Proceedings of the First ACM Workshop on Applied Computational Geometry (WACG), edited by Ming C. Lin and Dinesh Manocha, May 27–28, 1996, pp. 33–38. (31 papers submitted, 12 accepted.) Proceedings to appear as *Lecture Notes in Computer Science*, Springer-Verlag.

Column-Based Strip Packing using Ordered and Compliant Containment

Karen Daniels[⋆1] and Victor J. Milenkovic[⋆⋆2]

[1] Harvard University and University of Miami
[2] University of Miami

Abstract. The *oriented strip packing* problem is very important to manufacturing industries: given a strip of fixed width and a set of many (> 100) nonconvex polygons with 1, 2, 4, or 8 orientations permitted for each polygon, find a set of translations and orientations for the polygons that places them without overlapping into the strip of minimum length. Heuristics are given for two versions of strip packing: 1) translation-only and 2) oriented. The first heuristic uses an algorithm we have previously developed for translational containment: given polygons $P_1, P_2, \ldots, P_k$ and a fixed container C, find translations for the polygons that place them into C without overlapping. The containment algorithm is practical for $k \leq 10$. Two new containment algorithms are presented for use in the second packing heuristic. The first, an *ordered containment* algorithm, solves containment in time which is only linear in k when the polygons are a) "long" with respect to one dimension of the container and b) ordered with respect to the other dimension. The second algorithm solves *compliant containment*: given polygons $P_1, P_2, \ldots, P_{l+k}$ and a container C such that polygons $P_1, P_2, \ldots, P_l$ are already placed into C, find translations for $P_{l+1}, P_{l+2}, \ldots, P_{l+k}$ and a nonoverlapping translational *motion* of $P_1, P_2, \ldots, P_l$ that allows all $l + k$ polygons to fit into the container without overlapping.
The performance of the heuristics is compared to the performance of commercial software and/or human experts. The results demonstrate

⋆ Harvard University, Division of Applied Sciences, and University of Miami. Email: daniels@das.harvard.edu. This research was funded by the Textile/Clothing Technology Corporation from funds awarded by the Alfred P. Sloan Foundation and by NSF grants CCR-91-157993 and CCR-90-09272.

⋆⋆ University of Miami, Department of Math and Computer Science. Email: vjm@cs.miami.edu. This research was funded by the Textile/Clothing Technology Corporation from funds awarded to them by the Alfred P. Sloan Foundation, by NSF grant CCR-91-157993, and by a subcontract of a National Textile Center grant to Auburn University, Department of Consumer Affairs.

that fast containment algorithms for modest values of k ($k \leq 10$) are very useful in the development of heuristics for oriented strip packing of many ($k >> 10$) polygons.

1 Introduction

Many industries fabricate parts by cutting them from stock material. Often, stock material comes in rolls or sheets of fixed width and indeterminate length. Such industries need to solve the *strip packing* problem: pack a set of polygons onto a strip of fixed width and minimum length. If the material has a grain then usually each polygon is permitted only a fixed number (1, 2, 4, or 8) of orientations. Figure 1 shows an example of oriented strip packing from the apparel industry. Each part has four possible orientations except for the rectangles which have either one or two.[3] These layouts can have hundreds of parts, and oriented strip packing is NP-complete. Most layouts are currently done by a human using a CAD/CAM system, and the labor cost and the cost of wasted material are considerable.

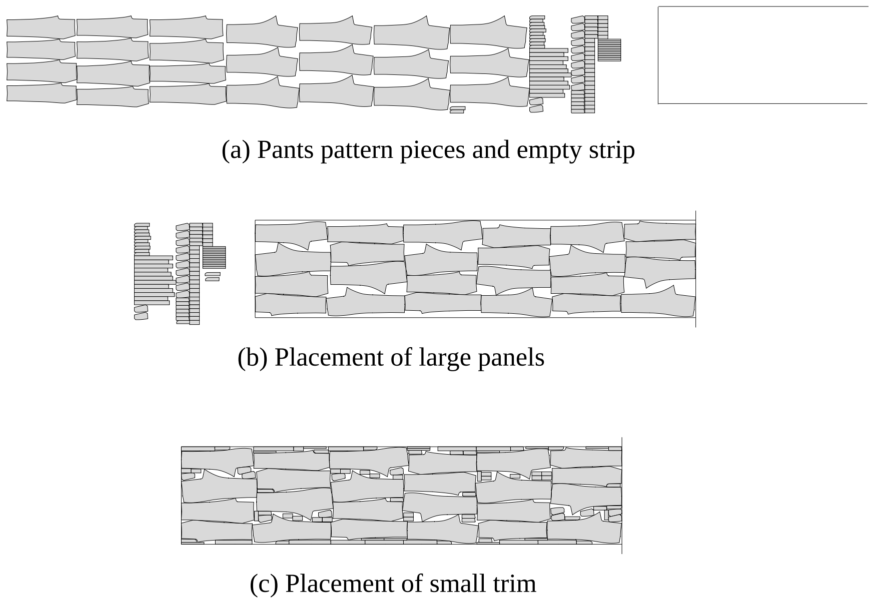


Fig. 1. An oriented strip packing task from the apparel industry

Recently we have developed fast algorithms for translational *containment*

[3] For some parts, a small *tilt* is also permitted (typically up to 3 degrees). However, we deal with tilt in a post-processing *compaction* [11, 18, 12] phase.

[1, 16, 2, 13, 4, 3, 15]: given nonconvex polygons $P_1, P_2, \ldots, P_k$ and a nonconvex polygonal container C, find translations for the polygons that pack them into C. For a modest number of polygons ($k \leq 10$), our containment algorithms find a solution or determine that none exists in a few minutes on a desk top workstation. Unless P = NP, one probably cannot hope to generalize this algorithm in a way that will be practical for $k > 100$ and/or multiple orientations. However, it is our contention that a fast translational containment algorithm for modest k is a very useful tool for the development of good heuristics for oriented strip packing.

