On-line Searching in Simple Polygons*

Sven Schuierer

Institut für Informatik
Universität Freiburg
Am Flughafen 17, Geb. 051
D-79110 Freiburg, Germany

email: schuiere@informatik.uni-freiburg.de

Technischer Bericht 116

November 1998

Abstract

In this paper we study the problem of a robot searching for a visually recognizable target in an unknown simple polygon. We present two algorithms. Both work for arbitrarily oriented polygons. The search cost is proportional to the distance traveled by the robot. We use competitive analysis to judge the performance of our strategies. The first one is a simple modification of Djikstra's shortest path algorithm and achieves a competitive ratio of $2n - 7$ where n is the number of vertices of the polygon. The second strategy achieves a competitive ratio of $1 + 2 \cdot (2k)^{2k}/(2k-1)^{2k-1}$ if the polygon is k-spiral. This can be shown to be optimal. It is based on an optimal algorithm to search in geometric trees which is also presented in this paper.

Consider a geometric tree with m leaves and a target hidden somewhere in the tree. The target can only be detected if the robot reaches the location of the target. We provide an algorithm that achieves a competitive ratio of $1 + 2m^m/(m-1)^{m-1}$ and show its optimality.

1 Introduction

An important problem in robotics is to plan a path from a starting point to a goal. Recently, this problem has been extensively investigated in the case that the robot does not have any prior knowledge of its environment and the position of the goal is unknown. In such a setting the robot gathers local information through its on-board sensors and tries to find a "short" path from the start point to the goal. The search of the robot can be viewed as an *on-line* problem since the robot has to make decisions about the search based only on the part of its environment that it has explored before. This invites the application of competitive analysis

*This research is supported by the DFG-Project "Diskrete Probleme", No. Ot 64/8-2.

to judge the performance of an on-line search strategy [ST85, PY91]. In competitive analysis we compare the distance traveled by the robot to the length of the shortest path from its starting point s to the goal g. The ratio of the distance traveled by the robot to the length of a shortest path from s to g maximized over all possible locations s and g is called the *competitive ratio* of the search strategy.

One of the fundamental problems in this setting is searching on m concurrent rays [BYCR93, Gal80]. Here a point robot is imagined to stand at the origin of m rays and one of the rays contains the goal g whose distance to the origin is unknown. The robot can only detect g if it stands on top of it. There is a simple optimal algorithm with a competitive ratio of $1 + 2m^m/(m-1)^{m-1}$. If the rays are allowed to contain branching vertices, then we obtain a geometric tree. Icking presents an algorithm to search in a geometric tree with m leaves that has a competitive ratio of $8m - 3$ [Ick94, Kle97]. This algorithm can be applied to search in a simple polygon and achieves a competitive ratio of $8n - 3$ if the polygon has n vertices.

In this paper we show how to adapt the optimal deterministic strategy to search on m concurrent rays to geometric trees, i.e., trees embedded in d-dimensional Euclidean space. We obtain an algorithm with a competitive ratio of $1 + 2m^m/(m-1)^{m-1} \leq 1 + 2em$ if the tree contains m leaves which can be shown to be optimal. The algorithm we obtain can be directly applied to searching in simple polygons which leads to a competitive ratio of $1 + 2 \cdot (2k)^{2k}/(2k-1)^{2k-1}$ if the polygon boundary can be decomposed into k convex and k reflex chains. It is not necessary for our algorithm to know the number k in advance. We also present a different strategy which is a modification of Dijkstra's shortest path algorithm with a competitive ratio of $2n - 7$.

Our algorithms work in any simple polygon. This should be contrasted with previous search algorithms that require restricted classes of polygons as their input [DHS95, DI94, Kle92, Kle94, LOS96, LOS97].

The paper is organized as follows. In the next section we introduce some definitions. In Section 3 we present the algorithm to search in a geometric tree and analyse its competitive ratio. Section 4 shows how the algorithm can be applied to searching in a simple polygon. It also introduces a second algorithm based on Dijkstra's algorithm for searching in a simple polygon. We conclude with some final remarks in Section 5.

2 Definitions

Let $\mathcal{C}$ be a simple closed curve consisting of n line segments such that no two consecutive segments are collinear. We define a *simple polygon* P to be the union of $\mathcal{C}$ and its interior. A vertex of P is the end point of a line segment on the boundary of P. A vertex is called *convex* if the subtending angle in the interior of P is less than $\pi/2$ and reflex otherwise.

Definition 2.1 *A* geometric tree $T = (V, E)$ *is a tree embedded into* $\mathbb{E}^d$ *such that each* $v \in V$ *is a point and each edge* $e \in E$ *is a polygonal path whose end points lie in* V*. The paths of* E *intersect only at points in* V*, and they do not induce any cycles.*

We only consider unbounded trees, that is, we assume that the leaves are rays that extend to infinity (see Figure 1).

If p and q are two points inside P, then we denote the line segment from p to q by $\overline{pq}$. The shortest path from p to q is denoted by $shp(p, q)$. The union of all shortest paths from

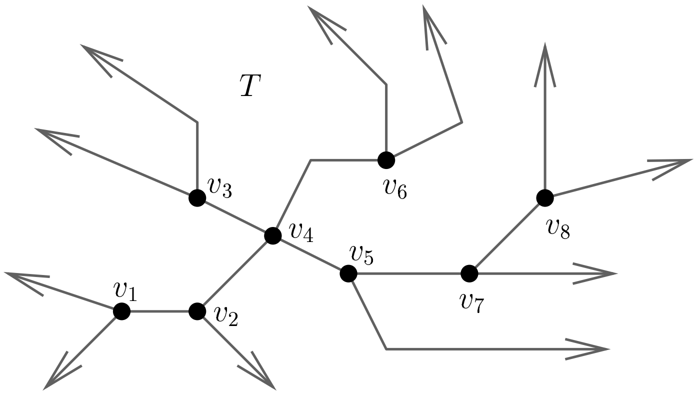

Figure 1: A geometric tree T embedded in $\mathbb{E}^2$ with 11 leaf rays and 8 vertices.

a fixed point p to the vertices of P forms a geometric tree called the *shortest path tree* of p [Kle97]. It is denoted by T_p.

If $\mathcal{P}$ is a path in P, then the *visibility polygon* $vis(\mathcal{P})$ of $\mathcal{P}$ is defined as set of all points in P that are seen by at least one point of $\mathcal{P}$. A maximal line segment of the boundary of $vis(\mathcal{P})$ that does not belong to the boundary of P is called a *window* of $vis(\mathcal{P})$. A window w splits P into two parts one of which contains $vis(\mathcal{P})$. The part that does not contain $vis(\mathcal{P})$ is called the *pocket of* w. In general, a line segment that touches the polygon boundary exactly in its end points is called a *chord*. A chord splits the polygon into two disconnected regions.

Let A be an algorithm to search for a target g in an environment class $\mathcal{E}$. We denote the length of the path traveled by the robot to find g using algorithm A starting at s in $E \in \mathcal{E}$ by $L_A(s, g, E)$ and the length of the shortest path from the robot location s to g by $L_{opt}(s, g, E)$. The algorithm A is called c-competitive if, for all environments $E \in \mathcal{E}$ and starting and goal positions $s, t \in E$,

$$L_A(s, t, E) \leq c \cdot L_{opt}(s, g, E).$$

The value c is called the *competitive ratio* of algorithm A.

