Limited Gaps

Karen Daniels [*] *Victor Milenkovic*[†]
Center for Research in Computing Technology
Division of Applied Sciences
Harvard University
Cambridge, MA 02138

1 Introduction

For the past few years, we have been engaged in a project on automatically generating layouts for the apparel industry. A layout or *marker* is a non-overlapping placement of clothing pattern pieces (polygons) on a rectangular strip of material of fixed width. The task is to find the optimal (minimum length) layout. Marker making is NP-hard, and it is important to decompose the task as much as possible for the sake of running time, even if that means finding only a near-optimal layout.

Pants pattern pieces fall into two distinct groups: large panels and smaller trim pieces. One decomposition strategy is to first place the large panels and then place the smaller trim in the unused regions of material among the panels. The length of the final layout is determined by the panel placement. Humans employ this strategy, and they appear to obtain near-optimal layouts for pants.

After the panels are placed, the shape of the unused material has great complexity in terms of connectivity and number of edges. The human eye naturally decomposes this material into *gaps* of significant area. This second decomposition is also essential to reducing the time required for the layout. This paper describes our efforts to give a sound mathematical basis and a set of algorithms to accomplish this decomposition. We call each resulting region a *limited gap*.[1]

1.1 The Trim Layout Task

In the trim layout problem, trim is a set of (possibly nonconvex) polygons $P = \{p_1, p_2, \ldots, p_k\}$ to be packed into a nonconvex polygonal container C, usually with multiple components (see Figure 1), resulting from subtracting the panels from the rectangle of material. The goal is to fit as many items as possible into the container. This is an NP-hard problem, and is a generalization of 2D bin packing, where the rectangles (bins and items) are now allowed to be nonconvex polygons, and the bins need not be identical (each bin corresponds to a component of the container polygon). Assuming, $P \neq NP$, the cost of filling a container is exponential in k. We assume here that only translations on the elements of p are allowed.[2]

1.2 Decomposing C into Gaps

In its undecomposed form, the container set C is the complement of the set of obstacles in a rectangle of material. To improve tractability, we need to find a *gap set* $G = \{g_1, g_2, \ldots, g_n\}$ with the *containment property*: if $p \in P$ is placed inside C, then it is contained entirely inside some $g \in G$. The use of a gap set greatly improves the tractability of the layout task: it is usually possible to perform the layout by packing individual gaps. Smaller gaps and more gaps are desirable; overlap among gaps is undesirable.

The set of connected components of C has the containment property, but it usually has too few elements to improve tractability much in practice. We will define a set $R_P \subseteq C$ called the *reachable region* of P. The connected components of R_P form a much better gap set than C, but it still is often insufficiently decomposed. However, it is the largest cardinality gap set we know of with non-overlapping gaps.

If we permit gaps to overlap, then it is possible for polygons placed in different gaps to overlap. However, if we construct the gaps properly, then the probability of overlap will be small. We will give a rigorous notion of this probability. Further, for any given probability π, we will show how to construct a gap set, called *limited*

[*]This research was funded by the Textile/Clothing Technology Corporation from funds awarded to them by the Alfred P. Sloan Foundation.

[†]This research was 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]Later we noticed the double connection of the name to the retail side of the apparel business.

[2]In practice, some discrete orientation changes (reflections and rotation by multiples of 90 degrees) are also allowed.

gaps, such that the probability that pieces in different gaps overlap is less than π.

There is a tradeoff between the number and size of gaps and the probability. Many small gaps are easier to fill with trim pieces,[3] but the probability of overlap between trim pieces in different gaps is higher. Since π is a parameter to the construction of the limited gaps, we allow the possibility of finding the best value of π in the tradeoff.

1.3 Overview

Section 2 describes the container decomposition methods we experimented with before arriving at the notion of the limited gaps. Section 3 defines the limited gaps in terms of the probability π and gives an algorithm for constructing them. Section 4 presents two methods for improving the efficiency of limited gap construction. These methods introduce the new concepts of *breaking polygons* and *inking polygons*.

2 The Challenge of Constructing Gap Sets

Before defining the *limited gaps*, we review the other container decomposition methods we experimented with.

