RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT BONN
INSTITUT FÜR INFORMATIK I

Elmar Langetepe

Online Motion Planning

MA INF 1314

Summersemester 2017

Contents

List of Figures

List of Algorithms

Introduction

This lecture considers tasks for autonomous agents. In general, constructing autonomous machines is a very complex challenge and has many different engineering and scientific aspects, some of which are given in the following list.

- Elektronic devices
- Mechanical devices
- Control/Process engineering
- Artificial Intelligence
- Softwareengineering
- ⋮
- Plans: Algorithmic/Motion planning
- Full information (offline)/ Incomplete information (online)
- Input: Geometry of the Environment

As part of the algorithm track of the master program we will concentrate on the item *Algorithms*. That is, we concentrate on the description and analysis of efficient schedules for solving motion planning tasks for autonomous agents. Besides, we concentrate on problem definitions and models that take the geometry of the scene into account. In this sense the scientific aspects of this course are part of the scientific area called Computational Geometry. Furthermore we consider online problems, which means that the full information of the problem is not given in advance. The agent has to *move* around and collects more information.

We will mainly concentrate on the ground tasks of autonomous agents in unknown environments such as

- Searching for a goal,
- Exploration of an environment,
- Escaping from a labyrinth,

and we consider different abilities of the agents some of which are

- Continuous/discrete vision,
- Touch sensor/compass,
- Building a map/constant memory.

The first concern is that we construct correct algorithms which always fulfil the task. Second we concentrate on the efficiency of the corresponding strategy. We would like to analyse performance guarantees and would like to provide for formal proofs. The course is related to the undergraduate course on *Offline motion planning*. In the offline case the information for the task is fully given and we only have to compute the best path for the agent. The offline solution will be used as a comparison measure for the online case. This is a well known concept for online problems in general.

Chapter 1

Labyrinths, grids and graphs

In this section we first concentrate on discrete environments based on grid structures. For the grid structure we consider an agent that can move from one cell to a neighbouring cell with unit cost. We start with the task of searching for a goal in a very special grid environment. After that we ask for visiting all cells, which means that we would like to explore the environment. For this task the grid environment is only partially known, by a touch sensor the agent can only detect the neighbouring cells. The agent can build a map. Exploration and Searching are closely related. If we are searching for an unknown goal, it is clear that in the worst-case the whole environment has to be explored. The main difference is the performance of these *online* tasks. As a comparison measure we compare the length of the agent's path to the length of the optimal path under full information. Thus, in the case of searching for a goal, the comparison measure is the shortest path to the goal.

At the end of the section we turn over to the exploration task in general graphs under different additional conditions.

1.1 Shannons Mouse Algorithm

Historically the first online motion planning algorithm for an autonomous agent was designed by Claude Shannon [Sha52, Sha93] in 1950. He considered a 5×5 cellular labyrinth, the inner walls of the labyrinth could be placed around arbitrary cells. In principle, he constructed a labyrinth based on a grid environment; see Figure 1.1.

The task of his electronical mouse was to find a target, i.e. the cheese, located on one of the fields of the grid. The target and the start of the mouse were located in the same *connected component* of the *grid labyrinth*. The electronical mouse was able to move from one cell to a neighbouring cell. Additionally, it could (electronically) mark any cell by a label N, E, S, W which indicates in which direction the mouse left the cell at the last visit. This label is updated after leaving the cell. With theses abilities the following algorithm was designed.

Algorithm 1.1 Shannons Maus

- Initialize any cell by the label N for 'North'.
- While the goal has not been found:
  starting from the label direction, search for the first cell in clockwise order that can be visited. Change the label to the corresponding direction and move to this neighbouring cell.

Sutherland [Sut69] has shown that:

Theorem 1.1 *Shannon's Algorithms (Algorithm 1.1) is correct. For any labyrinth, any starting and any goal the agent will find the goal, if a path from the start to the goal exists.*

Figure 1.1: Shannons original mouse labyrinth.

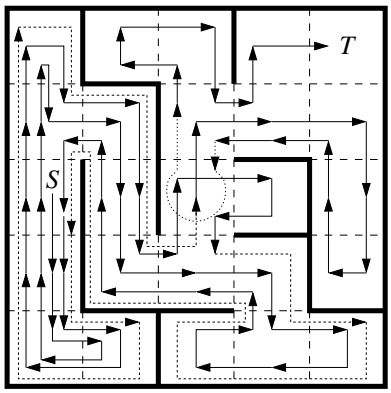


Figure 1.2: An example of the execution of Shannons Algorithm.

Proof. We omit the goal and show that any cell in the connected component of the start will be visited infinitely often. □

Exercise 1 *Formalize the above proof sketch!*

As shown in Figure 1.2 the path of Shannons Mouse is not very efficient.

1.2 Intuitive connection of labyrinths, grids and graphs

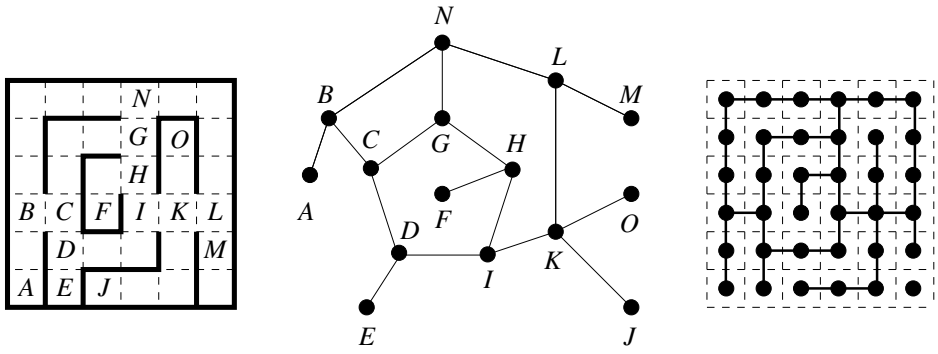
For a human a labyrinth consists of corridors and connection points. In this sense the environment for Shannons task can be considered to be a labyrinth. Obviously any such labyrinth can be modeled by a planar graph.[1] More precisely the environment for Shannons task is a grid graph. Figure 1.3 shows the corresponding intuitive interpretations.

For any intuitive labyrinth there is a labyrinth-graph. On the other hand for any planar graph we can build some sort of labyrinth. This is not true for general graphs. For example the complete graph K_5 has no planar representation and therefore a correspondance to a labyrinth does not exist.

1.3 A lower bound for online graph exploration

We consider the following model. Assume that a graph $G = (V,E)$ is given. If the agent is located on a vertex it detects all neighbouring vertices. Let us assume that moving along an edge can be done with

[1] A graph, that has an intersection free representation in the plane.



(i) Labyrinth (ii) Labyrinthgraph (iii) Gittergraph

Figure 1.3: Labyrinth, labyrinth-graph and gridgraph.

unit cost. The task is to visit all edges and vertices and return to the start. The agent has the ability of building a map. If we apply a DFS (depth first search) for the edges we will move along any edges twice. DFS can run online. The best offline strategy has to visit any edge at least once. In this sense DFS is a 2-approximation.

The comparison and approximation between online and offline is represented by the following concept. A strategy that runs under incomplete information is denoted as an **Online–Strategy**. On the other hand an **Offline–Strategy** solves the same task with full information. In the above example the offline strategy is the shortest round trip that visits all edges of the graph.

The performance measure for Online-Algorithms is the so-called *competitive ratio*.

Definition 1.2 (Sleator, Tarjan, 1995)
Let Π be a problem class and S be a strategy, that solves any instance $P \in \Pi$.
Let $K_S(P)$ be the cost of S for solving P.
Let $K_{opt}(P)$ be the cost of the optimal solution for P.

The strategy S is denoted to be c–competitive, if there are fixed constants $c, \alpha > 0$, so that for all $P \in \Pi$

$$K_S(P) \leq c \cdot K_{opt}(P) + \alpha$$

holds.

The additive constant α is often used for starting situations. For example if we are searching for a goal and have only two unknown options, the goal might be very close to the start, the unsuccessful step will lead to an arbitrarily large competitive ratio. This is not intended. Sometimes we can omit the additive constant, if we have additional assumptions. For example we can assume that the goal is at least distance 1 away from the start.

As already mentioned DFS on the edges visits any edge at most twice. There are graphs where the optimal offline solution also has to visit any edge twice. For such examples DFS is optimal with ratio 1. Now we are searching for a lower bound for the competitive ratio. That is, we would like to construct example such that any possible online strategy fails within a ratio of 2.

Theorem 1.3 *(Icking, Kamphans, Klein, Langetepe, 2000)*
For the online-exploration of a graph $G = (V, E)$ for visiting all edges and vertices of G there is always an arbitrarily large example such that any online strategy visits roughly twice as much edges in comparison to the optimal offline strategy. DFS always visit no more than twice as much edges against the optimum. *[IKKL00a]*

Proof. The second part is clear because DFS visits exactly any edge twice. Any optimal strategy has to visit at least the edges.

The robot should explore a gridgraph and starts in a vertex s. Finally, the agent has to return to s. We construct an *open* corridor and offer two directions for the agent. At some moment in time the agent has explored ℓ new vertices in the corridor. If this happens we let construct a conjunction at one end s' of the corridor. At this bifurcation two open corridors are build up which *run* back into the direction of s. If the agent proceeds one of the following events will happen.

1. The agent goes back to s.
2. The agent has visited more than $\ell+1$ edges in one of the new corridors.

Let ℓ_1 denote the length of the part of the starting open corridor into the opposite direction of s'. Let ℓ_2 and ℓ_3 denote the length of the second and third open corridor.

We analyse the edge visits $|S_{ROB}|$ that an arbitrary strategy S_{ROB} has done so far.

1. $|S_{ROB}| \geq 2\ell_1 + (\ell - \ell_1) + 2\ell_2 + 2\ell_3 + (\ell - \ell_1) = 2(\ell + \ell_2 + \ell_3)$, see Figure 1.4. Now we close the corridors at the open ends. From now on the agent still requires $|S_{OPT}| = 2(\ell + \ell_2 + \ell_3) + 6$ edge visits, where S_{OPT} is the optimal strategy if the situation was known from the beginning. Thus we have: $|S_{ROB}| \geq 2|S_{OPT}| - 6$.

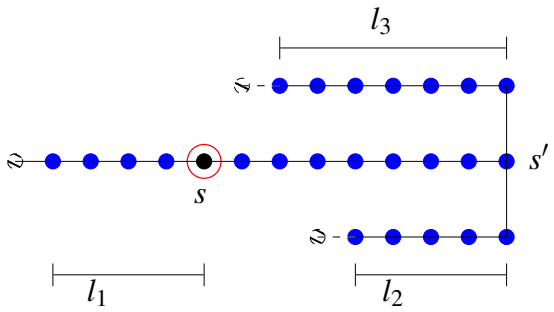


Figure 1.4: The agent return to s.

2. W.l.o.g. the agent has explored $\ell+1$-ten vertices in corridor 3. We have $|S_{ROB}| \geq 2\ell_1 + (\ell - \ell_1) + 2\ell_2 + (\ell+1)$. We connect corridor 3 with corridor 1(see Figure 1.5) and close corridor 2. The agent still requires $\ell + 1 + 2(\ell_2 + 1) + (\ell - \ell_1)$ edge visits; in total at least $4\ell + 4\ell_2 + 4 = 4(\ell + \ell_2) + 4$ edge visits. From $|S_{OPT}| = 2(\ell+1) + 2(\ell_2+1) = 2(\ell+\ell_2) + 4$ we conclude $|S_{ROB}| \geq 2|S_{OPT}| - 4 > 2|S_{OPT}| - 6$.

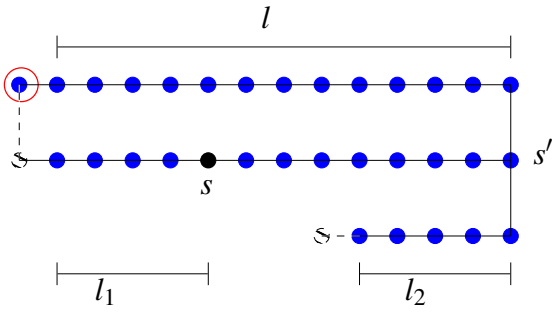


Figure 1.5: The agent has visited $\ell+1$ vertices in corridor 3.

We have $|S_{ROB}|/|S_{OPT}| \geq 2 - 6/|S_{OPT}|$. We also have $|S_{OPT}| \geq 2(\ell+1)$ and conclude $2 - 6/|S_{OPT}| > 2 - 6/2\ell = 2 - 3/\ell$. For arbitrary $\delta > 0$ we choose $\ell = \lceil 3/\delta \rceil$ and conclude $|S_{ROB}|/|S_{OPT}| > 2 - \delta$. □

Remark 1.4 *There are always examples so that the optimal exploration tour visits any edge twice.*

Corollary 1.5 *DFS for the Online-Edge-Exploration of general graphs is 2–competitive and optimal.*

Exercise 2 *Show that the same competitive ratio holds, if the return to the starting point is not required.*

Exercise 3 *Consider the problem of exploring the vertices (not the edges) of a graph. If the agent is located at a vertex it detects the outgoing edges but along non-visited edges it is not clear which vertex lies on the opposite side. Does DFS applied on the vertices result in a 2-approximation?*

1.4 Exploration of grid environments

Next we consider a simple discrete grid model. The agent runs inside a grid-environment. In contrast to Shannons the inner obstacles consist of full cells instead of single blocked edges.

We would like to design efficient strategies for such grid environments. First, we give a formal definition.

Definition 1.6

- A **cell** c is a tupel $(x,y) \in \mathbb{N}^2$.
- Two cells $c_1 = (x_1,y_1), c_2 = (x_2,y_2)$ are **adjacent**, if $:\Leftrightarrow |x_1 - x_2| + |y_1 - y_2| = 1$. For a single cell c, exact 4 cells are adjacent.
- Two cells $c_1 = (x_1,y_1), c_2 = (x_2,y_2)$, $c_1 \neq c_2$ are **diagonally adjacent**, if $:\Leftrightarrow |x_1 - x_2| \leq 1 \wedge |y_1 - y_2| \leq 1$. For a single cell c, exact 8 cells are diagonally adjacent.
- A **path** $\pi(s,t)$ from cell s to cell t is a sequence of cells $s = c_1, \ldots, c_n = t$ such that c_i and c_{i+1} are adjacent for $i = 1, \ldots, n-1$.
- A **gridpolygon** P is a set of path-connected cells, i.e., $\forall c_i, c_j \in P : \exists$ path $\pi(c_i, c_j)$, such that $\pi(c_i, c_j) \in P$ verl"auft.

The agent is equipped with a touch sensor so that the agent scans the adjacent cells and their nature (free cell or boundary cell) from its current position. Additionally, the agent has the capability of building a map. The task is to visit all cells of the gridpolygon and return to the start. This problem is NP-hart for known environments; see [IPS82]. We are looking for an efficient Online-Strategy. The agent can move within one step to an adjacent cell. For simplicity we count the number of movements.

The task is related to vacuum-cleaning or lawn-mowing. A cell represents the size of the tool, the tool should visit all cells of the environment. A general polygonal environment P can be approximated by a grid-polygon.

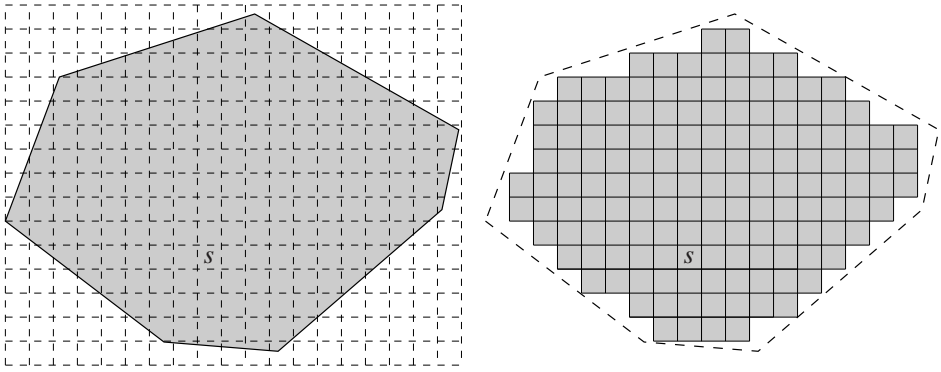


Figure 1.6: A polygon P and the gridpolygon $P_\square$ as a reasonable approximation.

The starting position and orientation of the tool fixes the grid and all connected cells which are entirely inside P belong to the approximation $P_\square$; see Figure 1.6. For any gridpolygon P' we use the following notation. Cells that do not belong to P' but are diagonally adjacent to a cell in P' are called boundary cells. The common edges of the boundary cells and cells of P' are the boundary edges. Let $E(P')$ denote the number of boundary cells or E for short, if the context is clear. The number of cells is denoted by $C(P')$ or C respectively.

From Theorem 1.3 we can already conlcude a lower bound of 2 for the competitive ratio of this problem. On the other hand DFS on the cells finishes the task in $2C-2$ steps

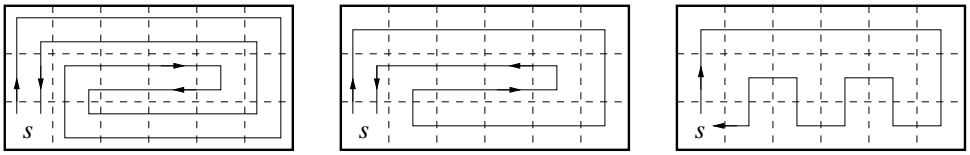


Figure 1.7: Ist DFS optimal?

Exercise 4 *Give a formal proof that for a gridpolygon P the DFS strategy on the cells requires exactly* $2C-2$ *steps for the exploration (with return to the start) of P.*

But is DFS really the best strategy in general? For fleshy environments DFS obviously is not very efficient. Besides the lower bound construction makes use of corridors only. Compare Figure 1.7: After DFS has visited the *right* neighbour of *s* the environment is fully known and we can improve the strategy. It seems that even the optimal solution could be found in an online fashion in this example. On the other hand there are always *skinny* corridor-like environments where DFS is the best online strategy. Altogether, we require a case sensitive measure for the performance of an online strategy that relies on the existence of large areas. The existence of large *fleshy* areas depends on the relationship between the number of cells C and the number of (boundary) edges E. In Figure 1.7 the environment has 18 edges and 18 cells. In corridor-like environments we have $\frac{1}{2}E \approx C$ in fleshy environments we have $\frac{1}{2}E \ll C$; see also Figure 1.8.

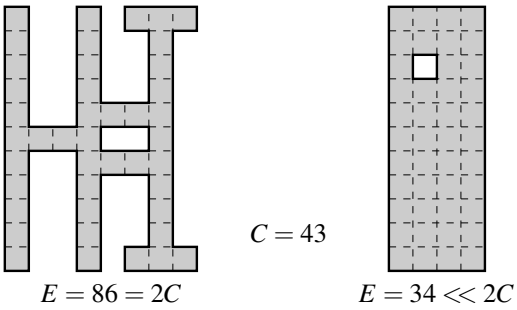


Figure 1.8: The number of boundary edges E in comparison to the number of cells C is a measure for the existence of *fleshy* or *skinny* parts.

1.4.1 Exploration of simple gridpolygons

We first consider *simple* gridpolygons P which do not have any *inner* boundary cell, i.e., also the set of all cells that do not belong to P are path connected.

Note that the lower bound of 2 is not given, because the lower bound construction in the previous section requires the existence of inner obstacles. We make use of a different construction.

Theorem 1.7 *Any online strategy for the exploration (with return to the start) of a simple gridpolygon P of C cells, requires at least* $\frac{7}{6}C$ *steps for fulfilling the task.*

Proof. We let the agent start in a corner as depicted in Figure 1.9(i) and successively extend the walls. Assume that the agent decides to move to the east first. By symmetry we apply the same arguments, if the agent moves to the south. For the second step the agent has two possibilities (moving backwards can be ignored). Either the strategy leaves the wall by a step to the south (seeFigure 1.9(ii)) or the strategy follows the wall to the east (see Figure 1.9(iii)).

In the first case we close the polygon as shown in Figure 1.9(iv). For this small example the agent requires 8 steps whereas the optimal solution requires only 6 steps which gives a ratio of $\frac{8}{6} \approx 1.3$.

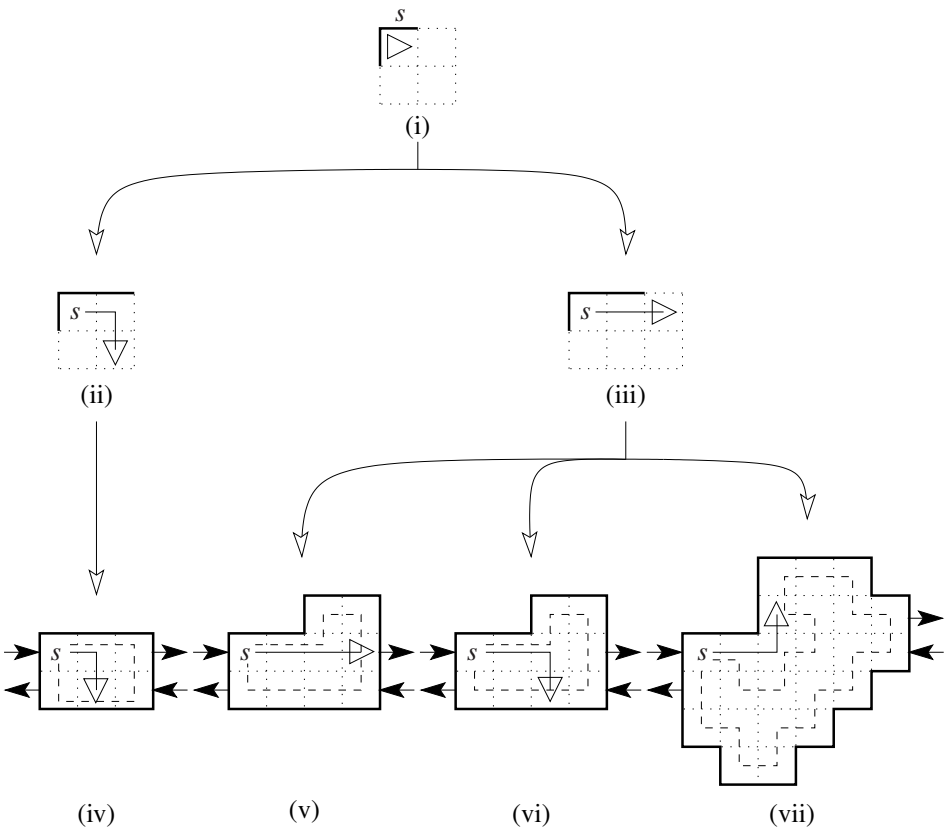


Figure 1.9: A lower bound construction for the exploration of simple gridpolygons.

In the second case we proceed as follows: If the robot leaves the wall (the wall runs upwards), we close the polygon as depicted in Figure 1.9(v) or (vi), respectively. In this small example the agent requires 12, respectively, whereas 10 steps are sufficient.

In the last and most interesting case the agent follows the wall upwards and we present the sophisticated polygon of Figure 1.9(vii). In the offline case an agent requires 24 steps. The online agent already made a mistake and can only finish the task within 24 steps. This can be shown by a tedious case distinction of all further movements. We made use of an implementation that simply checks all possibilities for the next 24 steps. There was no such path that finishes the task. For all cases we guarantee have a worst-case ratio of $\frac{28}{24} = \frac{7}{6} \approx 1.16$.

We use this scheme in order to present a lower bound construction of arbitrary size. Any block has an entrance and exit cell which are marked by corresponding arrows; see Figure 1.9(iv)–(vii). If an agent moves inside the next block, the game starts again. Since the arrows only point in east or west direction we take care that the concatenated construction results in a simple gridpolygon of arbitrary size. as required. □

Note that the arbitrary-size condition in the above proof is necessary. Assume that we can only construct such examples of fixed size D. This will not result in a lower bound on the competitive ratio. Any reasonable algorithm will explore the fixed envirnment with komeptitive ratio 1 since $\alpha \gg D$ exists, with $|S_{\mathrm{ALG}}| \leq |S_{\mathrm{OPT}}| + \alpha$.

We consider the exploration of a simple gridpolygon by DFS and formalize the strategy; see Algorithm 1.2. The agent explores the polygon by the "Left-Hand-Rule", i.e. the DFS preference is Left before Straight-On before Right. The current direction (North, West, East or South) is stored in the variable *dir*. The functions cw(*dir*), ccw(*dir*) und reverse(*dir*) result in the corresponding directions of a rotation by 90° in clockwise or counter-clockwise order or by a rotation of 180°, respectively. The predicate unexplored(*dir*) is true, if the adjacent cell in direction *dir* is a cell of the environment, which was not visited yet.

Algorithm 1.2 DFS

DFS:

Choose *dir*, such that reverse(*dir*) is a boundary cell;
ExploreCell(*dir*);

ExploreCell(*dir*):

// Left-Hand-Rule:
ExploreStep(ccw(*dir*));
ExploreStep(*dir*);
ExploreStep(cw(*dir*));

ExploreStep(*dir*):

if unexplored(*dir*) **then**
  move(*dir*);
  ExploreCell(*dir*);
  move(reverse(*dir*));
end if

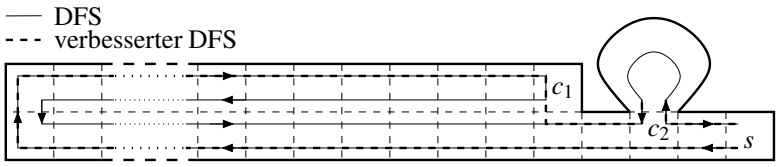


Figure 1.10: First simple improvement of DFS.

A first simple improvement for DFS is as follows:

> If there are no unexplored adjacent cells around the current cell, move back along the shortest path (use all already explored cells) to the *last* cell, that still has an unexplored neighbouring cell.

Figure 1.10 sketches this idea: After visiting c_1 the pure DFS will backtrack along the full corridor of width 2 and reach cell c_2 where still something has to be explored. With our improvement we move directly from c_1 to c_2. Note that for the shortest path we can only make use of the already visited cells. We have no further information about the environment.

By this argument we no longer use the step "move(reverse(*dir*))" in the procedure ExploreStep. After the execution of ExploreCell we can no longer conclude that the agent is on the same cell as before. Therefore we store the current position of the agent and use it as a parameter for any call of ExploreStep. The function unexplored(*base*, *dir*) gives "True", if w.r.t. cell *base* there is an unexplored adjacent cell in direction *dir*. We re-formalize the behaviour as follows:

Algorithm 1.3 DFS with optimal return trips

DFS:

Choose *dir*, such that reverse(*dir*) is a boundary cell;
ExploreCell(*dir*);
Move along the shortest path to the start;

ExploreCell(*dir*):

base := current position;
// Left-Hand-Rule:
ExploreStep(*base*, ccw(*dir*));
ExploreStep(*base*, *dir*);
ExploreStep(*base*, cw(*dir*));

ExploreStep(*base*, *dir*):

if unexplored(*base*, *dir*) **then**
  Move along the shortest path
    among all visited cells to *base*;
  move(*dir*);
  ExploreCell(*dir*);
end if

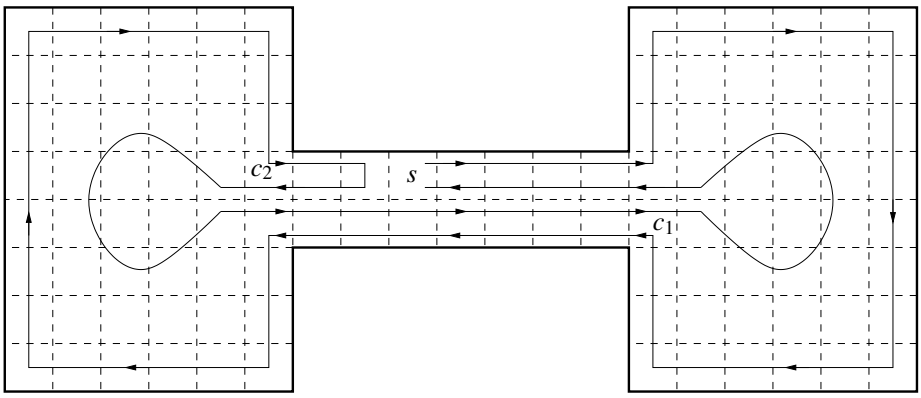


Figure 1.11: Second improvement of DFS.

Algorithm 1.4 SmartDFS

```
SmartDFS:
    Choose direction dir, such that reverse(dir) is a boundary cell;
    ExploreCell(dir);
    Move along the shortest path to the start;

ExploreCell(dir):
    Mark the cell with its layernumber;
    base := current Position;
    if not SplitCell(base) then
      // Left-Hand-Rule:
      ExploreStep(base, ccw(dir));
      ExploreStep(base, dir);
      ExploreStep(base, cw(dir));
    else
      // Choose different order:
      Calculate the type of the components by the layernumbers
        of the surrounding cells;
      if No component of typ (III) exists then
        Move one step by the Right-Hand-Rule;
      else
        Visit the component of type (III) last.
      end if
    end if

ExploreStep(base, dir):
    if unexplored(base, dir) then
      Move along the shortest path along
        the visited cells to base;
      move(dir);
      ExploreCell(dir);
    end if
```

For a second kind of improvement we consider the gridpolygon Figure 1.11. In this example the current DFS variant fully surrounds the polygon. Finally the agent has to move back from c_2 to c_1 so that the corridor of width 2 is visited almost 4 times. Obviously it would be better to first fully explore the component at c_1 move to the other component at c_2 and finally move back to the start. In this case the critical corridor will be visited only once. So, if the exploration splits the polygon into components that have to be considered, we have to take care which component should be visited first.

A cell (like the cell c_1) where the remaining polygon definitely splits into different parts is called a **split-cell**. At the first visit of split-cell c_1 in Figure 1.11 it seems to be better to not apply the Left-Hand preference. This depends on the location of the starting point, because we have to move back at the end. The idea can be formulated as follows.

> If the unexplored part of the polygon definitely is splitted into different components (i.e., the graph of unexplored cells is splitted into different components), try to visit the unexplored part that does not contain the starting point.

This idea leads to the Algorithm 1.4 (SmartDFS). It remains to decide, which component actually *contains* the starting point. For this we introduce some notions. Until the first split happens we apply the Left-Hand-Rule and successively explore the polygon layer by layer from the outer boundary to the inner parts. We require a formal definition of the layers.

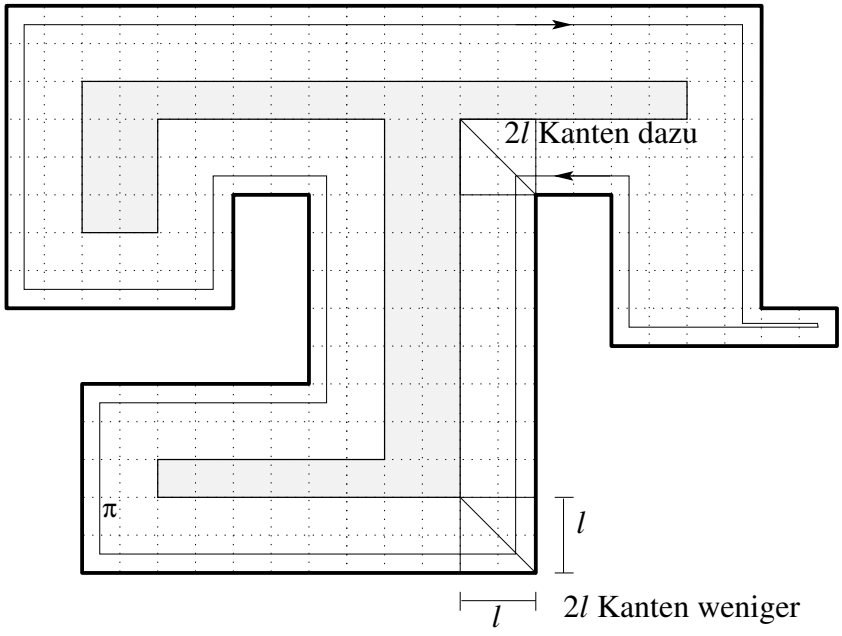


Figure 1.12: The ℓ-Offset of gridpolygon P.

Definition 1.8 Let P be a (simple) gridpolygon. The cells of P that share a boundary edge belong to the first layer, the **1-Layer** of P. The gridpolygon that stems from P without the 1-Layer is called the **1-Offset** of P. Recursively, the **2-Layer** of P, is the **1-Layer** of the **1-Offset** of P and the **2-Offset** of P is the **1-Offset** of the **1-Offset** of P and so on.

Note that the ℓ-Offset of a gridpolygon need not be connected and finally the Offsets will decrease to an empty polygon. The definition is totally independent from any strategy. Fortunately, during the execution of SmartDFS on a simple gridpolygon, we can successively mark and store the layers for any visited cell. The ℓ-Offset has an interesting property.

Lemma 1.9 *The non-empty ℓ-Offset of a simple gridpolygon P has at least 8ℓ edges less than P.*

Proof. We surround the boundary of the gridpolygon in clockwise order and visit all boundary edges along this path. Let us assume that the offset remains a single component. For a left turn the ℓ-Offset 2ℓ looses 2ℓ edges for a right turn the ℓ-Offset 2ℓ wins 2ℓ edges We can show that there are 4 more right turns than left turns. So the ℓ-Offset has at least 8ℓ edges less than P. Even more edges will be cancelled, if the polygon fell into pieces. □

Exercise 5 *Show that for any surrounding of the boundary of a simple gridpolygon in clockwise order there are 4 more rigth turns than left turns. Make use of induction.*

Exercise 6 *Show that in the above proof the non-empty ℓ-Offset will loose even more edges, if it consists of more than one connected component. Show the statement for the* 1*-Offset.*

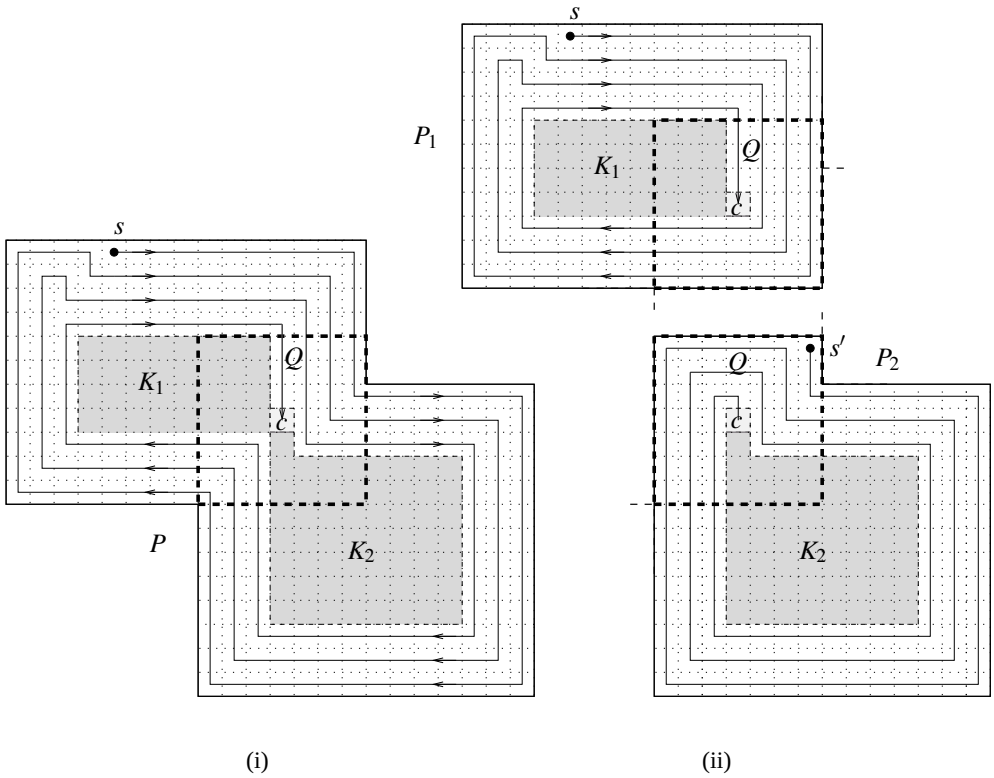


Figure 1.13: Decomposition at a split-cell.

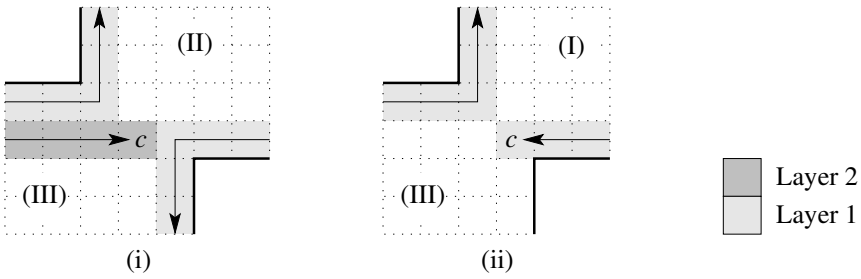


Figure 1.14: Three types of components.

We consider Figure 1.13(i): In the 4. Layer for the first time a split-cell c occurs. Now we decompose the polygon into different components[2]:

[2]Let $A \dot{\cup} B$ denote the **disjoint union** $A \dot{\cup} B = A \cup B$ mit $A \cap B = \emptyset$.

$$P = K_1 \dot{\cup} K_2 \dot{\cup} \{\text{visisted cells of } P\},$$

where K_1 denotes the component that was visited last. SmartDFS recursively works on K_2, returns to c and proceeds with K_1.

By the layernumbers we would like to avoid the situation of Figure 1.11. We will find the split-cell in layer ℓ, which gives three types of components; see Figure 1.14:

(I) Component K_i is *fully* surrounded by layer ℓ.
(II) Component K_i is *not* surrounded by layer ℓ (may be touched by the split-cell only).
(III) Component K_i is *partly* surrounded by layer ℓ (not only touched by the split cell).

Obviously, if a split-cell occurs, we should visit the component of type (III) last because the starting point lies in the outer layers of this component.

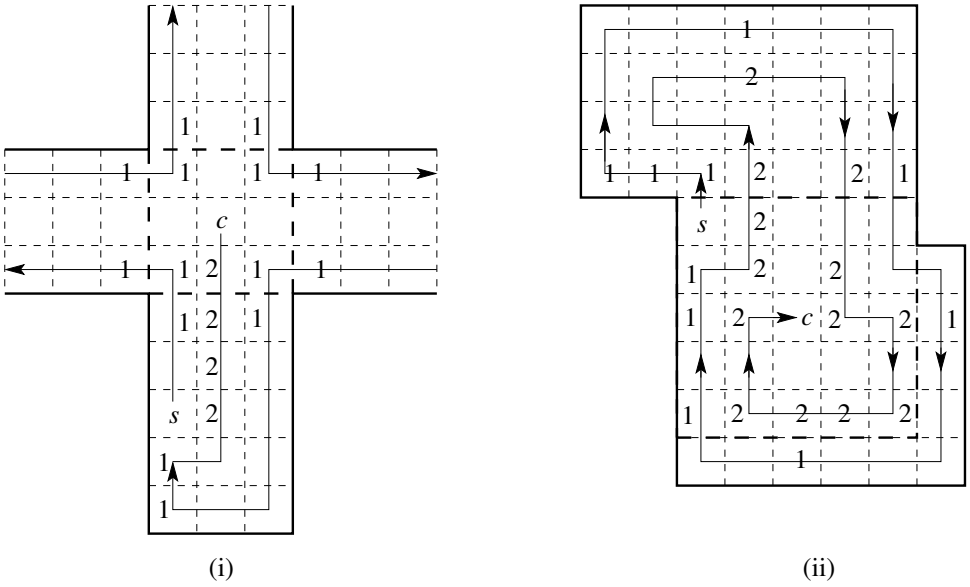


Figure 1.15: Special cases: No component of typ (III) exists.

There are some situations where the a component of type (III) does not exist. For example if the split-cell is the first cell on the next layer, or the component of the starting point was just explored (efficiently). More precisely:

(a) The component with the starting point on its layer was just fully explored in the current layer; see Figure 1.15(i). In this case the order of visiting the remaining components is not critical, we can choose an arbitrary order. This example also shows that at a split-cell more than two components has to be visited. We simply apply one the next step by changing to the Right-Hand-Rule.

(b) Two components have been fully surrounded, because at the split-cell we change from layer ℓ to $\ell+1$; see Figure 1.15(ii). In all other cases at least one additional visited cell is marked with layer number of the split-cell. We can conclude that layer ℓ was closed with the split-cell. This means that the starting point is not part of the layer of the component where the agent currently comes from. Because the agent normally moves by the Left-Hand-Rule, it suffices to apply the Right-Hand-Rule in this case also.

Altogether in both cases we simply apply the Right-Hand-Rule for a single step.

For the overall analysis at a split-cell we consider two polygons P_1 und P_2 as depicted in Figure 1.13(i). Here we detect the component of type (III). K_2 is a component of type (II). Let Q be a

rectangle of edge length (width or height) $2q+1$ around the split cell c so that

$$q := \begin{cases} \ell, & \text{if } K_2 \text{ has type (I)} \\ \ell - 1, & \text{if } K_2 \text{ has type (II)} \end{cases}.$$

Now choose P_2 so that $K_2 \cup \{c\}$ is the q-Offset of $K_2 \cup \{c\}$. The idea is that the rectangle Q will be *added* so that P_2 has the desired form. Now let $P_1 := ((P \backslash P_2) \cup Q) \cap P$, comp. Figure 1.13. The intersection with P is necessary, since there are cases where Q does not totally fit into P. We would like to apply arguments recursively for P_2 and P_1. Let us consider them separately as shown in Figure 1.13(ii). We have choosen P_1, P_2 und Q in a way so that the paths in $P_1 \backslash Q$ und $P_2 \backslash Q$ did not change w.r.t. the paths already performed for P[3]. The already performed paths that lead in P from P_1 to P_2 and from P_2 to P_1 will be used and adapted so that the paths outside Q will not change; see Figure 1.13. We can consider P_1 and P_2 separately.

We know that any cell has to be visited at least once. Therefore we count the number of steps $S(P)$ for polygons P as follows. It is the sum of the cells, $C(P)$, of P plus the extra cost $excess(P)$ for the overall *path* length.

$$S(P) := C(P) + excess(P).$$

The following Lemma gives an estimate for the extra cost w.r.t. the above decomposition around a split-cell.

Lemma 1.10 *Let P be a gridpolygon, c a split-cell, so that two remaining components K_1 und K_2 has to be considered. Assume that K_2 is visited first. We conclude:*

$$excess(P) \leq excess(P_1) + excess(K_2 \cup \{c\}) + 1.$$

Proof. The agent is located at cell c and decides to explore $K_2 \cup \{c\}$ starting from c and return to c. This gives additional cost at most $excess(K_2 \cup \{c\})$, note that the part $P_2 \backslash (K_2 \cup \{c\})$ can only help for the return path. Because c was already visited, we count one additional item for the excess of visited cells. After that we proceed with the exploration of P_1 and require $excess(P_1)$ for this part. □

For the full analysis of SmartDFS we have to prove some structural properties:

Lemma 1.11 *The shortest path between to cells s and t in a simple gridpolygon P with $E(P)$ boundary edges consists of at most $\frac{1}{2}E(P) - 2$ cells.*

Proof. W.l.o.g. we assume that s and t are in the first layer, otherwise we can choose different s or t whose shortest path is even a bit longer. Consider the path, π_L, in clockwise order in the first layer from s to t and the path, π_R, in counter-clockwise order in the first layer from s to t. Connecting π_L and π_R gives a full roundtrip. As in the proof of Lemma 1.9 counting the edges gives 4 more edges than cells which gives

$$|\pi_R| + |\pi_L| \leq E(P) - 4$$

visited cells.

In the worst case both path have the same length, which gives $|\pi(s,t)| = |\pi_R| = |\pi_L|$, and $2|\pi(s,t)| \leq E(P) - 4 \Rightarrow |\pi(s,t)| \leq \frac{1}{2}E(P) - 2$. □

Lemma 1.12 *Let P be a gridpolygon and let c be a split-cell. Define P_1, P_2 und Q as above. For the number of edges we have:*

$$E(P_1) + E(P_2) = E(P) + E(Q).$$

[3] For the uniqueness of this decomposition into P_1 and P_2 we remark that P_1 and P_2 are connected, respectively and $P \cup Q = P_1 \cup P_2$ and $P_1 \cap P_2 \subseteq Q$ holds.

Proof. For arbitrary gridpolygons P_1 and P_2 we conclude

$$E(P_1)+E(P_2)=E(P_1\cup P_2)+E(P_1\cap P_2).$$

Let $Q':=P_1\cap P_2$, we have:

$$\begin{aligned} E(P_1)+E(P_2) &= E(P_1\cap P_2)+E(P_1\cup P_2)\\ &= E(Q')+E(P\cup Q)\\ &= E(Q')+E(P)+E(Q)-E(P\cap Q)\\ &= E(P)+E(Q), \text{ since } Q'=P\cap Q \end{aligned}$$

□

Exercise 7 *Show that for arbitrary two gridpolygons P_1 and P_2 we have $E(P_1)+E(P_2)=E(P_1\cup P_2)+E(P_1\cap P_2)$.*

Using all these arguments we can show:

Theorem 1.13 *(Icking, Kamphans, Klein, Langetepe, 2000)*
For a simple gridpolygon P with C cells and E boundary edges the strategy SmartDFS required no more than

$$C+\frac{1}{2}E-3$$

for the exploration of P (with return to the start). This bound will be attained exactly in some environments. [IKKL00b]

Proof. By the above arguments it suffices to show $excess(P)\leq\frac{1}{2}E-3$. We give a proof by induction on the number of components.

