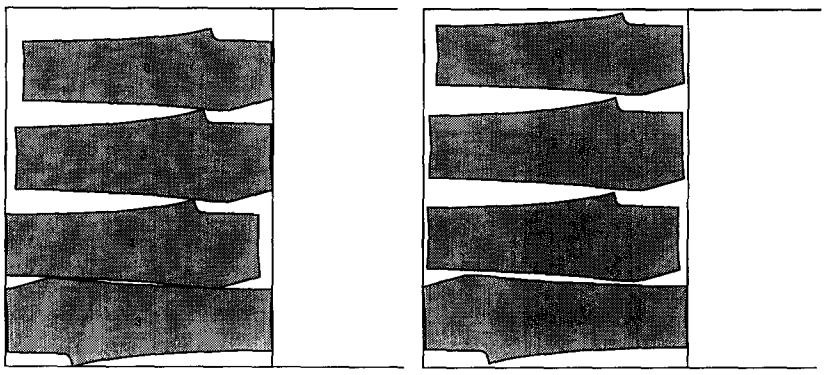

Fig. 4. An example of floating.

cally based simulation methods is that these algorithms run very slowly.

Spring model methods (also called *penalty methods*) [28,31] allow the pieces to overlap. For each pair of overlapping pieces, there is a repulsive force proportional to the amount of overlap. From these forces, the motion of the pieces is computed by numerical integration. Standard techniques carry out this integration by cutting time into steps. Small steps are required to ensure accuracy and numerical stability of the integration; however, the smaller the step, the greater the computational cost. It is generally very difficult to choose the correct step size, especially when many different forces are involved. Layout compaction involves hundreds of frequently changing contacts, and consequently, the spring model is not suitable for compaction.

Contact force model methods (also called *analytical methods*) [5,6] have been recently studied in computer graphics. This method introduces contact forces at contact points to prevent overlap. A system of differential equations is solved to find a set of consistent contact forces. The bodies then move with constant velocities according to the contact forces. A collision detection algorithm finds the time-interval until the next collision time. The simulation proceeds from time-interval to time-interval until the local minimum is reached. Contact force methods have the advantage that the time-intervals are much larger and more easily computed than the time-steps of the spring model methods. However, there still are difficulties. Solving the system of differential equations to figure out the contact forces is time-consuming for problems of non-trivial size. Moreover, finding consistent contact forces is itself NP-complete if there are vertex-vertex contacts [5,30]. Finally, we find that in the case of many contacts the local minimum is only reached after many time-intervals because the pieces 'rattle' against each other.

Stoya [37,38] uses an optimization approach similar to our first velocity based optimization model (see next section). For polygons with many vertices (more than twenty), he simplifies the polygons and the configuration space to overcome the costs inherent in the velocity-based approach. Thus, his technique only generates an approximate compaction.

1.4. Results

The main contributions of this paper are two optimization models which yield algorithms for performing various compaction and/or separation tasks: a *velocity-based* model and a *position-based* model.

The velocity-based model allows us to remove the concept of mass and force and directly compute velocities. This model yields a new, more efficient time-interval based simulation technique for translational compaction. We had expected that direct computation of velocities would allow our algorithm to solve the compaction problem in a reasonable amount of time. Our new simulation techniques run at least an order of magnitude faster than previous methods. Unfortunately, the algorithm still took hours to run on typical markers on a 28MIPS workstation because of the number of times it has to recompute velocities. We view this as an important negative result that demonstrates the inherent limitation of the physically based approach.

Our second, position-based model yields a compaction algorithm that directly computes a new set of positions for the polygons without using velocities, time-steps, time-intervals or any sort of simulation. Instead, this new model contains artificial constraints that select a convex feasible subset of the solution space and thus make a direct calculation possible via linear programming. The artificial constraints are based on the configuration space concept from the field of robotics plus a new *locality heuristic*. Because of the extra constraints, the algorithm does not immediately reach a local minimum of the original compaction problem, but a small number (less than five in practice) of repetitions suffices to reach the local minimum. Using this model, our compaction algorithm runs in almost real time in practice and solves industrially significant problems.

Our new position-based model also yields a separation algorithm that eliminates overlaps in a marker. Originally, compaction was our only goal,

but we noticed that the compaction algorithm also accomplished separation. This occurred because the slack variables for each non-overlapping constraint in our model exactly correspond to the amount of overlap between pairs of polygons. We realized that we could make these variables explicit in the constraints and add them to the original objective of our model. The resulting position-based model can eliminate overlaps, apply forces, and even float pieces away from each other (negative overlap) in any desired combination. The separation algorithm can be applied in database-driven automated marker generation. A CAD company in the textile industry is now vigorously pursuing the database-driven automated marker making idea.

1.5. Organization of the paper

The following section formalizes the compaction problem and discusses its complexity. Section 3 describes a physically based simulation method for compaction. This technique solves a velocity-based optimization model to generate a set of velocities for the set of polygons and then simulates motion until a new contact occurs. We discuss its long running time for tightly packed markers and why that is an inherent drawback of physically-based methods. Section 4 gives background knowledge about configuration spaces and Minkowski sums, which play a central role of in the position-based model. Section 5 describes this new model and gives an algorithm based on it that directly calculates new positions for the polygons. As a result, this algorithm does not use simulation nor does it have time as an explicit parameter. Section 6 defines the problem of separating overlapping polygons, studies its complexity, and shows how the position-based model yields a separation algorithm that finds a locally optimal solution to this problem. Section 7 demonstrates that by combining the separation algorithm and a database of human-generated markers, we can have an automated marker making system that very quickly generates markers close to human performance. Finally, Section 8 gives empirical evidence for the efficiency of the compaction algorithm in marker making. It also describes how it fits into our current project for automatically generating trouser markers.

2. Compaction: Definition and analysis

To study its complexity, we find it convenient to define *compaction* as a two-dimensional motion planning problem. A given set of polygons in a rectangular container can move simultaneously. In addition, the right boundary of the container can move, allowing the rectangle to change length but not height. During a legal motion, the polygons cannot overlap each other or the boundary of the container. The goal of the motion is to minimize the area of the container by moving the right boundary inward as much as possible. A translational compaction problem is a compaction problem in which the polygons are not allowed to rotate. This paper covers only translational compaction.

Section 2.1 provides an analysis of the complexity of compaction and contrasts its complexity with that of the original *layout* problem. One can think of *layout* as compaction without the non-overlap restriction: polygons can move 'through' each other or 'jump over' each other without restriction, as long as the final placement is non-overlapping. As it turns out, compaction is PSPACE-complete, and layout is 'merely' NP-complete. We can define a *local optimum* to the compaction problem to be a set of positions such that any legal motion requires the right boundary to move rightward. Even finding a locally optimal compaction may require exponential time. Notwithstanding these theoretical results, Section 2.2 gives reasons why compaction is easier than layout in practice.

2.1. Complexity of compaction and layout

Theorem 2.1. *Two-dimensional translational compaction is NP-hard even for rectangles.*

Proof. We reduce the NP-hard PARTITION problem to compaction. The PARTITION problem is defined as follows: given a set S of integers, decide if S can be partitioned into two

subsets S_1 and S_2 such that the sum of the elements of S_1 equals the sum of the elements of S_2. We reduce PARTITION to compaction by the construction shown in Fig. 5. For an instance $\{a_1, a_2, \ldots, a_n\}$ of PARTITION, we build rectangles of height one and width a_i, $i = 1, \ldots, n$. We put the rectangles into a container of height two. PARTITION has a solution if and only if the blocks can be compacted to the length $\frac{1}{2}\sum a_i$. □

Notice that this proof does not consider the motion because it is trivial to generate it from the starting and finishing configurations. Hence, this same proof shows the following.

Corollary 2.2. *Layout is NP-hard.*

It turns out that compaction is harder than layout because of the non-overlapping constraints on the pieces during the motion. Moving from one configuration to another might involve complicated motions of many pieces. A closely related coordinated motion planning problem is the *warehouseman* problem, which is the problem of deciding whether a set of rectangular objects within a rectangular region can reach a goal configuration from an initial configuration through translations. Hopcroft et al. showed that the warehouseman problem is PSPACE-hard [16].