2.1 Compaction

We have a *compaction* tool [4] which plans a motion for the set of placed pieces under a set of applied "forces." We attempted to eliminate holes and narrow necks in the container set by applying a leftward force to each panel polygon. However, we found that this strategy had three shortcomings: 1) it failed to close all the narrow necks, 2) although the containers were simply connected, unnaturally large regions still existed, and 3) contact detection suffered from numerical instability. Through the use of a configuration space approach, the compaction algorithm avoids the question of contact detection in order to run quickly and numerically robustly. However, in order to generate a good decomposition, it was necessary to explicitly detect contacts, and this proved to be numerically unstable. This instability was exacerbated by the apparel industry's policy of rounding polygon coordinates before saving them in a data file.

[3]The problem is still NP-hard since we must also decide which pieces go into which gaps!

2.2 Partitioning Squares

Our second attempt involved choosing a narrowness tolerance γ. We formed a *partitioning square* with side of length γ and used this to partition containers. The partitioning process was conceptually analogous to centering a partitioning square at each vertex of each obstacle, and then removing those squares for which the intersection with the set of obstacles is connected (see Figure 2). The difficulties with this approach are: 1) it introduces artifacts (pieces of squares) into the layout, 2) it fragments the containers in a way that leaves many small regions which are not usable by the set of items to be placed, and 3) choosing a universal γ is inherently impossible. The fragmentation issue can be resolved by testing each region, but this is computationally expensive. Choosing γ is not possible because P does not naturally give rise to a narrowness value. Adopting a value corresponding to regions which no item can fit through is insufficient, because an item might be able to reach into an arbitrarily narrow neck from both sides, thus making the neck usable (see Figure 3).

2.3 Other Methods

We considered other methods, such as computing a Voronoi diagram of the obstacles, using edges of the obstacles as sites [2]. However, such methods also rely on a narrowness tolerance γ, which is fundamentally flawed.

3 Limited Gaps

We first define the set of *limited gaps* for a container C and a set P of polygons when P consists of congruent copies of a single polygon p. Then we give a straightforward extension of the theory to multiple shapes.

3.1 Minkowski Sum

The construction of the limited gaps is based on polygonal region set operations (union, intersection, complement) and the *Minkowski sum*. Given two polygonal regions A and B, the Minkowski sum and difference are defined,

$$A \oplus B = \{a+b \mid a \in A \text{ and } b \in B\} \qquad A \ominus B = \overline{\overline{A} \oplus B},$$

where overline denotes the set complement. These are also called the *dilation* and *erosion* of A by B.

If we designate $B+t$ to be the region B translated by vector t, then A intersects $B+t$ if and only if $t \in A \oplus -B$. Here $-B$ represents the "opposite" of B, $-B = \{-b \mid b \in B\}$. Similarly, the set of translations t such that $B+t$ is entirely inside A is $A \ominus -B$.

We have developed and implemented robust and efficient algorithms for set operations and Minkowski sums of polygonal regions.

3.2 Valid Regions and Reachable Regions

We start with a container C which is the polygonal region resulting from subtracting a set of placed polygons from a rectangle of material. The set $V_p = C \ominus (-p)$ is the *valid region* for p in C: the set of translations that place p inside C. The set $R_p = V_p \oplus p$ is the *reachable region* of p inside C: the set of points that can be covered by some translated copy of p inside C.

A gap set G_R consisting of the connected components of R_p has the *containment property* (Section 1.2), and the components have the added advantage of being non-overlapping. Unfortunately, components of R_p can contain arbitrarily narrow necks. Ideally, each component of R_p is generated by a single component of V_p. In fact, dilation by p can blend together two components of V_p for cases in which p cannot fit through a narrow neck of C but two copies of p can touch if they reach into the neck from different sides (see Figure 3).

By dilating each individual component of V_p we can create a gap set G_V with higher cardinality than G_R and with no narrow necks. Unfortunately, these gaps can be highly overlapping. The *limited gap* set will be somewhere "in between" G_V and G_R.

One advantage of R_p is that it provides a nontrivial upper bound on the efficiency[4] of a layout with respect to an item, as follows. The efficiency of packing copies of p in C cannot exceed the ratio of area(R_p) to area(C). Most packing density bounds in the literature apply only when the polygons of P are convex.

The area of R_p can also be used as a criterion for judging the placement of obstacles: larger R_p is better. Thus, the area of R_p can also be used in a greedy layout algorithm in order to choose a placement position for each piece which results in the most usable space for the remaining unplaced pieces.