Induction base:
Assume that there is no split-cell. For the exploration of a single component, SmartDFS visits all cells exactly once and return to the start. For visiting all cells we require $C-1$ steps. Now the excess is the shortest path back. By Lemma 1.11 $\frac{1}{2}E-2$ steps suffices which gives the conclusion

Induction step:
Consider the (first) decomposition at a split-cell c. Let K_1,K_2,P_1,P_2,Q be defined as above, assume that K_2 is visited last. We have:

$$\begin{aligned} excess(P) &\leq excess(P_1)+excess(K_2\cup\{c\})+1 \text{ (Lemma 1.10)}\\ &\leq_{\text{(I.A.)}} \frac{1}{2}E(P_1)-3+\frac{1}{2}\underbrace{E(K_2\cup\{c\})}-3+1\\ &\qquad\qquad \leq E(P_2)-8q \text{ (Lemma 1.9)}\\ &\leq \frac{1}{2}\Big[\underbrace{E(P_1)+E(P_2)}\Big]-4q-5\\ &\qquad\qquad \leq E(P)+4(2q+1) \text{ (1.12, Def. of } Q)\\ &\leq \frac{1}{2}E(P)-3 \end{aligned}$$

□

A Java-Applet for the Simulation of SmartDFS and different strategies can be found at:

`http://www.geometrylab.de/`

Finally, we would like to show, how to compute the offline shortest paths in gridpolygons Of course the Dijkstra algorithm can also be applied on the gridgraph, but this algorithm does not use the grid structure directly. As an alternative we apply Algorithm 1.5 (C. Y. Lee, 1961, [Lee61]), the running time is only linear in the number of overall cells. The algorithm simulates a wave propagation starting from the goal. Any cell will be marked with a label indicating the distance to the goal. Obstacles *slow down* the propagation a bit; see Figure 1.16. When the wave reaches the starting point s, we are done with the first phase. For computing the path we start at s and move along cells with strictly decreasing labels. Obviously, the shortest path need not be unique.

Algorithm 1.5 Algorithm of Lee

Shortest path from s to t in a gridpolygon

```
Datastructur: Queue Q
// Initialise
Q.InsertItem(t);
Mark t with label 0;
// Wave propagation:
loop
  c := Q.RemoveItem();
  for all Cells x such that x is adjacent to c and x is not marked do
    Mark x with the label of label(c) + 1;
    Q.InsertItem(x);
    if x = s then break loop;
  end for
end loop
// Backtrace:
Move along cells with strongly decreasing labels from s to t.
```

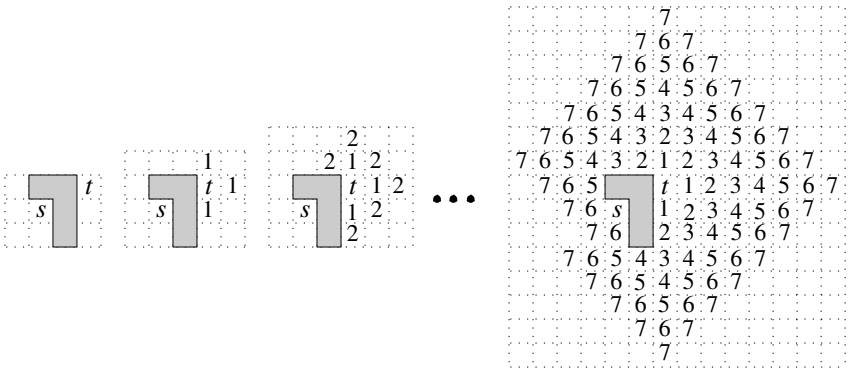


Figure 1.16: Wave-Propagation.

1.4.2 Competitive ratio of SmartDFS

The corridor of width 3, see Figure 1.7, indicates that the competitive ratio of SmartDFS should be better than 2. SmartDFS runs 4 times though the corridor whereas the shortest path visits any cell only once. This gives roughly a ratio of $\frac{4}{3}$. We will show that this is the worst-case for SmartDFS. The gap between e $\frac{7}{6}$ and $\frac{4}{3}$ is small.

For the analysis we first give a precise definition of the structure of parts of gridpolygons which will be explored in an optimal fashion. The SmartDFS Strategy does not make any detours within these passages.

For a *corridor* of widths 1 this is abviously true. But also corridors of width 2 will be passed optimally, since SmartDFS runs forth and back along different tracks; see Figure 1.17. We give a formal definition of the *narrow passages*.

Definition 1.14 The set of cells that can be deleted such that the layernumber of the remaining cells do not change are called narrow passages of P.

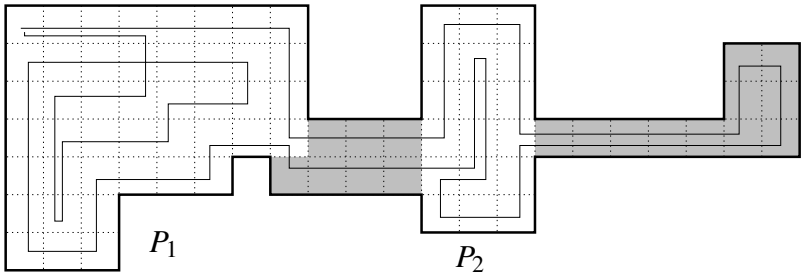


Figure 1.17: SmartDFS is optimal in narrow passages.

SmartDFS passes narrow passages optimally since they allow an optimal forth and back pass-through. There are no additional detours at the entrance and exit of a narrow passage because they consist of cells in the first layer. They can be considered as *gates*. The entrance and exit is always precisely determined.

The idea is to consider polygons without narrow passages first. There is a fixed relationship between edges and cells.

Lemma 1.15 *Let P be a simple gridpolygon without narrow passages and without a split-cell in the first layer. We have*

$$E(P) \leq \frac{2}{3}C(P) + 6\,.$$

Proof. A 3×3 gridpolygon has precisely this property, $C(P) = 9$ und $E(P) = 12$. Any gridpolygon with the above conditions can be reduced by successively removing columns or rows such that in each step the property remains true and such that always at least 3 cells and at most 2 edges will be removed. This is an exercise below.

Starting backwards from the property $E(P_0 = \frac{2}{3}C(P_0) + 6$ we will maintain the bound $E(P_i) \leq \frac{2}{3}C(P_i) + 6$ since we add at least 3 cells and add at most 2 edges. Finally, $E(P) \leq \frac{2}{3}C(P) + 6$ holds. □

First, we show that the overall number of exploration steps of SmartDFS decreases for the given class of polygons.

Lemma 1.16 *A simple gridpolygon P with $E(P)$ edges and $C(P)$ cells, without narrow passages and without a split-cell in the first layer well be explored by SmartDFS with no more than $S(P) \leq C(P) + \frac{1}{2}E(P) - 5$ steps.*

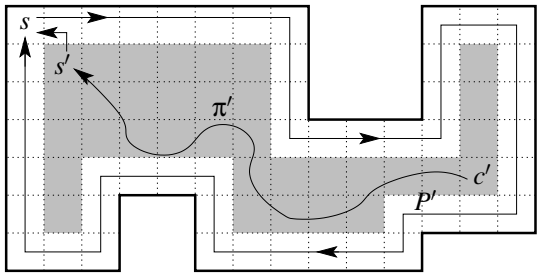


Figure 1.18: A simple gridpolygon without narrow passages and no split-cell in the first layer has the property $E(P) \leq \frac{2}{3}C(P)+6$. After the first coil SmartDFS starts in the 1-Offset P'. The return path to c' from an arbitrary point in P' is shorter than $\frac{1}{2}E(P)/2-2$.

Proof. From Theorem 1.13 we conclud $S(P) \leq C(P)+\frac{1}{2}E(P)-3$. By the properties of P, SmartDFS performs a full first round from s to the first cell s' in the second layer. After that, in principle we start SmartDFS again at s' in a gridpolygon (1-Offset P' of P); see P' in Figure 1.18. P' is path connected and by Lemma 1.9 P' has 8 edges less than P;

The cells in the first layer have been visited optimally the path length from s to s'coincidence with the number of cells in the first layer. Finally, we have to count two additional steps from s' to s. Altogether, we require $S(P) \leq C(P)+\frac{1}{2}(E(P)-8)-3+2 = C(P)+\frac{1}{2}E(P)-5$ steps. □

With the statements above we will be able to prove the main result.

Mit diesen Vorbereitungen können wir die kompetitive Schranke beweisen.

Theorem 1.17 *(Icking, Kamphans, Klein, Langetepe, 2005) The SmartDFS strategie for the exploration of simple gridpolygons is $\frac{4}{3}$-competitive!* *[IKKL05]*

Proof.

Let P be a simple gridpolygon. First, we remove the narrow passages from P. We know that the entrance and exits over the gates by SmartDFS are optimal. We obtain a sequence P_i, $i = 1,\ldots,k$ of gridpolygons connected by narrow passages. See for example P_1 and P_2 in Figure 1.17.

We can consider the gridpolygons P_i separately. We can also assume different starting points. The movement between the gates count for the required additional steps. It is sufficient to show $S(P_i) \leq \frac{4}{3}C(P_i)-2$ for any subpolygon. This bound exactly holds for $3 \times m$ gridpolygons for even m; see Figure 1.19.

We show the bound by induction over the number of splil-cells.

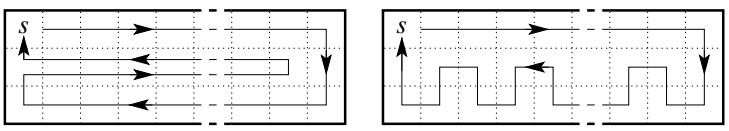


Figure 1.19: In a corridor of width 3 and with even lenght the bound $S(P) = \frac{4}{3}S_{\mathrm{Opt}}(P)-2$ holds.

Induktion-Base: If P_i has no split-cell, there is also no split-cell in the first layer. We apply Lemma 1.16 and Lemma 1.15 and obtain:

$$S(P_i) \quad \leq \quad C(P_i)+\frac{1}{2}E(P_i)-5$$

$$\begin{aligned} &\leq C(P_i) + \frac{1}{2}\left(\frac{2}{3}C(P_i) + 6\right) - 5 \\ &= \frac{4}{3}C(P_i) - 2\,. \end{aligned}$$

Induktion-Step: If there is no split-cell in the first layer we can apply the same arguments as above. Therefore, we assume that the first split occurs in the first layer. Two cases can occur as depicted in Figure 1.20.

In the first case the component of type (II) was not visited before and we define $Q := \{c\}$. The second case occurs, if the split-cell c is diagonally adjacent to a cell c'; compare Figure 1.20(ii), (iii) and (iv). We build the smallest rectangle Q that contains c and c'. In case (ii) and (iii) Q is a square of size 4. In case (iv) by simple adjacency Q is a rectangle and $|Q| = 2$.

Analogously to the proof of Theorem 1.13 we split the polygons into parts P' and P'' both containing Q.

Here P'' is of type (I) or (II) and P' the remaining polygon. das Polygon der Komponenten vom Typ (I) oder (II) und P' das andere.

For $|Q| = 1$ (see Figure 1.20(i)) we have $S(P_i) = S(P') + S(P'')$ and $C(P_i) = C(P') + C(P'') - 1$. We apply the induction hypothesis on P' and P'' (they have one split-cell less) and obtain:

$$\begin{aligned} S(P_i) &= S(P') + S(P'') \\ &\leq \frac{4}{3}C(P') - 2 + \frac{4}{3}C(P'') - 2 \\ &\leq \frac{4}{3}C(P_i) + \frac{4}{3} - 4 \;<\; \frac{4}{3}C(P_i) - 2\,. \end{aligned}$$

For $|Q| = 4$ we argue that by the union we will save some steps that will occur for the separate explorations. We consider P' and P'' separately, first. The movements from c' to c (and c to c') count in both polygons. For the complete P_i the path from c' to c (and c to c') are given either P' or in P'', this means that we save $4 = |Q|$ steps.

We have $S(P_i) = S(P') + S(P'') - 4$ and $C(P_i) = C(P') + C(P'') - 4$. By induction hypothesis for P' and P'' we conclude:

$$\begin{aligned} S(P_i) &= S(P') + S(P'') - 4 \\ &\leq \frac{4}{3}C(P') + \frac{4}{3}C(P'') - 8 \\ &= \frac{4}{3}\left(C(P') + C(P'') - 4\right) - \frac{8}{3} \\ &< \frac{4}{3}C(P_i) - 2\,. \end{aligned}$$

The case $|Q| = 2$ is left as an exercise.

Altogether an optimal strategy requires $\geq C(P_i)$ steps or $\geq C(P)$ in total and we have a competitive ratio of $\frac{4}{3}$. □

Exercise 8 *Analyse the remaining case $|Q| = 2$ in the above proof.*

If we compare the result to Theorem 1.7 there is a gap of size $\frac{1}{6}$ between $\frac{7}{6}$ and $\frac{4}{3}$. Recently, both parts have been improved. There is a lower bound of $\frac{20}{17}$ and an upper bound of $\frac{5}{4}$ shown by Kolenderska et. al 2010. In principle the strategy is a local improvement of SmartDFS and the lower bound is an extension of our construction. The result comes along with a tedious case analysis.

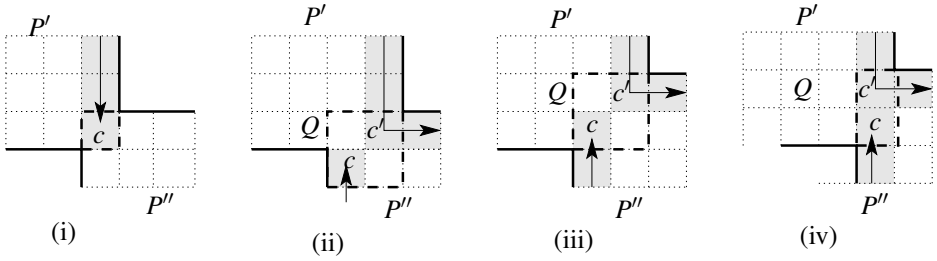


Figure 1.20: A gridpolygon P_i that is separated into components of type (I) or (II) at the split-cell. The rectangle Q is always inside P_i.

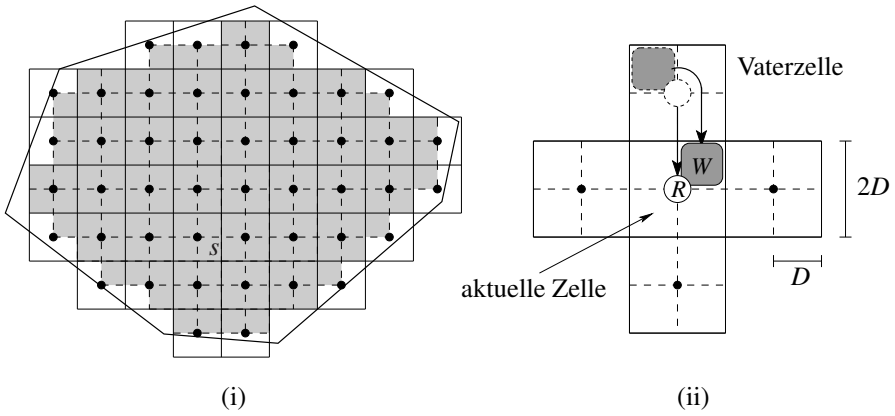


Figure 1.21: $2D$-cells and $D \times D$ sub-cells.

1.4.3 Exploration of general gridpolygons

For the more general exploration of gridpolygons we first slightly change the model[4]: We consider an agent that is located at the center of 4 cells of size $D \times D$. The tool for the exploration still has size $D \times D$ as before and moves freely around the agent. More precisely, we consider 4 **sub-cells** of size $D \times D$ and unify them to a $2D$-cell[5]; see Figure 1.21(i).

It can happen that for the $2D$-cell, not all sub-cells belong to the initial gridpolygon, since some of the sub-cells simply belong to the boundary. Such $2D$-cell are denoted as **partially occupied cells**.

In Figure 1.21(i) all cells intersected by the original polygonal segments are partially occupied (compare also reffigfigOnline/PolyToGrid on page 8). The agent is always located in the center of the $2D$-cell. Analogously to the SmartDFS model, the agents scans the four adjacent $2D$-cells. The tool moves freely around the agent, we would like the count the number of steps of the tool; see also Figure 1.21(ii).

The current cell of the agent is denoted as *current cell*. The *parent cell* of the agent is the cell where he is actually coming from. In the beginning we initially an arbitrary adjacent $2D$-cell as the parent cell.

The strategy "Spanning-Tree-Covering" (STC) constructs a spanning tree for all connected $2D$-cells that are also not occupied. The tool moves along the spanning tree by the Left-Hand-Rule. The construction can be done fully online. The $2D$-cells are detected by the Right-Hand-Rule. Obviously by this approach the tool exactly visits any cell at most once by following the spanning tree. Figure 1.22(i) shows an example for the efficient exploration of all non-occupied cells by $2D$-Spiral-STC. As mentioned before, for the start we can choose an arbitrary parent cell.

The disadvantage of $2D$-Spiral-STC is, that we do not visit sub-cells by that tool which actually lie in the connected component of the sub-cells. Now we relax the behaviour of $2D$-Spiral-STC. The strategy

[4] We will see later that the change was only done for the reason of a convenient analysis and description.

[5] In the following a cell always denotes a $2D$-cell.

Algorithm 1.6 2D-Spiral-STC

2DSPSTC(*parent*, *current*):

Mark *current* as visited.
while *current* has unvisited neighbour cell **do**
- From *parent* search in ccw order for a neighbouring cell *free*, which is not marked as visited and is not partially occupied.
- Build the spanning tree edge from *current* to *free*.
- Move the tool by Left-Hand-Rule along the spanning tree edge to the first sub-cell of *free*.
- Call 2DSPSTC(*current*, *free*).

end while
if $current \neq s$ **then**
- Move by the Left-Hand-Rule along the spanning tree edge back from *current* to the first sub-cell of *parent*.

end if

Algorithm 1.7 SpiralSTC

SPSTC(*parent*, *current*):

Mark *current* as visited.
while *current* has unvisited neighbour cell **do**
- From *parent* search in ccw order for the first neighbouring cell *free*.
- Build a spanning tree edgs from *current* to *free*.
- Move the tool along the spanning tree edge to the first sub-cell of *free*. The movement depends on the local situation. For double-sided edges the tool moves by Left-Hand-Rule along the edge. For single-sided edges the tool might change to the other (left) side of the spanning tree edge in order to avoid an occupied sub-cell for reaching the corresponding sub-cell.
- Call SPSTC(*current*, *free*).

end while
if $current \neq s$ **then**
- Move along the spanning tree edge back from *current* to the first possible sub-cell of *parent*. The movement depends on the type of the edge, as mentioned above.

end if

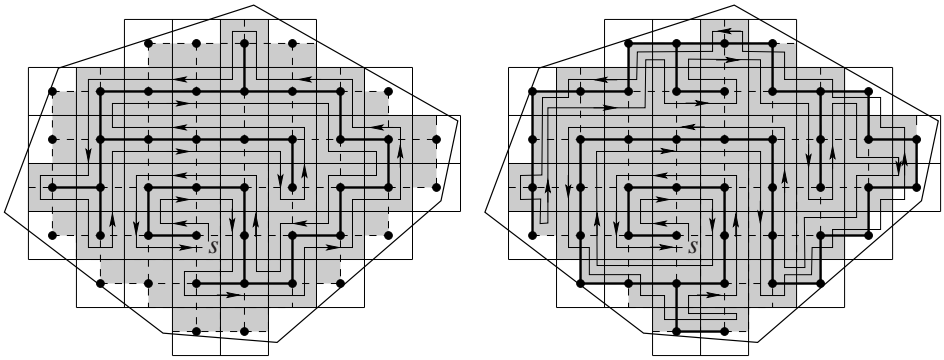


Figure 1.22: Examples for (i) 2D-Spiral-STC and (ii) Spiral-STC.

Spiral-STC (Algorithm 1.7) also constructs a spanning tree in an online fashion. But we also insert a corresponding edge if a partially occupied 2D-cell contains sub-cells that are still reachable by the tool. In this case the tool cannot always move the the Left-Hand-Rule along the spanning tree edge. The tool has to avoid occupied sub-cells and visits some sub-cells more than once. For systematically analysing the corresponding additional sub-cell visits of the tool we make use of the following notion:

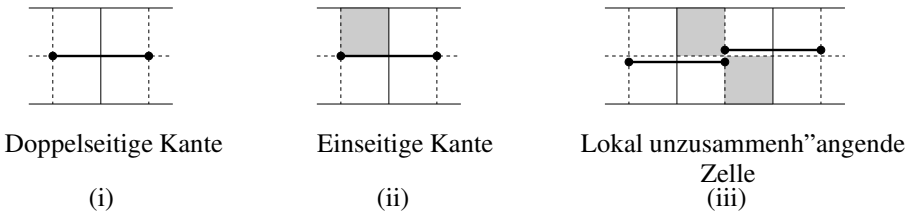


Figure 1.23: (i) Double-sided edge, (ii) one-sided edge, (iii) locally disconnected 2D-cell.

Definition 1.18 A spanning tree edge construced by STC in a gridpolygon P is denoted as
(i) **double-sided edge**, if all adjacent sub-cells belong to the gridpolygon P (Figure 1.23(i)),
(ii) **single-sided edge**, if at least one of the adjacent sub-cells is a boundary sub-cell of P (Figure 1.23(ii)).

Double-sided edges are handled in the same way as in the 2D-Spiral-STC strategy. Single-sided edges impose a detour for the tool, some sub-cells will be visited more than once since the tool changes to the other side of the spanning tree edge. For the analysis we will consider the corresponding cases systematically. A special case occurs, if the situation imposes two spanning tree edges for the same cell from different directions. The cell is locally disconnected in this case; see Figure 1.23(iii). This 2D-cell will be visited twice from different directions. For simplicity we internally double the corresponding vertex and the spanning tree has exactly one incoming edge for any vertex. For the analysis we have to take care that we count the cell only once. An example of the execution of Spiral-STC is shown in Figure 1.22(ii).

By the preference rule for the 2D-cells the Spiral-STC constructs spanning trees with many windings. This is not always intended, especially for lawn-mowing or vacuum-cleaning a tool should try to avoid so many turns. The number of turns might also be part of the cost model. The Scan-STC variant has a fixed given preference for vertical or horizontal edges. We would like to make local decision for the construction of spanning tree edges. In our examples we prefer a vertical scan of the gridpolygon. For this we extend the sensor model and allow to have information about all diagonally adjacent 2D-cells of a current cell.

The idea is that the construction of a horizontal edge will be postponed, if it is clear that we can also reach the 2D-cell by another vertical spanning tree edge. To keep the rule simple we only look ahead as

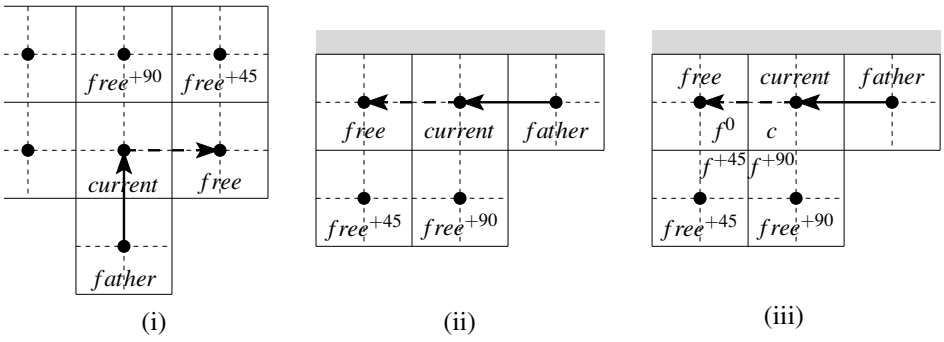


Figure 1.24: Avoid horizontal edges with the Scan-STC.

indicated in Figure 1.24 (i) and (ii). Here we currently would like to build a horizontal edge. The agent is located at cell *current* and is looking (in ccw order starting from *father*) for the first free cell *free*. If there is a counterclockwise path from *free* over $free^{+45}$ and $free^{+90}$ back to the current cell, we change the preference and build a spanning tree edge to $free^{+90}$. Here $free^{+45}$ lies on the sam row as *free* and is the the next cell in ccw order from *free*. $free^{+90}$ is the next cell in ccw order from $free^{+45}$ in the same column as *current*.

If the full turn exists, the cell *free* will also be reached from $free^{+45}$ be a vertical edges and $free^{+45}$ can be reached from $free^{+90}$. Note that we have extended the sensor model in this case and also have information about diagonally adjacent edges.

Analogously, we can also consider partially occupied $2D$-cells and apply the same idea. For the corresponding avoidance rule we consider the sub-cells c, f^0, f^{+45} and f^{+90} instead if the cells *current*, *free*, $free^{+45}$ und $free^{+90}$; see Figure 1.24(iii).

By the above idea we could define a strategy $2D$-Scan-STC that corresponds to $2D$-Spiral-STC. We skip this step and directly define a Scan-STC Algorithm that makes use of the sub-cells c, f^0, f^{+45} and f^{+90} by the same arguments. If f^{+45} and f^{+90} are also free, we will reach f^0 from f^{+45} and in turn f^{+45} from f^{+90}. Algorithm 1.8 summarizes this behaviour.

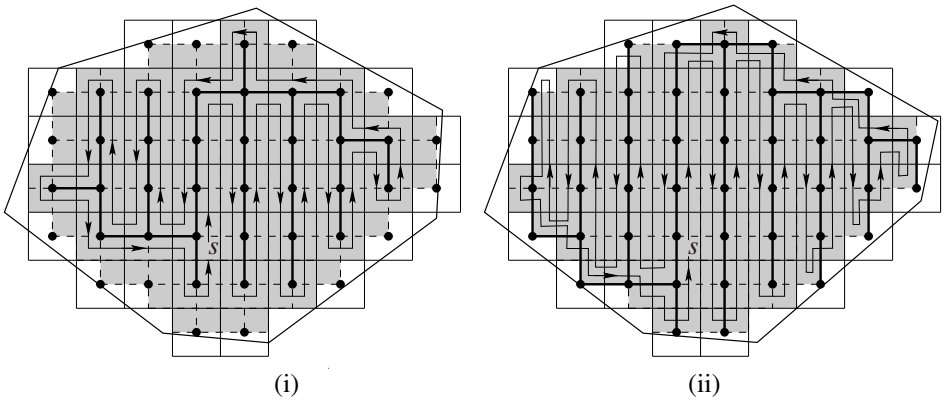


Figure 1.25: Examplle for (i) $2D$-Scan-STC, (ii) Scan-STC.

Algorithm 1.8 ScanSTC

SCSTC(*parent*, *current* **):**

Mark *current* as visited.

while *current* has unvisited neighbouring cell **do**

• From *parent* search in ccw order for the fisrt non-visited neighbouring cell *free*.

if Spanning tree edge from *current* to *free* is horizontal and sub-cells f^{+45} and f^{+90} are free **then** $free := free^{+90}$.

end if

• Build a spanning tree edge from *current* to *free*.

• Move the tool along the spanning tree edge to the first sub-cell of *free*. The movement depends on the local situation. For double-sided edges the tool moves by Left-Hand-Rule along the edge. For single-sided edges the tool might change to the other (left) side of the spanning tree edge in order to avoid an occupied sub-cell and reach the corresponding sub-cell.

• Call SCSTC(*current*, *free*) auf.

end while

if $current \neq s$ **then**

• Move along the spanning tree edge from *current* back to the first possible sub-cell of *parent*. The movement depends on the type of the edge, as mentioned above.

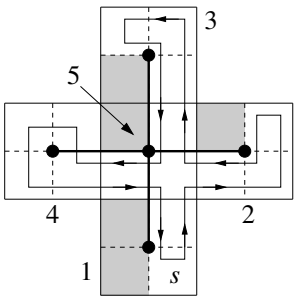
end if

Theorem 1.19 *(Gabriely, Rimon, 2000)*
Let P be a gridpolygon with C sub-cells. Let K be the number of all sub-cells, which are diagonally adjacent to an occupied (boundary) sub-cell[6]*. The gridpolygons P will be explored by Spiral-STC and Scan-STC in time $O(C)$ and space $O(C)$. There are no more than*

$$S \leq C + K$$

exploration steps, S, for the tool. *[GR03]*

Proof.
Correctness:
Both algorithms construct a spanning tree by DFS such that any $2D$-cell which has reachable D sub-cells will be visited. The tool moves along the spanning tree on both sides – as long as the path is not blocked – and visits all sub-cells that are *touched* by the spanning tree.



Zelle	”Ubergr.	Intern	Gesamt	Randzellen
1	0	1	1	2
2	1	2	3	3
3	1	2	3	3
4	1	1	2	2
5	1	2	3	3

Figure 1.26: Estimating the double visits of sub-cells by STC locally.

Path length:
The number of steps for the tool is essential the sum of the visited sub-cells C. If the tool changes to the left side of a spanning tree a detour has to be made and some sub-cells will be visited more than once. Beyond C we simply count the number of sub-cells that are visited more than once and locallly charge the sub-cells of a $2D$-cell for these visits.

[6] K can be estimated by the number of sub-cells in the first layer of P.

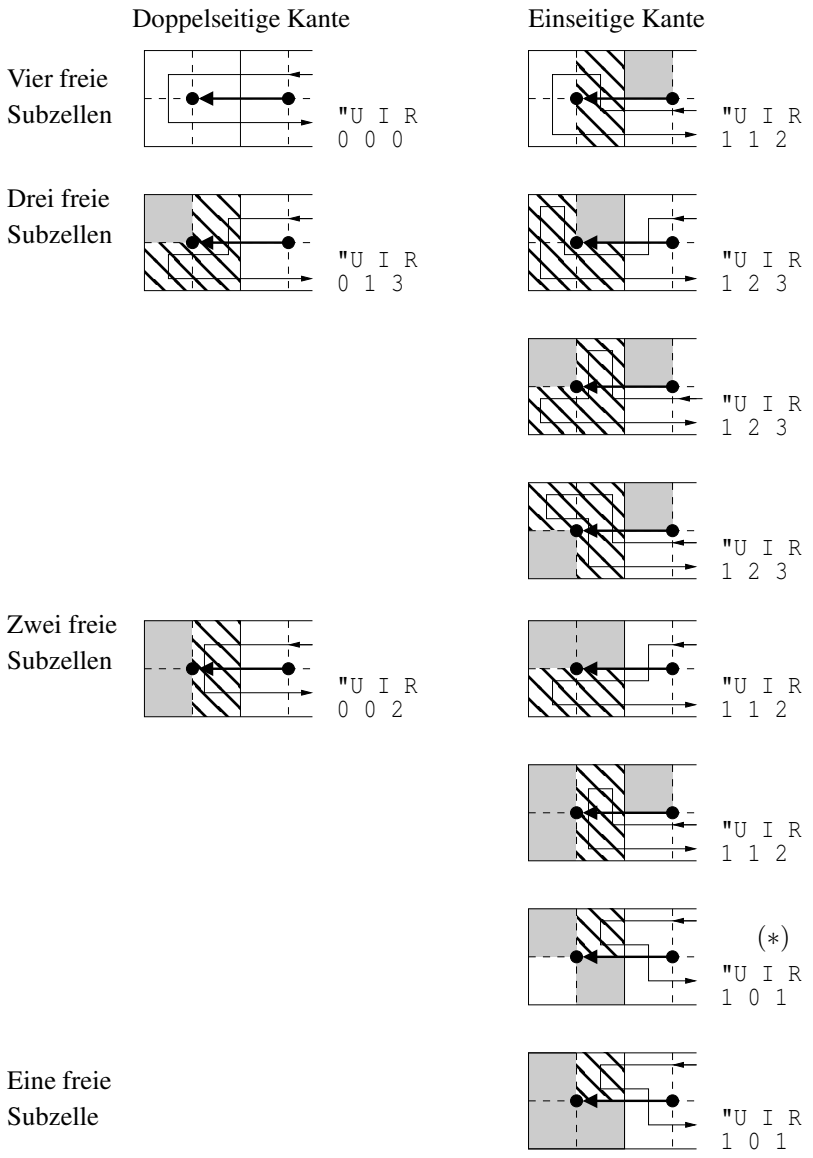


Figure 1.27: Analysis of STC, all possible cases.

We differentiate between *inner* double visits and *intra* double visits. The latter one occur during the movement inside a 2D-cell if a sub-cell is visited again. The former one occur if we leave a 2D cell c along the spanning tree to a neighbouring cell and the corresponding sub-cell was visited before. For this double visit we also charge the 2D cell c, since it was responsible for the detour.

Any 2D-cell c is visited for the first time by an incoming spanning tree edge. The inner-cell double visit will occur only if the cell c is left again along this edge. Figure 1.26 shows an example for counting inner and intro double visits. For cell 1 sub-cell s is visited twice, an intra double visit. The sub-cell above s is also visited twice, but by the movement back for 5 to 1 along the spanning tree edge. Therefore 2D-cell 5 is charged for this by an inner double visit.

The table of2D-cells Figure 1.26 shows the number of inner and intra double visits for any 2D-cell. We charge the 2D-cells for these double visits. On the other hand, for any 2D-cell we also count the number of sub-cells that are diagonally adjacent to a boundary sub-cell. The corresponding boundary sub-cell need not lie inside the 2D-cell itself. Surprisingly, the sum of inner and intra double visits does never exceed the number sub-cells with diagonally adjacent neighbours. This is also given in the table of Figure 1.26.

For a full systematic proof we refer to Figure 1.27. Any 2D-cell c is visited by some spanning tree

edge for the first time and the inner double visits can only occur on this edge. Therefore it is sufficient to consider the $2D$-cell without other outgoing spanning tree edges. For any intra detours only sub-cells of the current cell are responsible. For the inner detour only the parent cell was responsible.

We distinguish between double sided and single sided edges and between the number of boundary sub-cells inside the corresponding $2D$-cell c. We always count inner and intra double visits and compare the sum to the number of sub-cells adjacent to boundary sub-cells.

For all reasonable cases the sum of double visits is always covered locally by the number sub-cells adjacent to boundary sub-cells. The case marked with $(*)$ is a bit tricky. The corresponding $2D$ cell might also be visited by another spanning tree edge. This is not critical because there is only 1 double visit in this case for each sub-case. They can be handled separately.

Running time and space requirement
The tool performs at most $C + K \leq 2C$ steps. Any movement is computed locally in $O(1)$ time. The corresponding overall information required does not exceed $O(C)$. □

Finally, we consider the Scan-variants of the STC-Algorithms. We would like to give a rough estimate for the efficiency in avoiding horizontal edges by $2D$-Scan-STC.

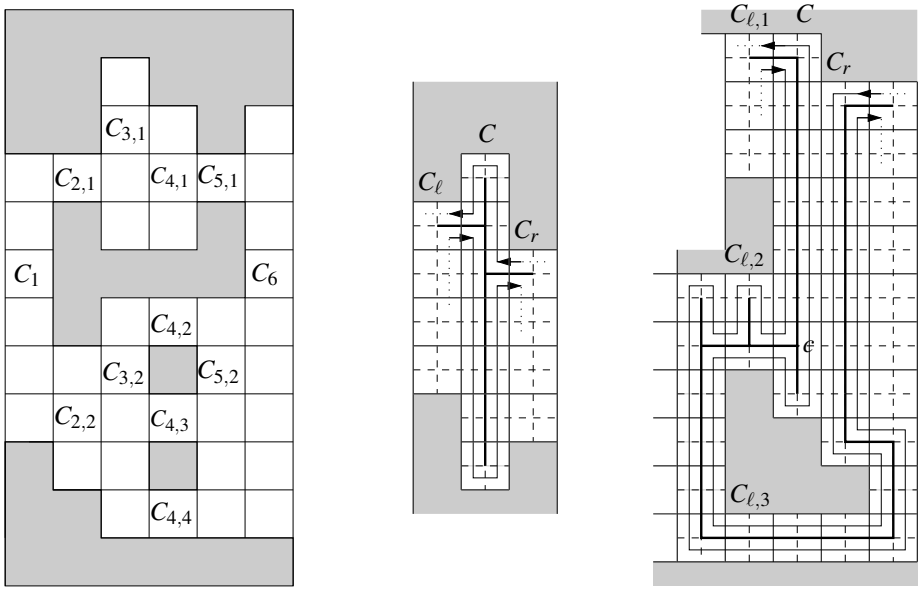


Figure 1.28: (i) Columns and the change of connectivity, (ii) Columns without changes, (iii) Difficult online situation.

We consider columns of the gridpolygon and from left to right we count the change of the connectivity from a column to its neighbour on the right. For example on Figure 1.28(i) there is a numbering of the columns and the number of different vertical components of the columns. From left to right we sum up all differences in the number of components of a column to its neighbour. In Figure 1.28(i) column C_1 has one component and in column C_2 this component split into two components $C_{2,1}$ and $C_{2,2}$. This gives a difference of 1. The components $C_{5,1}$ and $C_{5,2}$ of column C_5 run together in C_6 to a single component. This also is a change of 1 in the difference. Of course also many parts might be involved. We count the changes of any component separately. Let Z denote the sum of all these local changes.
The number Z is a measure for the additional horizontal edges of the spanning tree of Scan-STC against an optimal number of spanning tree edges:

Theorem 1.20 *(Gabriely, Rimon, 2000)*
Let P be a gridpolygon. Let H_{Opt} denote the minimal number of spanning tree edges among all $2D$ spanning trees of P. Let Z be the above number of connectivity changes for the columns of the $2D$-cells.

2D-Scan-STC constructs a spanning tree with at most

$$H_{\mathrm{STC}} \leq H_{\mathrm{Opt}} + Z + 1$$

horizontal edges. *[GR03]*

Proof.(Sketch)
If there is no change in a $2D$ column, the optimal spanning tree and $2D$-Scan-STC will visit and leave the column only once; compare Figure 1.28(ii). The main problem is that by $2D$-Scan-STC a connected component of a column will be left by the spanning tree to the same side more than once. This can only happen, if there are changes in the connectivity; see Figure 1.28(iii). □

Concluding remarks

Arkin, Fekete and Mitchell gave some approximation results for the offline exploration of gridpolygons; see [AFM00]. Betke, Rivest und Singh considered a variant of the exploration problem. They introduced the following piecemeal-condition: The agent has to explore an environment with rectangular obstacles and has to return to the start from time to time (charging an accumulator); see [BRS94]. A strategy for this problem for general grid-environments stems from Albers, Kursawe und Schuierer [AKS02].

1.5 Constrained graph-exploration

We consider the problem of the exploration of an unknown graph $G = (V,E)$ starting from some fixed vertex $s \in V$. This means that we would like to visit all edges and vertices of G. First, we consider unit-weights which means that any visit of an edge has cost 1. Different from the previous section we consider a constrained version of the exploration, due to the following practical models. Let r denote the radius or depth of the graph w.r.t. s. I.e., r is the maximal length of a shortest path from s to some vertex $v \in V$. Let us first assume that r is known, but not the graph itself.

1. The agent is bounded by a tether of length $\ell = (1+\alpha)r$ (for example a cable constraint).
2. The agent has to return to the start after any $2(1+\alpha)r$ steps (for example an accumulator has to be recharged).
3. A large graph should be explored up to a given fixed depth d (for example for searching a close by target).

The above third variant will be applied to a searching heuristic with increasing depth, later. First, we show some simple simulation resutls. If an algorithm for the tether variant is known, one can also establish an accumulator strategy with some extra cost.

Lemma 1.21 *Given a tether variant strategy with tether length $l = (1+\alpha)r$ and overall cost T. For any $\beta > \alpha$ there is an accumulator-strategy with cost $\frac{1+\beta}{\beta-\alpha}T$*

Proof. We design the accumulator strategy by following the tether strategy. After any $2(\beta-\alpha)r$ steps we move back from the current vertex v to the start, recharge the agent and move back to v. Then we proceed with the next step of length $2(\beta-\alpha)r$ of the tether strategy path. In the tether strategy for any vertex v, we are never more than $(1+\alpha)r$ away from the start. That is $2(\beta-\alpha)r+2(1+\alpha)r = 2(1+\beta)r$ always result in correct loops. The strategy is correct.

On the other hand, we have cost T for following the tether path and additional cost for moving back and force. We move back at most $\frac{T}{2(\beta-\alpha)r}$ times and require $2(1+\alpha)r$ steps for any movement. This gives total cost:

$$T + \frac{T}{2(\beta-\alpha)r} \cdot 2(1+\alpha)r \quad = \quad T\frac{\beta-\alpha+1+\alpha}{\beta-\alpha} = \frac{1+\beta}{\beta-\alpha}T\,.$$

□

Exercise 9 *Given an accumulator strategy S with accumulator size $2(1+\beta)r$ and overall cost T. For a given $\alpha > \beta$ design an efficient tether strategy that makes use of S so that the cost of the tether strategy is $f(\alpha,\beta)\cdot T$. Determine $f(\alpha,\beta)$ precisely.*

We can also consider the Offline-variant of the problem. In this case the graph is fully known. To the best of our knowledge the complexity of the Offline-variant (computing the best strategy) is not known. Since in the case that the tether is very long, the Hamiltonian-path problem appears to be part of the problem, the problem is assumed to be NP-hard.

If the optimal Offline-strategy is not known, we can design an Offline-strategy that approximates the optimal strategy. We consider the accumulation variant and assume that the accumulator has size $4r$.

Lemma 1.22 *Let us assume that an accumulator of size $4r$ is given. There is a simple Offline algorithm that explores a graph of depth r with no more than $6|E|$ steps.*

Proof. We consider the DFS walk among the edges of the graph which requires $2|E|$ steps. Now we split this overall path into pieces of size $2r$. Similarly to the simulation in the proof above we successively move to the start vertices of these subpaths, follow the DFS path for $2r$ steps and return to the start after that. In total the accumulator of size $4r$ is sufficient. Moving along the DFS path gives $2|E|$ steps. There are no more than $\left\lceil \frac{2|E|}{2r} \right\rceil$ sub-paths that require no more that $\left\lceil \frac{|E|}{r} \right\rceil 2r$ steps in total. We have $\left\lceil \frac{|E|}{r} \right\rceil 2r \leq \left(\frac{|E|}{r} + 1 \right) 2r \leq 2|E| + 2r$ which shows that $4|E| + 2r \leq 6|E|$ is sufficient. □

From now on we consider only the tether variant, for the accumulation variant similar results can be shown. A first simple idea is to take the tether length for the DFS walk into account.

Just performing DFS is not always possible. A BFS approach is always possible but results in too many exploration steps; see Figure 1.29. Therefore we apply DFS with the tether restriction as given in Algorithm 1.9. There is a backtracking step, if the tether is fully exhausted. We call this algorithm bDFS for bounded DFS.

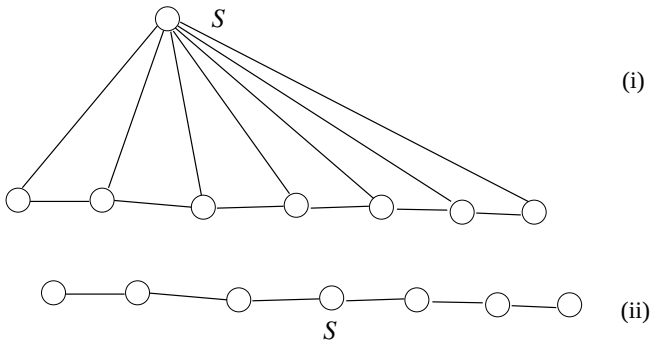


Figure 1.29: (i) A Graph with n vertices and with depth $r = 1$, pure DFS would require a tether of length $n-1$. (ii) A graph of depth n, BFS with a tether of length n requires $\Omega(n^2)$ steps.

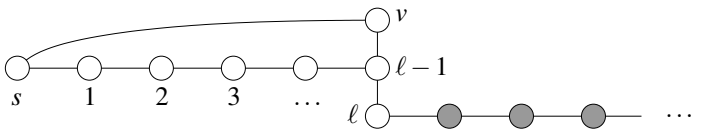


Figure 1.30: bDFS does not reach all vertices.

Algorithm 1.9 boundedDFS

bDFS(v**,** ℓ **):**
  if $(\ell = 0) \vee$ (no adjacent non-explored edges) **then**
    RETURN
  end if
  for all non-explored edge $(v,w) \in E$ **do**
    Move from v to w along (v,w).
    Mark (v,w) as *explored*.
    bDFS(w, $\ell - 1$).
    Move back from w to v along (v,w).
  end for

In general bDFS is not sufficient for the full exploration of a graph. For example in Figure 1.30 we have the problem that the dark-colored vertices cannot be reached, if the algorithm first chooses the path along the vertices $1, 2, \ldots, \ell - 1$, visits vertex l, v and s and winds back to the start s. The path from s over v is short enough but will not be considered by bDFS.

Therefore we would like to call bDFS from different sources. The aim is to achieve a constant competitive algorithm. In Algorithm 1.10 we maintain a set of (edge) disjoint trees $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ with root vertices $s_1, s_2, \ldots, s_k$, respectively. The trees still contain incomplete vertices where not all adjacent edges have been visited. We choose a tree T_i with the minimal distance from s to root s_i among all trees of $\mathcal{T}$. From this tree we prune subtrees T_{w_j} with root vertices w_j, so that w_j is a certain distance ($minDist = \frac{\alpha r}{4}$) away from s and T_{w_j} has a certain minimal depth (determined over $minDepth = \frac{\alpha r}{2}$). Those trees will be inserted into $\mathcal{T}$. The pruning forces the trees of $\mathcal{T}$ to have a minimum size, it is still worth visiting them.

After pruning, the rest of T_i will be explored by DFS and if an incomplete vertex will be found, we start bDFS with the current remaining tether length for the exploration of *new* edges. The newly explored edges and vertices build a graph G'. If G' has incomplete vertices, we construct a spanning tree T' with a root vertex s', where s' is the vertex in T' closest to s in the current overall explored graph G^*. T' will be inserted into $\mathcal{T}$. After the overall DFS (and bDFS) walk in T_i we delete all trees of $\mathcal{T}$ that are now fully explored. Some of the trees in $\mathcal{T}$ might have common vertices. We merge those trees and build a new spanning tree for them with a new root vertex.

A scheme of the algorithm is shown in Figure 1.31. We have done the prune step by values $(2,4)$. Otherwise, we have to build very large example graphs.

