Touring Polygons, Parts Cutting, and q-Rectangles

Fajie Li and Reinhard Klette

Computer Science Department, The University of Auckland, Tamaki Campus
Auckland, New Zealand.

Abstract

Given a sequence of k simple polygons in a plane, a start point p, and a target point q. We approximately compute a shortest path that starts at p, then visits each of the polygons in the specified order, and finally ends at q. So far no solution was known if the polygons are pairwise disjoint and non-convex. By applying a rubberband algorithm, we give an approximative algorithm with time complexity in $\kappa(\varepsilon) \cdot \mathcal{O}(n)$, where n is the total number of vertices of the given polygons, and function $\kappa(\varepsilon)$ is as

$$\kappa(\varepsilon) = (L_0 - L)/\varepsilon \tag{1}$$

where L_0 is the length of the initial path, and L is the true (i.e., optimum) path length.
The given rubberband algorithm can also be applied to solve approximately three NP-complete or NP-hard 3D Euclidean shortest path (ESP) problems in time $\kappa(\varepsilon) \cdot \mathcal{O}(k)$, where k is the number of layers in a stack which contains the defined obstacles.

Keywords: rubberband algorithm, shortest paths, touring polygons, parts cutting, q-rectangles.

1 Introduction

This paper reports about applications of a rubberband algorithm, which was originally proposed in [1], and then studied in detail in [12]. Before specifying the rubberband algorithm here in this paper, at first we describe two closely related problems, the touring polygons problem (TPP) and the parts cutting problem, whose approximate solutions will be given in this paper. Section 2 then presents the basic principle and a few issues of the rubberband algorithm, and Section 3 illustrates its applications to TPP and q-rectangles. Section 4 concludes the paper.

1.1 Touring Polygons Problem

We recall some notations from [4], which introduced the touring polygons problem. Let π be a plane, which is identified with $\mathbb{R}^2$. Consider polygons $P_i \subset \pi$, where $i = 1, 2, \ldots, k$, and two points p, $q \in \pi$. Let $p_0 = p$ and $p_{k+1} = q$. Let $p_i \in \mathbb{R}^2$, where $i = 1, 2, \ldots, k$. Let $\rho(p, p_1, p_2, , \ldots, p_k, q)$ denote the path $pp_1p_2 \ldots p_kq \subset \mathbb{R}^2$. Let $\rho(p, q) = \rho(p, p_1, p_2, , \ldots, p_k, q)$ if this does not cause any confusion. If $p_i \in P_i$ such that p_i is the first (i.e., along the path) point in $\partial P_i \cap \rho(p, p_i)$, then we say that path $\rho(p, q)$ *visits* P_i at p_i, where $i = 1, 2, \ldots, k$.

The *unconstrained* TPP is defined as follows:

How to find a shortest path $\rho(p, p_1, p_2, , \ldots, p_k, q)$ such that it visits each of the polygons P_i in the given order $i = 1, 2, \ldots, k$?

Assume that for any i, $j \in \{1, 2, \ldots, k\}$, $\partial P_i \cap \partial P_j = \emptyset$, and each P_i is convex; this special case is dealt with in [4]. The given algorithm runs in $\mathcal{O}(kn\ log(n/k))$ time, where n is the total number of all vertices of all polygons $P_i \subset \pi$, for $i = 1, 2, \ldots, k$.

According to [4], "one of the most intriguing open problems" identified by their results "is to determine the complexity of the TPP for (pairwise) disjoint nonconvex simple polygons".

Algorithm 2 in Section 3.1 answers this problem by providing an approximate algorithm running in time $\kappa(\varepsilon) \cdot \mathcal{O}(n)$, where n is the total number of vertices of all polygons.

1.2 The Parts Cutting Problem

In a variety of industries, such as clothing, window manufacturing, or mechanic, it is necessary to cut a set of parts (modeled by polygons) from large sheets of paper, cloth, glass, metal, and so forth. Motivated by such applications, [9] introduced the following three restrictions for cutting scenarios:

1. *The continuous cutting problem*: here the path of the cutting tool visits each object (i.e., polygon) to be cut just once. The tool can start an object cut at any point on its frontier, but must cut the

entire object before it travels to the next object. (Accordingly, the same frontier point must be used for entry and departure from the object.)

2. *The endpoint cutting problem*: here the tool can enter and exit the object only at some predefined frontier points; however, it may cut the object in sections (i.e., it may visit an object repeatedly).

3. *The intermittent cutting problem*: this is the most general version of the problem in which the object can be cut in sections and there is no restriction on the points that can be used for entry or exit.

[9] focused on the continuous cutting problem where each object is a polygon. They also called this problem the *plate-cutting traveling salesman problem* (P-TSP).

It is a generalization of the well-known traveling salesman problem (TSP) [11]. The P-TSP further generalizes the generalized TSP (GTSP) [10, 14]. If each polygon degenerates into a single vertex, then P-TSP becomes the TSP which is known to be NP-hard [5]. It follows that P-TSP is NP-hard as well.

