Algorithms for Fence Design

Robert-Paul Berretty *University of Utrecht, Utrecht, The Netherlands*
Ken Goldberg *University of California at Berkeley, CA, USA*
Mark H. Overmars *University of Utrecht, Utrecht, The Netherlands*
A. Frank van der Stappen *University of Utrecht, Utrecht, The Netherlands*

Abstract

A common task in automated manufacturing processes is to orient parts prior to assembly. We address sensorless orientation of a polygonal part on a conveyor belt by a sequence of stationary fences across this belt. Since fences can only push against the motion of the belt, it is a challenging problem to compute fence designs which orients a given part. In this paper, we give several polynomial-time, algorithms to compute fence designs which are optimal with respect to various criteria. We address both frictionless and frictional fences. We also compute modular fence designs in which the fence angles are restricted to a discrete set of angles instead of an interval.

1 Introduction

Many automated manufacturing processes require parts to be oriented prior to assembly. A part feeder takes in a stream of identical parts in arbitrary orientations and outputs them in a uniform orientation. Part feeders often use data obtained from some kind of sensing device to accomplish their task. We consider the problem of *sensorless orientation* of parts, in which the initial pose of the part is assumed to be unknown. In sensorless manipulation, parts are positioned and/or oriented using passive mechanical compliance. The input is a description of the part shape and the output is a sequence of open-loop actions that moves a part from an unknown initial pose into a unique final pose. Among the sensorless part feeders considered in literature are the parallel-jaw gripper [9, 13], a single pushing jaw [2, 14, 15, 17], a conveyor belt with a sequence of (stationary) fences placed along its sides [7, 18, 21], a conveyor belt with a single rotational fence (1JOC) [1], a tilting tray [12, 16], and vibratory plates and programmable vector fields [5, 6].

The pushing jaw [2, 14, 15, 17] orients a part by an alternating sequence of pushes and jaw reorientations. The problem of sensorless orientation by a pushing jaw is to find a sequence of push directions that will move the part from an arbitrary initial orientation into a single known final orientation. Such a sequence is referred to as a *push plan*. Goldberg [13] showed that any polygonal part can be oriented by a sequence of pushes. Chen and Ierardi [9] proved that any polygonal part with n vertices can be oriented by $O(n)$ pushes.

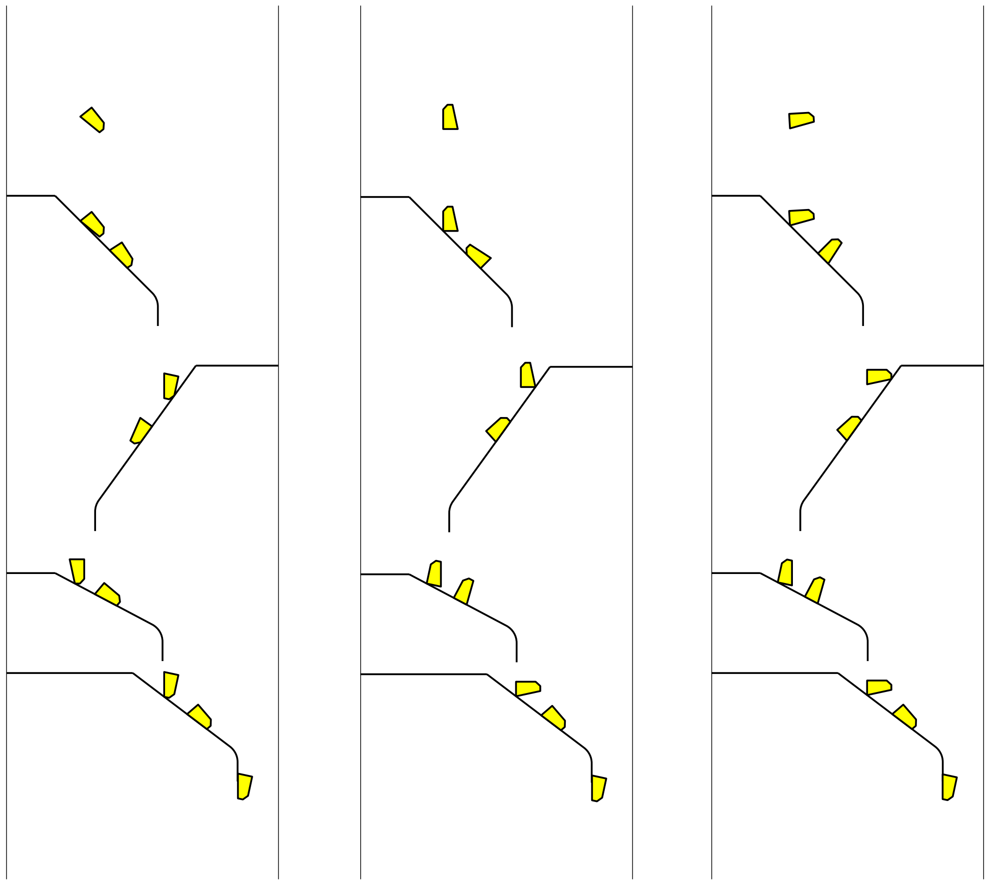

Figure 1: Three overhead views of the same conveyor belt and fence design. The traversals for three different initial orientations of the same part are displayed. The traversals show that the part ends up in the same orientation in each of the three cases.

They showed that this bound is tight by constructing (pathological) n-gons that require $\Omega(n)$ pushes to be oriented. Goldberg gave an $O(n^2)$ algorithm for computing the shortest push plan for a polygon.

The problem of *fence design* is to determine a sequence of fence orientations (see Figure 1) such that the fences with these orientations align the part as it moves down a conveyor belt and slides along these fences [7, 18, 21]. The motion of the belt effectively turns each slide into a push action by the fence in the direction normal to the fence. The fact that the direction of the push, i.e., the normal at the fence, must have a non-zero component in the direction opposite to the motion of the belt imposes a restriction on successive push directions. Fence design can be regarded as finding a constrained sequence of push directions (see Subsection 2.2 for the actual constraints). The additional constraints make fence design considerably more difficult than sensorless orientation by a pushing jaw. Wiegley *et al.* [21] conjectured that a fence design exists for any polygonal part. They gave an exponential algorithm for computing the shortest sequence of fences for a given part.

In [4], we proved the conjecture that a fence design exists for any polygonal part. A more eleborate version of the proof can be found in [3]. Also, we gave an $O(n^3 \log n)$ algorithm for computing a fence design of minimal length (in terms of the number of fences used). In this paper, we first review this algorithm, which is easy to implement. The program can be tuned to take into account certain quality requirements on the fence design, like minimum and maximum (successive) fence angles to prevent impractical steep and shallow fences and a long series of fences on a single side of the belt, which would require an impractically wide conveyor belt. The cost of the incorporation of quality measures is an increase of the algorithm's running time to $O(n^4)$.

The algorithm forms the basis for the new algorithms to compute fence designs, which we present in this paper. The graph based algorithm can be modified to compute frictional fence designs, i.e. when there is friction between the part and the fences. Also, we can change the algorithm to compute fence designs which use modular fences of fixed angles. The running time of the algorithm is then $O(mn^2)$, with m the number of available fence angles.

In [4], we showed that fence designs of length $O(n)$ exist for a large class of parts. In this paper, we give a new, output-sensitive algorithm which computes the shortest fence design. This algorithm runs in time $O(kn \log n)$, with k the number of fences in the design. We can adopt this output sensitive algorithm to compute modular fence designs in similar time bounds.

In the first part of the paper, we assume zero friction between the part and the fences. Later, we incorporate Coulomb friction (see e.g. [15]) between the part and the fence. The friction between the part and the belt is assumed to be uniform. We assume quasi-static motion, i.e. the belt speed is low enough so the parts do not bounce when they hit a fence. Since any push action acts on the convex hull of the part rather than on the part itself, we assume without loss of generality that the part under consideration is convex. The fences are assumed to be long enough so that the part rotates into a stable pose [17]. Furthermore, we assume that the the parts do not have meta-stable edges, i.e. the perpendicular projection of the center-of-mass on an edge does not intersect a vertex of the edge.

This paper is organized as follows. In Section 2, we first review the key notion of a push plan, and identify the constraints on the relative push angles for fence design. Section 3 reviews the $O(n^3 \log n)$ algorithm for computing the shortest fence design for a part. In Section 4 we present the output sensitive algorithm which runs in $O(kn \log n)$ time. In Section 5 we investigate frictional fences, and in Section 6 we look at modular fences. In Section 7 we draw some conclusions and give several interesting open questions.

2 Push Plans and Fence Designs

2.1 The push function

In this section we focus on the push function of a part. Let P a convex polygonal part with n vertices and center-of-mass c. We assume that a fixed coordinate frame is attached to P. Directions are expressed relative to this frame. The contact direction of a supporting line l of P is uniquely defined as the direction of the normal of l pointing into P. The radius function $r : [0, 2\pi) \to \mathbb{R}^+$ maps a direction ϕ onto the distance from c to the supporting line of P with contact direction ϕ. For a polygonal part, the radius function is a continuous piecewise sinusoidal function, and can be computed in $O(n)$ time by checking each vertex [15]. The final orientation of a part that is being pushed can be determined from its radius function.

In most cases, parts will start to rotate when pushed. If pushing in a certain direction does *not* cause the part to rotate, then the contact normal at one of its points of contact with the jaw passes through the center-of-mass [15]. We refer to the corresponding direction of the contact normal as an *equilibrium* push direction or orientation. If pushing does change the orientation, then this rotation changes the orientation of the pushing device relative to the part. We assume that pushing continues until the part stops rotating and settles in a stable equilibrium pose, which is an equilibrium with a preimage of non-zero length.

The *push function* $p : [0, 2\pi) \to [0, 2\pi)$ links every orientation ϕ to the orientation $p(\phi)$ in which the part P settles after being pushed by a jaw with initial contact direction ϕ (relative to the frame attached to P). The rotation of the part due to pushing causes the contact direction of the jaw to change. The final orientation $p(\phi)$ of the part is the contact direction of the jaw after the part has settled. The equilibrium push directions are the fixed points of p.

The push function p of a polygonal part consists of steps, which are intervals $I \subset [0, 2\pi)$ for which $p(\phi) = C$ for all $\phi \in I$ and some constant $C \in I$. The steps of the push function are easily constructed from the radius function r. If the part is pushed in a direction corresponding to a point of non-horizontal tangency of the radius function then the part will rotate in the direction in which the radius decreases. The part finally settles in an orientation corresponding to a local minimum of the radius function. As a result, all points in the open interval I bounded by two consecutive local maxima of the radius function r map onto the orientation $\phi \in I$ corresponding to the unique local minimum of r on I.