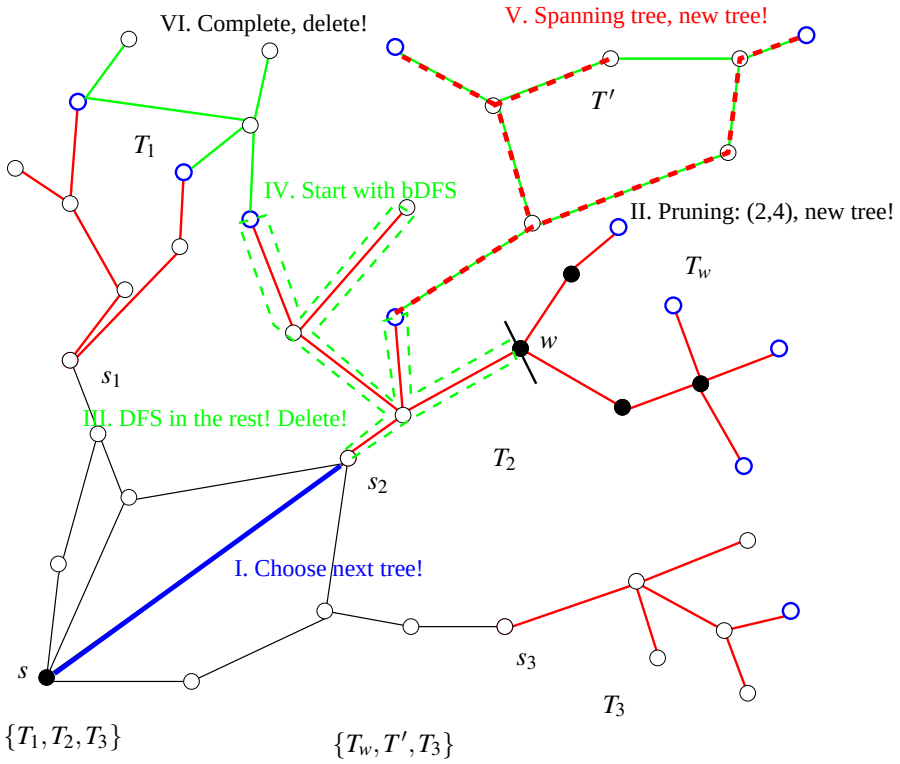


Figure 1.31: The algorithm maintains a set of disjoint trees $\mathcal{T} = \{T_1, T_2, T_3\}$ and choose the tree T_2 with minimal distance $d_{G^*}(s, s_i)$. After that the tree is pruned. Subtrees of distance 2 away from s_2 with vertices inside that have distance at least 4 from s_2 are cut-off. After that DFS starts on the rest of T_2 and starts bDFS on the incomplete vertices. Here some new graphs G' will be explored and we build spanning trees T' for them. Some trees in $\mathcal{T}$ get fully explored. T_w und T' are added to $\mathcal{T}$, the tree T_2 is deleted.

In the following let $\mathrm{d}_{G'}(v, w)$ denote the distance between vertices v and w in the subgraph or tree G'. $G^* = (V^*, E^*)$ denotes the currently known part of G.

The algorithm makes us of the following subdivision of vertices:

non-explored a vertex, which was never been visited before.

incomplete a vertex already visited before but some of the adjacent edges are still non-explored.

Algorithm 1.10 CFS

CFS(s**,** r**,** α **)**

$\mathcal{T} := \{\{s\}\}$.
repeat
  $T_i :=$ closest subtree of $\mathcal{T}$ to s in G^*.
  $s_i :=$ vertex of T_i closest to s n G^*.
  $(T_i, \mathcal{T}_i) :=$ prune(T_i, s_i, $\frac{\alpha r}{4}$, $\frac{\alpha r}{2}$).
  $\mathcal{T} := \mathcal{T} \setminus \{T_i\} \cup \mathcal{T}_i$.
  explore($\mathcal{T}, T_i, s_i, (1+\alpha)r$).
  Delete fully explored trees from $\mathcal{T}$.
  Merge the trees of $\mathcal{T}$ with common vertices.
  Define a root vertex closest to s in G^*.
until $\mathcal{T} = \emptyset$

prune(T**,** v**,** *minDist*, *minDepth* **)**

$v :=$ Wurzel von T.
$\mathcal{T}_i := \emptyset$.
for all $w \in T$ with $\mathrm{d}_T(v,w) =$ *minDist* **do**
  $T_w :=$ subtree of T with root w.
  if maximale Distanz between v and a vertex in $T_w >$ *minDepth* **then**
    // Cut-Off T_w from T ab:
    $T := T \setminus T_w$.
    $\mathcal{T}_i := \mathcal{T}_i \cup \{T_w\}$.
  end if
end for
RETURN $(T, \mathcal{T}_i)$

explore($\mathcal{T}$**,** T**,** s_i**,** ℓ **)**

Move from s to s_i along shortest path in G^*.
Explore T by DFS. If incomplete vertex occurs, do:
  $\ell' :=$ remaining tether length.
  bDFS(v, ℓ').
  $E' :=$ set of newly explored edges.
  $V' :=$ set of vertices of E'.
  Calculate spanning tree T' for $G' = (V', E')$.
  Define root vertex of T' closest to s in G^*
  $\mathcal{T} := \mathcal{T} \cup \{T'\}$.
Move back from s_i to s.

explored a vertex, that was visited and all adjacent edges have been explored.

Additionally, for the bDFS walk we mark the edges as 'non-explored' or 'explored'.

Lemma 1.23 *The following properties hold during the execution of the CFS–Algorithm:*
(i) Any incomplete vertex belongs to a tree in $\mathcal{T}$.
(ii) Until $G^* \neq G$, there is always an incomplete vertex $v \in V^*$ so that $d_{G^*}(s,v) \leq r$.
(iii) For any chosen root vertex s_i: $d_{G^*}(s,s_i) \leq r$.
(iv) After pruning T_i is fully explored by DFS. All trees $T \in \mathcal{T}$ have size $|T| \geq \frac{\alpha r}{4}$.
(v) All trees $T \in \mathcal{T}$ are disjoint (w.r.t. edges).

Proof.
(i) Follows directly from the construction of the trees by bDFS and Pruning. No incomplete vertex is missing.
(ii) Assume that for all $v \in V^*$ we have $\mathrm{d}_{G^*}(s,v) > r$ and let v be an incomplete vertex of V^*. In G there is a shortest path $P(s,v)$ from s to v with length $\leq r$. Along $P(s,v)$ there is a first vertex w that does not belong to G^*. Thus its predecessor w' along $P(s,v)$ belongs to V^* and is incomplete. We have $\mathrm{d}_{G^*}(s,w') \leq r$.
(iii) Follows from (ii), the root of a corresponding tree T is always the vertex of T closest to s.
(iv) We show the property by successively considering the upcoming trees. Or by induction on the number of pruning steps. In the beginning the algorithm starts with bDFS at the root s. Either, the graph will be fully explored and we are done, or bDFS have exhausted the tether of length $(1+\alpha)r$ and have visited more than $(1+\alpha)r$ edges. The single spanning tree T has size $|T| \geq (1+\alpha)r > \frac{\alpha r}{4}$.
Let us assume that the condition holds for the trees inside $\mathcal{T}$ and the next pruning step happens. Now by the next iteration we are choosing tree T_i with root s_i closest to s among all trees in $\mathcal{T}$. After that we prune T_i. The rest of T_i has still size $|T_i| \geq \frac{\alpha r}{4}$ since we cut off subtrees T_w with distance $\geq \frac{\alpha r}{2}$ away from s_i. For a corresponding subtree T_w we conclude $|T_w| \geq \frac{\alpha r}{2} - \frac{\alpha r}{4} = \frac{\alpha r}{4}$ since there is a vertex inside T_w that is at least distance $\frac{\alpha r}{2}$ away from s. Now consider the remaining DFS/bDFS combination on (the rest of) T_i. The distance from s to s_i is at most r.
Any incomplete vertex in the current T_i has at most distance $\frac{\alpha r}{2}$ from s_i otherwise this vertex would be part of a tree T_w that has to be considered in the pruning step. This means that at any incomplete vertex there is a rest tether of length $\frac{\alpha r}{2}$ which can be used for the bDFS part. If the exploration results in another spanning tree T' with incomplete vertices, this tree has size at least $\frac{\alpha r}{2}$.
Finally fully explored trees are deleted from $\mathcal{T}$ which is not critical. Additionally, some other trees might be merged and still have incomplete vertices. These trees only grow.

□

Finally, we show:

Theorem 1.24 *(Duncan, Kobourov, Kumar, 2001/2006)*
The CFS–Algorithm for the constrained graph-exploration of an unknown graph with known depth is $(4+\frac{8}{\alpha})$–competitive. *[DKK06, DKK01]*

Proof. We split the cost for any appearing subtree T_R. Let $K_1(T_R)$ denote the cost for moving from s to s_i in G^*. Let $K_2(T_R)$ denote the cost of DFS for T_R and let $K_3(T_R)$ denote the cost for the bDFS exploration done for the incomplete vertices starting at T_R. The trees are edge disjoint.

The total cost is a sum of the cost for any T_R. We have

$\sum_{T_R} K_3(T_R) \leq 2 \cdot |E|$, since bDFS only visits non-explored edges (twice).

$\sum_{T_R} K_2(T_R) = \sum_{T_R} 2 \cdot |T_R| \leq 2 \cdot |E|$, the cost for all DFS walks.

For $K_1(T_R)$ we have $K_1(T_R) = 2 \cdot \mathrm{d}_{G^*}(s,s_i) \leq 2r$. The complexity of any T_R is at least $\frac{\alpha r}{4}$ which gives $|T_R| \geq \frac{\alpha r}{4}$ for the number of edges. We conclude $r \leq \frac{4|T_R|}{\alpha}$ and

$$\sum_{T_R} K_1(T_R) \leq \sum_{T_R} 2r \leq \frac{8}{\alpha} \sum_{T_R} |T_R| \leq \frac{8}{\alpha}|E|$$

Altogether, the algorithm makes $(4+\frac{8}{\alpha})|E|$ step whereas any optimal algorithm visits at least any edge once. □

In general we assume that α is a small constant with $0 < \alpha < 1$. The above proof works for any $\alpha > 0$. The cost of the algorithm for known depth r are within $O(|E|/\alpha)$. More precisely we can show that actually $O(|E|+|V|/\alpha)$ steps are made. For this we have a closer look at the cost. bDFS work on the edges only. The DFS walk work on trees where the number of vertices is the same as the number of edges. Some of these vertices appear in two trees, so by a factor of 2 we are on the save side. The movements from s to s_i are analysed over the size of spanning trees, where vertices and edges are also the same.

The cost $K_1(T_R)$ and $K_2(T_R)$ sum up to $\left(2+\frac{8}{\alpha}\right)2|V|$.

Altogether there is an $\Theta(|E|+|V|/\alpha)$ algorithm for the exploration of arbitrary graphs.

Corollary 1.25 *The CFS–Algorithm for the constrained graph-exploration of an unkown graph with known depth has optimal exploration cost* $\Theta(|E|+|V|/\alpha)$.

Now we have some possibilities for extensions. First, we assume that the depth of the graph is unknown in the beginning. Next we would like to consider weighted edges.

1.5.1 Restricted graph-exploration with unknown depth

Let is now assume that the radius, say R, of the unknown graph G is not known. From a practical point of view, spending some cable is costly and we would like to extend the tether only if it is necessary. A first simple idea is that we guess the depth, say r, and successively double its length until the algorithm finally explores the whole graph. Obviously, the repeated application of the CFS-algorithm runs in $O(\log r|E|)$ step. As shown above we can also refine the analysis of this approach. For any bDFS step we make use of the already visited edges and directly *jump* to incomplete vertices (now with larger tether length). Therefore the bDFS steps are still subsumed by $2|E|$ steps. But we still have to take the movements to the roots of the trees into account as well as the DFS movements on the new subtrees. Therefore we have the following result.

Corollary 1.26 *Applying the CFS–Algorithmus by successively doubling the current depth r gives an algorithm that explores an unknown graph G with unknown depth R with* $\Theta(|E|+(\log R)|V|/\alpha)$ *steps.*

We will now show that we can get rid of the log-factor by successively adjusting r appropriately. We only exchange two calls in the main procedure. In principle, instead of the known value r we successively make use of $r := d_{G^*}(s,s_i)$, which is the smallest distance from s to one of the roots of the trees in $\mathcal{T}$.

More precisely, we exchange prune(T_i, s_i, $\frac{\alpha r}{4}$, $\frac{\alpha r}{2}$) wird by prune(T_i, s_i, $\frac{\alpha d_{G^*}(s,s_i)}{4}$, $\frac{9\alpha d_{G^*}(s,s_i)}{16}$) and explore($\mathcal{T}$, T_i, s_i, $(1+\alpha)r$) by explore($\mathcal{T}$, T_i, s_i, $(1+\alpha)d_{G^*}(s,s_i)$). This means that the pruning-step is done with the values $\frac{\alpha d_{G^*}(s,s_i)}{4}$ and $\frac{9\alpha d_{G^*}(s,s_i)}{16}$ and the eplore-step is done with tether length $(1+\alpha)d_{G^*}(s,s_i)$.

In the beginning we have $d_{G^*}(s,s_i) = 0$, therefore we make use of some fixed constant c in the beginning and use $r := max(d_{G^*}(s,s_i),c)$. Let $d_{G^*}(s,T)$ denote the shortest distance from s to some vertex in T inside G^*.

Lemma 1.27 *For the CFS–Algorithmus with unknown depth R we have the following properties:*

(i) Any incomplete vertex belongs to a tree in $\mathcal{T}$.

(ii) There is always an incomplete vertex $v \in V^*$ *with* $\mathrm{d}_{G^*}(s,v) \le r$, *until* $G^* \neq G$.

(iii) For the closest root s_i *we have:* $\mathrm{d}_{G^*}(s,s_i) \le r$.

(iv) For all trees $T \in \mathcal{T}$ *we have* $|T| \ge \frac{\max(d_{G^*}(s,T),c)\alpha}{4}$. *After pruning the remaining tree will be fully explored by DFS.*

(v) All trees ever considered in $\mathcal{T}$ *are (edge) disjoint.*

Proof. For the proof of (i),(ii),(iii) and (v) we apply the same arguments as in the proof of Lemma 1.23. It remains to show that (iv) holds. The main difference is that the size of a tree T is directly correlated to the distance from s to T, this is different from the previous argumentation.

Let us first show that the remaining tree T_i (after pruning) will be fully explored by DFS. For any vertex v in T_i we have $d_{T_i}(s_i,v) \leq \frac{9d_{G^*}(s,s_i)\alpha}{16}$, otherwise v has been cut of by pruning. Thus we have

$$(1+\alpha)d_{G^*}(s,s_i) - d_{G^*}(s,s_i) - d_{T_i}(s_i,v) \geq \frac{7d_{G^*}(s,s_i)\alpha}{16},$$

which shows that the tether is long enough T_i will be fully explored by DFS.

By induction over the number of pruning steps we will finally show: $\forall T \in \mathcal{T} : |T| \geq \frac{\max(d_{G^*}(s,T),c)\alpha}{4}$.

In the beginning we apply bDFS from the start with tether length c. Either we explore the whole graph or we have $|T| \geq (1+\alpha)c > \frac{\alpha c}{4}$ for the resulting spanning tree T. For simplicity we assume $d_{G^*}(s,T_i) > c$ from now on.

We would like to show that for any tree T_w, resulting from the pruning of some T_i, we have $|T_w| \geq \frac{d_{G^*}(s,T_w)\alpha}{4}$. Also the remaining tree T_i has this property.

For the remaining tree T_i (after pruning), we conclude $d_{G^*}(s,T_i) = d_{G^*}(s,s_i)$ and pruning guarantees $|T| \geq \frac{d_{G^*}(s,T)\alpha}{4}$. For a tree T_w pruned from T_i we have: $|T_w| \geq \frac{9d_{G^*}(s,s_i)\alpha}{16} - \frac{d_{G^*}(s,s_i)\alpha}{4} = 5\frac{d_{G^*}(s,s_i)\alpha}{16}$ by the pruning values. Additionally, we have $d_{G^*}(s,T_w) \leq d_{G^*}(s,s_i) + d_{G^*}(s_i,w) = (1+\frac{\alpha}{4})d_{G^*}(s,s_i)$, since the root w of T_w is exactly $\frac{\alpha d_{G^*}(s,s_i)}{4}$ steps away from s. Für $0 < \alpha < 1$ we conclude: $d_{G^*}(s,T_w) < \frac{5d_{G^*}(s,s_i)}{4}$ and together with the above inequality we have $|T_w| > \frac{d_{G^*}(s,T_w)\alpha}{4}$.

Finally, we have to analyse the emerging spanning trees T_v, which will be constructed from the bDFS steps starting during the DFS walk in T_i. Such a tree T_v *starts* at some incomplete vertex v in T_i. We have $d_{G^*}(s_i,v) \leq \frac{9\alpha d_{G^*}(s,s_i)}{16}$, otherwise v would have been pruned and could not be a leaf of the rest of T_i any more. Thus we have $d_{G^*}(s,T_v) \leq d_{G^*}(s,s_i) + d_{G^*}(s_i,v) < \frac{25d_{G^*}(s,s_i)}{16}$ or $d_{G^*}(s,s_i) > \frac{16d_{G^*}(s,T')}{25}$. If T_v is fully explored, we are done, since the tree will be deleted. Assume that T_v still has incomplete vertices. As mentioned above we have $d_T(s_i,v) \leq \frac{9\alpha d_{G^*}(s,s_i)}{16}$. Starting from v there was a remaining tether length of $\frac{7\alpha d_{G^*}(s,s_i)}{16}$ for the construction of the incomplete T_v, which gives $|T_v| \geq \frac{7\alpha d_{G^*}(s,s_i)}{16}$. Application of $d_{G^*}(s,s_i) > \frac{16d_{G^*}(s,T_v)}{25}$ gives $|T_v| > \frac{7\alpha d_{G^*}(s,T_v)}{25} > \frac{d_{G^*}(s,T_v)\alpha}{4}$. Either we have explored everything behind v or the spanning tree T_v has size $|T_v| > \frac{d_{G^*}(s,T_v)\alpha}{4}$.

We have considered any emerging $T \in \mathcal{T}$! □

Theorem 1.28 *(Duncan, Kobourov, Kumar, 2001/2006)*
Applying the CFS–Algorithm with the adjustments above results in a correct restricted graph-exploration of an unknown graph with unknown depth. The algorithm is $(4+\frac{8}{\alpha})$*–competitive. [DKK06, DKK01]*

Proof. We apply the same analysis as in the proof of Theorem 1.24. For the analysis of the movements from s to the roots of the trees we make use of the correlation $|T_R| > \frac{d_{G^*}(s,T_R)\alpha}{4}$. □

For the number of steps we can also refine the analysis, analogously.

Corollary 1.29 *The above CFS–Algorithm for the restricted exploration of an unknown graph with unknown depth requires* $\Theta(|E|+|V|/\alpha)$ *exploration steps, which is optimal.*

Finally, we would like to argue that the usage of a look-ahead of αr is necessary for attaining linear optimal exploration cost (i.e., in comparison to $|E|$ and $|V|$. This can be shown for the accumulator variant as follows. First, it is clear that an accumulator of size $2r$ is not sufficient for exploring all edges. The graph in Figure 1.32 has depth 6, but exploring all edges requires an accumulator of size 13.

This means that an accumulator size $2r+1$ is necessary. We show that an accumulator of size $2r+d$ for constant d is not sufficient in the sense of performing no more than $C \cdot |E|$ exploration steps.

Lemma 1.30 *For the accumulator variant with accumulator size* $2r+d$ *for constant d, there are examples do that any algorithm attains at least* $\Omega\left(|E|^{\frac{3}{2}}\right)$ *exploration steps.*

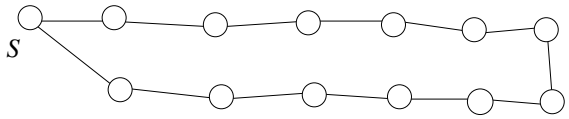


Figure 1.32: A graph of depth $r = 6$ that cannot be explored by an accumulator of size $2r$.

Proof. We consider the following example as given in Figure 1.33. Starting from s there is a path of length $\frac{n}{2}$ that visits a clique of size $\frac{n}{2}+1$. Moving forth and back along the path requires n steps, the depth of the graph is $\frac{n}{2}+1$. Exploration with accumulator size $n+2+d$ means that we have to visit the clique $\Omega\left(\frac{n^2}{d}\right)$ times since the clique has $\Omega\left(n^2\right)$ edges. This gives $\Omega\left(\frac{n^2}{d}\cdot n\right) = \Omega\left(n^3\right)$ exploration steps. The statement follows from $|E| \in \Theta(n^2)$. □

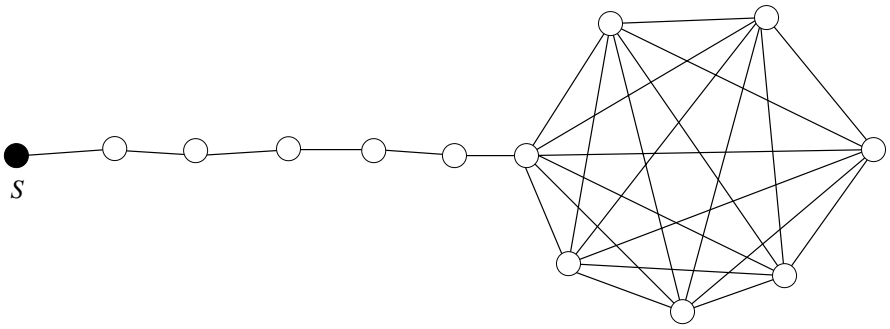


Figure 1.33: A graph with $n+1 = 13$ vertices. A path of length $\frac{n}{2}$ visits a clique of size $\frac{n}{2}+1$. Any accumulator strategy with accumulator size $n+2+d$ requires $\Omega(n^3)$ steps.

With a similar argument we conclude that an sub-linear extension of the accumulator, i.e., size $2r+o(r)$, is not sufficient for attaining a linear cost strategy. Let us briefly repaet the small-o notation. For real valued functions or series f and g we define $f \in o(g)$, if and only if $\lim_{r\to\infty} \frac{f(r)}{g(r)} \to 0$ holds. Therefore we conclude $r \in o(r^2)$, $c \in o(r)$ for any constant c and also $\frac{1}{r} \in o(1)$. By the above arguments and example we can show that $\Omega\left(\frac{n^3}{f(n)}\right)$ exploration steps are necessary for an accumulator of size $n+2+f(n)$. For $f(n) = n^{1-\varepsilon}$ (this means $f \in o(n)$) we have to perform $\Omega(|E|^{1+\varepsilon})$ exploration steps.

Note, that for the tether variant up to our knowledge there is no such statement that a tether of length $r+o(r)$ is necessary for attaining $O(|E|)$ exploration cost.

We have shown that we can explore any graph (online and offline) with at most $\Theta(|V|+|E|)$ exploration steps. These are the pure cost for the motion of the agent. In the literature this is also denoted as the *mechanical cost*; see also [DJMW91]. Besides, there are also some *computational cost*, for the planning and preparation of the strategy.

For example the computational cost of the CFS-Algorithm have to be analysed for the following tasks:

- Build the spanning trees
- Update the shortest paths to the trees of $\mathcal{T}$
- Merge the trees
- Detect fully explored trees
- Prune a tree
- Maintain the list $\mathcal{T}$
- Apply DFS/bDFS

For unit-lenght edges some of the above tasks can be done very efficiently. The overall approach can be easily extended to weighted graphs (positive edge weights).

Exercise 10 *Analyse the computational cost for the CFS-Algorithm in $O-$notation for $|E|$ and/or $|V|$.*

Exercise 11 *Show that the CFS-Algorithm approach also works for graphs with positive edges weights. How do we have to adjust the CFS-Algorithm?*

1.5.2 Mapping of an unknown graph

Finally, in this section we would like to show the influence of different capabilities of the agent. Up to now we assumed that an already visited vertex or edge will be recognized at the next visit. This means that we have marked any visited edge and vertex.

Let us now assume that the agent cannot mark parts of the environment. We do not have any *landmarks*. We still assume that we have enough storage for constructing the sub-graph detected so far.

The following model is taken from Dudek et al.; see[DJMW91]. The agent has no orientation and no compass. At any vertex the outgoing edges are presented in the same order. This order need not represent a planar embedding. If the agents visits the vertex from different incoming edges, the order will be consistent. This means that there is a fixed cyclic order, the relative presentation of the order stems from the edge where the agent currently comes from. Figure 1.34 shows an example of a relative order. By this order, the agent knows where he was coming from and can also return to this vertex. Since the storage is not limited, it is possible to remember a return path. Let us for example assume that the agent visits vertex v_2 by edge e_1 and then visits the second edge e_3 in ccw-order from e_1. If the agent moves back along e_3 to v_2, it already knows that it was recently coming from the first edge in ccw order, which is e_1. The agent can make use of this return path. If the agent visits a vertex in a forward step, it has no idea which of the vertices the visited vertex actually is.

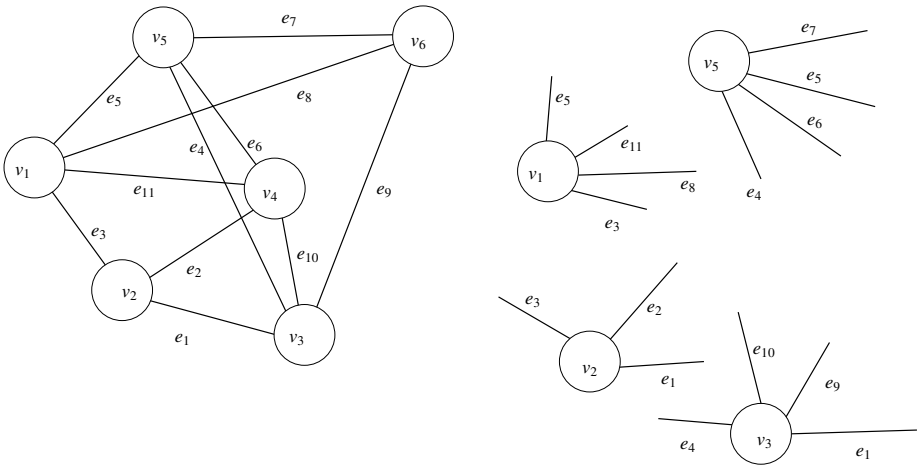


Figure 1.34: A graph $G = (E,V,S)$. A local cyclic order need not correspond to the embedding of the graph.

Is it possible to build a map of the graph and to locate oneself inside the graph? The offline input is a triple $G = (V,E,S)$, where by S for any vertex the cyclic local order of the edges is given.

First, it is easy to see that without further capabilities, one can not fully detect a given graph. Figure 1.35 shows two different regular graphs of fixed degree 3. For an agent the information on any vertex is exactly the same. It is not possible to distinguish between the two variants. At least one marker is necessary.

Corollary 1.31 *Let $G = (E,V,S)$ be a graph with local cyclic edge order- Without a marker an online agent cannot build a correct map of the graph.*

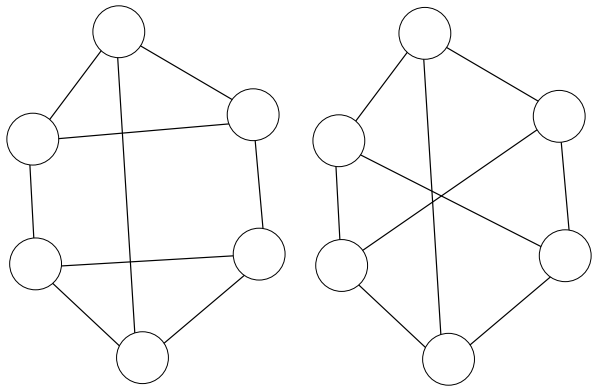


Figure 1.35: Two different regular graphs of degree 3, an agent cannot distinguish them without a marker.

Exercise 12 *Give a formal argument that the graphs in Figure 1.35 are different. Which class of graphs can be correctly detected by an online agent without a marker?*

A single marker (or pebble) is sufficient as shown by Dudek et al. [DJMW91]. We describe the corresponding *Marker*-Algorithm. The algorithm maintains the current known graph G^* and a list L of non-determined (seen but not correctly detected) edges. In the beginning the starting vertex is known and its outgoing edges belong to L. They are given in the cyclic order.

In the main step, the algorithm choose an edge e of L starting at a detected vertex b and moves to a vertex u along the edges $e = (b,u)$. Now the agent sets the marker on u, moves back to b along e and searches for the pebble in G^*.

Case 1: The pebble was not found in G^*. In this case we add the edge $e = (b,u)$ to G^* w.r.t. the cyclic order. All outgoing edges of u different from e will be inserted into the list L of non-determined edges.

Case 2: The marker has been found at some vertex $v \in G^*$. If there is more than one non-determined outgoing edges at $v = u$, we cannot precisely detect e. Therefore we take the marker, move back to b, place the pebble there, move back to v again and successively check the non-determined edges. Finally, we will detect the edge e and add it to G^* by the local order.

The above algorithm is simple and correct. By construction in any step an additional edge will be correctly detected. The number of exploration steps is restricted by $O(|E| \times |V|)$ and the same holds for the computational cost. We assume that the graph is not a multigraph and has no loop edges (v,v). Besides, we assumed that any edge has unit-length.

Theorem 1.32 *(Dudek, Jenkin, Milios, Wilkes, 1991)*
Let $G = (E,V,S)$ be a graph with given cyclic local order of the edges. By the use of one marker it is possible to fully detect the structure of the graph by online navigation with $O(|E| \times |V|)$ exploration steps and also overall $O(|E| \times |V|)$ computational cost.

Proof.

Let $G^* = (V^*,E^*,S^*)$ be the current graph during the execution of the Marker-Algorithm. Setting the marker has cost $O(1)$, searching for the marker in G^* can be done by DFS by $O(|V^*|)$ steps. Moving back and force along a path can be done in $O(|V^*|)$ steps as well. The traversal cost are considered for any edge, which gives $O(|E| \times |V|)$ steps in total.

For unit-edge length the computational cost are precisely the same for any edges we have to compute the shortest paths between two vertices. The effort is bounded by $O(|V^*|)$. This gives $O(|E| \times |V|)$. □

Exercise 13 *Explain why the cyclic order of the edges is necessary for the above Marker-Algorithm. Where is it used during the execution of the algorithm?*

Exercise 14 *Analyse the mechanical and the computational cost of the marker algorithm for graphs with positive edge weights.*

1.6 Online TSP for planar graphs

In this section we consider the task of visiting all vertices in an unknown (planar) graph with edge weights. We have seen before that visiting all edges and vertices can be done within a constant competitive ratio. Visiting the vertices only in a general weighted graph is a much more complicated task.

Here we consider the model of a traveling salesperson that arrives at a certain city (vertex) and sees the traffic signs with distance information (edge weights) to the neighbouring cities (neighbouring vertices). Note that the edges need not be line segments! The vertices are thus identified by the names. The task is to visit all vertices of the unknown network. If the network is given, the task to visit all vertices is simply the Offline TSP problem which in known to be an NP-hard problem.

Exercise 15 *Give an example that in the above model, for general graphs there is no constant competitive algorithm for solving the Online TSP problem.*

The online problem is modelled by an unknown graph $G = (V,E)$. Note that for a planar graph the edges need not be line segments. During the execution of the algorithm we make use of the following subdivision of vertices of G:

visited A vertex, that has was been visited.

boundary An unvisited vertex adjacent to a visited vertex.

unknown Neither boundary vertex nor visited vertex.

Additionally we make use of the following notion for the edges of G:

explored Both endpoints have been visited.

boundary One endpoint is visited and one end point is a boundary vertex.

unknown None of the endpoints has been visited.

The main idea again is to make use of a DFS walk for the vertices but at some points we will change the preference rule, if the corresponding path to the vertex is long in comparison to other movements. We will now develop an algorithm ShortCut that results in a 16-approximation against the optimal offline path. The main idea of the algorithm is the usage of *shortcuts*. For two vertices v and w of G by $|vw|$ we denote the length of the edge (v,w) and by $d_G(v,w)$ we deonte the shortest path between v and w in G. In the following note that the parameter $\delta \geq 0$ will be specified later on in the analysis.

Exercise 16 *Give an example that in the above model, a pure application of DFS for the vertices does not result in a constant competitive ratio.*

Definition 1.33 A boundary edge (x,y) *blocks* a boundary edge (v,w) if

1. $|xy| < |vw|$ and
2. $d_G(v,x) + |xy| < (1+\delta)|vw|$

holds. A boundary edge (v,w) is a *shortcut*, if no other boundary edge blocks (v,w).

Intuitively blocking means that the currently blocked edge (v,w) might be worth visited after (x,y) was visited, because the traversal back from y over x toward v is bounded by $(1+\delta)|vw|$.

We now explain the algorithm Algorithm 1.11. For each vertex v the strategy maintains a list $Incident(v)$ of edges incident to v for the DFS walk. For each boudary edge e there is a list $Block(e)$ of boundary edges that block e.

The algorithm adapts DFS. Assume that we are at vertex v and would like to proceed with edge (v,w). If (v,w) is not blocked, we will traverse the edge and visit w. In the analysis later on, if we move along (v,w), the edge (v,w) is *charged* for the cost. If (v,w) is blocked, we will not move along the edge.

Now assume that the searcher visits a boundary edge (x,y) such that (x,y) is explored now and another boundary edge (v,w) gets unblocked. Now we construct a *jump edge* $j(y,w)$ from y to w, obviously we have $d_G(y,w) < (2+\delta)|vw|$ which can be considered as the cost of this artificial edge. This edge can be used as any other edge in the DFS preference of the vertices. If $j(v,w)$ is used at some point in time for visiting w, the edge (v,w) will be *charged* for this movement.

In the DFS preference we assume that we move along an edge from vertex x to some vertex y. We first build potential jump edges and update the incidence information. Then we look at all boundary vertices z incident to y. If there is a shortcut (y,z) or a jump edge (y,z) was constructed before we traverse such an edge recursively with the same procedure. If there are no longer boundary vertices around y, we move back to x along the currently known best path.

Algorithm 1.11 Shortcut

Shortcut(x, y, G):

```
Traveling from x, we have visited y for the first time
for all Boundary edges (v,w) do
  if Visiting y caused Block((v,w)) to become empty then
    Build jump edge j(y,w), append jump edge on Incident(y) and Incident(w)
  end if
end for
for all Edges (y,z) ∈ Incident(y) do
  if z is boundary vertex and (y,z) shortcut then
    Traverse the egde (y,z)
    Shortcut(y, z, G )
  else if z is boundary vertex and (y,z) is jump edge then
    Move to z along the shortest known path
    Shortcut(y, z, VARG )
  end if
end for
Return to x along the shortest known path
```

We simply start the above algorithm by a call Shortcut(s, s, G) for the starting vertex s.

Only vertices adjacent to s can be considered in the beginning and all known edges are boundary edges, apart from loops (s,s) which is actually not forbidden. Finally the traveling salesman return to the start s. We have the following properties:

Lemma 1.34

1. *By application of the Shortcut Algorithm 1.11 any vertex of a graph G will be visited.*
2. *If during execution of Algorithm 1.11 at some point a boundary edge (x,y) blocks a boundary edge (v,w), then (x,y) blocks (v,w) until either y or w is visited.*
3. *If after traversing a boundary edge (x,y) another boundary edge (v,w) became a shortcut and a jump edge $j(y,w)$ was added, we have $d_G(y,w) < (2+\delta)|vw|$.*

Proof. Property 3. was already shown above and stems from the second condition of Definition 1.33. Assume that 1. does not hold and that after termination not all vertices of G have been visited. So there are still boundary edges. Let (x,y) be a boundary edge with smallest value $|xy|$, vertex y ist still unvisited. If the traversal of (x,y) was not delayed by blocking after the visit of x, we are done. Otherwise, (x,y) did become and remain a shortcut (smallest edge) at some visit of a vertex z. In this moment from z a jump edge (z,y) was build and traversed toward y afterwards. Property 2. holds since until neither y nor w will be visited, the edges remain boundary edges and the blocking condition always hold. □

Now we can prove our main statement!

Theorem 1.35 *(Kalayanasundaram, Pruhs, 1994)*
For planar graphs the above Shortcut algorithm is 16-competitive.

Proof. Let P be the set of all *charged* edges as interpreted in the sense above. Note that the edge (x,y) can be charged if it becomes a shortcut edge for visiting y, although the edge (x,y) need not be traversed, since a jump edge was constructed. The total length of the tour can only be at most $2(2+\delta)|P|$, where the factor 2 stems from the DFS traversal effect of visiting at most any traversed edge twice. The factor $(2+\delta)$ covers direct visits a shortcut and the usage of jump edges; see point 2. of Lemma 1.34.

Consider the Minimum Spanning Tree (MST) of G. Additionally, we assume that the MST contains the minimum number of edges from P, if there is more than one MST. (If all edge weights are different, this condition can obviously be omitted.) Obviously, we have $|MST| \leq |OPT|$ because OPT is also a spanning tree for the vertices of G. We now show that $|P| \leq (1+2/\delta)|MST|$ holds This gives $2(2+\delta)|P| \leq 2(2+\delta)(1+2/\delta)|MST| \leq 2(2+\delta)(1+2/\delta)|OPT|$ and minimizing over δ gives $\delta = 2$.

We consider the planar embedding of $P \cup MST$ which contains the MST and some additional edges stemming from P, which are denoted as *chords*. Let C denote the set of all chords. The goal is that the chords still has to be analysed by portions of the MST. Also consider a round trip R around the MST by intuitively blowing up the MST so that R becomes a thin polygon around the MST with perimeter length $|R| = 2|MST|$. We consider a planar embedding of R and C such that R builds a *ring* and the chords are placed around the ring as shown in Figure 1.36. Now we further split R at some vertex such that we do not break the chords; see Figure 1.37. For convenience the path obtained by the ring will also be denoted as R.

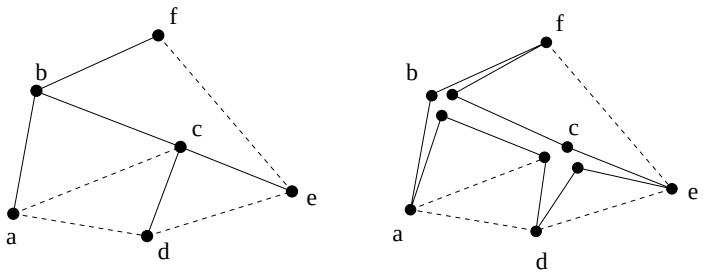


Figure 1.36: The MST can be extended to a round trip R and the chords are adjusted correspondingly.

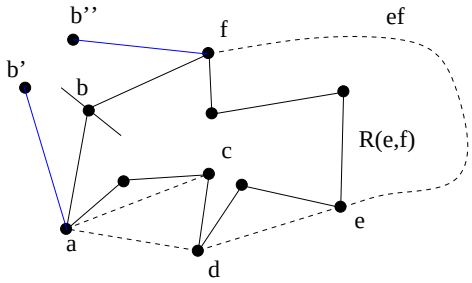


Figure 1.37: Blowing up and splitting of the ring R. We compare a path $R(e,f)$ to the chord (e,f). The chord (a,c) lies inside chord (a,d).

In the following for an edge (x,y) we assume that x was visited before y in the above shortcut algorithm. W.r.t. the path R a chord (x,y) lies *inside* chord (v,w) if the subpath $R(v,w)$ from v to w along R encounters the vertices x and y. For example in Figure 1.37 the chord (a,c) lies inside chord (a,d).

To prove $|P| \leq (1+2/\delta)|MST|$ it obviously suffices to show that $|C| \leq \frac{|R|}{\delta}$ holds. We consider the chords recursively from inside to outside. Finally, we would recursively replace $R(x,y)$ by (x,y). A chord (x,y) is called *good*, if $|R(x,y)| \geq (1+\delta)|xy|$ holds. We show that any chord is good.

Suppose for a while that $|R(x,y)| < (1+\delta)|xy|$. An edge (v,w) of $R(x,y)$ is called *large* if $|vw| \geq |xy|$ holds. We show that there is at least one large edge in $R(x,y)$. The edge (x,y) has been charged at some point t in time. At this moment t the vertex y has been visited. This means that at time t there is at least one boundary edge along the path $R(x,y)$ otherwise y was already visited. Let (v,w) be the first such boundary edge while traversing from x to y along $R(x,y)$. We must have $d_G(x,v)+|vw| < (1+\delta)|xy|$ at this moment since $|R(x,y)| < (1+\delta)|xy|$ was assumed. If $|xy| > |vw|$ holds, the edge (x,y) was still blocked by (v,w), a contradiction to the assumption that (x,y) has been charged, thus $|xy| \geq |vw|$ holds and (v,w) is large.

Now we show that under the assumption $|R(x,y)| < (1+\delta)|xy|$ there is at least one large edge that was not charged and thus belongs purely to the MST. Among the edges of $R(x,y)$ consider a large edge (v,w) that was charged last at some point t in time. There must be another boundary edge on the path $R(x,y)+(x,y)-(v,w)$ from v to w at this moment. Let (a,b) be the first one while moving from v to w. We still assume $|R(x,y)| < (1+\delta)|xy|$ and have $|xy| \geq |vw|$. We also have $d_G(v,a)+|ab| \leq |R(x,y)|+|xy|-|vw|$ (note that b might identical to w). We conclude $d_G(v,a)+|ab| < (1+\delta)|vw|$. Since (v,w) is not blocked, we conclude $|ab| \geq |vw|$, and (a,b) is also large.

Altogether under the assumption $|R(x,y)| < (1+\delta)|xy|$ there is a non charged large edge (a,b) which belongs to the MST, only. So the spanning tree $MST-(a,b)+(x,y)$ either has smaller cost than the MST for $|ab| > |xy|$ or for $|ab| = |xy|$ the spanning tree $MST-(a,b)+(x,y)$ induces less chords. A contradiction.

Now for $|R(x,y)| \geq (1+\delta)|xy|$ we apply the argument recursively. For an inner chord we first replace $R(x,y)$ by (x,y) and obtain R'. We consider $|R|$ as a budget. From $|R|$ we subtract $\delta|xy|$ which is no longer required. For R' we can apply the same arguments for the next inner chord (x',y'), thus obtaining R'' by replacing $R'(x',y')$ by (x',y'). From the budget $|R|$ we subtract $\delta|xy|+\delta|x'y'|$. For this recursive process the length of the original ring R has to be large enough such that $|R|-\delta|C| \geq 0$ holds and finally we conclude $|C| \leq \frac{|R|}{\delta}$. □

Exercise 17 *Why can we apply the above analysis only for planar graphs?*

Exercise 18 *Minimize* $2(2+\delta)(1+2/\delta)$ *over* δ *analytically.*

Finally we consider the computation time for the Shortcut Algorithm.

Lemma 1.36 *(Kalayanasundaram, Pruhs, 1994)*
For a planar graph with n vertices the Shortcut algorithm runs in time $O(n^2 \log n)$.

Proof. First note that for the planar graph we have also $O(n)$ edges and single source Dijkstra runs in $O(n \log n)$.

For the blocking process, for any boundary edge (v,w) we maintain a list $BlockedBy(e)$ of the edges which are blocked by $e=(v,w)$. Additionally $Block(e)$ is the list of edges that block $e=(v,w)$. We make use of cross-pointers between the two lists. If a boundary edge (x,y) is encountered for the first time we initialize $BlockedBy((x,y))$ and $Block((x,y))$ by running single source Dijkstra from x thus finding all (v,w) with $|xy| < |vw|$ and $d_G(v,x)+|xy| < (1+\delta)|vw|$ and also all (a,b) with $|ab| < |xy|$ and $d_G(x,a)+|ab| < (1+\delta)|xy|$. This gives running time $O(n^2 \log n)$ in total.

For traversing a jump edge we again have to apply Dijkstra. For at most $O(n)$ jump edges we have $O(n^2 \log n)$.

Finally, an explored edge (x,y) no longer blocks edges, therefore for any $(v,w) \in BlockedBy((x,y))$ we remove (x,y) in $Block((v,w))$ by the cross pointers in constant time. If necessary we build jump edges in constant time. Any edge is inserted and deleted from *any* list *BlockedBy* and *Block* at most once. We have $O(n)$ lists in total. So the total running time is in $O(n^2)$. Altogether, the running time is bounded by $O(n^2 \log n)$. □

Chapter 2

Polygonal enviroments

In this chapter we turn over to planar environments modelled by (a set of) simple polygons. We assume that a set of simple polygons P_i for $i = 1, \ldots, k$ is given. Two polygons do neither intersect nor touch each other. The number of polygons is finite in the sense that any circle of fixed radius contains only finitely many *obstacles* P_i.

In the following sections an agent tries to escape from a labyrinth (modelled by polygons) or tries to find a target t in a polygonal environment. We assume that the agent is point-shaped and thus consider curves in the plane. From a practical point of view one might think that a physical robot somehow *follows* the corresponding curve without actually running precisely on it. Additionally, in the followning configurations, the agent has only a limited storage.

We make use of the following conventions. If the coordinates of the target are given, the task of the agent is denoted as "Navigation". On the other hand, if the coordinates are not known the task of the agent is "Searching". We will consider different sensor models.

Some of the following algorithms – for example the Plegde-algorithm and some Bug-Variants – have been implemented as interactive Java-Applets, see

`http://www.geometrylab.de/`

2.1 Escape from the labyrinth

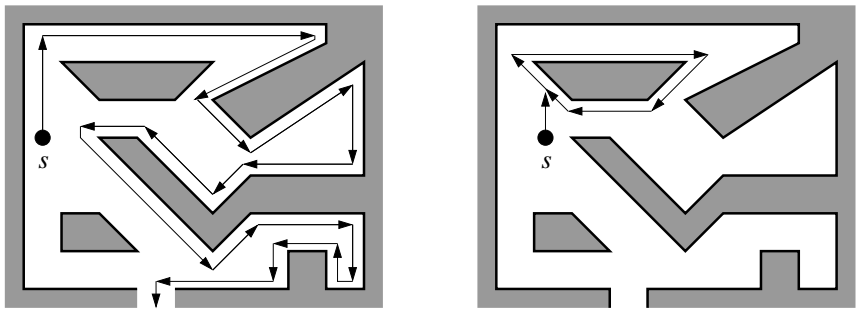
The task of the agent is to escape from a polygonal environment. In the geometric sense the agent escapes, if it finally hits a circle that contains all obstacles.

