Transactions in Operational Research, 6:525-554, 1999.

Translational Polygon Containment and Minimal Enclosure using Mathematical Programming

Victor J. Milenkovic* Karen Daniels †

Abstract

A new algorithm is given for the two-dimensional translational *containment* problem: find translations for k polygons which place them inside a polygonal container without overlapping. Both the polygons and the container can be non-convex. The algorithm is based on mathematical programming principles. The containment algorithm is generalized to solve *minimal enclosure* problems: find the minimal area enclosing square or minimal area enclosing rectangle for k translating polygons.

The containment algorithm consists of new algorithms for *restriction*, *evaluation*, and *subdivision* of two-dimensional configuration spaces. Restriction establishes lower bounds through relaxation and the solution of linear programs. Evaluation establishes upper bounds by finding potential solutions. Subdivision branches, when necessary, by introducing a cutting plane. The algorithm shows that either the upper bound of the objective (overlap) is exactly zero or the lower bound is greater than zero. Hence, it gives an exact solution to the containment problem.

Experiments show that new containment algorithm clearly outperforms purely geometric containment algorithms. For data sets from the apparel industry, it can solve containment for up to ten non-convex polygons in practice. An implementation of the algorithm given in this paper has been licensed by Gerber Garment Technologies, the largest provider of textile CAD/CAM software in the U.S., and they are incorporating it into an existing CAD/CAM software product.

*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 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.

†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.

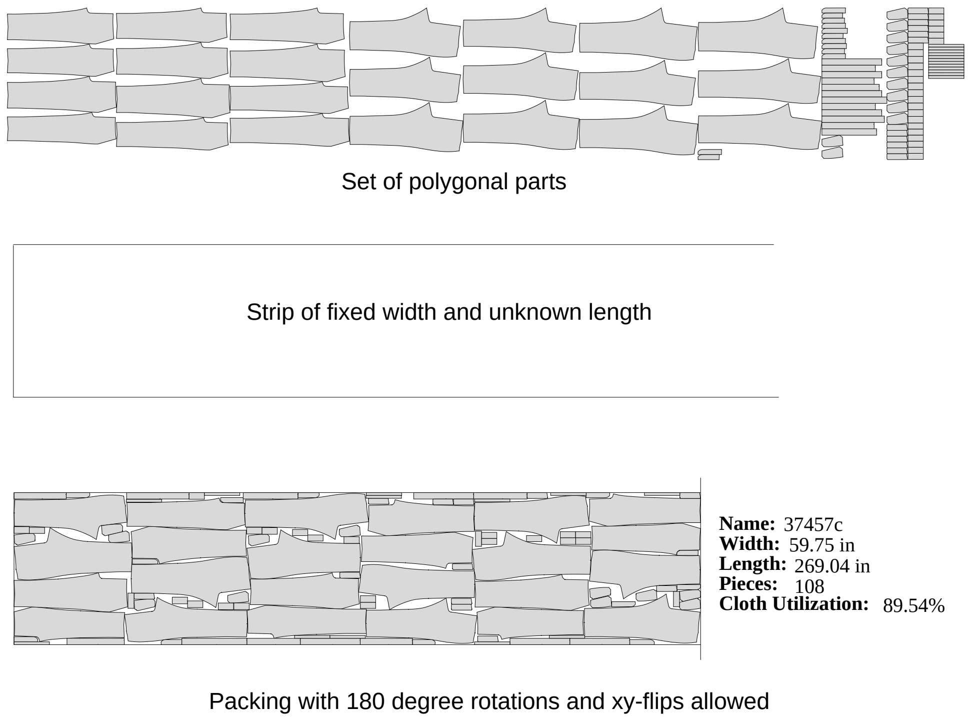

Figure 1: A marker making task in the apparel industry

1 Introduction

A number of industries generate new parts by cutting them from stock material: cloth, leather (hides), sheet metal, *etc.* These industries require good solutions to *containment* or *minimal enclosure* problems. *Containment* is the question of whether a given set of part shapes can be fit, without overlapping, into a given container. *Minimal enclosure* is the problem of minimizing the size (area, length, or other measure) of the container for a given set of parts. One might think that a few good layouts might suffice for a given industry, but such is not the case. One large apparel manufacturer "solves" about 1500 minimal enclosure problems *per week*. As illustrated in Figure 1, these are *strip-packing* problems: find an enclosure with fixed width and minimum length. Human employees generate these near-optimal layouts by hand, and we estimate that their total compensation is about three million dollars per year for that one company. Large companies are highly automated, and they could easily substitute software for humans. However, no current software (even our own) can reliably find enclosures within one percent[1] of those generated by expert humans for the number of part shapes they routinely lay out (one to five hundred). (See [4, 12, 14, 31, 13] for a survey of current results.)

Material such as cloth has a grain, and it sometimes has a "nap" (velvet or corduroy) or a colored pattern (stripes or plaid). A part cut out of such material has only one, two, four, or possibly eight allowed orientations, not arbitrary rotation. For such materials, containment can be decomposed into two subproblems: 1) selection of discrete orientations and 2) *translational* containment. *Translational containment*, containment with fixed orientations, is clearly an important subproblem. Textile manufactures use polygonal representations for their shapes because their cutting accuracy is not high enough to require spline curve representations. Therefore, it is reasonable to

[1] One percent is about the break-even point of labor vs. material costs.

assume that the shapes are polygons. The focus of this paper is translational containment and translational minimal enclosure of polygonal shapes.

Containment is NP-hard and translational containment is NP-complete (Section 1.3).[2] Containment is the "feasibility question" for minimal enclosure, and thus the appropriate complexity question is the hardness of containment. Techniques of combinatorial optimization such as integer or dynamic programming have been successfully applied to NP-complete packing problems such as layout of rectangles [17, 1, 19]. However, non-convex shapes appear to be too irregular to permit *direct* application of these techniques. Approximating individual non-convex shapes by rectangles or even their convex hulls leads to very non-optimal approximating enclosures or solutions to containment.

Even if P≠NP, it is still worth developing *exact* algorithms for containment and minimum enclosure. First, the shapes arising from industry may not correspond to an exponential time worst case. Second, even if an exact algorithm has exponential running time, it still might be useful in the development of approximate algorithms or heuristics. An approximate or heuristic strategy for hundreds of shapes can benefit from an exact "subroutine" that can solve containment or find the minimal enclosure for smaller groups of shapes [4]. For example, a common layout strategy is to cluster irregular polygons into a rectangle and then apply rectangle packing algorithms. As we stated in the previous paragraph, replacing *individual* non-convex shapes by an enclosing rectangle results in poor packing densities. However, replacing an entire group of five to ten pieces by the minimum area rectangular enclosure (for the group under translation) yields a much higher density packing. In recent work, we use the translational containment algorithm of this paper as part of a heuristic to solve industrial layout problems, even ones with multiple orientations allowed for the shapes [5]. In our work, we have noted that most applications of containment involve at least as many infeasible inputs as feasible inputs, and thus it is important that the "subroutine" detect infeasibility quickly. In the language of mathematical programming, it is just as important to establish a lower bound (no solution), as it is to establish an upper bound (find a solution).

1.1 New Algorithm

This paper presents a new algorithm for translational containment: given polygons $P_1, P_2, P_3 \ldots, P_k$ and a container polygon Q, find translations vectors $t_1, t_2, t_3, \ldots, t_k$ for the polygons that place then in the container without overlapping. This algorithm is generalized to solve several versions of *minimum enclosure*, where the container Q varies over a class, and the goal is to solve containment for a minimum area element of the class. Minimum enclosure is solved for three classes of fixed orientation rectangles: 1) fixed aspect ratio (*e.g.* squares), 2) fixed width and variable length (strip packing), and 3) arbitrary width and length.

The new algorithms are the first to make full use of the principles of mathematical programming (MP), and hence we designate them *MP containment* and *MP minimum enclosure*.[3] In order to use mathematical programming, we transform containment in three ways. First, we replace Q by its set complement $P_0 = \overline{Q}$. Containment becomes *placement*: find translations $t_0, t_1, t_2, \ldots, t_k$ that place $P_0, P_1, P_2, \ldots, P_k$ without overlapping.[4] Second, we consider placement to be a special case of a more general optimization problem, *global overlap minimization*: find translations $t_0, t_1, t_2, \ldots, t_k$

[2] Because continuous rotations make containment nonlinear, it is not clear whether rotational containment is *in* NP, even for polygonal shapes.

[3] In talks and a technical report, we also call it *LP* containment because the mathematical programming is ultimately reduced to linear programming.

[4] Clearly, one can fix $t_0 = (0, 0)$ without changing the problem.

that minimize the maximum pairwise overlap. Third, we this express overlap minimization as an optimization problem: minimize

$$\text{maximum overlap} \;=\; \max_{0 \le i < j \le k} \|t_j - t_i - u_{ij}\|, \tag{1}$$

where $\|\ \|$ is a length function on vectors, given,

$$u_{ij} \in U_{ij}, \qquad 0 \le i < j \le k, \tag{2}$$

where u_{ij} is a vector variable and U_{ij} is a non-convex polygonal region that depends on P_i and P_j. If U_{ij} is chosen properly, then the Equation 1 correctly computes the maximum overlap.

Following the pattern of many algorithms of mathematical programming, the containment algorithm uses three procedures: *restriction*, *evaluation*, and *subdivision*. *Restriction* determines regions of U_{ij} such that if u_{ij} is in this region, the objective must be positive, and it discards these regions. Like many techniques for establishing a lower bound, restriction uses a type of relaxation combined with linear programming. *Evaluation* finds a *local* minimum of the maximum overlap. *Subdivision* splits the current problem into two sub-problems.

As one would expect, the containment algorithm first applies restriction. If the entire set U_{ij} is discarded for some i, j, then there is no solution to containment; otherwise, the algorithm applies evaluation. If the local minimum is zero, then this is the global minimum and containment is solved; otherwise, the algorithm applies subdivision and recurses on the two sub-problems.

Section 1.2 relates the new containment algorithm to previous work by the authors, and Section 1.3 discusses its relation to other work. Section 2 shows how to construct the U_{ij} regions and gives the algorithm for restriction. Section 3 gives the algorithm for evaluation. Section 4 covers subdivision and proves that the algorithm subdivides to depth $O(k^2m^4 + km^2n^2)$, where m is the maximum number of edges among the polygons $P_1, P_2, \ldots, P_k$ and n is the number of edges of the container Q. Section 5 shows how to generalize the containment algorithm to solve minimum enclosure. Finally, Section 6 gives the results of experiments that show that the containment algorithm is practical for up to $k = 10$ polygons and demonstrates that it is the fastest containment algorithm to date. An implementation of the algorithm given in this paper has been licensed by Gerber Garment Technologies, the largest provider of textile CAD/CAM software in the U.S., and they are incorporating it into an existing CAD/CAM software product.