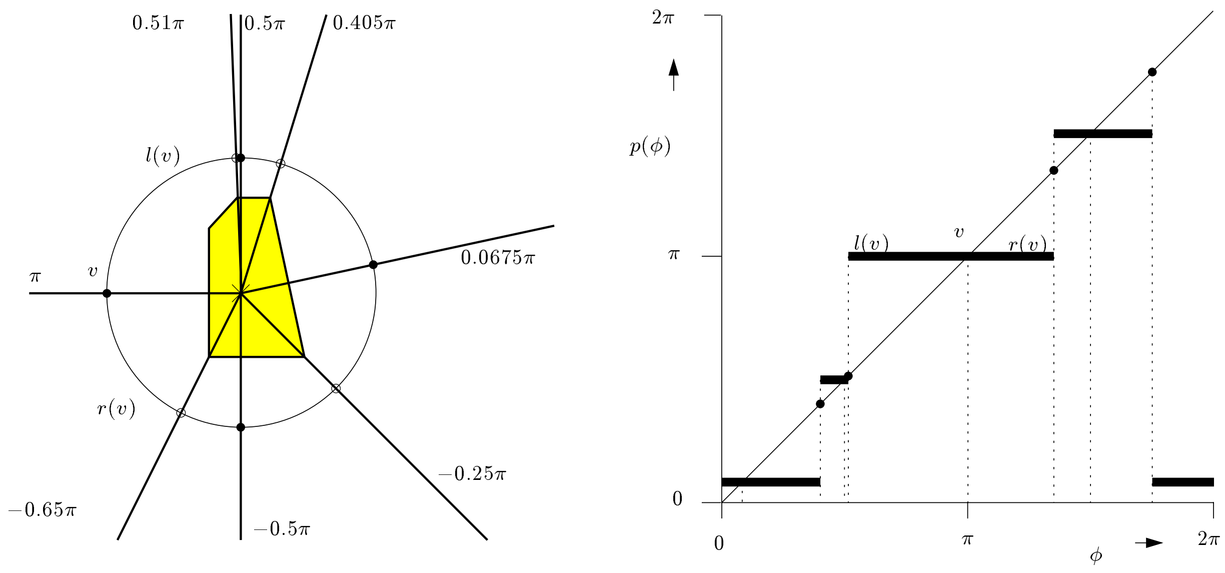

Figure 2: A polygonal part and its push function. The minima correspond to normals to polygon edges that intersect the center-of-mass. The maxima correspond to tangents to polygon vertices whose normals intersect the center-of-mass. The horizontal steps of the push function are angular intervals between two successive maxima.

(Note that ϕ itself maps onto ϕ because it is a point of horizontal tangency.) This results in the steps of the push function. In preparation for the next sections, we define two open intervals $l(v) = \{\phi < v | p(\phi) = v\}$ and $r(v) = \{\phi > v | p(\phi) = v\}$ for each fixed point—or equilibrium orientation—v of the push function. We refer to these intervals as v's left and right environment respectively. The interval $l(v)$ corresponds to the half-step left of $v = f(v)$ and $r(v)$ corresponds to the half-step right of $v = f(v)$ (see Figure 2). Note that an equilibrium v is stable if $l(v) \cup r(v) \neq \emptyset$. We denote by $\mathcal{A} = a_0, \ldots, a_{m_s-1}$ the cyclic sequence of stable equilibria, cut at some arbitrary orientation, and ordered by increasing angle. Besides the steps, there are isolated points satisfying $p(\phi) = \phi$ in the push function, corresponding to local maxima of the radius function. Figure 2 shows a polygonal part and its push function. The final orientation of the part while being pushed is described by the push function. In [3] we showed that any step function having only non-zero-length half-steps represents a polygonal part, and is therefore a valid push function.

A push function p is said to have period d if $p(\phi) = p((\phi + d) \bmod 2\pi)$ for all $\phi \in [0, 2\pi)$. Any part can only be oriented up to (periodic) symmetry in its push function. The sequence $\mathcal{A}$ of stable equilibra is said to have cycle d if and only if $|r(a_i)| = |r(a_{i+d})|$, and $|l(a_i)| = |l(a_{i+d})|$ for all $0 \leq i < m_s$. (Indexing is modulo m_s.)

We use the abbreviation p_α to denote the (shifted) push function defined by

$$p_\alpha(\phi) = p((\phi + \alpha) \bmod 2\pi),$$

for all $\phi \in [0, 2\pi)$. Note that $p_\alpha(\phi)$ is the final orientation of a part in initial orientation ϕ after a reorientation by α followed by a push. We can now define a *push plan*.

Definition 1 *A* push plan *is a sequence* $\alpha_1, \ldots, \alpha_m$ *such that* $p_{\alpha_m} \circ \ldots \circ p_{\alpha_1}(\phi) = \Phi$ *for all* $\phi \in [0, 2\pi)$ *and some fixed* $\Phi \in [0, 2\pi)$.

An important property of the push function is monotonicity, or the order preserving property [16]. A sequence $s_1, \ldots, s_n$ of elements of a set S is *ordered* if $s_1, \ldots, s_n$ are encountered in order when the generating cycle of S is traced *once*, starting from s_1. A function $p : S \to S$ is *monotonic* if for any ordered sequence $s_1, \ldots, s_n$ the sequence $p(s_1), \ldots, p(s_n)$ is also ordered.

2.2 Fence design

In this section we address the problem of designing a series of fences $f_1, \ldots, f_m$ that will orient P when it moves down a conveyor belt and slides along these fences $f_1, \ldots, f_m$. Let us assume that the conveyor belt moves from top to bottom, as indicated in the overhead view in Figure 4. We distinguish between left fences, which are placed along the left belt side, and right fences, which are placed along the right side. The fence angle β_i of a fence f_i denotes the angle between the upward pointing vector opposing the motion of the belt and the normal to the fence with a positive component in upward direction. The motion of the belt turns the sliding of the part along a fence into a push by the fence. The direction of the push is—by the zero friction assumption—orthogonal to the fence with a positive component in the direction opposing the motion of the belt. Thus, the motion of the belt causes any push direction to have a positive component in the direction opposing the belt motion. We now transform this constraint on the push direction relative to the belt into a constraint on successive push directions relative to the part.

Sliding along a fence f_i causes one of P's edges e to align with the fence. The curved tip of the fence [7] guarantees that e is aligned with the belt sides as P leaves the fence. If f_i is a left fence then e faces the left belt side (see Figure 3). If f_i is a right fence, it faces the right side. Assume f_i is a left fence. At the moment of leaving f_i, hence, after the push, the contact direction of f_i is perpendicular to the belt direction and towards the right belt side. So, the reorientation of the push is expressed relative to this direction. Figure 3 shows that the reorientation α_{i+1} is in the range $(0, \pi/2)$ if we choose f_{i+1} to be a left fence. If we take a right fence f_{i+1} then the reorientation is in the range $(\pi/2, \pi)$. A similar analysis can be done when P leaves a right fence and the edge e faces the left belt side. The result is given in the following table.

f_i	$\alpha_{i+1} \in I_{f_i, f_{i+1}}$	f_{i+1}
left	$(0, \pi/2)$	left
left	$(\pi/2, \pi)$	right
right	$(-\pi, -\pi/2)$	left
right	$(-\pi/2, 0)$	right

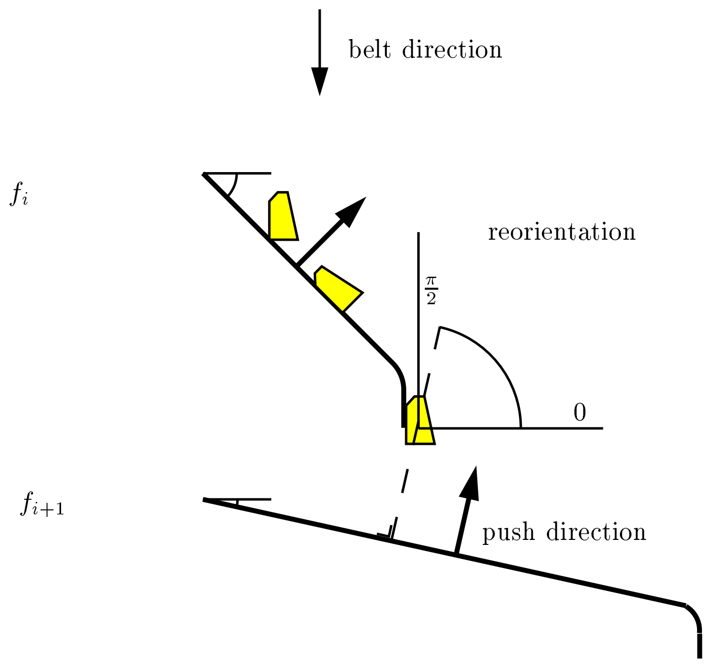

Figure 3: For two successive left fences, the reorientation of the push direction lies in the range $(0, \pi/2)$.

The table shows that the type of fence f_i imposes a bound on the reorientation α_{i+1}. Application of the same analysis to fences f_{i-1} and f_i and reorientation α_i leads to the following definition of a valid fence design [21].

Definition 2 *A* fence design *is a push plan $\alpha_1, \ldots, \alpha_m$ satisfying for all $1 \leq i < m$:*

- $\alpha_i \in (0, \frac{\pi}{2}) \cup (-\pi, -\frac{\pi}{2}) \Rightarrow \alpha_{i+1} \in (0, \frac{\pi}{2}) \cup (\frac{\pi}{2}, \pi)$
- $\alpha_i \in (-\frac{\pi}{2}, 0) \cup (\frac{\pi}{2}, \pi) \Rightarrow \alpha_{i+1} \in (-\frac{\pi}{2}, 0) \cup (-\pi, -\frac{\pi}{2})$.

The push plan on the left in Figure 4 satisfies the constraints of Definition 2, and is therefore also a fence design.

