A Robust and Fast Algorithm for Computing Exact and Approximate Shortest Visiting Routes

Håkan Jonsson

Department of Computer Science and Electrical Engineering, Luleå University of Technology, SE-971 87 Luleå, Sweden hj@sm.luth.se

Abstract. Given a simple n -sided polygon in the plane with a boundary partitioned into subchains some of which are convex and colored, we consider the following problem: Which is the shortest route (closed path) contained in the polygon that passes through a given point on the boundary and intersects at least one vertex in each of the colored subchains? We present an optimal algorithm that solves this problem in $O(n)$ time. Previously it was known how to solve the problem optimally when each colored subchain contains one vertex only. Moreover, we show that a solution computed by the algorithm is at most a factor $\frac{2+c}{c}$ times longer than the overall shortest route that intersects the subchains (not just at vertices) if the minimal distance between vertices of different subchains is at least c times the maximal length of an edge of a subchain. Without such a bound its length can be arbitrarily longer. Furthermore, it is known that algorithms for computing such overall shortest routes suffer from numerical problems. Our algorithm is not subject to such problems.

1 Introduction

Much research has been devoted to the study of shortest paths and algorithms that compute such paths during the recent years. In many problems the shortest paths must not only go free of obstacles but also visit a set of objects [1]. Maybe the most famous example is the *Traveling Salesman Problem* (TSP) [2, 3] in which the solution is the shortest route (closed path) that visits a given set of points in the plane. In fact, the literature contains a rich mix of TSP-like problems where the path and the objects must lie within a simple polygon [4]. These include the *Zookeeper's Problem* in which the objects are convex and connected to the boundary of the simple polygon, and the shortest route (the *shortest zookeeper route*) that visits the objects must not enter their interior [5, 6]. For these problems there also exists algorithms that compute provably good approximations [7, 8]. However, a major drawback with many of these algorithms is that they suffer from numerical problems [9].

In this paper we present a simple linear-time algorithm for a related path-problem. The algorithm is numerically robust and computes an approximate solution to the Zookeeper's Problem. In the related problem we are given a simple n -sided polygon with a boundary partitioned into subchains, some of

which are convex and colored, and asked for the shortest route contained in the polygon that passes through a given point on the boundary and intersects at least one vertex in each of the colored subchains. Previously it was known how to solve the problem if each colored subchain contains one vertex only. We show that the algorithm we present here computes a solution at most a factor $\frac{2+c}{c}$ times longer than the overall shortest route that intersects the subchains (not just at vertices) if the minimal distance between vertices of different subchains is at least c times the maximal length of an edge of a subchain. Note that without such a bound the length of the solution can be arbitrarily longer and that the overall shortest route mentioned here is a shortest zookeeper route for the colored subchains.

The algorithms by Jonsson [7] and Tan [8] compute approximations at most a constant times longer than the shortest zookeeper route in all cases. However, as we discuss in Section 5, the approximation factor of the first is worse when $c \geq \frac{2}{5}$ and the second suffer from numerical problems.

In our presentation below we first outline a rather inefficient $O(n^3)$ time algorithm using dynamic programming. By observing that the parts of different shortest paths that bridge consecutive colored subchains do not intersect and by proving that the involved length functions are convex we show, in Section 3.1, how to solve the problem in $O(n^2)$ time. The latter algorithm computes and concatenates many paths and using the fact that shortest paths from points close to each other are very similar we finally improve the running time of the algorithm to $O(n)$.

2 Preliminaries

The input to the algorithm consists of an n -sided simple polygon $\mathbf{P}$ located in the Euclidean plane and a designated point s on the boundary of $\mathbf{P}$. We assume that the boundary of $\mathbf{P}$ is oriented clockwise around its interior so that locally the interior is to the right during a traversal of the boundary. The polygon is defined by the coordinates of its vertices which are stored in an array and in the order they appear along the boundary. Each edge of $\mathbf{P}$ is represented by the vertices that delimit it. The point s is referred to as the *entrance*. It is stored together with a pointer to the edge that contains it.

The input also consist of a set of m pairwise disjoint and convex subchains $\{C_1, \dots, C_m\}$ of the boundary of $\mathbf{P}$ where the indices indicate the order in which the subchains are encountered during a clockwise scan of the boundary of $\mathbf{P}$. The subchains are indexed so that the entrance lies between C_m and C_1 . The order induces a natural order also on the vertices of a subchain so that there is always a first and a last vertex. Each subchain is represented by two indices into the array of polygon vertices that point to these extreme vertices. We use $v_{i1}, v_{i2}, \dots, v_{ik_i}$ to denote the vertices of subchain C_i indexed in clockwise order along $\mathbf{P}$, where k_i denotes the total number of vertices in C_i .