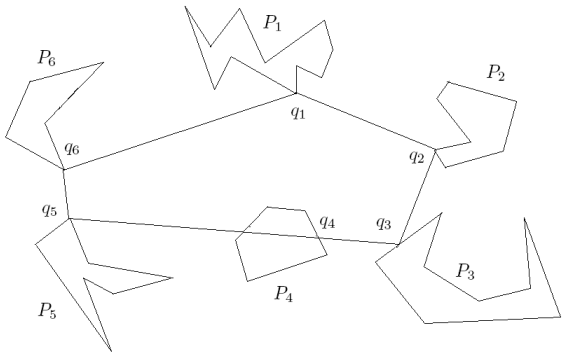

Figure 1: Illustration for the simplified P-TSP in [9] where polygons are assumed to be given in a particular order.

The difficulty of the P-TSP caused [9] to consider the problem with an additional condition, assuming now that all polygons are given in a particular order (see Figure 1). [9] then solved this simplified P-TSP by a heuristic approach based on a Lagrange relaxation method [6, 8], without providing a complexity analysis for this proposed approach. As a follow-up of the work in [9], [3] then proved that a further simplified P-TSP (i.e., only convex polygons, and a particular order of those polygons) is solvable in polynomial time (see Figure 2). If, additionally, the start point is also given (see Figure 3), then [4] claim that they can solve the same problem in time $\mathcal{O}(kn\ log(n/k))$, where n is the total number of vertices of all polygons, $\partial P_i \subset \pi$, and $i = 1, 2, \ldots, k$.

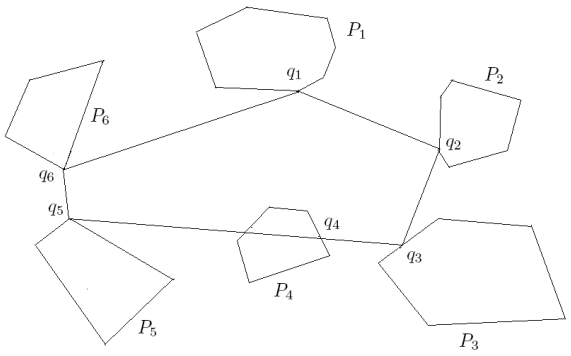

Figure 2: Illustration for the further simplified P-TSP in [3] also assuming convex polygons.

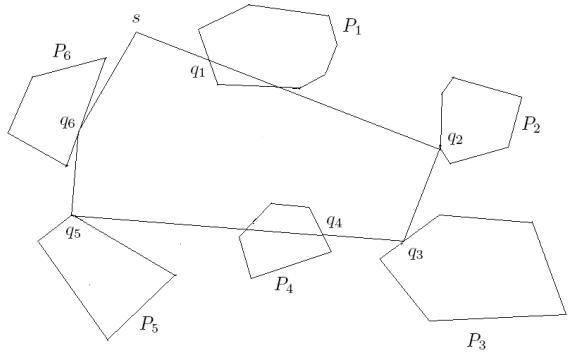

Figure 3: Illustration for the P-TSP as considered in [4], now also with a given start point s.

2 Rubberband Algorithm

We explain the basic idea of a rubberband algorithm using the following very simple 2D example:

Assume a degenerate case of an *unconstrained* TPP where each polygon P_i contracts into a single line segment s_i, where $i = 1, 2, \ldots, k$; see Figure 4. The following algorithm is a simplified version ("arc version") of a *rubberband algorithm*. It illustrates the basic principle of the original rubberband algorithm.

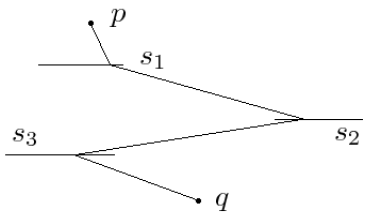

Figure 4: Degenerate case of the TPP, for $k = 3$.

Algorithm 1

1. Let $\varepsilon = 10^{-10}$ (the chosen accuracy).
2. Compute the length l_1 of the initial path $\rho = \langle p, p_1, p_2, \ldots, p_k, q\rangle$.
3. Let $q_1 = p$ and $i = 1$.
4. While $i < k - 1$ do:

4.1 Let $q_3 = p_{i+1}$.

4.2 Compute a point $q_2 \in s_i$ such that $d_e(q_1, q_2) + d_e(q_3, q_2) = \min\{d_e(q_1, q) + d_e(q_3, q) : q \in s_i\}$.