The relation between push plans and fence design immediately gives a worst case lower bound on the length of fence designs of $\Omega(n)$, which is presented in the paper of Chen and Ierardi [9]. The left and right environments of the stable equilibria of the parts of [9] which lead to this lower bound have the same length, with only a few exceptions. In their paper, Chen and Ierardi also give a linear upper bound on the length of a general push plan. Unfortunately, in general, this upper bound does not carry over to fence design, since the reorientations of the jaw do not satisfy the constraints of fence design. For a large class of parts, of which the lengths of the left and right environments are unique to some extent, the linear upper bound is valid for fence design, though.

3 An Algorithm for Computing Fence Designs

In this section we give an algorithm [4] which enables us to compute fence designs in time $O(n^3 \log n)$. This algorithm is based upon a discretization of the fence design problem, which leads to a polynomial-size graph representing the fence design problem. A search in this graph gives a feasible fence design. The algorithm is used as a basis for Sections 4 to 6. In the Section 3.1, we discuss the algorithm. In Section 3.2 its implementation.

3.1 A graph based algorithm

As every fence puts the part in a *stable* equilibrium orientation, the part is in one of these $m_s = O(n)$ orientations as it travels from one fence to another. This transforms the fence design problem into a discrete problem. After a first fence, the part can be in any of the stable orientations $a_0, \ldots, a_{m_s-1}$. The problem is to reduce the set of possible orientations of P to one stable orientation a_i by a sequence of fences. We build a directed graph on all possible *states* of the part as it travels from one fence to a next fence. A state consists of a set of possible orientations of the part plus the type (left or right) of the last fence, as the latter imposes a restriction on the reorientation of the push direction. Although there are $O(2^{m_s})$ subsets of $\{a_0, \ldots, a_{m_s-1}\}$, we shall see that we can restrict ourselves to subsets consisting of intervals of adjacent stable equilibria. Any such interval can be represented by the closed interval s defined by the first and last stable orientation a_i and a_j of the interval. The resulting graph has $O(m_s^2)$ nodes.

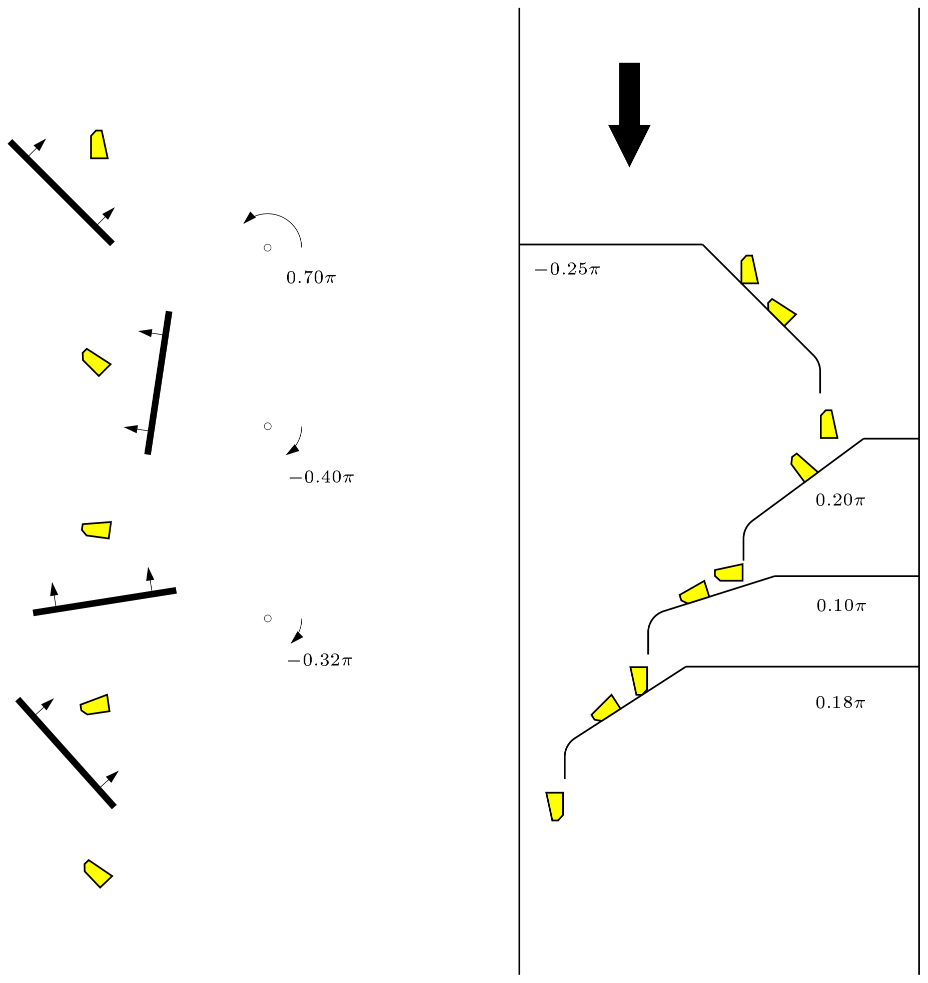

Figure 4: The left picture shows a plan for a pushing jaw. The jaw reorientations are (from top to bottom) 0.70π, -0.40π, and -0.32π. The conveyor belt on the right orients the same part. The fence angles, which are measured relative to the upward vector opposing the direction of belt motion, are -0.25π, 0.20π, 0.10π, and 0.18π. (Note that the sequence of fence orientations is not the same as the sequence of orientations of the pushing jaw because the curved fence tip rotates the stable edge to become aligned with the belt direction.)

Consider two graph nodes (s,t) and (s',t'), where $s = [a_i, a_j]$ and s' are intervals of stable equilibria and t and t' are fence types. Let $I_{t,t'}$ be the open interval (of length $\pi/2$) of reorientations admitted by the successive fences of types t and t' according to the table in Subsection 2.2. There is a directed edge from (s,t) to (s',t') if there is an angle $\alpha \in I_{t,t'}$ such that a reorientation of the push direction by α followed by a push moves any stable orientation in s into a stable orientation in s'. To check this condition, we determine the preimage $(u,w) \supseteq s'$ of s' under the (monotonic) push function. Observe that if $|s| = a_j - a_i < w - u$, any reorientation in the open interval $r = (u - a_i, w - a_j)$ followed by a push will map s into s'. We add an edge from (s,t) to (s',t') if the intersection of r and the interval $I_{t,t'}$ of admitted reorientations is non-empty, and label this edge with $r \cap I_{t,t'}$. For convenience, we add a source and a sink to the graph. We connect the source to every node (s,t) in the graph for which s contains all m_s stable equilibria, and we connect every node (s,t) with $s = [a_i, a_i]$ to the sink, if the part has no symmetry in its push function. If the parts push function is symmetric, then the parts stable orientations have a cycle. Let d denote the cycle of the parts push function. The part is oriented if the interval of possible orientations is $[a_i - d, a_i]$, for any $i \in [0, \ldots, m_s]$ (indexing is modulo m_s). For symmetric parts, we connect these nodes to the sink. Every path from the source to the sink now represents a fence design; a fence design of minimum length corresponds to the shortest such path and can be found by a breadth-first search from the source.

Theorem 3 *The shortest path in the graph corresponds to the shortest fence design, if a fence design exists.*

Proof: We prove that each path corresponds to a fence design and vice versa. The theorem follows immediately if we realize that every edge in the path corresponds to a fence in the design.

($\Rightarrow$) If there is a path, we must prove that there is a corresponding fence design. Since there is an edge from (s,t) to (s',t') if and only if the successive fences t and t' allow for a reorientation that maps s into s', this follows immediately from the construction of the graph.

($\Leftarrow$) If there is a fence design, we prove there is a path in the graph that represents this fence design. Let $f_1, \ldots, f_N$ be a fence design. We track the possible orientations of the fence design, and prove by induction that for every set of possible orientations, there is a node in the graph, and furthermore, there is a path from the source to such a node. Let A_i denote the set of all possible orientations of P leaving f_i. It is easy to see that for each A_i there are multiple nodes that include the set of possible orientations.

After the first fence f_1, all stable orientations are possible. Since we added edges from the source to all nodes containing all stable orientations, these nodes are reachable.

We now assume that for fence f_i having type t in our fence design the nodes (s,t) with $s \supseteq A_i$ are reachable from the source. Let t' be the fence type of f_{i+1}. Let (s',t') be a node such that $s' \supseteq A_{i+1}$. Let (u,w) denote the preimage of s'. Since the push function is monotonic and by existence of the fence design which maps A_i onto A_j, there is an admitted reorientation α_{i+1} by f_{i+1} such that $(u - \alpha_{i+1}, w - \alpha_{i+1}) \supset A_i$. Therefore, let (s,t) be a node such that $(u - \alpha_{i+1}, w - \alpha_{i+1}) \supset s \supseteq A_i$. There is an edge from (s,t) to

(s', t'), and there is a path from the source to (s, t). Since $s' \supseteq A_{i+1}$ is arbitrary, all (s', t') with $s' \supseteq A_{i+1}$ are reachable from the source. □

An important observation is that some graph edges are redundant if we are just interested in a fence design of minimum length. Consider a node (s, t) and all its outgoing edges to nodes $(s' = [a_i, a_j], t')$ for a fixed left endpoint a_i and a fixed fence type t'. We show that only one of these outgoing edges is sufficient. The following lemma is the key to this result.

Lemma 4 *Let (s, t), (s', t'), and (s'', t') be nodes with $s' = [a_i, a_j]$ and $s'' = [a_i, a_k]$ and let $s' \supset s''$. If there are edges from (s, t) to both (s', t') and (s'', t'), then the edge from (s, t) to (s', t') can be deleted without affecting the length of the shortest path in the graph.*

Proof: Assume that τ is a path from source to sink containing the edge $((s, t), (s', t'))$ and assume $((s', t'), (s''', t'''))$ succeeds this edge in τ. Because $s' \supset s''$, there must also be an edge $((s'', t'), (s''', t'''))$ in the graph. Hence, we can replace the edges $((s, t), (s', t'))$ and $((s', t'), (s''', t'''))$ in τ by $((s, t), (s'', t'))$ and $((s'', t'), (s''', t'''))$ without affecting the length of τ. □

The repeated application of Lemma 4 to the graph (until no more edges can be deleted) leads to a reduced graph in which every node has just one outgoing edge per set of nodes with intervals with a common left endpoint and with a common fence type. The single edge from the initial graph that remains after the repeated application of Lemma 4 is the one to the node corresponding to the shortest interval. Since there are $O(m_s) = O(n)$ possible left endpoints and just two fence types, the number of outgoing edges from one node is reduced to $O(n)$. The total number of edges of the reduced graph is therefore $O(n^3)$.