The algorithm outputs a (closed) path. We use $\pi(p, q)$ to denote the shortest path in $\mathbf{P}$ that connects p with q . The shortest path in $\mathbf{P}$ that starts at s , ends

at v_{ij} , and intersects at least one vertex in each of the subchains $C_1, C_2, \dots, C_i$ in this given order is denoted S_{ij} and referred to as a *shortest vertex path*. Each vertex of a subchain is stored with the shortest vertex path that ends at the vertex. Rather than explicitly storing the entire path S_{ij} at v_{ij} its length from s is stored together with a pointer to the first vertex in C_{i-1} it intersects; for $i = 1$, there is no preceding subchain and the pointers instead refer to the entrance s . The rationale behind this is that it reduces the storage needed to a total of $O(n)$ while it is still possible to reconstruct the actual path by tracing the pointers between subchains back to s and connecting them with shortest paths. We use S_i to denote the set of all shortest vertex paths S_{ij} for $j \in [1, \dots, k_i]$.

3 The Algorithm

The algorithm proceeds in five steps:

1. Compute the shortest paths in $\mathbf{P}$ from s to each of the vertices of C_1 . This gives us the paths S_1 .
2. FOR $i := 2$ TO m DO
 - (a) Compute S_i using S_{i-1} .
3. Compute the shortest paths in $\mathbf{P}$ from s to each of the vertices of C_m .
4. FOR $i := 1$ TO k_m DO
 - (a) Connect the shortest path in $\mathbf{P}$ from s to the vertex v_{mi} of C_m with the shortest vertex path to that vertex.
5. Report the shortest of the paths formed in the previous step as the result.

Steps 1 and 3 can readily be solved in $O(n)$ time using *shortest path trees* [10] after $\mathbf{P}$ has been triangulated in $O(n)$ time [11]. Given the paths computed in Step 3, Steps 4 and 5 are straightforward. However, in each iteration of Step 2 there are $O(n^2)$ shortest paths between vertices in the subchains and each of these paths have size $O(n)$. We next show that only $O(n)$ of these paths need to be considered and that we can find them in $O(n)$ steps. The issue of how to perform the computation of the shortest paths efficiently is treated in Section 3.2.

3.1 The Computation in Step 2

The computation in each iteration of Step 2 takes place locally in a subpolygon of $\mathbf{P}$ bounded by shortest paths. During iteration i this subpolygon is denoted $\mathbf{R}'_i$ and defined as the union of all shortest paths between C_i and C_{i-1} . $\mathbf{R}'_i$ is bounded by two additional convex chains apart from the subchains. They are the shortest path U_i between the first vertex of C_i and the last vertex of C_{i-1} , and the shortest path L_i connecting the first vertex of C_{i-1} with the last vertex of C_i . All four chains that bound $\mathbf{R}'_i$ bulge towards the interior of $\mathbf{R}'_i$. Since the shortest path in $\mathbf{P}$ that connects a given sequence of vertices can be computed in $O(n)$ time [5], we have:

Lemma 1. *All regions $\mathbf{R}'_i$ can be computed in a total of $O(n)$ time.*

In each iteration of Step 2, the algorithm traverses the chain C_i one vertex at a time starting at the first vertex. For each vertex v_{ij} encountered the vertex $v_{(i-1)k}$ in C_{i-1} through which S_{ij} passes is computed by traversing the chain C_{i-1} backwards in the direction towards its first vertex. The latter traversal continues as long as the length of the shortest visiting path from v_{ij} , via the vertices considered in C_{i-1} and further on to s along one of the chains in S_{i-1} , decreases. It halts whenever an additional step would increase its length or the first vertex of C_{i-1} is reached. That this gives the vertex through which S_{ij} pass follows from the following:

Lemma 2. *The length of the shortest path from v_{ij} on C_i via a point x on C_{i-1} that visits at least one (arbitrary) point on $C_1, \dots, C_{i-2}$ in order and ends at s is a convex function in x .*