The agent is point-shaped and makes use of a touch-sensor for following the wall by Left-Hand-Rule or Right-Hand-Rule. Additionally, we allow that the agent can count its total turning angles in a single ; see Figure 2.2(i). The agent realizes when it hits the enclosing circle. In this case the agent successfully escaped from the labyrinth.

2.1.1 Pledge-Algorithm

Algorithm 2.1 Pledge-Algorithmu

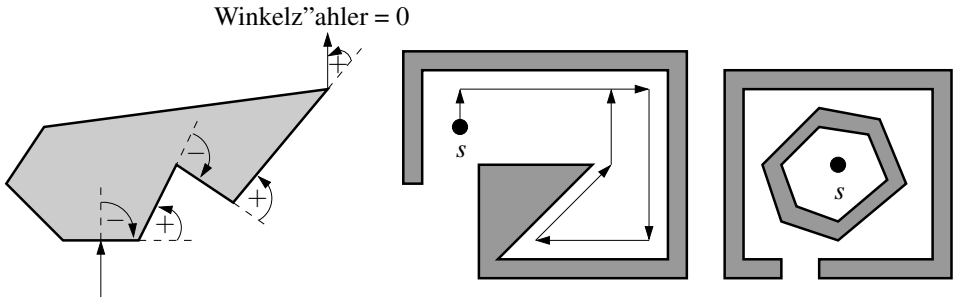
1. Choose direction φ, turn agent into direction φ.
2. Move into dircetion φ, until an obstacle is met.
3. Turn right and follow the wall by Left-Hand-Rule.
4. Follow the wall, sum up the overall turning angles, until the angular counter gets zero, in this case GOTO (2).



(i) Hier funktioniert die ”Linke-Hand-Regel” (ii) Hier nicht

Figure 2.1: Simple strategies cannot be successful.

Note that, simple counting schemes or movements cannot succeed. For example, following the wall by Left-Hand-Rule until the agent points again into direction φ and leave the obstacle right now can result in infinite loops; see Figure 2.2(ii). Just following the boundary could let the agent stuck inside the labyrinth; see Figure 2.1.



(i) (ii) (iii)

Figure 2.2: (i) Angular counter. (ii) Leave-condition “angular counter mod $2\pi = 0$” is not sufficient. (iii) The agent cannot escape.

The following simpe Pledge-Algorithm (Algorithm 2.1) solves the problem. For the correctness proof we require structural properties.

Lemma 2.1 *The angular counter of the Pledge-Algorithm never attains a positive value.*

Proof.

- The angular counter is initialized by zero.
- After hitting an obstacle, the counter gets negative by the first right turn.
- The counter changes continuously, the agent leaves the obstacle if the counter gets zero again.

$\Rightarrow$ the statement is true.

□

Lemma 2.2 *If the agent does not leave the labyrinth by the Pledge-Algorithm, at the end there will be a finite path $\pi_\circ$, so that the agent follows this path infinitely often.*

Proof. The path generated by the Pledge-Algorithm is a polygonal chain. The set of vertices S stem from the vertex set of all obstacles plus vertices inside edges that occur when the algorithm leaves an obstacle at a vertex and hits an edges afterwards. The algorithm leaves an obstacle only at a vertex. Thus the overall number of possible vertices is bounded. If the Pledge leaves an obstacle at the same vertex twice, after that the same path $\pi_\circ$ will be used again and again, since the algorithm is deterministic and we left the same vertex with the same counter twice.

Therefore, if such path does not exists and the agent is not successful, finally the path will not leave an obstacle anymore. Obviously, the path $\pi_\circ$ exists in this case. □

Lemma 2.3 *Assume that the agent does not leave the labyrinth by the Pledge-Algorithm, the above mentioned finite path $\pi_\circ$ does not have self-intersections.*

Proof. An intersection can only occur at the boundary, since all movements (segments) in the *free space* are arranged in parallel. [1]

Let us assume that a part B of $\pi_\circ$ intersects with a former path A of $\pi_\circ$ at the boundary of some P_i. This means that there is a point $z \in P_i$ where B (running from the free space) hits A for the first time. The corresponding segment $e \in P_i$ points into a fixed direction. Therefore after turning, the turning angle, $C_B(z')$, of B at some point $z' \in e$ closely behind z and the turning angle, $C_A(z')$, differ by $2k\pi$ for some $k \in \mathbb{Z}$. If e cause turning angle $-\beta$ for B at z, we have $C_B(z') = -\beta$ and $C_A(z') = -\beta + 2k\pi$ with $k \in \mathbb{Z}$.

For $k > 0$ we have:

$$C_A(z') = -\beta + 2k\pi > -\pi + 2\pi = \pi \; \lightning \text{ by Lemma 2.1.}$$

For $k = 0$ it is clear that $C_A(z') = C_B(z')$ and A and B will never diverge, this contradicts the given situation that A and B are parts of $\pi_\circ$.

From $k < 0$ we conclude $C_A(p) < C_B(p)$ for all points p between z' and the first point z'', where the paths of A and B separate. From $C_A(p) < C_B(p)$ we conclude that at z'' the path B has $C_B(z'') = 0$ and leaves the obstacle. □

Finally, we conclude:

Theorem 2.4 *(Abelson, diSessa)*
For any given polygonal labyrinth, the Pledge-Algorithm will be able to escape from the labyrinth from any starting point, from which a successful path exists. *[Ad80]*

Proof. Assume, that the agent is not successful in the given situation. By Lemma 2.2 follows the finite path $\pi_\circ$ again and again, and the path $\pi_\circ$ has no self-intersections. Either the agent runs along $\pi_\circ$ in ccw-order or in cw-order. In ccw-order the angular counter will increase by $+2\pi$ in any round, which contradicts Lemma 2.1. In cw-order the angular counter will decrease by -2π in any round, thus at some point, the agent cannot leave an obstacle any more and in turn $\pi_\circ$ already belongs to a single obstacle. The agent follows the wall of an obstacle by Left-Hand-Rule and in cw-order, this can only mean that the agent is enclosed by the obstacle; see Figure 2.2(iii). There is no way out of the labyrinth from the given starting point. □

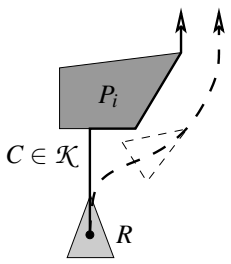
2.1.2 Pledge-Algorithm with sensor errors

The correctness proof of the Pledge-Algorithm (Theorem 2.4) make use of the assumptions that a point-shaped agent counts the angles without any errors and moves precisely into a specified starting direction. setzt voraus, dass der Roboter punktf'ormig ist und fehlerfrei arbeitet. As already mentioned physically the agent need not be a point, we can assume that the agent requires some room for its movement and in principle follows a curve calculated by the Pledge-Algorithm. The main problem is that the curve itself is not computed exactly by the robots abilities. For example the agent cannot precisely measure the turning angles at the boundary and cannot precisely follow a direction.

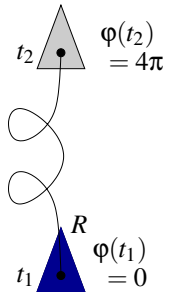
If the agent makes gross faults, we cannot assume that the pledge-like behaviour will succeed. Is there an error bound for the sensors that still allows to escape by a pledge-like behaviour?

[1] All segments that are not part of the boundary of some P_i.

The general idea is as follows. We would like to design a class of curves $\mathcal{K}$ of the configuration space. Such a curve will be computed by an agent with sensor errors and imprecise measurements. Any curve from $\mathcal{K}$ represents a possible path for the escape. As mentioned above, the agent is guided by the computed curve and moves close to it. We would like to define sufficient conditions for the curves, such that the escape is guaranteed, if a corresponding path exist. The curve always have precise orientation. Its computation might be erroneous.

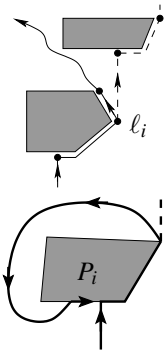


As the curves guides the agent, for convenience, we can consider curves in the configuration space by $C(t) = (P(t), \varphi(t))$ where $P(t) = (X(t), Y(t))$ is the location in the work space and $\varphi(t)$ the current turning angle. Note that two full turns will result in a turning angle of 4π instead of zero. Therefore $\varphi(t) \in \mathbf{R}$! A configuration (x, y, φ) is **half-free**, if the curve touches an obstacle in the work space and **free**, if no obstacle in the work space is met. The set of all half-free points is denoted by $\mathcal{C}_{\text{half}}$ whereas the set of all free points is denoted by $\mathcal{C}_{\text{free}}$. Points, where the curve moves from $\mathcal{C}_{\text{free}}$ to $\mathcal{C}_{\text{half}}$ are called Hit-Points, h_i. Points, where the curve leaves $\mathcal{C}_{\text{half}}$ and enters $\mathcal{C}_{\text{free}}$ are denoted as Leave-Points, ℓ_i. For simplicity, we will also denote the corresponding time parameter by h_i or ℓ_i, respectively.



The Pledge-Algorithm has two movement modi. Either the agent follows the wall and counts turning angles or the agent moves in the free space. Both movements can be erroneous, the agent diverges from the starting direction in the free space and drifts off or the agent cannot count turning angles precisely and will leave the obstacle earlier or later than in the original pledge path.

In principle we have to avoid that the agent moves in infinite loops. The figure shows that a large drift can easily result in a loop. The error of the agent is too large. But also a small deviation in the free space after each leave-point can sum up to a large total drift and an infinite loop. Figure 2.3 shows an example where there are small drifts after each leave that finally results in an infinite loop. This means that the direction in the free space should be globally stable.



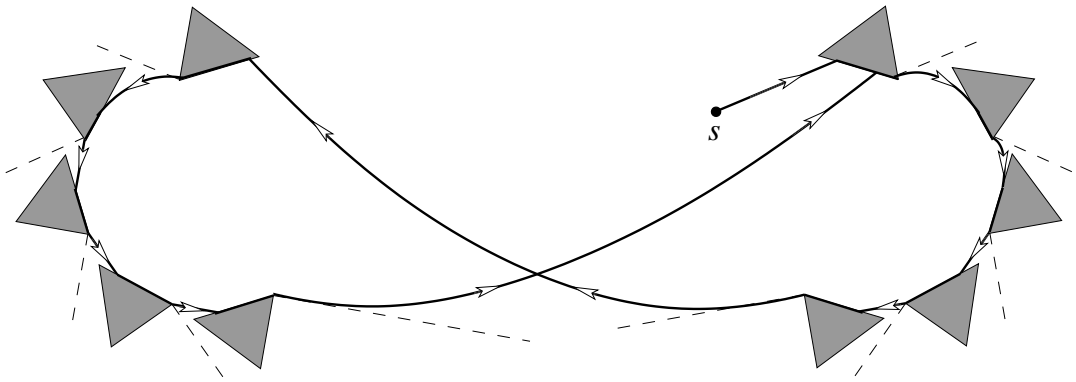


Figure 2.3: Small deviations sum up to a large deviation.

Additionally, for leaving the environment it is highly recommended that the agent at least moves into a certain global direction. One might also think that an erroneous compass will at least allow us to move generally into a half-plane. Therefore we require a $\mathcal{C}_{\text{free}}$-condition:

$$\forall t_1, t_2 \in C : P(t_1), P(t_2) \in \mathcal{C}_{\text{free}} \Rightarrow |\varphi(t_1) - \varphi(t_2)| < \pi$$

which guarantees that the angular counter in the free space maximally differs from the starting direction $\varphi(s)$ only by a fixed value. If we make use of a compass it seems to be reasonable to think that we can guarantee $\forall t : \varphi(t) \in]-\frac{\pi}{2}, +\frac{\pi}{2}[$ in the free space for starting direction zero.

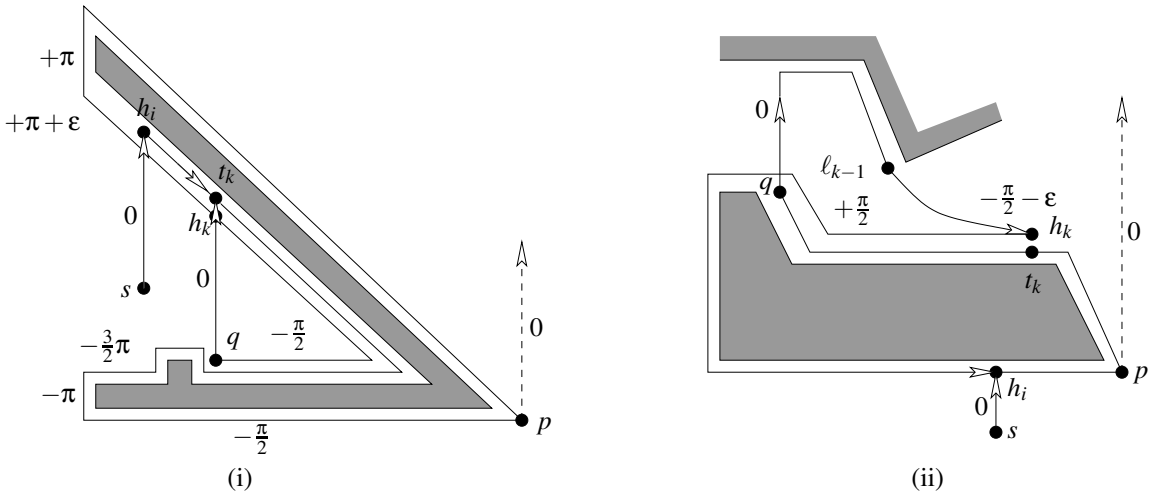


Figure 2.4: A local overturn of the angular counter can result in infinite loops.

Unfortunately, the $\mathcal{C}_{\text{free}}$-condition is not sufficient. We have to combine it with the angular counter at the obstacles. Figure 2.4 shows two examples where the agent overturns the counter at the obstacle for a while (because of sensor errors) but obtains overall precise values later for the leaving condition. This has nothing to do with the free space condition. The infinite loop happens at the obstacle. In this case the agent passes the first true leave-point p and leaves the obstacle later at a point q which is also a legal leave-point.

In Figure 2.4(i) by the overturn the curve mets the same obstacle h_k at again. In Figure 2.4(ii) by the overturn the agent first visit another obstacle but then returns via some leave-point ℓ_{k-1} again to the starting obstacle at h_k. In both case $P(h_k)$ is visited twice and during the first visit at t_k the angular counter was overturned. In the left hand side figure the angular counter at t_k is larger than π, in the right hand side figure the counter at t_k is $+\frac{\pi}{2}$. Figure 2.4(ii) shows a second error source. Since the angular counter at h_k is $-\frac{\pi}{2}-\varepsilon$ both errors also sum up to an error larger than π. We state that for the hit-point h_k the angular counter of a previous visit was overturned. Together with the error at h_k there is an overall overturn larger than π.

We subsume the requirement in the $\mathcal{C}_{\text{half}}$-condition:

$$\forall h_i, t \in C : P(t) = P(h_i) \Rightarrow \varphi(t) - \varphi(h_i) < \pi\,.$$

We can also think about the compass with a deviation of $\pi/2$ into both directions. This means that we can overturn the angle counter at the obstacle by less than $\pi/2$. Together with a deviation of less than $-\pi/2$ in the free space we might hit this point again but $\varphi(t) - \varphi(h_i) < \pi$ holds.

Definition 2.5 Let $\mathcal{K}$ be a class of curves in $\mathcal{C}_{\text{free}} \cup \mathcal{C}_{\text{half}}$ such that any curve from $\mathcal{K}$ fulfils the following conditions:

(i) Parameterized pledge like curve with turn-angles and position:
$C(t) = (P(t), \varphi(t))$ with $P(t) = (X(t), Y(t))$
(ii) At the boundary the curve surrounds obstacles by Left-Hand-Rule.
(iii) Any leave-point is a vertex of an obstacle.
(iv) $\mathcal{C}_{\text{free}}$-condition holds: $\forall t_1, t_2 \in C : P(t_1), P(t_2) \in \mathcal{C}_{\text{free}} \Rightarrow |\varphi(t_1) - \varphi(t_2)| < \pi$
(v) $\mathcal{C}_{\text{half}}$-condition holds: $\forall h_i, t \in C : P(t) = P(h_i) \Rightarrow \varphi(t) - \varphi(h_i) < \pi$

Obviously, the path constructed by the error-free Pledge-Algorithm does belong to $\mathcal{K}$. For the correctness proof that any curve of $\mathcal{K}$ that is constructed in a pledge-like fashion will escape from the labyrinth. We show some important structural properties of curves in $\mathcal{K}$.

Lemma 2.6 *A curve $C \in \mathcal{K}$ has no self-intersection.*

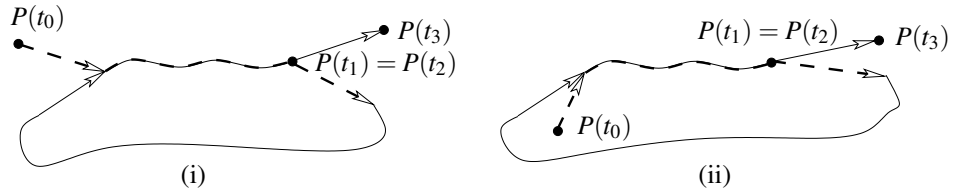


Figure 2.5: The difference between (i) crossing and (ii) touching at t_2.

Note that a curve from $\mathcal{K}$ might touch itself; see Figure 2.5.

Proof. Assume that C has an intersection, consider the first intersection. There are parameter t_1 and t_2 with $t_1 < t_2$ and $P(t_1) = P(t_2)$ and the first intersection occurs at $P(t_2)$.

This means that between t_1 and t_2 there is a cw or ccw turn. If the intersection lies in the free space, obviously the $\mathcal{C}_{\text{free}}$-condition does not hold. Thus we can assume that $P(t_2)$ is in $\mathcal{C}_{\text{half}}$ holds. Consider the case of a cw loop as depicted in Figure 2.6.

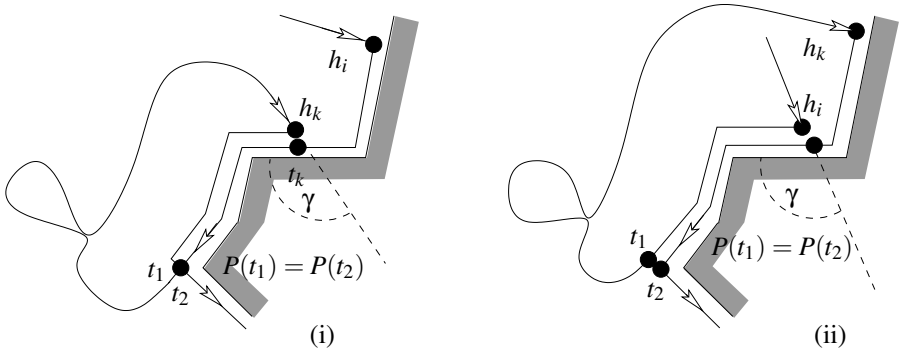


Figure 2.6: Cw loop and two cases.

The curve hits the obstacle at h_i, follows the wall, leaves the obstacle at time t_2 comes back to the obstacles at $h_k > h_i$ and has an intersection at t_2; see Figure 2.6(i). If the point $P(h_k)$ was not visited between h_i and t_1 there is only a touching event at t_2; see Figure 2.6(ii).

Let $\varphi(h_k^+)$ deonte the angular counter after the agent turns into the direction of the corresponding edge. we have $\varphi(h_k^+) = \varphi(h_k) + \gamma$, where γ denotes the turning angle at the edge. We have $-\pi < \gamma < 0$. Additionally, by the full cw turn we conclude $\varphi(h_k^+) = \varphi(t_k) - 2\pi$. Also the $\mathcal{C}_{\text{half}}$-holds and we obtain:

$$\begin{aligned} & \varphi(t_k) - \varphi(h_k) < \pi \\ \Leftrightarrow \quad & \varphi(h_k^+) + 2\pi - \varphi(h_k) = \varphi(h_k) + \gamma + 2\pi - \varphi(h_k) < \pi \\ \Leftrightarrow \quad & \gamma < -\pi \ \lightning \end{aligned}$$

Similar arguments hold for a loop in ccw order, which is an exercise. The first intersection cannot exists. By induction there is no intersection at all. □

Lemma 2.7 *A curve $C \in \mathcal{K}$ hits any edge of the environments at most once.*

Proof. Assume that a single edge e has two hit-points. After the first hit h_i of edge e the curve can only leave the obstacle at a vertex and then comes back to e at h_k; see Figure 2.7.

In $P(h_i)$ and $P(h_k)$ the agent turns clockwise in order to follow the edge e which gives turning angles $-\pi < \gamma_i, \gamma_k < 0$. Let $\varphi(h_i^+)$ and $\varphi(h_k^+)$ denote the turning angles as in the proof of Lemma 2.6 (turning angle directly after the hit-point). Let w. l. o. g. $\varphi(h_i^+) = 0$.

In h_i^+ and h_k^+ the curve follow the edge e, and the direction differs only by $2j\pi$ for some $j \in \mathbb{Z}$ This means $\varphi(h_k^+) = 2j\pi, j \in \mathbb{Z}$.

For $j \neq 0$, and with $\varphi(h_i) = -\gamma_i$ and $\varphi(h_k) = \varphi(h_k^+) - \gamma_k$ we conclude $|\varphi(h_k) - \varphi(h_i)| = |2k\pi - \gamma_k + \gamma_i| > \pi$, which contradicts to the $\mathcal{C}_{\text{free}}$-condition.

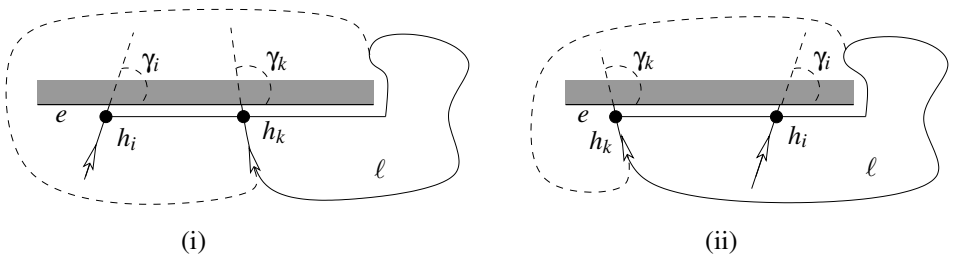


Figure 2.7: A curve from $\mathcal{K}$ hits any edge once.

Therefore we conclude $j = 0$ and also $\varphi(h_k^+) = 0$. But we can argue that there be a full cw or ccw turn from $P(h_i)$ to $P(h_k)$ without intersections; see Figure 2.7. The curve C made a full turn with angular counter change of $\pm 2\pi$. This means that $\varphi(h_k^+) = \pm 2\pi$ should hold. $\lightning$ □

By Lemma 2.7 we can now prove that the condition from Definition 2.5 are sufficient. First, we require a helping lemma. If the curve gets stuck onto a single obstacle, the obstacle should enclose the curve.

Lemma 2.8 *If a curve $C \in \mathcal{K}$ does not leave an obstacle anymore the curve is enclosed by the obstacle.*

Proof. If the curve does not leave an obstacle after the last hit-point, the path along the boundary is repeated infinitely often. The path can be in cw or in ccw order which means a counter change of $+2\pi$ for ccw order or a counter change of -2π in cw order for any round. In the first case at some point the counter gets arbitrarily large for any point on the boundary. So also for the last hit-point and the $\mathcal{C}_{\text{half}}$-condition is violated. This means we can only have a cw order loop and by the Left-Hand-Rule the curve has to be enclosed by the obstacle. □

Theorem 2.9 *(Kamphans, Langetepe, 2003)*
An agent, who follows a path from $C \in \mathcal{K}$ will escape from any labyrinth and from any position, if an escape path exists. *[KL03]*

Proof. If there is an escape path the agent and the curve is not enclosed by an obstacle. Therefore the curve $C \in \mathcal{K}$ will leave any obstacle after a while. Since any edge is hit at most once by Lemma 2.7 there will be no hit any more after a while. The $\mathcal{C}_{\text{free}}$-condition takes care that the agent steadily moves into a halfplane w.r.t. a given direction. Thus we will escape from the environment. □

2.1.3 Applications

We would like to consider the impact of Theorem 2.9. Let us first assume that we make use of a compass for counting the total turns around the obstacles and for holding the direction in $\mathcal{C}_{\text{free}}$. If the deviation from the starting direction is never larger than $(-\pi/2, \pi/2)$, such a compass will help us leaving the labyrinth.

In this case in $\mathcal{C}_{\text{free}}$ we can guarantee an angular range $(-\frac{\pi}{2}, +\frac{\pi}{2})$. Along the boundary the absolut error is smaller than $|\frac{\pi}{2}|$, the maximal positive value along the boundary is smaller than $+\frac{\pi}{2}$, the minimal value in $\mathcal{C}_{\text{free}}$ is larger than $+\frac{\pi}{2}$, this yields the $\mathcal{C}_{\text{half}}$-condition.

$$\forall h_i, t \in C : P(t) = P(h_i) \Rightarrow \varphi(t) - \varphi(h_i) < \pi\,!$$

Corollary 2.10 *The usage of a compass with absolut deviation smaller than $\frac{\pi}{2}$ will help to leave a labyrinth by a pledge like algorithm.*

Next assume that we would like to make use of small deviations of the environment itself. Therefore, we consider obstacles that consists of axis-parallel edges, only. For such orthogonal polygons, we can

simply sum up the reflex vertices (inner angle $> \pi$, $+1$) and the convex vertices (inner angle $< \pi$, -1) as indicated and will leave an obstacle with counter value 0; see Figure 2.8(i). If we guarantee a deviation in the range $(-\frac{\pi}{2}, +\frac{\pi}{2})$ in the free space, we can successfully apply the pledge algorithm. After a hit, we only have to find out whether the edge is horizontal or vertical. For vertical edges we simply slip along the edge and wait for the next hit. Thus we start the movement along the boundary with angular counter $+1$, following a horizontal edge.

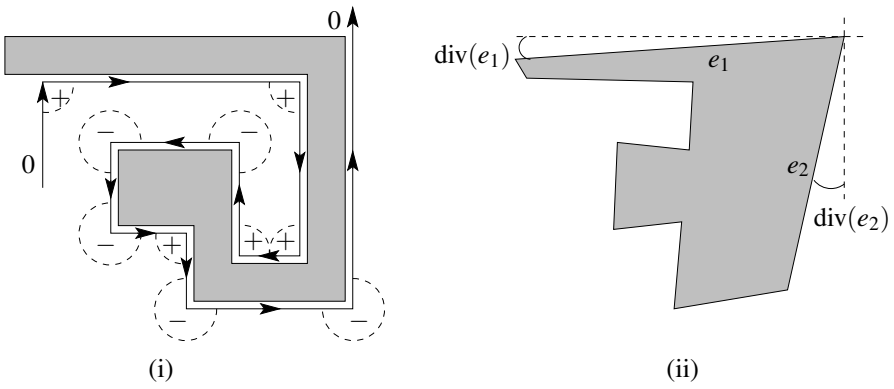


Figure 2.8: (i) Counting the angles in orthogonal polygons, (ii) pseudo-orthogonal polygon with deviation div.

More generally for such a counting argument we have to take care that: Folgende drei Bedingungen sind also einzuhalten:

1. Reflex and convex vertices can be distinguished: Count the rotation correctly.
2. Maximal deviation from the starting direction. Interval of size π.
3. Distinguish: Horizontal/verticale edge.

Now let us assume that the polygons are not exactly axis-parallel but roughly as shown in Figure 2.8(ii).

By, $\mathrm{div}(e)$, for edge Kante $e = (v, w)$ we define the smallest deviation from a vertical or horizontal edge passing through w or v. This deviation should be small in total for all edges. Additionally, we would like to take care that the number of reflex and convex vertices of the polygon fits to an axis-parallel polygon.

Definition 2.11 A polygonal scene is δ-pseudo orthogonal for $\delta > 0$, if for any polygon we have (number if convex vertices) = (Number of reflex vertices) + 4 and $\mathrm{div}(P) := \max_{e \in P} \mathrm{div}(e) \leq \delta$ holds.

For δ-pseudo orthogonal we would like to proceed as indicated above and have to fulfil the above three conditions. Let us additionally assume that the angle counter device on the obstacle has a deviation (error) of no more than ρ. The following corollary shows some legal value for ρ and δ also with the interplay of the free space movements.

Corollary 2.12 *Let P be a δ-pseudo orthogonal scene and let us assume that we count the angles with precision ρ such that $\delta + \rho < \frac{\pi}{4}$. If we take care that the deviation from the starting direction in C_{free} is no more than $\frac{\pi}{4} - 2\delta - \rho$, the simple reflex/convex counter pledge like algorithm helps us to leave a labyrinth.*

Proof. We have to distinguish between reflex and convex vertices at the boundary, otherwise the $+1$, -1 counting will be erroneous. We consider the worst-case situation for our measurement. Let us assume that at a vertex we measure the outer angle γ as shown in Figure 2.9. For $\gamma < \pi$ we assume that we have a reflex vertex and for $\gamma > \pi$ we assume that the vertex is convex.

convex vertex reflex vertex

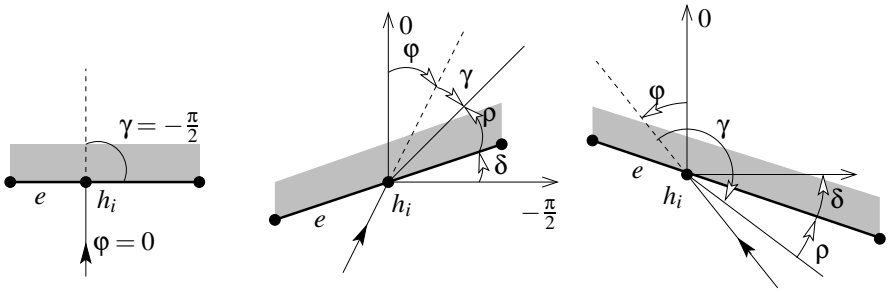
Figure 2.9: Maximal deviation from the outer angle (γ) and the assumed (dashed) angle for a convex or reflex vertices.

We would like to guarantee a correct detection for tha maximal error and deviation. The correct angle of a convex vertex is $\frac{3}{2}\pi$ and the correct angle of a reflex vertex is $\frac{\pi}{2}$. Therefore we require:

$$\frac{3}{2}\pi - 2\delta - \rho > \pi \quad \text{und} \quad \frac{\pi}{2} + 2\delta + \rho < \pi$$
$$\Leftrightarrow 2\delta + \rho < \frac{\pi}{2} \qquad \Leftrightarrow 2\delta + \rho < \frac{\pi}{2}.$$

Additionally, we would like to distinguish between horizontal and vertical edges after hitting an edge. Either we slip along the vertical edge or we start at the horizontal with the simple counter. Again we assume the worst-case situation; see Figure 2.10.

We measure the turning angle γ for the corresponding edge. For a horizontal edges this is exact $-\frac{\pi}{2}$; see Figure 2.10(i). If this angle is between $-\frac{\pi}{4}$ and $-\frac{3\pi}{4}$ we conclude that we have a horizontal edge. Otherwise the edge is assumed to be vertical. Note that γ is always negative. We assume that the deviation from the starting direction is φ.



(i) (ii) (iii)

Figure 2.10: Hitting a horizonal edges (i) in the error-free case, (ii) for small absolut γ, (iii) for large absolut γ.

In Figure 2.10(ii) the deviations φ, δ and ρ should make $|\gamma|$ as small as possible and still smaller than $-\frac{\pi}{4}$, φ is negative. In Figure 2.10(iii) the deviations φ, δ and ρ should make $|\gamma|$ as large as possible and larger than $-\frac{3\pi}{4}$, φ is positive. We conclude from Figure 2.10(ii)

$$\gamma = -\frac{\pi}{2} - \varphi + \delta + \rho < -\frac{\pi}{4} \Leftrightarrow -\frac{\pi}{4} + \delta + \rho < \varphi,$$

and from Figure 2.10(iii)

$$\gamma = -\left(\frac{\pi}{2} + \varphi + \delta + \rho\right) > -\frac{3}{4}\pi \Leftrightarrow \frac{\pi}{4} - \delta - \rho > \varphi.$$

We We detect horizontal edges precisely if $\varphi(h_i) \in]-\frac{\pi}{4} + \delta + \rho, \frac{\pi}{4} - \delta - \rho[$ holds. Therefore we require $\delta + \rho < \frac{\pi}{4}$. A maximal deviation of $\frac{\pi}{4} - \delta - \rho$ would be enough for correct detections. Since we might start the free space move with an error of δ at a vertex we require $\frac{\pi}{4} - 2\delta - \rho$ for the deviation. □

Exercise 19 *In the above corollary we can set* $\delta = 0$ *and* $\rho = 0$ *and require that we do not deviate in the free space by an angle of* $\pi/4$*. Why is this different to the error-free case where an error of less than* $\pi/2$ *was allowed for the free space movements.*

2.2 Navigation with touching sensor

We distinguish between the term Navigation for visiting a given target (known coordinates) and the term Searching for searching for an unknown goal (unknown coordinates). The family of the so-called Bug-Algorithms are the first algorithms for the navigation task in polygonal environments[2]. The first simple strategies have been introduced by Lumelsky and Stepanov [LS87], extensions and modifications came from Sankaranarayanan et al. [SM92, SV90a, SV90b, SV91]. Many variants have been discussed since then. Bug-variants have been practically used for the navigation of some of the Mars rovers like Sojourner or Bridget, (see also RoverBug, [LB99]).

In the following we assume that the coordinates of the target are known and that the agent have a finite storage so that coordinates of points and /or length of (sub)path can be stored. The agent also is aware of the coordinates of its current position, for example by GPS.

Any Bug-algorithm runs with the same principle and actions: The agent moves toward the target until an obstacle is visited (Move-To-Target action) Then the agent follows the wall of the obstacle for a while (Follow-Wall action) until some condition triggers the next movement in the free space toward the target. The leaving condition is the main difference between the Bug-variants.

We assume that the agent R is point-shaped and equipped with a touch sensor for the Follow-Wall action. We make use of the following notations:

- $|pq|$ denotes the distance between two points p and q,
- $D := |st|$ denotes distance from start s to target t,
- π_S denotes the path of a strategy S from s to t; $|\pi_S|$ denotes the length of this path where $|\pi_S| := \infty$, if there is no such path,
- $\mathrm{U}(P_i)$ denotes the perimeter of the obstacle P_i.

2.2.1 Strategies of Lumelsky and Stepanov

The first algorithm Algorithm 2.2, Bug1, leaves an obstacle P_j at a point $p \in P_i$ that is the closest point to the target. This defines a sequence of **Hit-Points** h_i, where the agent hits an obstacle and **Leave-Points** ℓ_i, where the agent leaves an obstacle. Since the coordinates of the target and the coordinates of the current position are known, the agent can calculate the corresponding distances. Additionally, by successively counting small steps, the agent can calculate the path length of the path along the boundary during the circumnavigation and also the path length to the currently computed optional leave-point. Additionally, the path length (along the boundary) to the With these values the agent can perform step 3 of Algorithm 2.2 Figure 2.11 shows an example for the path of Bug1.

We assume that there is a finite number of polygonal obstacles and that the obstacles do not touch or intesect. The number of polygons is finite in the sense that any circle of fixed radius contains only finitely many *obstacles* P_i.

Theorem 2.13 *(Lumelsky, Stepanov, 1985)*
Strategy Bug1 finds a path from a starting point s to a target t, if such a path exists. *[LS87]*

Proof. For the sequence of hit- und leave-points we have

$$|st| \geq |h_1 t| \geq |\ell_1 t| \ldots \geq |h_k t| \geq |\ell_k t|.$$

[2] In this case Bug is not meant as a synonym for an error.

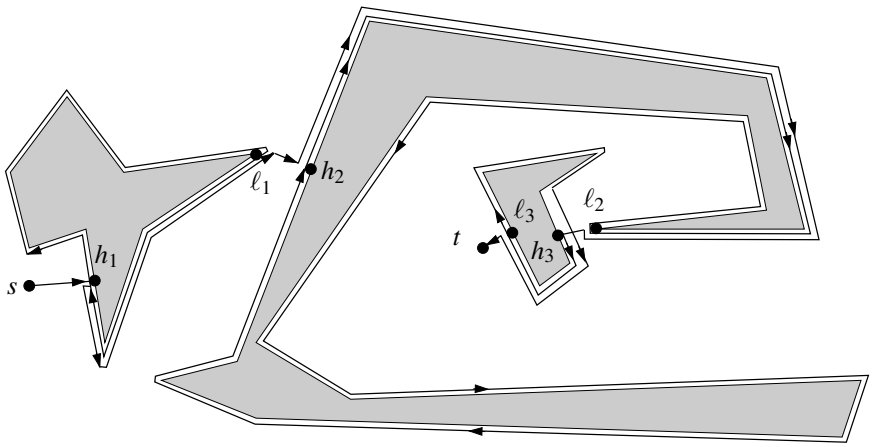


Figure 2.11: Example execution of strategy Bug1.

Algorithm 2.2 Bug1

0. $\ell_0 := s$, $i := 1$
1. From ℓ_{i-1} move toward the target, until
   (a) Target is visited: Stop!
   (b) An obstacle is reached at point h_i. If $h_i = \ell_{i-1}$: The goal cannot be reached.
2. Surround the obstacle in cw-order — keep track of the point ℓ_i on the boundary with the shortest distance to t —, until
   (a) Target is visited: Stop!
   (b) h_i is reached.
3. Move along the shortest path along the boundary to ℓ_i.
4. Increase i, GOTO 1.

Since for any visited obstacles we choose a leave-point that is closest to the target, any obstacles can be left. Otherwise, if this is not the case, the obstacle would fully enclose the target. This also means that we have a strict $>$ in the above sequence. Any obstacle can be visited only once and the finite number of obstacles within the circle of radius $|st|$ around t lead to a finite sequence which ends at the target. □

For the performance we conclude:

Theorem 2.14 *(Lumelsky, Stepanov, 1985)*
Let π_{Bug1} denote the path from s to t, for the successful application of the strategy Bug1. [LS87] We have:

$$|\pi_{\text{Bug1}}| \leq D + \frac{3}{2}\sum_i \mathrm{U}(P_i).$$

Proof. We subdivide the path into the movements along the boundary of the obstacles P_i and the movements in the free space. Since step 3. of Algorithm 2.2 makes use of a shortest path we have path length $\frac{3}{2}\sum \mathrm{U}(P_i)$ for any visited obstacle. It remains to calculate the length D' for the free space movements. We show that $D' \leq D$ holds.

$$D' \quad = \quad |sh_1| + |\ell_1 h_2| + \ldots + |\ell_{k-1} h_k| + |\ell_k t|$$

$$\begin{aligned} &\leq |sh_1| + |\ell_1 h_2| + \ldots + |\ell_{k-1} h_k| + |h_k t| \\ &= |sh_1| + |\ell_1 h_2| + \ldots + |\ell_{k-1} t| \\ &\ldots \\ &\leq |sh_1| + |\ell_1 h_2| + |h_2 t| \\ &= |sh_1| + |\ell_1 t| \\ &\leq |sh_1| + |h_1 t| = |st| = D \end{aligned}$$

We can compare the above result with the lower bound Theorem 2.23 and conclude that in comparison to any other Bug-strategy the strategy Bug1 can be consodered to be $\frac{3}{2}$-competitive. □

Corollary 2.15 *Bug1 is $\frac{3}{2}$-competitive in comparison to arbitrary Bug-like online strategyies.*

In the next variant we would like to avoid complete circumnavigations of the obstacles. Therefore we make use of a line G passing through the segment st. At any time during the Wall-Follow action we will try to move toward the target if we reach a point at G that is closer to t than the previous hit-point; see Algorithm 2.3. Note that by this action, it is possible to visit an obstacles more than once which was impossible for Bug1. h_j and ℓ_j do no longer denote hit- and leave-points of the j-th obstacle.

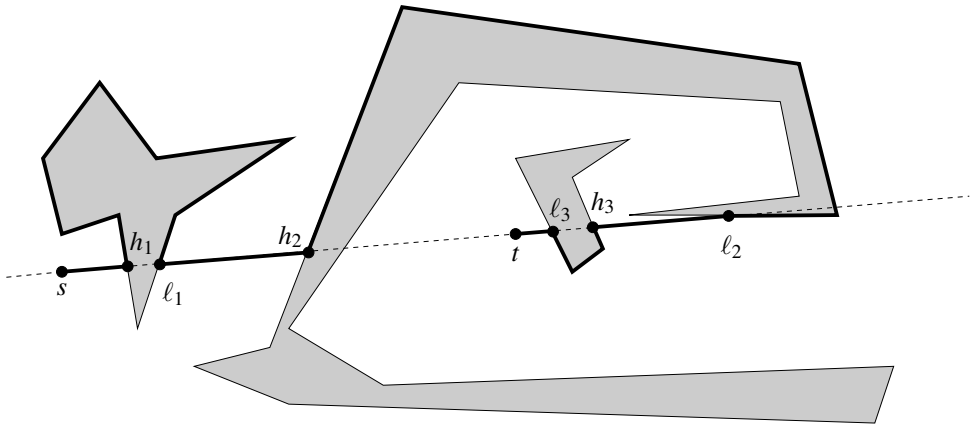


Figure 2.12: Example of the execution of the strategy Bug2.

Figure 2.13 shows an example, where an obtacle is visited more than once. After hit-point h_3 the agent does not leave the obstacle at p_1 or ℓ_1 since $|h_3 t|$ is smaller thanthe distance of p_1 and ℓ_1 to t. At p_2 and p_3 the agent cannot leave the obstacle since the segments $\overline{p_{2/3}t}$ are blocked by the obstacle.

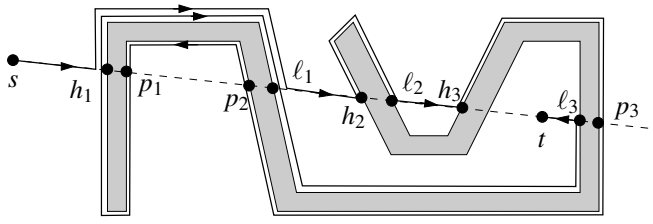


Figure 2.13: The execution of Bug2 can lead to several visits of the same obstacle.

The number of polygons is finite in the sense that any circle of fixed radius contains only finitely many *obstacles P_i*.

Lemma 2.16 *The strategy Bug2 meets finitely many obstacles.*

Proof. In step 2b of Algorithm 2.3 the agent leaves an obstacle only if $|\ell_j t| < |h_j t|$ holds. Since the circle of radius $|\overline{st}|$ around t contains only finitely many obstacles we can hit only finitely many obstacles. □

Algorithm 2.3 Bug2

0. $\ell_0 := s$, $j := 1$
1. From ℓ_{j-1} move toward the target, until
   (a) Target is reached: Stop!
   (b) An obstacle is visited at h_j.
2. Surround the obstacle in cw-order, until
   (a) Target is reached: Stop!
   (b) The line passing segment st is visited at point q, $|qt| < |h_j t|$ and $\overline{qt}$ is *free*, such that we can leave the obstacle from q toward the target.
   Set $\ell_j := q$, $j := j+1$ and GOTO 1.
   (c) h_j is visited again without reaching a point q as in described in b). The target cannot be reached. erreicht werden.

The number of surroundings depend on the intersections of the line passing through st with the boundary of the relevant obstacles.

Lemma 2.17 *Let n_i denote the number of intersections of the line $\overleftrightarrow{st}$ passing through st with the boundary of polygon P_i. The strategy Bug2 visits any point of P_i only $\frac{n_i}{2}$ times.*

Proof.
Bug2 successively defines pairs (h_j, ℓ_j) of hit- und leave-points and by the leave condition we have

$$|h_j t| > |\ell_j t| > |h_{j+1} t|.$$

This means that any point is only once a leave-point or a hit-point and any intersection point can appear only in one pair (h_j, ℓ_j). On the other hand a single pair can only lead to one full surrounding, if the same hit-point is visited, the strategy stops. We have at most $\frac{n_i}{2}$ pairs and surroundings. □

Finally we conclude that we have only finitely many relevant intersections and either the strategy visits a current hit-point again and the corresponding obstacle enlcoses the target or we will finally succeed.

Corollary 2.18 *Strategy Bug2 is successful, if the target can be reached.*

The performance of Bug2 is given in the following statement.

Theorem 2.19 *(Lumelsky, Stepanov, 1985)*
Let π_{Bug2} denote the path from s to t, for the successful application of strategy Bug2. We have

$$|\pi_{\text{Bug2}}| \leq D + \sum_i \frac{n_i \, \mathrm{U}(P_i)}{2}.$$

Here P_i is an obstacle that is visited during the execution of Bug2. *[LS87]*

Proof. The term $\sum \frac{n_i \mathrm{U}(P_i)}{2}$ follows from Lemma 2.17. For the length of the free space movements, say D', between the obstacles, we make use of the same arguments as in the proof of Theorem 2.14 and conclude $D' \leq D$. □

Bug2 is not always better than Bug1. Obviuously, in the presence of convex obstacles, Bug2 outperforms Bug1.