3 Searching in Trees

In this section we consider the problem of searching in geometric trees. Before we describe our strategy to search in trees, we briefly discuss the optimal algorithm to search for a point in m concurrent rays.

3.1 Multi-way Ray Search

The model we consider is the following. A point robot is placed at the meeting point or origin of m concurrent rays and has to find a point g which is located in one of the rays. The distance of the point g is unknown to the robot but it is given a lower bound δ on the distance to g. In the following we assume without loss of generality that $\delta = 1$. The robot can detect the point g only when it reaches g. A deterministic strategy which achieves the optimal competitive ratio works as follows [BYCR93, Gal80]. The robot visits the rays one by one in a round robin fashion until the point g is reached. On every ray, the robot goes a certain distance and turns back if the point g is not found and explores the next ray. The distance from the origin that the robot walks before the i-th turn is determined by the function

$$f(i) \quad = \quad \left(\frac{m}{m-1}\right)^i,$$

that is, each time the robot visits a new ray it increases the distance traveled by a factor of $m/(m-1)$.

It is easy to see that the worst case ratio of the distance traversed by the robot to the actual distance of g from the origin (i.e., the competitive ratio of this strategy) is

$$1 + 2\frac{m^m}{(m-1)^{m-1}} \leq 1 + 2em.$$

For instance, if $m = 2$, then the competitive ratio is 9.

3.2 Geometric Trees

We assume that a point robot is located on an edge or at a vertex of the geometric tree T, and it has to find a target hidden somewhere in T. We assume that all the vertices of T have degree greater than 2 since a vertex of degree 2 can be removed by joining the polygonal paths of two edges into one.

At first it seems that the problem of searching in geometric trees can be solved trivially by using the optimal strategy to search on m rays if the tree has m leaves. However, there is one crucial difference. In the ray searching problem the number m of the rays to be searched is known in advance whereas in a geometric tree the number of branch vertices of the tree and their degree is unknown. Icking provides a simple way to avoid this problem by dividing the search into *phases* [Ick94]. In phase i all the branches are explored to a depth of 2^i. The phase ends when the last branch is visited. Then the search depth is doubled, and a new phase starts. It can be easily seen that the competitive ratio of this strategy is at most

$$\frac{2\sum_{i=0}^{k} m2^i + 2(m-1)2^{k+1} + 2^k}{2^k} \leq 8m - 3,$$

as the robot explores at most $m-1$ branches to a distance of 2^{k+1} before the target is detected on branch m at a distance of $2^k + \varepsilon$, for some $\varepsilon > 0$.

In the following we show that by adapting the strategy to search on m rays more carefully, one can achieve a competitive ratio of at most $1 + 2m^m/(m-1)^{m-1}$; that is, searching in a tree with m leaves has the same competitive ratio as searching on m concurrent rays.

We assume again that we are given a lower bound δ on the distance to the target as well as the distance to the closest branch vertex. In the applications that we present such a bound can be easily derived. As above we can set $\delta = 1$. Assume that in the beginning m_1 rays are incident to the root of the tree. We start off with the optimal search strategy to search m_1 rays. The initial step length is set to $d_1 = (m_1 - 1)^{m_1}/m_1^{m_1}$, that is, the robot explores only a fraction of the lower bound in the first m_1 steps. This guarantees that every ray has been visited at least once before a branch vertex is detected.

The strategy works as follows. Assume the currently explored tree has m leaves $l_1, \ldots, l_m$. The robot searches the m root-to-leaf paths by simulating the strategy to search on m concurrent rays. Let $\mathcal{P}_i$ be the path from the root to leaf l_i. The robot travels on each path $\mathcal{P}_i$

for a distance of d_i where d_i is the current step length and then returns to the origin. What happens if new edges branching off $\mathcal{P}_i$ are discovered is described below. The requirement that the robot should return to the origin is, of course, overly restrictive since the branching pattern of the tree often makes it unnecessary to return to the root. Instead, the robot only needs to return to the last vertex that belongs to both $\mathcal{P}_i$ and the new path $\mathcal{P}_{i+1}$ to be explored. While in practice a robot using our algorithm would of course make use of such shortcuts, it is conceptually more convenient to assume that the robot returns to the origin; that is, we treat the paths $\mathcal{P}_i$ as separate rays.

As indicated above, it may happen that the robot discovers new edges while traveling on $\mathcal{P}_i$. Once it discovers a new edge on $\mathcal{P}_i$, it reduces its step length d_i by a factor of $(1-1/m^2)^m$ and increases m by one where m is the number of leaves of the currently explored tree. This only happens the first time a new edge branching off $\mathcal{P}_i$ is discovered. All other new edges are just collected in a list without affecting d_i or m. After the robot has traversed a distance of d_i, it returns to the first vertex at which a new edge was detected. It chooses one of the newly discovered edges e, increases the step length by $m/(m-1)$ and follows e. If there are still other new edges that remain to be explored, then the step length d_i is again reduced by a factor of $(1-1/m^2)^{m-1}$ and m is increased by one before e is explored. Once all unexplored branches are exhausted, the robot returns to the origin and chooses the next root-to-leaf path.

A description of our strategy in a more algorithmic notation can be found in Table 1. We use $\lambda_{\mathcal{P}}$ to denote the explored length of a path $\mathcal{P}$.

Algorithm *Tree-Search* makes use of a couple of procedure calls which we explain in the following. The procedures *enqueue* and *dequeue* are the standard queue operations. The procedure *first* just returns the first element of list L. Finally, the procedure *shrink* is described below.

Note that the index i can be removed without affecting the computation of the algorithm. It is introduced in order to simplify the analysis of the algorithm. In particular, the assignments in Steps 35, 41, and 42 are unnecessary if the index i is removed. Moreover, the value d_i^* introduced in the algorithm does not affect the computation; in particular, the assignments and calls in Steps 9, 18, 23, 32, and 39 could be omitted. But again, keeping track of the value d_i^* simplifies the analysis.

We also need to describe how the robot follows the path $\mathcal{P}$ in Step 19 since the length of $\mathcal{P}$ may be much smaller than the current step length. So first the robot follows $\mathcal{P}$ to its end point unless Case 3 occurs before. Let p be the end point of $\mathcal{P}$, and assume the robot reaches p. If p is not a vertex, then p belongs to a polygonal path of an edge e, and the robot continues to follow this polygonal path. If p is a vertex of T, then Step 26 ensures that there is a unique edge e that is incident to p and that belongs to $\mathcal{P}$. The robot follows the polygonal path of e until one of the three cases in Algorithm *Tree-Search* occurs.

The procedure *shrink* is defined as follows.

Algorithm Shrink
Input: a list $L = [\mathcal{P}_1 \ldots, \mathcal{P}_k]$ consisting of paths $\mathcal{P}_j$ and two numbers α and δ
Output: the list L where the explored length of $\mathcal{P}_j$ is reduced by a factor of $\alpha^{j-\delta}$
1 **for** $j = 1$ **to** k **do** $\lambda_{\mathcal{P}_j} = \alpha^{j-\delta}\lambda_{\mathcal{P}_j}$

It is easy to see that algorithm *Tree Search* is correct since clearly all the edges of T are visited. If no new edges are discovered anymore, then the step length of the root-to-leaf paths increases exponentially by a factor of $m/(m-1)$ after each step. This guarantees that the target is eventually found.