1.2 Related Work by the Authors

The authors have worked on layout problems since 1990. Although notation varies, it was known at that time that a solution to containment must satisfy,

$$t_j - t_i \in U_{ij}, \qquad 0 \le i < j \le k, \tag{3}$$

where $P_0 = \overline{Q}$ is the complement of the container and where U_{ij} is the "un-overlap" region for P_i and P_j. Section 2.1 discusses the construction of these regions. Equation 3 is the basis for all of our layout algorithms. It is clear that Equation 3 has a solution if an only if the maximum overlap (Equation 1) equals zero. This section discusses the relation of MP containment and MP minimum enclosure to our previous algorithms: *compaction*, *geometric* containment, *naive LP* containment, and *theoretical LP* containment.

In his Ph.D. thesis, Li [21, 23] gave an algorithm for *compaction*: *local* translational minimum enclosure over rectangular containers of fixed width. This algorithm requires the complement $\overline{U_{ij}}$

to be star-shaped (entirely visible from some internal point) or at least expressed as the union of a few star-shaped regions. The compaction algorithm iterates the following two steps: 1) replace each U_{ij} by a convex inner approximation and 2) use linear programming to minimize the area of the container. Step 2 also eliminates overlaps if they are sufficiently small. We generalized this algorithm to local overlap minimization, although this result was never published. The new MP evaluation algorithm in Section 3 uses a different convex approximation and linear program. This allows it to compute a local overlap minimum even in the case of large overlaps, non-zero local minima, and U_{ij} which are not the complements of star-shaped regions.

In her Ph.D. thesis, Daniels [4, 7] developed a purely geometric algorithm for translational containment. Although it does not use mathematical programming, this *geometric containment algorithm* uses restriction, evaluation, and subdivision. *Geometric restriction* repeats the following substitution for $0 \leq i < j \leq k$,

$$U_{ij} \leftarrow U_{ij} \cap (U_{ih} \oplus U_{hj}), \qquad 0 \leq h \leq k, \tag{4}$$

where $U_{ji} = -U_{ij}$ and $\oplus$ denotes the Minkowski sum (Section 2.1). *Geometric evaluation* alternates the following two steps: 1) replace the smallest area U_{ij} by a single point and 2) apply geometric restriction. *Geometric subdivision* uses horizontal or vertical cuts. The geometric containment is *approximate*: in order to guarantee a running time bound, it may be necessary to accept a non-zero overlap. For maximum overlap at most $1/N$ times the diameter of the container, it visits $\mathrm{O}(N^k)$ hypotheses.

MP containment differs significantly from geometric containment in all three stages: evaluation, restriction, and subdivision. MP evaluation uses overlap minimization, but geometric evaluation does not. MP restriction gives an exact solution for convex U_{ij}: it discards exactly those regions corresponding to non-zero overlap. In contrast, geometric restriction gives an exact solution for $k = 2$, even if the inputs are non-convex. In fact, the two restriction methods complement each other, and we find it useful to apply both. Finally, since since convexity benefits MP restriction, MP subdivision uses a more sophisticated technique that increases the convexity of the sub-problems.

MP containment is the first to use overlap minimization, but two earlier containment algorithms also use linear programming: *naive LP* containment and *theoretical LP* containment. The *naive LP* containment algorithm [27] uses the same relaxation technique as MP containment: replace each U_{ij} by its convex hull. Unlike MP evaluation, naive LP containment only checks the feasibility of the relaxed problem but does not optimize any measure. Using the feasibility check, it either eliminates all of the U_{ij} regions or none of them, but it cannot discard part of these regions, as does MP restriction. It also uses a much simpler subdivision technique. Assuming fixed k, naive LP containment is guaranteed to visit at most,

$$\mathrm{O}\left(\frac{(2kmn + k^2m^2)^{2k+1}}{k!}\right) = \mathrm{O}\left((mn + m^2)^{2k+1}\right),$$

sub-problems (each of which involves solving a linear program). By adding a modified version of MP restriction to the naive LP algorithm, essentially carrying out only one iteration, Milenkovic created a *theoretical LP* containment algorithm [24] which visits

$$\mathrm{O}\left(\frac{(6kmn + k^2m^2)^{2k}}{(k-6)!}\right) = \mathrm{O}\left((mn + m^2)^{2k}\right),$$

subproblems (which is optimal in the worst case) at a total cost of $\mathrm{O}(\log kmn)$ per subproblem. In other words, it solves each linear program in $\mathrm{O}(\log kmn)$ average time! To obtain this running time, it is necessary to use a modified and rather intricate version of the simplex method to solve the linear programs. An earlier version of the theoretical LP algorithm and the MP algorithm appear in a technical report [25].

The experiments in Section 6 demonstrate that the local overlap minimization of MP evaluation is essential for the good practical running time of MP containment. It runs much faster than naive LP containment, and Section 4.4 shows that if MP containment used the same (less sophisticated) subdivision technique as the theoretical LP algorithm, it would visit the same order of sub-problems, fewer than the naive LP algorithm. Hence MP containment offers the best of both worlds: the best practical running time and, if one desires, the optimal theoretical bound on the number of sub-problems in the worst case. The theoretical LP algorithm has a better worst case time bound per sub-problem, but in order to solve each sub-problem in $O(\log kmn)$ average time, the theoretical algorithm cannot do overlap minimization (upper bound) and it can apply only one iteration of restriction (lower bound). In contrast, MP containment establishes better upper and lower bounds before subdividing. In practice, one always establishes the best possible upper and lower bounds for a problem before branching.

1.3 Other Related Work.

In previous work, we established that *strip packing* (known as *marker making* in the apparel industry) is NP-hard [26]: given 1) a set of shapes, 2) allowed orientations for each shape (rotations and reflections), and 3) a rectangle of fixed width and indeterminate length, find the non-overlapping layout of minimum length. It was shown that *translational* marker making is NP-complete: each shape is allowed only a single orientation. The same proof works for non-rectangular containers [4, 8] to show that translational containment is NP-complete. For all of these proofs, the number of polygons k is considered to be the complexity of the problem.

In general, industrial automatic layout systems use heuristics which cannot detect infeasibility or determine if they have found the optimal enclosure. These systems use practical efficiency techniques such as inner and outer approximations, and thus the running time increases with the tightness of the fit.[5] We have licensed our software, including the containment algorithms described here, and we believe them to be superior to what is currently available.[6]

Suppose k is fixed, and the complexity is measured by m and n where m is the maximum number of edges among the polygons $P_1, P_2, \ldots, P_k$ and n is the number of edges of the container Q. In this case, solving containment requires $\Omega((mn)^{2k})$ time in the worst case [24]: specifically, the set of valid configurations has $\Omega((mn)^{2k})$ components. It may not require this much time to generate a single solution. MP containment, described in this paper, only generates a single solution. It can easily be modified to generate "all" solutions ("all" meaning at least one from each component of the solution space), but we do not claim that it would remain practical. Theoretical LP containment has the same running time bound whether it generates a single solution or "all" solutions. It is easy to write down a "brute force" containment algorithm that generates and tests "all" solutions in $O((mn + m^2)^{2k} \log mn)$ time, assuming "constant" k, by iterating over all choices of $2k$ contacting pairs among the polygons and container. Unfortunately, on all *infeasible* inputs, the brute force algorithm has $\Omega((mn)^{2k})$ running time, which is very impractical. It is unlikely to be useful on feasible inputs either.

There have been a number of translational containment algorithms for *convex* polygons [16, 10, 27,

[5]Incidentally, although we do not currently do so, we also could apply these practical efficiency techniques.
[6]Companies closely guard their products, and so it is difficult to compare our algorithms to others.

24]. For non-convex polygons and container, Avnaim and Boissonnat [3, 2] use the Minkowski sum and convex decomposition to solve containment in $O(m^2 n^2 \log mn)$ time for $k = 1$ and $O(m^4 n^4 \log mn)$ time for $k = 2$, and Avnaim [2] gives a $O(m^{14} n^6 \log mn)$ time algorithm for $k = 3$. Their algorithms will also work in the rotational case, but the complexities are higher. Avnaim and Boissonnat place three non-convex polygons into a rectangle or parallelogram container in $O(m^{60} \log m)$ time. We are not aware of any algorithms for $k > 3$ and a non-convex container other than those of the previous section: geometric containment, naive LP containment, and theoretical LP containment.

2 Restriction

The MP containment algorithm has three stages: restriction, evaluation, and subdivision. Restriction eliminates portions of the U_{ij} regions (Equation 2) where the minimum total overlap is non-zero. The restriction algorithm is based on a form of relaxation: replace each U_{ij} region by its convex hull. Once this is done, linear programming can be used to bound the overlap from below in subsets of these convex hulls. Any lower bound for the relaxed problem is valid for the original problem. As promised in Section 1.1, this section first defines the U_{ij} regions and proves that Equation 1 does indeed correspond to the maximum overlap.

2.1 The "Un-Overlap" Regions

In this paper, upper case letters, such as P_i, Q, and U_{ij} denote (polygonal) point sets in the plane. Lower case letters, such as t_i and u_{ij} denote vectors. The symbols $+$, $-$, and $\cdot$ denote vector sum, difference, and dot product, respectively. These operators are extended to point sets, for example,

$$-A = \{-a \mid a \in A\} \qquad \text{and} \qquad A + t = \{a + t \mid a \in A\}.$$

The *Minkowski sum* [28, 18, 29, 30] of two point-sets (of $\mathbf{R}^2$ in the case of this paper) is defined

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

Let $|A|$ denote the number of edges of A. It is well-known that $|A \oplus B| = \Theta(|A|^2 |B|^2)$ [20]. Although the notation may have been different, the following lemma probably goes back to Minkowski.

Lemma 2.1 *The intersection $(P_i + t_i) \cap (P_j + t_j)$ is non-empty if and only if $t_j - t_i \in P_i \oplus -P_j$.*

If we set

$$U_{ij} = \overline{P_i \oplus -P_j}, \qquad 0 \le i < j \le k, \tag{5}$$

the complement of the overlap region, then we obtain the "un-overlap" region of Equation 3. If $t_j - t_i$ lies outside U_{ij}, then $P_i + t_i$ and $P_j + t_j$ overlap. A reasonable measure of overlap is the minimum motion required to un-overlap the two polygons (also known as the *intersection depth* [11]). This in turn is the minimum distance from $t_j - t_i$ to the set U_{ij}:

$$\min_{u_{ij} \in U_{ij}} \|t_j - t_i - u_{ij}\|.$$

Hence, the objective in Equation 1 is indeed the maximum overlap.

Note: since we will be restricting the U_{ij} regions, we denote the initial regions, before restriction, as U_{ij}^{init}, $0 \le i < j \le k$.