Proof. The shortest path through x consists of two parts. One is the length of the shortest path from v_{ij} to a point x on the subchain C_{i-1} . Since C_{i-1} is convex it follows that the function is also convex (see [12–14]). The other function is the length of the shortest path $s_{i-1}(x)$ in $\mathbf{P}$ that starts at the point x , intersects at least one (arbitrary) point on $C_{i-2}, \dots, C_1$ in this order, and ends at s . For $i = 1$, $|s_1(x)|$ is convex since — as we argued above — it is the shortest path between a single point and a convex chain. For $i > 1$ assume that $|s_{i-2}(x)|$ is convex. By the *reflection principle*¹, and the fact the a shortest path is locally optimal at each point, the parts of the shortest visiting paths $s_{i-1}(x_1)$ and $s_{i-1}(x_2)$ where $x_1 \neq x_2$ diverge between C_{i-2} and C_{i-1} [12]. From this, and the convexity of the subchains, we conclude that the length of the part of $s_{i-1}(x_1)$ between C_{i-2} and C_{i-1} is convex as well. From this the lemma follows since the sum of two convex functions is in itself a convex function.

In fact, as a consequence of Lemma 2 there could be two vertices where the (minimal) lengths of the shortest paths are equal, in which case we choose the vertex closest to the first point of C_{i-1} (along C_{i-1}). In this case the minima itself is located between the vertices². When the shortest visiting path S_{ij} has been found, the traversal along C_i continues to the next vertex $v_{i(j+1)}$ and the vertex on C_{i-1} through which $S_{i(j+1)}$ passes is found (again) by traversing C_{i-1} . This is repeated until the end of C_i is reached. We have:

Lemma 3. *$S_{i(j+1)}$ intersects C_{i-1} either at the vertex where S_{ij} intersects C_{i-1} or at a vertex closer to the first vertex of C_{i-1} .*

Proof. Assume that contrary to the lemma S_{ij} intersects C_{i-1} closer to the first vertex of C_{i-1} than $S_{i(j+1)}$. This then means that the parts of S_{ij} and

¹ Attributed to Heron of Alexandria [15]. In optics, the reflection principle is also referred to as *Snell's law of reflection*, which was discovered by Willebrord van Roijen Snell[ius] in 1621 but not known until 1703 when Christiaan Huygens published Snell's result in his *Dioptrica* [16].

² This is also true in cases when there is but one vertex at which the length is minimized but where the minima does not coincide with the vertex.

$S_{i(j+1)}$ that go from C_i to C_{i-1} intersect. Let $v_{(i-1)k}$ be the vertex where $S_{i(j+1)}$ intersects C_{i-1} and let $v_{(i-1)(k+1)}$ be the vertex where S_{ij} intersects C_{i-1} . Then, since $S_{ij} = \pi(v_{ij}, v_{(i-1)(k+1)}) \cup S_{(i-1)(k+1)}$ is the shortest possible,

$$|\pi(v_{ij}, v_{(i-1)(k+1)})| + |S_{(i-1)(k+1)}| \leq |\pi(v_{ij}, v_{(i-1)k})| + |S_{(i-1)k}|. \quad (1)$$

It is well-known how to prove that the sum of the lengths of two opposing sides of a convex quadrilateral is less than the sum of the lengths of the diagonals. By similar reasoning, and the fact the $\mathbf{R}'_i$ is bounded by shortest paths, follows that $|\pi(v_{ij}, v_{(i-1)k})| + |\pi(v_{i(j+1)}, v_{(i-1)(k+1)})| < |\pi(v_{ij}, v_{(i-1)(k+1)})| + |\pi(v_{i(j+1)}, v_{(i-1)k})|$, which together with Eq. 1 gives us that

$$|\pi(v_{i(j+1)}, v_{(i-1)(k+1)})| + |S_{(i-1)(k+1)}| - |\pi(v_{i(j+1)}, v_{(i-1)k})| < |S_{(i-1)k}|. \quad (2)$$

But $S_{i(j+1)}$ is also a shortest possible path why $|\pi(v_{i(j+1)}, v_{(i-1)k})| + |S_{(i-1)k}| \leq |\pi(v_{i(j+1)}, v_{(i-1)(k+1)})| + |S_{(i-1)(k+1)}|$, from which we conclude that $|S_{(i-1)k}| \leq |\pi(v_{i(j+1)}, v_{(i-1)(k+1)})| + |S_{(i-1)(k+1)}| - |\pi(v_{i(j+1)}, v_{(i-1)k})|$ in contradiction to Eq. 2. From this the lemma follows.