4.3 Update ρ by replacing p_i by q_2.

4.4 Let $q_1 = p_i$ and $i = i + 1$.

5.1 Let $q_3 = q$.

5.2 Compute $q_2 \in s_k$ such that $d_e(q_1, q_2) + d_e(q_3, q_2) = \min\{d_e(q_1, q) + d_e(q_3, q) : q \in s_k\}$.

5.3 Update ρ by replacing p_k by q_2.

6. Compute the length l_2 of the updated path $\rho = \langle p, p_1, p_2, \ldots, p_k, q\rangle$.
7. Let $\delta = l_1 - l_2$.
8. If $\delta > \varepsilon$, then let $l_1 = l_2$ and go to Step 3. Otherwise, Stop.

The accuracy parameter in Step 1 can be chosen such that maximum possible numerical accuracy is guaranteed on the given computer.

In the rest of this paper, we call

$$\{s_1, s_2, \ldots, s_k\}$$

the *step set* of the rubberband algorithm, and each s_i is a *step element* of the rubberband algorithm, where $i = 1, 2, \ldots, k$.

A step set may be such that two of the steps do have identical endpoints. For example, let the input for Algorithm 1 be as follows (see also Figure 5):

$s_1 = q_1q_2$, $s_2 = q_2q_3$, $q_1 = (0, 0)$, $q_2 = (2, 4)$, $q_3 = (3, 0)$, $p = (1, 0)$, and $q = (2, 0)$.

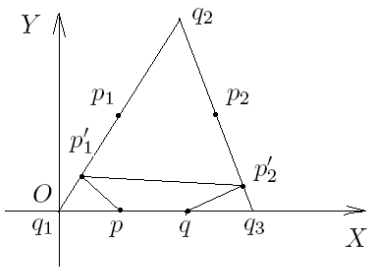

Figure 5: Illustration for identical endpoints of steps.

I	δ
1	-0.8900
2	-0.1752
3	-0.0019
4	$-1.2935e-005$
5	$-8.4435e-008$
6	$-5.4930e-010$
7	$-3.5740e-012$

Table 1: Number I of iterations and resulting δs for the initialization illustrated by Figure 5 [i.e., with $p_1 = (1, 2)$ and $p_2 = (2.5, 2)$ as initialization points].

To initialize, let p_1 and p_2 be the centers of s_1 and s_2, respectively [i.e., $p_1 = (1, 2)$, and $p_2 = (2.5, 2)$]. We obtain that the length of the initialized polyline $\rho = \langle p, p_1, p_2, q\rangle$ is equal to 5.5616. Algorithm 1 finds the shortest path $\rho = \langle p, p'_1, p'_2, q\rangle$ where $p'_1 = (0.3646, 0.7291)$, $p'_2 = (2.8636, 0.5455)$ and the length of it is equal to 4.4944 (see Table 1, which lists resulting δs for the number I of iterations).

Now we assume a different initialization, such that $p_1 = p_2 = q_2$; in this case, the output of Step 4.2 in Algorithm 1 will be wrong: the calculated path equals $\rho = \langle p, p'_1, p'_2, q\rangle$, where $p'_1 = q_2$ and $p'_2 = q_2$, and its length equals 8.1231. (Referring to Lemma 16 of [12], we see that $p_1 \neq p_0$ and p_2 in this example.)

We call a situation as in this initialization example a *degenerate path* within an application of a rubberband algorithm, and it may occur within initialization, or at a later iteration of the algorithm. In general, it is defined by the occurrence of at least two identical vertices of an initial or updated polygonal path. Such a degenerate case causes Step 4.2 in Algorithm 1 to fail.

A degenerate path can be dealt with approximately: we will not allow $p_2 = q_2$. To do so, we remove sufficiently small segments from both segments s_1 and s_2. The following shows how to handle such a degenerate case (for example) for the assumed input data.

We modify the initial values of x_1 and x_2, and of y_1 and y_2 as follows:

$$\begin{aligned} \delta' &= 2.221 \times 10^{-16} \quad \text{(see reason below)} \\ x_1 &= 2 - \delta' \quad \text{and} \quad y_1 = 2 \times x_1 \\ x_2 &= 2 + \delta' \quad \text{and} \quad y_2 = -4 \times (x_2 - 3) \\ p_1 &= (x_1, y_1) \quad \text{and} \quad p_2 = (x_2, y_2) \end{aligned}$$

Furthermore, let the accuracy be

I	δ	I	δ	I	δ	I	δ
1	-5.4831e-007	7	-1.2313	13	-7.0319e-010	19	8.8818e-016
2	-6.2779e-006	8	-2.0286	14	-4.5732e-012	20	8.8818e-016
3	-7.7817e-005	9	-0.2104	15	-3.0198e-014	21	-8.8818e-016
4	-9.6471e-004	10	-0.0024	16	-8.8818e-016	22	8.8818e-016
5	-0.0119	11	-1.6550e-005	17	8.8818e-016	23	-8.8818e-016
6	-0.1430	12	-1.0809e-007	18	-8.8818e-016	24	0

Table 2: Number I of iterations and resulting δs, for the step set shown in Figure 5, with $p_1 = (2-\delta', 2(2-\delta'))$ and $p_2 = (2+\delta', -4((2+\delta')-3))$ as initialization points and $\delta' = 2.221\text{e-}16$.

$$\varepsilon = 1.0 \times 10^{-100}$$

The length of the initialized polyline $\rho = \langle p, p_1, p_2, q\rangle$ is equal to 8.1231. Algorithm 1 will calculate the shortest path $\rho = \langle p, p_1', p_2', q\rangle$, where $p_1' = (0.3646, 0.7291)$ and $p_2' = (2.8636, 0.5455)$, and its length equals 4.4944 (see Table 2 for resulting δs in dependence of the number I of iterations).