This paper gives heuristics for strip packing which are based on containment and are domain-independent. We first give a heuristic for translation-only strip packing. This heuristic is based on packing one column at a time. We found that to generalize it to oriented strip packing, we needed two extensions of our containment algorithm. The first, *ordered containment*, solves containment very quickly (running time depends only linearly on k) when the polygons are "long" with respect to one dimension of the container (the column) and ordered with respect to the other dimension. The second, *compliant containment*, packs k additional polygons into a container already containing l polygons. The l polygons are permitted nonoverlapping motion but are forbidden to "jump" over each other. Using these two new containment algorithms, we give a heuristic for oriented strip packing.

1.1 Related Work

Strip packing and other forms of layout are well studied problems, and we refer the reader to surveys [1, 5, 6, 22]. The idea of column-based strip packing goes back to the 1960s. Gurel [9, 8, 10] used a graph-theoretical approach to place pieces in columns. In industrial practice, columns are sometimes referred to as "breaks". Our work on translational containment is described in [1, 16, 2, 13, 4, 3, 15] (see [1] for a survey of containment research).

Figure 1a illustrates the pieces required to manufacture twelve pairs of pants. In earlier work, we use a column-based method to place the large panel pieces (Figure 1b) [17], and then a containment-based method to place the smaller trim pieces[4] (Figure 1c) [1]. These methods generate pants layouts within the same production quality range as that imposed on human experts. The panel placement heuristic packs each column with the most "difficult" set of remaining panels. By assuming a particular x-translation for each panel, it can check each possibility very quickly. Ordered containment (Section 3) finds column packings without knowing the x-translations, and this permits us to generalize the "worst"-first strategy to a domain-independent heuristic. (See Section 7 for further comparison of the panel placement heuristic with our new work.)

For the most part, our work has focused on algorithms, not heuristics. Part of the reason for this emphasis is that it is difficult to measure the performance

[4] Our pants layout software is part of our Automatic Marker Making Toolkit, which has been licensed by Gerber Garment Technologies. The toolkit also includes compaction, overlap reduction, and containment software.

of heuristics objectively. However, recently a large domestic auto manufacturer challenged us with a large ($k = 50$) translation-only strip packing problem. Using our containment algorithm and a simple column-based heuristic (Section 2.3), we generated a solution with 83.94% cloth utilization.[5] By comparison, a naive first-fit strategy only achieved 81.71% utilization. The best commercial software achieved 85%, and the best human performance was 86.5%.[6]

1.2 Outline

The paper is organized as follows. Section 2 gives definitions and background relevant to containment, it presents the salient parts of our current containment algorithm, and then it describes a simple column-based heuristic for translational strip packing. Section 3 discusses ordered containment. Section 4 introduces compliant containment. Section 5 gives our column-based strip packing heuristic for polygons with multiple orientations, which relies on both ordered and compliant containment. Section 6 describes our implementation and gives results. Section 7 discusses future work.

2 Definitions, Background, and Translational Strip Packing

2.1 Definitions

The *Minkowski sum* [19, 7, 20, 21] of two point-sets (of $\mathbf{R}^2$ in the case of this paper) is defined

$$A \oplus B = \{a + b \,|\, a \in A,\, b \in B\}.$$

For a point-set A, let $\overline{A}$ denote the set complement of A and define $-A = \{-a \,|\, a \in A\}$. For a vector t, define $A + t = \{a + t \,|\, a \in A\}$. Note that $A + t = A \oplus \{t\}$.

A containment algorithm translates k polygonal regions $P_1, P_2, \ldots, P_k$ into a polygonal container C without overlap. If we denote $P_0 = \overline{C}$ to be the complement of the container region, then containment is equivalent to the *placement* of $k+1$ polygons $P_0, P_1, P_2, \ldots, P_k$ in nonoverlapping positions. For translations t_i and t_j, $P_i + t_i$ and $P_j + t_j$ do not overlap if and only if $t_j - t_i \in U_{ij}$, where

$$U_{ij} = \overline{P_i \oplus -P_j}, \qquad 0 \leq i, j \leq k, i \neq j. \tag{1}$$

The set U_{ij} is the two-dimensional *configuration space* for placing P_j with respect to P_i. Clearly, $U_{ij} = -U_{ji}$. A *valid configuration* $\langle t_0 = (0,0), t_1, t_2, \ldots, t_k \rangle$ of $P_1, P_2, \ldots, P_k$ satisfies

$$t_j - t_i \in U_{ij}, \qquad 0 \leq i < j \leq k. \tag{2}$$

[5] Cloth utilization is the ratio of the sum of the areas of the placed polygons to the area of the strip in which they are placed, expressed as a percentage.

[6] These two results were for a slightly harder problem: the pieces had small buffer zones, and the cutting rules disallowed some types of piece contacts. To keep the challenge challenging, the company has asked us not to reveal the dataset.

We set $t_0 = (0, 0)$ because the container is not permitted to move. For the set of all valid configurations, we denote the range of $t_j - t_i$ by U^*_{ij}.

2.2 Containment Algorithm

Our containment algorithm [16] takes as input a *hypothesis* $\mathcal{U} = \langle U_{ij}, 0 \leq i < j \leq k\rangle$. Equation 1 gives the initial values of the U_{ij}. (We sometimes denote these by U^{init}_{ij}.) The containment algorithm applies *restriction*, *evaluation*, and *subdivision* to the hypothesis. *Valid restriction* replaces some or all of the U_{ij} by subsets of themselves without losing any valid configurations for the hypothesis (Equation 2). *Evaluation* attempts to find at least one valid configuration for a hypothesis. *Subdivision* splits the hypothesis into two subhypotheses. The containment algorithm applies these three operations until it finds a solution or until all hypotheses are restricted to the empty set.