It may seem that the warehouseman problem is harder than the compaction problem because the compaction problem need not force a particular final configuration, just one which optimizes the objective. In other words, there might be many configurations that satisfy the goal of compaction, which is to shorten the container's length. However, the authors [21] have shown that a restricted version of the warehouseman problem can be reduced to the compaction problem, and this restricted version is still PSPACE-hard.

Theorem 2.3 [21]. *Compaction is PSPACE-hard, even for a collection of rectangles, when the right boundary of the container is not allowed to move to the right.*

The PSPACE-hardness of the problem shows that moving from one configuration to another by translation alone might require an exponential number of steps. The authors have constructed layouts that require an exponential number of moves to compact, even to a local optimum [21]. Please note that this result does *not* imply that PSPACE ≠ P because there may be some way of proving (in polynomial time) that the motion *exists* without actually constructing it.

Theorem 2.4. *Translational compaction is in PSPACE.*

Proof. Given a layout with n polygons, the configuration space for non-overlapping translational motion is $2n$-dimensional. For each pair of polygons P and Q, the constraint that vertex A of P lies on the 'correct' side of edge BC of Q represents a half-space in the configuration space. The constraint that A lies on BC is a hyperplane. There are only a polynomial number of such vertex-edge hyperplanes. Cutting the configuration space of non-overlapping positions with these hyperplanes partitions the space into an exponential number of convex regions. The points in any such convex region R all lie on the same side of all the hyperplanes. Each region can be represented by a set S of choices of side for each of the hyperplanes. If any point p within a region R represents a non-overlapping configuration, then the entire region does. To see why, let q be any other point in R. Line segment $\overline{pq}$ lies in R since R is convex. Furthermore, p and q lie on the same side of every hyperplane. Therefore, a mo-

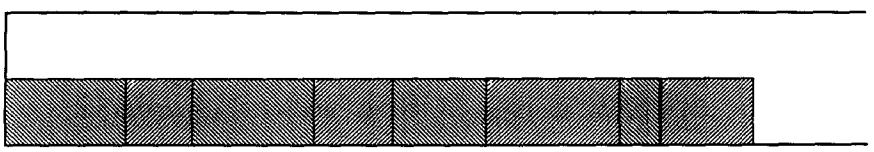

Fig. 5. Reduction of PARTITION to compaction.

tion from p to q crosses no hyperplane, and this motion causes no overlaps.

For any given set S of choices of side, linear programming can determine if S corresponds to a nonempty region R, and it can calculate the point $p \in R$ that optimizes the compaction objective: right boundary of layout is farthest to the left. A polynomial time geometric algorithm can check that p corresponds to a non-overlapping layout. By changing a single choice of side, one can generate a neighbor of R. All neighboring regions can be computed in polynomial time. The set of valid regions forms a graph with less than 2^H nodes and less than $H \cdot 2^H$ links, where H is the number of hyperplanes. Given any region R, Savitch's algorithm for graph connectivity can determine if there is a path from the starting state to R using $O(H^2)$ space (and exponential time) [33]. Thus using only polynomial space, one can run through all sets of choices S and determine the one which is reachable and minimizes the objective. □

Corollary 2.5. *Layout is in NP.*

Proof. To find the layout with a given efficiency E, nondeterministically 'guess' the set S corresponding to that layout and verify it. To verify, use linear programming to generate the region R corresponding to S and the optimal configuration $p \in R$. Use standard geometric algorithms to check for overlaps. □

Remark. In principle, Canny's algorithm [7] can solve the leftward compaction problem exactly in time exponential in the number of degrees of freedom. However, the number of degrees of freedom here is linear in the input size. Therefore, his algorithm requires exponential time for our application.

2.2. Complexity of compaction in practice

Compaction is NP-hard but not in NP because it can take an exponential number of moves to reach a final state. In fact, compaction is PSPACE-hard, and there exist layouts that require an exponential number of moves to compact, even to a local optimum. Compaction is very hard because the polygons must move without overlapping each other, and this restriction makes the search space very complicated.

These hardness results run counter to intuition. One would think it easier to 'improve' a layout than to generate one 'from scratch'. The reason that theory does not agree with intuition is that theory does not distinguish between different types of failure: a nonoptimal placement is just as bad as an infeasible (overlapping) placement. In practice, an infeasible layout is useless, but a nonoptimal one might often be acceptable. In all likelihood, almost every layout used in industry is nonoptimal. Compaction must deal with a more complex search space than layout, but every point in this search space is feasible. A compaction algorithm may fail to find an optimal compaction; in fact, unless PSPACE = P, it cannot always find the globally optimal compaction in polynomial time. Even so, a local optimum (or any sort of improvement) is still a very useful result.

The theoretical hardness results have two practical implications. First, the best one can hope for is a locally optimal compaction, which is what our algorithm finds. Second, even if a compaction algorithm only finds a local optimum, it still can have an exponential running time. We assume here that the algorithm explicitly searches the space of non-overlapping motions. We have not shown that finding a local optimum is PSPACE-hard in general, but we expect this to be true. To prove this, one would have to construct a general-purpose two-dimensional mechanical computer whose sole source of power is the inward motion of one wall of its container. We conjecture that such a construction is possible.

3. Compaction using a velocity-based optimization model

In this section, we give a compaction algorithm based on a simulation of translational motion of rigid bodies. This algorithm computes the velocities of the bodies using a velocity-based optimization model. By allowing us to ignore the dynamic

properties of rigid bodies and to concentrate on kinematics, the optimization model yields a compaction algorithm which is simpler than current contact force model algorithms (see Section 1.3).

Under the velocity-based model, a polygon in the layout is a rigid polygonal body governed by two geometric properties: 1) every point on the body has the same velocity, and 2) the body is not penetrable by other bodies. The following optimization model, in which implicit 'pseudo'-forces determine the motions of the polygons, replaces the usual Newtonian physics formulation.

We start by assigning a velocity v_i, where $|v_{ix}|, |v_{iy}| \leq 1$, and a desired direction (the pseudo-force) f_i to each piece i. The model's objective is to maximize

$$\sum_{i=1}^{n} f_i \cdot v_i,$$

i.e., find the velocities that move the pieces in the desired directions as fast as possible.

The velocities must satisfy a set of non-penetration constraints. These constraints are calculated based on the pairs of pieces that are currently in contact. The purpose of these constraints is to make sure that once the pieces start to move with the velocities calculated from these constraints, the pieces that are currently in contact cannot interpenetrate in an infinitesimal time-interval.

We now describe how to build the non-penetration constraints. Let P and Q be two pieces in contact. Assume vertex A of polygon P is touching edge BC of polygon Q. We shall call A and BC a *vertex-edge touching pair*. By our convention, we order the points on the boundary of a polygon counterclockwise, i.e., the interior of polygon Q is on the left side of the directed edge BC. Let polygon P move with velocity u and polygon Q with velocity v. After a time interval t, A is moved to $A' = (A + ut)$ and edge BC is moved to $B'C'$ where $B' = (B + vt)$ and $C' = (C + vt)$ (see Fig. 6).

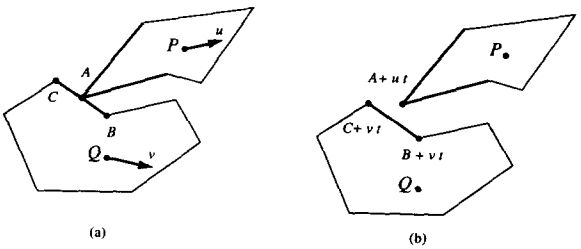

Fig. 6. (a) The vertex A of P touches the edge BC of Q. Polygon P moves with velocity u, and Q moves with velocity v; (b) the position of P and Q after time t.

The condition that A' does not penetrate $B'C'$ is

$$A'B' \times C'B' \geq 0,$$

that is,

$$((B + vt) - (A + ut)) \times ((B + vt) - (C + vt)) \geq 0.$$

This implies

$$(B - A) \times (B - C) + (B - A) \times (u - v)t \geq 0$$

and

$$AB \times CB + AB \times (u - v)t \geq 0.$$

Since A was touching BC,

$$AB \times CB = 0,$$

and since $t > 0$, the non-penetration constraint is

$$AB \times (u - v) \geq 0, \tag{1}$$

which is a linear constraint on the velocity. The vertex-vertex touching case is broken into several vertex-edge touching cases.