Of course, if we leave the accuracy to be equals

$$\varepsilon = 1.0 \times 10^{-10}$$

then the algorithm will stop sooner, after less iterations. The algorithm was implemented on a Pentium 4 PC using Matlab 7.04. If we change the value of δ' into

$$\delta' = 2.22 \times 10^{-16}$$

then we obtain the same wrong result as that for identical points $p_1 = p_2 = q_2$. This is because the computer is not able to recognize a difference between x_1 and $x_1 \mp 2.22 \times 10^{-16}$. However, for practical applications, the value

$$\delta' = 2.221 \times 10^{-16}$$

should be small or accurate enough in general (in this particular implementation environment).

3 Applications

We approximately solve two computational problems using particular versions of a rubberband algorithm.

3.1 Touring of Polygons

The main algorithm (Algorithm 2) in this subsection answers the open problem mentioned in Section 1.1 by an approximate solution.

It is obtained as a modification of Algorithm 1. (The difference between Algorithm 1 and Algorithm 2 is defined by Steps 4.2 and 5.2.)

Algorithm 2

1. Let $\varepsilon = 10^{-10}$ (the accuracy).
2. Compute the length l_1 of the path $\rho = \langle p, p_1, p_2, \ldots, p_k, q\rangle$.
3. Let $q_1 = p$ and $i = 1$.
4. While $i < k - 1$ do:

4.1. Let $q_3 = p_{i+1}$.

4.2. Compute a point $q_2 \in \partial P_i$ (see the proof of Lemma 53 in [12]) such that $d_e(q_1, q_2) + d_e(q_3, q_2) = \min\{d_e(q_1, q') + d_e(q_3, q') : q' \in \partial P_i\}$ where ∂P_i is the frontier of polygon P_i.

4.3. Update ρ by replacing p_i by q_2.

4.4. Let $q_1 = p_i$ and $i = i + 1$.

5.1. Let $q_3 = q$.

5.2. Compute $q_2 \in \partial P_k$ such that $d_e(q_1, q_2) + d_e(q_3, q_2) = \min\{d_e(q_1, q) + d_e(q_3, q) : q \in \partial P_k\}$.

5.3. Update ρ by replacing p_k by q_2.

6. Compute the length l_2 of the updated path $\rho = \langle p, p_1, p_2, \ldots, p_k, q\rangle$.
7. Let $\delta = l_1 - l_2$.
8. If $\delta > \varepsilon$, then let $l_1 = l_2$ and go to Step 3. Otherwise, Stop.

See Figure 7 for an input example, and Figure 8 for a few measured time complexities, illustrating the $\kappa(\epsilon) \cdot \mathcal{O}(n)$ time complexity of the algorithm (see

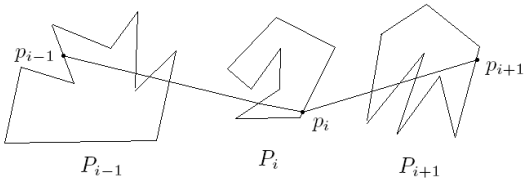

Figure 6: Illustration for the initialization for Steps 4.2 and 5.2 in Algorithm 2.

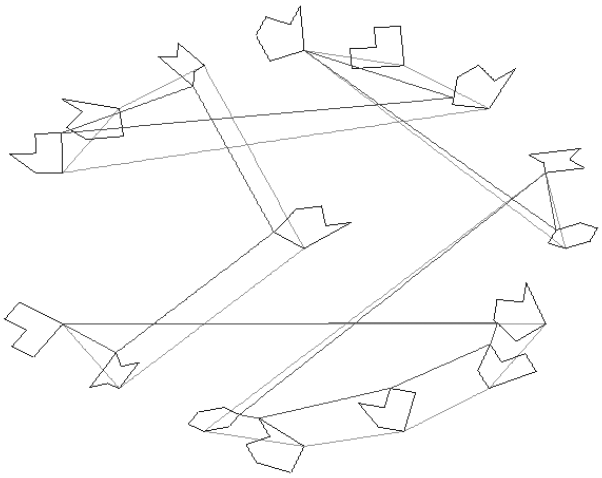

Figure 7: Input example for Algorithm 2.

[12] for a proof of correctness and time complexity). Algorithms for solving the safari, zookeeper, constrained TPP, and watchman route problem can be obtained as modifications of Algorithm 2.