Theorem 5 *The reduced graph contains $O(n^2)$ nodes and $O(n^3)$ edges.*

The (reduced) graph can be constructed in the following way. First we compute the push function and store it in such a way that preimages can be found in $O(1)$ time. For each node (s, t), left endpoint a_i, and fence type t', we must determine the shortest interval $s' = [a_i, a_j]$ such that an edge exists between (s, t) and (s', t'). We can do this by a binary search on j. Since checking whether an edge exists between a pair of nodes corresponds to computing the preimage of an interval (which can be done in constant time), this binary search takes $O(\log n)$ time. As a similar binary search must be performed for each combination of a node (s, t), a left endpoint a_i, and a fence type t', the total time required to compute the graph edges is $O(n^3 \log n)$. A breadth-first search of the graph takes $O(n^3)$ time (see e.g. [10]). This yields the following theorem.

Theorem 6 *The computation of the optimal fence design for a polygonal part P with n vertices takes $O(n^3 \log n)$ time.*

Let τ be a path in the graph from the source to the sink. Every edge of τ corresponds to a non-empty angular interval of possible reorientations of the push direction. We simply pick the midpoint of every such interval as the reorientation, and get a push plan which is a fence design. We can easily compute the fence angles from the reorientation angles on the path, using the properties of the fence framework [21].

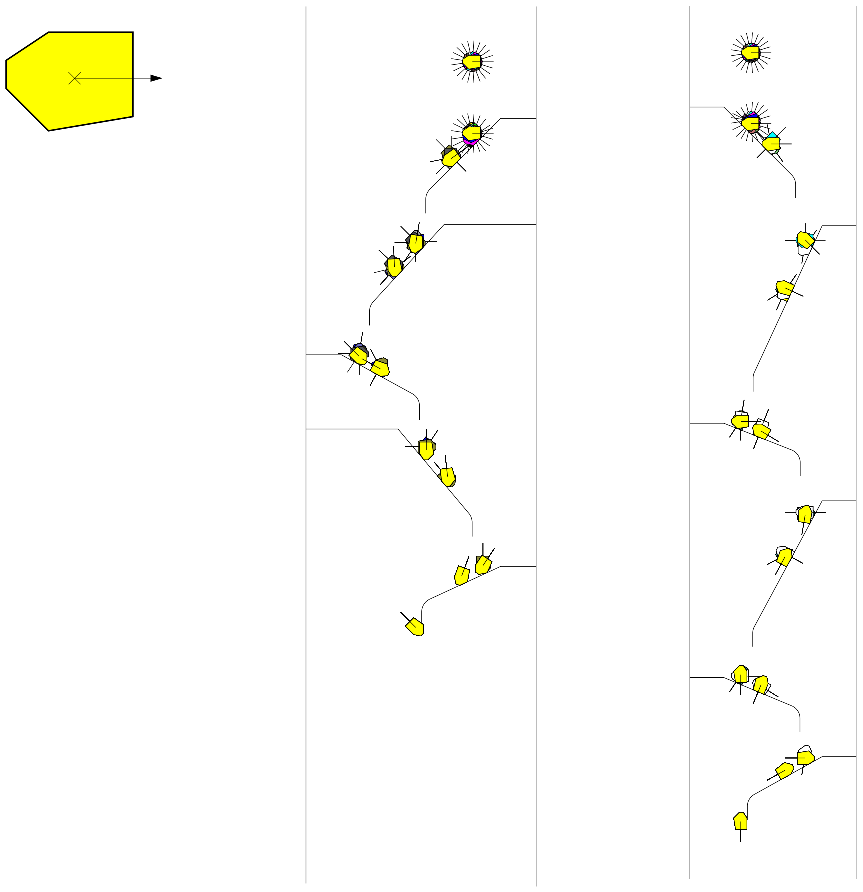

Figure 5: An optimal design of five fences, and a design of six alternating fences allowing for a narrower belt. Every line segment emanating from the part represents a possible orientation of the part.

3.2 Implementation

We implemented the above algorithm to test its behavior in practice. This turned out to be rather easy, using only some basic geometric computations for the push function, and some standard graph algorithms. The resulting code is rather fast; it returned fence designs within a fraction of a second for all parts we tried. (It should be noted that all parts we used here have only ten edges. When the number of edges goes up the cubic behavior of the algorithm will blow up the time required.) All fence designs shown in this paper were generated by the program. Our implementation offers the user the additional possibility of adding costs to graph edges. By doing so, the user can prevent the algorithm from outputting certain types of fence designs. Assigning high costs to edges between any pair of nodes of the same fence type t, for example, will cause the algorithm to output a sequence of alternating (left and right) fences if such a sequence exists. Alternating sequences are often preferred over sequences containing cascades of left (or right fences), as they generally allow for narrower conveyor belts (see Figure 5). Different cost assignments can be found to prevent e.g. unwanted steep and shallow fences. The costs make it impossible to apply Lemma 4 to reduce the graph size, as this may cause the removal of equally long or longer paths with lower cost from the graph. The size of the resulting graph is therefore $O(n^4)$. Dijkstra's algorithm (see e.g. [10]) has been used to find the minimal cost path through the graph in time $O(n^4)$.

4 An Output-Sensitive Algorithm

A disadvantage of the algorithm of the previous section is the high running time of $O(n^3 \log n)$, even if the computed fence design turns out to be short. In [4, 20] we showed that, for a large class of parts, the fence designs have linear length, and for most long and thin parts, the fence design has constant length. This suggests that an output sensitive algorithm, whose running time depends on the length, is to be preferred. In this section we present such an algorithm which calculates the shortest fence design in $O(kn \log n)$ time, with n the combinatorial complexity of the part and k the number of fences of the computed design. Using this algorithm, most fence designs can be computed in $O(n^2 \log n)$, or even $O(n \log n)$ time.

4.1 Maintaining the best intervals

The main idea of the algorithm is to maintain the shortest interval of possible orientations after k fences, instead of precomputing the whole graph of all possible intervals of orientations. This is basically the same technique as used by Goldberg's algorithm to compute push plans [13]. Goldberg maintains the interval of possible orientations, and greedily shrinks this interval per application of the pushing jaw. We, however, must take into account the constraints of fence design. It is not sufficient to maintain a single shortest interval of possible orientations. Lemma 4 indicates that it is sufficient to maintain for each stable orientation the shortest interval starting at this orientation. Also, to be able to compute the possible reorientation of the jaw, we have to have two copies, one for the shortest interval with a plan that ends with a left fence, and another copy for a plan ending with a right fence. We denote the last orientation in the interval starting with a_i on a fence of type t after k fences by $a_i^t(k)$. Let us denote the sets containing all intervals after k fences ending with a left or a right fence by $\mathcal{L}(k)$ and $\mathcal{R}(k)$ respectively. After one fence, we have the following sets of shortest intervals:

$$\begin{aligned}
\mathcal{L}(1) &= \{[a_0, a_0^{\mathcal{L}}(1)], [a_1, a_1^{\mathcal{L}}(1)], \ldots, [a_{m_s}, a_{m_s}^{\mathcal{L}}(1)]\} \\
&= \{[a_0, a_{m_s}], [a_1, a_0], \ldots, [a_{m_s}, a_{m_s-1}]\} \\
\mathcal{R}(1) &= \{[a_0, a_0^{\mathcal{R}}(1)], [a_1, a_1^{\mathcal{R}}(1)], \ldots, [a_{m_s}, a_{m_s}^{\mathcal{R}}(1)]\} \\
&= \{[a_0, a_{m_s}], [a_1, a_0], \ldots, [a_{m_s}, a_{m_s-1}]\}
\end{aligned}$$

We shall now explain how to compute the sets of intervals $\mathcal{L}(k)$ and $\mathcal{R}(k)$, given $\mathcal{L}(k-1)$ and $\mathcal{R}(k-1)$. We show how to compute the intervals in $\mathcal{L}(k)$, since computing the

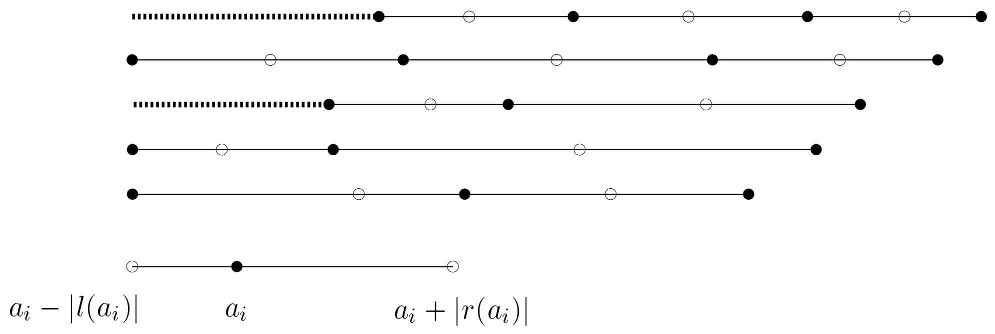

Figure 6: The candidate intervals for $[a_i, a_i^{\mathcal{L}}(k)]$. The two intervals with the trailing dashed line cannot be mapped onto $a_i - |l(a_i)|$ by a valid fence design. The best interval is the interval with the leftmost right endpoint in the figure.

intervals in $\mathcal{R}(k)$ is analogous. Let us consider the interval starting with orientation a_i and suppose that we want to compute the orientation $a_i^{\mathcal{L}}(k)$ such that the length of $[a_i, a_i^{\mathcal{L}}(k)]$ is minimal. Since the last fence we want to use is a left fence, we are allowed to change the push direction by angles in $(0, \frac{\pi}{2})$ and $(\frac{\pi}{2}, \pi)$, if the $k-1$'th fence was a left or right fence respectively. This constrains the possible alignment of the previous calculated shortest intervals in $\mathcal{L}(k-1)$ and $\mathcal{R}(k-1)$ with a_i. The preimage of a_i under the push function is $(a_i - |l(a_i)|, a_i + |r(a_i)|)$. Only intervals which can be mapped onto this preimage can possibly result in an interval of the form $[a_i, b]$ and therefore are *candidates* for the shortest interval $[a_i, a_i^{\mathcal{L}}(k)] \in \mathcal{L}(k)$. We have to determine the best interval among the candidate intervals from $\mathcal{L}(k-1)$ and $\mathcal{R}(k-1)$, i.e. the interval with the shortest possible image, which has to start with a_i, under the shifted pushfunction.