We build a velocity-based model containing linear constraints for all vertex-edge touching pairs. The resulting linear program is then solved to get a new set of velocities which do not cause inter-penetration among the polygons, at least over an infinitesimal time-interval. Next, a collision detection algorithm [5,7] finds the time $t_{\min}$ at which the first collision occurs between a vertex-edge pair which was not previously part of the constraints. The simulation proceeds by the time interval $t_{\min}$: polygon i is moved by $v_i \cdot t_{\min}$. At this point, some original vertex-edge touching pairs break up and some new pairs form. Before we go on, we need to recalculate the constraints with the newly formed vertex-edge touch pairs. The algorithm repeats the above process until $t_{\min}$ found by the collision detection algorithm is 0, which means the polygons cannot move further.

To calculate t_{min}, the collision detection algorithm examines every pair of translating pieces. For a pair of translating pieces P and Q, where P moves with velocity u and Q moves with velocity v, the first collision of P and Q occurs between a vertex of P and an edge of Q or vice versa. Thus, we can update $t_{\min}$ by examining the collision time for every vertex of P with every edge of Q and vice versa. During the algorithm, $t_{\min}$ always holds the earliest collision time detected so far. Initially, $t_{\min}$ can be assigned a large value within which a collision must occur. If a vertex A of P and an edge BC of Q collide before t_{min}, t_{min} is updated to the collision time of A and BC. Otherwise, $t_{\min}$ remains the same.

Notice that the collision time between A moving with velocity u and BC moving with velocity v is the same as the collision time between A moving with relative velocity $u - v$ and a stationary BC. Thus, there can be a collision between A and BC within time $t_{\min}$ if and only if the line segment AA' intersects the line segment BC, where $A' = A + (u - v)t_{\min}$. The intersection between AA' and BC can be detected and calculated by the numerically robust methods in [24] and [27].

The above description captures the essential steps that a collision detection algorithm must perform. We would like to point out that techniques from computational geometry [12] [32] can reduce the running time of some of these steps. For example, it is not necessary to examine all $\frac{1}{2}n(n-1)$ pairs of pieces, where n is the total number of pieces. Rather, one need only to consider the pairs of pieces that collide with each other before they could collide with other pieces. We refer these pairs as *adjacent* pairs, and they form the edges of an *adjacency* graph on the pieces. If the marker is tightly packed and each piece is in contact with its neighbors, the adjacency graph is planar and therefore has no more than $3n$ edges. Even if the marker is loosely packed, the number of adjacent pairs is essentially linear in the number of pieces. We quickly identify potential neighbors in $O(n \log n)$ time by replacing each piece by its bounding box and then applying a sweepline algorithm.

The expensive step in contact force model methods (see Section 1.3) is that of solving a system of differential equations. The velocity-based model replaces this step with a less expensive step of solving a linear program. Even so, this algorithm is still very slow on practical problems involving more than one hundred polygons, and it is not numerically robust. The main reason for the slow running time is the large number of time intervals it takes before reaching the local minimum. When the layout is getting tight, the chance of collision increases dramatically and the time interval for each step becomes very small. The more polygons there are in a layout, the more frequently the algorithm takes small time intervals. The small time step can also cause numerical difficulties in setting up constraints and collision detection. For a marker of 45 polygons, the running time is about 40–50 minutes on a 28 MIPS Sun SparcStation™.

However, we observe that the algorithm can quickly move a set of sparsely laid polygons into a 'relatively' tight state, perhaps within 0.25% of a local optimum. Therefore, it can still serve as a good alternative to the existing physically based simulation methods when dealing with loosely packed layouts. If a relatively tight state is good enough, one could also replace the polygons by simpler outer approximations, as Stoyan has done with his compaction technique (Section 1.3). Our observation is that humans generate very tightly packed layouts which are closer than 0.25% to a local optimum, and so the velocity-based algorithm is not a practical way to compact these layouts.

4. The theory of the Minkowski sum

Our next optimization model is based on the concept of *Minkowski sums*. Minkowski sums are widely used in robot motion planning [7,22,34] and in image analysis [35]. Throughout the automated marker making project [19,25,26], we have successfully shown the great utility of the Minkowski sum (and its dual, an operation called the *Minkowski difference*) in packing and placement problems. By using Minkowski sums and differences, we can convert a polygon–polygon

intersection (overlap) query and a polygon–polygon containment query into point-in-polygon location queries.

In this paper, the algorithms presented use the Minkowski sum only. Therefore, this section concentrates on the definition and properties of the Minkowski sum.

4.1. Definition and properties

We present the definition of the Minkowski sum in set theoretic terms. The lemmas and theorems in this section are valid for discrete or continuous point sets in Euclidean d-space.

Definition 4.1. Let A and B be two point sets in the Euclidean space. The *Minkowski sum* of A and B, denoted by $A \oplus B$, is a point set:

$$A \oplus B = \bigcup_{b \in B} A^b,$$

where A^b is A translated by b:

$$A^b = \{a + b \mid a \in A\}.$$

Lemma 4.1 shows that $A \oplus B$ can be written equivalently as an algebraic sum of A and B. The algebraic sum is sometimes easier to apply. Hence, this lemma can also serve as an alternate definition of the Minkowski sum.

Lemma 4.1. $A \oplus B = \{a + b \mid a \in A \wedge b \in B\}$.

Proof. ($\Rightarrow$) By definition, $x \in A \oplus B$ implies that there exists $b \in B$ such that $x \in A^b$. It follows that there exists $a \in A$ such that $x = a + b$. Therefore, x is also an element of the set on the right-hand side. ($\Leftarrow$) Similar. □

Corollary 4.2. $A \oplus B = B \oplus A$.

The following lemma shows that the 'shape' of the Minkowski sum is translationally invariant. Consequently, if the point sets A and B can only change location but not orientation, we need to compute their Minkowski sum just once. When A and B are placed in new locations, we translate their Minkowski sum accordingly.

Lemma 4.3. *Let A and B be two point sets. Let s and t be two points. Then*

$$A^s \oplus B^t = (A \oplus B)^{s+t}.$$

Proof. The proof is straightforward from Lemma 4.1. □

4.2. Application: Intersection detection

Theorem 4.4. *Let A and B be two point sets and x be a point in the plane. Then $A \cap B^x \neq \emptyset$ if and only if $x \in A \oplus (-B)$, where $(-B) = \{-b \mid b \in B\}$ is the reflective image of B with respect to the origin of the global coordinate system.*

Proof: ($\Rightarrow$) Let $y \in A \cap B^x$. We have $y \in A$ and $y \in B^x$. That is, there exists $b \in B$ such that $y = b + x$. Therefore, we have $x = y + (-b)$. Note that $y + (-b)$ is a point in $A \oplus (-B)$. Thus, $x \in A \oplus (-B)$. ($\Leftarrow$) If $x \in A \oplus (-B)$, then we have $x = a + (-b)$ for $a \in A$ and $b \in B$. Rewrite $x = a + (-b)$ as $a = x + b$. Therefore, a belongs to both A and B^x which implies $A \cap B^x \neq \emptyset$. □

Corollary 4.5. $A^s \cap B^t \neq \emptyset$ *if and only if* $(t - s) \in A \oplus (-B)$.

Proof. A^s and B^t intersect if and only if they intersect after they are both translated by $(-s)$, i.e., if and only if $A \cap B^{t-s} \neq \emptyset$. □

From Corollary 4.5, we immediately obtain the useful fact that A and B intersect if and only if $A \oplus (-B)$ contains the origin.

In the marker making applications, there is a local coordinate frame attached to each polygon. The coordinates of the vertices of the polygon are relative to the local coordinate system. The location of a polygon in the global coordinate system is given by the global coordinates of the polygon's local origin. Let $P(p)$ denote the fact that the local origin of polygon P is placed at point p in the global coordinate system. By convention, when the location for P is missing, the default value is the global origin. As it applies to two polygons P and Q, Corollary 4.5 has the geometric interpretation that $P(p)$ and $Q(q)$ intersect if and only if $q - p$ is in the Minkowski sum polygon $P \oplus (-Q)$.

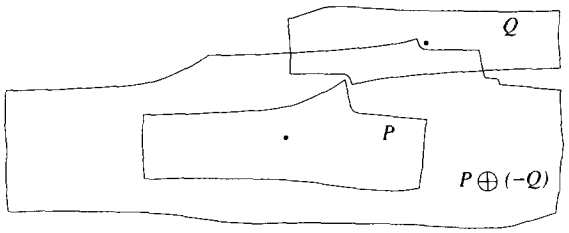

Fig. 7. The Minkowski sum and non-overlapping placement.