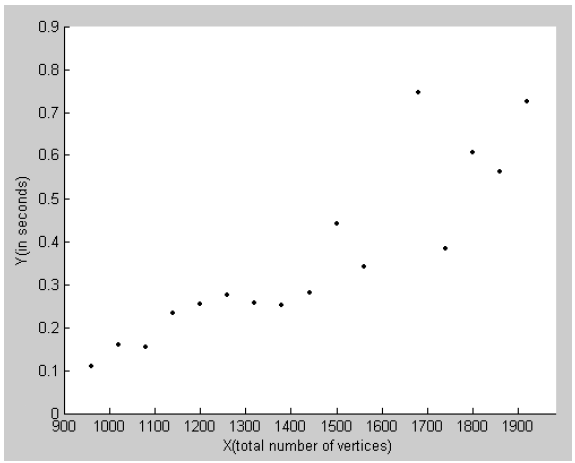

Figure 8: Measured run times for Algorithm 2.

The following procedure handles situations where the case of a degenerate path needs to be covered. Such a case may occur, and needs to be dealt with when we apply Algorithm 3 (which is a modified version of Algorithm 2) to the unconstrained TPP where polygons are not necessarily pairwise disjoint.

Procedure 1

Input: A point p and two polygons P_1 and P_2 such that $p \in \partial P_1 \cap \partial P_2$ (see Figure 9).

Output: A point $q \in \partial P_1$ such that $d_e(q,p) \leq \varepsilon$ and $q \notin \partial P_2$.

1. Let $\varepsilon = 10^{-10}$ (the accuracy).

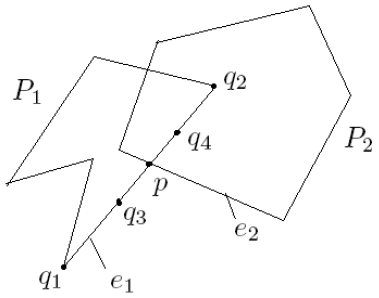

Figure 9: Illustration for Procedure 1.

2. Find a point $e_j \in E(P_j)$, where $j = 1, 2$, such that $p \in e_1 \cap e_2$.

3. Let $e_1 = q_1q_2$. Let q_3 and q_4 be two points in two segments q_1p and q_2p, respectively (see Figure 9) such that $d_e(q_j,p) \leq \varepsilon$ and $q_j \notin \partial P_2$, for $j = 3, 4$.

4. Let $q = \min\{q_3, q_4\}$ (with respect to lexicographic order).

5. Output q.

If there exist i, $j \in \{1, 2, \ldots, k\}$ such that $i \neq j$ and $\partial P_i \cap \partial P_{i+1} \neq \emptyset$, then we modify Algorithm 2 as follows: Let $p_0 = p$, $p_{k+1} = q$, $P_0 = p$, and $P_{k+1} = q$. (The difference between Algorithm 3 and Algorithm 2 is defined by Steps 4.1a and 5.1a.) Let $P = \{p, p_1, p_2, \ldots, p_k, q\}$.

Algorithm 3

1. Let $\varepsilon = 10^{-10}$ (the accuracy).

2. Compute the length l_1 of the path $\rho = \langle p, p_1, p_2, \ldots, p_k, q\rangle$.

3. Let $q_1 = p$ and $i = 1$.

4. While $i < k - 1$ do:

4.1a. If $(p_i = p_{i-1} \wedge p_i \neq p_{i+1}) \vee (p_i \neq p_{i-1} \wedge p_i = p_{i+1}) \vee (p_i = p_{i-1} \wedge p_i = p_{i+1})$, then apply Procedure 1 to compute a point p_i such that $p_i \neq p_{i-1}$ and $p_i \neq p_{i+1}$.

4.1. Let $q_3 = p_{i+1}$.

4.2. Compute a point $q_2 \in \partial P_i$ (see the proof of Lemma 57 in [12]) such that $d_e(q_1,q_2)+d_e(q_3,q_2) = \min\{d_e(q_1,q') + d_e(q_3,q') : q' \in \partial P_i\}$.

4.3. Update P by replacing p_i by q_2.

4.4. Let $q_1 = p_i$ and $i = i + 1$.

5.1a. If $(p_k = p_{k-1} \wedge p_k \neq p_{k+1}) \vee (p_k \neq p_{k-1} \wedge p_k = p_{k+1}) \vee (p_k = p_{k-1} \wedge p_k = p_{k+1})$, then apply Procedure 1 to compute a point p_k such that $p_k \neq p_{k-1}$ and $p_k \neq p_{k+1}$.

5.1b. Let $q_3 = q$.