For purposes of this paper, we only need to describe restriction and subdivision. *Geometric restriction* replaces U_{ij} by $U_{ij} \cap (U_{ih} \oplus U_{hj})$, $0 \leq h \neq i \neq j \leq k$. *Linear programming (LP) restriction* computes the range R_{gh} of $t_h - t_g$ such that t_g and t_h are part of a valid configuration satisfying $t_j - t_i \in \mathrm{CH}(U_{ij})$, $0 \leq i < j \leq k$, where $\mathrm{CH}(U_{ij})$ is the convex hull of U_{ij}. It replaces U_{gh} by $U_{gh} \cap R_{gh}$. Our algorithms repeatedly apply restriction until a steady state is reached. Both geometric and LP restriction are valid restrictions.

Subdivision selects one U_{ij} in $\mathcal{U}$, splits U_{ij} into U'_{ij} and U''_{ij}, and creates subhypotheses $\mathcal{U}'$ and $\mathcal{U}''$ by substituting either U'_{ij} or U''_{ij} for U_{ij}. If U_{ij} has multiple components, subdivision splits off one component. If not, it cuts U_{ij} with a line which is chosen using a *distance-based subdivision* technique [1, 16]. Distance-based subdivision ultimately cuts each U_{ij} into convex regions. If all U_{ij} in the current hypothesis $\mathcal{U}$ are convex, then the output of LP restriction is U^*_{ij} (the range of $t_j - t_i$ with respect to the valid configurations of $\mathcal{U}$).

2.3 Translational Strip Packing Heuristic

Given our containment algorithm, we can define a simple column-based translational strip packing heuristic. Suppose we are only willing to run our containment algorithm with $k \leq 5$ (since we have to call it many times). Order the polygons by diminishing length (x-dimension). To start a new column, set its length to that of the longest unplaced piece. Repeatedly attempt to place the next available polygon into the column using the procedure given below. If some polygon does not fit, skip to the next one. If no more polygons fit into a column, start a new column. The heuristic is column-greedy: once a column is placed, that column is never removed from the layout.

To place a polygon into the current column, *remove* the last four polygons placed in the column (or all of them if there are less than four) and then run containment to place them back in along with the new polygon. This is the heuristic we used on the auto manufacturer's challenge problem. The four steps back/five steps forward strategy permits the algorithm to correct "mistakes" in the placement of the last few polygons. This strategy generates a much tighter packing than a greedy one-polygon-at-a-time strategy.

3 Ordered Containment

Our column-based strip packing algorithm repeatedly solves containment for a column and a set of polygons to place in the column. Polygons are either "short" or "long" in the x-direction. "Short" polygons can be placed side by side in the column. "Long" polygons must be stacked in the column and cannot "pass" each other. Once the "long" polygons are placed, they have a particular *ordering* in the y-direction.

This section gives a rigorous definition of an *ordering* and a *strong ordering* for a set of polygons in a container. It then gives an *ordered containment* algorithm that finds a solution to containment for a strong ordering or reports that none exists. The running time of the algorithm depends on the complexity of the polygons, but it depends only linearly on k.

Using ordered containment, we can rapidly find ways of packing sets of "long" polygons into the container. We pack the short polygons using *compliant containment* which is described in Section 4.

3.1 Orderability

For $1 \leq i \neq j \leq k$, define $U_{ij}(C) = U_{ij}^{\text{init}} \cap (U_{i0}^{\text{init}} \oplus U_{0j}^{\text{init}})$ which is U_{ij} after (geometric) restriction with respect to the container. We define P_i and P_j to be *orderable* ("long") when $U_{ij}(C)$ has at least two connected components $U_{ij}^{+}(C)$ and $U_{ij}^{-}(C)$ corresponding to the placement of P_j above or below P_i, respectively. If $U_{ij}(C)$ has more than two components, we can add the extra components to whichever of $U_{ij}^{+}(C)$ or $U_{ij}^{-}(C)$ is nearest. By definition, $U_{ij}^{+}(C) = U_{ji}^{-}(C)$. In practice, we only consider P_i and P_j to be orderable if we can partition the components using a horizontal line. The component(s) above the line are $U_{ij}^{+}(C)$ and the component(s) below the line are $U_{ij}^{-}(C)$.

An ordering is *strong* if for every ordered triple h, i, j such that P_i is between P_h and P_j, P_h cannot overlap P_j: $U_{hi}^{+}(C) \oplus U_{ij}^{+}(C) \subseteq U_{hj}^{+}(C)$.

3.2 Ordered Containment Algorithm

Suppose w.l.o.g. that $P_1, P_2, \ldots, P_k$ are strongly ordered in order of increasing index with respect to C. It takes $\mathrm{O}(k^3)$ time to verify that an input ordering is strong (Section 6 describes a way to avoid this cost). This section gives an algorithm for solving containment for this ordering in time which is only linear in k. (Of course, it also depends on the number of vertices of $P_1, P_2, \ldots, P_k$ and C.) We begin by showing how to generate U_{0j}^{*}, $1 \leq j \leq k$, with respect to the ordering, in time which is only linear in k.

Lemma 1. *Given strongly ordered $P_1, P_2, \ldots, P_k$, U_{0j}^{*}, $1 \leq j \leq k$, with respect to that ordering can be generated in time which is, as a function of k, $\mathrm{O}(k)$.*

Proof. Consider the following three loops, whose combined running time, as a function of k, is $\mathrm{O}(k)$:

$$\begin{array}{ll} \textbf{For } i \leftarrow 1 \textbf{ to } k, & U_{0i} \leftarrow U_{0i}^{\mathrm{init}}. \\ \textbf{For } i \leftarrow 2 \textbf{ to } k, & U_{0i} \leftarrow U_{0i} \cap (U_{0,i-1} \oplus U_{i-1,i}^{+}(C)). \\ \textbf{For } i \leftarrow k-1 \textbf{ to } 1, & U_{0i} \leftarrow U_{0i} \cap (U_{0,i+1} \oplus U_{i+1,i}^{-}(C)). \end{array}$$