Since Q's local origin coincides with the global origin, $(-Q)$ is simply the polygon Q rotated 180 degrees around its local origin.

Fig. 7 shows that whenever the local origin of polygon Q is outside the Minkowski sum of P and $(-Q)$, Q cannot overlap P.

4.3. Algorithms for computing Minkowski sums

Previously proposed algorithms for computing Minkowski sums have been limited to convex polygons and simple polygons (a polygon is a simple polygon if it is non-self-intersecting and without holes). Guibas et al. [13] compute the Minkowski sum of two convex polygons in linear time by merging the edges of the two polygons. For general polygons, let us call an edge formed by the sum of a vertex in one polygon and an edge of another polygon (or *vice versa*) a candidate edge for those two polygons. Any edge on the boundary of the Minkowski sum of polygons P and Q is a subset of a candidate edge. If there are n vertices in P and m vertices in Q, then there are $O(mn)$ candidate edges. A natural idea for generating the Minkowski sum is to calculate the arrangement [12] of the candidate edges in $O(m^2n^2 \log(nm))$ time. The algorithms in [17] and [1] for calculating the Minkowski sum of two simple polygons followed this idea. Kaul et al. introduced the concept of vertex-edge supporting pairs which reduces the number of candidate edges. Despite this reduction, they show that in the worst case, the Minkowski sum of two simple polygons can have $O(m^2n^2)$ edges and the same number of holes.

The class of *star-shaped* polygons (see [32]) is more general than convex polygons but less general than simple polygons. For any polygon P, a *kernel point* is a point k in the interior that can 'see' the entire polygon: for any other point $u \in P$, the entire segment $\overline{ku}$ lies inside P. For convex polygons, the *kernel* (the set of kernel points) equals the polygon plus its interior. For simple polygons, the kernel may or may not be empty. A polygon is star-shaped if its kernel is not empty. We now show how to compute the Minkowski sum of star-shaped polygons. First, we prove a crucial property for the Minkowski sum of two star-shaped polygons that greatly simplifies the computing of the Minkowski sum.

Theorem 4.6. *The Minkowski sum of two star-shaped polygons is also a star-shaped polygon.*

Proof. Let P and Q be two star-shaped polygons. Let k_1 be a kernel point of P and let k_2 be a kernel point of Q. It suffices to show the $k_0 = k_1 + k_2$ is a kernel point of $P \oplus Q$. To see this, let u be an arbitrary point in $P \oplus Q$. By definition, there exist $v \in P$ and $w \in Q$ such that $u = v + w$. Since P and Q are star-shaped, we have that P contains $\overline{k_1 v}$ and that Q contains $\overline{k_2 w}$ (see Figure 8). Therefore, the Minkowski sum $P \oplus Q$ contains $\overline{k_1 v} \oplus \overline{k_2 w}$. The parallelogram $\overline{k_1 v} \oplus \overline{k_2 w}$ has k_0 at one end of a diagonal and $u = v + w$ at the other end. Hence, $P \oplus Q$ contains $\overline{k_0 u}$. It follows that k_0 is a kernel point of $P \oplus Q$. □

Remark. The theorem shows that star-shaped polygons are 'closed' under Minkowski sum operations. The only previously known class of poly-

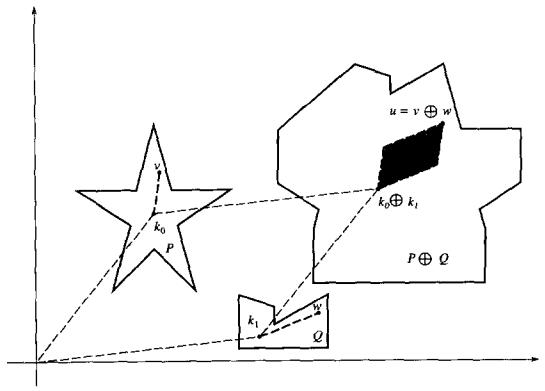

Fig. 8. The Minkowski sum of two star-shaped polygons.

gons closed under the Minkowski sum is the class of convex polygons.

Theorem 4 6 implies that the Minkowski sum of two star-shaped polygons cannot have holes. Thus, the computation of the Minkowski sum is reduced to calculating the outer envelope [12] of the arrangement of the $O(mn)$ candidates by an angular sweepline algorithm. The outer envelope of $O(mn)$ segments can have $O(mn\alpha(mn))$ segments where $\alpha(\cdot)$ is the extremely slowly growing inverse of the Ackermann's function [14]. For practical purposes, it can be considered a constant. The straightforward implementation of the angular sweepline algorithm runs in $O(mn\alpha(mn)\log(mn))$ time. Hershberger [15] presented an algorithm for calculating the outer envelope of n line segments in $O(n \log n)$ time. Therefore, we have the following.

Theorem 4.7. *The Minkowski sum of two star-shaped polygons can be computed in* $O(mn \log(mn))$ *time.*

Currently we are using a numerically robust implementation of the angular sweepline algorithm for computing the Minkowski sum of star-shaped polygons. We observe that if we are 'unsure' if a point q lies on the Minkowski sum, we can always project from the kernel point p through q and take the farthest intersection with a candidate edge. It is always safe to replace a point with this farthest intersection. This observation ensures the robustness of the algorithm. We have encountered data containing a few non-star-shaped polygons, but our studies have shown that all these polygons can be expressed as a union of two and very rarely three star-shaped polygons. For those non-star-shaped polygons, we have an algorithm to decompose them into a small number of star-shaped ones.

5. Compaction using a position-based optimization model

In this section, we describe a compaction algorithm that directly calculates new positions for the polygons without the use of time-intervals or simulation. Instead, this algorithm uses a *position-based* optimization model. The model contains artificial constraints that restrict the solution space to a convex feasible region. These artificial constraints are generated using a *locality heuristic*. This heuristic relies on Minkowski sums of polygonal regions to create an explicit representation of the set of non-overlapping positions of each pair of polygons. The objective in the model is a linear function of the positions of the polygons. The algorithm solves this model using linear programming, generating a new set of positions for the polygons. Since artificial constraints were added, the new set of positions may not be a local minimum of the objective for the original compaction problem. The algorithm computes a new model and repeats, stopping when two consecutive models are identical. In practice, the algorithm requires very few iterations to find a local minimum of the original compaction problem. This improved algorithm runs two orders of magnitude faster than the previous velocity-based algorithm.

5.1. A locality heuristic

We represent each polygon in a local coordinate system. The position of polygon P is given by the global coordinate c_P of its local origin. We assume that the local origin of each polygon is a kernel point of that polygon. (If any polygon is not star-shaped, then we decompose it into star-shaped polygons and add the constraint that these polygons must move as one unit.) Recall that $P(p)$ is a copy of P with its local origin positioned at p. From the previous section, we have that polygon $Q(q)$ intersects polygon $P(p)$ if and only if $(q-p) \in P \oplus (-Q)$. Hence, to guarantee that P and Q do not overlap, we must force $q-p$ to stay in $\overline{P \oplus (-Q)}$, the exterior of $P \oplus (-Q)$.

First, our intention is to formalize the compaction problem as a linear programming problem. Therefore, we must find a set of linear constraints on the position variable $q-p$ which represents the fact that the point $q-p$ must stay in $\overline{P \oplus (-Q)}$. One way of building such a linear

constraint is to specify a line segment and restrict $q-p$ to lie in the proper half-plane delimited by the line segment. For instance, given a line segment AB, if we let D denote $q-p$, the following cross product:

$$DA \times BA \geq 0,$$

specifies that D lies in the half-plane delimited by and on the right-hand side (with respect to the direction from A to B) of AB. The cross product expands to the linear constraint

$$(B_y - A_y)q_x - (B_x - A_x)q_y - (B_y - A_y)p_x + (B_x - A_x)p_y + A_xB_y - A_yB_x \geq 0,$$

where A_x, A_y, B_x, B_y are constants.