5.2. Compute $q_2 \in \partial P_k$ such that
$d_e(q_1, q_2) + d_e(q_3, q_2) =$
$\min\{d_e(q_1, q') + d_e(q_3, q') : q' \in \partial P_k\}$

5.3. Update P by replacing p_k by q_2.

6. Compute the length l_2 of the updated path $\rho = \langle p, p_1, p_2, \ldots, p_k, q\rangle$.

7. Let $\delta = l_1 - l_2$.

8. If $\delta > \varepsilon$, then let $l_1 = l_2$ and go to Step 3. Otherwise, Stop.

Section 11.5 of [12] applies this algorithm to show that the TPP for not-necessarily pairwise disjoint nonconvex simple polygons can be approximately computed in polynomial time:

Theorem 1 ([12], Theorem 37) *The unconstrained TPP can be solved in $\kappa(\varepsilon) \cdot \mathcal{O}(n)$ time, where n is the total number of vertices of all polygons involved.*

Finding the exact solution of this problem is NP-hard due to the following

Theorem 2 ([4], Theorem 6) *The touring polygons problem (TPP) is NP-hard, for any Minkowski metric L_p ($p \geq 1$) in the case of nonconvex polygons P_i, even (already) in the unconstrained ($F_i = \mathbb{R}^2$) case with obstacles bounded by edges having angles of 0, 45, or 90 degrees with respect to the x-axis.*

3.2 q-rectangles

We denote by Π a *simple polyhedron* (i.e., a compact polyhedral region which is homeomorphic to a unit ball) in the 3D Euclidean space, which is equipped with an xyz Cartesian coordinate system. Let E the set of edges of Π; $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices of Π.

For $p \in \Pi$, let π_p be the plane which is incident with p and parallel to the xoy-plane. The intersection $\pi_p \cap \Pi$ is a finite set of simple polygons; a singleton (i.e., a single point) is considered to be a degenerate polygon. Let P be one of those simple polygons, defined by p and Π.

Any simple polygon P, being one connected component of $\pi_p \cap \Pi$, is a *critical polygon* (of Π, and with respect to p).

Any vertex p defines in general a finite set of critical polygons. We start with cases where any vertex p only defines one critical polygon, and this is even required to be convex:

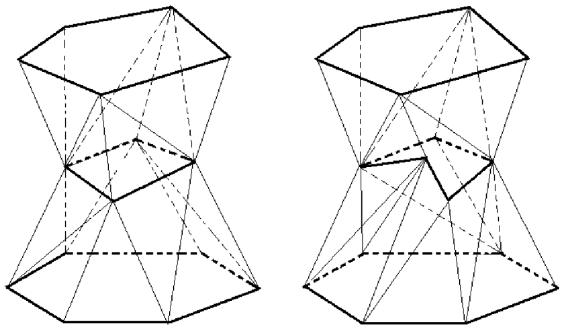

Figure 10: Left: a type 1 polyhedron. Right: type 2 polyhedron.

We say that a simple polyhedron Π is a *type 1* polyhedron iff any vertex p defines exactly one convex critical polygon. We say that a simple polyhedron Π is a *type 2* polyhedron iff any vertex p defines exactly one simple critical polygon.

Figure 10 shows a type 1 polyhedron on the left, and a type 2 polyhedron on the right.

In this section we apply Algorithm 2 to two special cases of ESP problems which can be solved efficiently. We also point out that the "complements" of these two problems are both NP-complete or NP-hard.

We generalize the notion of a critical polygon. We assume that a generalized Π is a simply connected (possibly unbounded) polyhedron, and we allow that the resulting (generalized) critical polygons are unbounded. For example, a generalized critical polygon may have a vertex at infinity, or it can be the complement of a critical polygon.

We recall some concepts introduced in [13]. Let (x_0, y_0, z_0) be a point in 3D space. Let

$$\begin{aligned} S_1 &= \{(x, y, z_0) : x_0 \leq x < \infty \wedge y_0 \leq y < \infty\} \\ S_2 &= \{(x, y, z_0) : -\infty < x \leq x_0 \wedge y_0 \leq y < \infty\} \\ S_3 &= \{(x, y, z_0) : -\infty < x \leq x_0 \wedge -\infty < y \leq y_0\} \\ S_3 &= \{(x, y, z_0) : x_0 \leq x < \infty \wedge -\infty < y \leq y_0\} \end{aligned}$$

S_i is called a *q-rectangle of type i*, where $i = 1$, 2, 3, 4. Furthermore, let (x_1, y_1, z_0) be a point in 3D space such that $x_1 > x_0$ and $y_1 > y_0$. Let

$$\begin{aligned} S_h &= \{(x, y, z_0) : -\infty < x < \infty \wedge y_0 \leq y \leq y_1\} \\ S_v &= \{(x, y, z_0) : x_0 \leq x \leq x_1 \wedge -\infty < y < \infty\} \end{aligned}$$