3.3 Defining Limited Gaps using Probability of Interference

Let $v, v' \subseteq V_p$ be two connected components. If $v \oplus p$ and $v' \oplus p$ do not intersect, then we can independently place copies of p into C using translations from v and v' without interference. If $(v \oplus p) \cap (v' \oplus p) \neq \emptyset$, then we need to compute a probability Prob(v, v') that $(p + t) \cap (p + t') \neq \emptyset$. We assume a uniform distribution on the connected components of V_p, so $t \in v$ and $t' \in v'$ are chosen uniformly. Assuming we can compute Prob(v, v'), we can finally define limited gaps. Given a probability π, components v and v' *interfere* with each other if Prob$(v, v') \geq \pi$. Partition the components of V_p according to the transitive closure of the interference relation into *interference sets*. Finally, dilate each interference set by p to generate a gap. The resulting set G_π of gaps is the *limited gap set* for probability π. There are variations on this definition, but the point is that one can select the probability π of interference for the gap set.

We calculate Prob(v, v') as follows. For each $t \in v$, the set of $t' \in v'$ such that $(p + t) \cap (p + t') \neq \emptyset$ is $((p+t) \oplus (-p)) \cap v'$, and the probability that this choice of t results in interference is Prob$(\{t\}, v') = $ area$(((p + t) \oplus (-p)) \cap v')/$area$(v')$. The integral of Prob$(\{t\}, v')$ over all $t \in v$ gives us Prob(v, v').

We do not have an exact algorithm for this integral, but we can approximate it by sampling $t \in v$. We can improve the efficiency by sampling only the region $v \cap ((v' \oplus p) \oplus -p)$ that can result in a non-zero value of Prob$(\{t\}, v')$. After computing the average value of this probability, we normalize by multiplying it by area$(v \cap ((v' \oplus p) \oplus -p))/$area$(v)$.

Finally, if P contains multiple shapes, then we can similarly define and compute Prob(v, v') for $v \subseteq V_p$ and $v' \subseteq V_{p'}$. Let V_P be the set of all valid region components. We partition V_P according to interference. For each partition set $\Pi \subseteq V_P$, we dilate each component $v \in \Pi$ by the appropriate $p \in P$, and then take the union $\cup_{v \in \Pi}(v \oplus p)$ to generate the gap corresponding to Π.

4 Efficient Computation of Limited Gaps

In order to efficiently construct the limited gaps, we want to quickly partition the set of valid region components into *interference sets*. We present two helpful partitioning techniques. The first technique involves finding a *breaking polygon*: a polygon $\mathcal{B}$ that can be translated to fit inside any $p \in P$.[5] The second technique uses the notion of *inking polygons*: q *inks* p if the reachable region of q inside p is all of p. Section 4.1 discusses the breaking polygon technique. Section 4.2 introduces the more powerful concept of inking polygons.

[4] Efficiency is the ratio of used material to total area.

[5] We use the term *breaking polygon* because the reachable region $(C \ominus -\mathcal{B}) \oplus \mathcal{B}$ "breaks" C into smaller components.

4.1 Breaking Polygons

Let us denote the breaking polygon of P by $\mathcal{B}$. To simplify the presentation, we assume that each $p \in P$ has its origin positioned so that $\mathcal{B} \subseteq p$ without translation. Since $\mathcal{B} \subseteq p$, it follows that $V_p \subseteq V_{\mathcal{B}}$.

We can use $\mathcal{B}$ to quickly group together valid region components which might intersect each other. Clearly, two components of V_p cannot intersect each other unless they are in the same component of $V_{\mathcal{B}}$. Furthermore, since each connected component of V_p is *entirely* within a connected component of $V_{\mathcal{B}}$, the inclusion test consists only of a point-in-polygon test.

Since we want to maximize the number of components of $V_{\mathcal{B}}$, we want to maximize the area of $\mathcal{B}$. Finding the polygon of maximum area which can be translated to lie inside of each $p \in P$ is an interesting open problem which we do not solve here. In practice, we find the maximum area axis-parallel rectangle (MAAPR) (see [1]) of each $p \in P$. We reposition each p so that its origin is at the center of its MAAPR. We choose $\mathcal{B}$ to be the intersection of all $p \in P$. This guarantees that $\mathcal{B}$ contains a rectangle with the minimum width and minimum height of all the MAAPRs. This, in turn, provides a lower bound on the area of the best breaking polygon.

Unfortunately, the reachable region $R_{\mathcal{B}}$ of the breaking polygon is not necessarily a superset of R_p for each $p \in P$. Consequently, we are not guaranteed that if two valid region components v_1 and $v_2 \in V_P$ are in different connected components of $V_{\mathcal{B}}$, then they do not interfere. In fact, there can be a significant amount of interference.