Geometrically, a set of linear constraints defines a convex region which is the intersection of the half-planes associated with the constraints. Conversely, from a convex region in the plane, we can derive a unique set of linear constraints on $q-p$. Furthermore, $q-p$ satisfies the set of linear constraints associated with a convex region if and only if the convex region contains $q-p$.

If we select a (non-empty) convex region $R \subseteq \overline{P \oplus (-Q)}$, we can form a set of linear constraints which forces $q-p$ to stay in R, thus ensuring that P and Q do not overlap. If $\overline{P \oplus (-Q)}$ were convex, we would use $\overline{P \oplus (-Q)}$ itself as the convex region to build constraints. Unfortunately, $\overline{P \oplus (-Q)}$ can never be convex, and therefore any convex region we choose can only be a subset of $\overline{P \oplus (-Q)}$. By forcing $q-p$ to stay in a convex subset $R \subseteq \overline{P \oplus (-Q)}$, we artificially restrict the freedom of the relative motion of P and Q. To avoid restricting too much, we want R to cover as large an area of $\overline{P \oplus (-Q)}$ as possible.

Let c_P and c_Q be the current positions of the origins of P and Q. We want the new relative position $q-p$ to be reachable from the current relative position $c_Q - c_P$ via a continuous motion of P and Q. Hence, we also require that the convex region include $c_Q - c_P$.

The purpose of the *locality heuristic* is to find a region $R \subseteq \overline{P \oplus (-Q)}$ with the three requirements we just identified: convex, large, and containing the point $c_Q - c_P$. The locality heuristic finds such a convex region as follows. First, it calculates a point on the boundary of the Minkowski sum that is 'nearest' (to be defined below) to $c_Q - c_P$. Starting from that point, it walks clockwise and counterclockwise along the boundary of the Minkowski sum. When walking clockwise (with respect to the origin of the Minkowski sum), it always makes left turns. If the next edge turns to the left (at a concave vertex of the Minkowski sum), it follows it. Otherwise, it extends the current edge until it intersects the Minkowski sum, and it resumes the walk from that point. This procedure continues until it can extend the current edge to infinity. Restarting from the original 'nearest' point, it proceeds analogously in the counterclockwise direction, making right turns instead of left turns (see Fig. 9). The heuristic outputs a set of constraints consisting of the half-planes to the 'outside' of each Minkowski edge it encounters. The intersection of these half-planes is a convex subset R of the feasible solution space.

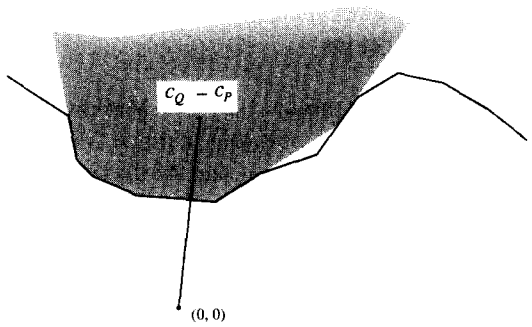

Fig. 9. The nearest convex region in the exterior of the Minkowski sum.

Under the locality heuristic, the 'nearest' point is not defined to be the boundary point with the closest Euclidean distance to $c_Q - c_P$. Instead, it is the intersection of the segment from the origin to point $c_Q - c_P$ with the boundary of the Minkowski sum. If $c_Q - c_P$ is inside the Minkowski sum, as it would be when we are separating overlapping polygons, this segment does not intersect the boundary. In this case, we extend the segment into a ray from the origin through the point $c_Q - c_P$, and we compute the point at which this ray intersects the boundary. This choice of nearest point is important for the

correct operation of the separation algorithm and is described more fully in Section 6.

5.2. The compaction algorithm

We first present a brief description of the algorithm and then give further explanation. As in the algorithm in Section 3, we assign a desired direction f_i to each polygon; however, instead of assigning a velocity, we use a variable p_i representing the position of the polygon.

A position based compaction algorithm.

for $i \leftarrow 1, 2, 3, \ldots, n$ **do**
  Input piece P_i;
  Input constant 'force' f_i on piece P_i;
  Input current position c_i of piece P_i;
  Initialize variable $p_i \leftarrow c_i$;
end
do
  $L \leftarrow$ the list of adjacent pairs of pieces returned from a sweepline algorithm;
  $S \leftarrow \emptyset$; / * *S is the set of linear constraints generated so far* * /
  foreach adjacent pair (P_i, P_j) in L **do**
    Compute $\overline{P_i \oplus (-P_j)}$;
    Apply the locality heuristic to $\overline{P_i \oplus (-P_j)}$ and $c_j - c_i$;
    $S_{ij} \leftarrow$ the set of linear constraints on $p_j - p_i$ generated by the locality heuristic;
    $S \leftarrow S \cup S_{ij}$;
  end
  Set up a linear program
    maximize $\sum_{i=1}^{n} f_i \cdot p_i$ *subject to* S;
  Get the set of new positions p_i $(i = 1, \ldots, n)$ by solving the linear program;
  for $i \leftarrow 1, 2, 3, \ldots, n$ **do**
    $c_i \leftarrow p_i$; / * *move piece* P_i *to* p_i * /
  end;
until the objective function $\sum_{i=1}^{n} f_i \cdot p_i$ stops increasing.

The non-overlapping constraints of the position-based model are built in the following way. We first use a sweepline algorithm to find all the adjacent pairs of polygons. For each adjacent pair, we find the set of half-plane constraints generated by the locality heuristic in the previous section.

The optimization model is set up with all the constraints from each adjacent pair and the objective function

$$\text{maximize} \sum_{i=1}^{n} f_i \cdot p_i.$$

This model is a linear program, and its solution is a set of positions with two properties: it satisfies the current set of constraints, and it represents a maximal motion in the desired directions.

After moving to the new positions, the system reaches a minimum with respect to the set of constraints generated by the locality heuristic. Given the new positions as inputs, the locality heuristic may generate different convex regions. In this case, the motion planning algorithm must repeat. Note that the algorithm uses the same linear objective function for each iteration. Either the value of the objective function increases, which triggers the next iteration, or it stays the same, at which point the system has reached a local minimum for the original compaction problem.

The final position is a local minimum for the unrestricted solution space because of the manner in which the locality heuristic constructs the restricted convex subset for adjacent pieces. For any pair of adjacent pieces P_i and P_j, the constructed convex region $R_{ij} \subseteq \overline{P_i \oplus (-P_j)}$ is identical to $\overline{P_i \oplus (-P_j)}$ in some open ball about $c_j - c_i$. Therefore, the restricted solution space is identical to the unrestricted solution space within some $2n$-dimensional ball about the starting configuration $(c_1, c_2, \ldots, c_n)$. If the starting configuration is a global minimum of the restricted space, it is a local minimum of the unrestricted space.

Furthermore, each step of the algorithm moves the polygons from one point to another in the same convex subset of the non-convex feasible (non-overlapping) solution space. Therefore, the straight line motion for each step stays within the solution space, and the total motion is a piecewise linear subset of the solution space. (Actually, manufacturers would not mind if the polygons

'leapfrogged' over each other on the way to a more compact layout. We have recently developed a generalization of our compaction algorithm based on mixed integer programming which does exactly that.)

6. Separation of overlapping polygons

In this section, we consider the problem of separating overlapping polygons. Given a set of overlapping polygons, the problem is to find a set of translations of the polygons which eliminates all overlaps and has the minimum total translation.

First we show that finding a global minimum for the total translation is NP-complete. Next we show how to modify the compaction algorithm of the previous section to find a local minimum of the separation problem.