Lemma 3 implies that there is no need to backtrack along C_{i-1} . Therefore, the traversal along C_{i-1} in one iteration of Step 2 continues from where the traversal halted in the previous iteration. Since the traversals both begin at one end of the subchains, we have:

Corollary 1. *C_i and C_{i-1} are traversed once each throughout the computation of the set S_i .*

From Corollary 1 follows that $O(n)$ shortest paths are considered throughout Step 2. Computing one of them individually can be done in $O(n)$ time [17]. However, this time complexity can be reduced by utilizing similarities between shortest paths that start and end close to each other. Below we show how to compute *all* paths in total $O(n)$ time.

3.2 Efficient computation of shortest paths

During the traversals performed in Step 2 the lengths of the shortest paths between the vertices considered are needed to judge when the optimal path has been found. As mentioned one could compute the shortest paths, and hence their lengths, from scratch when needed. We take another, more efficient, approach here: The shortest path from one vertex is obtained by modifying the shortest path from the closest preceding vertex. Assuming that the shortest path from the vertex where the traversal starts has already been computed we show how the shortest path from the other vertex is computed.

It is helpful to think of the computation as the movement of the start point p of a shortest path with fixed end point q in the other subchain along the edge that connects the two vertices while the shortest path $\pi(p, q)$ is maintained. Every shortest path has a *type* which is the ordered sequence of points at which

the path bends and finally ends. The type uniquely determines its path since a shortest path is the concatenation of line segments (shortest paths) between bends. Now, imagine that p is moved and consider how the path $\pi(p, q)$ and its type change. In general, what happens is that the first edge of the shortest path changes only. The rest of the path is unaffected by the move and the type stays invariant. However, at certain points either the first and second edges become colinear, in which case they are merged into one edge and the first bend disappears from the type, or the first edge hits a vertex of $\mathbf{P}$, in which case the first edge is split in two and a new bend is introduced in the type. The points where this happens are called *event points*.

At an event point the start point and the first two bends b_1 and b_2 of the shortest path are colinear³. There are four closely related variants depending on where in $\mathbf{R}'_i$ the bends reside:

- Both bends lie in L_i . (Fig. 1a)
- Both bends lie in U_i . (Fig. 1b)
- Bend b_1 lie in U_i while b_2 lie in L_i . (Fig. 1c)
- Bend b_1 lie in L_i while b_2 lie in U_i . (Fig. 1d)

Fig. 1. The four different kinds of event points e that might occur. The shortest paths from points p_1 and p_2 in the neighborhood of e have their first bends at different points (the shaded areas show some parts of the exterior of the polygon).

The continuous movement of p is broken down into a finite number of consecutive discrete movements between neighboring event points. A move is performed by first computing the next closest event point of each variant. The point p is then moved to the closest one and the type is updated accordingly, and the next movement is performed. This is repeated until p has reached the vertex.

³ We also consider q to be a bend.

We have informally described how one step in a traversal along an edge in one of the subchains transforms one shortest path into another by repeatedly computing event points. In fact, the union of shortest paths from event points to q is the shortest path map of q with respect to the edge along which p is moved. Recall that the shortest path map of a point on the boundary of a simple polygon is the partitioning of the polygon into regions such that the shortest paths from the point to any pair of points in a region bend at the same vertices [18]. Indeed, the procedure outlined above, where event points are computed, is an incremental construction of such a map. We still do not compute shortest path maps using the algorithms in the literature and there are two reasons for this. First of all it takes $O(n)$ time to compute a shortest path map from scratch and we need several maps which would be too costly. Second, and most important, the two traversals along C_i and C_{i-1} are performed in such a way that one end point of the shortest path is always fixed while the other is moved. Although our procedure and the algorithms for computing shortest path maps are closely related they are not the same. What we describe should be seen not as the construction of a single map rooted in q but merely as a part of the construction of a set of shortest paths between the subchains C_{i-1} and C_i during the traversals.

We now turn to the technical details on how to compute the event points efficiently. From now on we assume that p belongs to C_i and q belongs to C_{i-1} , and concentrate on the traversal along C_i ; the traversal along the other subchain is carried out analogous. To compute event points we make use of two shortest paths from p . One to q and one to the first vertex $v_{(i-1)1}$ of C_{i-1} . The two shortest paths $\pi(p, q)$ and $\pi(p, v_{(i-1)1})$ are related in that $\pi(p, v_{(i-1)1})$ lie between L_i and $\pi(p, q)$

Fig. 2. The four next potential event points on C_i (the four points on C_i below the point p) when the computation of the next shortest vertex path has reached p .