Let $U_{0j}^{(2)}$ denote the value of U_{0j} after the second loop. We can prove by induction that $U_{0j}^{(2)}$ is *downwardly valid*: for each $t_j \in U_{0j}^{(2)}$, there exists a valid sub-configuration $\langle t_1, t_2, \ldots, t_j \rangle$ ($t_i \in U_{0i}$, $1 \le i \le j$, and $t_i - t_h \in U_{hi}^{+}(C)$, $1 \le h < i \le j$). This is vacuously true for $j = 1$. Suppose it is true for j. Clearly, $U_{0,j+1}^{(2)} \subseteq U_{0j}^{(2)} \oplus U_{j,j+1}^{+}(C)$. Hence, for each $t_{j+1} \in U_{0,j+1}^{(2)}$, there exists $t_j \in U_{0j}^{(2)}$ such that $t_{j+1} - t_j \in U_{j,j+1}^{+}(C)$. Choose a valid sub-configuration $\langle t_1, t_2, \ldots, t_j \rangle$. For $1 \le h < j$, $t_{j+1} - t_h = t_j - t_h + t_{j+1} - t_j \in U_{hj}^{+}(C) \oplus U_{j,j+1}^{+}(C) \subseteq U_{h,j+1}^{+}(C)$, by the definition of strong ordering. Therefore $\langle t_1, t_2, \ldots, t_{j+1} \rangle$ is a valid sub-configuration, and $U_{0,j+1}^{(2)}$ is downwardly valid. Similarly, we can define *upwardly valid* and analyze the third loop. The output U_{0j} of the algorithm is both upwardly and downwardly valid and a subset of U_{0j}^{init}. Therefore, for each $t_j \in U_{0j}$, there exists a valid configuration $\langle t_1, t_2, \ldots, t_k \rangle$. Now we argue that no configurations which are valid with respect to the strong ordering are eliminated. The geometric restriction $U_{ij} \leftarrow U_{ij} \cap (U_{ih} \oplus U_{hj})$, $0 \le h \ne i \ne j \le k$ is a valid restriction [1, 3]. For this reason, the operations $U_{0i} \leftarrow U_{0i} \cap (U_{0,i-1} \oplus U_{i-1,i}^{+}(C))$ and $U_{0i} \leftarrow U_{0i} \cap (U_{0,i+1} \oplus U_{i+1,i}^{-}(C))$ in the second and third loops, respectively, are both valid restrictions with respect to the strong ordering. Hence, the output is U_{0j}^{*}, as defined in Section 2.1, with respect to this choice of ordering.

Lemma 2. *Given strongly ordered $P_1, P_2, \ldots, P_k$, we can solve ordered containment in time which is, as a function of k, $\mathrm{O}(k)$.*

Proof. Let FIND_U_{0j}^{*} be a procedure consisting of the three loops in the proof of Lemma 1. Consider the following algorithm for a given strong ordering. Call FIND_U_{0j}^{*}. By Lemma 1, it creates U_{0j}^{*}, $1 \le j \le k$. If any $U_{0j}^{*} = \emptyset$, the algorithm reports infeasibility and terminates. Otherwise, select any j and any $t_j \in U_{0j}^{*}$. One can select positions t_i, $1 \le i \ne j \le k$, consistent with t_j, as follows:

$$\begin{array}{l} \textbf{For } i \leftarrow j+1 \textbf{ to } k, \textbf{ choose } t_i \in U_{0i}^{*} \cap (U_{i-1,i}^{+}(C) + t_{i-1}). \\ \textbf{For } i \leftarrow j-1 \textbf{ to } 1, \textbf{ choose } t_i \in U_{0i}^{*} \cap (U_{i+1,i}^{-}(C) + t_{i+1}). \end{array}$$

Lemma 1 shows that the cost of FIND_U_{0j}^{*}, as a function of k, is $\mathrm{O}(k)$, so the total cost of the algorithm as a function of k is also $\mathrm{O}(k)$. To establish that it yields a valid configuration, we first observe that polygons adjacent in the ordering are prevented from overlapping by the two operations $U_{i-1,i}^{+}(C) + t_{i-1}$ and $U_{i+1,i}^{-}(C) + t_{i+1}$. Because adjacent polygons cannot overlap and the ordering is strong, nonadjacent polygons cannot overlap. Thus, any configuration produced is valid. Now we need only show that the loops run to completion (i.e. $U^*_{0i} \cap (U^+_{i-1,i}(C) + t_{i-1}) \neq \emptyset$ and $U^*_{0i} \cap (U^-_{i+1,i}(C) + t_{i+1}) \neq \emptyset$). W.l.o.g. we discuss the first loop. The second loop of FIND_U^*_{0j} guarantees that $U_{0i} \subseteq (U_{0,i-1} \oplus U^+_{i-1,i}(C))$. By assumption, $U^*_{0,i-1}$, U^*_{0i}, and $U^+_{i-1,i}(C)$ are all nonempty, so we have: $U_{0i} \cap (U_{0,i-1} \oplus U^+_{i-1,i}(C)) \neq \emptyset$. Now, by the definition of $U^*_{0,i-1}$, for any $t_{i-1} \in U^*_{0,i-1}$, there exists $t_i \in U^*_{0,i}$ such that t_{i-1} and t_i are part of a valid configuration. We therefore have $t_i - t_{i-1} \in U^+_{i-1,i}(C)$. As $t_{i-1} + t_i - t_{i-1} = t_i$, we obtain $t_i \in U_{0,i-1} \oplus U^+_{i-1,i}(C)$, which establishes that $U^*_{0i} \cap (U^+_{i-1,i}(C) + t_{i-1}) \neq \emptyset$.

4 Compliant Containment

In a *compliant containment* problem, there are l polygons $P_1, P_2, \ldots, P_l$ placed in a container C and there are k additional polygons $P_{l+1}, P_{l+2}, \ldots, P_{l+k}$ which we also wish to place into the container. The polygons in the container are permitted to move but not "jump" over each other. The new polygons can be placed anywhere. This section gives a theoretical algorithm and a practical algorithm for compliant containment.