6.1. Lower bounds

Theorem 6.1. *The separation of overlapping polygons is NP-complete even for rectangles.*

Proof. Since rotation is not allowed, we can use the same argument as in Corollary 2.5 or [25] to show that the problem is in NP. To show it is NP-hard, we again reduce PARTITION to the problem. Fig. 10 shows the construction of the reduction. Let $(a_1, a_2, \ldots, a_n)$ be the integers in an instance of PARTITION. Let $B = \frac{1}{2}\sum_{i=1}^{n} a_i$. Place n rectangular pieces of height one and length a_i $(i = 1, \ldots, n)$ inside a rectangular space of size $2 \times B$. In this initial placement, the pieces overlap each other. Surround the rectangular space by many additional rectangular 'blocks' that would make it hard to enlarge its size. For one piece to move from one position to another position inside the rectangular space, it needs to move no more than one unit vertically and B units horizontally. The total motion of the pieces is less than $B + B^2$. One cannot increase the length or the width of the rectangular space without moving the surrounding blocks a total of $B + B^2$ units distance. Therefore, PARTITION has a solution if and only if the total displacement in the separation is less than $B + B^2$. The construction can be done in polynomial time. □

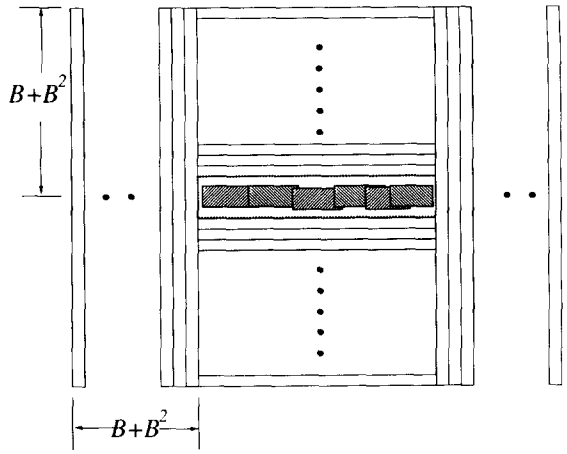

Fig. 10. Reduction of PARTITION to separation of overlapping polygons.

For the cases we are dealing with, the polygons just slightly overlap. We can show that separating slightly overlapping polygons is no easier. For simplicity, we define the degree of overlap r for two overlapping polygons P and Q as

$$r = \max\left(\frac{\text{area}(P \cap Q)}{\text{area}(P)}, \frac{\text{area}(P \cap Q)}{\text{area}(Q)}\right).$$

We say the two polygons are *slightly* overlapping if $r < 1/c$ for some constant integer $c > 1$.

Theorem 6.2. *The separation of overlapping polygons is NP-complete even if the polygons just slightly overlap each other.*

Proof. We will reduce the NP-hard 2-PARTITION problem to this problem. The 2-PARTITION problem is as follows: given nonnegative integers n, $a_1, \ldots, a_n$ with $\sum_{j=1}^{n} a_j = 2B$, can these integers be partitioned into two subsets such that each subset contains $\frac{1}{2}n$ integers and the sums of the two subsets are equal? We use a construction similar to that in the previous proof. Each piece still has height one but width $2Bc + a_i$. The rectangular space in the middle now has size $2 \times (2Bnc + B)$. We divide the n pieces into two rows

each containing $\frac{1}{2}n$ pieces within the middle space. The two rows do not overlap vertically. Within each row, we evenly spread out the pieces. The degree of overlap between two adjacent pieces in each row is at most

$$2B/(2Bc + a_i) \leq 1/c$$

because $2B$ is the total overlap. If there is a solution to 2-PARTITION, the moves needed to make the pieces non-overlapping are as follows: exchange two vertically adjacent pieces when necessary and do horizontal adjustment within each row. The vertical exchanges have a total displacement of at most $2n$. The total horizontal displacement within each row is at most nB because each piece has at most B horizontal displacement. So with less than $2n(B + 1)$ total displacement, we can solve the separation problem. We will put $2n(B + 1)$ surrounding blocks on each side to restrict the movement of the pieces within the middle space. □

It is easy to see that the constant c can be replaced by a polynomial $F(B, n)$ and the proof still works.

6.2. *The separation algorithm*

This section describes a modification of the compaction algorithm that finds a local minimum for the separation problem using a similar position-based optimization model

For two overlapping polygons P and Q, if we displace P by d_P and Q by d_Q, then the two polygons will not overlap in their new positions if and only if $c_Q - c_P + d_Q - d_P$ is outside the Minkowski sum $P \oplus (-Q)$. The vector from $c_Q - c_P$ to the closest boundary point on the Minkowski sum yields the shortest vector $d_Q - d_P$ that separates the two polygons. Suppose we constrain $c_Q - c_P + d_Q - d_P$ to remain within a convex subset of the exterior of $P \oplus (-Q)$ that touches the closest boundary point. If we so constrain every pair of overlapping polygons, then if a feasible solution with respect to the constraints exists, the solution will give displacements that separate the polygons.

Remark. we use the 'nearest' point, as defined by the locality heuristic, instead of the Euclidean closest point. This choice guarantees a feasible solution if the following three conditions hold:

- the polygons are not restricted to stay within a bounding box (such as the rectangular sheet of material);
- all polygons are star-shaped;
- no two polygons have their local origins at the same global position.

To see this is the case, consider shrinking all the polygons toward their local origins the same amount until they are not overlapping. Then scale up the whole layout about the global origin until the polygons are their original size. The result is a non-overlapping layout. However, this feasibility proof depends on the fact that each 'nearest' point stays at the same spot on each shrinking polygon. This holds for the given definition of 'nearest' point. It does not hold for the Euclidean closest point. Normally, we cannot ignore the sheet of material and the bounding box constraint it imposes, but at least the other two conditions hold for the layouts we consider. If the bounding box does not have to grow too much, then perhaps a sheet of material of that size is available.

As shown in Fig. 11, if two polygons are P and Q are slightly overlapping, then the difference $c_Q - c_P$ between the polygon positions will be slightly inside the Minkowski sum $P \oplus (-Q)$. The

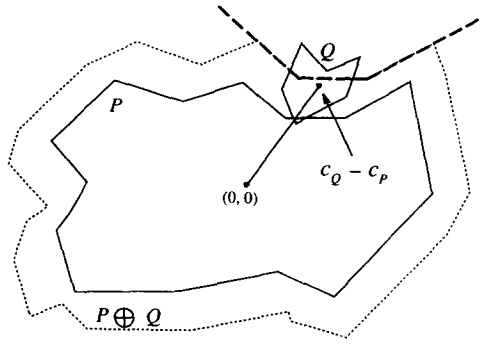

Fig. 11. The Minkowski sum of two slightly overlapped polygons.

convex region (dashed boundary) found by the locality heuristic still gives a good indication of the direction of motions to separate the two polygons. Thus, we can use the locality heuristic from the last section to find the convex regions. The constraints are the same as those of the compaction algorithm in the previous section. The objective function becomes

$$\text{minimize} \sum_{i=1}^{n} |q_{ix}| + |q_{iy}|,$$

where $q_i = p_i - c_i$ is the displacement of polygon P_i from its current position to its new position. We can use a standard technique in linear programming to eliminate the absolute values. If the set of constraints of this linear program is feasible, then the solution gives the locally minimal set of displacements that will separate the polygons.

7. Layout made easy

The separation algorithm of the previous section solves an outstanding problem in database-driven systems for automated marker making. Such systems, developed by several CAD/CAM firms, use a database of high quality human-generated markers.

Some human marker makers with twenty or more years of experience can generate extremely high quality markers. The markers generated by marker makers with only two or three years of experience are often 1% lower in efficiency than the markers made by the most experienced marker makers. Commercially available systems automatically generate markers at least 5% below the efficiency of the best human-generated markers. It is thought that if these systems could somehow start with the top quality human-generated markers as an initial configuration, then they could generate much more efficient markers. This is the idea behind a database marker making system.

Given a new set of polygons to be placed, the typical database matching system finds the marker in the database that has a similar combination of sizes and shapes. The system matches each new polygon to be placed to a polygon in the human-generated marker, and it assigns the position and orientation of the matched marker polygon to its corresponding new polygon. It uses a 'one polygon at a time' strategy to place each new polygon in a new marker at the assigned position and orientation. If the polygon overlaps previously placed polygons in the new marker, its position must be 'corrected' so that it does not overlap these polygons. Once a polygon is correctly placed, its position is frozen. Without a coordinated overlap correction method, this form of 'one polygon at a time' correction can grossly alter the layout of the marker and make it difficult or even impossible to place the rest of the polygons anywhere near the position of their matching pieces. In many cases the correction algorithm fails to find a non-overlapping position for the polygon. As a result, these systems do not replace human marker makers.

With our separation algorithm, we can give a simpler and more reliable layout algorithm. First, lay out each polygon at the corresponding position and orientation of its matching polygon in the human-generated layout. Do not try to eliminate overlaps. After placing all the polygons, apply the separation algorithm to find a nearby feasible non-overlapping placement if one exists. If no such placement exists, increase the length of the marker to allow such a placement. Afterwards, apply the leftward compaction algorithm to shorten the length.