4.2 Inking Polygons

We say polygon q **inks** polygon p if and only if $(p \ominus (-q)) \oplus q) = p$. A set of polygons Q is an **inking set** of a set of polygons P if, for every $p \in P$, $\cup_{q \in Q}((p \ominus (-q)) \oplus q) = p$. An inking set is valuable because $R_Q = \cup_{q \in Q} R_q$ is a superset of each reachable region R_p (recall $R_q = (C \ominus q) \oplus q$, the reachable region of q in C). Therefore, for $p, p' \in P$, if $v \subseteq V_p$ and $v' \subseteq V_{p'}$ are connected components, then $v \oplus p$ and $v' \oplus p'$ can intersect (and hence Prob$(v, v') > 0$) only if they lie in the same component of R_Q. Furthermore, since $v \oplus p$ either lies entirely inside or entirely outside each component of R_Q, we can simply perform a point-in-polygon test of any point of $v \oplus p$. It is easy to generate this point without computing all of $v \oplus p$.

For an inking set to be useful, R_Q should not be too much larger than R_P. On the other hand, R_Q should be easier to compute. The first condition implies that we desire the inking set whose smallest member has maximum possible area. The second condition implies that we desire the inking set of minimum complexity (total number of vertices). The inking set of maximal area is the set P itself. The inking set of minimum complexity is a single point, but if we specify non-zero area, then this is an open question. We desire the answer to a doubly open question: of the inking sets of minimum complexity, find the one whose smallest member has maximum area. The next section gives a lower bound on the complexity of an inking set based on the notion of a *hitting set*.

4.2.1 Hitting Sets

Let a, b, c be three consecutive vertices of some polygon $p \in P$ such that angle abc opens into the interior of p. If an inking set Q inks p, then there must be a set of consecutive vertices $a'b'c'$ on some $q \in Q$ such that angle $a'b'c'$ is a subset of abc: if q is translated to place b' on top of b, then a' and c' are on or in the interior of angle abc. We say that angle $a'b'c'$ *hits* angle abc.

We can map angle abc to an interval (arc) on the unit circle by translating b to the center of the circle and taking the arc in the interior of abc. Clearly, $a'b'c'$ hits abc if and only if its interval is a subset. We refer to the set of *angles* or *intervals* of P as that set of arcs generated by all triples of consecutive vertices abc of polygons $p \in P$.

Given a set of intervals on the unit circle, a *hitting point-set* S is a set of points such that each interval contains at least one point of S in its interior. Clearly the complexity of Q is at least as great as the size of the minimum hitting point-set of the angles of P.

Theorem 4.1 *A minimal hitting point-set for n intervals on the unit circle can found in* $O(n^2)$ *time. (The total number of vertices in P is n.)*

Proof: We first give an $O(n \log n)$ time algorithm for finding the minimal hitting set for a set of intervals of the real line. Then we show how to use this algorithm as part of an $O(n^2)$ time algorithm for intervals on a circle.

The first step in the line interval algorithm is to initialize the hitting set $\mathcal{H}$ to $\emptyset$. We then sort the intervals by increasing x-coordinate of their right endpoints, and process them in this order. For each interval i, if $\mathcal{H}$ contains a point that hits i, then we do not change $\mathcal{H}$. Otherwise, we add the right endpoint[6] of i to $\mathcal{H}$.

This algorithm runs in $O(n \log n)$ time because we sort the intervals. It can be shown that each step of the algorithm yields the minimal hitting set (for the current

[6] Actually, to avoid degeneracy, we add a point just slightly to the left of the right endpoint of i.

set of intervals) whose rightmost point is furthest to the right.

Our algorithm for intervals on a circle first sorts the interval endpoints, creating a set of $2n$ event points and $2n$ "sub-intervals" (between event points). The algorithm has an outer loop which considers each of these $2n$ sub-intervals, in turn. We add a single hitting point to a sub-interval, and thus we can eliminate from consideration any interval that contains this sub-interval. We can now cut the circle anywhere in this sub-interval and then apply the line algorithm. This gives the minimal hitting set for this choice of sub-interval. After considering all $2n$ sub-intervals, we know which choice yields the minimum hitting set. We only have to sort once, so the running time is in $O(n^2)$. ∎

We can generalize the notion of hitting point-set to hitting interval-set: recall an interval of Q hits an interval of P if it is a subset. Clearly, we can replace each point in the hitting point-set by the sub-interval in which that point lies. However, this might not yield the hitting interval-set whose minimum length interval is maximized. Since we do not want "skinny" inking polygons (to avoid inking narrow necks of C), we want to maximize this minimum angle. This will also tend to maximize the minimum area of an inking polygon.