A single interval's image which starts with a_i can vary in length, depending on the position of the interval's left endpoint before applying the (shifted) push function. Any position of this left endpoint in $l(a_i) \cup \{a_i\} \cup r(a_i)$ results in an image starting with a_i. The image is shortest, though, when aligned as close as possible to $a_i - |l(a_i)|$. Therefore, our first goal is to determine the intervals which can align with $a_i - |l(a_i)|$. Also, we want to determine intervals which can be mapped onto angles in $(a_i - |l(a_i)|, a_i + |r(a_i)|)$. These two sets of intervals form the afore-mentioned candidates for the shortest interval $[a_i, a_i^{\mathcal{L}}(k)] \in \mathcal{L}(k)$. The candidate of which the right endpoint, after proper alignment with a_i, is leftmost determines the shortest interval $[a_i, a_i^{\mathcal{L}}(k)] \in \mathcal{L}(k)$, and is therefore the best candidate. The position of the right endpoint of a candidate interval is dependent on the length of the interval, as well as on the leftmost position onto which we can possibly map the left endpoint of this interval with a valid change of the push direction. Figure 6 depicts the orientation a_i together with five candidate intervals, of which two cannot be mapped onto $a_i - |l(a_i)|$. The alignment of the intervals closest to $a_i - |l(a_i)|$ is depicted in the picture. The two dashed lines correspond to the difference between $a_i - |l(a_i)|$ and the best alignment of the intervals.

A naive, but simple, algorithm to compute the sets $\mathcal{L}(k)$ and $\mathcal{R}(k)$ is to:

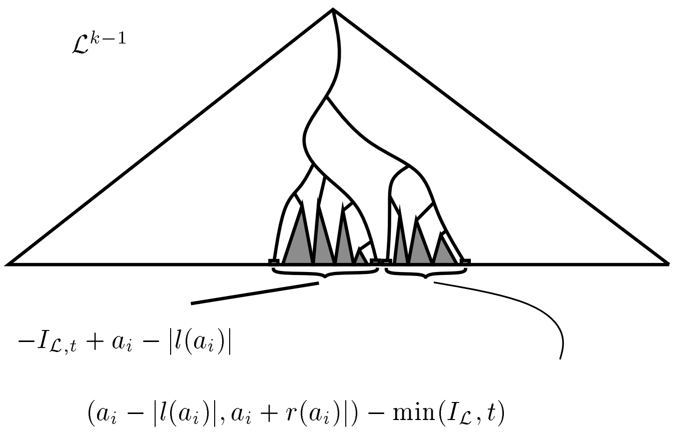
$\mathcal{L}^{k-1}$
$-I_{\mathcal{L},t} + a_i - |l(a_i)|$
$(a_i - |l(a_i)|, a_i + r(a_i)|) - \min(I_{\mathcal{L}}, t)$

Figure 7: The two queries from the range tree $\mathcal{L}(k-1)$ for $a_i^t(k)$.

1. Test for each pair of an orientation a_i and a fence type, all intervals in $\mathcal{L}(k-1)$ and $\mathcal{R}(k-1)$ and check if there is a valid reorientation of the jaw such that these intervals align with $a_i - |l(a_i)|$, or possibly map onto $(a_i - |l(a_i)|, a_i + |r(a_i)|)$ otherwise.

2. Determine the interval with the smallest possible right end point with respect to its best alignment with $a_i - |l(a_i)|$.

3. Compute the image of the push function of this interval and store this image in appropriate set $\mathcal{L}(k)$ or $\mathcal{R}(k)$.

It takes linear time to determine the best best candidate per interval. Therefore, this naive approach leads to an $O(kn^2)$ algorithm.

4.2 Updating in $O(n \log n)$ time

We now show how to speed up the part which determines the best candidate to logarithmic time per interval. To achieve this, we build a *range tree* (see e.g. [11]) on the starting orientations of the intervals in $\mathcal{L}(k-1)$ and $\mathcal{R}(k-1)$, which allows us to find in logarithmic time sets of the candidate intervals for any starting orientation after k fences. A range tree is a balanced binary search tree. The leaves of the tree contain the intervals of the previous step, ordered on starting orientation. The nodes of the tree contain splitting values, which guide the search. The subset of keys which are contained in the subtree is referred to as the *canonical subset* of this subtree. Let us store in each node of the range tree, the position of the left-most right endpoint of the canonical subset of intervals of the subtree rooted at this node, and furthermore, the length of the shortest interval of the same canonical subset. We can now easily derive the candidate interval with the leftmost right-interval, thereby reducing the number of candidates to $O(\log n)$, as we shall outline in the next three paragraphs.

Let us first try to understand which range queries are necessary and sufficient to compute a new interval. Assume that we want to compute $a_i^t(k)$, i.e. the right endpoint of the shortest interval starting with a_i after k fences; the k'th fence being of type t. We first analyze which intervals from $\mathcal{L}(k-1)$ align with $a_i - |l(a_i)|$, and which intervals map onto $(a_i - |l(a_i)|, a_i + |r(a_i)|)$. The interval from $\mathcal{L}(k-1)$ must be reoriented by an angle in $I_{\mathcal{L},t}$—$I_{\mathcal{L},t}$ being the angular interval of possible reorientations of the jaw by a left fence and a fence of type t, see the table in Section 2.2. Intervals $[a_j, a_{j'}]$ with $a_j \in a_i - |l(a_i)|$ minus angles in $I_{\mathcal{L},t}$ can be mapped onto $a_i - |l(a_i)|$. If, on the other hand, the minimal angle $\min(I_{\mathcal{L},t})$ in $I_{\mathcal{L},t}$, cannot map a_j onto $a_i - |l(a_i)|$, a_j can be mapped onto $(a_i - |l(a_i)|, a_i + |r(a_i)|)$, if $a_j + \min(I_{\mathcal{L},t}) \in (a_i - |l(a_i)|, a_i + |r(a_i)|)$. Querying from $\mathcal{R}(k-1)$ is similar. Therefore, we query as follows from $\mathcal{L}(k-1)$ and $\mathcal{R}(k-1)$:

The intervals which align with $a_i - |l(a_i)|$ correspond to two range queries of length $\frac{\pi}{2}$; $-I_{\mathcal{L},t} + a_i - |l(a_i)|$ in $\mathcal{L}(k-1)$, and $-I_{\mathcal{R},t} + a_i - |l(a_i)|$ in $\mathcal{R}(k-1)$. Since these intervals align with $a_i - |l(a_i)|$, the shortest such interval (which is stored with the nodes) is the candidate per canonical subset. There is a logarithmic number of canonical subsets, and consequently a logarithmic number of candidates.

The intervals which can only be mapped onto $(a_i - |l(a_i)|, a_i + |r(a_i)|)$ receive a penalty which is added to the length of this interval. In the end, the interval which has the leftmost right endpoint is the best candidate of a canonical subset. There is a minor problem with calculating the leftmost right endpoint of a set of intervals. Since the push function is periodic, the leftmost point is dependent on the choice of the start of the 2π wraparound. The penalty of two consecutive intervals might differ 2π plus the desired difference. To overcome this problem, we compute two copies of the push function and store the both of them in the range tree—in the second copy of the push function the values are 2π higher than in the first copy. This way, any range of intervals can be found as one piece of the range tree. We query $(a_i - |l(a_i)|, a_i + |r(a_i)|) - \min(I_{\mathcal{R},t})$ in $\mathcal{R}(k-1)$ and $(a_i - |l(a_i)|, a_i + |r(a_i)|) - \min(I_{\mathcal{L},t})$ in $\mathcal{L}(k-1)$. To overcome wraparound problems, we make sure that $a_i + |r(a_i)| > a_i - |l(a_i)| > \min(I_{\mathcal{R},t})$, by adding 2π to some of the values, if necessary. We derive a logarithmic number of canonical subsets. The values of their leftmost right endpoints are too high to compete with the lengths of the candidates of the former two queries. We want the difference between $a_i - |l(a_i)|$, and the right endpoint. Therefore, we subtract $a_i - |l(a_i)|$ from the found right endpoints, and derive a logarithmic number of appropriate candidates.

In Figure 7 the queries from $\mathcal{L}(k-1)$ for $a_i^t(k)$ are depicted. The query on the left corresponds to intervals of which the starting orientation can be mapped onto $a_i - |l(a_i)|$. The length of the shortest interval of the canonical subsets of this query determines the best candidate for this query. The query on the right corresponds to intervals of which the starting orientation can be mapped onto $(a_i - |l(a_i)|, a_i + |r(a_i)|)$. The difference of the leftmost right endpoint of the canonical subsets of this query and $a_i - |l(a_i)|$ determines the best candidate for this query. Together with the corresponding queries from $\mathcal{R}(k-1)$, the shortest interval starting with a_i on a fence of type t is determined. The total number of candidates resulting from the four range queries is $O(\log n)$. The best candidate of the four queries is the interval which determines the value of $a_i^t(k)$, and is computed in $O(\log n)$

time. This leads to the following lemma.

Lemma 7 *Let there be given two range trees; one in which two copies on two folds of the unit circle of the intervals in $\mathcal{L}(k-1)$ are stored; and one for the intervals in $\mathcal{R}(k-1)$, stored in a similar way. Both trees are ordered on the left endpoints of the intervals. If in the nodes the shortest interval as wel as the leftmost right endpoint of the corresponding canonical subsets are stored, then, the sets $\mathcal{L}(k)$ and $\mathcal{R}(k)$ can be computed in $O(n \log n)$ time.*

The building time of the interval tree is $O(n \log n)$ time. First, we presort the intervals on their length, and their right endpoint. Then, we build the range-tree buttom up in linear time. The query takes $O(\log n)$ per calculated new shortest interval. We have $O(n)$ new intervals. Per fence we spend $O(n \log n)$ time.

4.3 Incrementally computing the fence design

We want to use the described method to compute fence designs. This implies that the algorithm has to terminate when the part is oriented up to symmetry. If the part has no rotational symmetry, then the part is oriented if there is one stable orientation in an interval of possible orientations. If, on the other hand the part has rotational symmetry, then we have to detect if the part is oriented up to symmetry. This can be done in constant time as well, by comparing the indices of the computed intervals. We can easily precompute the desired difference of the indices.