If we have freedom in both dimensions of the marker, we can always find a nearby non-overlapping placement (see the Remark in Section 6.2). If the width is fixed, our separation algorithm can fail to find a feasible placement. However, this will only happen when gross alterations are required to create a valid marker. We cannot expect to do better because the general problem is NP-complete.

Recent studies in computational geometry offer some general technique on matching polygon shapes [2,3]. However, these techniques do not take advantage of the properties that the polygons in a specific application area possess, and thus they are expensive to use. Domain specific knowledge can sometimes provide direct and ef-

fective heuristics for matching and substituting polygons. Take trouser markers as an example. There are certain common characteristics about the shape of a basic component such as a front panel piece. In addition, the size specification governs the dimensions of the pieces, i.e., the panel pieces for trousers of the same waist and length all have approximately the same dimensions. Although the exact shapes of front panel pieces differ from style to style, same-sized panel pieces have very similar shapes. Type and size alone can usually decide whether two pieces match in a database-driven marker making system.

Currently, the separation algorithm works as the engine of our proposed database-driven marker making system. It does not mandate any specific shape similarity matching rule. Ultimately, experience may decide the best matching rules, rather than mathematics. The user of our algorithm, a CAD company in textile industry, is currently designing effective matching rules for their customers' markers.

8. Performance

We have combined our algorithms into one separation/compaction algorithm with multiple user-selected options. This section describes the performance of our algorithm and its application to database-driven automatic marker making.

8.1. Running time and robustness

Because the adjacency graph is planar, there are only a linear number of adjacent pairs of polygons. A sweepline algorithm finds all these pairs. The convex region found by the locality heuristic usually has only a few edges on its boundary. Thus, in practice, the total number of constraints is linear in the input size. The algorithm typically runs in two to five iterations before it stops. Leftward compaction for a marker of 120 polygons, with the largest polygons having nearly 100 vertices, runs in 20 seconds on a 28 MIPS Sun SparcStation™.

The algorithm is also numerically very robust. It handles degenerate cases, slightly overlapped inputs and inaccurate Minkowski sums quite well. In particular, the vertices of the Minkowski sums we use have been rounded to the nearest integer lattice point.

8.2. Improvement in cloth utilization

The algorithm has improved the cloth utilization of many production quality human-generated markers (markers which have actually gone to the cutting room). The average improvement for the markers from a large jeans manufacturers is 0.32%; the average savings in material is 1.36 inches per marker. As Section 1.2 indicated, a 0.1% improvement in efficiency would save about two million dollars in material for the manufacturer. In the leftward compaction example shown in Fig. 1, the efficiency of the marker has increased by nearly 1% and total length shortened by 5 inches.

Recently, two apparel manufacturers (one of blue jeans, the other of trousers) tested our software and achieved average improvements of 0.15% and 0.2%, respectively. We estimate that compaction could reduce their material costs by one-quarter million dollars and one-half million dollars per year. These companies currently make every effort to tightly pack their markers. Our own tests on a lingerie company's markers indicate that compaction can achieve an average improvement of one percent or more. These markers appear to be generated semi-automatically and are therefore less efficient than the trouser markers.

8.3. Database-driven marker making

Using a naive polygon matching algorithm, we match a new set of polygons to the set of polygons in a human-generated marker. Fig. 12 shows the marker generated by substituting matching polygons. Using our separation/compaction algorithm, we eliminate the overlaps in the marker and then compact leftward. The resulting marker has an efficiency of 88.89%, which is comparable to human marker makers with two or three years of experience.

If we move three small polygons manually to the gap on the lower right corner of the marker

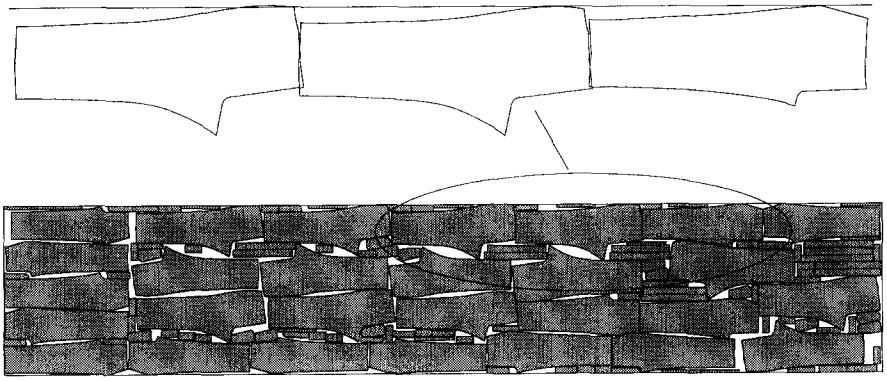

Fig. 12. Marker generated by matching to human-generating marker and using corresponding positions and orientations.

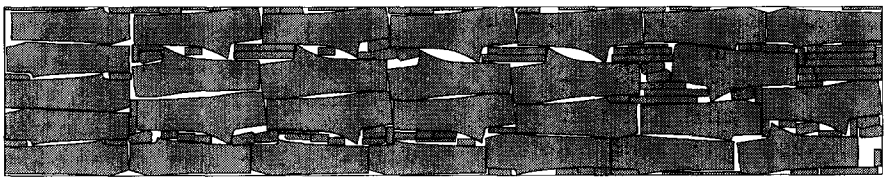

Fig. 13. Separation algorithm eliminates overlaps, and leftward compaction improves efficiency. Length = 312.64 in. Efficiency = 88.98%.

and run leftward compaction again, the efficiency increases to 89.49%, only 0.65% lower than the marker generated by an expert human (Fig. 15) for the same set of polygons. This shows that starting from a good initial configuration can greatly reduce the complexity of marker making and shows the applicability of our separation algorithm for database-driven marker making.

8.4. Extensions

In recent work, we have extended our position-based model to allow compaction with small rotations of the polygons [20]. We have two different methods. One extension involves relaxation of individual polygons: finding an orientation of each that gives it the largest free space. In the other extension, we consider the exact nonlinear optimization model for the entire set of polygons involving both translations and rotations. Instead of solving this model directly, we solve a linearized version. Translation-only separation eliminates small overlaps resulting from the linearization.

We have also created a mixed integer pro-

Fig. 14. Marker after manual adjustment of three small polygons. Length = 310.87 in. Efficiency = 89.48%.

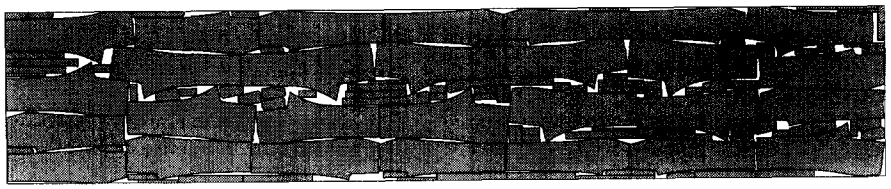

Fig. 15. An expert human generates this marker for the same set of pieces. Length = 308.61 in. Efficiency = 90.14%.

gramming version of position-based optimization that can find optimal layouts of small numbers of polygons [8]. Using the MINTO optimization package, we can sometimes create optimal layouts of up to nine polygons in a few minutes.

9. Concluding remarks

We have presented here the first fast and practical algorithm for separating and compacting a layout of convex and non-convex polygons. The algorithm is very effective in a real world application: marker making in apparel manufacturing. The algorithm is general and versatile. By simply modifying the objective function, we use the algorithm to perform different tasks in all stages of marker making: leftward compaction, opening gaps to place new pieces, and separation for database oriented automatic marker making. The separation/compaction algorithm uses a new position-based model that we have recently extended to compaction with rotation and optimal layout.

We have seen several researchers in the area of cutting and packing introduce the same concept using many different names: configuration space, Minkowski sum, hodograph, non-overlap space, and so forth. We hope our work will help others realize that these are all the same concept – the Minkowski sum – and that the field of computational geometry offers both theoretical and practical results on the Minkowski sum that they can draw on. We believe we have shown the great utility of these geometric algorithms, and we hope that our work will encourage others to become interested in the use of these and other algorithms from the field of computational geometry.