Algorithm Tree-Search
Input: a geometric tree T with root r, a target g
Output: a path in T from r to g

1 let $e_1, \ldots, e_m$ be the edges incident to r
2 **for** $i = 1$ **to** m **do**
3   follow e_i for a distance of $\lambda_{e_i} = (1 - 1/m)^{m-i}$ and return
4 $i = m + 1$
6 $m_i = m$ $\triangleright$ m_i is the number of known leaves
7 $\alpha_i = m_i/(m_i - 1)$ $\triangleright$ the step length is multiplied by α_i in each new step
8 $d_i = \alpha_i \cdot 1$ $\triangleright$ d_i is the current step length
9 $d_i^* = (1/\alpha_i)^{m_i - 1}$ $\triangleright$ d_i^* is a lower bound for $d(r, g)$ on the current path
10 $E_i = \texttt{[]}$ $\triangleright$ E_i is a queue containing the newly encountered edges and paths, resp.
11 $L_i = [e_1, \ldots, e_{m_i}]$ $\triangleright$ L_i is a queue containing the known paths
12 **loop**
13   **if** $E_i = \texttt{[]}$ **then** $\delta_i = 0$ $\triangleright$ δ_i is an indicator variable
14         $\mathcal{P} = L_i.\mathit{dequeue}()$
15   **else** $\delta_i = 1$
16         $\mathcal{P} = E_i.\mathit{dequeue}()$
17   **if** $E_i \neq \texttt{[]}$ **then** $d_i = (1 - 1/m_i^2)^{m_i - 1} d_i$
18             $L_i.\mathit{shrink}(1 - 1/m_i^2, 1)$
19   follow $\mathcal{P}$ **until**
20   **Case 1**: g is detected: goto g; **exit loop**
21   **Case 2**: a new vertex v is reached:
22   **if** $E_i = \texttt{[]}$ **then** $d_i = (1 - 1/m_i^2)^{m_i} d_i$
23           $L_i.\mathit{shrink}(1 - 1/m_i^2, \delta_i)$
24           $v_i^* = v$
25   let $e_1, \ldots, e_k$ be the edges incident to v
26   **for each** edge e_j **do**
27   let $\mathcal{Q}_j$ be $\mathcal{P}$ concatenated with e_j
28       $E_i.\mathit{enqueue}(\mathcal{Q}_2, \ldots, \mathcal{Q}_k)$
29       $\mathcal{P} = \mathcal{Q}_1$
30   **go to** 19
31   **Case 3**: the distance traveled on $\mathcal{P}$ is at least d_i:
32       $\lambda_{\mathcal{P}} = d_i$
33       $L_i.\mathit{enqueue}(\mathcal{P})$
34   **if** $E_i = \texttt{[]}$ **then** return to the root r
35           $m_{i+1} = m_i$
36   **else** return to v_i^*
37           $m_{i+1} = m_i + 1$
38       $\alpha_{i+1} = m_{i+1}/(m_{i+1} - 1)$
39       $d_{i+1}^* = \lambda_{L_i.\mathit{first}()}$
40       $d_{i+1} = \alpha_{i+1} d_i$
41       $E_{i+1} = E_i$; $v_{i+1}^* = v_i^*$; $L_{i+1} = L_i$
42       $i = i + 1$
43 **end loop**;

Table 1: Algorithm *Tree Search.*

3.3 Analysis of Algorithm *Tree Search*

In order to analyse the strategy we show that a number of invariants are maintained. Recall that δ_i is an indicator variable that is defined as follows

$$\delta_i = \begin{cases} 0 & \text{if } E_i = \texttt{[]} \text{ in Step 12} \\ 1 & \text{otherwise.} \end{cases}$$

Invariant 1 (Algorithm *Tree Search*) *At Step 12 L_i contains $m_i - \delta_i$ paths $\mathcal{P}_j$, for $1 \leq j \leq m_i - \delta_i$, with*

$$\lambda_{\mathcal{P}_j} \;=\; \alpha_i^{j-1}\, d_i^*, \qquad \textit{and, in addition,} \qquad d_i = \alpha_i^{m_i - \delta_i}\, d_i^*. \tag{1}$$

Proof: In order to show Invariant 1 we first note that the invariant clearly holds the first time the loop is entered. In this case $\lambda_{P_j} = (1/\alpha_i)^{m_i - j}$, for $1 \leq j \leq m_i$, and $d_i^* = (1/\alpha_i)^{m_i - 1}$. So assume that the invariant holds before Step 12. We distinguish two cases.

Case 1 *$E_i = \texttt{[]}$ in Step 12.*
This implies that $\delta_i = 0$. In this case the first path $\mathcal{P}_1$ of L_i is removed in Step 14 and then explored. After Step 33 L_i consists of the paths $[\mathcal{P}_2, \ldots, \mathcal{P}_{m_i}, \mathcal{P}]$. We again distinguish two cases.

Case 1.1 *No new vertex v of T is encountered in Step 21 while the robot follows $\mathcal{P}$.*
The explored length of $\mathcal{P}$ is in Step 33

$$\lambda_{\mathcal{P}} \;=\; d_i = \alpha_i^{m_i}\, d_i^* = \alpha_i^{m_i - 1}\, \lambda_{\mathcal{P}_2}$$

where the last two equalities follow from the invariant. Since $d_{i+1}^* = \alpha_i\, d_i^*$, it is easy to see that L_{i+1} again satisfies the invariant if the elements of L_{i+1} are renumbered appropriately. After Step 40 the value of d_{i+1} is

$$d_{i+1} \;=\; \alpha_{i+1}\, \alpha_i^{m_i}\, d_i^* = \alpha_{i+1}^{m_{i+1}}\, d_{i+1}^*$$

since $m_{i+1} = m_i$ as claimed.

Case 1.2 *A new branch vertex v is encountered the first time while the robot follows $\mathcal{P}$.*
The robot executes Step 21 and, when the end of the current path is reached, Step 36. Hence, $m_{i+1} = m_i + 1$. When the procedure *shrink* is called in Step 22, the path $\mathcal{P}_j$ is the $(j-1)$st path in L_i and, therefore, $\lambda_{\mathcal{P}_j}$ is reduced by a factor of $(1 - 1/m_i^2)^{j-1}$, for $2 \leq j \leq m_i$. The distance d_i is reduced by a factor of $(1 - 1/m_i^2)^{m_i}$ in Step 22. By the invariant we have after Step 22

$$\left(1 - \frac{1}{m_i^2}\right)^{j-1} \lambda_{\mathcal{P}_j} \;=\; \left(\frac{(m_i+1)(m_i-1)}{m_i^2}\right)^{j-1} \left(\frac{m_i}{m_i - 1}\right)^{j-1} d_i^* = \alpha_{i+1}^{j-1} d_i^*, \tag{2}$$

for $2 \leq j \leq m_i$, and similarly $d_i = \alpha_{i+1}^{m_i}\, d_i^*$ after Step 22. After the path $\mathcal{P}$ is added to L_i in Step 33, the invariant for L_{i+1} holds again since now $E_{i+1} \neq \texttt{[]}$ and L_{i+1} contains $m_i = m_{i+1} - 1$ tuples which satisfy Condition 1. After d_i is multiplied with α_{i+1}, we see as follows that d_{i+1} also satisfies Invariant 1 again. We have

$$d_{i+1}^* = \lambda_{\mathcal{P}_2} = \frac{m_i + 1}{m_i} d_i^* = \alpha_{i+1}\, d_i^*$$

and, hence,

$$d_{i+1} \;=\; \alpha_{i+1}\alpha_{i+1}^{m_i}d_i^* = \alpha_{i+1}^{m_{i+1}-\delta_{i+1}}d_{i+1}^*.$$