4.1 Compliant Containment Algorithm

The compliant containment algorithm sets up a containment problem for $P_1, P_2, \ldots P_{l+k}$. However, in the initial hypothesis, it replaces U^{init}_{ij} by $U^{\text{compliant}}_{ij} \subseteq U^{\text{init}}_{ij}$, $0 \leq i < j \leq l$. This restricts the motion of polygons $P_1, P_2, \ldots, P_l$. Then it runs the (ordinary) containment algorithm on this modified hypothesis.

If $P_1, P_2, \ldots, P_l$ are placed by the ordered containment algorithm of the previous section, then Lemma 1 shows how to obtain U^*_{0j}, $1 \leq j \leq l$, with respect to a given strong ordering. In this case, we can set $U^{\text{compliant}}_{ij}$, $1 \leq i < j \leq l$, equal to the following subset of U^{init}_{ij}, (which is a superset of U^*_{ij}): $U^{\text{init}}_{ij} \cap (U^*_{i0} \oplus U^*_{0j})$.

In general, the containment algorithm (Section 2.2) can generate a convex cover of the set of valid placements of $P_1, P_2, \ldots, P_l$ into C. Each element of the cover is a hypothesis $\langle U_{ij}, 0 \leq i < j \leq l\rangle$ with all U_{ij} convex. The current placement $\langle t^0_1, t^0_2, \ldots, t^0_l\rangle$ of $P_1, P_2, \ldots, P_l$ (the input to compliant containment) belongs to exactly one of these ($t^0_j - t^0_i \in U_{ij}$, $0 \leq i < j \leq l$), but others may cover the same connected component of valid solutions (the set of solutions reachable by a nonoverlapping motion of the polygons). For each of these hypotheses, the compliant containment algorithm substitutes the convex U_{ij} in place of U^{init}_{ij}, $0 \leq i < j \leq l$, and then solves containment for $P_1, P_2, \ldots, P_{l+k}$.

4.2 Practical Compliant Containment

The general compliant containment algorithm must solve many restricted containment problems. In practice, we consider only a single list $\langle U^{\text{compliant}}_{ij}, 0 \leq i < j \leq l\rangle$, where $U^{\text{compliant}}_{ij}$ is a superset of the union of the convex cover. This list may correspond to more than one connected component in the space of valid solutions. However, for our purposes, there is no harm in generating excess solutions to containment (except for the extra time we use to generate them). Define $U^*_{ij}(1 \ldots l)$ to be the range of values of $t_j - t_i$ for all valid placements of $P_1, P_2, \ldots, P_l$ into C. For $0 \leq i < j \leq l$, we set $U^{\text{compliant}}_{ij}$ equal to the connected component of $U^*_{ij}(1 \ldots l)$ which contains $t^0_j - t^0_i$.

To truly compute $U^*_{ij}(1 \ldots l)$, we would again have to compute a convex cover of the valid placements of $P_1, P_2, \ldots, P_l$ in C. However, we can create a superset of $U^*_{ij}(1 \ldots l)$ as follows. Subdivide and restrict hypotheses for this containment problem as usual (no evaluation), but put a limit on the depth of the subdivision tree. Take the union of each U_{ij} from each "leaf" (undivided) hypothesis. We refer to this method as *subdivision restriction* (of the initial hypothesis) [14]. Subdivision restriction is a generalization of union restriction [1].

5 Oriented Strip Packing Heuristic

This section describes a column-based heuristic for oriented strip packing. Like our translational strip packing heuristic, our oriented strip packing heuristic is column-greedy. To select a length (x-dimension) for the column, it sorts the unplaced pieces in order of decreasing maximum dimension (height or length). The length of the chosen orientation of the first unplaced piece in the list determines the column length. The heuristic packs "long" pieces into the column using ordered containment, and it packs "short" pieces into the column using compliant containment. After each column is placed, it is helpful to apply leftward gravity, which is a form of compaction [11, 18, 12], to reduce fragmentation of the available space.[7]

5.1 Packing "Long" Polygons

The heuristic maintains a list of *choices*. Each *choice* has 1) a set of "long" polygons, 2) an orientation for each of these polygons, 3) a strong ordering on this set. The initial list consists of the (1, 2, 4, or 8) ways of orienting the first polygon. The width of the column for each choice is equal to the x-dimension of that orientation of the first polygon. The heuristic then creates a list of 2-polygon choices, then a list of 3-polygon choices, and so forth, until it cannot add another polygon (or none of the remaining polygons are "long"). For each list, the set of polygons in each choice is the same.

Suppose the heuristic has already created the list of 3-polygon choices. It creates the list of 4-polygon choices as follows. Let P be the next unplaced polygon. For each choice and for each legal orientation of P and for each of the (four) possible ways P can be inserted into the ordering of the three polygons

[7] Using repeated solutions to linear programs, it is possible to plan a nonoverlapping motion of a set of polygon that minimizes any linear potential function on the positions of the polygons. This process is called *compaction* [11, 18, 12]. *Leftward gravity* minimizes the sum of the x-coordinates of polygon positions.

of the choice, the heuristic determines if P can be inserted there using ordered containment. Out of all the 4-polygon choices it generates in this manner, it keeps the best N for each different column length, where we measure goodness by the amount of vertical freedom: the height of the U^*_{01} region, where P_1 is the first polygon in the ordering with respect to C. If no 4-polygon choices are generated, we try the next unplaced polygon.

5.2 Packing "Short" Polygons

After the heuristic has packed as many "long" polygons as possible, each of the "long" polygons is allowed only compliant motion within its corresponding U^*_{0i}. The heuristic continues to generate lists of choices by adding one piece at a time. Let P_i be the next unplaced piece. For each choice in the list of choices and for each legal orientation of P_i, the heuristic restricts (Section 2.2) the initial hypothesis of the compliant containment problem. The connected components of U_{0i} are sorted in order of increasing area. For each component of the restricted U_{0i}, in order, the algorithm solves the compliant containment problem with P_i restricted to that component until either P_i fits or all the components of U_{0i} have been examined. The new list of choices is the N best successful outcomes. If there are no successful solutions to compliant containment, the heuristic skips to the next unplaced piece. After P_i is added, it is restricted to compliant motion. Along with each new choice, we store the current valid position and set $U^{\mathrm{compliant}}_{0i}$ of valid positions of P_i. We store U^*_{0i} for each "long" P_i and the selected component of U_{0i} for each "short" P_i.