Also, we want to know which fence design corresponds to our finally derived interval which corresponds to the oriented part. This can be accomplished by incrementally building the same graph as in Section 3, storing the used fences with the edges of the graph. We can find the fence design, by tracking back the path from the 'oriented' node to the source of the graph. The size of the graph increases by $O(n)$ per step of the algorithm. Tracking back the path takes $O(k)$ time.

The algorithm, therefore, runs in $O(kn \log n)$ time, k the number of steps of the algorithm. The following theorem summarizes the result of this section.

Theorem 8 *Let P be a polygonal part. A fence design which orients P up to symmetry can be computed in $O(kn \log n)$ time, with n the combinatorial complexity of P, and $k = O(n^2)$ the number of fences in the resulting design.*

Proof: A fence design which orients P up to symmetry exists [4, 3]. We can, by subtracting the indices of a current shortest interval, check in constant time if the part is oriented up to symmetry. Extending the fence design by one fence takes $O(n \log n)$ time. Maintaining the used fences takes $O(n)$ time per extension. The total running time is, therefore, $O(kn \log n)$ time, with k the number of fences in the design. □

Although we did not actually implement this output-sensitive algorithm, the algorithm is more or less as easy to implement as the graph based algorithm. The main difference between the two algorithms is the use of a range tree in the latter, which is basically a binary search tree. Another difference is the incremental construction of the graph, which is not any harder than precomputing the nodes and the edges. We believe the resulting code of the algorithm of this algorithm is faster than the code of the previous section, provided that any part can be oriented by a linear length fence design.

5 Friction

Up till now, we assumed that there was no friction between the fences and the part. In this section, we shall incorporate friction into the model.

5.1 The model

There are several ways to look at friction. We will use the simplified model of Coulomb friction (see e.g. [15]). Imagine a part which slides along a fence on a stable edge. The frictional force is a force opposite to the motion of the part. According to Coulomb's law, the frictional force is a fixed fraction of normal force acting on the part—the normal force is induced by the belt which presses the part onto the fence. This fraction is given by the *coefficient of friction* μ. A simple statement of Coulomb's law for a moving part is:

$$|f_{friction}| = \mu |f_{normal}|$$

If $\mu|f_{normal}|$ is larger than the resulting force in the direction of the part, then the part will remain at rest. The frictional force now balances the applied force by the belt, and Coulomb's law boils down to:

$$|f_{friction}| \leq \mu |f_{normal}|$$

Suppose we have a part on an fence, and we apply a downward force induced by the conveyor. In Figure 8, we have on the left, a part which remains at rest. The frictional force combined with the normal force cancel out the force of the belt. On the right, the part slides, the frictional force combined with the normal force to not balance the belt force. A graphical interpretation of the Coulomb friction is given by the cone with dihedral angle $2 \arctan \mu$. If the normal force combined with the frictional force is in the interior of the cone, the part remains stable, otherwise, the part will slide. It easy to see whether or not the combined normal and frictional force is in the cone. If we negate the belt force, we check whether or not this force is in the cone. In the former case, the part remains stable, in the latter, the part slides.

In his thesis, Mason [15] describes an easy test to predict the rotation of the part which has a vertex in contact with the fence. He gives an analysis which boils down to a very simple geometric check. Given the orientation of the part and the fence angle, we can draw three half lines emanting from the contact vertex. One is the line of pushing, which is opposite to the direction of the belt. The two other lines are the boundary edges of the friction cone. The area above the fence is now divided into four areas, which we enumerate

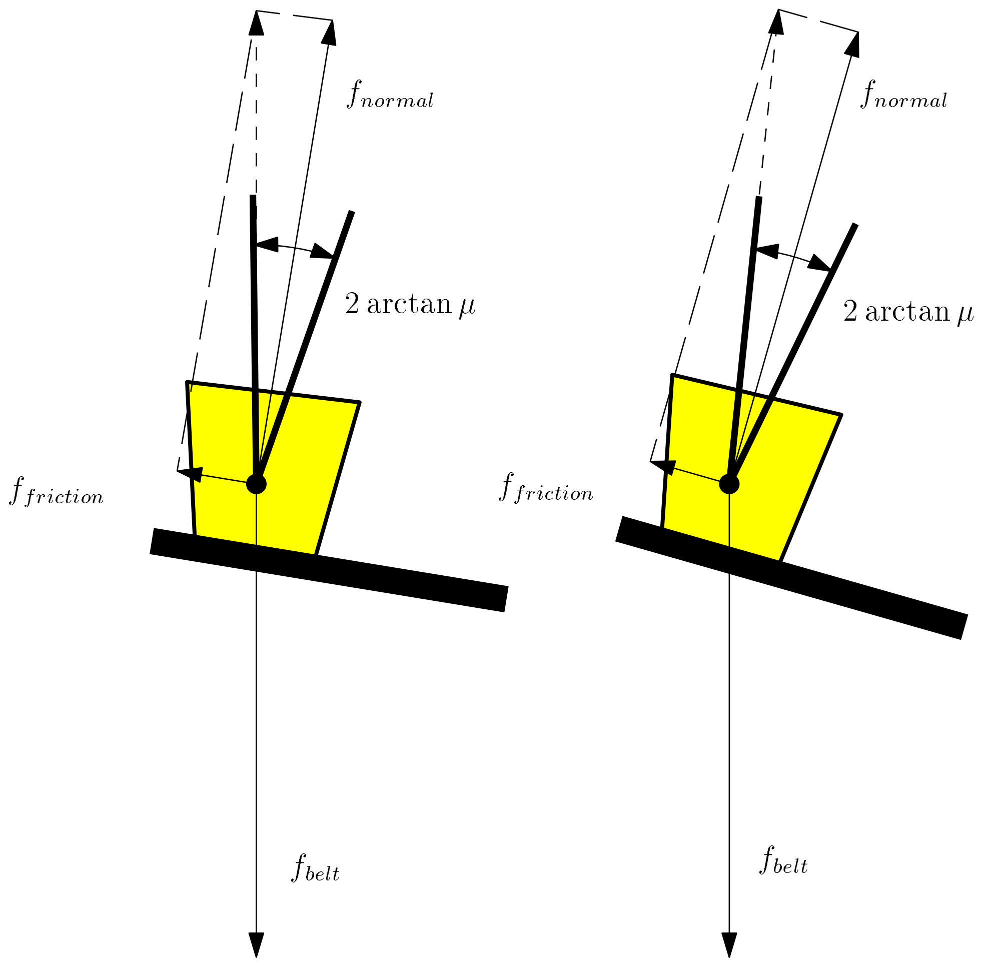

Figure 8: A part on a fence with friction. Left, the part does not slide, due to friction. Right, the part slides.

from left to right (see Figure 9(a)). The position of the center-of-mass now determines the rotation of the part. In area's **I** and **II**, the part will rotate counterclockwisely. In area's **III** and **IV**, the part will rotate clockwisely. If the center-of-mass is on the line separating area **II** from area **III**, then the part is in unstable equilibrium, and will neither rotate clockwisely, nor rotate counterclockwisely. Though, the part might either remain in the same position if the separating line is the line of pushing, or slide towards the end of the fence, otherwise.

To determine whether or not an edge is stable we test for both vertices of the edge if they predict counterclockwise or clockwise rotation. If the vertices disagree, the edge is stable, otherwise, the edge is unstable and the part will rotate as predicted.

We now incorporate the developed theory into the fence design model. The first observation we make is that we cannot make fences which are very shallow, because these fences

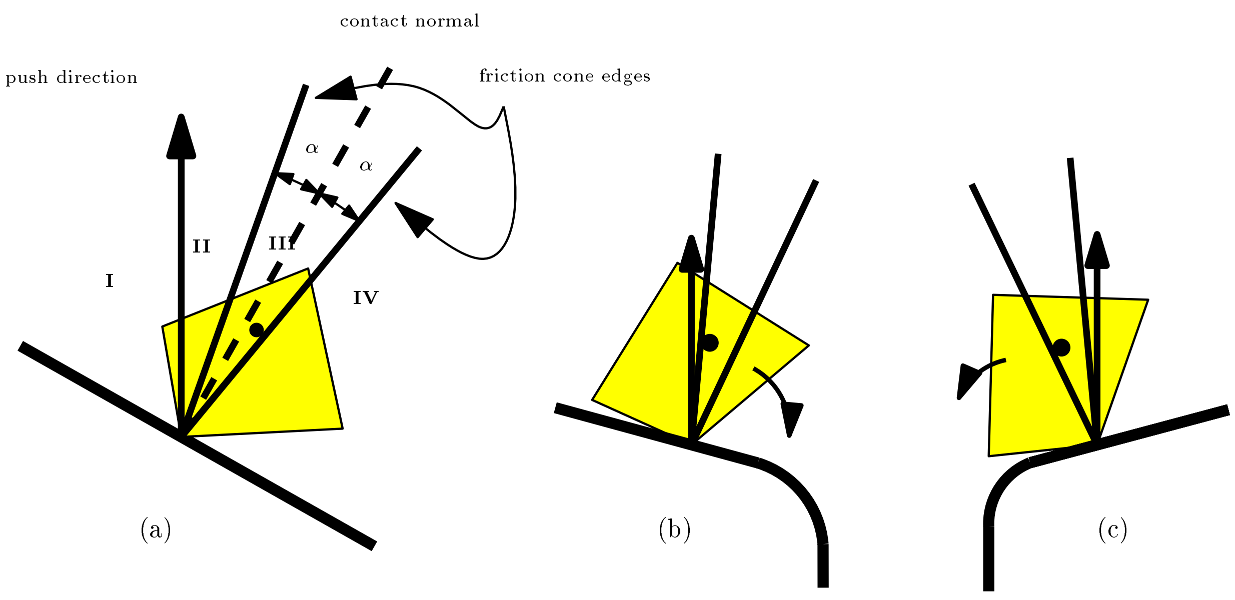

Figure 9: (a) The rotation of the part in vertex-contact with the fence. (b) and (c) The same part with the same orientation with respect to the fence. However, due to friction, the left fence rotates the part to another edge than the right fence.

do no longer allow the parts to slide off the fences. The (absolute) minimum fence angle must be strictly greater than $\arctan\mu$. Under this assumption, the line which separates area **II** from area **III** is the left edge of the friction cone, if the part is on a left fence, and the right edge of the friction cone otherwise. This implies that the push function, which predicts the rotation of the part, is no longer symmetric. There are orientations for which a left fence rotates the part clockwisely, and a right fence rotates the part counterclockwisely (see Figure 9(b),(c)). However, the push function for a part on a left fence is still monotonous, as is the push function for a part on a right fence. Moreover, one could look at the frictional fence as a frictionless fence which is $\arctan\mu$ shallower. The normal of this shallower frictionless fence coincides with described edge of the friction cone. However, the steepest frictionless fence which corresponds to a frictional fence is $\frac{\pi}{2} - \arctan\mu$ for a right fence, and $\arctan\mu - \frac{\pi}{2}$ for a left fence. Unfortunately, this implies that frictional fences are not able to orient any part up to symmetry. It is possible to construct parts which can only be oriented with a reorientation of the push direction push arbitrarily close to π [1]. Since the reorientation of the jaw closest to π is $\pi - \arctan\mu$ for a frictional fence, we can always construct a part which cannot be oriented by frictional fences.