S_h (S_v) is called a *horizontal (vertical) strip*. Finally, let

$$\begin{aligned} S_{h_1} &= \{(x, y, z_0) : x_0 \leq x < \infty \wedge y_0 \leq y \leq y_1\} \\ S_{h_2} &= \{(x, y, z_0) : -\infty < x \leq x_0 \wedge y_0 \leq y \leq y_1\} \\ S_{v_1} &= \{(x, y, z_0) : x_0 \leq x \leq x_1 \wedge y_0 \leq y < \infty\} \\ S_{v_2} &= \{(x, y, z_0) : x_0 \leq x \leq x_1 \wedge -\infty < y \leq y_0\} \end{aligned}$$

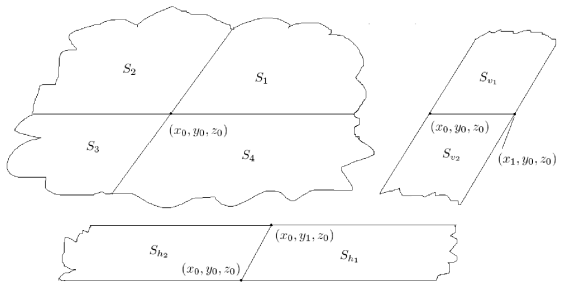

Figure 11: Axis-aligned rectangles.

According to their geometric shape, we notice that

> S_1 [S_2, S_3, S_4] is unbounded in direction $(+x, +y)$ [$(-x, +y)$, $(-x, -y)$, $(+x, -y)$];
>
> S_h [S_v] is unbounded in direction $\pm x$ [$\pm y$];
>
> S_{h_1} [S_{h_2}, S_{v_1}, S_{v_2}] is unbounded in direction $+x$ [$-x$, $+y$, $-y$].

S_i, S_h, S_v, S_{h_j}, and S_{v_j} are axis-aligned rectangles (see Figure 11), where $i = 1, 2, 3, 4$, and $j = 1, 2$. The stack $\mathcal{S}$ of axis-aligned rectangles is called *terrain-like* if, for at least one of the four directions $-x$, $+x$, $-y$, or $+y$, each rectangle in $\mathcal{S}$ is unbounded.

Example 1

Let Π be a simple polyhedron such that each generalized critical polygon is an axis-aligned rectangle. Let $p, q \in \Pi$ such that $p_z < q_z$. Let $V_{pq} = \{v : p_z < v_z < q_z \wedge v \in V\}$, where V is the set of vertices of Π. By Theorem 27 of [12], the Euclidean shortest path between p and q inside of Π can be computed in $\kappa(\varepsilon) \cdot \mathcal{O}(|V_{pq}|)$. Therefore, the 3D ESP problem can be solved efficiently in such a special case.

However, if we modify Π such that each generalized critical polygon is the complement of an axis-aligned rectangle, then the problem of finding the Euclidean shortest path between p and q inside of Π is NP-complete (!) because of the following

Theorem 3 ([13], Theorem 4) *It is NP-complete to decide wether there exists an obstacle-avoiding path of Euclidean length at most L among a set of stacked axis-aligned rectangles. The problem is (already) NP-complete for the special case that the axis-aligned rectangles are all q-rectangles of types 1 or 3.*

Example 2

We slightly modify Π as considered in Example 1: Let Π be a simply connected polyhedron such that each critical polygon is a triangle. By Theorem 27 of [12], the Euclidean shortest path between p and q inside of Π can be computed in $\kappa(\varepsilon) \cdot \mathcal{O}(|V_{pq}|)$ time. Therefore, the 3D ESP problem can be solved efficiently in such a special case.

However, if we modify Π such that each generalized critical polygon is the complement of a triangle, then the problem of finding the Euclidean shortest path between p and q inside of Π is NP-hard (!) because of the following

Theorem 4 (see [2]) *It is NP-hard to decide whether there exists an obstacle-avoiding path of Euclidean length at most L among a set of stacked triangles.*

We approximately solve these two very difficult problems, addressed in Theorems 3 and 4, in Section 3.3.

3.3 Three NP-complete or NP-hard Problems

As in Section 3.2, we again generalize the notion of a critical polygon, also allowing unbounded polygons. We now apply the generalized Algorithm 2 to approximately solve hard problems, characterized in Section 3.2 as being NP-complete or NP-hard.

Example 3

We modify Example 1 as follows: Let Π be a simply connected polyhedron such that each critical polygon is the complement of an axis-aligned rectangle. Let $p, q \in \Pi$ such that $p_z < q_z$. Let $V_{pq} = \{v : p_z < v_z < q_z \wedge v \in V\}$, where V is the set of all vertices of Π. By Theorem 32 of [12], the Euclidean shortest path between p and q inside of Π can be approximately computed in $\kappa(\varepsilon) \cdot \mathcal{O}(|V_{pq}|)$ time. Therefore, the 3D ESP problem can be approximately solved efficiently in such a special case.

Example 4

We modify Example 2 as follows (also just a slight modification of Π in Example 2: Let Π be a simply connected polyhedron such that each critical polygon is the complement of a triangle (or of a finite number of pairwise disjoint triangles). By Theorem 32 of [12], the Euclidean shortest path between p and q inside of Π can be approximately computed in $\kappa(\varepsilon) \cdot \mathcal{O}(|V_{pq}|)$ time.