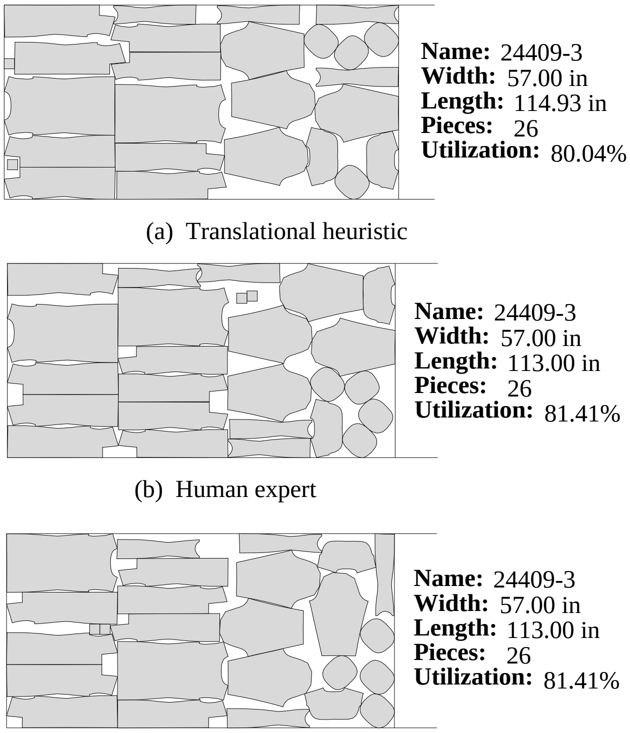
6 Implementation and Results

We have implemented our translational strip packing heuristic. As mentioned in Section 1.1, this heuristic achieved a cloth utilization of 83.94% for a layout of 50 polygons given to us by a large automotive manufacturer. The cloth utilization achieved by the best automatic layout system in this manufacturer's competition was 85%. The dataset for this example is proprietary and is therefore not shown in this paper.[8]

Figure 2a shows a smaller example of the translational heuristic applied to a 26-polygon layout from a large apparel manufacturer. This example was generated in less than one hour on a 50 MHz SPARCstation.[9] The cloth utilization of this example is 80.04% and leftward compaction does not improve it. Each polygon has up to 8 possible orientations. We supplied the translational heuristic with the same polygon orientations chosen by a human expert. The production-quality layout produced by a human expert has cloth utilization of 81.40%. Leftward compaction improves this to 81.41% (see Figure 2b).

[8] Information on how to contact the automotive manufacturer will be supplied by the authors upon request.

[9] SPARCstation is a trademark of SPARC, International, Inc., licensed exclusively to Sun Microsystems, Inc.



(a) Translational heuristic

(b) Human expert

(c) Oriented heuristic

Fig. 2. Compacted strip packings

We have implemented ordered containment and "practical" compliant containment, as well as our oriented strip packing heuristic. We currently retain the best 5 choices of orientation and strong ordering for a set of "long" polygons. It is cheaper to run the ordered containment algorithm without checking strongness and then verify the output, which is what we implemented. For any particular output configuration $\langle t_1, t_2, \ldots, t_k \rangle$, it takes $\mathrm{O}(k \log k)$ time to verify that $P_i + t_i$, $1 \leq i \leq k$, is a solution to containment.

Our current implementation of compliant containment uses only the simplest version of subdivision-restriction to determine the valid region for P_i (only restriction, no subdivision). Within compliant containment we currently limit the number of hypotheses examined to 10. (Note that a *hypothesis*, as defined in Section 2.2, is quite different from the notion of a *choice*). This reduces the running time but it occasionally prevents the algorithm from finding a position for a piece within a column. To compensate for this, after all columns have been placed we check the sparseness of the last column. If that column is too loose, we remove the column and attempt to place its pieces in previous columns using our *shrink-fit* technique.[10] This technique forms a scaled-down version of a piece and finds the nonoverlapping positions for it within the layout. It then uses overlap reduction to check if the unscaled piece fits at one of those locations.

For the 26-polygon example of Figure 2b, the layout achieved using our new algorithm is shown in Figure 2c. Note that the cloth utilization after compaction is the same as that of the compacted human-generated layout. Our result was produced in approximately 4 hours on a 50 MHz SPARCstation. No shrink-fit postprocessing was used. It might appear that the 81.41% cloth utilization obtained by both the human and our new algorithm is optimal for this dataset. One way to test this is to remove part of the end of the layout, guess a shorter length, and then run our ordinary translational containment algorithm using the shorter container. (In fact, if k is small, one can find the optimal strip length for a translational strip packing problem via binary search, as Daniels showed in her Ph.D. thesis [1].) For our 26-polygon example, we removed the last two columns (13 pieces) of the layout generated by our translational heuristic (Figure 2a) and then generated a "super-human" layout by starting with the length of the compacted human-generated layout as a goal, successfully solving containment, and then compacting. The result has cloth utilization of 81.60% and is shown in Figure 3.