5.2 Algorithms for frictional fence designs

In this section we discuss two algorithms to compute frictional fence designs, if one exists. Firstly, we discuss a modification of the graph based algorithm, and secondly we give a modification of the output sensitive algorithm. Let us first redefine the intervals of possible reorientations of the jaw for frictional fences.

f_i	$\alpha_{i+1} \in I^{\mu}_{f_i,f_{i+1}}$	f_{i+1}
left	$(0, \pi/2 - \arctan \mu)$	left
left	$(\pi/2, \pi - \arctan \mu)$	right
right	$(-\pi + \arctan \mu, -\pi/2)$	left
right	$(-\pi/2 + \arctan \mu, 0)$	right

This means that we can still use our graph based algorithm. However we have to modify the part which decides if a node can be connected to another node in the graph.

Consider two graph nodes (s,t) and (s',t'), where $s = [a_i, a_j]$ and s' are intervals of stable equilibria and t and t' are fence types. Let $I^{\mu}_{t,t'}$ be the open interval (of length $\pi/2 - \arctan \mu$) of reorientations admitted by the successive fences of types t and t' according to the table above. There is a directed edge from (s,t) to (s',t') if there is an angle $\alpha \in I^{\mu}_{t,t'}$ such that a reorientation of the push direction by α followed by a push moves any stable orientation in s into a stable orientation in s', and furthermore α is constructed by a frictional fence. To check this condition, we determine the preimage $(u,w) \supseteq s'$ of s', under the push function. Observe that if $|s| = a_j - a_i < w - u$, any reorientation in the open interval $r = (u - a_i, w - a_j)$ followed by a push will map s into s'. We add an edge from (s,t) to (s',t') if the intersection of r and the interval $I^{\mu}_{t,t'}$ of admitted reorientations is not empty. We label this edge with this intersection. Also, we add a source and a sink to the graph. We connect the source to every node (s,t) in the graph for which s contains all m_s stable equilibria, and we connect every node (s,t) with s an interval which is reduced up to symmetry to the sink. Every path from the source to the sink now represents a fence design; a fence design of minimum length corresponds to the shortest such path and can be found by a breadth-first search from the source.

Let τ be a path in the graph from the source to the sink. Every edge of τ corresponds to a non-empty angular interval of possible reorientations of the push direction. We pick the midpoint ϕ of every such interval, and determine if this reorientation is by left or a right fence. For a left fence, we add $\arctan \mu$ to the corresponding fence angle. For a right fence, we subtract $\arctan \mu$. We get a frictional fence design.

Theorem 9 *Let P be a polygonal part, with coefficient of friction μ with respect to the fences. There is an algorithm which computes a shortest fence design in time $O(n^3 \log n)$, if there is a fence design which orients P. If no fence designs exists, the algorithm detects failure in the same time bound.*

Also, we can modify the output-sensitive algorithm to compute the shortest fence design for frictional fences, if one exists. The modification by itself is quite straightforward.

Instead of using the table of Section 2.2, we use the modified table of this section to determine the possible reorientations of the jaw to extend the fence designs. A problem, however, is to determine when to stop augmenting the fence designs. Therefore, we have to add an extra stop criterion to ensure the algorithm does not try to extend the fence design when it becomes clear that this no longer makes sense.

Let us concentrate on the array which stores the current set of possible intervals. During one step of the algorithm, we replace items of the array by shorter items. We argue that, if an interval was not replaced in a step of the algorithm, then we need not to compute the new intervals derived from this interval in the succeeding step. This is because intervals monotonously decrease, and the intervals which would have been derived from this interval in the succeeding step, were already computed in the former step. The second stop criterion is to stop when no interval can be improved on in a single step. Checking if an interval is shrunk in a step of the algorithm can be done by storing a flag which is reset at the start of a step, and set if the interval is improved. Checking if any interval needs improvement, can be done on the fly. Storing and updating the flags does not influence the asymptotic running time. Since there are at most $O(n^2)$ intervals of stable equilibria, the algorithm terminates after at most $O(n^2)$ steps. This leads to an upper bound on the running time of $O(n^3 \log n)$. While improving the intervals, we also incrementally construct the graph corresponding to the found fence designs. At the end, deriving a frictional fence design from a path in the graph is accomplished in the same way as in the graph based method.

Theorem 10 *Let P be a polygonal part, having combinatorial complexity n. If there is a fence design that orients P, then a design can be computed in $O(kn \log n)$ time, with k the number of required fences, which orients the part up to symmetry. If no fence design exists, the algorithm reports failure in $O(n^3 \log n)$ time.*

If we could prove that any fence design has linear length, then this result would most likely carry over to the upper bound on the running time of the algorithm reporting failure. This would result in an upper bound of the running time of $O(n^2 \log n)$ of the frictional fence design algorithm.

6 Modular Fences

In this section, we discuss modular fence designs, i.e., designs for which the fence angles are to be taken from a discrete and finite set of angles. Modular fences can be useful in an industrial application of a conveyor to orient parts. There is no need for 'expensive' adjustable fences. The conveyor operator picks prefabricated fences out of a box and just attaches them along the sides of the belt. Figure 10 shows an example of a fence design constructed with modular fences of $\frac{\pi}{4}$ and $-\frac{\pi}{4}$, together with a fence design build with arbitrary fence angles. As is to be expected, the modular fence design requires more fences.

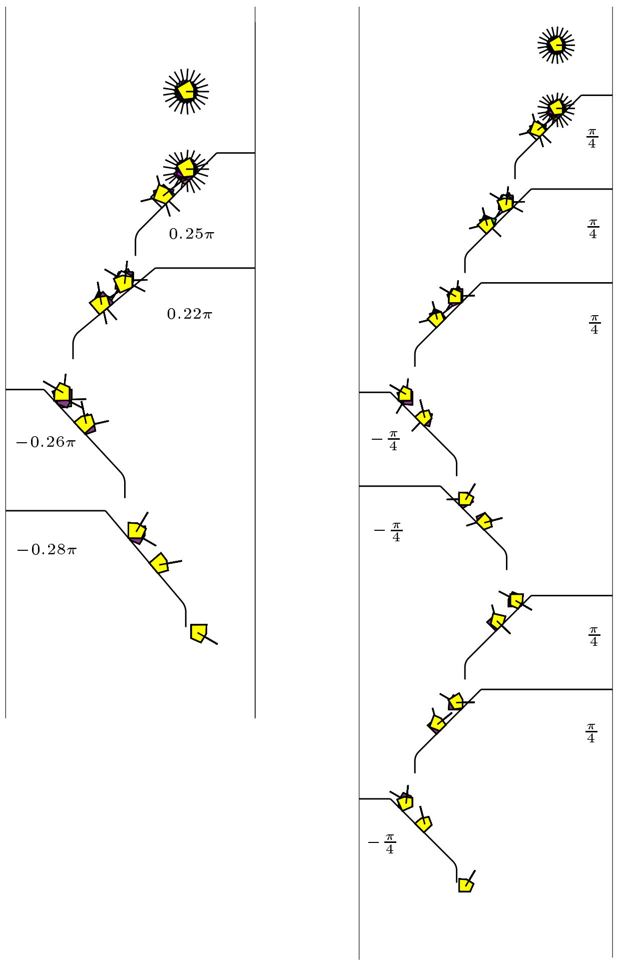

Figure 10: Two fence designs for the same part. On the left, any fence angle is available. On the right, only fences of $\frac{\pi}{4}$ and $-\frac{\pi}{4}$ are used.

6.1 The model

We first discuss how the discretization of the possible fence angles influences the possible reorientations of the jaw, and therefore the algorithms to compute fence designs.

Let m denote the number of fence angles. Every fence can correspond to two reorientations of the jaw. A left fence with fence angle $\phi \in (-\frac{\pi}{2}, 0)$ corresponds to a reorientation of the jaw of $\phi + \frac{\pi}{2}$ if it succeeds a left fence in the fence design, and $\phi - \frac{\pi}{2}$ if it succeeds a right fence. A right fence with fence angle $\phi \in (0, \frac{\pi}{2})$ corresponds to a reorientation of the jaw of $\phi + \frac{\pi}{2}$ if it succeeds a left fence in the fence design, and $\phi - \frac{\pi}{2}$ if it succeeds a right fence. The total number of possible reorientations of the jaw is bounded by $2m$.

Still, it is possible to construct parts which can only be oriented with a reorientation of the push direction push arbitrarily close to π [1]. This implies that one can always construct a part that cannot be oriented, given a set of modular fence angles.

The modular fences we treat in this section are friction free. Using the techniques of Section 5, one can easily modify the algorithms of this section to compute frictional modular fence designs as well.

6.2 Algorithms for modular fence designs

In this section we give two algorithms to compute modular fence designs (if they exist for the part). As in the section on frictional fences, the first algorithm is based on the graph based algorithm of Section 3, and the second is based on the output-sensitive algorithm of Section 4. However, the running times of the algorithms are surprisingly different. The former runs in $O(mn^2)$ and the latter in $O(mkn)$ time, m denoting the number of available fence angles, k the number of fences in the design. Since k can vary from $O(1)$ to $O(n^2)$ in the worst case, it is interesting to present both algorithms.

In the algorithms of this section, we want to be able to quickly compute the image under the push function of a value of the form $a_i + \alpha$, a_i a stable equilibrium, and α in the set of reorientations of the jaw by modular fences. Therefore, we precompute these values using the push function. The number of possible reorientations of the jaw is $2m$. These images are computed by tracing the push function, in time $O(mn)$. We store these values in a two dimensional array. We can now determine in constant time the value of $p(a_i + \alpha)$, for each stable equilibrium a_i and each modular reorientation of the jaw α.