2.2 MP Restriction Algorithm

Equation 3 is non-convex. However, the following relaxation corresponds to a convex, linear set of constraints:

$$t_j - t_i \in \mathrm{conv}(U_{ij}), \qquad 0 \leq i < j \leq k, \tag{6}$$

where $\mathrm{conv}(U_{ij})$ is the convex hull of U_{ij}. Let F_{gh} denote the set of feasible values of $t_h - t_g$ under the constraints of Equation 6. The next section shows how to compute F_{gh} using linear programming. The following substitution,

$$U_{gh} \leftarrow U_{gh} \cap F_{gh}, \tag{7}$$

only discards points of U_{gh} which lie outside F_{gh}. These points cannot correspond to a non-overlapping layout since all non-overlapping layouts are a solution to Equation 3 of which Equation 6 is a relaxation. Therefore, the substitution of Equation 7 only eliminates $u_{gh} \in U_{gh}$ corresponding to a layout with maximum overlap greater than zero.

The MP restriction algorithm iterates the basic restriction of Equation 7 for each possible pair g, h until the discarded area falls below some fixed fraction of the area of U_{gh}. In contrast, the naive LP containment algorithm (Section 1.2) simply determines the feasibility of Equation 6. If it is feasible, it retains all points of all U_{ij} regions. If it is infeasible, it discards them all. The theoretical LP containment algorithm uses a much more complicated algorithm than the one in the next section, but in essence it carries out the substitution of Equation 7 exactly once for each pair g, h. Hence, both naive LP restriction and theoretical LP restriction are weaker than the new MP restriction algorithm.

2.3 Constructing Feasible Regions

Clearly, F_{gh} depends on current list of U_{ij} polygons, which we denote $\mathcal{U} = \langle U_{ij} \mid 0 \leq i < j \leq k \rangle$. Previous restriction or subdivision may have already modified the regions in $\mathcal{U}$. Define a subroutine $\mathrm{boundpt}(\mathcal{U}, g, h, \omega)$, where ω is a unit vector, as follows: 1) solve the linear program with the constraints of Equation 6 and objective to maximize $\omega \cdot (t_h - t_g)$ and 2) return $t_h - t_g$. Clearly, the output of $\mathrm{boundpt}(\mathcal{U}, g, h, \omega)$ is a point $u_{gh} \in F_{gh}$ which maximizes $\omega \cdot u_{gh}$ and therefore lies on the boundary of F_{gh}. Of course, if the linear program of Step 1 is infeasible, then F_{gh} is empty.

Fix g and h. If calling boundpt with $\omega = (1, 0)$, $\omega = (0, 1)$, $\omega = (-1, 0)$, and $\omega = (0, -1)$ yields the same point each time, then F_{gh} is a single point. Otherwise, it will yield at least two distinct vertices of an inner approximation $F \subseteq F_{gh}$ (F might be a degenerate "bi-angle"). For each edge ab of F, let ω be the outer normal vector to ab, and let $c = \mathrm{boundpt}(\mathcal{U}, g, h, \omega)$. If $c \in ab$, then ab is an edge of F_{gh}. Otherwise, replace ab of F by ac and cb. This algorithm requires $2|F_{gh}|$ calls to boundpt: one for each vertex of F_{gh} and one for each edge. If this many calls is too costly in practice, then for any set Ω of query directions ω, the following intersection of half-planes is a superset of F_{gh}:

$$S(\Omega) = \bigcap_{\omega \in \Omega} \{(x, y) \mid \omega \cdot (x, y) \leq \omega \cdot \mathrm{boundpt}(\mathcal{U}, g, h, \omega)\}. \tag{8}$$

Replacing U_{gh} with $S(\Omega) \cap U_{gh}$ is also a valid (albeit less severe) restriction.

3 Evaluation

Equation 6 relaxes Equation 3 and a solution $\langle t_0, t_1, t_2, \ldots, t_k \rangle$ to the former may not be a solution to the latter. Evaluation moves this overlapping layout to a local minimum of the maximum overlap in Equation 1. If the local minimum has maximum overlap equal to zero, then it is a global minimum and a solution to containment. Even if the local minimum has non-zero maximum overlap, it acts as a guide to the subdivision algorithm of Section 4. This section describes the evaluation algorithm.

Just as restriction temporarily replaces each U_{ij} by a convex *superset*, namely its convex hull $\mathrm{conv}(U_{ij})$, the evaluation algorithm temporarily replaces each U_{ij} by a convex *subset* $I_{ij} \subseteq U_{ij}$. The algorithm applies an *overlap minimization step* which is to minimize the overlap measure of Equation 1 with respect to $u_{ij} \subseteq I_{ij}$, $0 \leq i < j \leq k$. It repeats this overlap minimization step with different choices of I_{ij} until it reaches a local overlap minimum with respect to the original U_{ij} regions. With the appropriate choice of vector length function, each step can be solved using linear programming.

Section 3.1 describes a linearized vector length function and proves it is a good approximation to the Euclidean length. Section 3.2 describes the overlap minimization step and how to implement it using linear programming. Section 3.3 proves the convergence properties of the minimization step. Finally, Section 3.4 shows how to construct I_{ij} which satisfy the condition sufficient to ensure convergence and which are "as large as possible". The latter property improves convergence in practice.

3.1 Length Functions

The only maximum overlap value that matters is zero, and thus it is not necessary to use the Euclidean length $|w| = \sqrt{w_x^2 + w_y^2}$. To allow us to use linear programming, we use a linear, convex length function. Let Ω be a set of unit vectors in the plane. Define,

$$|w|_\Omega = \max_{\omega \in \Omega} \omega \cdot w.$$

The ball $B_\Omega(t, r) = \{w \mid |w - t|_\Omega = r\}$ of radius r is a convex polygon bounded by edges at distance r from t and with outward unit normal vectors in Ω. Clearly, if Ω were the (infinite) set of all unit vectors, then $|w|_\Omega$ would become the Euclidean length and $B_\Omega(r)$ would become the circular region of radius r.

Lemma 3.1 *If* $(\cos(\pi i/4), \sin(\pi i/4)) \in V$, $i = 0, 1, 2, \ldots, 7$, *then*

$$0.91\,|w| \;\leq\; |w|_\Omega \;\leq\; |w|.$$

Proof: Since Ω is a set of unit vectors, the upper bound is clear. The set Ω includes at least one vector ω whose angle with w is no more than $\pi/8$. Using the trigonometric formula for the dot product,

$$\omega \cdot w \geq |\omega|\,|w| \cos(\pi/8) \geq 0.91|w|.$$

■

Given a distance function $|\ |_\Omega$, we can define the distance $\delta_\Omega(t, U)$ from a point t to a (closed, polygonal) region U,

$$\delta_\Omega(t, U) = \min_{u \in U} |t - u|_\Omega.$$

A point $u_{\min} \in U$ which attains the minimum ($|t - u_{\min}|_\Omega = \delta_\Omega(t, U)$) is called *a nearest point* of U to t. Note: the nearest point may not be unique.

For a given polygonal region U, a good way to choose Ω is to start with $(\cos(\pi i/4), \sin(\pi i/4))$, $i = 0, 1, 2, \ldots, 7$, and then add to Ω the outward unit normal of each edge of U. This way, if a nearest point of U to t happens to lie on an edge, then $\delta_\Omega(t, U)$ is the true Euclidean distance.

For the overlap minimization step in the next section, we need to define the nearest *element*–point, vertex, or edge–of U to t. If t lies in the interior of U, the nearest element is the point t; otherwise, the nearest element is a vertex or edge of U which we select as follows. Among all nearest points of U to t under the $|\ |_\Omega$ length, choose a point $u_{\min}$ which is closest to t under the Euclidean length. If $u_{\min}$ is a vertex v of U, then v is the nearest element; otherwise, the nearest element is the edge e of U which contains $u_{\min}$. Note: even if we first minimize the $|\ |_\Omega$ metric and then minimize the Euclidean metric, $u_{\min}$ might still be ambiguous, in which case one must "break the tie" arbitrarily.

Lemma 3.2 *Except for t belonging to a set of measure zero, the nearest element of U to t is constant in a neighborhood of t.*

Proof: A complete proof of this lemma requires a tedious case analysis, which we will merely summarize here. If the nearest element varies in every neighborhood of t, then a) t must be equidistant from two vertices, two edges, or an edge and a vertex, under the $|\ |_\Omega$ metric, or b) some concave vertex of U must be a nearest point to t, or c) two vertices of U are equidistant from t under the Euclidean metric.

In each case, t must satisfy some constraint which depends on the coordinates of up to two vertices or edges of U. The only constraint which could possibly be non-linear is (c), because the Euclidean metric is non-linear. However, we know that (c) corresponds to a perpendicular bisector, which is a line. Therefore all possible constraints are linear. Thus, if the selected element is not constant in a neighborhood of t, then t must lie within the union of a finite set of lines. ∎

3.2 The Overlap Minimization Step

This section describes the overlap minimization step and how to implement it using linear programming. The step takes as input a *configuration*, a list $\langle t_0, t_1, t_2, \ldots, t_k \rangle$ of vectors, which satisfies the convex, linear constraints of Equation 6. The step generates as output a new configuration which satisfies the same constraints and which diminishes the overlap measure of Equation 1. Each invocation of boundpt in MP restriction generates a configuration, and the very last invocation of boundpt generates the starting configuration for evaluation. The evaluation (overlap minimization) algorithm iterates the overlap minimization step until it converges to a local overlap minimum.

The overlap minimization step has four parts:

1. For each U_{ij}, $0 \le i < j \le k$, select a convex, polygonal subset $I_{ij} \subseteq U_{ij}$ which contains the nearest element of U_{ij} to $t_j - t_i$.

2. Apply the constraints of Equation 6 to the "perturbed" vectors $t_i + \Delta t_i$, $0 \le i \le k$, where t_i is considered to be a constant and Δt_i is a perturbation variable.

3. Under these constraints, find the vectors Δt_i, $0 \leq i < j \leq k$ which minimize the objective,

$$d_{\max} + c \sum_{i=0}^{k} (|\Delta t_{ix}| + |\Delta t_{iy}|). \tag{9}$$

   Here c is a constant, and

$$d_{\max} = \max_{0 \leq i < j \leq k} |t_j - t_i - u_{ij}|_\Omega, \tag{10}$$

   where the u_{ij} are variables which satisfy the constraints,

$$u_{ij} \in I_{ij}, \qquad 0 \leq i < j \leq k. \tag{11}$$

4. Update t_i, $0 \leq i \leq k$,

$$t_i \leftarrow t_i + \Delta t_i.$$

The evaluation algorithm repeats this step until the Δt_i are all $(0, 0)$ and hence the t_i are unchanged.