Corollary 2.20 *For a polygonal scene with convex obstacles the successful application of strategy Bug2 has path length*

$$|\pi_{\text{Bug2}}| \leq D + \sum_i \mathrm{U}(P_i).$$

Exercise 20 *Compare the variants Bug1 and Bug2. Present an example where Bug1 outperforms Bug2. Show that for both strategies the performance guarantee is tight.*

2.2.2 Strategies from Sankaranarayanan and Vidyasagar

Many variants of the Bug-strategies have been discussed. Many of them make use of more sensor power for local improvement. For example a VisBug2 strategy makes use of a visibility range and can find local short-cuts for the Bug2 path. We would like to mention some structural different variants from Sankaranarayanan and Vidyasagar. The reason is that we would like to show that some local optimization can have unexpected disadvatages.

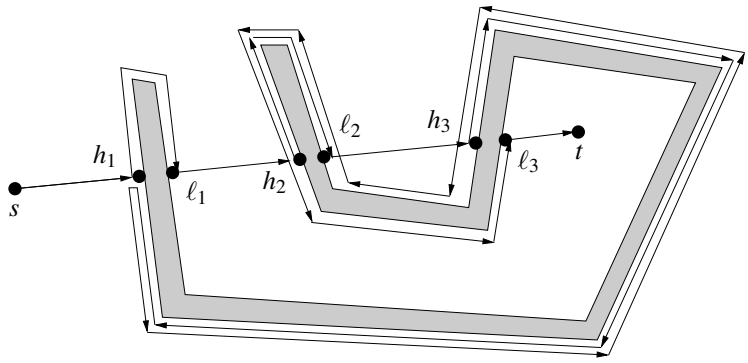


Figure 2.14: Example of the execution of Change1.

Bug1 fully surrounds any obstacle, Bug2 tries to avoid this by moving toward the goal a bit earlier. In this case a single obstacle can be visited many times. Algorithm 2.4 tries to avoid this behaviour: If a surrounding is started, and an old hit- or leave-point (not the current hit-point!) is visited, the strategy starts moving along the boundary in ccw-order; see Figure 2.14.

Theorem 2.21 *(Sankaranarayanan, Vidyasagar, 1990)*
For the length of the path of the successful application of strategy Chang1 we have *[SV90a]*

$$|\pi_{\text{Change1}}| \leq D + 2 \cdot \sum_i \mathrm{U}(P_i).$$

Proof. Exercise □

Strategy Change2 (Algorithm 2.5) differs from Change1 only in the leaving condition. The leave-point is not restricted to a point on the line $\overleftrightarrow{st}$. As soon as there is a point q on the boundary in the Follow-Wall action that is closer to the target than the distance $|ht|$ for the last hit-point,we will leave the obstacle toward the target, if this is possible. Note that such a behaviour can also be used for a variant of Bug2.

Theorem 2.22 *(Sankaranarayanan, Vidyasagar, 1990)*
For the length of the path of the successful application of strategy Chang1 we have *[SV90b]*

$$|\pi_{\text{Change2}}| \leq D + 2 \cdot \sum_i \mathrm{U}(P_i).$$

Proof. Exercise □

Exercise 21 *Present proofs for the above two Theorems. Show that the bounds are tight.*

Algorithm 2.4 Change1

0. $\ell_0 := s$, $i := 1$
1. Move from ℓ_{i-1} along the line passing s and t toward the target, until
   (a) Target is reached: Stop!
   (b) An obstacle is reached at h_i.
2. Surround the obstacle, until
   (a) Target is reached: Stop!
   (b) The line passing s and t is visited a some point q such that the distance from q to t is smaller than $h_i t$ and the segment $\overline{qt}$ is *free* (see refalgobug2).
   Set $\ell_i := q$, $i := i+1$ und GOTO 1.
   (c) A hit- or leave-point h_j orℓ_j with $j < i$ is visited: Move back to h_i in ccw-order and start ccw-order surrounding under condition (a), (b) oder (d) (not (c) again!)
   (d) h_i is visited again without reaching a point as indicated in (b) or (c). The goal is enclosed by an obstacle.

Algorithm 2.5 Change2

As Change1, but:

0. $\ell_0 := s$, $i := 1$
1. Move from ℓ_{i-1} along the line passing s and t toward the target, until

2.(b) A point q is visited such that the distance from q to t is smaller than $h_i t$ and the segment $\overline{qt}$ is *free* (see refalgobug2).
Set $\ell_i := q$, $i := i+1$ und GOTO 1.

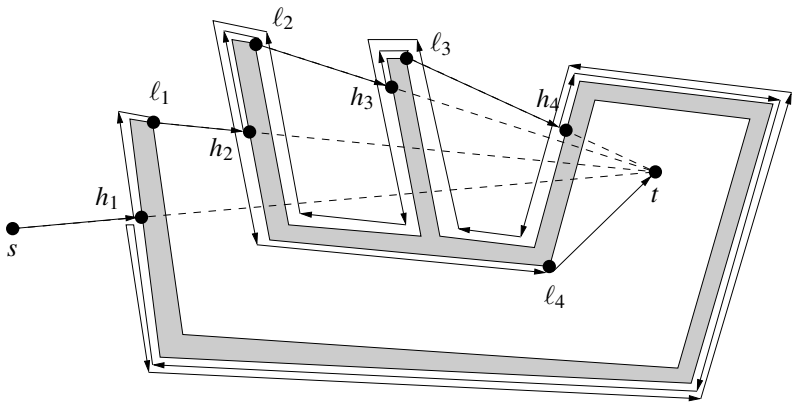


Figure 2.15: Example execution of strategy Change2.

2.2.3 Lower Bound

Finally, we would like to prove a lower bound for any kind of Bug-strategy that makes of the same category of actions. We would like to show that we can force any strategy to surround at least the relevant obstacles once. Note, that a lower bound is always a precise environment description.

Theorem 2.23 *(Lumelsky, Stepanov, 1985)*
For any strategy S (due to the action-model), and for any $K > 0$, there exist a strategy with arbitrary $D > 0$, sucht that for any $\delta > 0$: $|\pi_S| \geq K \geq D + \sum U(P_i) - \delta$. P_i are at least all obstacles that lie within the circle of radius D around the target. *[LS87]*

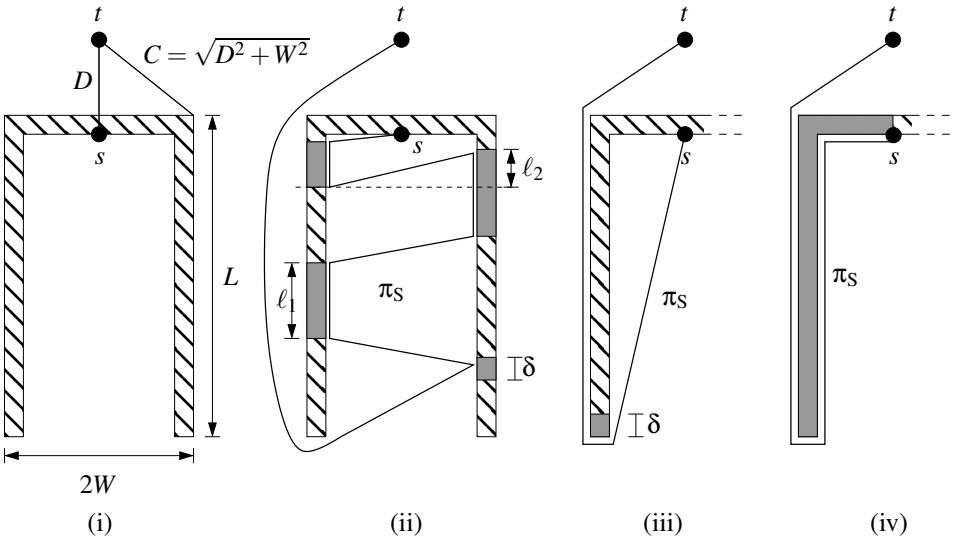


Figure 2.16: (i) "virtual" horse-shoe, (ii) "real" obstacles, (iii) shortest possible starting path, (iv) almost efficient path.

Proof. We consider a virtual horse-shoe as depicted in Figure 2.16(i) with width $2W$, thickness $\varepsilon \ll \delta$ and length L. The distance from s to t is D. The agent starts inside the horse-shoe. The idea is that the horse-shoe becomes (partly) real only in the Follow-Wall modus after touching the inner walls. For any hit we let a portion of length δ become real; see Figure 2.16(iii).

For the strategy in Figure 2.16(iv) we can let half of the horse-shoe become real. Therefore we let L and W be large enough such that $D + W - \sqrt{D^2 + W^2} \leq \delta/2$ and $L + W - \sqrt{L^2 + W^2} \leq \delta/2$ holds and this gives a path length of

$$|\pi_S| \geq \sqrt{L^2 + W^2} + L + \sqrt{D^2 + W^2} \quad \geq \quad D + W + L + W - \delta \tag{2.1}$$

$$= \quad D + 2(L+W) - \delta. \tag{2.2}$$

We can take the thickness ε into account by the use of some δ' instead of δ and obtain the desired result. So this analysis covers Figure 2.16(iv) and obviously also Figure 2.16(iii). We conclude $|\pi_S| \geq K \geq D + \sum \mathrm{U}(P_i) - \delta'$.

The remaining problem is that the strategy might decide to visit the left *and* right *inner arm* of the horse-shoe as given in Figure 2.16(ii). In this figure we have a problem with the obstacles parts on the right-hand side that *overlap* with obstacles parts from the left-hand side. For example in Figure 2.16(ii) l_2 at the right arm overlaps with some obstacle part at the left arm.

Let us neglect thickness ε for a while. Outside the horse-shoe the path length can be measured by $|\pi_O| \geq L + C$ with $C = \sqrt{D^2 + W^2}$. For the non-overlapping boundary parts, say P_i, we have $L \geq \sum \frac{1}{2} \mathrm{U}(P_i)$. Let π_I denote the path inside the horse-shoe. Let π_{I_1} be the part of π_I that simply *passes* the non-overlapping parts P_i. We have $|\pi_{I_1}| \geq \sum \frac{1}{2} \mathrm{U}(P_i)$ for all those obstacles parts. Let π_{I_2} be the remaining part of π_I. This path consists of the movements from left to right and the remaining overlapping parts, say P_j (for example l_2 in Figure 2.16(ii) belongs to some P_j). We conculde that together with the movements from left to right we have $|\pi_{I_2}| \geq \sum \mathrm{U}(P_j)$ for all such overlapping boundary parts P_j. In principle one half of the obstacle boundary is covered by passing P_j and the other half is covered by a movement to the other side.

Altogether, and a bit more precisesly (taking thickness into account) we conclude $|\pi_S| \geq \sqrt{D^2 + W^2} + \sum \mathrm{U}(P_k) - 2n\varepsilon$ for all n obstacles P_k. We have $n \leq \frac{2L}{\delta}$ and with $\varepsilon \leq \delta^2/(8L)$ we conclude $2n\varepsilon \leq \delta/2$. Altogether, $|\pi_S| \geq D + W + \sum \mathrm{U}(P_i) - \delta$ holds in this case. □

Chapter 3

Online searching for objects

In this chapter we collect several results that consider the task of searching for an unknown object. Different from the navigation task, the position of the object is not known. The object is detected due to the sensor abilities of the agent.

For example one might consider the case that the agent is equipped with a sight system. For example inside simple polygons we can assume that the agent is point-shaped and the visibility information is given by the visibility polygon.

Definition 3.1 Let P be a simple polygon and r a point with $r \in P$. The **visibility polygon** of r w.r.t P, $\mathrm{Vis}_P(r)$, is the set of all points $q \in P$, that are visible from r inside P, i.e., the line segment $\overline{rq}$ is fully inside P.

We start our consideration by searching for a goal inside a corridor polygon. The polygon can also be modelled by a line. Note that the visibility information is not helpful, if the corridor has many small kinks or caves, where the goal might be hidden.

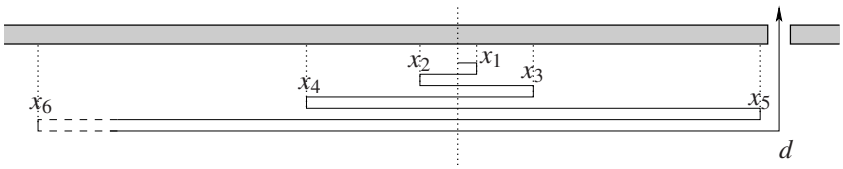


Figure 3.1: Searching for a door along a line.

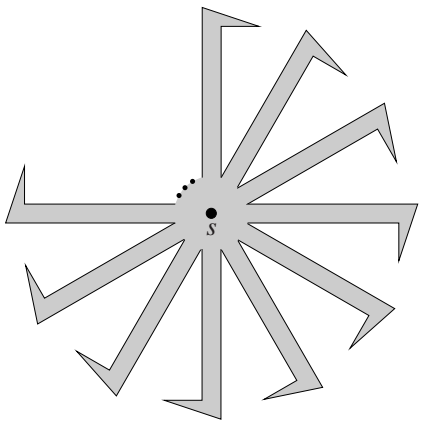


Figure 3.2: In which corridor lies the target point?

3.1 2-ray search and the Theorem of Gal

As a special case we consider the problem of searching for a point along a line (or on 2 rays emanating from a starting point). This problem is also known as the lost-cow or cow-path problem; see citekrt-sueor-93. The cow is searching for the hole in the fence or an agent is searching for a door along a wall. Form the starting position neither the direction nor the distance d to the goal (door/hole) is given. We assume that the agent has no sight system or as mentioned above at any point on the line the sight is very limited (for example by many kinks). Thus the agent detects the goal only by visiting the goal exactly.

The agent cannot concentrate on one direction, because the goal might be on the opposite side. Any reasonable strategy successively changes the direction and can be described by an infinite sequence $(x_1, x_2, x_3, \ldots)$ with $x_i > 0$. The agent runs x_1 steps to the right, returns to the start, runs x_2 steps to the left, returns to the start, runs $x_3 > x_1$ steps to the right again and so on; see Figure 3.1. For the searching depth x_j we can assume that they are monotonically increasing on each side, i.e., $x_{i+2} > x_i$.

We compare the length of the agents path to the shortest path to a goal. It is sufficient to consider the local worst-case situation. In the beginning the agent returns to the start by path of length $2x_1$. If the goal is located arbitrarily close on the opposite side, there is no competitive strategy since for any $C > 0$ there will be an $\varepsilon > 0$ such that $2x_1 + \varepsilon > C \cdot \varepsilon$ holds. Therefore we require an additive constant in this case. Alternatively, we assume that the goal is at least step 1 away from the start. These two interpretation are equivalent.

Exercise 22 *Show that for the 2-ray search problem, the following interpretations are equivalent: A strategy is C competitive with some additive constant A. A strategy is C competitive without additive constant but the goal is at least step B away from the start.*

We assume that the goal is at least one step away from the start. The local worst-case for the competitive ratio is that the agent slightly miss the target at distance d by an ε, performs another turn on the opposite side and visits the target at distance $d + \varepsilon$; see Figure 3.1. Now the task is to find a sequence $(x_1, x_2, x_3, \ldots)$ such that

$$\sum_{i=1}^{k+1} 2x_i + x_k \leq C \cdot x_k$$

holds for all k and C is as small as possible. This means that

$$\frac{\sum_{i=1}^{k+1} 2x_i + x_k}{x_k} = 1 + 2\frac{\sum_{i=1}^{k+1} x_i}{x_k}$$

has to be minimized for all k.

A reasonable stratagy doubles the distance all the time, that is $x_i = 2^i$; see Figure 3.1. Such a doubling heuristic is indeed optimal as shown in [BYCR93] and also in [Kle97]. The competitive ratio is bounded by $C = 9$.

Form $\sum_{i=1}^{k+1} x^i = \frac{x^{k+2}-1}{x-1} - 1$ we conclude $\frac{\sum_{i=1}^{k+1} 2^i}{2^k} = \frac{2^{k+2}-2}{2^k} = 4 - \frac{2}{2^k} < 4$ and attain 9 as the overall ratio which is attained asymptotically.

In the context of Search Games Gal [Gal80] has shown that under certain condition such functionals $F_k(f_1, f_2, \ldots, f_{k+1}) := \frac{\sum_{i=1}^{k+1} f_i}{f_k}$ can be minimized for all k by an exponential sequence $f_i = a^i$ for some $a > 1$. Here $X = (f_1, f_2, \ldots)$ is a sequence of positive values f_i and the functional F_k depends on $k+1$ successive entries of X. We can also write $F_k(f_1, f_2, \ldots)$ statt $F_k(f_1, f_2, \ldots, f_{k+1})$ it the number of entries used is clear from the context. We are searching for an optimal sequence X. More precisely: The supremum of $F_k(X)$ over alle k gives the perfomance of X (it need not be a maximum) and we are searching for a sequence X that gives the infimum on all such suprema (it need not be a minimum). Thus we are searching for X such that

$$\inf_Y \sup_k F_k(Y) = C \text{ und } \sup_k F_k(X) = C\,.$$

The following general result helps to optimize functionals in the sense explained above. Ee do not give a formal proof for it.

Theorem 3.2 *(Gal, Alpern, 2003)*
Given a sequence of functionals $F_k(X)$ for all $k \geq k_0$ and a sequence $X = (x_1, x_2, x_3, \ldots)$ with $x_i > 0$. For X let $k+i$ be the largest Index, so that $F_k(X)$ depends on x_{k+i}.

For two sequences $X = (x_1, x_2, x_3, \ldots)$ and $Y = (y_1, y_2, y_3, \ldots)$ let $X + Y := (x_1 + y_1, x_2 + y_2, x_3 + y_3, \ldots)$ and $\alpha \cdot X := (\alpha \cdot x_1, \alpha \cdot x_2, \alpha \cdot x_3, \ldots)$.

If F_k fulfils the conditions:

(i) F_k *ist stetig,*

(ii) F_k *ist unimodal:* $F_k(\alpha \cdot X) = F_k(X)$ *und* $F_k(X+Y) \leq \max\{F_k(X), F_k(Y)\}$,

(iii)

$$\liminf_{a \mapsto \infty} F_k\left(\frac{1}{a^{k+i}}, \frac{1}{a^{k+i-1}}, \ldots, \frac{1}{a}, 1\right) = \lim \inf_{\varepsilon_{k+i}, \varepsilon_{k+i-1}, \ldots, \varepsilon_1 \mapsto 0} F_k\left(\varepsilon_{k+i}, \varepsilon_{k+i-1}, \ldots, \varepsilon_1, 1\right),$$

(iv)

$$\liminf_{a \mapsto 0} F_k\left(1, a, a^2, \ldots, a^{k+i}\right) = \lim \inf_{\varepsilon_{k+i}, \varepsilon_{k+i-1}, \ldots, \varepsilon_1 \mapsto 0} F_k\left(1, \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_{k+i}\right),$$

(v) $F_{k+1}(f_1, \ldots, f_{k+i+1}) \geq F_k(f_2, \ldots, f_{k+i+1})$.

Then

$$\sup_k F_k(X) \geq \inf_a \sup_k F_k(A_a)$$

where $A_a = a^0, a^1, a^2, \ldots$ und $a > 0$. *[AG03]*

The Theorem says that due to the correctness of some natural conditions the functionals F_k can be minimized for all k by an exponential sequence. We can apply the Theorem on the 2-ray search problem and have to show that the requirements hold for $F_k(f_1, f_2, \ldots) := \frac{\sum_{i=1}^{k+1} f_i}{f_k}$. Obviously, F_k is continuous. Also unimodality holds. We have

$\frac{\sum_{i=1}^{k+1} \alpha \cdot f_i}{\alpha \cdot f_k} = \frac{\sum_{i=1}^{k+1} f_i}{f_k}$. Additionally, from the general statement $\frac{a}{b} \geq \frac{c}{d} \Leftrightarrow \frac{a+c}{d+b} \leq \frac{a}{b}$ (shown by simple equivalence) we conclude that for two sequences $X = (x_1, x_2, x_3, \ldots)$ and $Y = (y_1, y_2, y_3, \ldots)$ we have $F_k(X+Y) \leq \max\{F_k(X), F_k(Y)\}$. The corresponding limits for the special sequences $(\varepsilon_k, \varepsilon_{k-1}, \ldots, \varepsilon_1, 1)$ and $(1, \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_k)$, respectively run to infinity and the statemens (iii) and (iv) trivially hold. Also statement (v) holds because on the left hand side there is an additional number in the numerator. Altogether, $\frac{\sum_{i=1}^{k+1} f_i}{f_k}$ is minimized by $f_i = a^i$. Therefore, the remaining task is to find the best value for a. From $\sum_{i=1}^{k+1} a^i = \frac{a^{k+2}-1}{a-1} - 1$ and the fact that a^k has to increase arbitrarily we conclude that $f(a) := \frac{a^2}{a-1}$ has to be minimized. By analytic means we obtain $a = 2$ as the optimal value for minimization.

3.1.1 Generalization to m-rays

We can easily extend the problem by considering m corridors emanating from a common starting point which gives the so-called m-ray-search problem; see Figure 3.2. If we consider the situation as searching for a target point inside a polygon, we can already state that this problem is not competitive in general.

Assume that the caves at the end of the m corridors have distance 1 from the start. The agent has to look inside any cave and the adversary places the target in the last cave. This gives a path length of $2(m-1)+1$ versus the shortest path of length 1. So for any C there exists a polygon such that no strategy can guarantee a competitive ratio smaller than C. We just choose the m-corridor polygon for $C < 2(m-1)+1$. This means that the optimal competitive ratio for m rays should depend on m.

So we consider a fixed m and m rays that emanate from a common starting point s. The agent has no sight system and detects the goal only by an exact visit. We make use of the following notations. By f_j we denote the j-th step where the agent visits some ray at depth f_j. Let J_j denote the index of the *next*

visit of this ray. We require this index for describing the local worst case situation. So let (f_j, J_j) be the corresponding pairs for all j. Now the performance of a strategy $(f_1, f_2, \ldots)$ obviuously is given by

$$\sup_l \left(1 + 2\frac{\sum_{i=1}^{J_l-1} f_i}{f_l}\right).$$

In order to apply the Theorem of Gal we assume that the rays are visited in a periodic order ($J_j = j+m$) and with overall increasing depth ($f_j \leq f_{j+1}$). Figure 3.3 shows an example with some steps for $m = 4$. Now we have to minimize $F_k(f_1, f_2, \ldots) := \frac{f_k + 2\sum_{i=1}^{k+m-1} f_i}{f_k}$ over all k.

By the same arguments as before F_k fulfills the requirements of Theorem 3.2 and we conclude $\sup_k F_k(X) \geq \inf_a \sup_k F_k(A_a)$ for $A_a = a^0, a^1, a^2, \ldots$ and $a > 0$. In analogy to the 2-ray case we attain a function $f(a)$ independent from k and $f(a)$ has to be minimized. This optimization is a given as an exercise.

Exercise 23 *Minimize the functionals $F_k(f_1, f_2, \ldots) := \frac{f_k + 2\sum_{i=1}^{k+m-1} f_i}{f_k}$ and show that the (optimal) competitive ratio of the m-ray search problem for periodic and monotone strategies is $C = 1 + 2m\left(\frac{m}{m-1}\right)^{m-1}$. An optimal exponential strategy a^i for this case is given by $a = \frac{m}{m-1}$.*

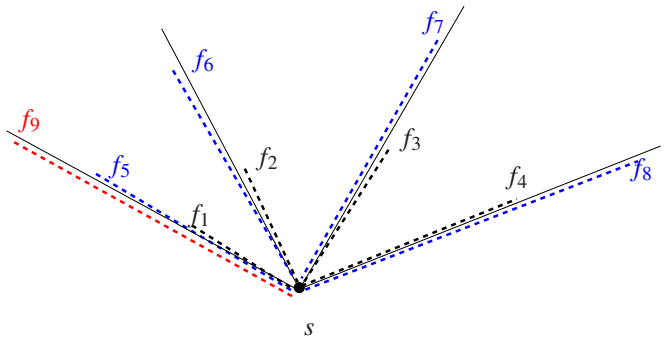


Figure 3.3: The first steps of a periodic and monotone strategy for $m = 4$ rays.

The above optimization was a simple application of Theorem 3.2. The main problem is that we assumed that there is an optimal strategy that is periodic and monotone. This is an instance of a more general problem. There are some motion planning problems where the existence of periodic and monotone optimal solutions is not known. The best upper bounds for competitive ratios are often achieved by the above assumption. Lower bounds are much harder to achieve. For the m-ray configuration the existence of periodic and monotone optimal solutions can be shown.

Lemma 3.3 *There is always an optimal competitive (with the best overall achievable ratio C) m-ray search strategy $(f_1, f_2, \ldots)$, that visits the rays in overall increasing depth and in periodic order.*

Proof. Assume that there is an arbitrary C-competitive strategy $(f_1, f_2, \ldots)$ with pairs (f_j, J_j) for the smallest attainable ratio C. First, we show that we can rearrange this strategy to a monotone C-competitive strategy $(f'_1, f'_2, \ldots)$, i.e., $f'_j \leq f'_{j+1}$ holds for all j.

Let j be the smallest index j such that $f_j > f_{j+1}$. The performance $(C-1)/2$ for the strategy $(f_1, f_2, \ldots)$ is represented in

$$\sum_{i=1}^{J_j-1} f_i \leq \frac{C-1}{2} f_j \quad : \quad \text{for index } j \tag{3.1}$$

$$\sum_{i=1}^{J_{j+1}-1} f_i \leq \frac{C-1}{2} f_{j+1} \quad : \quad \text{for index } j+1 \tag{3.2}$$

$$\sum_{i=1}^{J_l-1} f_i \leq \frac{C-1}{2} f_l \quad : \quad \text{for index } l \neq j, j+1 \tag{3.3}$$

We exchange f_j and f_{j+1} by $f'_j := f_{j+1}$ and $f'_{j+1} := f_j$. What happens to the performance above? Inequality (3.2) remains true, because we only increase the right hand side. If $J_{j+1} > J_j$ holds, also the first inequality (3.2) is maintained, because the original second inequality (for f_{j+1} on the right hand side) hold and the left hand side of inequality (3.1) is even smaller now. The remaining inequlities (3.3) are not concerned from this exchange.

The remaining task is to handle the case $J_{j+1} < J_j$. Here we have the problem of maintaining inequality (3.1). To overcome this problem we exchange the role of the rays of f_j und f_{j+1} directly after the index $j+1$ completely. After index $j+1$ any original visit of the ray of f_{j+1} is no applied to the ray of f_j and vice versa. Of course the exchange $f'_j = f_{j+1}$ and $f'_{j+1} = f_j$ is maintained. Now we do not have a problem with inequality (3.1) any more since the ray is visited early enough now. Inequality (3.2) is also maintained because we have the same next visits as before. Inequalities (3.3) do not change, they are not influenced by the exchange. In principle for $J_{j+1} < J_j$ and $f_j > f_{j+1}$ we exchange two complete rays beginning with index j.

For example if $f_1 > f_2$ holds and $J_1 = 7$ for ray K and $J_2 = 5$ for ray L, then after the exchange we visit K by $f'_1 := f_2$ then L by $f'_2 := f_1$, later K by $f'_5 := f_5$ and L by $f'_7 := f_7$ and so on. Figure 3.4 shows an example.

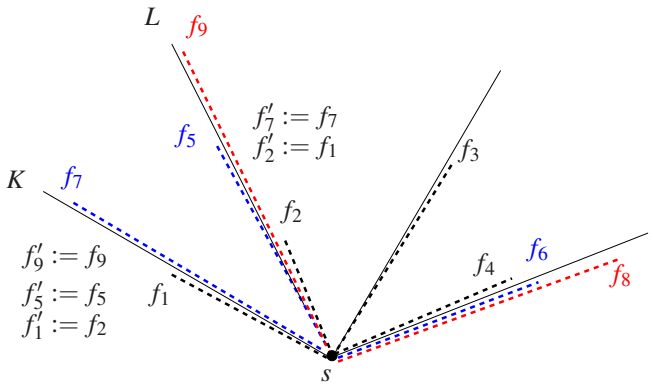


Figure 3.4: A non-periodic and non-monotone strategy. First, we exchange the values f_1 and f_2 only. But since $J_1 = 7 > J_2 = 5$ holds we fully exchange the role for the corresponding rays K and L.

Altogether, we obtain a C-competitive strategy $(f'_1, f'_2, \ldots)$ with $f'_j \leq f'_{j+1}$ for all j by applying the above exchange successively.

Finally, we construct a periodic strategy by the same idea. Consider a monotone strategy with a first index j such that $J_{j+1} < J_j$. We exchange the role of the corresponding rays after step $j+1$, which means that f_j and f_{j+1} remain on their place. Now $J'_{j+1} > J'_j$ holds. The ray with smaller f_j is visited earlier which maintains the ratio, the ray with next at visit J_j s visited later now but the original strategy maintains the ratio for the corresponding sum with f_j and we have f_{j+1} on the right hand side now. All other inequalities are not concerned.

Now after this change it might happen that some the monotonicity after step $j+1$ is no longer given. Then we apply the first rearrangement again and so on.

Altogether we obtain a monotone strategy with $J_{j+1} > J_j$ for all j and the same ratio C. Trivially, if $J_{j+1} > J_j$ holds for all j, this can only mean that $J_j = j + m$ holds for all j. □

3.1.2 Alternative approach: Equality

By Theorem 3.2 we obtained a very general approach for solving motion planning problems optimally. Somehow we can call this the *Optimality of exponential functions*. Here we would like to present an alternative. The corresponding paradigm can be denoted as *Optimality by equality*. Consider the 2-ray search problem. The optimal strategy $x_i = 2^{i-1}$ attains the competitive ratio asymptotically, but never reaches the ratio exactly. Even after the first round we attain $2x_1 + 1 \leq C \cdot 1$ and for $x_1 = 1$ and

$C = 9$ there is some room for the first step. Interestingly the strategy $x_i = (i+1)2^i$ fulfils the inequality $\sum_{i=1}^{k+1} x_i \leq \frac{C-1}{2} x_k$ by equality for all k. This can be easily shown by induction.

Exercise 24 *Show that the strategy $x_i = (i+1)2^i$ attains a competitive ratio of $C = 9$ and fulfils $\sum_{i=1}^{k+1} x_i = \frac{C-1}{2} x_k$ for all k.*

We will now show that this is not given by chance. Assume that there is an optimal C-competitive strategy for the 2-ray search problem. This means that there is a sequence $(x_1, x_2, \ldots)$ such that $\frac{\sum_{i=1}^{k+1} x_i}{x_k} \leq \frac{(C-1)}{2}$ holds for all k. In this case there is also always a strategy $(x'_1, x'_2, x'_3 \ldots)$ with $\frac{\sum_{i=1}^{k+1} x'_i}{x'_k} = \frac{(C-1)}{2}$ for all k. The proof for this statement works as follows: The functional $\frac{\sum_{i=1}^{k+1} x_i}{x_k}$ is strictly monotonically decreasing for x_k. If we increase x_k the functional decreases. On the other hand the $\frac{\sum_{i=1}^{k+1} x_i}{x_k}$ is strictly increasing for all x_j with $j \neq k$. This means that increasing x_j for $j \neq k$ will increase the functional.

We can assume that $x_i \geq 0$ holds. Now assume that there is a first index k such that inequality holds, i.e.: $\frac{\sum_{i=1}^{k+1} x_i}{x_k} < \frac{(C-1)}{2}$. If this holds already for the first index $k = 0$ with $x_0 := 1$ we simply decrease x_0 such that $x_1 + 1 = \frac{(C-1)}{2} x_0$ holds. Now consider $k > 0$ as the smallest index with $\frac{\sum_{i=1}^{k+1} x_i}{x_k} < \frac{(C-1)}{2}$. We decrease x_k in such a ways that $\frac{\sum_{i=1}^{k+1} x_i}{x_k} = \frac{(C-1)}{2}$ is given, which is possible in this case. All other inequalities remain valid. Obviously, we will attain $\frac{\sum_{i=1}^{k} x_i}{x_{k-1}} < \frac{(C-1)}{2}$ for the index $k-1$ and we will proceed for $k-1$ by the same argument until finally we reach $k = 0$ again. Note that by $x_1 + 1 = \frac{(C-1)}{2} x_0$, x_0 cannot decrease to 0. Thus for any such first index k we attain a monotonically decreasing sequence $(x_0, x_1, \ldots, x_k)$ that is bounded. This means that the above procedure will always converge to a sequence $(x_0, x_1, \ldots, x_k)$ such that $\frac{\sum_{i=1}^{j+1} x_i}{x_j} = \frac{(C-1)}{2}$ for $j = 0$ to k. This holds for any k. This means that for infinitely many steps there always exists a strategy that achieves equality in any step. Finally, we make use of a scalar A such that $A \cdot x_0 = 1$ holds and $(1, A \cdot x_1, \ldots, A \cdot x_k)$ is the desired sub-strategy.

The above arguments show that such a strategy exists for any k but the procedure is not constructive. Let us now assume that $\frac{\sum_{i=1}^{k+1} x_i}{x_k} = \frac{(C-1)}{2}$ holds for all k, we conclude $x_{k+1} = \sum_{i=1}^{k+1} x_i - \sum_{i=1}^{k} x_i = \frac{(C-1)}{2}(x_k - x_{k-1})$ and we are searching for the solution of the recurrence $\frac{(C-1)}{2}(x_k - x_{k-1}) = x_{k+1}$. We set $x_1 = \frac{(C-1)}{2} = \frac{(C-1)}{2}(x_0 - x_{-1})$ with starting values $x_0 = 1$ and $x_{-1} = 0$. The task is to find the smallest value for C such that the above recurrence attains a reasonable solution for the 2-ray search problem. This means that we will have a look at the methods for finding the solutions of recurrences. Here we concentrate on the methods proposed by [GKP98] for finding the a closed expression for any x_k. This method shows that $C \geq 9$ is required, which gives a second proof for the optimal ratio 9.

For solving a recurrence as shown in [GKP98] we can perform 4 steps. The general correctness of the method is proved in [GKP98].

A) Closed Form: We bring $\frac{(C-1)}{2}(x_k - x_{k-1}) = x_{k+1}$ with $x_{-1} := 0$ and $x_0 = 1$ into a closed formula that also holds for the starting values. For comparison to [GKP98] we use the notation g instead of x and set $D := \frac{(C-1)}{2}$. We have $g_0 = 0$, $g_1 = 1$ und $g_n = D(g_{n-1} - g_{n-2})$. By $[n = l]$ we denote a serie that has value 1 for $n = l$ and value 0 for all other n. We assume $g_n = 0$ for negative n. Thus a closed formula is given by $g_n = Dg_{n-1} - Dg_{n-2} + 1 \cdot [n = 1]$.

B) Building a power serie with coefficients g_n: We consider the power serie $G(z) := \sum_n g_n z^n$. Inserting the closed form of the preceding paragraph we have:

$$\begin{aligned}
\sum_n g_n z^n &= D\sum_n g_{n-1} z^n - D\sum_n g_{n-2} z^n + \sum_n [n=1] z^n \\
&= D\sum_n g_n z^{n+1} - D\sum_n g_n z^{n+2} + z \\
&= DzG(z) - Dz^2 G(z) + z.
\end{aligned}$$

C) Closed form for power serie $G(z)$: From (B) we conclude $G(z) = \frac{z}{1-Dz+Dz^2}$.

D) Developing the power serie $G(z)$: The remaining task is is to make use of (C) for the precise development of the power serie. We will sketch the procedure presented in [GKP98]. In general we have $G(z) = \frac{P(z)}{Q(z)}$. In our special case we conclude $P(z) = z$ and $Q(z) = 1 - Dz + Dz^2$. By function theory arguments a serie for $G(z)$ with the precise values for g_n is constructed; details are given in [GKP98]. The construction is based on the zeros of $Q(z)$. The main argument is that the zeros of $G(z)$ has to be real in order to obtain reasonable expressions for g_n. The overall conclusion that the zeros of $Q(z)$ are given by $z_{1,2} = \frac{1}{2} \pm \sqrt{\frac{1}{4} - \frac{1}{D}}$. The radicant is negative for $D < 4$ and there are no real-valued solutions in this case. The conclusion is that only $D \geq 4$ (or $C \geq 9$, respectively) guarantees reasonable values for g_n.

The details of calculating g_n precisley from the zeros of $G(z)$ are given in [GKP98]. We would like to present at least the calculations for the recurrence above for $D = 4$. First, $Q(z)$ is expressed by $Q(z) = q_0(1 - p_1 z)^{d_1} \cdots (1 - p_l z)^{d_l}$ where $\frac{1}{p_i}$ is a zero of Q of order d_i. In our example we have $Q(z) = (1-2z)^2$ with $q_0 = 1$, $p_1 = 2$ and $d_1 = 2$.

Now the coefficients g_n of $G(z)$ are given by $f_1(n)p_1^n + \cdots f_l(n)p_l^n$ where $f_i(n)$ is a polynomial of degree $d_i - 1$. In our example $f_1(n)$ has degree $d_1 - 1 = 1$ and the coefficient g_n of $G(z)$ is $f_1(n)2^n$.

Additionally, for the polynomial $f_i(n)$ of degree $d_i - 1$, the leading coefficient a_i is presented by

$$a_i = \frac{P\left(\frac{1}{p_i}\right)}{(d_i - 1)! q_0 \prod_{j \neq i} \left(1 - \frac{p_j}{p_i}\right)}.$$

In our special case we have $f_1(n) = a_1 n + a_0$ with $a_1 := \frac{P\left(\frac{1}{2}\right)}{(2-1)!1} = \frac{1}{2}$. Now we have $g_n = \left(\frac{1}{2}n + a_0\right) 2^n$. The remaining task is to determine a_0. This can be done by the starting values $g_1 = 1$ or $g_0 = 0$. For $1 = g_1 = \left(\frac{1}{2} + c\right) 2 = 1 + 2a_0$ we have $a_0 = 0$, the same holds for $g_0 = 0$. Therfore we conclude $g_n = n2^{n-1}$ which is exactly the above presented solution, which attained equality in any step. The competitive ratio is 9.

Altogether, in this section we have developed different methods for solving discrete motion planning problems by functionals.

3.1.3 2-ray search with bounded distance

Let us assume that in the beginning the maximal distance D from start to target point is given. This means that the rays are bounded by length D. We consider the 2-ray search problem; see Figure 3.5.

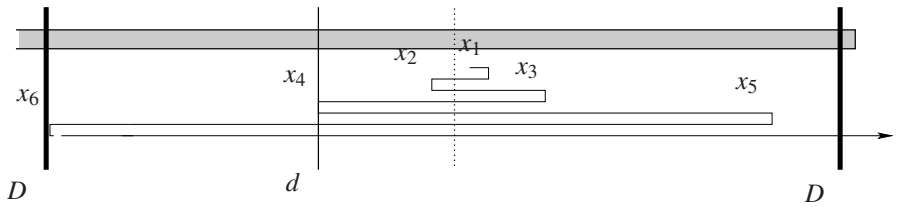


Figure 3.5: Falls wir wissen, dass das Ziel in einer Distanz D liegt, können wir die Strategie optimieren.

If the goal has precisely distance D from the start, the agents runs distance D to one side, back to the start and distance D to the opposite side. In the worst-case this gives a ratio of 3 which is optimal. Now let us assume that the goal is inside an interval $[1, D]$ away from the start, we would like to minimize the ratio $C = 9$.

Obviously, the optimal strategy checks precisely distance $x_k = D$ in the second last step on one side and attains the ratio for the last depth $x_{k-1} < D$ on the opposite side. The step x_{k+1} can be arbitrarily long, it will not decrease the ratio any more; see Figure 3.5. Because of these properties we can also ask

for the opposite question. Assume that a ratio $C < 9$ is given. What is the maximal depth D on both side, so that any goal in the distance interval $[1,D]$ away from the start will be found with ratio C.

More precisely, we would like to maintain the ratio $C < 9$ and maximize the second last step x_k. Since $C = 9$ is the overall optimal factor, for $C < 9$ we cannot guarantee factor C for distances up t ∞ or $-\infty$.

In [HIKL99] it was shown, that for $C < 9$ the maximal reach will be attained for a strategy that attains equality in any worst-case step. We can conclude $\frac{\sum_{i=1}^{j+1} x_i}{x_j} = \frac{(C-1)}{2}$ for $j = 0,1,\ldots,k-1$ holds, where $x_0 = 1$ and x_k is the maximal depth.

The choice of $x_0 = 1$ stem from the fact, that the goal is at least one step away from the start. If there is a sub-strategy $(x_1,x_2,\ldots,x_k)$ such that $x_1 < \frac{(C-1)}{2}$ holds. We can simply use a scalar $A > 1$ such that $Ax_1 = \frac{(C-1)}{2}$. The strategy $(Ax_1,Ax_2,\ldots,Ax_k)$ is C-competitive and has larger depth Ax_k. The argumentation that any inequality has to be fulfilled by equality is given in the proof below.

Let us have a look at the results: For $C = 6$ we conclude: $x_1 = 2.5$, $x_2 = 2.5(2.5-1) = 3.75$, $x_3 = 2.5(3.75-2.5) = 3.125 < x_2$, $x_4 = 2.5(x_3 - x_2) < 0$. This means that k is 2 and the strategy attains optimal reach 3.75, the worst-case is attained for $x_0 = 1$ and $x_1 = 2.5$. For $C = 7$ we obtain the strategy: $x_1 = 3$, $x_2 = 3(3-1) = 6$, $x_3 = 3(6-3) = 9$, $x_4 = 3(9-6) = 9$, $x_5 = 3(9-9) < x_3$. We have reach 9 and k equals 4. The worst-case for the ratio is attained at $x_0 = 1$, $x_1 = 3$, $x_2 = 6$ and $x_3 = 9$.

Theorem 3.4 *Let $C < 9$ be the given factor for the 2-ray search problem. For the maximal reach problem there is always an optimal strategy that attains equality in any step. [HIKL99]*

Proof. We would like to develop an alternative proof. Let us assume that there is a strategy that attains the maximal reach for given $C < 9$. The goal is at least one step away from the start. We have a sequence $(x_1,x_2,\ldots,x_k)$ such that $\frac{\sum_{i=1}^{j+1} x_i}{x_j} \leq \frac{(C-1)}{2}$ for $j = 0,1,\ldots,k-1$ holds with $x_0 := 1$ and x_k is the maximal depth.

Consider the first index j such that $\frac{\sum_{i=1}^{j+1} x_i}{x_j} < \frac{(C-1)}{2}$ holds. For $j = 0$ we have $x_1 < \frac{(C-1)}{2}$ holds. We can simply use a scalar $A_1 > 1$ such that $A_1x_1 = \frac{(C-1)}{2}$. The strategy $(A_1x_1,A_1x_2,\ldots,A_1x_k)$ is C-competitive and has larger depth Ax_k. In general let us assume that for $j > 0$ we have $\sum_{i=1}^{j+1} x_i < \frac{(C-1)}{2}x_j$. We enlarge x_{j+1} by factor A_{j+1} such that $\sum_{i=1}^{j} x_i + A_{j+1}x_{j+1} = \frac{(C-1)}{2}x_j$ holds. We exchange the current sequence by $(x_1,x_2,\ldots,x_j,A_{j+1}x_{j+1},A_{j+1}x_{j+2},\ldots)$ and still guarantee $\frac{\sum_{i=1}^{j+1} x'_i}{x'_j} \leq \frac{(C-1)}{2}$ for $j = 0,1,\ldots,k-1$. We have reach $x'_k = A_{j+1}x_k > x_k$ which gives the contradiction. □

Figure 3.6 shows the curve of the function f, where $f(C)$ is the maximal reach that can be attained for C. For any kink in the curve the strategy makes another turn. The number of turns increases. For $C = 7$ we have $k = 3$ turns and for $C = 6$ we have $k = 2$ turns!

The above function f is strictly monotonically increasing in C. This means that by binary search we can easily compute the best ratio C for given reach R. The corresponding reverse function is shown in Figure 3.7.

The above equality-paradigm can also be used for the m-ray search problem, if the given depth is the same on each ray; see [Sch01, Lan00]. For different intervals on the rays up to now no efficient optimization technique is known. Only for some few rays (< 4) a master thesis shows some results; see [Web07].

3.2 Searching for a ray in the plane

In this section, we consider the search for the origin, t, of a ray R in the plane, see Figure 3.8. The searcher has no vision, but recognizes the ray and the ray's origin as soon as the searcher hits the ray. The position of the ray is not known in advance. The searcher moves along a path, Π, starting at a given point, s. Eventually, Π will hit the ray R at a point p and the origin t is detected. The cost of the strategy is given by the length of the path from s to p (i.e., $|\Pi_s^p|$), plus the distance $|pt|$ from p to t. We measure the

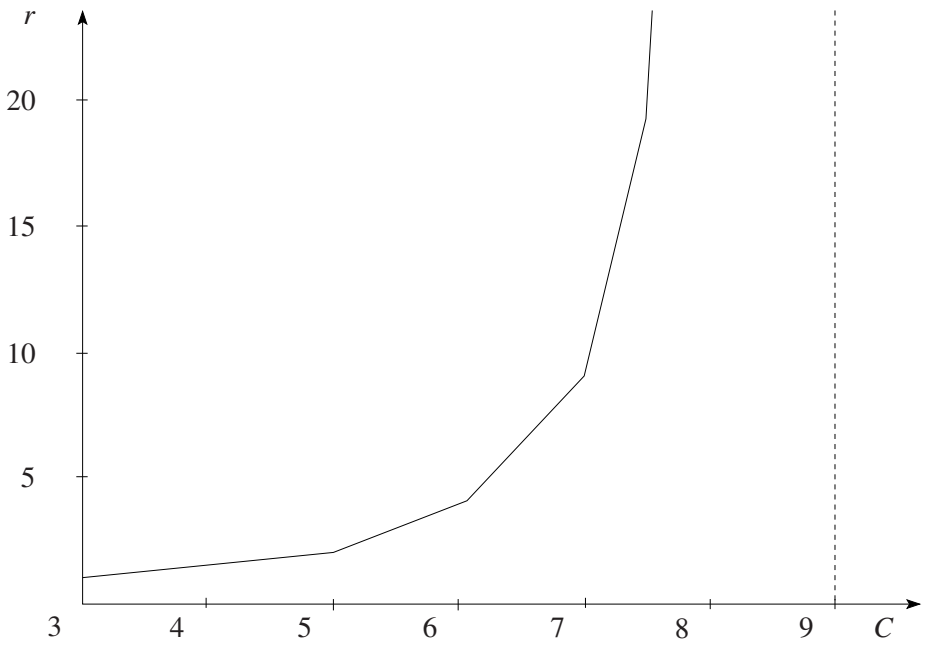


Figure 3.6: Maximal reach depending on the ratio $C < 9$.

quality of the path Π for the ray R using the competitive ratio $\frac{|\Pi_s^p|+|pt|}{|st|}$; that is, we compare the length of the searcher's path to the shortest path from s to t. We would like to find a search path Π that guarantees a competitive ratio not greater than C for all possible rays R in the plane. In turn, C should be as small as possible. Similar problems were discussed by Alpern and Gal [AG03]; for example, searching for an unknown line in the plane.