Acknowledgments

We would like to thank Prof. Fred Abernathy, Karen Daniels and Dr. Richard Szeliski for helpful discussions. Members of our research team, especially Karen Daniels, Lee Wexler and Venkatesh Reddy, have simplified the implementation of this work by building and maintaining Harvard Automated Marker Generation Testbed. Murray Daniels of MITRE Corporation provided the user interface software. Karen Daniels has carefully read the earlier drafts of this paper and offered many suggestions that enhanced the presentation of this paper. Dr. Veljko Milenkovic has kindly proofread the drafts of this paper and pointed out several typographical and grammatical errors.

References

[1] Agarwal, P.K., Sharir, M., and Toledo, S., "Applications of parametric searching in geometric optimization", in: *Proceedings of the Third ACM-SIAM Symposium on Discrete Algorithms,* SIAM, Philadelphia, PA, 1992, 72–82.

[2] Alt, H., Behrends, B., and Blömer, J., "Approximate matching of polygonal shapes", in: *Proceedings of the Seventh Annual ACM Symposium on Computational Geometry,* ACM, New York, 1991, 186–193.

[3] Alt, H. and Godau, M., "Measuring the resemblance of polygonal curves", in: *Proceedings of the Eight Annual ACM Symposium on Computational Geometry,* ACM, New York, 1992, 102–109.

[4] Anderson, R., Kahan, S., and Schlag, M., "An O(n log n) algorithm for 1-D tile compaction", in: M. Nagl (ed.), *Graph-Theoretic Concepts in Computer Science*, Lecture Notes in Computer Science, Vol. 411, Springer-Verlag, Berlin, 1990, 287–301.

[5] Baraff, D., "Analytical methods for dynamic simulation of non-penetrating rigid bodies", *Computer Graphics (Proceedings of SIGGRAPH)* 23/3 (1989) 223–232.

[6] Barzel, R., and Barr, A.H., "A modeling system based on

dynamics constraints", *Computer Graphics (Proceedings of SIGGRAPH)* 22/4 (1988) 179–187.

[7] Canny, J., *The Complexity of Robot Motion Planning*, MIT Press, Cambridge, MA, 1987.

[8] Daniels, K., Li, Z., and Milenkovic, V., "Multiple containment methods", Technical Report TR-12-94, Center for Computing Technology, Harvard University, Cambridge, MA, 1994.

[9] Daniels, K. and Milenkovic, V., "Limited gaps", in: M. Keil (ed.), *Proceedings of the Sixth Canadian Conference on Computational Geometry*, Saskatoon, Canada, 1994.

[10] Dobkin, D., Hershberger, J., Kirkpatrik, D., and Suri, S., "Implicitly searching convolutions and computing depth of collision", in: *Proceedings of the First SIGAL International Symposium on Algorithms*, Lecture Notes in Computer Science, Vol. 450, Springer-Verlag, Berlin, 1990, 165–180.

[11] Dyckhoff, H., "A typology of cutting and packing problems", *European Journal of Operational Research* 44 (1990) 145–159.

[12] Edelsbrunner, H., *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, 1987.

[13] Guibas, L., Ramshaw, L., and Stolfi, J., "A kinetic framework for computational geometry", in: *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science*, 1983, 100–111.

[14] Hart, S. and Sharir, M., "Nonlinearity of Davenport–Schinzel sequences and of generalized path compression schemes", *Combinatorica* 6 (1986) 151–177.

[15] Hershberger, J., "Finding the upper envelope of n line segments in $O(n \log n)$ time", *Information Processing Letters* 33 (1989) 169–174.

[16] Hopcroft, J.E., Schwartz, J.T., and Sharir, M., "On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the Warehouseman's problem", *The International Journal of Robotics Research* 3/4 (1984) 76–88.

[17] Kaul, A., O'Connor, M.A., and Srinivasan, V., "Computing Minkowski sums of regular polygons", in: T. Shermer (ed.), *Proceedings of the Third Canadian Conference on Computational Geometry*, Vancouver, BC, 1991, 74–77.

[18] Lengauer, T., "On the solution of inequality systems relevant to IC-layout", *Journal of Algorithms* 5/5 (1984) 408–421.

[19] Li, Z., and Milenkovic, V., "A compaction algorithm for non-convex polygons and its application", in: *Proceedings of the Ninth Annual ACM Symposium on Computational Geometry*, ACM, New York, 1993, 153–162.

[20] Li, Z., and Milenkovic, V., "Compaction of a 2D layout of non-convex shapes and applications", SIAM Conference on Geometric Design, 1993.

[21] Li, Z., and Milenkovic, V., "On the complexity of the compaction problem", in: A. Lubiw and J. Urrutia (eds.), *Proceedings of the Fifth Canadian Conference on Computational Geometry*, Waterloo, Ont., 1993, 153–162.

[22] Lozano-Pérez, T., and Wesley, M.A., "An algorithm for planning collision-free paths among polyhedral obstacles", *Communications of the ACM* 22 (1979) 560–570.

[23] Maple, D., "A hierarchy preserving hierarchical compaction", in: *Proceedings of the 27th Design Automation Conference*, ACM, New York, 1990, 375–381.

[24] Milenkovic, V., "Verifiable implementation of geometric algorithms using finite precision arithmetic", Ph.D. Thesis CMU-CS-88-168, Carnegie Mellon University, 1988.

[25] Milenkovic, V., Daniels, K., and Li, Z., "Automatic marker making", in: T. Shermer (ed.), *Proceedings of the Third Canadian Conference on Computational Geometry*, Vancouver, BC, 1991, 243–246.

[26] Milenkovic, V., Daniels, K., and Li, Z., "Placement and compaction of nonconvex polygons for clothing manufacture", in: *Proceedings of the Fourth Canadian Conference on Computational Geometry*, St. Johns, Nfld, August 1992, 236–243.

[27] Milenkovic, V., "Robust polygon modelling", *Computer-Aided Design* 25/9 (1993) 546–566.

[28] Moore, M., and Wilhelms, J., "Collision detection and response for computer animation", *Computer Graphics (Proceedings of SIGGRAPH)* 22/4 (1988) 289–298.

[29] Mosteller, R.C., Frey, A.H., and Suaya, R., "2-D compaction: a monte carlo method", in: P. Lasleber (eds.), *Advanced Research in VLSI*, MIT Press, Cambridge, MA, 1987, 173–197.

[30] Palmer, R.S., "Computational complexity of motion and stability of polygons", Ph.D. Thesis, Cornell University, January 1987.

[31] Platt, J.C., and Barr, A.H., "Constraint methods for flexible models", *Computer Graphics (Proceedings of SIGGRAPH)* 22/4 (1988) 279–287.

[32] Preparata, F.P., and Shamos, M.I., *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

[33] Savitch, W.J., "Relationships between nondeterministic and deterministic space complexities", *Journal of Computer and System Sciences* 9/3 (1970) 636–652.

[34] Schwartz, J.T., and Sharir, M., "Algorithmic motion planning in robotics", in: J. van Leeuwen (ed.), *Algorithms and Complexity*, Handbook of Theoretical Computer Science, Vol. A, Elsevier, Amsterdam, 1990, 391–430.

[35] Serra, J., *Image Analysis and Mathematical Morphology*, Vol. 1, Academic Press, New York, 1982.

[36] Shin, H., Sangiovanni-Vincentelli, A.L., and Séquin, C.H., "Two dimensional compaction by 'zone refining'", in: *Proceedings of the 23rd Design Automation Conference*, ACM, New York, 1986, 15–122.

[37] Stoyan, Y.G., "A basic problem of optimal cutting of materials for blanks of arbitrary 3D geometry space form", SICUP Workshop on Cutting and Packing at the ORSA/TIMS Joint National Meeting, 1992.

[38] Stoyan, Y.G., "Precise and approximate methods of solu-

tion of problems in irregular allocation of nonconvex polygons in a strip of minimal length", SICUP Workshop on Cutting and Packing at the IFORS Conference, 1993.

[39] Sweeney, P.E., and Paternoster, E.R., "Cutting and packing problems: A categorized, application-oriented research bibliography", *Journal of the Operations Research Society* 43/7 (1992) 691–706.

[40] Wong, C.K., "An optimal two-dimensional compaction scheme", in: P. Bertolazzi and F. Luccio (eds.), *VLSI: Algorithms and Architectures*, Elsevier, Amsterdam, 1985, 205–211.