The iterative step can be solved using linear programming as follows. Since I_{ij} is convex and polygonal, the condition $u_{ij} \in I_{ij}$ (Equation 11) can be expressed as a finite set of linear constraints. Since $d_{\max}$ has positive coefficient in the objective, Equation 10 can be replaced by a set of constraints of the form,

$$|t_j + \Delta t_j - t_i - \Delta t_j - u_{ij}|_\Omega \leq d_{\max},$$

which in turn can be expressed as the set of linear constraints,

$$\omega \cdot (t_j + \Delta t_j - t_i - \Delta t_j - u_{ij}) \leq d_{\max}, \qquad \omega \in \Omega.$$

Finally, to make the objective of Equation 9 linear, each vector Δt_i can be expressed using four non-negative variables,

$$\Delta t_i = (t_{ix}^+ - t_{ix}^-, t_{iy}^+ - t_{iy}^-),$$

and thus,

$$|\Delta t_{ix}| + |\Delta t_{iy}| = t_{ix}^+ + t_{ix}^- + t_{iy}^+ + t_{iy}^-.$$

Taken together, these new variables and constraints convert the overlap minimization step to a linear program.

3.3 Convergence of Overlap Minimization

The overlap minimization step only requires that each I_{ij} contain the nearest element (point, vertex, or edge) of U_{ij} to $t_j - t_i$. In the next section, we will show how to select I_{ij} which is also "as large as possible". However, for the purpose of proving convergence, it is only necessary that it contain this nearest element.

Lemma 3.3 *For sufficiently small $c > 0$, the overlap minimization step converges.*

Proof: Since each I_{ij} contains the nearest element and hence a nearest point to $t_j - t_i$, $d_{\max}$ equals the current overlap when $\Delta t_i = (0, 0)$, $0 \leq i \leq k$, and the objective of Equation 9 also equals the current overlap. If no motion away from $\langle t_0, t_1, t_2, \ldots, t_k \rangle$ reduces $d_{\max}$, then for every $c > 0$, the value of the objective will be minimum when $\Delta t_i = (0, 0)$, $0 \leq i \leq k$. If some motion does reduce $d_{\max}$, then for sufficiently small c, the objective will attain its minimum for non-zero Δt_i, $0 \le i \le k$: select c smaller than the partial derivative of the overlap in that direction.

Since $I_{ij} \subseteq U_{ij}$, it follows that,

$$\delta_\Omega(t_j + \Delta t_j - t_i - \Delta t_i, I_{ij}) \ge \delta_\Omega(t_j + \Delta t_j - t_i - \Delta t_i, U_{ij}).$$

Therefore, if $d_{\max}$ is reduced, then the overlap is indeed reduced. Hence, each step either leaves the t_i fixed or it reduces the overlap. Since the overlap is non-negative, the iteration must converge. ■

Let us designate the limit of the convergence as $\langle t_0^*, t_1^*, t_2^*, \ldots, t_k^* \rangle$, and let us designate the choice of convex, polygonal subsets for U_{ij} with respect to $t_j^* - t_i^*$ as I_{ij}^*.

Theorem 3.4 *Suppose that the nearest element of U_{ij} is constant in a neighborhood of $t_j^* - t_i^*$, $0 \le i < j \le k$. If so, then $\langle t_0^*, t_1^*, t_2^*, \ldots, t_k^* \rangle$ is a local overlap minimum for the U_{ij} regions.*

Proof: By the definition of limit, the overlap minimization algorithm must reach a step for which each $t_j - t_i$, $0 \le i < j \le k$, lies within the "neighborhood of constant nearest element" of $t_j^* - t_i^*$ for this and all subsequent overlap minimization steps. For $t_j' - t_i'$ in this neighborhood,

$$\delta(t_j' - t_i', I_{ij}) = \delta(t_j' - t_i', I_{ij}^*) = \delta(t_j' - t_i', U_{ij}),$$

because these distances equal the distance to the nearest element.

If $\langle t_0^*, t_1^*, t_2^*, \ldots, t_k^* \rangle$ were not a local overlap minimum for the U_{ij}, we could pick $\langle t_0', t_1', t_2', \ldots, t_k' \rangle$ such that $t_j' - t_i'$ lies within the constant-element neighborhood, $0 \le i < j \le k$, and such that,

$$\min_{0 \le i < j \le k} \delta(t_j' - t_i', U_{ij}) < \min_{0 \le i < j \le k} \delta(t_j^* - t_i^*, U_{ij}),$$

and therefore,

$$\min_{0 \le i < j \le k} \delta(t_j' - t_i', I_{ij}) < \min_{0 \le i < j \le k} \delta(t_j^* - t_i^*, I_{ij}),$$

for this and all subsequent minimization steps. This implies that the minimization could not converge to $\langle t_0^*, t_1^*, t_2^*, \ldots, t_k^* \rangle$. ■

We know from Lemma 3.2 that the conditions of Theorem 3.4 hold except for a set of measure zero. The coordinates of the vertices and edges of the U_{ij} regions would have to be in very special position for evaluation to converge to an element of this set of measure zero. If this ever were to happen, subsequent subdivision would eventually cut one of the offending U_{ij} regions into two parts, and eliminate the special case.

3.4 Constructing Convex Subsets

Given a (non-convex) polygonal planar region U and a point t, this section shows how to select a convex polygonal region $I \subseteq U$ such that I contains the nearest element of U to t and in addition I is "as large as possible". The latter is important to ensure good practical running times for the overlap minimization algorithm.

This *convex subset construction* algorithm first sets $I = U$, which is non-convex, and then performs a series of half-plane "cuts" on I: replace I by $I \cap H$, where H is a half-plane. We say that we "cut I by an edge e of U" if H is the natural half-plane $H(e)$ defined by e: $H(e)$ has e as part of its bounding line and it lies to the "interior side" of e. We say that we "cut I by a vertex v (with respect to t)" if H is $H(v,t)$: a half-plane who boundary contains v and which contains all points closer to t than v. In the case of the Euclidean metric, $H(v,t)$ is uniquely determined: its bounding line is perpendicular to the segment tv. In the case of the $|\ |_\Omega$ metric, the bounding line L of $H(v,t)$ is a tangent–or more precisely, a line of support–to the ball $B_\Omega(t,|v-t|_\Omega)$ of radius $|v-t|_\Omega$ about t. In case the line of support is not unique, the boundary is the line perpendicular to tv.

The convex subset construction has two stages, each of which performs a series of half-plane cuts on I. After both stages have completed, I is convex, and it is the output of the algorithm. Stage I ensures that I contains the nearest element, and Stage II ensures that it is convex and "large as possible".

Stage I: If the nearest element of U to t is a point, there is nothing to be done. If the nearest element is an edge e, cut I by e. If the nearest element is an edge or vertex, let e_{left} and e_{right} be the two neighbors on the boundary of U. Cut U by e_{left} and e_{right}.

Stage II: Let $d = \delta_\Omega(t, I)$. While there is some edge or vertex of I that lies farther than d away from t, do the following three steps.

1. Set d' equal to the distance to this edge or vertex.
2. Cut I by this edge or vertex.
3. Set $d = d'$.

Stage I ensures that I is convex in the vicinity of the nearest element. Stage II has the property that it only makes cuts that remove portions of I that are farther than d'. In particular, it cannot cut away the nearest element. Since it eventually visits every concave vertex (or cuts it away in an earlier cut), the output must be convex.

4 Subdivision

Subdivision divides the current problem into two sub-problems. After restriction and evaluation, the current state of the problem is a list $\mathcal{U} = \langle U_{ij}, 0 \leq i < j \leq k\rangle$ of regions, and a list of translation vectors $\langle t_0, t_1, t_2, t_3, \ldots, t_k\rangle$ ($t_0 = (0,0)$) which is a local overlap minimum. We refer to the list $\mathcal{U}$ of regions as a *hypothesis* and the list of vectors as a *configuration*. Subdivision selects a pair i, j and splits U_{ij} into two regions U_{ij}^- and U_{ij}^+. By replacing U_{ij} by either U_{ij}^- or U_{ij}^+ in $\mathcal{U}$, it creates two sub-hypotheses $\mathcal{U}^-$ and $\mathcal{U}^+$. Subdivision has two goals. The first goal is to make the convex hull $\mathrm{conv}(U_{ij})$ "stick out" of U_{ij}^{init} less (see Section 2.1). In particular, if we define the *maximum overlap* of $\mathrm{conv}(U_{ij})$ to be,

$$\max_{u \in \mathrm{CH}(U_{ij})} \delta(u, U_{ij}^{\text{init}}),$$

then subdivision should make the maximum overlap of $\mathrm{conv}(U_{ij}^-)$ and $\mathrm{conv}(U_{ij}^+)$ smaller.[7] The second goal is to drive the evaluation "away" from the current overlapping configuration $\langle t_0, t_1, t_2, t_3, \ldots, t_k\rangle$. Since the current configuration is at a local overlap minimum, the only way to prevent evaluation

[7]Since no linear programming is involved in subdivision, this section uses the Euclidean length, although one could still use the linearized length $|\ |_\Omega$.

from "falling into" the same local minimum is to make $t_j - t_i$ fall as far outside $\text{conv}(U_{ij}^-)$ and $\text{conv}(U_{ij}^+)$ as possible. Since evaluation satisfies the constraints of Equation 6, it cannot converge to the same configuration for either $\mathcal{U}^-$ or $\mathcal{U}^+$ as it did for $\mathcal{U}$ if $t_j - t_i$ lies outside $\text{conv}(U_{ij}^-)$ and $\text{conv}(U_{ij}^+)$.

The first goal of subdivision is accomplished by almost any reasonable splitting algorithm. The second goal is the focus of this section. This section gives a *distance-based* subdivision algorithm that creates sub-hypotheses which are the maximum possible *distance* from the current configuration. Because it fragments the regions as little as possible and uses the output of evaluation, this algorithm is the one we use in our practical MP containment algorithm. However, we cannot prove that a containment algorithm that uses it necessarily has a good theoretical running time bound. In separate work [24], we give a *combinatorial* subdivision algorithm that drives the evaluator to a configuration which is *combinatorially* different. The combinatorial algorithm gives containment a provable running time bound, but it sacrifices practicality for the sake of this bound. At the end of this section, we present a compromise "semi-theoretical" algorithm which has the same bound on the number of hypotheses as the theoretical algorithm.

Section 4.1 gives the high level description of distance-based subdivision and gives three cases. Section 4.2 gives an algorithm for selecting a subdividing line for the most difficult case. Section 4.3 proves that if the MP containment algorithm uses distance-based subdivision, it will only subdivide to finite depth. Finally, Section 4.4 provides an alternative subdivision algorithm that yields a tighter bound on the number of hypotheses, although it may not be as practical as distance-based subdivision.