3.2.1 The Window Shopper Problem

First, we consider the problem of finding a gift along a shopping window. The agent starts somewhere and looks toward the window. We assume that the item, t, gets into sight if the ray, R, from t to the seachers position, p, is perpendicular to the window. Then the searcher moves toward t.

This problem can be modelled as follows. W.l.o.g. we assume that the line of sight (i.e., the ray, R, we are looking for), is parallel to the X-axis, starts in $(1, y_R)$ for $y_R \geq 0$, and emanates toward the left side of the perpendicular ray R' (the *window*) which starts in $(1,0)$. The searcher starts in the origin $s = (0,0)$; see Figure 3.9. The goal (i.e., the ray's origin t) is discovered as soon as the searcher reaches its height, y_R. After the searcher has discovered the goal, it moves directly to the goal. Note that the shortest distance from s to R' can be fixed to 1 because scaling has no influence on the competitive ratio.

We would like to find a search path, Π, so that for any goal, t, the ratio $\frac{|\Pi_s^p|+|pt|}{|st|} \leq C$ holds, where C is the smallest achievable ratio for all search paths.

Theorem 3.5 *There is a strategy Π with an optimal competitive factor of $1.059\ldots$ for searching the origin of a ray, R, that emanates from a known ray R' perpendicular to R.* *[EFK+06]*

Proof. Apparently a good search path moves simultaneously along and towards the wall; that is, in positive X- and Y-direction. Note that the competitive factor for any reasonable strategy converges to 1 for goals with very small Y-coordinate and also for goals with a large Y-coordinate. Therefore, the first part of our path, Π, is a line segment up to a point (a,b). The second part is a curve, $f(x)$, that converges

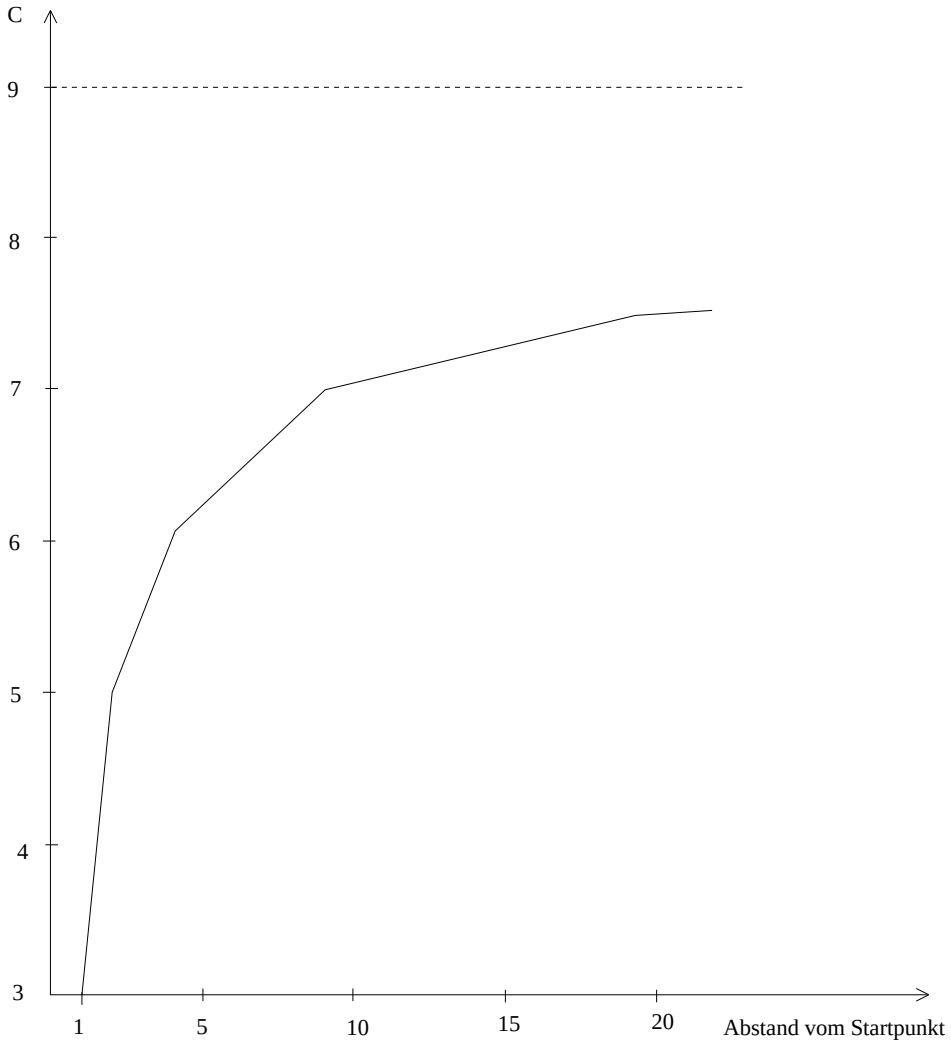


Figure 3.7: Optimal competitive ratio for given reach.

to the wall and maintains the competitive factor that was achieved by the line segment in the first part of the search; see Figure 3.9.

Thus, we solve two tasks.

1. We will design a search path Π that consists of the following three parts (or conditions); see Figure 3.10(i).

   Π_1: A straight line segment from $(0,0)$ to some point (a,b) where the competitive ratio strictly increases from $C = 1$ to $C_{\max}$ for goals from $(1,0)$ to $(1,b)$.

   Π_2: A strictly monotone curve f from (a,b) to some point $(1,D)$ on R' where the competitive ratio is exactly $C_{\max}$ for all goals from $(1,b)$ to $(1,D)$.

   Π_3: A ray starting from $(1,D)$ to $(1,\infty)$ where the competitive ratio strictly decreases from $C_{\max}$ to 1 for goals from $(1,D)$ to $(1,\infty)$.

   Furthermore, we prove that the full path Π is convex. The competitive ratio of Π is $C_{\max}$.

2. We will show that such a path is optimal and the best achievable ratio is $C_{\max}$.

We start with the second task. Let us assume that we have designed a search path Π with the given properties and let us assume that there is an optimal search path K with $K \neq \Pi$, see Figure 3.10(ii).

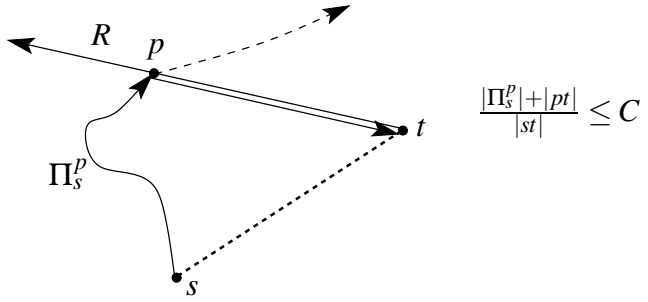


Figure 3.8: Die Suche nach dem Ursprung t eines Strahles R.

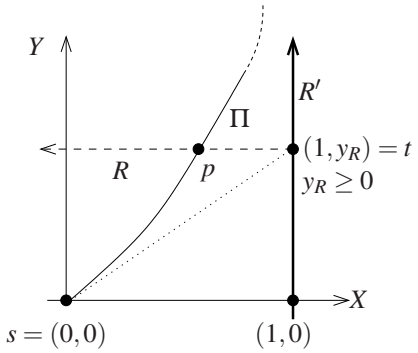


Figure 3.9: A strategy for the window shopper!

The path K might hit the ray B from $(1,b)$ to $(-\infty,b)$ at a point p_1 to the left of (a,b). Then the ratio $\frac{|K_s^{p_1}|+|p_1(1,b)|}{|s(1,b)|}$ is bigger than $C_{\max} = \frac{|s(a,b)|+|(a,b)(1,b)|}{|s(1,b)|}$. On the other hand, K might move to the right of (a,b) and hits Π_2 at a point p_2 between B and the ray D from $(1,D)$ to $(-\infty,D)$. In this case, the length of $K_s^{p_2}$ has to be bigger than $|\Pi_s^{p_2}|$ because Π is fully convex. Thus, the ratio $\frac{|K_s^{p_2}|+|p_2(1,p_{2_y})|}{|s(1,p_{2_y})|}$ is bigger than $C_{\max} = \frac{|\Pi_s^{p_2}|+|p_2(1,p_{2_y})|}{|s(1,p_{2_y})|}$, where p_{2_y} denotes the Y-coordinate of p_2. This also holds if K hits R' first and p_2 is equal to $(1,D)$; see the dotted path in Figure 3.10(ii).

This means that K has to follow Π from s up to some point beyond B and might leave Π_2 then. In this case K has at least the ratio $C_{\max}$ and Π is optimal, too.

It remains to show that we can design a path with the given properties. As already mentioned, the motivation for the construction is the following: In the very beginning the ratio starts from 1 and has to increase for a while, this is true for any strategy. Additionally, any reasonable strategy should be monotone in x and y. Moving backwards or away from the window will allow shortcuts with a smaller ratio. Therefore it is reasonable that we will get closer and closer to the window R' and the factor should decrease to 1. So, finally, we can hit R' because at the end the ratio will not be the worst case. Furthermore, in many applications strategies are designed by the fact that they achieve exactly the same factor for a set of goals. Altogether, we would like to design a path Π by the properties formulated above, and — as we already know — such a strategy is optimal.

With the first two conditions for Π_1 and Π_2 we determine a and b. We consider the line segment from the origin $(0,0)$ to (a,b) with $a,b>0$ to be parametrized by (ta,tb) for $t \in [0,1]$. The competitive factor for Π_1 is given by

$$C(t) = \frac{t\sqrt{a^2+b^2}+1-ta}{\sqrt{1+t^2b^2}}, \quad t \in [0,1].$$

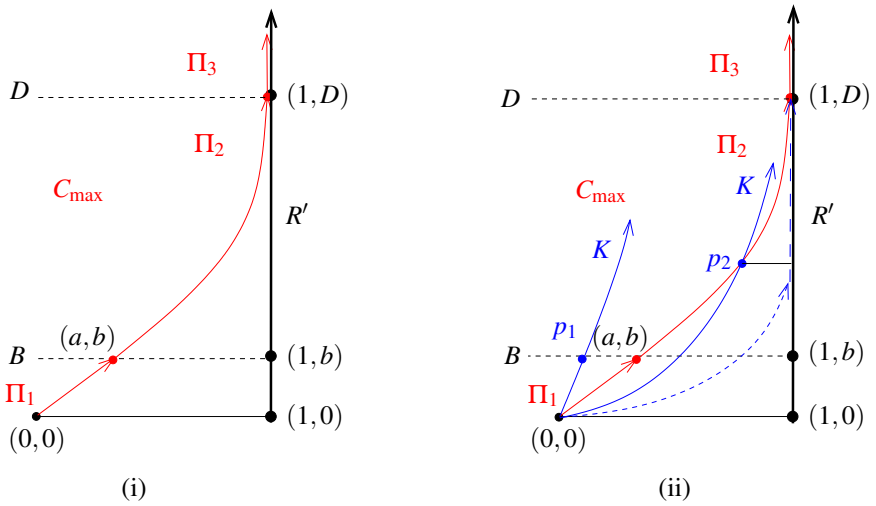


Figure 3.10: An arbitrary search path K is not better than Π.

We want $C(t)$ to be a monotone and increasing function. From $C'(t) \geq 0 \; \forall t \in [0,1]$ we conclude

$$\begin{aligned} C'(t) &= \frac{(\sqrt{a^2+b^2}-a)(1+t^2b^2)-(t(\sqrt{a^2+b^2}-a)+1)tb^2}{\sqrt{1+t^2b^2}(1+t^2b^2)} \geq 0 \quad \forall t \in [0,1] \\ &\Leftrightarrow \quad \sqrt{a^2+b^2}-a \geq tb^2 \quad \forall t \in [0,1] \\ &\Leftrightarrow \quad \sqrt{a^2+b^2}-a \geq b^2 \\ &\Leftrightarrow \quad a^2+b^2 \geq b^4+2ab^2+a^2 \\ &\Leftrightarrow \quad 1-2a \geq b^2 . \end{aligned}$$

Hence, $a \leq \frac{1-b^2}{2}$ follows. From now on we set $a := \frac{1-b^2}{2}$. For $t = 1$ and $a := \frac{1-b^2}{2}$ we obtain a competitive factor of

$$\begin{aligned} \frac{\sqrt{a^2+b^2}+1-a}{\sqrt{1+b^2}} &= \frac{\sqrt{(\frac{1-b^2}{2})^2+b^2}+1-\frac{1-b^2}{2}}{\sqrt{1+b^2}} = \frac{\sqrt{\frac{1-2b^2+b^4+4b^2}{4}}+\frac{1}{2}+\frac{b^2}{2}}{\sqrt{1+b^2}} \\ &= \frac{\sqrt{(\frac{1+b^2}{2})^2}+\frac{1}{2}(1+b^2)}{\sqrt{1+b^2}} = \sqrt{1+b^2} =: C . \end{aligned} \tag{3.4}$$

We can consider the line segment Π_1 also as a function of $x \in [0,a]$. Now, C is the worst case competitive factor for $x \in [0,a]$ and goals t between $[1,0]$ and $[1,b]$.

For Π_2 we construct a curve $f(x)$ for $x \in [a,1]$ that runs from $[a,b]$ to some point $[1,D]$ and achieves the ratio $C = \sqrt{1+b^2}$ for all goals t between $[1,b]$ and $[1,D]$. This means that the length of the path of the searcher (i.e., the line segment up to (a,b), the part of the curve f up to the height y_R, and the final line segment to the goal $(1,y_R)$) is equal to C times the Euclidean distance from the origin $(0,0)$ to the goal $(1,y_R)$. Thus, f can be defined by the differential equation

$$\sqrt{a^2+b^2}+1-x+\int_a^x \sqrt{1+f'(t)^2}\,dt = C \cdot \sqrt{1+f(x)^2}. \tag{3.5}$$

We would like to rearrange equation (3.5) in order to apply standard methods for solving differential equations. Derivating equation (3.5) and squaring twice gives

$$\begin{aligned} &\sqrt{1+f'(x)^2}-1 = \frac{C}{2} \cdot \frac{1}{\sqrt{1+f(x)^2}} \cdot 2f(x)f'(x) \\ \Leftrightarrow \quad &1+f'(x)^2-2\sqrt{1+f'(x)^2}+1 = C^2\,\frac{f(x)^2 f'(x)^2}{1+f(x)^2} \end{aligned}$$

$$\Leftrightarrow \quad f'(x)^2\left[1-C^2\frac{f(x)^2}{1+f(x)^2}\right]+2=2\sqrt{1+f'(x)^2}$$
$$\Leftrightarrow \quad f'(x)^4\left[1-C^2\frac{f(x)^2}{1+f(x)^2}\right]^2+4f'(x)^2\left[1-C^2\frac{f(x)^2}{1+f(x)^2}\right]=4f'(x)^2.$$

The curve f was assumed to be strictly monotone, which means $f'(x)\neq 0$. Therefore we have

$$\begin{aligned}
&\Leftrightarrow \quad f'(x)^2\left[1-C^2\frac{f(x)^2}{1+f(x)^2}\right]^2=4C^2\frac{f(x)^2}{1+f(x)^2}\\
&\Leftrightarrow \quad f'(x)^2=\left[\frac{1+f(x)^2}{1+(1-C^2)f(x)^2}\right]^2 4C^2\,\frac{f(x)^2}{1+f(x)^2}\\
&\Leftrightarrow \quad f'(x)^2=4C^2\frac{(1+f(x)^2)f(x)^2}{(1+(1-C^2)f(x)^2)^2}\\
&\Leftrightarrow \quad f'(x)=2C\frac{\sqrt{1+f(x)^2}f(x)}{1+(1-C^2)f(x)^2}.
\end{aligned}\tag{3.6}$$

Note that the point $(a,b)=(\frac{1-b^2}{2},b)$ lies on f and C is equal to $\sqrt{1+b^2}$. Altogether, we have to solve the differential equation

$$y'=1\cdot 2\sqrt{1+b^2}\frac{\sqrt{1+y^2}y}{1-b^2y^2}=1\cdot g(y)\tag{3.7}$$

for $y=f(x)$ with starting point $(\frac{1-b^2}{2},b)$.

Equation (3.7) is a first order differential equation $y'=h(x)g(y)$ with separated variables and point (k,l) on y. A general solution is given by

$$\int_l^y \frac{dt}{g(t)}=\int_k^x h(z)dz;$$

see Walter [Wal86]. Thus, we have to solve

$$\int_b^y \frac{1-b^2t^2}{2\sqrt{1+b^2}\sqrt{1+t^2}t}dt=\int_{(1-b^2)/2}^x 1\cdot dz=x-(1-b^2)/2$$

By simple analysis, we obtain

$$x=-\frac{b^2\sqrt{1+y^2}+\operatorname{arctanh}\left(\frac{1}{\sqrt{1+y^2}}\right)-\operatorname{arctanh}\left(\frac{1}{\sqrt{1+b^2}}\right)-\sqrt{1+b^2}}{2\sqrt{1+b^2}}$$

which is the solution for the inverse function $x=f^{-1}(y)$. By simple analysis we get

$$x'=\frac{1}{g(y)}=-\frac{(b^2y^2-1)}{2\sqrt{1+y^2}y\sqrt{(1+b^2)}}\geq 0 \text{ for } y\in[0,1/b]$$

and

$$x''=-\frac{(b^2y^2+2y^2+1)}{2(1+y^2)^{3/2}\sqrt{1+b^2}y^2}\leq 0 \text{ for } y\geq 0,.$$

Because $x=f^{-1}(y)$ is concave in the given interval, $y=f(x)$ is convex. Additionally, f^{-1} attains a maximum for $y=\frac{1}{b}$.

Altogether we have a situation for the inverse function $x=f^{-1}(y)$ for $y\in\left[0,\frac{1}{b}\right]$ as shown in Figure 3.11.

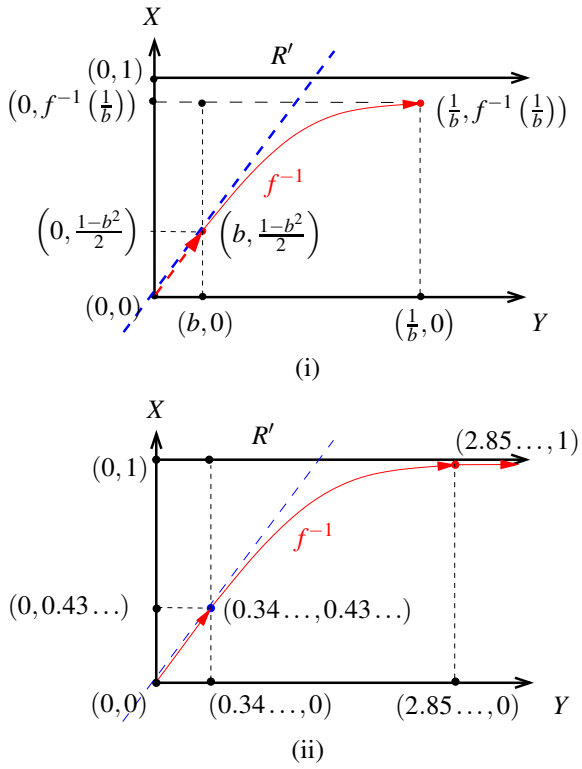


Figure 3.11: The inverse situation of the window shopper problem. The curve f^{-1} should hit the line $X = 1$.

Now, we have to find a value for b so that $f^{-1}(\frac{1}{b})$ is equal to 1, so that f^{-1} behaves as depicted in Figure 3.11(ii). That is, we have to find a solution for

$$1 = -\frac{b^2\sqrt{1+\frac{1}{b^2}} + \operatorname{arctanh}\left(\frac{1}{\sqrt{1+\frac{1}{b^2}}}\right) - \operatorname{arctanh}\left(\frac{1}{\sqrt{1+b^2}}\right) - \sqrt{1+b^2}}{2\sqrt{1+b^2}}. \tag{3.8}$$

This fixes b and, in turn, D to $\frac{1}{b}$. Note that in this case $y = f(x)$ has the desired properties for $x \in [a,1] = \left[\frac{1-b^2}{2},1\right]$.

We have already seen that $y = f(x)$ is convex for $x \in [a,1]$. Additionally, the line segment from $(0,0)$ to (a,b) is convex. To show that the conjunction of both elements is also convex, we have to show that the tangent to f at (a,b) is equal to a prolongation of the line segment; see Figure 3.11. In other words, we have to show $f^{-1\prime}(b) = \frac{a}{b} = \frac{1-b^2}{2b}$. This is equivalent to $\frac{1}{g(b)} = \frac{1-b^2}{2b}$ which is obviously true.

By solving equation (3.8) numerically, we get $b = 0.34\ldots$ This gives $D = \frac{1}{b} = 2.859\ldots$, $a = \frac{1-b^2}{2} = 0.43\ldots$ and a worst-case ratio $C = \sqrt{1+b^2} = 1.05948\ldots$ The corresponding curve f^{-1} is shown in Figure Figure 3.11(ii).

Altogether, by combining Π_1 (the line segment), Π_2 (the constructed curve f), and Π_3 (the ray from $(1,D)$ to $(1,\infty)$), we obtain a convex curve with the given properties and an optimal competitive factor of $C = \sqrt{1+b^2} = 1.05948\ldots$ □

3.2.2 General rays in the plane

Now we turn over to arbitrary rays in the plane. We will first show that a *logarithmic spiral* is an appropriate competitive strategy, finally we will construct a lower bound. A logarithmic spiral is defined in polar coordinates by $(\varphi, d \cdot e^{\varphi\cot(\alpha)})$ for $d > 0$ and $-\infty < \varphi < \infty$ see Figure 3.12 for an example. Note, that we can scale so that $d = 1$.

A logarithmic spiral has some nice properties. The center point of the spiral is given by the origin $s = (0,0)$. The angle α expresses the excentricity of the spiral. For every point p on the spiral the line through p and s and the tangent T_p on p build the same angle α. For $\alpha = \pi/2$ the spirals degenerates to a circle. For two points a and b on the spiral, the length of the spiral between a and b is given by $\frac{1}{|\cos\alpha|}(|bs| - |as|)$ for $|as| > |bs|$. This means that the length of the spiral from the center to some point b is given by $\frac{1}{|\cos\alpha|}|bs|$, for details see also [BSMM00].

The spiral expands successively and will finally hit every ray in the plane. Obviously, the worst case is attained for a tangent to the spiral. In the following we denote a spiral path by Π and the corresponding ratio by C_Π.

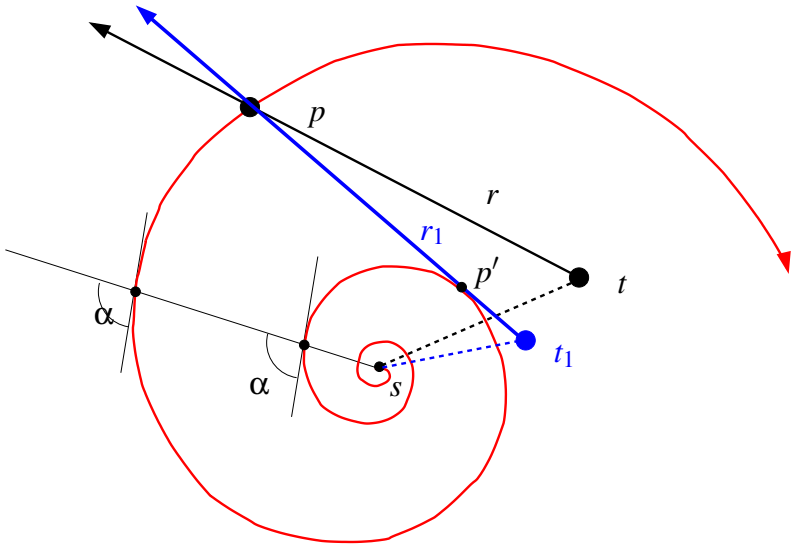


Figure 3.12: A logarithmic spiral is defined by an angle α. A tangent to the spiral will maximize the ratio.

Lemma 3.6 *Given a logarithmic spiral Π, the ray that maximizes the ratio C_Π is a tangent T to the spiral.*

Proof. Consider a ray r emanating from the point t, and the first intersection p with the spiral; see Figure 3.12. We can increase the ratio C_Π by rotating r counterclockwise around p until the ray is almost a tangent to the spiral. Additionally, t_1 gets closer to s. Note, that p' in Figure 3.12 is not actually an intersection, but the searcher moving on the spiral slightly misses the ray r_1 in p', but detects the ray in p. However, p' is arbitrarily close to the spiral; thus, we consider p' to be a point on the spiral. We call p' *tangent point*. □

Now we will proceed as follows, we will compute an optimal spiral Π given by an optimal angle α for the *orthogonal points*, q', on the tangent T_q, see Figure 3.13. Fortunately, the given ratio $C_\Pi(\alpha)$ is the same for all tangents! Afterwards, an adversary strategy can move the starting point of the ray along the tangent in order to maximize the ratio, see Figure 3.13. This means that the adversary strategy can choose the angle β.

By the law of sine we have $|st| = \frac{|sq'|}{\sin\beta}$ and $\frac{|tq'|}{\cos\beta} = |ts|$. Now we have $\frac{|\pi_s^p|+|pq'|}{|sq'|} = C_\Pi(\alpha)$ (the ratio for q') and $\frac{|\pi_s^p|+|pq'|+|q't|}{|st|} =: C_\Pi(\alpha,\beta)$ (the ratio for t). Substituting the above dependencies we have: $C_\Pi(\alpha,\beta) = C_\Pi(\alpha)\sin(\beta)+\cos(\beta)$. Altogether, we first will minimize $C_\Pi(\alpha)$ over α and then we maximize $C_\Pi(\alpha,\beta)$ over β.

Lemma 3.7 *We can minimize the ratio for the closest point, q', on a tangent T_q by choosing an optimal angle α. If we attain a ratio $C_\Pi(\alpha)$ for the orgin q', an adversary can move the starting point along the tangent which is determined by an angle β. The ratio will be given by $C_\Pi(\alpha,\beta) := C_\Pi(\alpha)\sin(\beta)+\cos(\beta)$.*

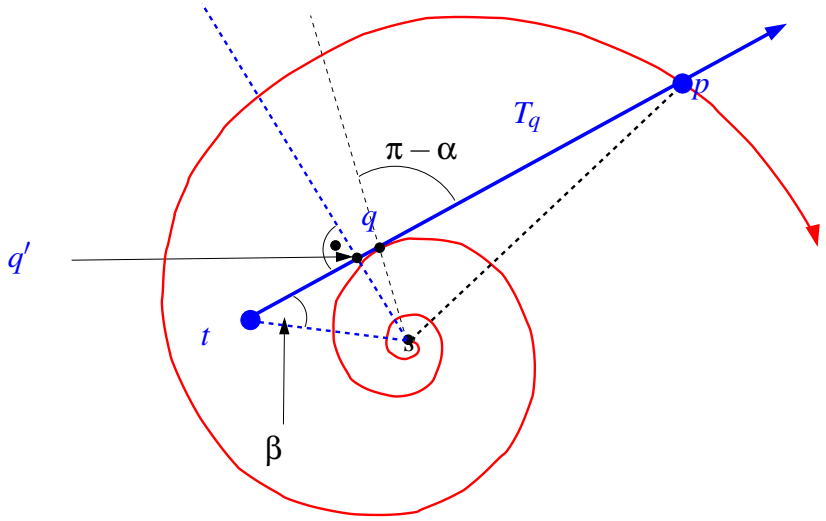


Figure 3.13: We would like to optimize the spiral for the closest point, q', from s on a tangent T_q.

Note, that it makes no sense for the adversary to move the starting point to the *right* of q', the ratio will obviously decrease.

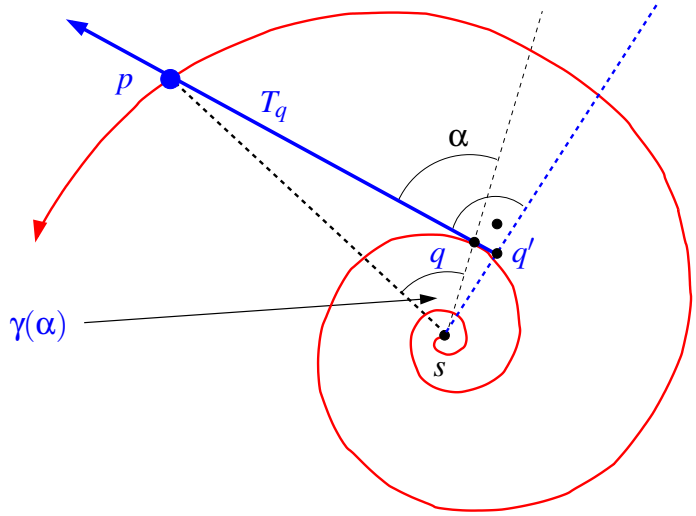


Figure 3.14: Let q' be the point on T_q with shortest distance to s. If the angle $\gamma(\alpha)$ in $\triangle qsp$ is given, we can determine the ratio for q'.

Now, we would like to compute the distance $|qq'|$. In the following we consider $\alpha < \pi/2$ and the spiral turns counterclockwise, see Figure 3.14. This has the advantage that the angles are positive in the mathematical sense. Let $\gamma(\alpha)$ denote the angle between sq and sp, see Figure 3.14. For $q := (\varphi_q, e^{\varphi_q \cot\alpha})$, we have $p := (\varphi_q + 2\pi + \gamma(\alpha), e^{(\varphi_q + 2\pi + \gamma(\alpha))\cot\alpha})$. The angle $\gamma(\alpha)$ can be determined by an equation. The proof of Lemma 3.8 is a simple exercise.

Note, that a line running through $q' = (\varphi_{q'}, r_{q'})$ and perpendicular to a line with angle $\varphi_{q'}$ is given in polar coordinates by $(\varphi, \frac{r_{q'}}{\cos(\varphi - \varphi_{q'})})$. In our case the tangent T_q is perpendicular to the line given by $\varphi_{q'}$ and runs through q'. In turn the angle $\varphi_{q'}$ is given by $\varphi_q - \pi/2 + \alpha$.

Lemma 3.8 *The angle $\gamma(\alpha) := \angle qsp$ is given by the solution to*

$$\frac{\sin\alpha}{\sin(\alpha - \gamma(\alpha))} = \mathcal{E}^{\cot\alpha(2\pi + \gamma(\alpha))}.$$

Lemma 3.8 gives us a formula for computing $\gamma(\alpha)$ at least numerically for every α. Therefore we will be able to compute the best spiral for a tangent T_q on q.

Theorem 3.9 *Given a spiral and a tangent, T_q, to the spiral. Let q' be the closest point to s on T_q. The ratio $C_\Pi(q')$ for q' on T_q depends only on the spiral parameter α and is given by*

$$C_{q'}(\alpha) = \frac{\mathcal{E}^{\cot\alpha(2\pi+\gamma(\alpha)))}}{\sin\alpha\cdot\cos\alpha} + \frac{\mathcal{E}^{b(2\pi+\gamma(\alpha))}\cdot\sin(\gamma(\alpha))}{\sin^2\alpha} + \cot\alpha\,.$$

Its minimum value is $22.49084026\ldots$ *for* $\cot\alpha_{opt} = 0.11371\ldots$ *or* $\alpha_{opt} := 1.4575\ldots$

Proof. Consider the triangle $\triangle psq$, see Figure 3.14. Because q is a point on the spiral we have $|sq| = \mathcal{E}^{\cot\alpha(\theta_q)}$ for some θ_q. Additionally, we have $\theta_p = \theta_q + 2\pi + \gamma(\alpha)$. Further, we have $\angle sqp = \pi - \alpha$. Applying the law of sine yields

$$\frac{|sp|}{\sin(\pi-\alpha)} = \frac{|sp|}{\sin\alpha} = \frac{|pq|}{\sin(\gamma(\alpha))} \Leftrightarrow |pq| = \frac{\mathcal{E}^{\cot\alpha(\theta_p)}\sin\gamma(\alpha)}{\sin\alpha} = \frac{\mathcal{E}^{\cot\alpha(\theta_q+2\pi+\gamma(\alpha))}\sin(\gamma(\alpha))}{\sin\alpha}.$$

Because the triangle $\triangle sq'q$ is right angled, we have $|qq'| = |sq|\cos\alpha = \mathcal{E}^{\cot\alpha(\theta_q)}\cos\alpha$; thus, the distance $|pq'| = |pq| + |qq'|$ is given as

$$|pq'| = \frac{\mathcal{E}^{\cot\alpha(\theta_q+2\pi+\gamma(\alpha))}\sin(\gamma(\alpha))}{\sin\alpha} + \mathcal{E}^{\cot\alpha(\theta_q)}\cos\alpha\,.$$

The length of the arc Π_s^p on the spiral from s to p is given by $|\Pi_s^p| = \frac{\mathcal{E}^{\cot\alpha(\theta_q+2\pi+\gamma(\alpha))}}{\cos\alpha}$, Now, using $|sq'| = |sq|\sin\alpha = \mathcal{E}^{\cot\alpha(\theta_q)}\sin\alpha$, we can compute the ratio $C_{q'}(\alpha)$:

$$\begin{aligned}
C_{q'}(\alpha) &= \frac{|\Pi_s^p| + |pq'|}{|sq'|} \\
&= \frac{\frac{1}{\cos\alpha}\mathcal{E}^{\cot\alpha(\theta_q+2\pi+\gamma(\alpha))} + \frac{1}{\sin\alpha}\mathcal{E}^{\cot\alpha(\theta_q+2\pi+\gamma(\alpha))}\sin\gamma(\alpha) + \mathcal{E}^{\cot\alpha\theta_q}\cos\alpha}{\mathcal{E}^{\cot\alpha\theta_q}\sin\alpha} \\
&= \frac{\frac{1}{\cos\alpha}\mathcal{E}^{\cot\alpha(2\pi+\gamma(\alpha))} + \frac{1}{\sin\alpha}\mathcal{E}^{\cot\alpha(2\pi+\gamma(\alpha))}\sin\gamma(\alpha) + \cos\alpha}{\sin\alpha} \\
&= \left(\frac{1}{\sin\alpha\cdot\cos\alpha} + \frac{\sin\gamma(\alpha)}{\sin^2\alpha}\right)\mathcal{E}^{\cot\alpha(2\pi+\gamma(\alpha))} + \cot\alpha\,.
\end{aligned}$$

We observe that $C_{q'}(\alpha)$ is independent of θ_q, that is, the value of $C_\Pi(q')$ is the same for every given tangent T. Now, the searcher is allowed to minimize the search costs by choosing an appropriate value for α. Evaluating $C_{q'}(\alpha)$ numerically yields a minimum value of $22.49084026\ldots$ for $\cot\alpha = 0.11371\ldots$ or $\alpha_{\text{opt}} := 1.4575\ldots$ □

Finally, an adversary is allowed to choose a starting point, t, along the tangent T_q. By Lemma 3.7 we have to choose β so that $C_\Pi(\alpha_{\text{opt}},\beta) = C(\alpha_{\text{opt}})\sin\beta + \cos\beta$ is maximal. Therefore we have to find a solution for $C(\alpha_{\text{opt}})\cos\beta - \sin\beta$ (first derivative in β) which gives $\beta_{\text{worst}} := 1.526363\ldots$ and $C_\Pi(\alpha_{\text{opt}},\beta_{\text{worst}}) = 22.51306056\ldots$

Corollary 3.10 *The spiral strategy with $\alpha_{opt} := 1.4575\ldots$ is optimal among all spirals and obtains a competitive factor of $C_\Pi(\alpha_{opt},\beta_{worst}) = 22.51306056\ldots$ for angle $\beta_{worst} := 1.526363\ldots$*

Finally, we are interested in a lower bound. To get a lower bound on the competitive ratio for our problem, we discuss the following subproblem: We require that the ray, R, we are looking for is part of a rays that emanates from the searcher's start point, s (i.e., the start point, s, lies on the the extension of R to a straight line)

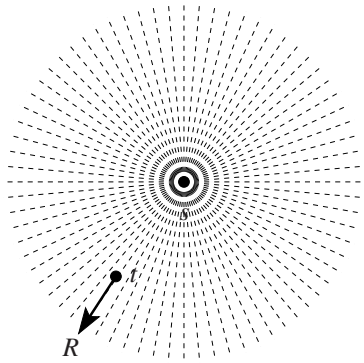


Figure 3.15: A ray, R, that emanates from t and is part of a ray that emanates from s.

If we consider the full bundle of lines passing through s, the given problem is equivalent to the problem of searching for a point in the plane as presented by Alpern and Gal [AG03]. We assume that the searcher detects the goal if it is swept by the radius vector of its trajectory; that is, the searchers knows the position of the goal as soon as it hits the ray emanating from the goal. Alpern and Gal [AG03] showed that among all monotone and periodic strategies, a logarithmic spiral represented by polar coordinates $(\theta, \mathcal{E}^{b\theta})$ gives the best search strategy in this setting. A strategy S represented by its radius vector $X(\theta)$ is called periodic and monotone if θ is always increasing and X also satisfies $X(\theta + 2\pi) \geq X(\theta)$.

The factor of the best achievable monotone and periodic strategy is given by 17.289…, see Alpern and Gal [AG03]. Note, that the searcher does not have to reach the ray's origin in this setting.

Unfortunately, it was not shown that a periodic and monotone strategy is the best strategy for this problem. Alpern and Gal state that it *seems to be a complicated task* to show that the spiral optimizes the competitive factor. Thus, the given factor cannot be adapted to be a lower bound to our problem. Therefore, we consider a discrete bundle of n rays that emanate from the start and which are separated by an angle $\alpha = \frac{2\pi}{n}$, see Figure 3.16. We are searching for a goal on one of the n rays.[1] Again, the goal is detected if it is swept by the radius vector of the trajectory. Note that if n goes to infinity we are back to the original problem. But we can neither assume that we have to visit the rays in a periodic order nor that the depth of the visits increases in every step. Thus, we represent a search strategy, S, as follows: In the kth step, the searcher hits a ray—say ray i—at distance x_k from the origin, moves a distance $\beta_k x_k - x_k$ along the ray i, and leaves the ray at distance $\beta_k x_k$ with $\beta_k \geq 1$. Then, it moves to the next ray within distance $\sqrt{(\beta_k x_k)^2 - 2\beta_k x_k x_{k+1} \cos\gamma_{i,i+1} + x_{k+1}^2}$, see Figure 3.16. Note, that any search strategy for our problem can be described in this way.

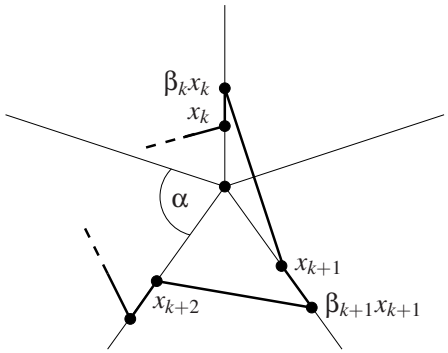


Figure 3.16: A bundle of n rays and the representation of a strategy.

Let us assume that the ray i is visited the next time at index J_k. The worst case occurs if the searcher

[1] Note that the searcher is not confined to walk on the rays, but can move arbitrarily in the plane; in contrast to the m-ray search problem.

slightly misses the goal while visiting ray i up to distance x_k. Instead, it finds the goal at step x_{J_k} on ray i arbitrarily close to $\beta_k x_k$. Either we have $x_{J_k} > \beta_k x_k$; that is, the searcher discovers the goal in distance x_{J_k} on ray i and moves $x_{J_k} - \beta_k x_k$ to the goal, or we have $x_{J_k} < \beta_k x_k$. In the latter case, the searcher moves $\beta_k x_k - x_{J_k}$ from x_{J_k} and finds the goal by accident. In both cases, the searcher moves $|x_{J_k} - \beta_k x_k|$ in the last step. Altogether, the competitive factor, $C(S)$, is bigger than

$$\frac{|x_{J_k} - \beta_k x_k| + \sum_{i=1}^{J_k-1} \beta_i x_i - x_i + \sqrt{(\beta_i x_i)^2 - 2\beta_i x_i x_{i+1} \cos\gamma_{i,i+1} + x_{i+1}^2}}{\beta_k x_k}.$$

By simple trigonometry, the shortest distance from $\beta_i x_i$ to a neighboring ray is given by $\beta_i x_i \sin\frac{2\pi}{n}$. Fortunately, this distance is smaller than the distance

$$\sqrt{(\beta_i x_i)^2 - 2\beta_i x_i x_{i+1} \cos\gamma_{i,i+1} + x_{i+1}^2}$$

to any other ray. Thus, we have

$$C(S) > \frac{\sum_{i=1}^{J_k-1} \beta_i x_i}{\beta_k x_k} \sin\frac{2\pi}{n}.$$

Altogether, we have to find a lower bound for $\frac{\sum_{i=1}^{J_k-1} f_i}{f_k}$, where J_k denotes the index of the next visit of the ray of x_k and $f_i = \beta_i x_i$ denotes the search depth in step i. Fortunately, this problem is the same problem as in the competitive analysis for the usual m-ray problem where the searcher can move only along the rays. It was shown in Lemma 3.3 (see also Gal [Gal80] and Baeza-Yates et al. [BYCR93]) that for this problem there is an optimal strategy that visits the rays with increasing depth and in a periodic order; that is, $J_k = k+n$ and $i = k$. Applying Theorem 3.2 the best achievable strategy is given by $f_i = (n/(n-1))^i$. Altogether, this results in a function

$$(n-1)\left(\frac{n}{n-1}\right)^n \sin\frac{2\pi}{n}$$

for n rays. We can make n arbitrarily big because our construction is valid for every n. Note that we also have a lower bound for the problem of searching a point in the plane; this lower bound is close to the factor that is achieved by a spiral search.

Theorem 3.11 *For the ray search problem there is no strategy that achieves a better factor than*

$$\lim_{n\to\infty} (n-1)\left(\frac{n}{n-1}\right)^n \sin\frac{2\pi}{n} = 17.079\ldots$$

Additionally, every strategy for searching a point in the plane achieves a competitive factor bigger then $17.079\ldots$ *(the optimal spiral achieves a factor of* $17.289\ldots$ *[Gal80]).*

3.3 Searching in street polygons

Now we consider a special class of polygons, such that a competitive search still can be performed. By the m-ray search problem we already know that a constant competitive strategy for searching a point in arbitrary polygons does not exist.

The following polygons resembles streets or rivers where the path to the endpoint is not arbitrary although the path can make many windings and there are many caves where the goal might be loacated. Formally, we define a street polygon as follows:

Definition 3.12 Let P be a simple polygon with two points s and t on the boundary. P is denoted as a **street** (polygon), if the two boundary chains P_L and P_R of P between s and t are weakly visible, i.e., any point from P_L sees at least one point from P_R and vice versa. [Kle91]

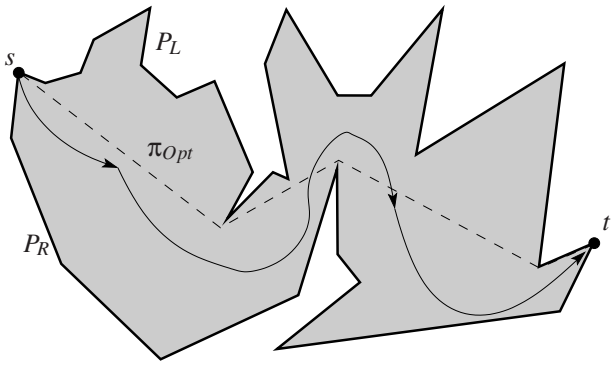


Figure 3.17: A street polygon.

Figure 3.17 shows an example. The main idea is that the shortest path from s to t also sees the boundary chains. Intuitively, if you use a street efficiently, you will always see the boundary chains.

Many structural properties have been proved for street polygons. For example, for a given polygon P one is interested on all possible pair of points (s,t) such that P is a street polygon. Surprisingly, this problem can be computed in linear time; see [THL98, DHN97]. In this section we consider the searching problem. That is, the start point s is given, the agent is equipped with a vision system and we are searching for a target t. The only information is, that P is a street for s and t. Against the shortest path to t we are searching for a competitive strategy with small ratio.

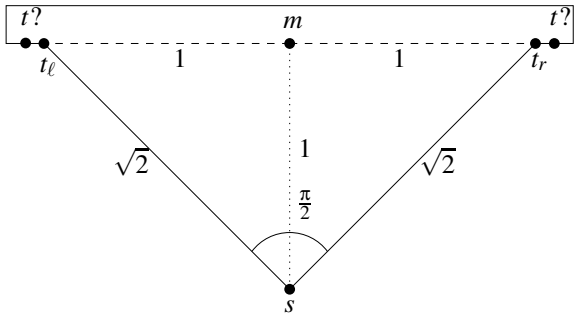


Figure 3.18: Lower bound for searching the target t.

A lower bound for the ratio in our problem can be constructed as follows.