Moreover, L_{i+1} now contains the $m_{i+1}-\delta_{i+1}$ paths $\mathcal{P}'_1,\ldots,\mathcal{P}'_{m_{i+1}-1}$ where $\mathcal{P}'_i = \mathcal{P}_{i+1}$, for $1 \le i \le m_{i+1}-2$, and $\mathcal{P}'_{m_{i+1}-1} = \mathcal{P}$. By Equation 2 we have

$$\lambda_{\mathcal{P}'_j} = \lambda_{\mathcal{P}_{j+1}} = \alpha_{i+1}^{j}d_i^* = \alpha_{i+1}^{j-1}d_{i+1}^*$$

and the invariant holds.

Note that it may happen that d_i is reduced so much in Step 22 that the robot has actually already traveled farther than the new value of d_i. In this case the robot just remains at v and acts as if Case 3 occurs. If the new step length d_{i+1} is still less than $d(r,v)$, then there is again nothing to do, and the robot considers the next edge, and so on. Finally, it will encounter an edge for which d_i is larger than $d(r,v)$. To see this, note that if there have been k new edges in the beginning, then the new step length to explore the last edge is $(\frac{m_i+k}{m_i+k-1})^{m_i+k-1}d_{i+1}^* = (\frac{m_i+k}{m_i+k-1})^{m_i+k-1}(\frac{m_i+1}{m_i})d_i^*$ which is larger than $(\frac{m_i}{m_i-1})^{m_i}d_i^*$. Since $(\frac{m_i}{m_i-1})^{m_i}d_i^*$ is the step length of the iteration in which the robot encountered v, $d_i > d(v,r)$ in the last step.

Note also that the second time the robot encounters a new branch vertex, the new edges are only added to E_i and nothing else happens.

Case 2 *$E_i \neq$ `[]` in Step 12.*
In this case $\delta_i = 1$, and the first path $\mathcal{P}$ of E_i is removed in Step 16. Moreover, since $E_i \neq$ `[]`, L_i contains the $m_i - 1$ paths $[\mathcal{P}_1,\ldots,\mathcal{P}_{m_i-1}]$ and $d_i = \alpha_i^{m_i-1}d_i^*$. First assume that after $\mathcal{P}$ is removed from E_i, $E_i =$ `[]` and no new vertex is encountered. Hence, after $\mathcal{P}$ is added to L_i in Step 33 and d_i is increased by a factor of $\alpha_{i+1} = \alpha_i$, the invariant holds again.

If after the removal of $\mathcal{P}$ from E_i in Step 16 still $E_i \neq$ `[]` or $E_i =$ `[]` and a new vertex is encountered by traveling on $\mathcal{P}$, then $\mathcal{P}_j$ is the jth path in L_i and, therefore, $\lambda_{\mathcal{P}_j}$ is reduced by a factor of $(1-1/m_i^2)^{j-1}$, for $1 \le j \le m_i - 1$ in the procedure *shrink*. We see as above that, after L_i is shrunk and d_i is reduced,

$$\lambda_{\mathcal{P}_j} \;=\; \alpha_{i+1}^{j-1}d_i^* \quad \text{and} \quad d_i = \alpha_{i+1}^{m_i-1}d_i^*,$$

for $1 \le j \le m_i - 1$, since $m_{i+1} = m_i$. Of course, now $\mathcal{P}_1$ is the first path of L_i. And as in the second case above we see that after the path $\mathcal{P}$ is added to L_i, the invariant for L_{i+1} holds again; and after d_i is multiplied with α_{i+1}, d_{i+1} also satisfies Invariant 1 again.

□

Note that, for each path $\mathcal{P}$ in L_i, the robot has explored $\mathcal{P}$ at least up to $\lambda_{\mathcal{P}}$; this is true when $\mathcal{P}$ is added to L_i, and afterwards $\lambda_{\mathcal{P}}$ is only changed in the Steps 17 and 22 and in both steps $\lambda_{\mathcal{P}}$ is reduced.

Invariant 2 (Algorithm *Tree Search*)
If $E_i \neq$ `[]`, then at Step 12 $d(v_i^*, r) \ge d_i^*/\alpha_i$.

Proof: To see this invariant we first note that as long as $E_i \neq$ `[]`, $\mathcal{P}_1$ remains the first path in L_i, that is, d_i^* remains unchanged. Moreover, as can easily be seen the explored length of $\mathcal{P}_1$ is not reduced if the procedure *shrink* is invoked. Secondly, the only step where v_i^* is changed is Step 22. This happens if $E_i =$ `[]` and a new vertex is reached. If v_i^* was already defined before, then the search is started at v_i^* and the new vertex v_{i+1}^* has a distance to r that is greater than the distance of v_i^* to r.

Hence, $d(v_i^*, r)/d_i^*$ is only increased unless the robot returns to r. Therefore, it suffices to show that the inequality holds if $E_i =$ `[]` in Step 12 and a new vertex is encountered on a path starting at the root r. In this case the first new vertex that the robot encounters is on the part of $\mathcal{P}$ that comes after $\lambda_{\mathcal{P}}$. By Invariant 1 $\lambda_{\mathcal{P}}$ equals d_i^*. Since $\mathcal{P}$ is removed from L_i, the first element of L_{i+1} is now $\mathcal{P}_2$ and $d_{i+1}^* = \lambda_{\mathcal{P}_2}$. By Invariant 1 $\lambda_{\mathcal{P}_2}$ after the procedure *shrink* is given by $\alpha_{i+1}\, d_i^*$. Hence, before Step 42 $d(v_{i+1}^*, r) \geq d_i^* = d_{i+1}^*/\alpha_{i+1}$ which proves the invariant. $\square$

Invariant 3 (Algorithm *Tree Search*) *If in Step 12 the robot is at a distance of d_{cur} to r, then the total distance traveled by the robot is at most*

$$2d_i^* \sum_{j=-\infty}^{m_i-1-\delta_i} \alpha_i^j + d_{cur}.$$

Proof: The invariant clearly holds when the loop is entered in Step 12. In the following we show that if the invariant holds at the beginning of the loop, then it also holds at the end. We distinguish two cases.

Case 1 $E_{i+1} =$ `[]`.

In this case the robot returns to r, and no new vertex is discovered during the traversal of $\mathcal{P}$. We have $m_{i+1} = m_i$ and $\delta_{i+1} = 0.$. Once the robot reaches r, Invariant 3 implies that the total distance traveled by the robot is at most

$$2d_i^* \sum_{j=-\infty}^{m_i-1-\delta_i} \alpha_i^j + 2d_i, \tag{3}$$

where $d_i = \alpha_i^{m_i-\delta_i}\, d_i^*$ by Invariant 1. If $E_i \neq$ `[]`, then $\mathcal{P}_1$ is still the first element of L_{i+1}, $d_{i+1}^* = d_i^*$, and by Invariant 1 Expression 3 equals

$$2d_i^* \sum_{j=-\infty}^{m_i-2} \alpha_i^j + 2\alpha_i^{m_i-1} d_i^* \quad = \quad 2d_i^* \sum_{j=-\infty}^{m_i-1} \alpha_i^j = 2d_{i+1}^* \sum_{j=-\infty}^{m_{i+1}-1} \alpha_{i+1}^j.$$

Hence, the invariant holds.