For a given arc length L, we can find the minimum hitting interval-set whose members are of length L. We can do this in $O(n^2)$ time using a modification of the point-set algorithm. The algorithm on the line is exactly the same, but "unraveling" the circle into a line becomes more intricate. We do not give the details here. For any given hitting set size h, we can do binary search on the value of L to find the smallest value of L whose minimum hitting interval-set is h or greater. Since L can take on at most $2n(2n-1)$ values, we only have to run the $O(n^2)$ hitting interval-set algorithm $O(\lg n)$ times.

4.2.2 Forming Inking Sets using Hitting Sets

Given the minimal sized hitting interval-set Q of h intervals for P (with maximum minimum length L), we show here how to construct an inking set consisting of a set of h isosceles triangles and a single square. The isosceles triangles ink regions near the vertices of P, and the rectangle inks the remaining area. If $h_{\min}$ is the minimum complexity of the inking set, then this algorithm yields an inking set of complexity $3h_{\min}+4$.

We first discuss the isosceles triangles. We associate an isosceles triangle with each hitting set interval i as follows. Consider the set S_i of angles abc of polygons $p \in P$ which are hit by i. For each angle abc, find the maximum height of a "corner" isosceles triangle whose top is b and whose sides are aligned with ba and bc such that the triangle is in the interior of p. Let H_i be the minimum height over all vertices in S_i. Now, form an inking set polygon T_i which is the isosceles triangle of height $H_i/2$ and top angle determined by i. T_i can ink a "corner" isosceles triangle for abc of height $H_i/2$.

The size of the square s is constrained by two factors: 1) the narrowest neck in P, and 2) the narrowest (and most diagonal) angle in P. To treat the first constraint, we build a Voronoi diagram of each polygon in P, under the L_1 metric [3]. If we surround each site with the largest square which fits, then s must be smaller than the smallest of these squares. Let w_1 be the resulting width. The second constraint requires that we calculate $w_2 = \min(H_i \sin(\alpha/2)/\sqrt{2})$ over all abc in P with angle α and hitting interval i. The width of s is $\min(w_1, w_2)$.

5 Future Work/Open Problems

Currently, we approximate the area integral of $\mathrm{Prob}(\{t\}, v')$ for $t \in v$ via sampling (Section 3.3). Find efficient algorithms for computing this integral exactly.

Given a collection of polygons P, find the translation t_p for each $p \in P$ such that the area of $\cap_{p \in P}(p+t_p)$ is maximized. This would yield the maximum area breaking polygon (Section 4.1).

Section 4.2 gives several open problems relating to inking sets.

6 Acknowledgements

The authors acknowledge helpful suggestions made by Murray Daniels, and programming assistance provided by Harvard undergraduate Shiv Chatterjee.

References

[1] K. Daniels, V. Milenkovic, and D. Roth. Finding the maximum area axis-parallel rectangle in a polygon. In *Proceedings of the Fifth Canadian Conference on Computational Geometry*, pages 322–327, 1993.

[2] S. Fortune. A Sweepline Algorithm for Voronoi Diagrams. *Algorithmica*, 2:153–174, 1987.

[3] D.T. Lee. Two-dimensional Voronoi diagrams in the L_p Metric. *Journal of the ACM*, 27:604–618, 1980.

[4] Z. Li and V. Milenkovic. The Complexity of the Compaction Problem. In *Proceedings of the 5th Canadian Conference on Computational Geometry*, pages 7–11, Waterloo, Canada, 1993.