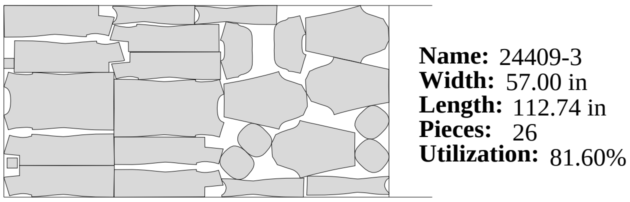


Fig. 3. Compacted result of translational heuristic followed by translational containment on pieces from last 2 columns

KGR, a manufacturer of women's apparel in Lawrence, Massachusetts, supplied us with two sets of production markers (layouts) created by human experts with at least ten years of experience. Table 1 and Table 2 list cloth utilization from the first and second test sets, respectively. The running time of our algorithm on these examples is approximately one hour per column, which is roughly five times slower than the human.[11]

For the first set of markers, our algorithm matched or exceeded human cloth utilization on 11 of 19 examples (without shrink-fit post-processing for a sparse last column, we achieved this for only 5 of 19). Figure 4 shows one such "super-human" example for 23 items. For markers in the first set, the KGR marker

[10] The *shrink-fit* technique is part of our Automatic Marker Making Toolkit.

[11] With increases in the speed of the underlying polygon operations, we expect that the running time of our prototype software will improve in the near future.

Table 1. First KGR test suite: cloth utilization (in %) of compacted layouts produced by expert KGR marker makers vs. ordered and compliant containment heuristic vs. heuristic followed by shrink-fit post-processing

Example	k	KGR	Heuristic	Heuristic+
54303-dn	8	75.99	76.32	
54303-cn	8	71.60	73.59	
54303-8q	8	79.30	79.30	
54303-en	16	84.71	84.21	
54303-pn	16	83.95	84.14	
72631-4q	16	81.90	77.75	77.75
72631-3m	16	81.09	77.70	
72631-tn	16	81.76	81.24	
14349-3j	18	84.85	79.96	
14349-fi	18	85.18	75.22	83.41
14349-wm	18	81.14	65.28	
34045-xq	18	73.66	73.50	75.61
34045-ym	18	73.84	73.41	75.49
34045-zu	18	73.98	71.29	74.51
74375-fn	23	78.45	77.44	77.87
74375-ru	23	79.33	75.96	80.06
54303-5m	24	76.80	76.89	
54303-6q	24	82.54	79.97	82.85
54303-7u	24	85.66	79.40	85.68

Table 2. Second KGR test suite: cloth utilization (in %) of compacted layouts produced by expert KGR marker makers vs. ordered and compliant containment heuristic followed by shrink-fit post-processing

Example	k	KGR	Heuristic+
797748MF1A7C	10	72.33	71.85
74373MF1A3C	10	67.87	67.87
74373ML1A7A	14	78.25	72.16
74373ML1A3A	14	76.14	77.85
74373MF1A7B	20	76.37	68.86
74373MS1A3C	22	73.33	75.41
74373MS1A7C	22	76.80	78.11
799948ML1A3B	30	84.12	79.93

makers sometimes constrain the legal orientations of the polygons beyond what is given in the data files in order to increase the quality of a garment. Without these additional constraints, it is possible that they might be able to improve their cloth utilization for these markers. For markers in the second set, the marker makers used only the constraints in the data files. These markers are ones which were particularly challenging for the humans. Our algorithm met or exceeded human performance on 4 of 8 examples in this set. Two examples are shown in Figure 5 and Figure 6. Figure 7 shows an example in which the human expert produces the superior layout.

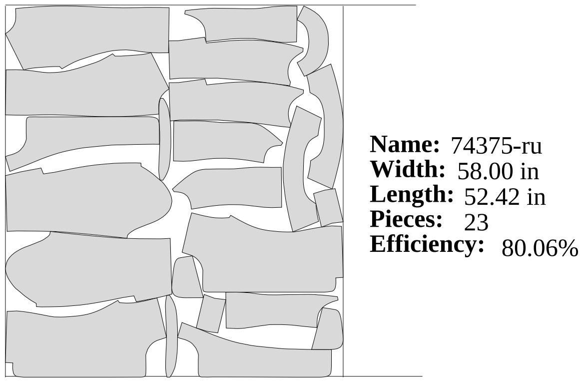


Fig. 4. Compacted result of oriented heuristic followed by shrink-fit post-processing for example 74375-ru of first KGR test suite

7 Future Work

Although our oriented strip packing software is still in an experimental phase, we view our new heuristic as a promising way to construct column-based layouts. In the future, we will consider adding multiple "short" polygons at the same time. Another possible improvement would allow the heuristic to "look-ahead" and assess the impact of current column choices on the packing of future columns. Our column-based pants panel layout algorithm [17] has this feature. Its look-ahead process tries to satisfy a "jaggedness" criterion for the end of the final column. We also plan to experiment with ways to reduce the column-greediness of the heuristic. One way is to "backtrack" (i.e. undo one or more columns). This feature exists in our panel layout algorithm. Alternatively, we can maintain multiple choices for each column as we place additional columns.

This work supports our thesis that good containment algorithms can lead to practical layout heuristics. It also emphasizes the value of interacting with industry. The layout from the auto industry motivated us to develop our first

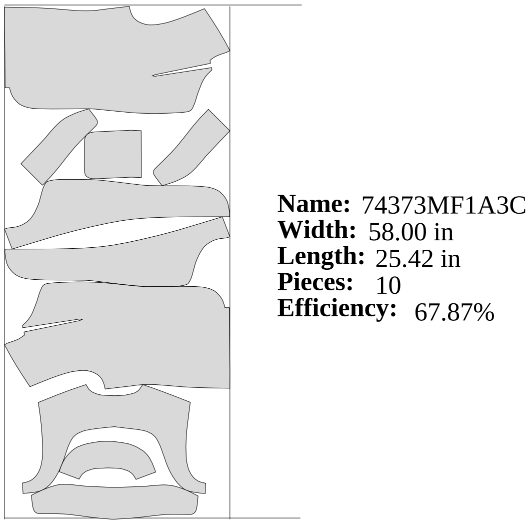


Fig. 5. Compacted result of oriented heuristic for example 74373MF1A3C of second KGR test suite