4.1 Distance-Based Subdivision.

Evaluation has computed a configuration $\langle t_0, t_1, ..., t_k \rangle$ which is a local overlap minimum. If the overlap is zero, there is no need to subdivide. If the overlap is not zero, the subdivision algorithm selects the pair i, j such that $P_i + t_i$ and $P_j + t_j$ are the most overlapped: $\delta_\Omega(t_j - t_i, U_{ij})$ is maximal. It then subdivides U_{ij} into U_{ij}^- and U_{ij}^+. There are three cases:

1. U_{ij} **has multiple components.** It sets U_{ij}^- equal to the component of U_{ij} which is nearest to $t_j - t_i$. It sets U_{ij}^+ equal to the union of the other components.

2. U_{ij} **is connected and every ray out of** $t_j - t_i$ **intersects** U_{ij}**.** It selects the point u of U_{ij} which is nearest to $t_j - t_i$. It splits U_{ij} into $U_{ij}^- = H^- \cap U_{ij}$ and $U_{ij}^+ = H^+ \cap U_{ij}$ where H^- and H^+ are the the half-planes to the left and right of line $(t_j - t_i)u$, respectively.[8]

3. U_{ij} **is connected and** $t_j - t_i$ **can "see to infinity" along some ray.** It selects a line L through $t_j - t_i$ which splits U_{ij} into U_{ij}^- and U_{ij}^+. The line is chosen to maximize

$$\min(\delta(t_j - t_i, \text{conv}(U_{ij}^-)),\ \delta(t_j - t_i, \text{conv}(U_{ij}^+))), \tag{12}$$

   the minimum Euclidean distance from $t_j - t_i$ to the convex hulls of the two halves of U_{ij}.

For Cases 1 and 2, determining the nearest component or point of U_{ij} to $t_j - t_i$ can clearly be done in $O(|U_{ij}|)$ time. The next section shows how to determine whether $t_j - t_i$ can "see to infinity"

[8] A point p is on the left of line ab triangle abp has counterclockwise circulation: *left* is from the point of view of an observer standing at a and facing toward b.

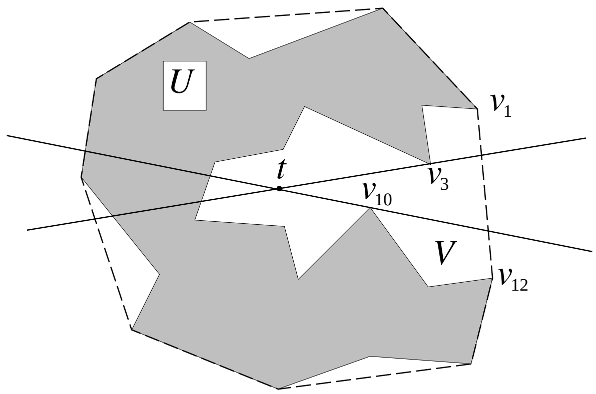

Figure 2: Region U has "bay" V containing t. Minimum angle is θ_3 and maximum is θ_{10}: $\alpha = \theta_{10}$ and $\beta = \theta_3 + 2\pi$.

and, if so, how to compute quickly the line described in Case 3. Let $L_{\max}$ denote the splitting line which maximizes the expression in Equation 12. The algorithm in the next section computes $L_{\max}$ using binary search on the angle of the line.

4.2 Choosing a Subdivision Line

To simplify notation, let t denote $t_j - t_i$ and U denote U_{ij} for the region to be split. If U has a single connected component, then the subdivision algorithm must choose a splitting line through t. Keep in mind that $t \in \operatorname{conv}(U)$ because evaluation satisfies the constraints of Equation 6. Let V be the component of $\operatorname{conv}(U) \cap \overline{U}$ which contains t. Either V is a "hole" in U or a "bay". The polygon V is a "bay" if and only if its boundary has an edge in common with $\operatorname{conv}(U)$. If V is a "hole", then Case 2 applies. If V is a bay but t cannot "see out of the bay", then Case 2 still applies. Case 3 only applies if t can see out of the bay. This section first shows how to calculate the range of angles $[\alpha, \beta]$, if any, through which t can see out of the bay. If the interval exists, then subdivision selects a splitting line by binary search on $\theta \in [\alpha, \beta]$.

As shown in Figure 2, let $v_1, v_2, v_3, \ldots, v_l$ be the vertices of the "bay" V, winding around t counterclockwise, where the "entrance" to the bay is the edge $v_l v_1$. (This is the edge V has in common with $\operatorname{conv}(U)$.) To determine $[\alpha, \beta]$, first assign an angle θ_i to each vertex v_i, $1 \leq i \leq l$, as follows. Set θ_1 to an angle corresponding to the direction $tv_1 = v_1 - t$: $(\cos\theta_1, \sin\theta_1)$ points in the direction of this vector. We say "an" angle because one could add any multiple of 2π. Next, for $2 \leq i \leq k$, set

$$\theta_i = \theta_{i-1} + \angle v_{i-1} t v_i, \tag{13}$$

where the angle is computed in a "signed" fashion to lie in the range $-\pi$ to π. If v_i is counterclockwise from v_{i-1} from the vantage point t, then the angle is positive; otherwise, it is negative. The point t can "see out" of V if the range of angles is less than a full circle:

$$\max_{1 \leq i \leq l} \theta_i - \min_{1 \leq i \leq l} \theta_i < 2\pi.$$

If this is so, then we can set,

$$\alpha = \max_{1 \leq i \leq l} \theta_i \qquad \text{and} \qquad \beta = \min_{1 \leq i \leq l} \theta_i + 2\pi.$$

If $[\alpha, \beta]$ exists, then Case 3 holds, and the subdivision algorithm must calculate a cutting line. For $\theta \in [\alpha, \beta]$, let $L(t, \theta)$ be the line through t with angle θ. Let $U^-(t, \theta, U)$ be the portion of U which lies to the left of $L(t, \theta)$–from the vantage point of someone standing at t and looking along $L(t, \theta)$ towards its intersection with U. Let $U^+(t, \theta, U)$ be the portion of U which lies to the right of $L(t, \theta)$. Define,

$$d^-(t, \theta, U) = \delta(t, \mathrm{conv}(U^-(t, \theta, U)), \qquad \text{and} \qquad d^+(t, \theta, U) = \delta(t, \mathrm{conv}(U^+(t, \theta, U)).$$

As indicated in Equation 12, the goal of subdivision is to select θ such that $\min(d^-(t, \theta, U), d^+(t, \theta, U))$ is maximized. The following lemma indicates that θ can be found by binary search.

Lemma 4.1 *For $\theta \in [\alpha, \beta]$, the function $d^-(t, \theta, U)$ is a non-increasing function of θ, and the function $d^+(t, \theta, U)$ is a non-decreasing function of θ.*

Proof: Let $\theta, \theta' \in [\alpha, \beta]$ such that $\theta' > \theta$. The lines $L(t, \theta)$ and $L(t, \theta')$ form two angles of size $\theta' - \theta$: one into the bay and one out of the bay. Region $U^-(t, \theta')$ is region $U^-(t, \theta)$ *plus* the portion of U that lies in the outward angle and *minus* the portion of U which lies in the inward angle. But the outward angle is empty because t can see out for all angles in $[\alpha, \beta]$. Therefore $U^-(t, \theta') \subseteq U^-(t, \theta)$. Since the distance of t to a set is a minimum over all elements of that set,

$$d^-(t, \theta', U) = \delta(t, \mathrm{conv}(U^-(t, \theta', U)) \leq \delta(t, \mathrm{conv}(U^-(t, \theta, U)) \leq d^-(t, \theta, U).$$

The proof for $d^+(t, \theta, U)$ is analogous. ■

Let $\theta_{\max}$ be the value of theta that maximizes $\min(d^-(t, \theta, U), d^+(t, \theta, U))$ over $\theta \in [\alpha, \beta]$. If for a particular θ, $d^-(t, \theta, U) > d^+(t, \theta, U)$, then Lemma 4.1 tells us that $\theta_{\max} > \theta$. Similarly if $d^-(t, \theta, U) < d^+(t, \theta, U)$, then $\theta_{\max} < \theta$. Hence we can use binary search to find $\theta_{\max}$. Furthermore, since the search is numerical, we can use linear interpolation to predict the most likely value of $\theta_{\max}$ once the search converges to a small interval.

This algorithm is approximate, but it rapidly converges to the accuracy of double precision floating point. Elsewhere, we have described an "exact" algorithm [6], but it still ultimately has to solve a fourth degree equation to compute $\theta_{\max}$. Even though there is a formula for the roots of the quartic, implementing the formula still requires computing square roots and fourth roots, which must be computed numerically.

The subdivision algorithm has one more "cleaning" step which is necessary for the proof in the next section. Let θ_i and θ_j be the vertex angles of Equation 13 that "bracket" $\theta_{\max}$: $\theta_{\max} \in [\theta_i, \theta_j]$ and no other vertex angle lies in this interval. If setting $\theta_{\max}$ equal to either θ_i or θ_j does not decrease $\min(d^-(t, \theta, U), d^+(t, \theta, U))$, then the algorithm does so. In other words, it "prefers" to return $L_{\max} = L(t, \theta_{\max})$ which passes through a vertex of the bay V.

4.3 Bounds on Subdivision

This section proves that the practical MP containment algorithm presented in this paper terminates after a finite number of subdivisions. Consider each hypothesis $\mathcal{U}$ to be a node in a binary tree with children $\mathcal{U}^-$ and $\mathcal{U}^+$. We show here that the maximum depth of a leaf node is $\mathrm{O}(k^2m^4 + km^2n^2)$ where n is the complexity of the container and km is a bound on the complexity of the polygons to be placed (Section 1.3).

We say that a vertex v of a region U is a *hull vertex* if it lies on the boundary of the convex hull $\mathrm{conv}(U)$. If a vertex is not a hull vertex it is a *non-hull vertex.* Define the *non-convexity* of a region U, denoted as $\mathrm{NC}(U)$, to be the number of connected components of U plus the number of non-hull vertices of U. This is a good measure of the non-convexity of U because if U is not convex, it can have multiple components or non-hull vertices or both. For a hypothesis $\mathcal{U}$, define

$$\mathrm{NC}(\mathcal{U}) = \sum_{0 \le i < j \le k} \mathrm{NC}(U_{ij}).$$

Lemma 4.2 and its corollaries show that intersection with a half-plane and thus MP restriction cannot increase non-convexity. Lemma 4.5 shows that distance-based subdivision generates sub-hypotheses whose non-convexity is less than their parent. These results imply that the depth of the subdivision tree cannot be greater than the non-convexity of the initial hypothesis of Equation 5 on page 6. By the results stated in Section 2.1, the initial hypothesis $\mathcal{U}^{\mathrm{init}}$ has $\mathrm{O}(k^2m^4 + km^2n^2)$ vertices, which is also a bound on its non-convexity. Since the subdivision tree has bounded depth, it must be finite, and hence MP containment terminates.