The nodes and the edges of the graph based algorithm are encoded the same way as in Section 3. The graph is built in a somewhat different way than before, though. A node of the graph again corresponds to a state of the part, encoded as an interval of possible orientations. The edges between the nodes are added a bit differently than before though. Let (s, t) be a node in the graph, $s = [a_i, a_j]$ is an interval of stable equilibria and t is a fence type. Since each modular fence has a fixed angle, we can for *each* fence angle compute the image of s under the push function. If we add edges from (s, t) to the nodes in the graph corresponding to these images, Lemma 4 states that the resulting graph suffices to compute shortest paths in the graph, corresponding to shortest fence designs, although there might be redundant edges.

We compute the images as follows. For each modular fence f, having type t', we compute the reorientation of the jaw $\alpha_{t,f}$ between a fence of type t, and the fence f. The interval of orientations after appending fence f to a design is $s' = [p(a_i + \alpha_{t,f}), p(a_j + \alpha_{t,f})]$. We add a directed edge in the graph from (s, t) to (s', t'). We label this edge with this modular push angle. For each node, we spend $O(m)$ time computing the outgoing edges.

Also, we add a source and a sink to the graph. We connect the source to every node (s, t) in the graph for which s contains all m_s stable equilibria, and we connect every node (s, t) with s an interval which is reduced up to symmetry to the sink. Every path from the source to the sink now represents a fence design; a fence design of minimum length corresponds to the shortest such path and can be found by a breadth-first search from the source.

The building time and the size of the graph are $O(mn^2)$. The search through the graph takes $O(mn^2)$ as well. We can, therefore, compute a fence design in $O(mn^2)$ time, if one exists. If no fence design exists which orients the part up to symmetry, then there is no path from the source to the sink in the graph, which is detected after $O(mn^2)$ as well. In the given bounds, m is the number of available fence angles. If we assume that a constant number of fixed angles is available, then the running time is $O(n^2)$.

Theorem 11 *Let P be a polygonal part, having combinatorial complexity n. Let m be the number of different available fence angles. If there is a fence design that orients P, then a shortest fence design which orients this part can be computed in $O(mn^2)$. Otherwise failure is detected in the same time bound.*

We now present an output sensitive algorithm which computes a fence design in $O(mkn)$ time, with k the number of fences. The algorithm is based on the output-sensitive algorithm of Section 4. If $k = O(1)$, this algorithm is asymptotically faster than the previously presented algorithms.

We store $O(n)$ intervals of possible orientations in an array. These intervals are the shortest intervals reachable with a fence design which is incrementally constructed. We have to take into account the fence type, and store therefore for each tuple of a stable orientation v and a fence type t the shortest interval starting with orientation v , with the last push by a fence of type t. When we augment the fence design with one fence, we can compute the new shortest intervals as follows. For each interval, we compute the m images of this interval. We check in the array of shortest intervals, if the new interval is shorter than the currently stored interval, and, if so, we replace the stored interval by the new interval. Looking up the image of an interval of stable orientations can be done in constant time, using our precomputed two-dimensional array with stored values of the push functions. Therefore, given the intervals after $k-1$ fences, computing the intervals after k fences is accomplished in $O(nm)$ time.

We stop when the part is oriented up to symmetry. This can be checked by comparing the indices of the intervals of possible orientations. Unfortunately, not every part can be oriented up to symmetry using a modular fence design. This means we have to add an extra stop criterion which determines if the intervals keep improving. We already discussed how to add this criterion without extra time overhead in Section 5, and we shall not repeat this discussion here, but just give the result in the following theorem.

Theorem 12 *Let P be a polygonal part, having combinatorial complexity n. Let m be the number of different available fence angles. If there is a fence design that orients P, then a design can be computed in $O(kmn)$, with $k = O(n^2)$ the number of required fences, which orients the part up to symmetry. If no fence design exists, the algorithm terminates in $O(mn^3)$ time.*

Since the best known upper bound on the length of a fence design is $O(n^2)$, the latter algorithm might be slower than the graph based algorithm. However, we believe that fence designs, like push plans [9], have at most linear length. It is an interesting open problem to prove or disprove this.

7 Conclusion

In this paper we addressed the algorithmic issues of sensorless part orientation by a sequence of fences across a conveyor belt. Frictionless fences can orient any polygonal part up to symmetry. In this paper, we gave several algorithms to compute fence designs. We first reviewed an easy to implement algorithm which runs in $O(n^3 \log n)$ time. This algorithm can be modified to compute fence designs which take into account various quality measures, such as avoidance of very steep or shallow fences. This modification increases the running time of the algorithm to $O(n^4)$.

The structure of the algorithm yields an $O(n^2)$ bound on the length of the shortest fence design. But for a large class of parts it is known that fence designs have linear length and for most long and thin parts, a feasible fence design consists of only a constant of number fences. In this paper, we presented the first output sensitive fence design algorithm which benefits from these bounds on the length of the plan. This algorithm runs in $O(kn \log n)$ time, with k the number of fences in the final design.

Furthermore, we gave algorithms to compute fence designs for frictional fences and modular fences, in which we allow only a discrete set of fence angles. The former turned out to be just a minor modification to the algorithms for computing frictionless fence designs. The latter, however, gave rise to a different approach to computing fence designs in time $O(mn^2)$, or $O(mkn)$, k the number of fences, m the number of fence angles. Unfortunately, not every part can be oriented with frictional or modular fences.

The results largely settle the algorithmic questions in computing fence designs, although some improvements in the running time might still be possible. The main interesting open problem left is the question which parts can be oriented with friction of with modular fences. Experiments show that for many parts a valid fence design still exists. An other open problem is the bound on the length of the fence design. Only an $O(n^2)$ upper bound is known at the moment. It is unknown whether this is indeed required ar whether linear length designs always exist. Also, looking into orienting algebraic parts [19] using fences is of interest. Finally, we want to mention the problems of dealing with uncertainty (see [8] for a first treatment) and generalization of fence design to 3D.

Acknowledgements

This research was supported in part by NATO Collaborative Research Grant CRG 951224. Berretty's research is supported by the Dutch Organization for Scientific Research (N.W.O.). Goldberg's research is supported in part by the National Science Foundation under IRI-9612491 and by Presidential Faculty Fellow Award IRI-9553197.

References

[1] S. Akella, W. Huang, K.M. Lynch, and M.T. Mason. Sensorless parts feeding with a one joint manipulator. In *IEEE International Conference on Robotics and Automation,*

1997. See also *Alg. for Robotic Motion and Manipulation* (J.-P. Laumond and M. Overmars (Eds.)), A.K. Peters, pages 229–238, 1996.

[2] S. Akella and M.T. Mason. Posing polygonal objects in the plane by pushing. In *IEEE International Conference on Robotics and Automation, Nice*, pages 2255–2268, 1992.

[3] R.-P. Berretty, K. Goldberg, M. Overmars, and A.F. van der Stappen. Computing fence designs for orienting parts. Technical report, UU-CS-1997-41, Dept. of Comp. Science, Utrecht University, 1997.

[4] R.-P. Berretty, K.Y. Goldberg, M.H. Overmars, and A.F. van der Stappen. On fence design and the complexity of push plan for orienting parts. In *Proc. 13th Ann. ACM Symp. on Computational Geometry.*, pages 21–29, 1997.

[5] K.-F. Böhringer, V. Bhatt, and K.Y. Goldberg. Sensorless manipulation using transverse vibrations of a plate. In *Proc. IEEE Int. Conf. on Robotics and Automation, Nagoya, Japan*, pages 1989–1996, 1995.

[6] K.-F. Böhringer, B.R. Donald, and N.C. MacDonald. Upper and lower bounds for programmable vector fields with applications to mems and vibratory plate part feeders. *Alg. for Robotic Motion and Manipulation,* J.-P. Laumond and M. Overmars (Eds.), A.K. Peters, pages 255–276, 1996.

[7] M. Brokowski, M.A. Peshkin, and K. Goldberg. Optimal curved fences for part alignment on a belt. *ASME Transactions of Mechanical Design*, 117, March 1995.

[8] J. Chen, K. Goldberg, M. Overmars, D. Halperin, K.-F. Böhringer, and Y. Zhuang. Shape tolerance in feeding and fixturing. em This proceedings, 1998.

[9] Y.-B. Chen and D.J. Ierardi. The complexity of oblivious plans for orienting and distinguishing polygonal parts. *Algoritmica*, 14:367–397, 1995.

[10] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge, Massachusetts; London, England, 1990.

[11] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry, Algorithms and Applications.* Springer, Berlin Heidelberg, 1997. ISBN 3-540-61270-X.

[12] M.A. Erdmann and M.T. Mason. An exploration of sensorless manipulation. *IEEE Journal of Robotics and Automation*, 4:367–379, 1988.

[13] K. Goldberg. Orienting polygonal parts without sensors. *Algoritmica*, 10(2):201–225, 1993.

[14] K. M. Lynch and M.T. Mason. Stable pushing: Mechanics, controllability, and planning. *Int. J. of Robotics Research*, 6(15):533–556, 1996.

[15] M. Mason. *Manipulator grasping and pushing operations.* PhD thesis, MIT, 1982. Published in *Robot Hands and the Mechanics of Manipulation*, MIT Press, Cambridge, MA, 1985.

[16] B.K. Natarajan. An algorithmic approach to the automated design of parts orienters. In *IEEE Annual Symposium on Foundations of Computer Science*, pages 132–142, 1986.

[17] M.A. Peshkin and A.C. Sanderson. The motion of a pushed sliding workpiece. *IEEE Journal of Robotics and Automation*, 4(6):569–598, 1988.

[18] M.A. Peshkin and A.C. Sanderson. Planning robotic manipulation strategies for workpieces that slide. *IEEE Journal of Robotics and Automation*, 4(5):524–531, 1988.

[19] A. Rao and K. Goldberg. Manipulating algebraic parts in the plane. *IEEE Transactions on Robotics and Automation*, 11, 1995.

[20] A.F. van der Stappen, K. Goldberg, and M. Overmars. Geometric eccentricity and the complexity of manipulation plans. Technical report, UU-CS-1996-49, Dept. of Computer Science, Utrecht University, 1996. To appear in *Algorithmica.*

[21] J. Wiegley, K. Goldberg, M. Peshkin, and M. Brokowski. A complete algorithm for designing passive fences to orient parts. *Assembly Automation*, 17(2):129–136, August 1997.