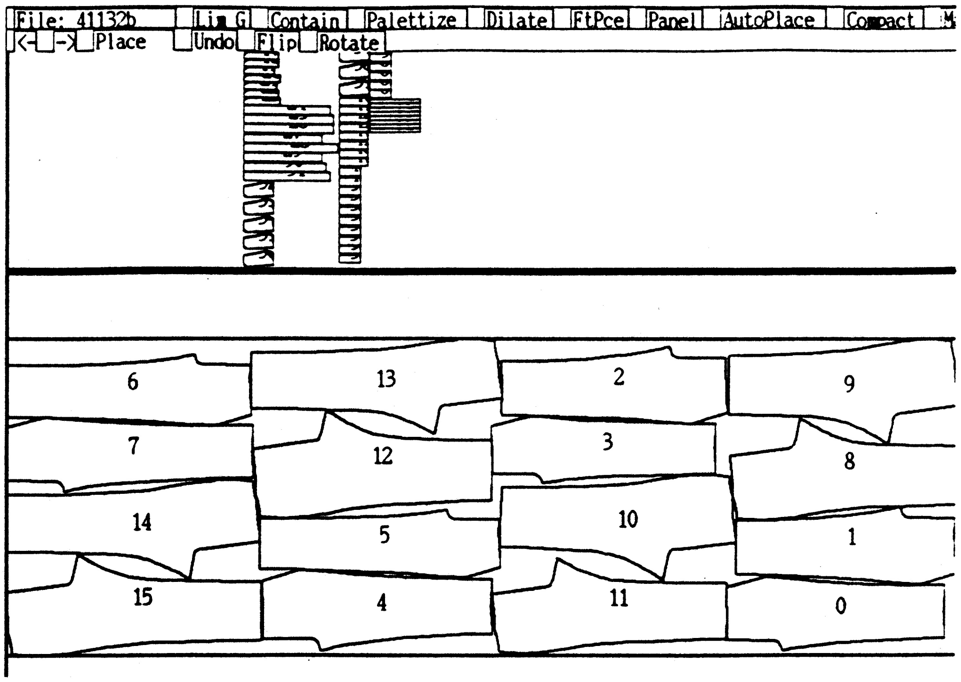

Figure 1: Trim Placement Task

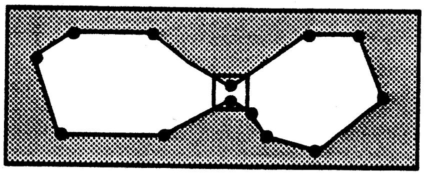

Figure 2: A Partitioning Square

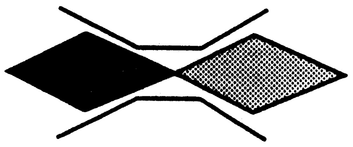

Figure 3: A neck which is usable although piece cannot pass through it

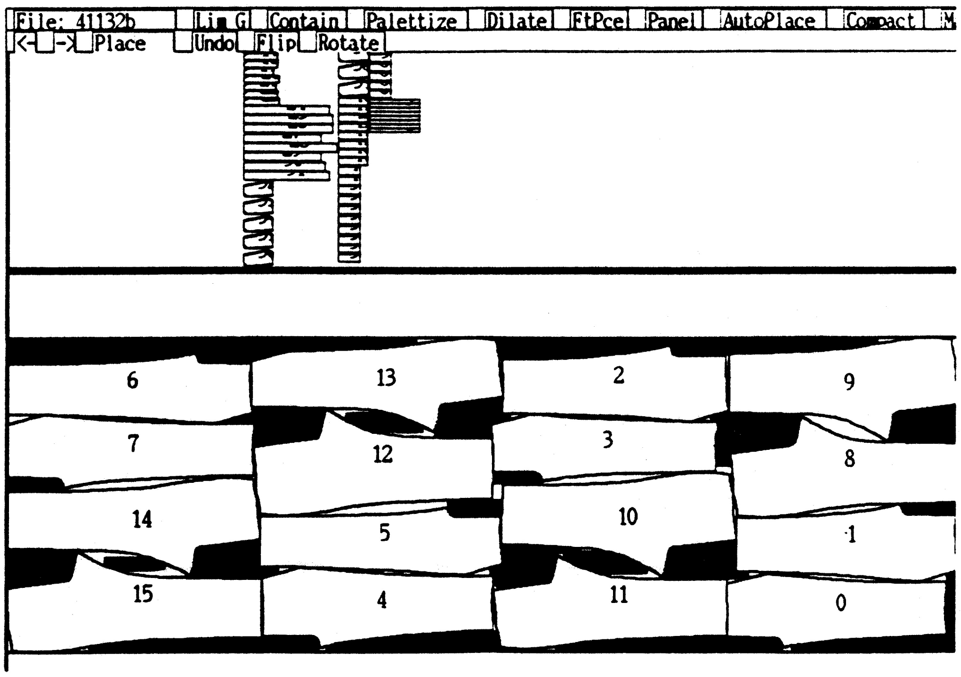

Figure 4: Limited Gaps Example