Lemma 4.2 *If U is a connected polygonal region and if line L splits U into U^- and U^+, then $\mathrm{NC}(U^-), \mathrm{NC}(U^+) \le \mathrm{NC}(U)$.*

Proof: Without loss of generality, L is the x-axis $y = 0$. Let H^- be the lower half-plane $y \le 0$ and let H^+ be the half-plane $y \ge 0$: $U^- = H^- \cap U$ and $U^+ = H^+ \cap U$. Orient the boundary of U such that the interior of U is "on the left": the outer boundary is counterclockwise, and the boundary of each "hole" in U is clockwise. Assume for now that L does not pass through a vertex of U. We will dispense with this non-degeneracy assumption later.

Let E be the set of components of $L \cap \overline{U}$. Each $e \in E$ is an edge of some component of $H^- \cap \overline{U}$ and of some component of $H^+ \cap \overline{U}$. Let E^+_{bounded} denote the set of $e \in E$ which lie on some *bounded* component of $H^+ \cap \overline{U}$. Let $N^+ = |E^+_{\mathrm{bounded}}|$. We claim that U^+ has at least N^+ non-hull vertices and that U^- has no more than $N^+ + 1$ components. Since, U has one component, this will imply that $\mathrm{NC}(U^-) \le \mathrm{NC}(U)$. The proof for U^+ is analogous.

U^+ **has at least** N^+ **non-hull vertices.** Let B^+_i be a bounded connected component of $H^+ \cap \overline{U}$. Let N^+_i be the number of edges in E^+_{bounded} which are part of the boundary of B^+_i. This means that the boundary of B^+_i has N^+_i chains which are subsets of of the boundary of U. Each chain must have at least one vertex in the interior of H^+, and since B^+_i is bounded, this vertex is non-hull. Since $N^+ = \sum N^+_i$, the first claim follows.

U^- **has no more than** $N^+ + 1$ **components.** The line L cuts the boundary of U into chains. The orientation of the boundary of U implies an orientation (direction) for each chain. We say that a chain $\gamma \subset H^-$ is *rightward* if its starting point on L has lesser x-coordinate than its ending point on L. *Leftward* is defined analogously. Because U is connected, each component of U^- must touch L. The boundary of each component consists of exactly one rightward chain followed by leftward chains alternating with edges from E. Thus, there is a one-to-one correspondence between components of U^- and rightward chains.

Let B^-_i be a connected component of $H^- \cap \overline{U}$, and let N^-_i be the number of rightward chains in the boundary of B^-_i. Since each rightward chain in H^- is on the boundary of some component of $H^- \cap \overline{U}$, $\sum N^-_i$ is the number of components of U^-. The boundary of B^-_i has $N^-_i + 1$ edges from E. If B^-_i is bounded, then at most one of these touches the unbounded component of $H^+ \cap \overline{U}$: if there were two, then we could trace a path in $\overline{U}$ from "positive infinity" down into B^-_i and out to "positive infinity" again. This path would separate U into two components. Hence, B_i^- has at least N_i^- of its edges belonging to E^+_{bounded}. If B_i^- is the unbounded component of $H^- \cap \overline{U}$, then it has the two unbounded edges of E on its boundary, and the rest of the edges from E on its boundary must be in E^+_{bounded}. Hence, $N_i^- - 1$ of its edges belong to E^+_{bounded}. It follows that $(\sum N_i^-) - 1 \leq N^+$ which implies $\sum N_i^- \leq N^+ + 1$. This establishes the second claim and the lemma (for non-degenerate L).

If L passes through a vertex of U, then we can perturb it slightly downward to L_{perturb}. As long as it does not hit another vertex, then the number of connected components in the corresponding U^-_{perturb} will be the same.[9] It is true that a non-hull vertex of U^+_{perturb} might lie on L, but it if does, it is a *hull* vertex of U^-. Therefore, we eliminate the same number of non-hull vertices from U^- as from U^-_{perturb}. ■

Corollary 4.3 *If U is a polygonal region and if line L splits U into U^- and U^+, then* $\mathrm{NC}(U^-), \mathrm{NC}(U^+) \leq \mathrm{NC}(U)$.

Proof: If U has multiple components, then we can apply Lemma 4.2 independently to each component. Since a non-hull vertex of a component of U is a non-hull vertex of U, decreasing the number of non-hull vertices of a component decreases the number of non-hull vertices of U by the same number. ■

Corollary 4.4 *MP restriction does not increase* $\mathrm{NC}(\mathcal{U})$.

Proof: MP restriction intersects each U_{ij} in $\mathcal{U}$ with a convex region C_{ij}. Intersecting U_{ij} with C_{ij} is equivalent to intersecting U_{ij} with the $|C_{ij}|$ half-planes corresponding to the edges of C_{ij}. ■

Lemma 4.5 *If distance-based subdivision splits U into U^- and U^+, then* $\mathrm{NC}(U^-), \mathrm{NC}(U^+) < \mathrm{NC}(U)$.

Proof: As in the definition of the subdivision algorithm, U denotes one of the U_{ij} regions and t denotes $t_j - t_i$ for the current configuration. The algorithm generates a line L through t which splits U into $U^- = H^- \cap U$ and $U^+ = H^+ \cap U$, where H^- and H^+ are the half-planes to the left and right of $L_{\max}$ (from the point of view of t looking towards U).

If t cannot "see to infinity" in any direction, then L is chosen to pass through the vertex u of U which is nearest to t. This vertex is a non-hull vertex of U, but it is a hull vertex of U^- and U^+. Furthermore, the component $V \subset \mathrm{conv}(U) \cap \overline{U}$ of t must have at least one vertex v^- in H^- and at least one vertex v^+ in H^+. If $L \cap V$ is in E^+_{bounded}, then v^+ will be one of the N_i^+ non-hull vertices eliminated from U^-. Analogously, v^- will be one of the non-hull vertices eliminated from U^+. Hence, u will be an "extra" non-hull vertex that is eliminated.

If t can "see to infinity", then it lies in a concave "bay" of U. The convex hull of U has an edge ab across the entrance to this bay. We note that none of the non-hull vertices of the bay play the role of one of the non-hull vertices in the proof of Lemma 4.2 because all of these lie on the boundary of a *bounded component* of $H^+ \cap \overline{U}$ or, analogously, $H^- \cap \overline{U}$. Since t can see to infinity along the splitting line L, each point in the bay ends up as part of an unbounded component.

[9] We are assuming that U^- is the closure of the intersection of the interior of H^- with U. In other words, we are using *regularized* set operations on the regions.

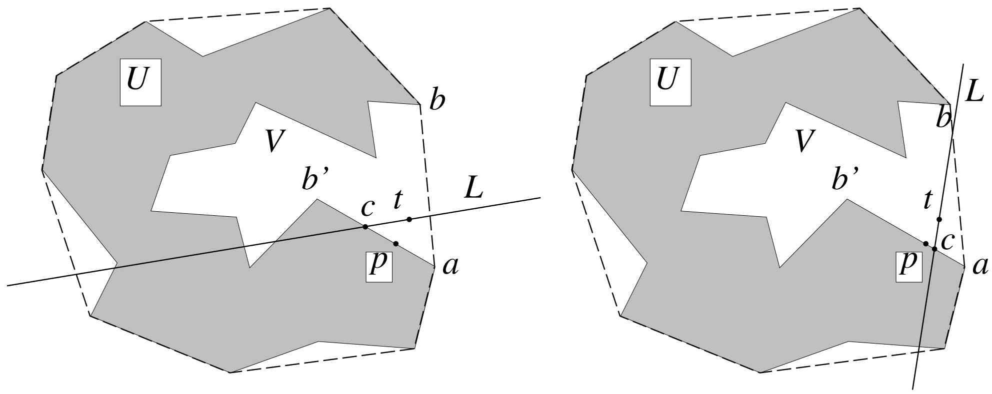

Figure 3: On the left, angle $act < 90$, and on the right, angle $act > 90$.

If the splitting line L passes through a vertex u of the bay, we are done: this non-hull vertex is an "extra" non-hull vertex that becomes a hull vertex of U^- and U^+. Let us suppose that L intersects U nearest to t in the interior of an edge $a'b'$. It suffices to show that $a' \neq a$ and $b' \neq b$. If this is so, then a' will be a non-hull vertex eliminated from U^+, and b' will be a non-hull vertex eliminated from U^-.

Suppose $a' = a$. It follows that $b' \neq b$; otherwise, there would be no bay. Let c be the intersection of L with ab'. Let p be the point of line ab' that is nearest to t. There are two cases to consider: 1) $\angle act$ is less than or equal to 90 degrees and 2) $\angle act$ is greater than 90 degrees. In each case, we will show a contradiction: distance-based subdivision would not have returned this particular L. See Figure 3.

1. If $\angle act \leq 90$, then p lies on the ray ca. Rotating L clockwise (moving c towards b'), does not change $d^-(t, \theta, U) = \min(|t-a|, |t-p|)$ and it does not decrease $d^+(t, \theta, U)$. This rotation decreases $\angle act$, and therefore it can continue until c reaches b or until L hits another vertex. If possible, the distance-based subdivision algorithm always returns L that passes through a vertex of the bay due to the "cleaning" step. Hence, it would have returned the rotated version L instead.

2. If $\angle act > 90$, then $d^-(t, \theta, U) = |t - c|$ and p belongs to $U^+(t, \theta, U)$ which means that,

$$d^+(t, \theta, U) \leq |t - p| < |t - c|.$$

Rotating L counterclockwise until c reaches p does not decrease the value of

$$\min(d^-(t, \theta, U), d^+(t, \theta, U)) = d^+(t, \theta, U).$$

This reduces the proof to the case that $\angle act \leq 90$. ■

Theorem 4.6 *The MP containment algorithm subdivides to* $\mathrm{O}(k^2m^4 + km^2n^2)$ *depth.*

Proof: The initial U_{ij}^{init}, $0 \leq i < j \leq k$ are generated according to Equation 5 on page 6. Polygon P_0, the complement of the container, has n vertices. Polygon P_i, $0 \leq i \leq k$, has $\mathrm{O}(m)$ vertices. Section 2.1 gives the known bounds on the complexity of the Minkowski sum, and these imply that $|U_{0i}^{\mathrm{init}}|$ is $\mathrm{O}(m^2n^2)$, $0 \leq i \leq k$, and $|U_{ij}^{\mathrm{init}}|$ is $\mathrm{O}(m^4)$, $1 \leq i < j \leq k$. These bounds imply that $\mathrm{NC}(\mathcal{U}^{\mathrm{init}})$ is $\mathrm{O}(k^2m^4 + km^2n^2)$.

Strictly speaking, for $1 \leq i < j \leq k$, U_{ij}^{init} is unbounded, and therefore Corollary 4.3 does not apply. However, we can intersect each U_{ij}^{init} with a sufficiently large square region (more than twice the width of the container). This will add only $O(k^2)$ vertices. The first MP restriction will eliminate the edges and vertices introduced by these squares.