If $E_i =$ `[]`, then $\delta_i = 0$, and $\mathcal{P}_2$ is the first element of L_{i+1} which implies that $d_{i+1}^* = \lambda_{\mathcal{P}_2} = \alpha_i\, d_i^*$ by Invariant 1. Expression 3 now equals

$$2d_i^* \sum_{j=-\infty}^{m_i-1} \alpha_i^j + 2\alpha_i^{m_i} d_i^* = 2d_{i+1}^* \sum_{j=-\infty}^{m_i-2} \alpha_i^j + 2\alpha_i^{m_i-1} d_{i+1}^* = 2d_{i+1}^* \sum_{j=-\infty}^{m_{i+1}-1} \alpha_j^j,$$

and the invariant holds again.

Case 2 $E_{i+1} \neq$ `[]`.
In this case the robot returns to v_i^*, i.e., $d_{cur} = d(v_i^*, r)$, $m_{i+1} = m_i + 1$, and $\delta_{i+1} = 1$.

First assume that $E_i =$ `[]`. Then, a new vertex is encountered while the robot follows $\mathcal{P}$. By Invariant 3 the total distance traveled by the robot is at most

$$2d_i^* \sum_{j=-\infty}^{m_i-1} \alpha_i^j + 2d_i - d(v_i^*, r), \tag{4}$$

since the robot starts at r, travels all the way to the end of $\mathcal{P}$ and then returns v_i^*. Here, $d_i = \alpha_{i+1}^{m_{i+1}-1} d_i^*$ by Invariant 1 and the fact that d_i has been shrunk in Step 22. As mentioned before, it may happen that the robot has traveled further than $\alpha_{i+1}^{m_{i+1}-1} d_i^*$ before the first new edge branching off $\mathcal{P}$ is discovered. In this case the robot only travels to v_i^* and stays there; now, the total distance traveled by the robot is at most

$$2d_i^* \sum_{j=-\infty}^{m_i-1} \alpha_i^j + d(v_i^*, r) \tag{5}$$

Since $E_i =$ `[]`, $\mathcal{P}_2$ is now the first element of L_i. In order to show that the invariant still holds at the end of the loop we need to prove that

$$2d_{i+1}^* \sum_{j=-\infty}^{m_{i+1}-1-\delta_{i+1}} \alpha_{i+1}^j + d_{cur} \tag{6}$$

is at least as large as Expression 4 or 5. Here, $d_{cur} = d(v_i^*, r)$, $\delta_{i+1} = 1$, and $d_{i+1}^* = \lambda_{\mathcal{P}_2} = \alpha_{i+1} d_i^*$ by Invariant 1 and the fact that $\mathcal{P}_2$ has been shrunk in Step 22. If we subtract Expression 4 from 6, then the term $2d_i = 2\alpha_{i+1}^{m_{i+1}-1} d_i^*$ cancels with $2\alpha_{i+1}^{m_{i+1}-2} d_{i+1}^*$, and we obtain

$$2d_i^* \left(\sum_{j=-\infty}^{m_i-1} \alpha_{i+1}^j - \sum_{j=-\infty}^{m_i-1} \alpha_i^j \right) + 2d(v_i^*, r) \geq 2d_i^* \left(\sum_{j=-\infty}^{m_i-1} \alpha_{i+1}^j - \sum_{j=-\infty}^{m_i-1} \alpha_i^j + 1 \right).$$

Similarly, if we subtract Expression 5 from 6, then we obtain

$$2d_i^* \left(\sum_{j=-\infty}^{m_i} \alpha_{i+1}^j - \sum_{j=-\infty}^{m_i-1} \alpha_i^j \right) \geq 2d_i^* \left(\sum_{j=-\infty}^{m_i-1} \alpha_{i+1}^j - \sum_{j=-\infty}^{m_i-1} \alpha_i^j + 1 \right).$$

We want to show that the above expression is non-negative. Since $d(v_i^*, r) \geq d_i^*$ by Invariant 2 and $\alpha_{i+1}^j \geq \alpha_i^j$, for $j < 0$, both of the above expressions are bounded from below by

$$2d_i^* \left(\sum_{j=0}^{m_i-1} \alpha_{i+1}^j - \sum_{j=0}^{m_i-1} \alpha_i^j + 1 \right) = 2d_i^* \left(\frac{\alpha_{i+1}^{m_i} - 1}{\alpha_{i+1} - 1} - \frac{\alpha_i^{m_i} - 1}{\alpha_i - 1} + 1 \right) = 2d_i^* f(m_i),$$

where

$$f(m) = (m+1) \left(\frac{m+1}{m} \right)^{m-1} - m \left(\frac{m}{m-1} \right)^{m-1}.$$

We now show that the function f is non-negative. In order to do so let us define the function

$$g(m,k) = m\left(\frac{m}{m-1}\right)^k,$$

with $k \geq 1$, $m \geq 2$. If we consider the partial derivative of g w.r.t. m, then we obtain

$$\frac{\partial g}{\partial m}(m,k) \quad = \quad \left(\frac{m}{m-1}\right)^k \frac{m-1-k}{m-1}$$

which has exactly one zero in the range $(1,\infty)$ at $m = k+1$. Since $\lim_{m\to 1+} g(m,k) = \lim_{m\to\infty} g(m,k) = \infty$, g has a minimum at $g(k+1,k)$. In particular, $g(m,m-1) \leq g(x,m-1)$, for all $x \in (1,\infty)$. Hence, $f(m) = g(m+1,m-1) - g(m,m-1) \geq 0$ as claimed. This implies that the distance traveled by the robot is, indeed, bounded by Expression 6.

Finally, we assume that $E_i \neq$ `[]`. By Invariant 3 the total distance traveled by the robot is at most

$$2d_i^* \sum_{j=-\infty}^{m_i-2} \alpha_i^j + 2d_i - d(v_i^*, r). \tag{7}$$

We want to show that

$$2d_{i+1}^* \sum_{j=-\infty}^{m_{i+1}-2} \alpha_{i+1}^j + d(v_i^*, r) \tag{8}$$

is at least as large as Expression 7. Since $E_i \neq$ `[]`, $\mathcal{P}_1$ is still the first element of L_i and, therefore, $d_{i+1}^* = \lambda(\mathcal{P}_1) = d_i^*$. Moreover, since by Invariant 1 $\alpha_{i+1}d_i = d_{i+1} = \alpha_{i+1}^{m_{i+1}-1} d_{i+1}^*$, $d_i = \alpha_{i+1}^{m_{i+1}-2} d_i^*$. If we subtract Expression 7 from 8, then we obtain again the sum

$$2d_i^* \left(\sum_{j=-\infty}^{m_i-1} \alpha_{i+1}^j - \alpha_i^j\right) + 2d(v_i^*, r)$$

which is non-negative as we have seen above which implies again that the distance traveled by the robot is, indeed, bounded by Expression 8.