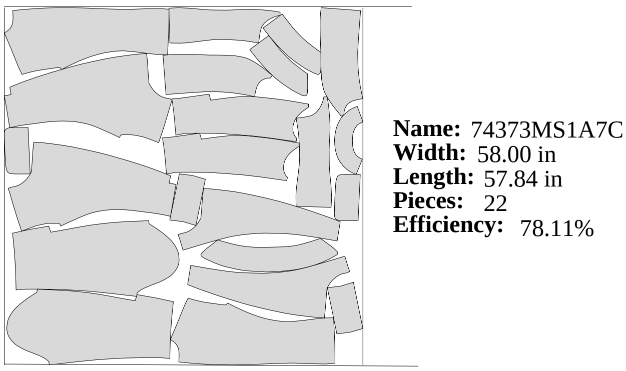
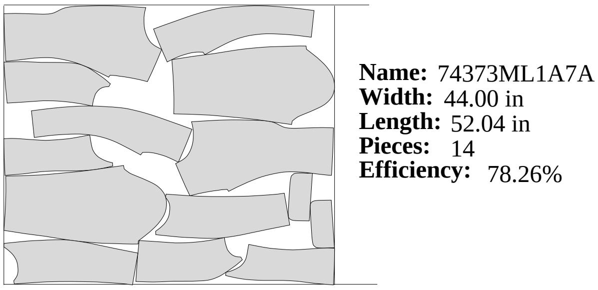


Fig. 6. Compacted result of oriented heuristic for example 74373MS1A7C of second KGR test suite

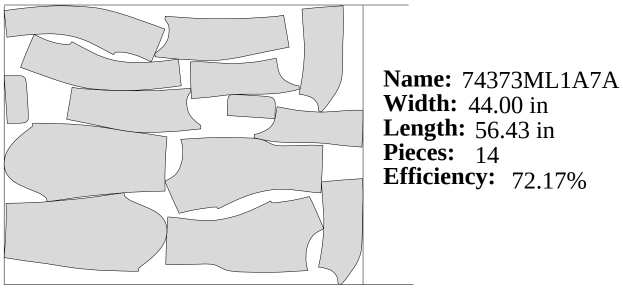
column-based heuristic. The data from KGR led us to develop the ordered containment and compliant containment algorithms and the second heuristic. Hence, our interaction with industry yielded new theory, algorithms, and heuristics. We plan to continue such fruitful interactions in the future.

References

1. K. Daniels. *Containment Algorithms for Nonconvex Polygons with Applications to Layout.* PhD thesis, Harvard University, 1995.



(a) Human expert



(b) Oriented heuristic

Fig. 7. Compacted results of human expert vs. oriented heuristic followed by shrink-fit post-processing for example 74373ML1A7A of second KGR test suite

2. K. Daniels and V. J. Milenkovic. Multiple Translational Containment, Part I: An Approximate Algorithm. *Algorithmica, special issue on Computational Geometry in Manufacturing, to appear.*
3. K. Daniels and V. J. Milenkovic. Multiple Translational Containment: Approximate and Exact Algorithms. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 205–214, 1995.
4. K. Daniels, V. J. Milenkovic, and Z. Li. Multiple Containment Methods. Technical Report 12–94, Center for Research in Computing Technology, Division of Applied Sciences, Harvard University, 1994.
5. K. A. Dowsland and W. B. Dowsland. Packing Problems. *European Journal of Operational Research*, 56:2 – 14, 1992.
6. H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operations Research*, 44:145–159, 1990.
7. L. Guibas, L. Ramshaw, and J. Stolfi. A Kinetic Framework for Computational Geometry. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, pages 100–111, 1983.
8. O. Gurel. Additional considerations on marker layout problem by graph theory. Technical Report 320-2945, IBM Scientific Centre, 1968.
9. O. Gurel. Marker layout via graph theory: An attempt for optimal layout of irregular patterns. Technical Report 320-2921, IBM Scientific Centre, 1968.

10. O. Gurel. Circular graph of marker layout. Technical Report 320-2965, IBM Scientific Centre, 1969.
11. Z. Li. *Compaction Algorithms for Non-Convex Polygons and Their Applications.* PhD thesis, Harvard University, Division of Applied Sciences, 1994.
12. Z. Li and V. J. Milenkovic. A Compaction Algorithm for Non-Convex Polygons and Its Application. In *Proceedings of the 9th Annual ACM Symposium on Computational Geometry*, pages 153–162, May 1993.
13. V. J. Milenkovic. Multiple Translational Containment, Part II: Exact Algorithms. *Algorithmica, special issue on Computational Geometry in Manufacturing, to appear.*
14. V. J. Milenkovic. A Symmetry Breaking Restriction. Personal communication, March 1995.
15. V. J. Milenkovic. Translational Polygon Containment and Minimal Enclosure using Linear Programming Based Restriction. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, 1996.
16. V. J. Milenkovic and K. Daniels. Translational Polygon Containment and Minimal Enclosure using Geometric Algorithms and Mathematical Programming. Technical Report 25–95, Center for Research in Computing Technology, Division of Applied Sciences, Harvard University, 1995.
17. V. J. Milenkovic, K. Daniels, and Z. Li. Placement and Compaction of Nonconvex Polygons for Clothing Manufacture. In *Proceedings of the 4th Canadian Conference on Computational Geometry*, pages 236–243, 1992.
18. V. J. Milenkovic and Z. Li. A Compaction Algorithm for Nonconvex Polygons and Its Application. *European Journal of Operations Research*, 84:539–560, 1995.
19. H. Minkowski. Volumen und Oberfläche. *Mathematische Annalen*, 57:447–495, 1903.
20. J. Serra. *Image Analysis and Mathematical Morphology*, volume 1. Academic Press, New York, 1982.
21. J. Serra, editor. *Image Analysis and Mathematical Morphology*, volume 2: Theoretical Advances. Academic Press, New York, 1988.
22. P. E. Sweeney and E. R. Paternoster. Cutting and Packing Problems: A Categorized, Application-Oriented Research Bibliography. *Journal of the Operational Research Society*, 43(7):691–706, 1992.