By Corollaries 4.2 and 4.4, MP restriction does not increase non-convexity. Component subdivision (Case 1) clearly generates sub-hypotheses with diminished non-convexity, and by Lemma 4.5, so does distance-based subdivision (Cases 2 and 3). Only restriction and subdivision can change the hypothesis. When the algorithm reaches a leaf hypothesis $\mathcal{U}$, $\mathrm{NC}(\mathcal{U}) = 0$, each U_{ij} in $\mathcal{U}$ will be convex. If the (linear) relaxed constraints of Equation 6 are infeasible, this hypothesis will restrict to null. Otherwise, any solution to the the relaxed constraints is an exact solution to containment. ■

Theorem 4.6 implies that the MP containment algorithm examines at most $2^{O(k^2m^4+km^2n^2)}$ hypotheses. It is plausible that each subdivision roughly halves the non-convexity of some U_{ij}. If this were true, then the depth of the subdivisions would be

$$\frac{k(k-1)}{2}\log(O(m^4)) + k\log(O(m^2n^2)),$$

and the number of hypotheses examined would be,

$$2^{\frac{k(k-1)}{2}\log(O(m^4))+k\log(O(m^2n^2))} = O(m^{2k(k-1)} + (mn)^{2k}).$$

To improve the running time in practice, the actual implementation of MP containment also applies geometric restriction to the initial hypothesis (see Section 6). The number of vertices in the geometric restriction of the initial hypothesis is not necessarily in $O(k^2m^4+km^2n^2)$. Consequently, the non-convexity of the hypothesis at the root of the hypothesis tree (and therefore the depth of the tree) may be larger than $O(k^2m^4+km^2n^2)$. However, because we apply Equation 5 (on page 6) only once, the number of vertices in the geometric restriction of the initial hypothesis is still finite. As a result, the depth of the hypothesis tree is still finite and the algorithm terminates after a finite number of subdivisions.

4.4 Semi-Theoretical Subdivision

This section describes an alternative form of subdivision that provides a bound on the number of hypotheses–nodes in the subdivision tree–rather than a bound on the depth of the subdivision tree. In other work [24], we also show how to reduce the time to evaluate each hypothesis to $O(\log kmn)$. The latter is a very complicated algorithm and proof, and it is clearly impractical because it also involves a much weaker form of restriction. Basically, it can allow only one iteration of the MP restriction algorithm. The subdivision technique is also less practical than distance-based subdivision because it generate more components and because it does not use the results of evaluation. However, it is very simple, and we can describe it here.

Recall that U^{init} is the initial U_{ij} region in the root hypothesis. The "semi-theoretical" subdivision algorithm splits a hypothesis $\mathcal{U}$ as follows. It finds some pair i, j such that some portion of some edge e_{ij} of U_{ij}^{init} intersection the interior of $\mathrm{conv}(U_{ij})$. It uses the line L which contains this edge as a splitting line.

This algorithm must terminate because when there is no such edge e_{ij} for a hypothesis $\mathcal{U}$, it must follow that $\mathrm{conv}(U_{ij}) \subseteq U_{ij}^{\mathrm{init}}$, for $0 \le i < j \le k$. Any configuration $\langle t_0, t_1, ..., t_k \rangle$ which satisfies the relaxed constraint Equations 6 must therefore be a solution to containment.

Theorem 4.7 *For constant k, "semi-theoretical" subdivision visits* $\mathrm{O}\left((mn+m^2)^{2k}\right)$ *hypothesis.*

Proof: Each edge of each U_{ij}^{init} region corresponds to a contact between an edge or vertex of P_i with a vertex or edge of P_j. Given $|P_0| = n$ and $|P_i| \leq m$, $1 \leq i \leq k$, the number of possible cutting lines is $\mathrm{O}(mn + m^2)$. This proof shows there is a way of "labeling" each hypothesis with a list of $2k$ lines (actually linear inequalities based on these lines). The number of such labels is $\mathrm{O}\left((mn+m^2)^{2k}\right)$.

As before, we intersect each U_{ij} region with a sufficiently large square. These squares add only a constant ($\mathrm{O}(k^2)$) number of additional lines. The root hypothesis has associated with it the linear inequalities corresponding to these lines. When we split a hypothesis, we associate a new inequality with each child corresponding to the half-planes of the splitting line.

We also associate with each non-root hypothesis a list of $2k$ critical inequalities which correspond to a feasible point for the set of inequalities associated with that hypothesis. This list must include the newly added inequality. This rule does not cause a problem because we know that the splitting line L cuts through the interior of the set of feasible points C_{ij} for $t_j - t_i$.

We argue that no two hypotheses $\mathcal{U}'$ and $\mathcal{U}''$ receive the same label. If $\mathcal{U}''$ is a descendant of $\mathcal{U}'$, then $\mathcal{U}''$ has its newly added inequality in its list which $\mathcal{U}'$ cannot have. If $\mathcal{U}'$ and $\mathcal{U}''$ are "cousins", let $\mathcal{U}$ be their common ancestor. Let L be the splitting line for $\mathcal{U}$, and let U_{ij} be the region that is split. If $\mathcal{U}'$ and $\mathcal{U}''$ have the same label, then they have the same feasible configuration $\langle t_0, t_1, ..., t_k \rangle$. However, U'_{ij} and U''_{ij} lie on opposite sides of L, and therefore $t_j - t_i$ must lie on L. If so, then U'_{ij} has the "left inequality" of L in its list and U'_{ij} has the "right inequality" of L in its list. Therefore the two lists are not the same after all.

Note: the last argument requires an assumption of general position among the different splitting lines. We will not go into the arguments required to remove this assumption. ■

5 Enclosure Problems

5.1 Minimum Area Square Enclosure.

Given any containment algorithm, we can find the smallest square[10] that encloses translated copies of $P_1, P_2, \ldots, P_k$ [4]. For a numerical algorithm: use binary search on the size of the square. For a combinatorial algorithm: given a feasible square, use *compaction* [22, 23, 21] to "shrink" the square to a local minimum. Any more shrinking will guarantee either infeasibility (if it is the global minimum) or at least a change in combinatorial structure. One can use symbolic perturbation [15, 32] to shrink the square an infinitesimal amount and force infeasibility or a change in structure. Compact again and repeat. In practice, we would just compact the feasible squares and then continue the (numerical) binary search. Doubtless there is a combinatorial way to do binary search using parametric searching.

5.2 Minimum Area Rectangle.

One cannot use containment and binary search to find the minimum area rectangular enclosure (of *arbitrary* aspect ratio). We need to solve the following: given shapes $P_1, P_2, \ldots, P_k$ and area α, is

[10] Or indeed the smallest similar copy of any star-shaped polygon.

there a rectangle of area α which contains translated copies of the shapes? With this, one can find the minimum α numerically by binary search.

Given $P_1, P_2, \ldots, P_k$, let $P_0 = \{(x, y) \mid x \leq 0 \text{ or } y \leq 0\}$ and let $P_{k+1} = \{(x, y) \mid x \geq 0 \text{ or } y \geq 0\}$. Calculate U_{ij}, $0 \leq i, j \leq k+1$ according to Equation 5 (page 6). Since P_0 and P_{k+1} intersect no matter where you put them, $U_{0,k+1}$ will be empty. "Artificially" set $U_{0,k+1} = \{(x, y) \mid xy \leq \alpha\}$. We claim 1) the hypothesis $\mathcal{U}$ has a valid configuration if and only if there is a rectangle of area α enclosing $P_1, P_2, \ldots, P_k$, and 2) only one minor modification to our algorithm is required. If $t_{k+1} - t_0$ is outside $U_{0,k+1}$, then instead of taking the nearest edge, we take the nearest point on the hyperbola. We set up the overlap minimization linear program using the tangent line at that point. We have not tested this algorithm, but we expect it to have quadratic convergence once it "homes in" on a configuration.

6 Implementation and Results

We implemented the MP containment algorithm[11] using the CPLEX 2.1 linear programming system (CPLEX Optimization, Inc.).[12] Most of the examples we selected are from the apparel industry. We produced each feasible containment example from an existing apparel layout by removing a group of neighboring items. To create an infeasible example, we moved some of the remaining items a small distance into the container. No examples were dropped.

An example of the effect of our new MP restriction is shown in Figure 4. The figure shows two two-component U_{0j}s. The boundaries of the unrestricted U_{0j}s are outlined in black, and the restricted U_{0j}s are depicted as black filled regions.

We compared the performance of our new MP containment algorithm with two of our previous algorithms [9, 7]: the geometric algorithm and the naive LP algorithm (see Section 1.3). All three algorithms apply geometric restriction (Equation 7 on page 7) to the initial hypothesis (Equation 5 on page 6). Table 1 gives running times for feasible containment problems. Each problem is labeled with the number and type: for instance, "6CN" means that six convex (C) polygons were placed into a non-convex (N) container. Running times are in seconds on a 50 MHz SPARCstation[13]. Table 2 gives running times for infeasible containment problems. Our new algorithm clearly outperforms both of the other algorithms. For five or more polygons, the difference is dramatic[14]. Table 3 compares the number of hypotheses evaluated by the naive LP containment algorithm and the new MP containment algorithm on some infeasible examples with $k = 4$ and no restriction applied. This table clearly demonstrates that overlap minimization yields "smarter" subdivision. Figure 5 and Figure 6 depict configurations obtained by our algorithm. Finally, Figure 7 shows the minimal square for four polygons.

[11] The experiments used a earlier version of the evaluation algorithm[25] which does not have a guarantee that I_{ij} was a subset of U_{ij}. Consequently, it requires an extra interpolation step to ensure convergence to a local overlap minimum. However, evaluate does not take up the lions share of the running time (restriction does), and therefore one should expect only slightly faster results with the newer evaluation algorithm.

[12] Currently, we are using CPLEX 3.0, but the experiments were all run using CPLEX 2.1.

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

[14] Although the running time of MP containment is substantially less than that of geometric containment for $k > 4$, the number of hypotheses examined by geometric containment can be less than the number examined by MP containment, as discussed in Section 1.5.1. That discussion also points out that geometric restriction is currently much slower than MP restriction.