Theorem 3.13 *(Klein, 1991)*
There is no strategy that finds the target t in a street with a path of length smaller than $\sqrt{2} \cdot \pi_{\text{Opt}}$. *The competitive ratio is at least* $\sqrt{2}$. *[Kle91]*

Proof. Consider Figure 3.18. The agent is located at s and*sees* t_ℓ and t_r. The target t lies behind one of them but the agent can only detect t if the line between t_ℓ and t_r is visited. Then the agent can move to t. If the agent visits the segmet between t_ℓ and t_r to the left (right) of the midpoint m, the target is positioned at the right (left). Thus the best the agent can achieve is moving directly to m. Thus we have (where $\varepsilon \longrightarrow 0$):

$$|\pi_{\text{Rob}}| = 2 \quad \text{und} \quad \frac{|\pi_{\text{Rob}}|}{|\pi_{\text{Opt}}|} = \frac{2}{\sqrt{2}} = \sqrt{2}.$$

□

In search of t we can make use of some structural properties. Consider Figure 3.19(i). The agent is located at s and does not see the caves (the shaded parts). A cave is generated by a corresponding reflex vertex[2] of the polygon. We can subdivide the current cave generating reflex vertices into the set of left

[2]Vertices, with inner angle $> \pi$.

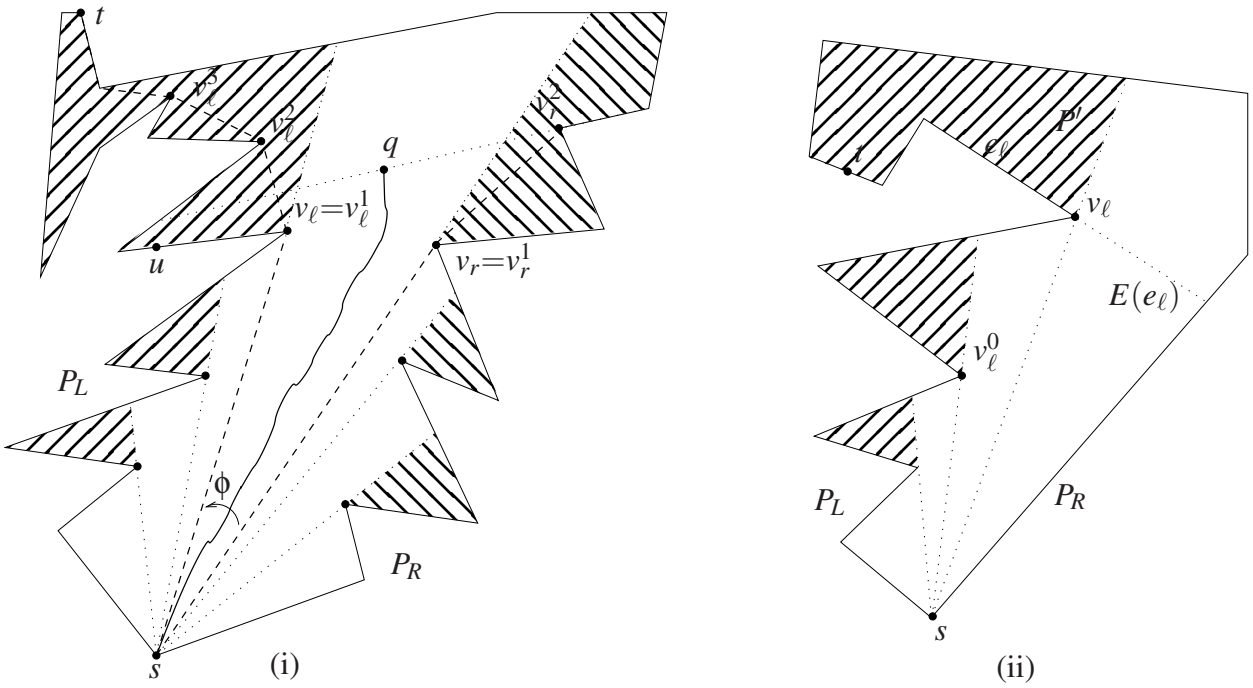


Figure 3.19: Typical situations for the task of searching the target in a street polygon.

reflex vertices (the cave is to the left) and right reflex vertices (the cave lies to the right). We call the vertices left or right reflex vertices, respectively.

Furthermore, we can consider the left reflex vertices in clockwise and the right reflex vertices in counter-clockwise order. One of these sequences can also be empty; in Figure 3.19(ii) there are no right reflex vertices.

We would like to argument that the unknown target t can only be located behind the rightmost left reflex vertex, say v_l, or the leftmost right reflex vertex, say v_r. The target cannot be located in one of the other caves. Assume that this is not the case. Assume that for example in Figure 3.19(i) the target is in the cave below v_l. In this case there is a point u on the right chain closely after v_l that does only see points on the right chain. This means that any reasonable strategy can concentrate on the current triangle of c, v_l and v_r, where c is the current location of the agent. It only makes sense to run into this triangle and let the opening angle at c increase.

If there is only one vertex v_l or v_r, it is clear that the target can only lie behind this remaining vertex and any reasonable strategy move directly to this vertex. It is also clear the the shortest path to the target has to run over this vertex. The same holds, when the target gets visible. The agent directly moves toward it.

Formally, we consider the following cases or events while the agent moves into the triangle of c, v_l and v_r.

- The target becomes visible. The agent moves toward it.
- The cave behing v_ℓ or v_r becomes visible and does not contain the target; as in point q in Figure 3.19(i). The goal has to be behind the remaining vertex, the agent directly moves toward it.
- Behind the current vertex v_ℓ or v_r another left or right reflex vertex becomes visible. For example v_ℓ^2 appears behind v_ℓ. In this case the current left reflex vertex changes from v_ℓ to v_ℓ^2. The agent runs into the triangle of c, v_ℓ^2 and v_r

The last event successively builds segments of convex chain constructed form reflex vertices $v_\ell^1, v_\ell^2, v_\ell^3, \ldots, v_\ell^i$ and $v_r^1, v_r^2, v_r^3, \ldots, v_r^j$ to the left and to the right starting from s. The agent only moves inside these two

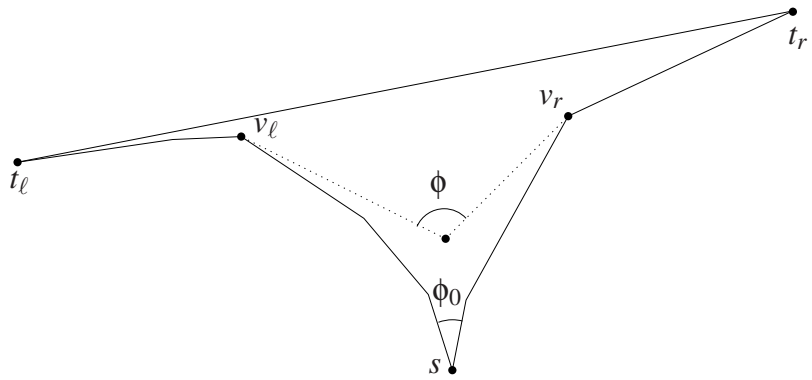


Figure 3.20: A funnel polygon.

chains. Therefore for simplicity we simply forget the original caves and only consider such funnel situations or so called funnel polygons. Beginning from s we have two convex chains that are finally closed by a segment t_l and t_r as shown in Figure 3.20. We assume that the current goal is either behind t_l or t_r. Actually there are two also caves behind t_l and t_r Altogether the funnel polygons will invoke the same path as in the original polygon with caves.

These funnel situations are the only situations that can provoke a detour. If one such situation is resolved, either the goal is reached or the agent is located at a point on the shortest path to the goal. This means that we can consider this situation as the main challenge. If we can guarantee a competitive ratio of C for any single funnel, we can combine the path to a C-competitive strategy in total.

Therefore we concentrate on such polygons.

Definition 3.14 A simple polygon is constructed by two convex chains P_L and P_R starting at a convex vertex s. The polygon can be closed by the segement $\overline{t_\ell t_r}$ of the endpoints of the chains; see seeFigure 3.20. such a polygon is denoted as a **funnel (polygon)**,

Another important observation for the exploration of the funnel is, that the opening angle ϕ for the current position c and the current active reflex vertices v_ℓ und v_r will increase monotonically for any reasonable strategy. The agent starts with a opening angle ϕ_0 at s and finally we will reach $\overline{t_\ell t_r}$ with opening angle 180°. Therefore it is quite natural to describe or parameterise a strategy by the opening angle ϕ.

First, we define a more general lower bound dedicated to the opening angle l ϕ. We can generalize Theorem 3.13 as follows:

Lemma 3.15 *For a funnel polygon with opening angle $\phi \leq \pi$ there is no strategy that has smaller path lenght than $K_\phi \cdot |\pi_{\text{Opt}}|$ against the shortest path to the goal, where*

$$K_\phi := \sqrt{1+\sin\phi}.$$

Any strategy is at least K_ϕ competitive.

Proof. Consider Figure 3.21. By the sam argument as in the proof of Theorem 3.13 the best an agent can do is moving directly to the midpoint m. Any other movement results in a larger detour since we can place the target afterwards. Now the agent sees the target and moves toward it. [3]. For $\phi \leq \pi$ we have

$$\frac{|\pi_S|}{|\pi_{\text{Opt}}|} = \frac{\ell\cos\frac{\phi}{2}+\ell\sin\frac{\phi}{2}}{\ell} = \sqrt{1+\sin\phi}.$$

□

[3]The path of length ε from v_ℓ orv_r to t need not be considered

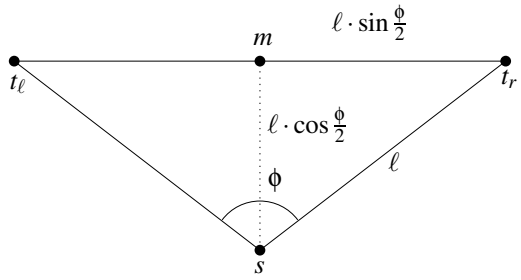


Figure 3.21: Generalized lower bound.

Note that for the final opening angle $\phi = \pi$ and $K_\phi = 1$ the agent will always move corretly, since the target is visible now. For $\phi = \frac{\pi}{2}$ we have the ratio $K_\phi = \sqrt{2}$ as in Theorem 3.13. For $0 \leq \phi \leq \pi$ the function K_ϕ gives a curve that starts at 1 rises up monotonically to $\sqrt{2}$ at $\frac{\pi}{2}$ and decreases monotonically toward 1 at π.

Assume that the agent explores a funnel starting from s with opening angle ϕ_0 and follows a path with monotonically increasing opening angles until $\overline{t_\ell t_r}$ is visited and $\phi = \pi$ holds.

For $\frac{\pi}{2} \leq \phi_1 < \phi_2$ we have $K_{\phi_1} > K_{\phi_2}$, and the competitive ratio for the overall exploration is dominated by the smaller angle. For $\phi_1 < \phi_2 \leq \frac{\pi}{2}$ we have $K_{\phi_1} < K_{\phi_2} \leq \sqrt{2}$, the ratio is dominated by the larger opening angle If the agent starts from an opening angle $\phi_0 < \frac{\pi}{2}$ along a path to angle $\phi = \pi$ there will always be a point such that the opnening angle $\phi = \frac{\pi}{2}$ is attained. Therefore the worst case ratio $\sqrt{2}$ is always included.

It seems to make sense to consider the case $\phi_0 < \frac{\pi}{2}$ and $\phi_0 \geq \frac{\pi}{2}$ separately. We start with $\phi_0 \geq \frac{\pi}{2}$. We already have a successful strategy for $\phi = \pi$. The following idea is that we apply a backward analysis that tells us how to prolong a successful strategy for opening angle ϕ_2 to a successful strategy for opening angle $\phi_1 < \phi_2$. By the following lemma we design a requirement for any path w from angle ϕ_1 to ϕ_2.

Lemma 3.16 *Let Π be a strategy that can reach the target of any funnel polygon with opening angle $\phi_2 \geq \frac{\pi}{2}$ by competitve ratio K_{ϕ_2}. We can extend this strategy to a K_{ϕ_1} competitive startegy for funnel polygons with opening angle ϕ_1 with $\phi_2 > \phi_1 \geq \frac{\pi}{2}$, if the path w between the two corresponding points fulfils the length condition Equation 3.9 for the current situation as depicted in Figure 3.22.*

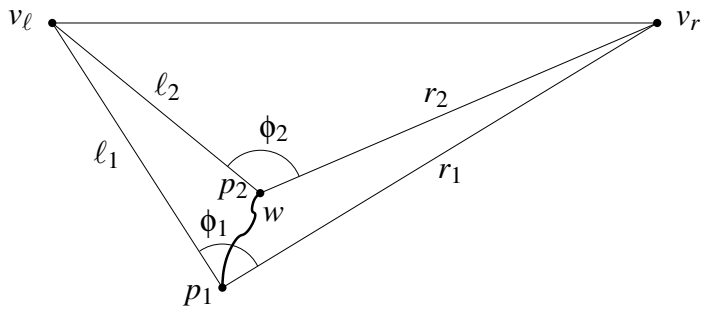


Figure 3.22: A path w from p_1 with angle ϕ_1 to p_2 with angle ϕ_2.

Proof. We consider a triangle with opening angle ϕ_1, start point p_1 and a path w to a point p_2 with opening angle ϕ_2; see Figure 3.22. From p_2 the agent can use the strategy Π for the angle ϕ_2 which is known by assumption. Π is K_{ϕ_2} competitive. Let us assume that during the movement w the vertices v_ℓ and v_r do not change.

Let ℓ_1 and ℓ_2 denote the distances from p_1 and p_2 to v_ℓ, as depicted in Figure 3.22, r_1 and r_2 are defined analogously. If the goal lies behind v_ℓ we can assume that the overall path length for $\pi^t_{p_1}$ from p_1 to t is:

$$|\pi^t_{p_1}| \leq |w| + K_{\phi_2} \cdot \ell_2.$$

We would like to guarantee that the overall strategy is K_{ϕ_1}-competitive, therefore we require: $K_{\phi_1} = \frac{|\pi'_{p_1}|}{|\pi_{\text{Opt}}|} \geq \frac{|w|+K_{\phi_2}\cdot\ell_2}{\ell_1}$, also

$$K_{\phi_1}\cdot\ell_1 \geq |w| + K_{\phi_2}\cdot\ell_2.$$

Analogously, if the goal is behind v_r, we require $K_{\phi_1}\cdot r_1 \geq |w| + K_{\phi_2}\cdot r_2$.

If we can guarantee that the path w from p_1 to p_2 fulfils the length condition

$$|w| \leq \min\{\, K_{\phi_1}\ell_1 - K_{\phi_2}\ell_2 \;,\; K_{\phi_1}r_1 - K_{\phi_2}r_2 \,\}, \tag{3.9}$$

we conclude that the overall strategy starting at p_1 attains a competitive ratio of K_{ϕ_1} for the funnel with opening angle ϕ_1.

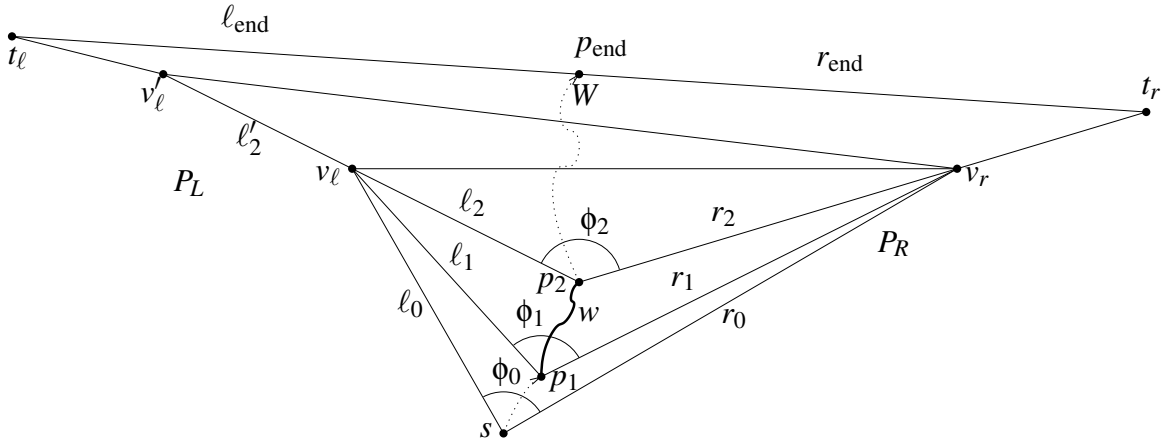


Figure 3.23: At p_2 a new left reflex vertex is detected.

Now it is clear that from time to time the reflex vertices in the funnel will change. The path w and the condition Equation 3.9 should still guarantee the above conclusion. Therefore we consider the situation that condition Equation 3.9 if fulfilled but precisely at p_2 there is a change of the reflex vertices as shown in Figure 3.23. In p_2 behind v_ℓ a new left reflex vertex v'_ℓ appears. Since Equation 3.9 holds we can conclude:

$$\begin{aligned} |w| &\leq K_{\phi_1}\ell_1 - K_{\phi_2}\ell_2 \\ &= K_{\phi_1}\ell_1 - K_{\phi_2}\ell_2 + K_{\phi_2}\ell'_2 - K_{\phi_2}\ell'_2 \\ &\leq K_{\phi_1}(\ell_1+\ell'_2) - K_{\phi_2}(\ell_2+\ell'_2) \end{aligned} \tag{3.10}$$

The last inequalityl is true, since from Lemma 3.15 for $\phi_2 > \phi_1 \geq \frac{\pi}{2}$ we have $K_{\phi_2} < K_{\phi_1}$. Note that $\ell_1+\ell'_2$ respectively $\ell_2+\ell'_2$ denote the lengths of the shortest paths from p_1 respectively p_2 to v'_ℓ. Equation Equation 3.10 says that the condition Equation 3.9 takes care that also for changes of the reflex vertices, we have obtain a K_{ϕ_1} competitive strategy at p_1 □

Assume that Equation 3.9 holds for all small changes of opening angles for the overall path W from s t op_{end}, we conclude

$$|W| \leq \min\{\, K_{\phi_0}\cdot|P_L| - K_\pi\ell_{\text{End}} \;,\; K_{\phi_0}\cdot|P_R| - K_\pi r_{\text{End}} \,\}.$$

Altogether we have a K_{ϕ_0} competitive strategy in this case.

Now it is sufficient to guarantee that the agent fulfils Equation 3.9 during the movements. The idea of fulfilling this requirement is as follows: The portions $K_{\phi_1}\ell_1 - K_{\phi_2}\ell_2$ and $K_{\phi_1}r_1 - K_{\phi_2}r_2$ somehow express how many path length w we can use in the next step for the left or the right location of the goal,

respectively. Since we do not know where the target will be at the end, we do not want to let one side have an advantage at this stage.

Therefore we would like to guarantee that both values are the same. This gives

$$K_{\phi_2}(\ell_2 - r_2) = K_{\phi_1}(\ell_1 - r_1)\,.$$

Fortunately, by this requirement we indeed define a special curve for any starting situation with angle ϕ_0 and length l_0 and r_0. Let $A = K_{\phi_0}(\ell_0 - r_0)$ The curve that fulfils the above equation all the time is given by

$$\begin{aligned} X(\phi) &= \frac{A}{2} \cdot \frac{\cot\frac{\phi}{2}}{1+\sin\phi} \cdot \sqrt{\left(1+\tan\frac{\phi}{2}\right)^2 - A^2} \\ Y(\phi) &= \frac{1}{2} \cdot \cot\frac{\phi}{2} \cdot \left(\frac{A^2}{1+\sin\phi} - 1\right). \end{aligned}$$

We will now explain how we have developed the formulas above.We choose a coordinate system with axis paralell to $v_l\,v_r$, the midpoint of $v_l\,v_r$ is the origin. We scale such that $|v_l\,v_r| = 1$. Let p be the point on the curve with opening angle ϕ; see Figure 3.24. We have starting values ϕ_0, l_0 and r_0 and set $A := K_{\phi_0}(\ell_0 - r_0)$.

In order to find p we have to fulfil two conditions. First, the difference $l(p) - r(p)$ of the distances from p to v_l and v_r has to equal $\frac{A}{K_\phi}$. The locus of all such point is a hyoerbola. Second the angle at p with respect to v_l and v_r has to be ϕ. The locus of all such points is a circle; see Figure 3.24. This holds because of the Thales' circle property.

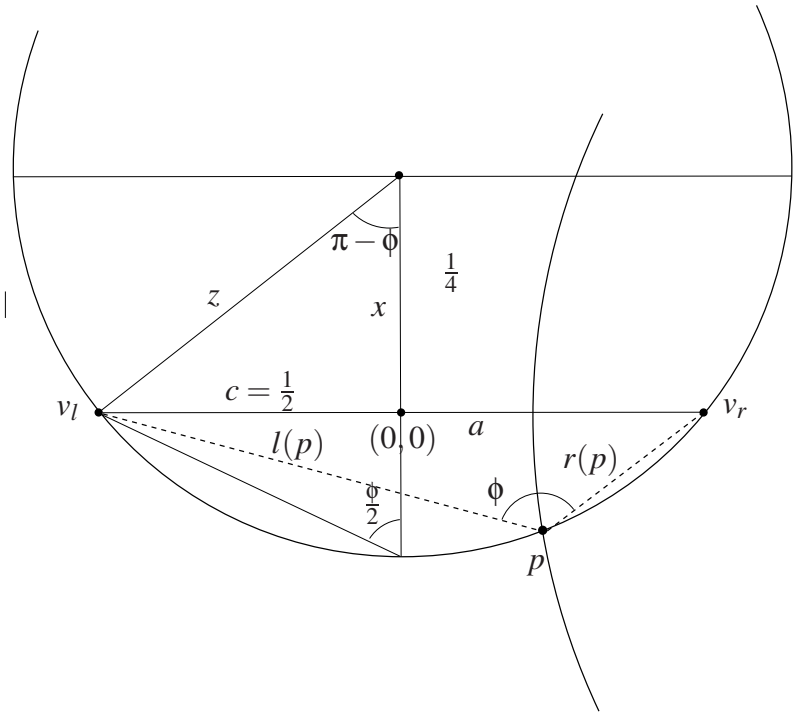


Figure 3.24: The left arc of the hyperbola is defined by v_l, v_r and $(l(p) - r(p)) = \frac{A}{K_\phi}$ and the circle running through v_l and v_r is defined by the opening angle ϕ.

The hyperbola is defined by

$$\frac{X^2}{a^2} - \frac{Y^2}{b^2} = 1\,,$$

where $2a = (l(p) - r(p)) = \frac{A}{K_\phi}$ and $b^2 + a^2 = c^2 = \frac{1}{4}$ holds. This gives $a^2 = \left(\frac{A}{2K_\phi}\right)^2$ and $b^2 = \frac{1}{4} - \left(\frac{A}{2K_\phi}\right)^2$. The circle is defined by

$$X^2 + (Y - x)^2 = z^2 \,. \tag{3.10}$$

This means that we have to calculate x and z. From the law of sine we conclude

$$\frac{z}{\sin\frac{\pi}{2}} = \frac{1}{2\sin(\pi - \phi)} = \frac{1}{2\sin\phi}$$
$$\frac{z - x}{\sin\left(\pi - \frac{\pi}{2} - \frac{\phi}{2}\right)} = \frac{z - x}{\cos\frac{\phi}{2}} = \frac{1}{2\sin\frac{\phi}{2}}$$

and therefore $z = \frac{1}{2\sin\phi}$ and

$$x = z - \frac{1}{2}\cot\frac{\phi}{2} = \frac{1}{2\sin\phi} - \frac{1}{2}\cot\frac{\phi}{2} = \frac{1 - 2\cos^2\frac{\phi}{2}}{4\sin\frac{\phi}{2}\cos\frac{\phi}{2}} = -\frac{\cot\phi}{2} \,.$$

The intersection of the hyperbola and the circle is indeed given by the above functions $X(\phi)$ and $Y(\phi)$. We have found the solutions by a computer algebra system. Here we simply verify that the solutions are correct. We insert the values into the hyberboly and the circle description.

$$\frac{X^2}{\left(\frac{A}{2K_\phi}\right)^2} - \frac{Y^2}{\left(\frac{1}{2}\right)^2 - \left(\frac{A}{2K_\phi}\right)^2} = 1 \tag{3.11}$$

$$X^2 + \left(Y + \frac{\cot\phi}{2}\right)^2 = \frac{1}{4\sin^2\phi}. \tag{3.12}$$

For (3.11) we have

$$\frac{\left(\frac{A}{2}\cdot\frac{\cot\frac{\phi}{2}}{1+\sin\phi}\sqrt{\left(1+\tan\frac{\phi}{2}\right)^2 - A^2}\right)^2}{\left(\frac{A}{2K_\phi}\right)^2} - \frac{\left(\frac{1}{2}\cot\frac{\phi}{2}\left(\frac{A^2}{1+\sin\phi} - 1\right)\right)^2}{\left(\frac{1}{2}\right)^2 - \left(\frac{A}{2K_\phi}\right)^2} =$$

$$\left(\frac{\cot\frac{\phi}{2}}{K_\phi}\right)^2\left(\left(1+\tan\frac{\phi}{2}\right)^2 - A^2\right) - \frac{\cot^2\frac{\phi}{2}\left(\left(\frac{A}{K_\phi}\right)^2 - 1\right)^2}{1 - \left(\frac{A}{K_\phi}\right)^2} =$$

$$\left(\frac{\cot\frac{\phi}{2}}{K_\phi}\right)^2\left(\left(1+\tan\frac{\phi}{2}\right)^2 - A^2\right) + \cot^2\frac{\phi}{2}\left(\left(\frac{A}{K_\phi}\right)^2 - 1\right) =$$

$$\cot^2\frac{\phi}{2}\left(\frac{\left(1+\tan\frac{\phi}{2}\right)^2}{1+\sin\phi} - 1\right) = 1 \,.$$

The conclusion is valid since the following identity holds.

$$1 + \sin\phi = 1 + \frac{2\tan\frac{\phi}{2}}{1 + \tan^2\frac{\phi}{2}} = \frac{\left(1+\tan\frac{\phi}{2}\right)^2}{1 + \tan^2\frac{\phi}{2}} \tag{3.13}$$

For showing (3.12) we proceed as follows:

$$\begin{aligned}
&\left(\frac{A}{2}\cdot\frac{\cot\frac{\phi}{2}}{1+\sin\phi}\sqrt{\left(1+\tan\frac{\phi}{2}\right)^2-A^2}\right)^2+\left(\frac{1}{2}\cot\frac{\phi}{2}\left(\frac{A^2}{1+\sin\phi}-1\right)+\frac{\cot\phi}{2}\right)^2 = \\
&\qquad\left(\frac{A}{2}\cdot\frac{\cot\frac{\phi}{2}}{1+\sin\phi}\right)^2\left(\left(1+\tan\frac{\phi}{2}\right)^2-A^2\right)+ \\
&\left(\frac{1}{2}\cot\frac{\phi}{2}\left(\frac{A^2}{1+\sin\phi}-1\right)\right)^2+\cot\frac{\phi}{2}\left(\frac{A^2}{1+\sin\phi}-1\right)\frac{\cot\phi}{2}+\left(\frac{\cot\phi}{2}\right)^2 = \\
&\qquad\left(\frac{A}{2}\cdot\frac{\cot\frac{\phi}{2}}{1+\sin\phi}\right)^2\left(1+\tan\frac{\phi}{2}\right)^2+ \\
&\left(\frac{1}{2}\cot\frac{\phi}{2}\right)^2\left(-2\frac{A^2}{1+\sin\phi}+1\right)+\cot\frac{\phi}{2}\left(\frac{A^2}{1+\sin\phi}-1\right)\frac{\cot\phi}{2}+\left(\frac{\cot\phi}{2}\right)^2 = \\
&\qquad\left(\frac{\cot\frac{\phi}{2}}{2}-\frac{\cot\phi}{2}\right)^2+\frac{A^2\cot^2\frac{\phi}{2}}{4(1+\sin\phi)}\left(\frac{\left(1+\tan\frac{\phi}{2}\right)^2}{1+\sin\phi}-2+2\frac{\cot\phi}{\cot\frac{\phi}{2}}\right) = \\
&\qquad\frac{1}{4\sin^2\phi}+\frac{A^2\cot^2\frac{\phi}{2}}{4(1+\sin\phi)}\left(\tan^2\frac{\phi}{2}+1-2+\frac{1-\tan^2\frac{\phi}{2}}{\tan\frac{\phi}{2}}\tan\frac{\phi}{2}\right) = \\
&\qquad\qquad\frac{1}{4\sin^2\phi}+\frac{A^2\cot^2\frac{\phi}{2}}{4(1+\sin\phi)}\cdot 0 = \frac{1}{4\sin^2\phi}.
\end{aligned}$$

Here we make use of the identity (3.13) and the equations

$$\left(\frac{\cot\frac{\phi}{2}}{2}-\frac{\cot\phi}{2}\right)^2=\frac{1}{4}\left(\frac{\sin\phi}{1-\cos\phi}-\frac{\cos\phi}{\sin\phi}\right)^2=\frac{1}{4}\frac{1}{\sin^2\phi}$$

and

$$\cot\phi=\frac{1-\tan^2\frac{\phi}{2}}{2\tan\frac{\phi}{2}}.$$

Finally, we have to prove that the above curve indeed fulfils the condition for any small piece w. Experimentally, we make use of the precise curve description and import it into Geogebra or Maple. Here we approximate the path between any two points by the corresponding segment. This procedure already indicates that assumption has to be true.

It can also be shown analytically. A lengthy, detailed proof is given in [IKL99] or [Lan00]. Figure 3.25 shows examples for the curve for different values of ϕ and A. The figure stems from a Maple plot.

We obtain the following result:

Corollary 3.17 *For a funnel polygon with opening angle $\phi_0 > \frac{\pi}{2}$ we will find any unknown target within a competitive ratio K_{ϕ_0}.*

Finally, for angles $0 < \phi_0 < \frac{\pi}{2}$ we can apply the same approach. Of course we can also apply the condition

$$K_{\phi_2}(\ell_2 - r_2) = K_{\phi_1}(\ell_1 - r_1)$$

for $\phi_1 < \phi_2 < \frac{\pi}{2}$.

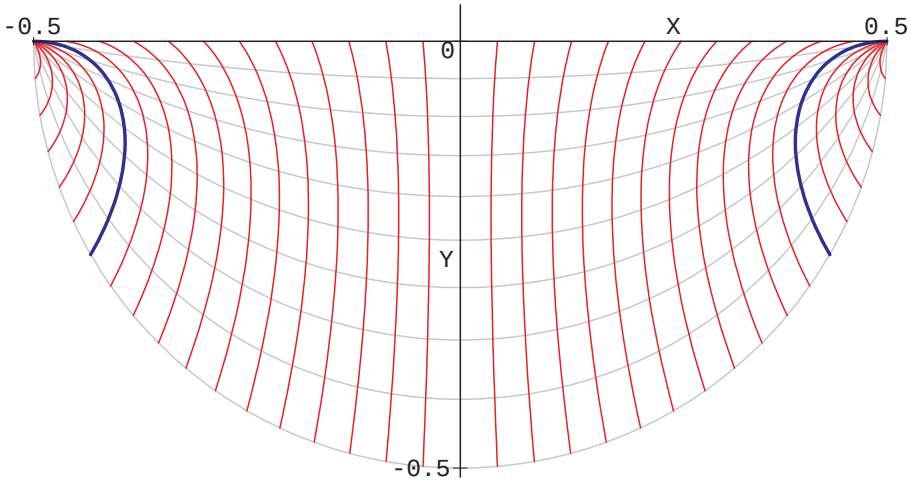


Figure 3.25: Curves $(X(\phi), Y(\phi))$ depending from ϕ and A.

Not that this will also result in a continuous extension of the curves of Figure 3.25. The problem is that these curve parts will not fulfil the condition Equation 3.9 because $K_{\phi_1} < K_{\phi_2}$ holds. Therefore we just insert the fixed ratio $\sqrt{2}$ which we would like to achieve at angle $\frac{\pi}{2}$. The factor $\sqrt{2}$ dominates all K_ϕ.

By the same arguments as before it is sufficient to guarantee

$$w \leq \min\{\sqrt{2}(\ell_1 - \ell_2)\ ,\ \sqrt{2}(r_1 - r_2)\}$$

for any small piece of our curve.

Again we would not prefer one side and set $\ell_1 - \ell_2 = r_1 - r_2$. This means that we are moving on the current angular bisector and call this startegy CAB (Current Angular Bisector); see also [IKL97, LOS96]. The analysis is also prenseted in [IKL99] oder [Lan00]. Note that if we apply the factor $\sqrt{2}$ for the angles above $\frac{\pi}{2}$ for the path w we will also define a curve but the above path length property for w does not hold.

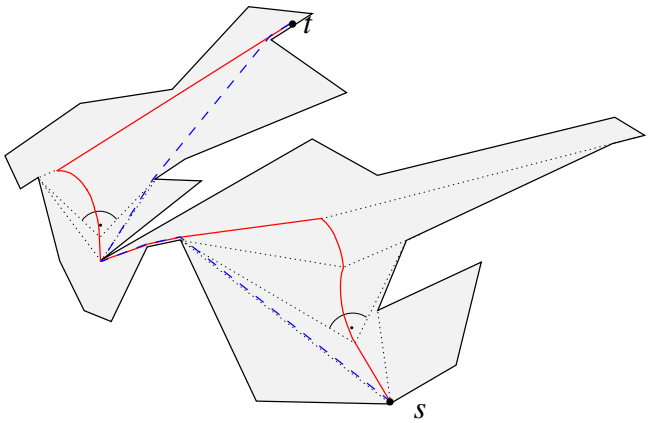


Figure 3.26: An example of the application of WCA.

Algorithm 3.1 summarizes the strategy, Figure 3.26 shows an example of its application. Altogether, the following result holds:

Theorem 3.18 *(Icking, Klein, Langetepe, Schuierer, Semrau, 1999)*
Searching for the target t inside an unknown street polygon can be performed by an optimal $\sqrt{2}$ competitive strategy. *[IKL99, SS99, IKL+04]*

We have implemented the optimal strategy under the name "WCA" (Worst-Case-Aware), an applet can be found here:

`http://www.geometrylab.de/`

Algorithm 3.1 Searching for the target of a street.

While target t is not visible:

- Compute extreme reflex vertices v_ℓ and v_r.
- FIf only on exist, move toward it.
- Otherwise repeat:
  - If a new reflex vertex v'_ℓ or v'_r is detected: Replace v_ℓ or v_r by v'_ℓ or v'_r, respectively.
  - Let ϕ be the current opening angle w.r.t. v_ℓ and v_r.
  - If $\phi \leq \frac{\pi}{2}$: Follow the current angular bisector
  - If $\phi > \frac{\pi}{2}$: Follow the curve represented by $X(\phi)$ and $Y(\phi)$ with the current value A.
- Until either v_ℓ or v_r is fully explored. Move to the vertex on the opposite side.

Move to the target t.

3.4 Optimal search paths

We now consider the problem of searching for a goal in more general environments such as polygons (with and without holes) graphs or trees. We consider the online and offline version. In the offline version the environment is known, the goal remains unknown. In any case a search path has to visit or *see* all possible goals of the given environment. Therefore any search path is also an exploration path for all goals.

If the agent has a vision system, for any goal t there will be a first point on the search path where t gets visible. After reaching this point the agent can move to t along the shortest path. The agent will only use this last path, if t is the goal. In this sense the search path itself does not visit the goals, it only *sees* any goal. On the contrary, if the agent has no vision, the search path has to visit any possible goal.

In the online version of the problem, additionally the agent has to gain more information about the enviroment for future computation. We have already seen that a general constant competitive search strategy does not exists for all groups of environments. For example, searching for the goal among m fixed corridors with a cave at distance 1 at the end of each corridor, results in a search path of length $2m-1$ whereas the shortest path to the last seen point is 1. Since m can go to infinity, there is no constant C for the ratio. On the other hand for the fixed configuration of m corridors (m is fixed) no other strategy than visiting all caves successively has a better ratio. This means that there is a large ratio for the last point visited against the shortest path, but any strategy has this large ratio. Therefore for a comparisons between good or bad search paths we just compare a path to the worst case ratio that any strategy has to cope with.

The following definition is made for arbitrary environments $\mathcal{E}$. For trees and graphs $G=(V,E)$ we consider two different variants w.r.t. the goal set. In the **vertex search** variant, the goals can only be located at the vertices of G. The agent need not necessarily visit all edges. In the **geometric search** variant, the goal can be located everywhere on the graph. Any search path has to visit all edges and vertices.

For the general definition we introduce the **goal set** $\mathcal{G}\subseteq\mathcal{E}$. For a graph $G=(V,E)$ and the vertex search problem we have $\mathcal{G}=V$, and $\mathcal{G}=V\cup E$ for the geometric search.

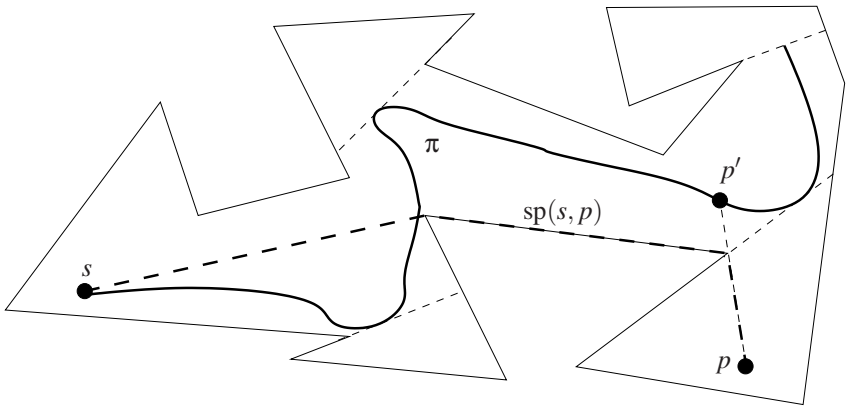


Figure 3.27: A search path π in a simple polygon. The point p' on π, is the first point on π such that p is *seen* from π.

Definition 3.19 Let $\mathcal{E}$ be an environment, $\mathcal{G}\subseteq\mathcal{E}$ a **goal set** and s a point inside $\mathcal{E}$. A **search path** π with start point s is a path in $\mathcal{E}$, such that π starts in s and detects any point of $\mathcal{G}$ at least once. The **performance** of the search path (denoted as search ratio) is defined by[4]

$$\mathrm{SR}(\pi) := \max_{p\in\mathcal{G}} \frac{|\pi_s^{p'}| + |p'p|}{|\mathrm{sp}(s,p)|},$$

[4] $\pi_{\mathrm{Strat}}{}_a^b$ denotes the portion of π_{Strat} from a to b, $\mathrm{sp}(a,b)$ denotes the shortest path from a to b in $\mathcal{E}$.

where p' denotes the first point along π, such that p is detected from π; compare Figure 3.27. An **optimal search path** π_{opt} is a search path for E and $\mathcal{G}$ with minimal search ratio over all search paths in $\mathcal{E}$.

For agent without visiom we have $p' = p$ for all points of $\mathcal{G}$ from $\mathcal{E}$. The performance is given by:

$$\text{SR}(\pi) = \max_{p \in \mathcal{G}} \frac{|\pi_s^p|}{|\text{sp}(s,p)|}.$$

For some environments, it is known that the computation of the optimal search path is an NP-hart problem, i.e. for graphs [KPY96]. For some other environments it is even not clear how the optimal search path can be computed. Computing an optimal search path is a difficult task, therefore we are looking for good and easy to compute approximations. We would also like to approximate the optimal search path in the online version, i.e., we are looking for a constant C_S, such that for any environment we guarantee

$$\text{SR}(\pi_{\text{onl}}) \leq C_S \cdot \text{SR}(\pi_{\text{opt}}).$$

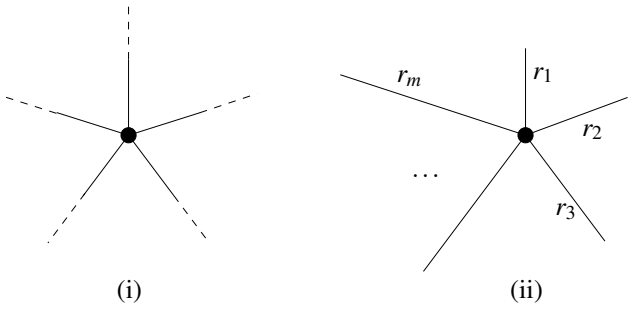


Figure 3.28: (i) m ray, (ii) m segments of different length.

At first place the optimal search path is computed for the offline version and can be handled as a comparison measure for the online version. If the offline optimal search path is not known, any online approximation is also an offline approximation. Let us consider some examples:

1. If we are searching for a goal on m rays, that emanate from a common start point s, the online and the offline version coincidence. We do not have more information in the offline version.. The search strategy that visits ray (i mod m) with depth $\left(\frac{m}{m-1}\right)^i$ in the i-the step has the best competitve ratio among all possible strategies. Therefore trivially this is also the optimal search path. This means we have an approximation of the search ratio by factor $C_S = 1$.

2. If we slightly relax the above example and replace the m rays by segments of different length $r_1, r_2, \ldots, r_m$ as show in Figure 3.28, the problem of computing the optimal search path is still unknown for the offline case. We have to compute the optimal competitive strategy in the offline version. Approximations are possible by applying and adjusting the optimal competitive strategy of the ray version (all $r_i = \infty$).

3. In case of a tree environment the optimal search path for the vertex search can be computed in exponential time, when the tree is given. We consider any permutation of the vertex set and calculate the search ratio of the corresponding path. So we will find the best search path.

4. For the geometric search in trees we cannot apply the above (example 3) strategy since the goal might be located along an edge. Example 2 is a special case of the geometric search version on general trees, therefore the optimal offline search path is still unknown also for general trees in this case.

Algorithm 3.2 Searchpath by doubling exploration depth

- Let $\text{Expl}_{\text{onl}}(d)$ be a competitive (online) strategy for the exploration of an environment up to depth d. The strategy finally returns to the start.
- Successivly explore environment $\mathcal{E}$ by increasing depth, applying $\text{Expl}_{\text{onl}}(2^i)$ for $i = 1,2,\ldots$ from start point s.

Some of the above example cry for an approximation of the search ratio. The same holds for the online version. The general idea for an approximation is as follows. For the given environment we successively apply a constant competitive (online or offline) exploration strategies with increasing depth $d = 2^i$ in analogy to the doubling heuristic used by searching for the door along a wall (doubling heuristic; see Algorithm 3.2). Let Expl_{onl} denote a competitive online strategy that explores the environment and returns to the start. Let $\text{Expl}_{\text{onl}}(d)$ denote a sub-strategy that performs an exploration restricted to all goals in the (possibly larger) environment that are no more than distance d away from the start. Expl_{opt} and $\text{Expl}_{\text{opt}}(d)$ denote the corresponding offline strategies for these problems. Furthermore, $\pi_{\text{Expl}_{\text{onl}}}$ etc. denote the corresponding paths.

Lemma 3.20 *Let $\mathcal{E}$ be an environment, such that an agent without vision system is searching for a goal. Let $\text{Expl}_{\text{onl}}(d)$ be a C-competitive strategy that explores $\mathcal{E}$ up to distance d. By the use of the doubling heuristic (Algorithm 3.2) we achieve a $4C$-approximation of the optimal search ratio.*

Proof. $\text{Expl}_{\text{onl}}(d)$ is competitive, which means that there is a constant C such that for all enviroments $\mathcal{E}$ we have

$$|\pi_{\text{Expl}_{\text{onl}}(d)}| \leq C \cdot |\pi_{\text{Expl}_{\text{opt}}(d)}|. \tag{3.14}$$

Also the optimal search path π_{opt} finally visits all points with distance d. Let $\text{last}(d)$ be the last point at distance d from s that is detected (and visited) by π_{opt}. The performance of this point is $\frac{|\pi_{\text{opt}_s^{\text{last}(d)}}|}{d}$. This is a lower bound of the search ratio, the general performance of π_{opt} cannot be better than the performance ratio in $\text{last}(d)$. Therefore we give the following lower bound:

$$\text{SR}(\pi_{\text{opt}}) \geq \frac{|\pi_{\text{opt}_s^{\text{last}(d)}}|}{d}. \tag{3.15}$$

The optimal search path π_{opt} applied from s to $\text{last}(d)$ explores all goals at depth d and $\pi_{\text{opt}_s^{\text{last}(d)}}$ is an exploration path for depth d. If we return from $\text{last}(d)$ to the start s by the shortest path of length d, we obtain an exploration tour that returns to the start. This overall path is not shorter that the optimal depth d restricted exploration tour (with return to s), we conclude

$$|\pi_{\text{Expl}_{\text{opt}}(d)}| \leq |\pi_{\text{opt}_s^{\text{last}(d)}}| + d. \tag{3.16}$$

From Equation 3.15 and Equation 3.16 we have

$$|\pi_{\text{Expl}_{\text{opt}}(d)}| \leq d \cdot (\text{SR}(\pi_{\text{opt}}) + 1). \tag{3.17}$$

The strategy applies $\text{Expl}_{\text{onl}}(d)$ with increasing exploration depth $d = 2^0, 2^1, 2^2, \ldots$ The worst case for a ratio is attained, if we muss a goal with distance $2^j + \varepsilon$ in the round for exploration depth $d = 2^j$

and detect and visit this point almost at the end of the round with exploration depth $d = 2^{j+1}$. This gives a worst case search ratio for each round by

$$\begin{aligned}
\mathrm{SR}(\pi) &\leq \frac{\sum_{i=1}^{j+1} |\pi_{\mathrm{Expl}_{\mathrm{onl}}(2^i)}|}{2^j + \varepsilon} \\
&\underset{(3.14)}{\leq} \frac{C}{2^j} \sum_{i=1}^{j+1} |\pi_{\mathrm{Expl}_{\mathrm{opt}}(2^i)}| \\
&\underset{(3.17)}{\leq} \frac{C}{2^j} \sum_{i=1}^{j+1} 2^i \cdot (\mathrm{SR}(\pi_{\mathrm{opt}}) + 1) \\
&\leq C \cdot \left(\frac{2^{j+2} - 1}{2^j} \right) \cdot (\mathrm{SR}(\pi_{\mathrm{opt}}) + 1) \leq 4C \cdot (\mathrm{SR}(\pi_{\mathrm{opt}}) + 1).
\end{aligned}$$

□

For trees we have an optimal exploration strategy for any d with a ratio of $C = 1$ by DFS:

Corollary 3.21 *(Koutsoupias, Papadimitriou, Yannakakis, 1996)*
For any tree we can approximate the optimal search ratio by a factor of 4.