Example 5

Let $\mathcal{S}$ be a stack of k horizontal or vertical strips. The Euclidean shortest path among $\mathcal{S}$ can be approximately computed in $\kappa(\varepsilon) \cdot \mathcal{O}(k)$ time. Finding

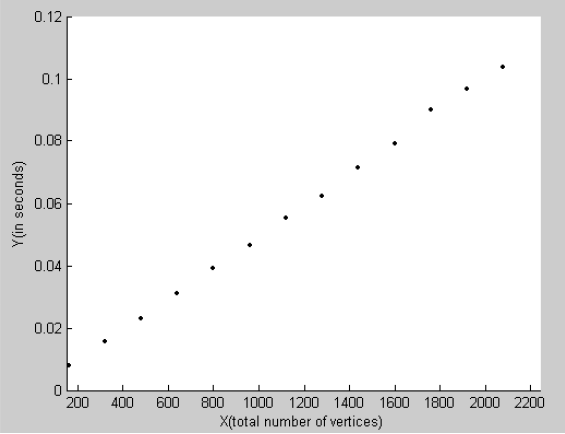

Figure 12: Illustration of measured run time for Algorithm 2 applied to q-rectangles.

the exact solution is very hard (NP-complete!) because of the following

Theorem 5 ([13], Theorem 5) *It is NP-complete to decide whether there exists an obstacle-avoiding path of Euclidean length at most L among a finite number of stacked horizontal and vertical strips.*

Example 6

Let $\mathcal{S}$ be a stack of k terrain-like axis-parallel rectangles. The Euclidean shortest path among $\mathcal{S}$ can be approximately computed in $\kappa(\varepsilon) \cdot \mathcal{O}(k)$ time. The best known algorithm for finding the exact solution has a time complexity in $\mathcal{O}(k^4)$ due to the following

Theorem 6 ([13], Theorem 6) *Let $\mathcal{S}$ be a stack of k terrain-like axis-parallel rectangles. The Euclidean shortest path among $\mathcal{S}$ can be computed in $\mathcal{O}(k^4)$ time.*

4 Conclusions

The paper presented at first in detail a simple rubberband algorithm for introducing into the basic ideas of this algorithmic approach. It explained how to deal with cases of degenerated paths (by means of an example, and also by presenting a general procedure for dealing with such cases).

Then the paper informed about various results when applying versions of rubberband algorithms, illustrating this way that rubberband algorithms define a widely applicable class of algorithms solving difficult geometric problems by approximate solutions in κ-linear time.

References

[1] T. Bülow and R. Klette. Digital curves in 3D space and a linear-time length estimation algorithm. *IEEE Trans. Pattern Analysis Machine Intelligence*, **24**:962–970, 2002.

[2] J. Canny and J.H. Reif. New lower bound techniques for robot motion planning problems. In Proc. *IEEE Conf. Foundations Computer Science*, pages 49–60, 1987.

[3] M. Dror. Polygon plate-cutting with a given order. *IIE Transactions*, **31**:271–274, 1999.

[4] M. Dror, A. Efrat, A. Lubiw, and J. Mitchell. Touring a sequence of polygons. In Proc. *STOC*, pages 473–482, 2003.

[5] M. R. Garey, R. L. Graham, and D. S. Johnson. Some NP-complete geometric problems. In Proc. *ACM Sympos. Theory Computing*, pages 10–22, 1976.

[6] A. M. Geoffrion. Lagrangean relaxation and its uses in integer programming. In *Mathematical Programming Study*, volume 2, pages 82–114. North Holland, Amsterdam, 1974.

[7] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, **2**:209–233, 1987.

[8] M. Guignard and S. Kim. Lagrangean decomposition: a model yielding stronger Lagrangean bounds. *Mathematical Programming*, **39**:215–228, 1987.

[9] J. Hoeft and U. S. Palekar. Heuristics for the plate-cutting traveling salesman problem. *IIE Transactions*, **29**:719–731, 1997.

[10] G. Laporte, H. Mercure, and Y. Nobert. Generalized traveling salesman problem through n clusters. *Discrete Applied Mathematics*, **18**:185–197, 1987.

[11] E. Lawler, J. Lenstra, A. Rinnooy Kan, and D. Shmoys. *The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization*. John Wiley and Sons, New York, 1985.

[12] F. Li and R. Klette. Exact and approximate algorithms for the calculation of shortest paths. IMA Minneapolis, Report 2141 on *www.ima.umn.edu/preprints/oct2006*.

[13] J. S. B. Mitchell and M. Sharir. New results on shortest paths in three dimensions. In Proc. *SCG*, pages 124–133, 2004.

[14] C. E. Noon and J. C. Bean. An efficient transformation of the generalized traveling salesman problem. *INFOR*, **31**:39–44, 1993.