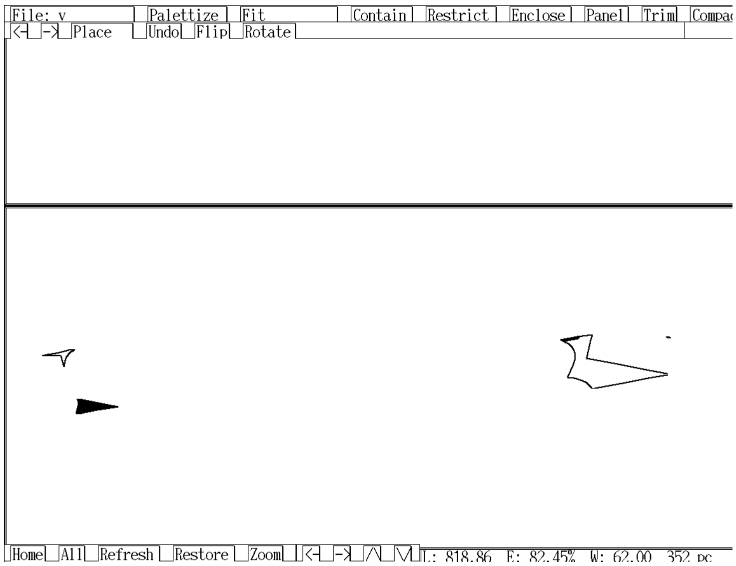

Figure 4: MP restriction of two two-component U_{0j}s

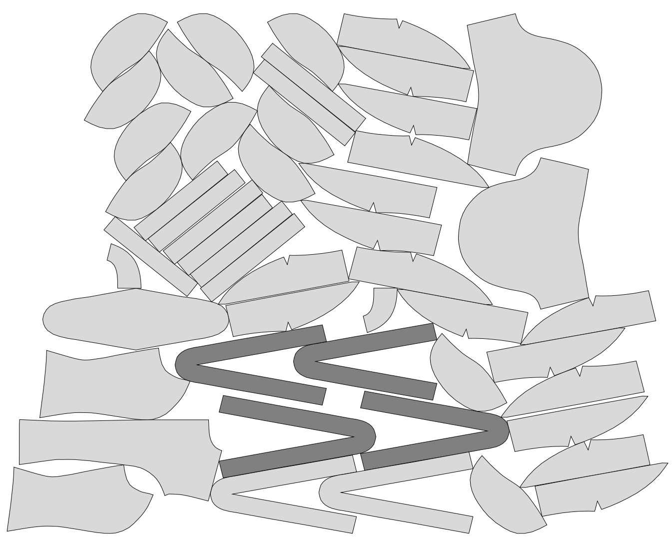

Figure 5: 4NN example: time = 15 seconds

Type	GEOM	NAIVE LP	MP
3NN	19	22	9
3NN	2	4	4
4CC	25	1	2
4CC	21	10	11
4CN	113	84	4
4NN	155	19	33
4NN	80	24	35
4NN	35	43	15
4NN	319	89	10
5CC	126	6	3
5NN	> 300	> 300	24
6CC	75	53	5
6CN	> 300	> 300	184
6NN	> 300	> 300	86
6NN	> 900	> 900	593
7CC	823	162	7
7CN	> 300	> 300	72
7NN	> 600	> 600	176
8CC	> 300	> 300	34
10CC	-	-	14
10NN	-	-	183

Table 1: Running times for feasible examples in seconds

Type	GEOM	NAIVE LP	MP
4NN	47	152	135
4NN	138	3037	493
4NN	> 600	> 600	416
5NN	> 600	> 600	103

Table 2: Running times for infeasible examples in seconds

Type	NAIVE LP	MP	Type	NAIVE LP	MP
4CN	32	27	4CN	42	11
4NN	493	153	4NN	1736	311
4NN	153	103	4NN	436	323

Table 3: Number of hypotheses evaluated for infeasible problems (no hypothesis restriction)

Figure 6: 6NN example: time = 126 seconds

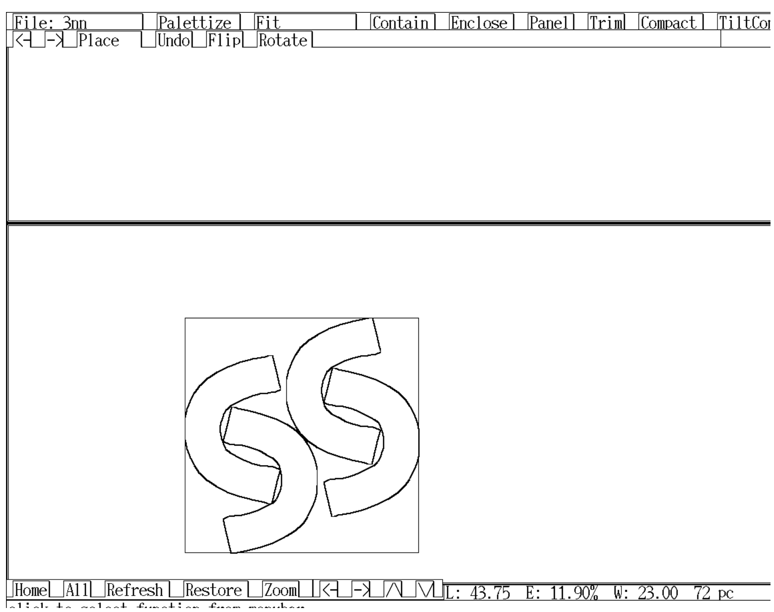

Figure 7: Minimal enclosing square of four polygons, each with 65 vertices: time = 122 seconds (binary search only–no compaction).

7 Conclusion

This paper has presented a new algorithm for two-dimensional translational containment based on mathematical programming principles. It also gives algorithms, based on the same approach, for finding the minimal enclosing square and the minimal area enclosing rectangle for a collection of translating polygons. All of these algorithms provide exact solutions.

In our experiments, our new containment algorithm clearly outperforms purely geometric containment algorithms. For data sets from the apparel industry, it can solve containment for up to ten non-convex polygons in a non-convex container in practice. This important practical result is achieved by following our restrict/evaluate/subdivide algorithmic paradigm, whose steps correspond to tightening upper and lower bounds on the amount of overlap in a layout and introducing cutting planes when necessary. Our new algorithms for the individual steps in the paradigm use linear programming as well as techniques from computational geometry.

Combining the power of linear programming with techniques from computational geometry gives us a translational containment "tool" which can be used to perform many tasks related to layout, including ones for which each item can have a discrete set of orientations. For example, we can pack multiple containers by first generating groups which fit into each container and then matching groups to containers [4]. This type of problem arises in a variety of settings, such as in the second phase of a two-phase layout problem. Recently, we used containment to generate entire layouts, one column at a time [5].

In addition to using the containment tool directly, one can apply our *approach* to containment to solve a variety of previously unsolved problems. Our solutions to minimal enclosure problems in this paper and Milenkovic's recent results on finding minimal area convex hulls [24] illustrate this. Another example is the work we have begun on finding dense lattice packings of multiple shapes[15] under translation.

We believe that combining mathematical programming with computational geometry is a fruitful approach for tackling many NP-hard geometric optimization problems. Our work with linear programming and our brief experiments so far with mixed-integer programming are just the beginning of our exploration. Recent results in integer programming may well provide a starting point for our future work.

References

[1] M. Adamowicz and A. Albano. Nesting two-dimensional shapes in rectangular modules. *Computer-Aided Design*, 8:27–33, 1976.

[2] F. Avnaim. *Placement et déplacement de formes rigides ou articulées.* Thèse de doctorat en sciences, Université de Franche-Comté, Besançon, France, 1989.

[3] F. Avnaim and J.-D. Boissonnat. Simultaneous containment of several polygons. In *Proc. 3rd Annu. ACM Sympos. Comput. Geom.*, pages 242–250, 1987.

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

[15] We assume there are many copies of each different shape.

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

[6] K. Daniels and V. Milenkovic. Distance-based subdivision for translational LP containment. In *Proc. 8th Canad. Conf. Comput. Geom.*, pages 196–201, 1996.

[7] K. Daniels and V. J. Milenkovic. Multiple Translational Containment, Part I: An Approximate Algorithm. *Algorithmica*, 19:148–182, 1997.

[8] K. Daniels and V. J. Milenkovic. Multiple Translational Containment, Part I: An Approximate Algorithm. *Algorithmica*, 19:148–182, 1997.

[9] Karen Daniels and Victor Milenkovic. Multiple translational containment: Approximate and exact algorithms. In *Proc. 6th ACM-SIAM Sympos. Discrete Algorithms (SODA '95)*, pages 205–214, 1995.

[10] O. Devillers. Simultaneous containment of several polygons: analysis of the contact configurations. *Internat. J. Comput. Geom. Appl.*, 3(4):429–442, 1993.

[11] D. Dobkin, J. Hershberger, D. Kirkpatrick, and S. Suri. Computing the intersection-depth of polyhedra. *Algorithmica*, 9:518–533, 1993.

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

[13] Kathryn A. Dowsland and William B. Dowsland. Solution Approaches to Irregular Nesting Problems. *European Journal of Operational Research*, 84:506–521, 1995.

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

[15] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9:66–104, 1990.

[16] S. J. Fortune. A fast algorithm for polygon containment by translation. In *Proc. 12th Internat. Colloq. Automata Lang. Program.*, volume 194 of *Lecture Notes in Computer Science*, pages 189–198. Springer-Verlag, 1985.

[17] P. Gilmore and R. Gomory. Multistage Cutting Stock Problems of Two and More Dimensions. *Operations Research*, 13:94–120, 1965.

[18] L. J. Guibas, L. Ramshaw, and J. Stolfi. A kinetic framework for computational geometry. In *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 100–111, 1983.

[19] M. Haims and H. Freeman. A multistage solution of the template-layout problem. *I.E.E.E. Transactions on Systems Science and Cybernetics*, 6:145, 1970.

[20] A. Kaul, M.A. O'Connor, and V. Srinivasan. Computing Minkowski Sums of Regular Polygons. In *Proceedings of the 3rd Canadian Conference on Computational Geometry*, Vancouver, British Columbia, 1991.

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

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

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

[24] V. Milenkovic. Translational polygon containment and minimal enclosure using linear programming based restriction. In *Proc. 28th Annu. ACM Sympos. Theory Comput. (STOC 96)*, pages 109–118, 1996.

[25] V. Milenkovic and K. Daniels. Translational polygon containment and minimal enclosure using geometric algorithms and mathematical programming. Technical Report 25-95, Center for Research in Computing Technology, Division of Applied Sciences, Harvard University, Cambridge, MA, 1995.

[26] V. Milenkovic, K. Daniels, and Z. Li. Automatic marker making. In *Proc. 3rd Canad. Conf. Comput. Geom.*, pages 243–246, 1991.

[27] V. J. Milenkovic. Multiple Translational Containment, Part II: Exact Algorithms. *Algorithmica*, 19:183–218, 1997.

[28] H. Minkowski. Volumen und Oberfläche. *Mathematische Annalen*, 57:447–495, 1903.

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

[30] J. Serra, editor. *Image Analysis and Mathematical Morphology*, volume 2: Theoretical Advances. Academic Press, New York, 1988.

[31] 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.

[32] C. K. Yap. A geometric consistency theorem for a symbolic perturbation scheme. *J. Comput. Syst. Sci.*, 40:2–18, 1990.