□

3.3.1 The Competitive Ratio

Once we have established a bound on the distance traveled by the robot, it is easy to compute the competitive ratio of Algorithm *Tree Search*. So assume that the robot detects the target g. We again distinguish two cases. First assume that $E_i =$ `[]` at Step 12. Then, the current search path of the robot starts at the root r of T. By Invariant 1 the explored length of $\mathcal{P}$ is d_i^*. Hence, $d(r,g) > d_i^*$. On the other hand, by Invariant 3 the distance traveled by the robot is at most

$$2d_i^* \sum_{j=-\infty}^{m_i-1} \alpha_i^j + d(r,g).$$

Hence, the competitive ratio is given by

$$\begin{aligned}\frac{2d_i^* \sum_{j=-\infty}^{m_i-1} \alpha_i^j + d(r,g)}{d(r,g)} &\leq 1 + 2\frac{d_i^*(m_i-1)\alpha_i^{m_i}}{d_i^*} = 1 + 2\frac{m_i^{m_i}}{(m_i-1)^{m_i-1}} \\ &\leq 1 + 2\frac{m^m}{(m-1)^{m-1}}\end{aligned}$$

where m is the number of leaves of T.

Now assume that $E_i \neq$ [] at Step 12. Then, the current search path of the robot starts at the vertex v_i^*. By Invariant 2 the distance of v_i^* to r is at least d_i^*/α_i. Hence, $d(r,g) > d_i^*/\alpha_i$. On the other hand, by Invariant 3 the distance traveled by the robot is at most

$$2d_i^* \sum_{j=-\infty}^{m_i-2} \alpha_i^j + d(r,g).$$

Hence, the competitive ratio is given by

$$\begin{aligned}\frac{2d_i^* \sum_{j=-\infty}^{m_i-2} \alpha_i^j + d(r,g)}{d(r,g)} &\leq 1 + 2\frac{d_i^*(m_i-1)\alpha_i^{m_i-1}}{d_i^*/\alpha_i} = 1 + 2\frac{m_i^{m_i}}{(m_i-1)^{m_i-1}} \\ &\leq 1 + 2\frac{m^m}{(m-1)^{m-1}}\end{aligned}$$

as above.

The above algorithm is optimal since m concurrent rays also form a geometric tree. As we have seen before, $1 + 2m^m/(m-1)^{m-1}$ is a lower bound on the competitive ratio of searching on m concurrent rays [BYCR93, Gal80].

We have proven the following theorem.

Theorem 3.1 *There is a strategy to search in a geometric tree with m leaves that has a competitive ratio of $1 + 2m^m/(m-1)^{m-1}$. If we consider unbounded geometric trees, then there is no strategy with a lower competitive ratio.*

4 Searching in Simple Polygons

In this section we present an algorithm for a point robot to search for a target g inside an arbitrarily oriented simple polygon P. We assume that the robot is equipped with a vision system that allows it to identify the target and a range finder which returns the local visibility polygon of the robot. Our algorithm is based on an approach by Icking [Ick94, Kle97]. However, we present a new analysis of the strategy. We assume that in the beginning the robot is located at a point s in a polygon P with n vertices.

At first we observe that there is a simple lower bound of $\Omega(n)$ for searching in a simple polygon [Kle97]. If we consider the polygon shown in Figure 2, then we obtain a lower bound on the competitive ratio of $n/2-1$. By placing the target g in the last spike that is explored by the strategy, an adversary can force any strategy to generate a path of length $(2(n/4)-1)d$ whereas the shortest path from s to g has a length of $d+\varepsilon$. Here, d is the length of a spike in the polygon. This shows that no strategy can have a better competitive ratio than $n/2-1$ in an arbitrarily oriented simple polygons with n edges.

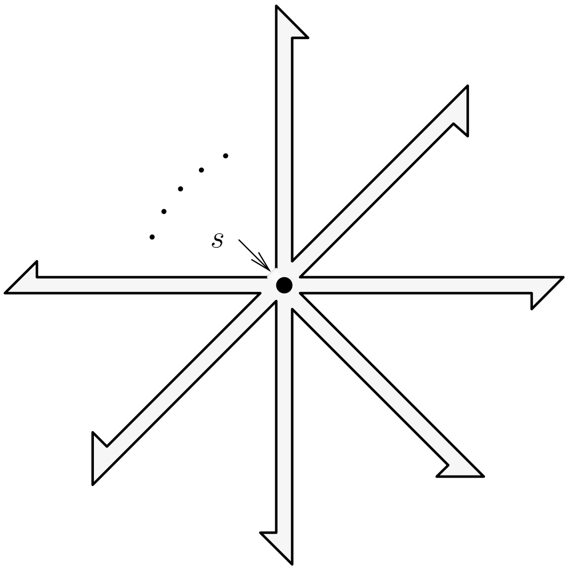

Figure 2: A polygon for which every search strategy has a competitive ratio of $n/2 - 1$.

Let T_s be the *shortest path tree* of s. Recall that the shortest path tree of s is defined as the union of all shortest paths from s to vertices of P. It consists of $n+1$ nodes, which are s and the n vertices of P, of which $l \leq n+1$ are leaves. We first show that it suffices to explore T_s in order to find g.

If g is not visible from s, then $shp(s, g)$ intersects a number of vertices. Let v be the last vertex that is intersected by $shp(s, g)$ (see Figure 3). Clearly, the line segment from v to g is contained in P. Once the robot has reached v, then it sees g and moves directly to g. Hence, if $L_A(s, v)$ denotes the length of the path that the robot travels in order to reach the vertex v using algorithm A, then the competitive ratio of A is given by

$$\frac{L_A(s,v) + d(v,g)}{d(s,v) + d(v,g)} \leq \frac{L_A(s,v)}{d(s,v)}.$$

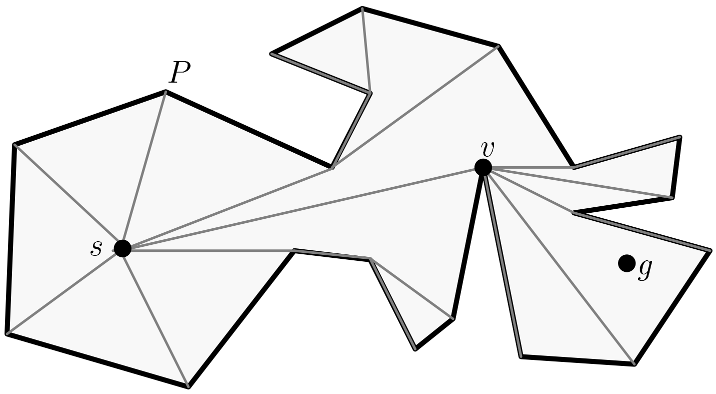

Figure 3: There is a vertex on T_s from which g is visible.

This implies that we only need an algorithm to explore the vertices of P. This can be easily achieved by applying Algorithm *Tree Search* to T_s. That T_s is not known in the beginning is not a problem as the following lemma shows.

Lemma 4.1 *If $\mathcal{P}$ is a path in the simple polygon P, then the visibility polygon $vis(\mathcal{P})$ of $\mathcal{P}$ is geodesically convex, that is, the shortest path between any two points in $vis(\mathcal{P})$ is contained in $vis(\mathcal{P})$.*

Proof: The proof is by contradiction. So consider two points p and q that are contained in the visibility polygon $vis(\mathcal{P})$ of $\mathcal{P}$ and assume that the shortest path from p to q is not contained in $vis(\mathcal{P})$. Then, there is a window w of $vis(\mathcal{P})$ that is crossed by $shp(p,q)$.