During the movement of p , when event points are computed, we maintain both shortest paths by computing event points based on both paths (Fig 2). As described above this can be done by extending edges incident to the first two bends, and compute their intersection with the edge that contain p .

Lemma 4. *The set S_i can be computed in $O(|\mathbf{R}'_i|)$ time.*

Proof. By Corollary 1, each vertex of $\mathbf{R}'_i$ is inserted and removed from the types of the paths at most once. Moreover, since the four polygonal chains that bound $\mathbf{R}'_i$ are convex, each of the vertices also appear as first and second bends at most once each. While being such a bend they define at most two event points (either as shown in Fig. 1a and 1c or Fig. 1b and 1d), by which the lemma follows.

Then, by Lemma 1, Lemma 4, and the fact that $\sum_i |\mathbf{R}'_i|$ belongs to $O(n)$, we finally have:

Theorem 1. *Given a simple n -sided polygon with a boundary partitioned into subchains some of which are convex and colored, the shortest route (closed path) contained in the polygon that pass through a given point on the boundary and intersects at least one vertex in each of the colored subchains can be computed in $O(n)$ time.*

4 Approximating a shortest zookeeper route

Our algorithm computes a path restricted to pass through vertices of the polygon. If this restriction is lifted we get a problem that has been studied extensively in the literature, namely the *Zookeeper's Problem* which asks for the shortest route in $\mathbf{P}$ that visits all convex and colored subchains in the boundary of $\mathbf{P}^4$.

It is easy to see that the route we compute can be arbitrary longer than a shortest zookeeper route in the worst case. Consider a polygon in which there is one subchain only with just one long edge and the entrance is located on some other edge of the polygon a small distance away from the mid-point of the subchain. Then the shortest zookeeper route merely follows the short path over to the mid-point (or another point on the subchain even closer) and back while the approximation is a path that goes far away to one of the vertices bounding the subchain and back again.

However, if there is a bound on the length of the edges of $\mathbf{P}$ our algorithm actually is a provably good approximation. Such bounds arise naturally in practical applications where an environment is sampled and the number of samples is much greater than the number of objects described or the objects are sampled in much more detail than the rest of the environment.

Let S_{opt} denote a shortest zookeeper route in $\mathbf{P}$ and let A_{opt} denote a solution computed by our algorithm. We then have:

⁴ The original Zookeeper's Problem asks for a route that intersects a set of *disjoint convex polygons* each of which shares an edge with $\mathbf{P}$ but since the route never enters the interior of the convex polygons, the original formulation and our is the same.

Lemma 5. *Let a be the longest distance between any pair of consecutive vertices in $\mathbf{P}$ that belong to the same chain. Then, if the distance from any vertex in a chain to any other vertex not in the same chain is at least ca , $\frac{A_{\text{opt}}}{S_{\text{opt}}}(c) \leq \frac{2+c}{c}$.*

Proof. The path S_{opt} visits one edge per chain (possibly at one of its end points). In fact, the path consists of $m + 1$ parts each of which spans between edges on two consecutive chains or connects the entrance and a subchain. To prove the lemma we show a bound on the length of the shortest path A that visits at least one end point of each of these edges compared with the length of S_{opt} . An upper bound on $|A|$ is then also an upper bound on $|A_{\text{opt}}|$ since $|A_{\text{opt}}| \leq |A|$.

Consider that part, of the $m + 1$ parts, that lie between $\overline{v_{(i-1)k}v_{(i-1)(k+1)}}$ on C_{i-1} and $\overline{v_{ij}v_{i(j+1)}}$ on C_i for some $i \in [1..m + 1]$, where the first and last edge are equal to s . Let S denote the part of S_{opt} that connect these edges.

S is a shortest path and therefore it lies between $\pi(v_{(i-1)k}, v_{i(j+1)})$ and $\pi(v_{(i-1)(k+1)}, v_{ij})$. Moreover, a maximum length shortest path between one of $\{v_{(i-1)k}, v_{(i-1)(k+1)}\}$ and one of $\{v_{ij}, v_{i(j+1)}\}$ follows parts of $\overline{v_{(i-1)k}v_{(i-1)(k+1)}}$, $\overline{v_{ij}v_{i(j+1)}}$, and S . The length of such a path is then at most $2a + |S|$ while $|S|$ is at least ca . The ratio $\frac{A}{S_{\text{opt}}}$ is maximized for $|S| = ca$ by which the lemma follows.

5 Numerical Robustness

It has been known for more than a decade that algorithms based on the reflection principle suffer from inherent numerical problems and all algorithms to date that compute exact solutions to the Zookeeper's Problem are no exceptions [9].

There are two previous algorithms that compute approximate zookeeper routes. The algorithm by Tan [8] achieves a better factor of approximation than the algorithm we present in this paper. Using our terminology, it computes the following points (called *images*) on the subchains: The point s_1 on C_1 that is closest to the start point s , the point s_2 on C_2 that is closest to s_1 , the point s_3 on C_3 that is closest to s_2 , and so on. Then the concatenation of the shortest paths connecting consecutive images is a zookeeper route at most $\sqrt{2}$ times longer than the shortest zookeeper route. However, it is not numerically robust. Consider a polygon that spirals inwards and in which the colored subchains are located such that the shortest path from each image s_{i-1} to s_i is a line segment which does not touch the boundary of the polygon. In this case, since the computed images are closest points on lines to points who are themselves closest points on lines, the computed result can be expected to exhibit poor numerical accuracy. The algorithm we present in this paper and the algorithm by Jonsson [7] makes use of input data (coordinates) and intersections between lines through polygon vertices. Neither of them experience these kinds of numerical problems.

6 Conclusions

We have presented a linear-time algorithm that computes a shortest visiting route for vertices in convex subchains that are contained in the boundary of a

simple polygon. If the subchains are described in much greater detail than the rest of the polygon, or if the distances between subchains are greater than the lengths of the edges of the cages, the computed route is an approximate solution to the Zookeeper's Problem.

It would be interesting to investigate further the influence of the shape and size of the polygon and the cages on the factor of approximation. Another important and intriguing open problem is whether it is possible to compute an exact solution to the Zookeeper's Problem in linear time or not.

References

1. Mitchell, J.S.B.: Shortest paths and networks. In Goodman, J.E., O'Rourke, J., eds.: Handbook of Discrete and Computational Geometry. CRC Press LLC (1997) 445–466
2. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B., eds.: The Traveling Salesman Problem. Wiley, New York, NY (1985)
3. Papadimitriou, C.H.: The Euclidean traveling salesman problem is NP-complete. Theoret. Comput. Sci. **4** (1977) 237–244
4. Jonsson, H.: The Euclidean Traveling Salesman Problem with Neighborhoods and a Connecting Fence. PhD thesis, Luleå University of Technology (2000)
5. Chin, W.P., Ntafos, S.: Optimum zookeeper routes. Info. Sci. **63** (1992) 245–259
6. Tan, X.: Shortest zookeeper's routes in simple polygons. Inform. Process. Lett. **77** (2001) 23–26
7. Jonsson, H.: An approximative solution to the Zookeeper's Problem. Information Processing Letters **87** (2003) 301–307
8. Tan, X.: Approximation algorithms for the watchman route and zookeeper's problems. Discrete Applied Mathematics **136** (2004) 363–376
9. Hershberger, J., Snoeyink, J.: An efficient solution to the zookeeper's problem. In: Proc. 6th Canad. Conf. Comput. Geom. (1994) 104–109
10. Guibas, L.J., Hershberger, J., Leven, D., Sharir, M., Tarjan, R.E.: Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. Algorithmica **2** (1987) 209–233
11. Chazelle, B.: Triangulating a simple polygon in linear time. Discrete Comput. Geom. **6** (1991) 485–524
12. Bepamyatnikh, S.: An $O(n \log n)$ algorithm for the Zoo-keeper's problem. Comput. Geom. Theory Appl. **24** (2002) 63–74
13. Guibas, L.J., Hershberger, J.: Optimal shortest path queries in a simple polygon. J. Comput. Syst. Sci. **39** (1989) 126–152
14. Hershberger, J.: A new data structure for shortest path queries in a simple polygon. Inform. Process. Lett. **38** (1991) 231–235
15. Toussaint, G.T.: Special issue on computational geometry. In: Proceedings of the IEEE. (1992) 1347–1363
16. Sabra, A.I.: Theories of Light from Descartes to Newton. Oldbourne, London (1967)
17. Lee, D.T., Preparata, F.P.: Euclidean shortest paths in the presence of rectilinear barriers. Networks **14** (1984) 393–410
18. Hershberger, J.: An optimal visibility graph algorithm for triangulated simple polygons. Algorithmica **4** (1989) 141–155