It is easy to see that $vis(\mathcal{P})$ is a simply connected set. Since both p and q are contained in $vis(\mathcal{P})$, $shp(p,q)$ crosses w twice, once to enter the pocket of w and once to leave it again. But then the part of $shp(p,q)$ inside the pocket of w can be replaced by the part of w between the cross points. This yields a shorter path—a contradiction. □

Note that Lemma 4.1 is not true anymore if we allow the polygon to contain holes. But if the polygon is simple, then the robot can compute the distance between two points that it sees by Lemma 4.1.

Therefore, the robot can construct T_s on-the-fly as it executes Algorithm *Tree Search*. This immediately leads to a search algorithm with a competitive ratio of at most $1 + 2n^n/(n-1)^{n-1} \leq 1 + 2en$ for searching for a target in a simple polygon, which considerably improves the $(8n-3)$-competitive algorithm of Icking, which also uses a search algorithm for geometric trees to explore the shortest path tree [Ick94, Kle97].

However, the analysis of the above algorithm can be further improved by the following observation (see also [Kle97]).

Lemma 4.2 *If s and g are two points in P, then the last node of T_s that is intersected by $shp(s,g)$ and that is different from g is an internal node of T_s.*

Proof: If s sees g, then the claim is trivially true. So assume that s does not see g. Let v' be the vertex of $shp(s,g)$ before v. Since v' does not see g and v is contained in the shortest path from v' to g, it is easy to see that v is a reflex vertex. Let e be the edge of P incident to v that is not seen by v'. Clearly, the shortest path from v' to the other end point of e contains v as well. Hence, v is not a leaf of T_s. □

Hence, in order to find the vertex v it is not necessary to consider the leaves of T_s. If we prune the m leaves of the tree, then the remaining tree T_s^* can have no more than m leaves as the pruning does not introduce new root-to-leaf paths.

Therefore, the number of leaves m^* of T_s^* is at most $n/2$. If we apply Algorithm *Tree Search* to T_s^*, then we obtain a strategy to search in polygons with a competitive ratio of $1 + en$. Note that since the robot can compute the shortest path between two points, it is easy to decide if a vertex is a leaf of T_s or not. Moreover, if we denote the number of reflex vertices of P by r, then the competitive ratio of our algorithm is $1 + 2r^r/(r-1)^{r-1} \leq 1 + 2er$ since the vertices of T_s^* obviously consist only of reflex vertices.

Thus, we have proven the following theorem.

Theorem 4.3 *If P is a polygon with r reflex vertices, then there is an algorithm to search in P with a competitive ratio of $1 + 2er$.*

Simulating Dijkstra's algorithm on T_s^* yields an even better algorithm with a competitive ratio of $1+2(r-1)$. Here, the robot repeatedly chooses the closest unvisited vertex v, follows $shp(s,v)$, and returns if g is not visible from v.[1] Note again that even though P is not known to the robot, the closest unvisited vertex is visible to the robot. The competitive ratio is now just

$$\begin{aligned}\frac{2\sum_{v' \text{ visited}} d(s,v') + d(s,v)}{d(s,v)} &\leq 1 + 2|\{v' \mid v' \text{ is visited}\}| \\ &\leq 1 + 2(|T_s^*| - 1) \;\leq\; 1 + 2(r-1)\end{aligned}$$

since $d(s,v) \geq d(s,v')$, for all vertices v' that are visited during the search. However, the competitive ratio of the simulation of Dijkstra's algorithm depends on the *total number of vertices* of T_s^* whereas the competitive ratio of the algorithm based on Algorithm *Tree Search* only depends on the *number of leaves* m^* of T_s^*.

In particular, if $|T_s^*| \geq m^*e + 1$, then Algorithm *Tree Search* yields a better competitive ratio. A simple example are spiral polygons where Algorithm *Tree Search* achieves a competitive ratio of 9 in the worst case whereas the simulation of Dijkstra's algorithm on T_s^* can have a competitive ratio of up to $2n-7$. This is also an upper bound for the competitive ratio of the simulation of Dijkstra's algorithm since there are at least three convex vertices in a simple polygon.

4.1 k-spiral Polygons

We show in the following that in general the application of Algorithm *Tree Search* to T_s^* has a competitive ratio of $1 + 2(2k)^{2k}/(2k-1)^{2k-1} \leq 1 + 4ek$ in polygons whose boundary can be decomposed into k convex and k reflex chains. As we already observed above, if v is a convex vertex of a polygon, then v is a leaf of T_s and, therefore, does not belong to T_s^*. Hence, we only need to consider reflex vertices.

Lemma 4.4 *Let v be a reflex vertex of P and v_1 and v_2 the vertices that are adjacent to v in P. If v is an internal node of T_s, then either $shp(s,v_1)$ or $shp(s,v_2)$ contains v.*

Proof: Let v be a reflex vertex that is an internal node of T_s. We show that for one of the vertices v_i, $i = 1,2$, the shortest path from s to v_i contains v by contradiction. Let e_i be the edge from v to v_i, for $i = 1,2$, and v^* be the lowest common ancestor of v_1 and v_2 in T_s. We assume that neither $shp(v^*, v_1)$ nor $shp(v^*, v_2)$ contains v. The concatenation of $shp(v^*, v_1)$ with e_1, e_2, and $shp(v_2, v^*)$ forms a simple closed polygonal curve $\mathcal{C}$. Since v is an internal node of T_s, there is one vertex v' of P such that $shp(s,v')$ contains v. Hence, $shp(s,v')$ intersects $\mathcal{C}$ once in a point p_1 to enter $\mathcal{C}$ and to reach v and once in a point p_2 to leave $\mathcal{C}$ and to reach the vertex v' (see Figure 4a). Assume that p_2 belongs to $shp(s,v_2)$. Since subpaths of shortest paths are shortest paths themselves, there are two distinct shortest paths from s to p_2, one path that is contained in $shp(s,v')$ and that intersects v and one path that is contained in $shp(s,v_2)$ and that does not intersect v—a contradiction since shortest paths are unique in a simple polygon. □

With Lemma 4.4 we can prove the following result.

[1]The same algorithm was independently suggested by I. Semrau [Sem97].

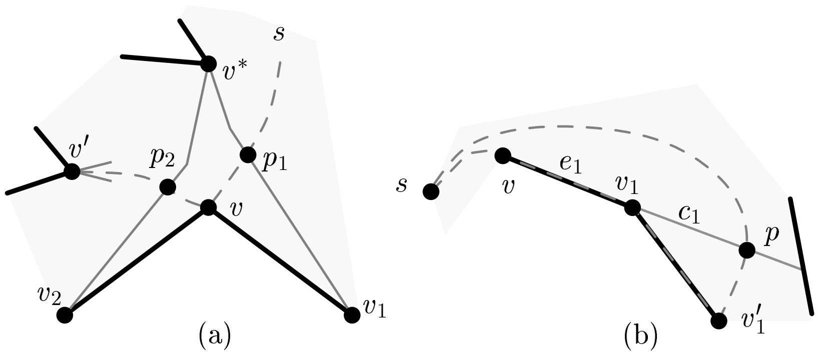

Figure 4: (a) If v is not adjacent to v_1 and v_2, then v is a leaf of T_s. (b) The point p is on the extension of the last link of $shp(s, v_1)$.

Lemma 4.5 *Let v be a vertex of P and v_1 and v_2 the vertices that are adjacent to v in P. If v, v_1, and v_2 are reflex vertices, then either v is a leaf of T_s or v is not a leaf T_s^*.*

Proof: Let v be a reflex vertex of P such that the adjacent vertices v_1 and v_2 of v are also reflex. Assume that v is not a leaf of T_s. We show that v is also not a leaf of T_s^*.

By Lemma 4.4 there is one vertex of v_1 and v_2, say v_1, such that $shp(s, v_1)$ contains v. We show that v_1 is not a leaf of T_s. This implies that v_1 also belongs to T_s^* and is a child of v, which proves the claim.

Let e_1 be the edge of P that connects v and v_1 and v_1' the vertex that is adjacent to v_1 and different from v. Let c_1 be the chord that is collinear with e_1 and incident to v_1 (see Figure 4b). Since v_1 is a reflex vertex, c_1 exists and v is contained in the region of c_1 that contains s whereas v_1' is contained in the other region of c_1. Hence, the shortest path from v_1' to s crosses c_1 in a point p. Since p is on the extension of the last edge $\overline{vv_1}$ of $shp(s, v_1)$, $shp(s, p)$ contains v_1 and v. Therefore, p equals v_1 and v_1 is not a leaf of T_s. □

Lemma 4.5 implies that if v is a leaf of T_s^*, then v is a reflex vertex such that at least one of its adjacent vertices is a convex vertex. Hence, v is the end vertex of a reflex chain. If the boundary of P can be decomposed into k convex and k reflex chains, then there are $2k$ end vertices of reflex chains. Therefore, if we apply Algorithm *Tree Search* to T_s^*, then the competitive ratio is at most $1 + 2 \cdot (2k)^{2k}/(2k-1)^{2k-1}$ as claimed. The polygon shown in Figure 5 can be easily generalized to a polygon P_k whose boundary consists of k convex and k reflex chains such that the competitive ratio of any search strategy for P_k is at least $1+2\cdot(2k)^{2k}/(2k-1)^{2k-1}$. In this sense our algorithm to search in simple polygons is optimal.

We have shown the following theorem.

Theorem 4.6 *There is an algorithm to search in simple polygons that has a competitive ratio of $1 + 2 \cdot (2k)^{2k}/(2k-1)^{2k-1}$ on the class of polygons whose boundary can be decomposed into k convex and k reflex chains.*

5 Conclusions

We present the Algorithm *Tree Search* to search for a target in an unknown geometric tree T. Algorithm *Tree Search* simulates the optimal search strategy to search in m concurrent

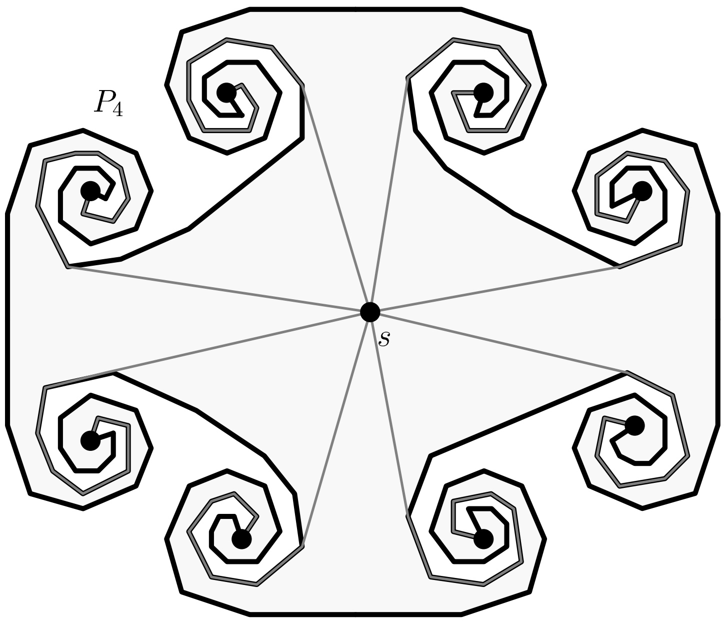

Figure 5: The boundary of the polygon P_4 can be decomposed into four convex and four reflex chains. A search strategy in P_4 has a competitive ratio of at least $1 + 2 \cdot (8^8/7^7) \sim 41.7$.

rays and achieves an optimal competitive ratio of $1 + 2m^m/(m-1)^{m-1}$ if m is the number of leaves of T.

In the second part we show how to apply the algorithm to the problem of searching for a target in unknown simple polygons. We present two algorithms, one based on Algorithm *Tree Search* and the other based on a simulation of Dijkstra's algorithm. The first algorithm achieves a competitive ratio of $1 + en$ whereas the second achieves a competitive ratio of $2n - 7$. However, if the polygon boundary can be decomposed into k convex and reflex chains, then the search algorithm based on Algorithm *Tree Search* achieves a competitive ratio of $1 + 4ek$ whereas the competitive ratio of the algorithm based on the simulation of Dijkstra's algorithm may be arbitrarily high.

The problem of searching in polygons is far from solved. There is a lower bound of $n/2$ for the competitive ratio which still leaves a significant gap compared to algorithms presented above. Also most environments in practice cannot be modeled by simple polygons. Hence, future research should also address the problem of searching in polygons with holes.

References

[BYCR93] R. Baeza-Yates, J. Culberson, and G. Rawlins. Searching in the plane. *Information and Computation*, 106:234–252, 1993.

[DHS95] A. Datta, Ch. Hipke, and S. Schuierer. Competitive searching in polygons—beyond generalized streets. In *Proc. Sixth Annual International Symposium on Algorithms and Computation*, pages 32–41. LNCS 1004, 1995.

[DI94] A. Datta and Ch. Icking. Competitive searching in a generalized street. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 175–182, 1994.

[Gal80] S. Gal. *Search Games.* Academic Press, 1980.

[Ick94] Ch. Icking. *Motion and Visibility in Simple Polygons.* PhD thesis, Fernuniversität Hagen, 1994.

[Kle92] R. Klein. Walking an unknown street with bounded detour. *Comput. Geom. Theory Appl.*, 1:325–351, 1992.

[Kle94] J. M. Kleinberg. On-line search in a simple polygon. In *Proc. of 5th ACM-SIAM Symp. on Discrete Algorithms*, pages 8–15, 1994.

[Kle97] R. Klein. *Algorithmische Geometrie.* Addison-Wesley, 1997.

[LOS96] A. López-Ortiz and S. Schuierer. Generalized streets revisited. In J. Diaz and M. Serna, editors, *Proc. 4th European Symposium on Algorithms*, pages 546–558. LNCS 1136, 1996.

[LOS97] A. López-Ortiz and S. Schuierer. Position-independent near optimal searching and on-line recognition in star polygons. In F. Dehne, A. Rau-Chaplin, J.-R. Sack, and R. Tamassia, editors, *Proc. 5th Workshop on Algorithms and Data Structures*, pages 284–296. LNCS 1272, 1997.

[PY91] Papadimitriou and Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991.

[Sem97] I. Semrau. Personal communications, 1997.

[ST85] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.