[KPY96]

An interesting result, because the optimal search path and the optimal search ratio is unknown.

For other environments, the main problem is finding competitive strategies for the depth restricted exploration. For general graphs $G = (V, E)$ we have introduced CFS-Algorithm in Section 1.5 on page 31. This algorithm can be used for depth restricted exploration for depth $r := d$. There is a problem with this strategy, since we use a rope of length $(1 + \alpha)d$ and guarantee a competitive factor of $4 + \frac{8}{\alpha}$, we guarantee the exploration only for depth d.

Since we explore the graph with rope length $(1 + \alpha)d$ it might happen that also parts of the graph with distance larger than d will be explored. In the offline optimal exploration path such parts will never be visited. The workaround for this problems is as follows. We compare the restricted depth strategy with for depth d (that partially visits depth βd) to the optimal offline exploration with depth βd. In this case we are on the safe side. In the case of CFS we have $\beta = 1 + \alpha$. Also the comparison ratio might depend on β. We make use of a ratio C_β such that $\mathrm{Expl}_{\mathrm{onl}}(d) \leq C_\beta \cdot \mathrm{Expl}_{\mathrm{opt}}(\beta d)$ holds. For the CFS we have $C_\beta = 4 + \frac{8}{\alpha}$.

Theorem 3.22 *(Fleischer, Kamphans, Klein, Langetepe, Trippen, 2003)*
Let $\mathcal{E}$ be an environment where an agent without vision system is searching for a target. Let $\mathrm{Expl}_{\mathrm{onl}}(d)$ *be strategy for the depth restricted online exploration of $\mathcal{E}$ with* $\mathrm{Expl}_{\mathrm{onl}}(d) \leq C_\beta \cdot \mathrm{Expl}_{\mathrm{opt}}(\beta d)$. *We can search in $\mathcal{E}$ by the doubling heuristic (Algorithm 3.2) and attain a eine $4\beta C_\beta$-approximation of the optimal search path and search ratio.* *[FKK+04]*

Proof. In pure analogy to the proof of Lemma 3.20, only changing the version of Equation 3.14. □

Corollary 3.23 *For general graphs and online geometric search we can approximate the optimal search ratio by a factor of* $4(1 + \alpha)(4 + \frac{8}{\alpha})$.

In the above version without vision we always guaranteed that the last point, $\mathrm{last}(d)$, detected at distance d is also exactly visited at this moment in time. For an agent with a vision system it might happen that the search paths visits a point $\mathrm{last}(d)$ from which the last point at distance d is detected and seen but $\mathrm{last}(d)$ has not distance d to the start.

We can no longer conclude $|\mathrm{sp}(s, \mathrm{last}(d))| \leq d$, which was required for the bound Equation 3.15. Fortunately, for the agent with vision system we can at least guarantee that $|\mathrm{sp}(s, \mathrm{last}(d))| \leq |\pi_{\mathrm{opt}_s^{\mathrm{last}(d)}}|$ holds. For moving back to the start from $\mathrm{last}(d)$ we can use the same path back.

This gives a different lower bound for the optimal search ratio against the optimal offline exploration tour, which is:

$$\mathrm{SR}(\pi_{\mathrm{opt}}) \geq \frac{|\pi_{\mathrm{opt}_s^{\mathrm{last}(d)}}|}{d} \geq \frac{|\pi_{\mathrm{Expl}_{\mathrm{opt}}(d)}|}{2d} \Longleftrightarrow |\pi_{\mathrm{Expl}_{\mathrm{opt}}(d)}| \leq 2d \cdot \mathrm{SR}(\pi_{\mathrm{opt}}).$$

Altogether we attain a factor of

$$\begin{aligned} \frac{\sum_{i=1}^{j+1} |\pi_{\mathrm{Expl}_{\mathrm{onl}}(2^i)}|}{2^j} &\leq C_\beta \cdot \frac{\sum_{i=1}^{j+1} |\pi_{\mathrm{Expl}_{\mathrm{opt}}(\beta 2^i)}|}{2^j} \leq 2C_\beta \cdot \frac{\sum_{i=1}^{j+1} \beta 2^i \,\mathrm{SR}(\pi_{\mathrm{opt}})}{2^j} \\ &\leq 8\beta C_\beta \cdot \mathrm{SR}(\pi_{\mathrm{opt}}) . \end{aligned}$$

Theorem 3.24 *Let $\mathcal{E}$ be an environment where an agent with vision system is searching for a target. Let $\mathrm{Expl}_{\mathrm{onl}}(d)$ be a strategy for the depth restricted online exploration of $\mathcal{E}$ with $\mathrm{Expl}_{\mathrm{onl}}(d) \leq C_\beta \cdot \mathrm{Expl}_{\mathrm{opt}}(\beta d)$. We can search in $\mathcal{E}$ by the doubling heuristik (Algorithm 3.2) and attain a $8\beta C_\beta$-approximation of the optimal search path and search ratio.* [FKK+04]

With this general framework we can approximate optimal search path for polygons also in an online fashion. The main task is the design of exploration strategies which will be the subject of the next section.

For the negative side we will now show some examples where an agent (without a vision system) cannot approximate the offline optimal search path with a constant factor in the online version. Lower bounds are achieved by counter examples. For some graph configuration we show that the search ratio is constant (the competitive ratio is small) whereas any online strategy can be forced to make arbitrary large detours against the shortest path to some goals. In comparison Corollary 3.23 for the geometric search has used the property that the CFS Algorithm has running time of $(4+8/\alpha)|E(d)|$ for depth restricted exploration. If the goal set is restricted to the vertices, the result will not help us anymore.

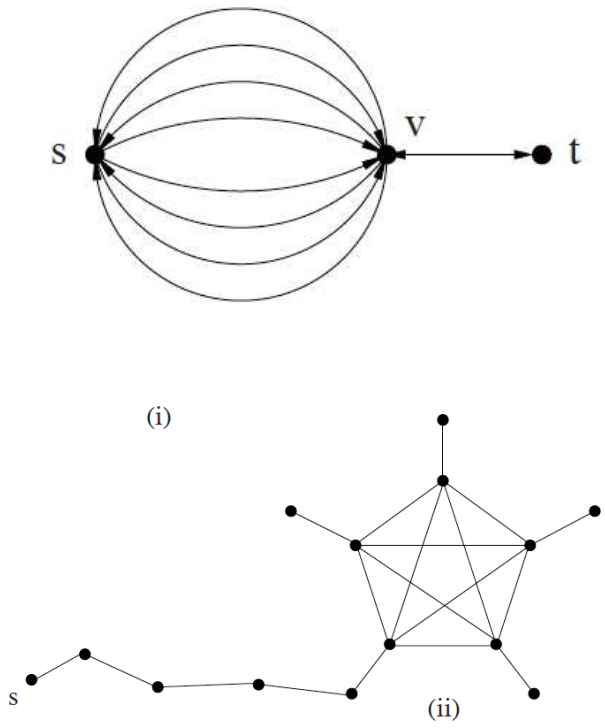


Figure 3.29: The optimal search path for goal set V cannot be approximated by a constant factor for (i) planar graphs with multiple edges and (ii) general graphs without multiple edges.

Theorem 3.25 *For the following graph configuration we can show that otimal offline search path cannot be approximated by an online search startegy with a constant factor.*

1. *Planar graphs $G = (V,E)$ with multiple edges and goal set V.*
2. *General graphs $G = (V,E)$ even without multiple edges and goal set V.*
3. *Directed graphs $G = (V,E)$ with goal set $E \cup V$.*

Exercise 25 *Show that for directed graphs $G = (V,E)$ with goal set $E \cup V$ a constant approximation of the optimal search path and search ratio is not possible.*

Exercise 26 *Consider planar graphs $G = (V,E)$ with goal set V. Does a constant approximation of the optimal search path and search ratio exist?*

Proof.

1. In Figure 3.29(i) the optimal search path visits vertices v and t with search ratio 1. Any online strategy will be forced to visit all multiple edges before t is visited. This gives a ratio of $\frac{k}{2}$ for arbitrary k.
2. In Figure 3.29(ii) the optimal search path visit the satellites of the k-clique from s in $3k$ steps. The distance from s to the clique is also k. This gives a search ratio of at most 4. An online strategy will be forced to visit all inner edges first, before the satellites will be visited. Therfore $\Omega(k^2)$ steps will be required and the search ratio is $\Omega(k)$.

□

The next paragraph will handle exploration strategies by an agent with a vision system. An analogous negative result (no constant competitive search path approximation) will be achieved for polygons with obstacles (or holes).

Interestingly, for all negative examples, there is already no constant competitive online exploration strategy for the corresponding goal set. This is extended to the negative result for the search path approximation. Altogether, the conjecture is that both statements are equivalent in general

Already proved: $\exists$ constant-competitive, (depth restricted) exploration strategy $\Rightarrow \exists$ online search strategy with constant search ratio approximation.

Conjecture: $\nexists$ constant-competitive exploration strategy and $\exists$ 'extentable' lower bound $\Rightarrow \nexists$ online search strategy competitive against search ratio.

An extension trick for the lower bound can be seen in Figure 3.29(ii), the path to the k-clique was extended such that the search ratio of the optimal search path is constant. A similar idea is applied for polygons with holes; see Figure 4.22.

Chapter 4

Exploration in polygons

We would like to consider the exploration task for polygons by an agent equipped with a vision system. The results can be applied to the framework of the preceding section. We are searching for a short path that *sees* all points in the polygon at least once. For a simple polygon the overall shortest such paths can be computed in polynomial time, if the polygon is given. There are also online algorithms that explores an a priori unknown simple polygon by a constant competitive strategy in comparison to the shortest offline path. For polygons with obstacles (holes) no such algorithms exist.

4.1 Simple polygons

A simple polygons is enclosed by a simple polygonal chain without self intersections. In the competitive sense we compare online exploration strategies with offline strategies.

The problem of computing the shortest round trip that sees all points in the polygon was introduced by 1986 by Chin and Ntafos as the Shortest Watchman problem; see [CN86]. Since then many authors have considered the Shortest Watchman Route (SWR) problem, sone of which have been erroneous. Other have been improved in the running time. Currently, it is meant to be common sense that the following result gives the best algorithm.

Theorem 4.1 *(Dror, Efrat, Lubiw, Mitchell, 2003)*
For a simple polygon with n vertices and a start point s, there is an algorithm that computes the Shortest Watchman Route in time $O(n^3 \log n)$. *[DELM03]*

First, we consider simple polygons and within this class of polygons special subclasses; see Figure 4.1. Polygons of these classes allow efficient computations.

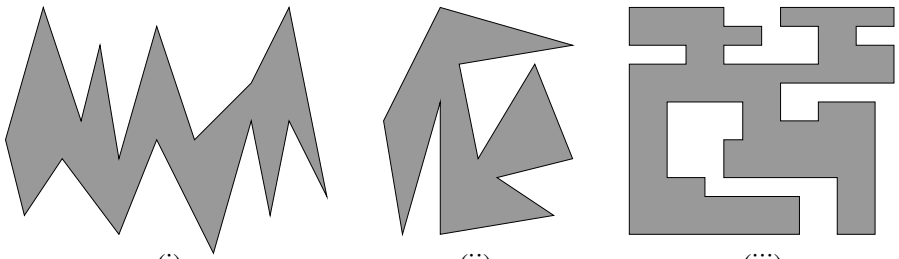


Figure 4.1: (i) X–monotone polygon, (ii) non–monotone polygon, (iii) rectlinear polygon.

Definition 4.2 A simple polygon P is denoted as **monotone**, if there exists a line ℓ, such that for any line l' orthogonal to l the intersection $P \cap l'$ is path-connected. This means that the intersection $P \cap l'$ is a single segment, a point or empty. If ℓ is in parallel to the Y-axis, the polygon P is denotde as **y–monotone**.
A simple polygon P is denoted as **rectilinear**, if any inner angle is either of 90° or of 270°.

The most simple case for the computation of a SWR is given for monotone and rectilinear polygons:

Theorem 4.3 *(Chin, Ntafos, 1986)*
For a rectilinear and monotone polygon, the SWR can be computed in $O(n)$ time. *[CN86, CN88]*

Exercise 27 *Present a linear time algorithm for the proof of the above Theorem.*

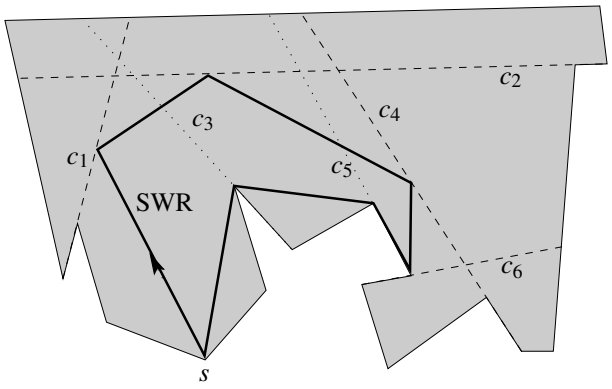


Figure 4.2: A polygon and necessary cuts (dotted), essential cuts (dashed) and the Shortest Watchman Route.

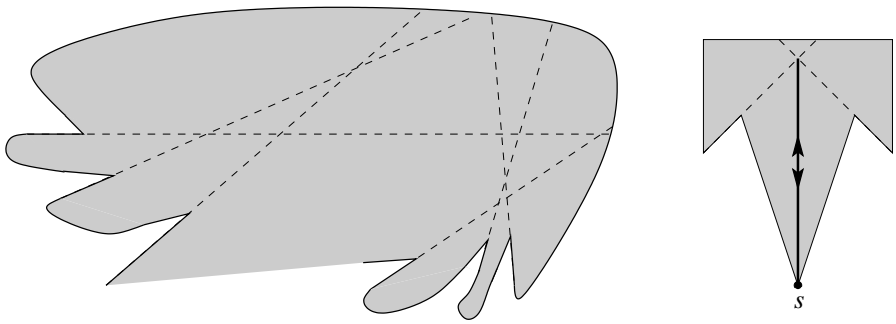


Figure 4.3: (i) A "corner" situation: Several cuts intersect and in a row and a single cut intersects more than one other cut. (ii) A polygon and its SWR.

In general, for the computation of the SWR we can concentrate on discrete parts of the polygon. It suffices to visit the *essential cuts*, defined as follows. The invisible parts of the polygons lie behind reflex vertices, i.e., vertices with inner angle larger than π.

Definition 4.4 Consider the extension of a an edge of a reflex vertex that points into the inner part of the polygon until it hits the boundary. Such segments are denoted as **cuts**. For the two cuts starting at a reflex vertex the cut emanating from the invisible edge (w.r.t. the start point) has to be crossed in order to see both edges. These cuts are called **necessary cuts**. For a necessaryl cut c_i let P_{c_i} denote the sub-part of the polygon P behind c_i w.r.t. the start point. The agent has to move inside P_{c_i}. A necessary cut c_i **dominates** a necessary cut c_j if $P_{c_i} \subset P_{c_j}$ holds. In this case any path from the start that visits P_{c_i} visits the larger polygon P_{c_j} first. A necessary cut c_i that is not dominated by any other necessary cut is denoted as an **essential** cut. It is clear that for the SWR it is sufficient to visit all essential cuts.

Necessary cuts that will be dominated will be explored on the path to the corresponding essential cut. Figure 4.2 shows an example with necessary and essential cuts. Here c_3 and c_5 are not essential, any path to c_4 will visit the cuts. More precisely, c_4 dominates c_3 and c_5. With the help of the cuts we can formulate some structural properties:

(i) The SWR and any other exploration tour has to visit all essential cuts. The set of essential cuts is the smallest set of cuts that has to be visited for seeing the whole polygon.
(ii) If the essential cuts do not intersect, they have to be visited in their order along the boundary. In this case from the SWR the cuts will be visited by specular reflection. The incoming angle for the visit of each cut is the same as the outgoing angle.
(iii) If some essential cuts intersect in a row, we call this a "corner" situation. In this case it might happen that some cuts are just passed by the SWR and are not visited by specular reflection; see Figure 4.3. This makes the corner situation difficult.

For a polygon and a start point s we can order the cuts by the order they appear along the boundary, independent from the position of the corresponding reflex vertex; see Figure 4.2. In the corner situation the SWR need not visit the essential cuts in this order; see Figure 4.4.

Interestingly, the corresponding polygons P_{c_i} are still visited in the order of the corresponding cuts. In Figure 4.4 we have the visiting order $P_1, P_2, P_3, P_4, P_5, P_6$. This is meant as follows.

Although, we first enter P_3 the SWR actually visits P_1 and P_2 at a single point first. By chance we are also in P_3 at this point, visit P_3 immediately and the order is maintained in this sense. Pre-visits do not count. This means that the task is: Computed the shortest tour that visits the polygons P_{c_i} by the order along the boundary.

4.2 Rectilinear polygons

We will pick up the above idea later on. First we consider the simple case of a rectilinear polygon. In a rectilinear polygon everything is less complicated. We do not have complicated corner situations. Essential cuts have successive intersections for max three orthogonal cuts; see Figure 4.4. We conclude.

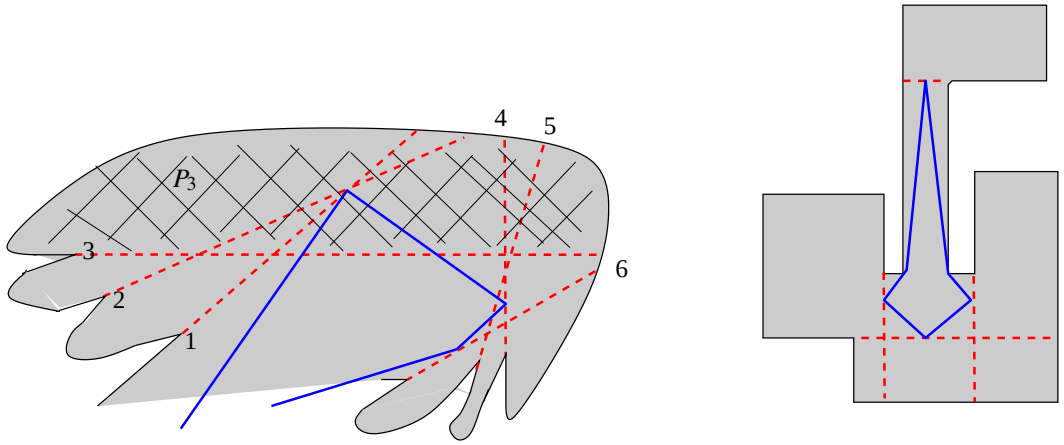
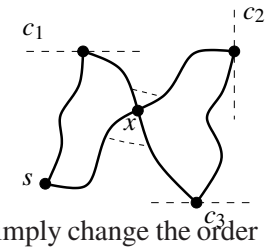


Figure 4.4: In a corner situation the SWR visits the polygons P_{c_i} (here P_i) by the order of the corresponding essential cuts along the boundary. In rectilinear polygon essential cuts will never be passed.

Lemma 4.5 *For a rectlinear polygon the SWR visits the essential cuts by the order along the boundary.*

Proof. For the rectinlinear case a corner situation can occur with maximal three cuts, where the first and the last one run run in parallel and do not intersect. Moving into the corresponding polygons P_{c_i} gives a detour. It is needless to pass a cut in order to reach another cut. Therefore all cuts will be visited one after the other.



Assume that the visits do not follow the order along the boundary. In this case the SWR R will have an intersection somewhere; see the Figure. We can simply change the order locally in order to obtain R' that runs from s c_1, x, c_2, c_3, x to s. This is also a tour that has the same

property. In the vicinity of the intersection x we can even locally (and globally) improve the tour by some shortcuts. □

Lemma 4.5 gives the key-idea for the computation of the SWR:

Theorem 4.6 *(Chin, Ntafos, 1986)*
The Shortest Watchman Route in a simple, rectilinear polygon can be computed in $O(n)$ time. [CN86, CN88]

Proof. Algorithm 4.1 computes the SWR in a rectilinear polygon, Figure 4.5 shows an example.

The essential cuts can be commputed in $O(n)$ time (exercise left to the reader). It has to be shown that P'', has no more than $O(n)$ edges or triangles. All other running times stem from standard offline algorithms for polygons. We consider dual graph, T^*, of the triangulation. Successively visiting the cuts along the corresponding triangles is simply Depth-First runs through T^*. Any edge of T^* is visited exactly twice. This means that also any triangle occur only twice in P'', the number of triangles and edges in P'' is in $O(n)$. □

Exercise 28 *Show that the essential cuts in a rectilinear polygon of n vertices can be computed in $O(n)$ time.*

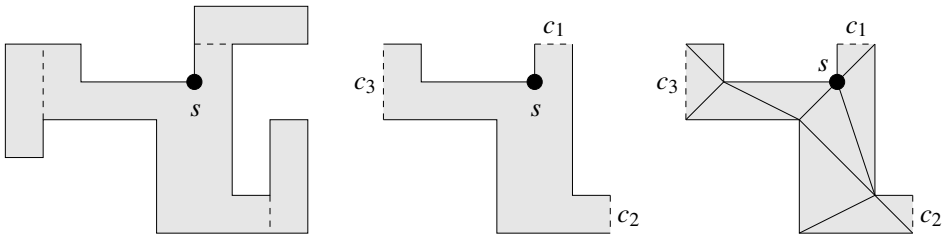
Algorithm 4.1 Shortest Watchman Route for rectilinear polygons

- Compute the essential cuts $c_1, \ldots, c_k$ and order them along the boundary from s. $O(n)$
- Cut of the corresponding sub-polygons P_{c_i} behind the cuts. This gives a polygon P'. Some of the essential cuts get smaller. $O(n)$
- Compute a triangulation for P'. $O(n)$
- Compute a chain of triangles P'' by the "Roll–Out" of P': $O(n)$:
  - Let $P^{(1)}$ be the relevant triangles of P' along the path in the dual graph of the triangulation, T^* , from s to c_1.
  - For any essential cut c_i, $i = 2, \ldots, k$: Extend $P^{(i-1)}$ to $P^{(i)}$ by the chain of the relevant triangles along the boundary of P' on the path from c_{i-1} to c_i and reflected at the c_{i-1}.
  - Extend $P^{(k)}$ to P'' as above by the relevant triangles on the path from c_k to s and by reflection on c_k. There will be a copy s' of s.
- P'' is a sequence of triangles. Compute the shortest path π from s to s' in P''. $O(n)$
- The SWR can be build by mirroring back the line segments of the path at the cuts c_i.

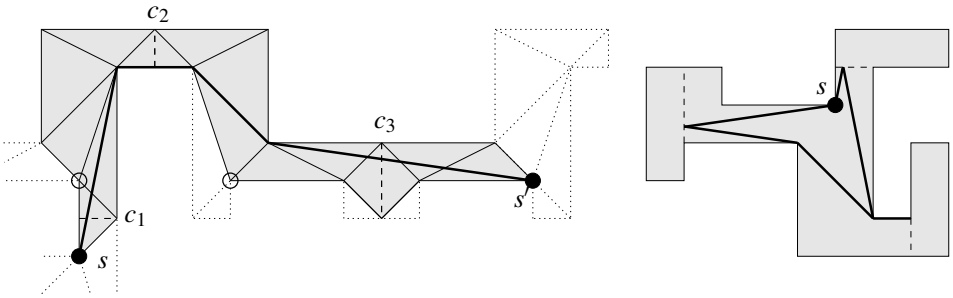
Algorithm 4.1 can be applied to any polygon in the same way, if any essential cut of the polygon intersect with exactly one other essential cut. In this case Lemma 4.5 holds. In general polygons this will not be the case. Many essential cuts can intersect in a row with multiple intersections of a single cut with others. We call such situations a "corner" situation. In a corner, the order of the visits of the cuts is non-trivial.

First, we would like to argue, that the above algorithm can be easily made depth-restricted. For this we only have to restrict the set of essential cuts. An essential cut blocks the visibility of points closely behind the reflex vertex of the cut. We consider a non-visible point that has the closest distance to the start s. In principle this point is arbitrary close to the reflex vertex. So the distance to the reflex vertex gives the distance to the cut. In Figure 4.6(i) the rightmost essential cut has distance $l > d$.

We would like to see all points in P with distance less than or equal to d from s. Let $P(d)$ denote this part of P. Obviously, it is sufficient to visit all essential cuts that has a distance $\leq d$. $> d$ to the start s; see Figure 4.6(ii). We apply the same algorithm.



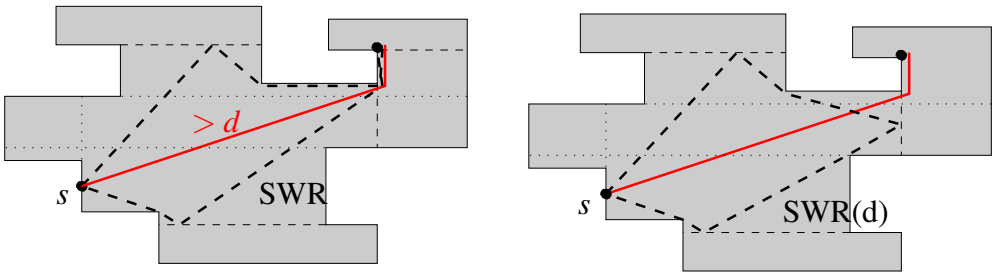
(i) Wesentliche Cuts (ii) ”Au”seres abschneiden (iii) Triangulation



(iv) Roll-Out und k”urzester Weg (v) Shortest Watchman Route

Figure 4.5: Computing the SWR in a rectinlinear polygon.

For simple, rectilinear polygons we conclude: $\text{Expl}_{\text{OFF}}(d) = \text{Expl}_{\text{opt}}(d)$. This means that for the offline case we have $\beta = 1$ and $C_\beta = 1$ for the exploration of $P(d)$ and the application of Theorem 3.24 gives an 8-approximation of the optimal search ratio. Suchpfades.



(i) (ii)

Figure 4.6: Computation of the SWR for all points with distance $\leq d$ from s in a rectilinear polygon. It is sufficient to ignore all cuts of distance $> d$.

In the online version of the problem, the poygon is a priori not known. Nevertheless, we can design an efficient online algorithm. There are no corner situations and we can visit the cuts of the reflex vertices by the Greedy-Algorithm 4.2; see also Figure 4.7. Starting from s at the boundary we successively expand the visible part of the boundary and always approach the next reflex vertex by a move orthogonal to its cut. This gives an L_1-optimal exploration path. We have the following result:

Theorem 4.7 *(Deng, Kameda und Papadimitriou, 1991)*
A simple, rectilinear polygon can be explored online optimally w.r.t. the L_1-metric and with a competitive ratio of $\sqrt{2}$ w.r.t. the L_2-metric [1]. *[DKP98]*

[1] For the L_1-metric or Manhatten-metric the distance between two points $p = (p_x, p_y)$ and $q = (q_x, q_y)$ is defined by $d(p,q) := |p_x - q_x| + |p_y - q_y|$; in the L_2- or Euclidean metric we have $d(p,q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$.

Algorithm 4.2 Online exploration of a rectlilinear polygon

while Polygon is not fully explored **do**
  Consider the next reflex vertex along the boundary in cw order.
  Move orthogonally to the corresponding cut.
end while

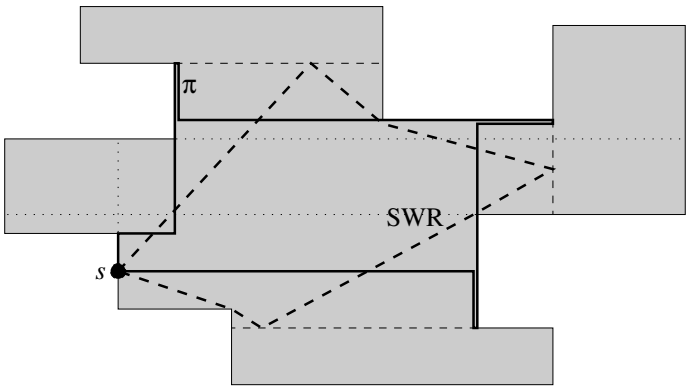


Figure 4.7: Path of the online heuristic and the SWR w.r.t. the L_2-metric in a rectilinear polygon.

Proof. We give a sketch of the proof. The Greedy-exploration approach give an optimal L_1-path, since the algorithm successively creates locally optimal L_1-paths. In the first step the first cut will be visited orthogonally by an optimal L_1-paths. Assume that we are already along an optimal L_1-path and have visited a set of cuts in this fashion. The next cut is again visited orthogonally on the shortest L_1-path. By induction the agent moves along an overall shortest L_1-path for visiting the necessary cuts.

We still have to move back. For this we simply assume that at the start point s there is an *artificial* necessary cut. Also this last cut will be visited by an optimal L_1-path, which gives an overall optimal L_1 round trip.

For the comparison to the optimal L_2-SWR, we use the following sketch. Also the L_2-SWR visits the essential cuts in the order along the boundary. We shift the L_1-path to the outer boundary such that the L_2 SWR path is included and the L_1-path still has the same length. For any two point of a segment of the L_2-SWR there is an optimal L_1-path which can be considered to consist of two segments, we only have to check detours of triangles; see Figure 4.9.

Thus, we consider a single triangle and by scaling we can argue that we have to consider the maximum of the $f(x,y) = x+y$ for $x^2+y^2 = 1$. This means that $f(y) = y+\sqrt{1-y^2}$ has to be maximized. We have $f'(y) = 1-\frac{y}{\sqrt{1-y^2}}$ and the f' gets 0 for $y_{\max} = \frac{1}{\sqrt{2}}$. This is a maximum of f and we have $x_{\max} = \frac{1}{\sqrt{2}}$ and $f(x_{\max}, y_{\max}) = \sqrt{2}$. □

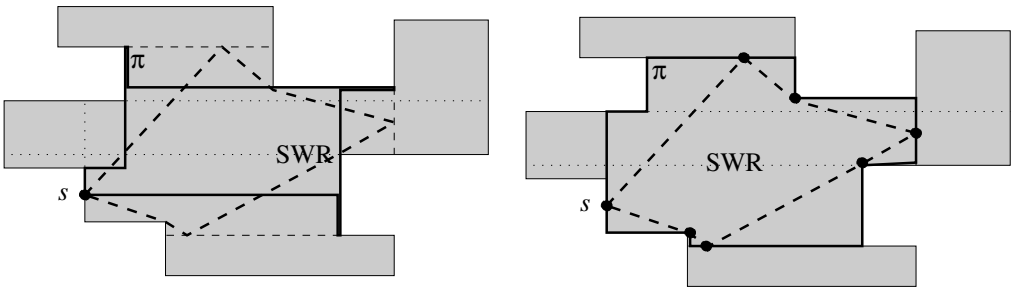


Figure 4.8: *Shifting* an L_1-optimal path, such that the L_2-SWR is inside. The analysis of the detour for triangles is sufficient.

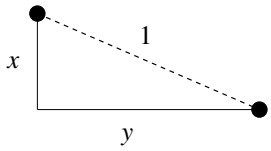


Figure 4.9: The worst-case detour in a triangle is $\sqrt{2}$.

For the online variant we can restrict the algorithm to explore the polygon up to depth d. As before we ignore all cuts where the reflex vertex has distance $> d$. The $\sqrt{2}$-approximation against the SWR up to depth d remains valid. Therefore for the application of Theorem 3.24 we conclude $\beta = 1$ and $C_\beta = 1$ and attain a $8\sqrt{2}$-approximation of the search ratio and the optimal search path.

Corollary 4.8 *The optimal search path in a simple, rectilinear can be approximated within a factor of* 8 *in the offline case and within a factor of* $8\sqrt{2}$ *in the online case.*

4.3 General simple polygons

As in the previous section we first concentrate on the offline computation of a SWR in a simple polygon. As already shown in Figure 4.4 the sub-polygons P_{c_i} of the essential cuts will be visited in the order along the boundary. More generally we extract the following general computation task, which finally ends in Theorem 4.1. A touring-a-sequence-of-polygons gives a generalization of the SWR computation.

Definition 4.9

(i) In the simple **Touring Polygon Problem** (TPP) version a sequence of simple, convex and disjoint polygons $P_1, P_2, \ldots, P_k$ with n edges in total is given. Furthermore, a start point s and a target point t is fixed. We are searching for the shortest path that starts in s, visits the polygons P_i in the order given by the index i and ends in t.

(ii) In the general version of the TPP, the path between two successive polygons P_i und P_{i+1} ($i = 0, \ldots, k$; $P_0 := s; P_{k+1} := t$) can be forces to run in a so-called **fence-polygon** F_i. The fence F_i is a simple polygon that contain P_i and P_{i+1}. Additionally, the polygons might overlap, i.e., the intersection of P_i and P_j need not be empty. In the presence of a fence for polygons P_i and P_{i+1}, it is allowed that only the boundary parts of P_i and P_{i+1} that do not belong to the boundary of the fence form a convex chain. We call this part the **facade** of P_i or P_{i+1}, respectively. More precisely facade$(P_i) := \partial P_i \setminus \partial F_{i-1}$.

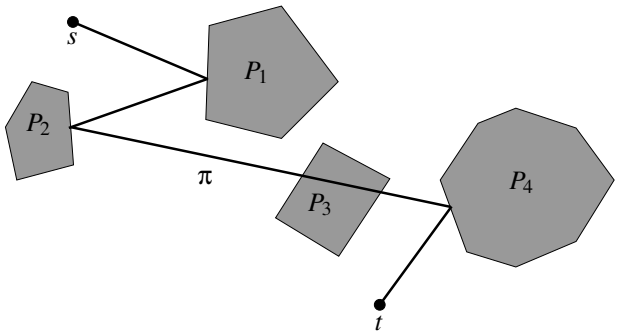


Figure 4.10: An example for the simple version of the Touring Polygon Problem.

The interpretaion of the TPP is as follows: It can happen that for $j < i$ a polygon Polygon P_i has been visited by chance before polygon P_j is visited, the first visit will be ignored, the polygon P_i has to be visited again. More precisely the visit of P_i is valid, if the polygons $P_1, \ldots P_{i-1}$ have been visited

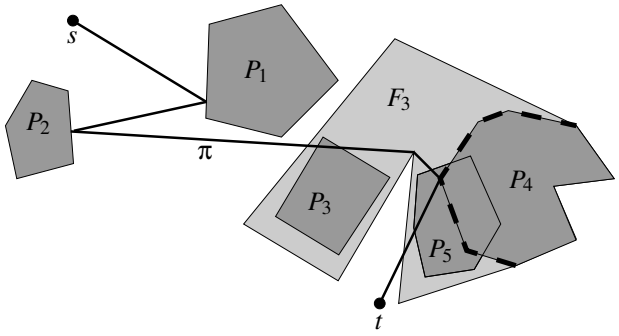


Figure 4.11: An example for the general Touring Polygon Problem.

in this order before, Figure 4.10 shows an example for the simple TPP configuration and Figure 4.11 examplifies the general case. The dashed part of the boundary of P_4 is the facade of P_4. Note, that P_5 was visited before P_4 is entered, we "register" the visit of P_5 after P_4 was visited

Theorem 4.10 *(Dror, Efrat, Lubiw, Mitchell, 2003)*
For the general TPP with k polygons, $k+1$ fences and n edges in total for all polygons and fences there is an algorithm that computes a query structure for the TPP in $O(k^2 n \log n)$ time. The query structure has a complexity of $O(kn)$. For a fixed start point s and for any query target point t the shortest TPP path can be computed in k"urzeste TPP-Pfad Zeit $O(k \log n + m)$ where m denotes the number of segments of the shortest TPP path. *[DELM03]*

Now let us come back to our initial SWR problem. We now sketch the proof of Theorem 4.1.
Proof. Let us assume that P and a start point s on the boundary is given. We construct a TPP input $(P_1,\ldots,P_k,F_1,\ldots,F_k,s,t)$ as follows. Let c_i be the i-th essential cut of P along the boundary of P and P_{c_i} the corresponding sub-polygon. We set $P_i := P_{c_i}$. Any fence will be the polygon P itself, which is $F_i := P$. The facades of any F_i is given by the cut c_i. Finally, we set $t := s$ for returning to the start. The SWR is the shortest path that starts at s visits the polygons P_i in the given order inside the polygon P and ends at s. The cuts c_i build convex facades for the possibly non-convex polygons P_i. This gives exactly the task in the corresponding TPP. The complexity of the facades is in $O(1)$ and the complexity of the fence is in $O(n)$. We can have $\Omega(n) = k$ many polygons P_{c_i}. The running time is in $O(n^3 \log n)$. □

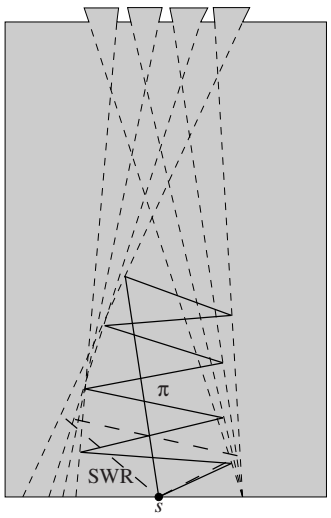


Figure 4.12: A greedy-exploration of the reflex vertices is not competitive in a non-rectilinear polygon.

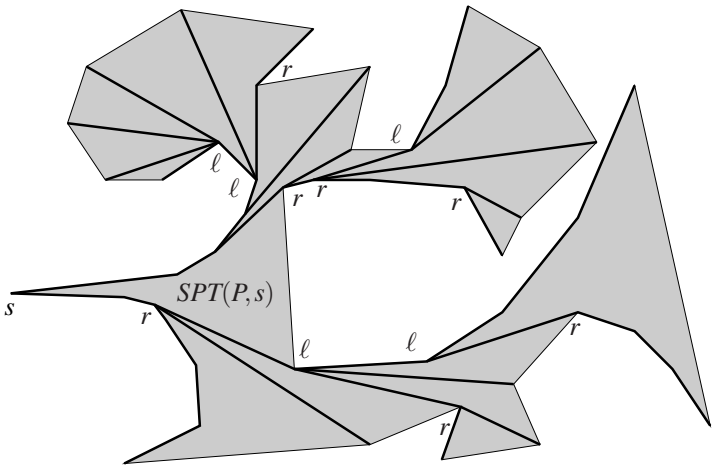


Figure 4.13: Polygon, Shortest Path Tree and examples for right and left reflex vertices.

Finally, we consider the online exploration of general simple polygons. The greedy approach for rectilinear polygons explored the cuts of the reflex vertices in the order (of the vertices!) along the boundary. Let us assume that in a general polygon we get some more information and all cuts are given. If we explore the cuts in the order of the corresponding reflex vertices and construct the shortest (optimal) path for this visiting order, the corresponding path can be arbitrarily large in comparison to the SWR of the polygon. Figure 4.12 shows an example where the greedy approach with additional information does not succeed w.r.t. a constant competitive approximation.

Figure 4.12 also shows that it makes sense to *bundle* the reflex vertices and subdivide them into cuts that will be detected if the agent moves to the left and cuts that will be detected, if the agent moves to the right. This is what the corresponding SWR does in principle. We would like to formalize this idea by categorizing the reflex vertices correspondingly.

Definition 4.11 Let P be a simple polygon and s be a start point at the boundary of P. The **Shortest Path Tree**, SPT(P,s), contains the shortest paths inside P that runs from s to the all vetices of P. The SPT is the smallest set of segments that contains all the paths. W.r.t. the SPT a reflex vertex v of P is denoted as a **left vertex**, if the SPT(P,s) makes a counter clockwise turn at v and **right vertex**, if the SPT(P,s) makes a clockwise turn at v; see Figure 4.13. The interpretation is that w.r.t. the path from s, v lies to the left or v lies to the right of the preceding vertex.

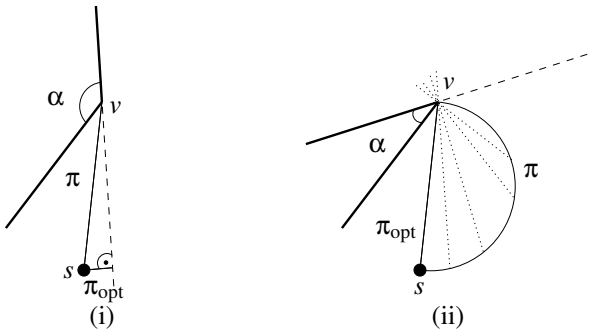


Figure 4.14: Looking around the corner in a competitive fashion.

Different from the rectilinear case we will not approach the reflex vertices orthogonally, we make use of circular arcs. Consider Figure 4.14. The agent is located at s and detects the reflex vertex v. The angle α for the cut is unknown because vertex v blocks the corresponding edge. Assume the agent moves directly toward v. An adversary will choose a very large angle α — as in Figure 4.14(i) — such that an

arbitrary short path orthogonal to the cut is sufficient. In this sense the direct path to the vertex is not competitive.

Therefore we explore the vertex (or its cut) by a half-circle starting at s around the midpoint of sv and radius $|sv|/2$. For approaching the cut this gives a competitive ratio of at most $\frac{\pi}{2}$. For the above looking-around-a-corner problem the exploration by the half-circle is not the overall best strategy, as will be shown in the next section. In comparison to the optimal corner strategy the half-circle strategy can be easily analysed and has nice properties against the shortest path.

The half-circle exploration is not the overall best strategy for looking around a corner. A refined analysis shows the following result:

Theorem 4.12 *(Icking, Klein, Ma, 1993)*
The problem of looking around a corner can be solved within an optimal competitive ratio of ≈ 1.212. *[IKM94]*

We first formally show the competitive ratio of the half-circle strategy for detecting the cut and also give a simple lower bound.

Theorem 4.13 *The unknown cut of a reflex vertex in a simple polygon can be detected by a half-circle strategy within a competitive ratio of* $\pi/2$ *against the shortest path to the cut. It can be shown that there is no online strategy that explores any corner (visit the cut) within a ratio less than* $\frac{2}{\sqrt{3}}$.

Proof. We consider the normalized version of the problem from Figure 4.15. For the offline optimal solution either the vertex O will be visited directly or the cut will be approached orthogonally, the cut is not know which is indicated by the unknown angle φ. for $\varphi \in [0,\pi_2]$ the orthogonal distance $\sin\varphi$ gives the optimal solution. For $\varphi \in [\pi_2,\pi]$ the shortest path to O of length 1 is optimal.

We compare the optimal solutions to the half-circle strategy for any φ. Until the half-circle finally hits O at angle $\varphi = \pi/2$ (and therefore for all $\varphi \in [\pi_2,\pi]$), the half-circle strategy has arc length φ for any $\varphi \in [0,\pi_2]$. For all possible cuts with angle $\varphi \in [\pi_2,\pi]$ we attain a ratio $G(\varphi) = \frac{\pi/2}{1}$. For the case $\varphi \in [0,\pi_2]$ we have $H(\varphi) = \frac{\varphi}{\sin\varphi}$. The first derivatives gives $H'(\varphi) = \frac{\sin\varphi - \varphi\cos\varphi}{\sin^2\varphi}$ and by simple analysis we have $H'(\varphi) > 0$ for $\varphi \in (0,\pi/2]$. Therefore in both cases the ratio $\frac{\pi}{2}$ is the worst-case.

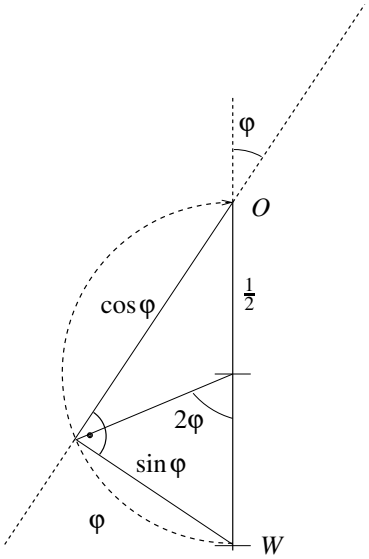


Figure 4.15: The optimal path to the unknown cut either is given by the direct path to O of length 1 for $\varphi \in [\pi_2,\pi]$ or is given by an orthogonal path of length $\sin\varphi$ for $\varphi \in [0,\pi_2]$. For the half-circle strategy the worst-case ratio is attained at $\varphi = \pi/2$ with a ratio of $\pi/2$.

For the lower bound we consider Figure 4.16. We provide a bit more information for the online strategy. Either φ is exactly $\frac{\pi}{6}$ or $\varphi = \frac{\pi}{2}$. In the first case the optimal path has length $\sin \pi/6$ and in the second case the optimal path has length 1.

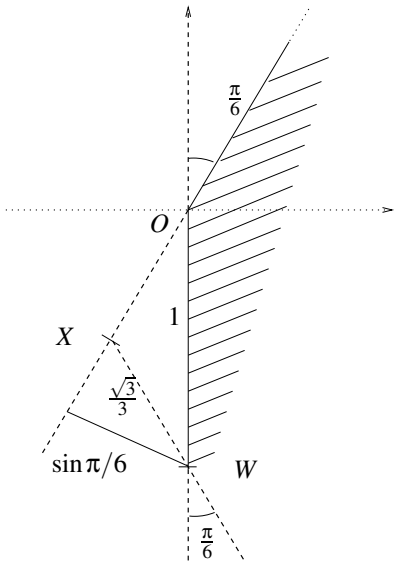


Figure 4.16: The lower bound construction gives a ratio of $\frac{2}{\sqrt{3}}$. If the strategy visits the $\pi/6$-cut to the right to X, the $\pi/6$-cut is the given cut. If the strategy visits the $\pi/6$-cut to the left to X, the $\pi/2$-cut is the given cut. Both cases gives a ratio of $\frac{2}{\sqrt{3}}$.

Any strategy will visit the $\pi/6$-cut somewhere (may be also at the end at point O). Therefore we consider the isosceles triangle with ground length OW and two angles of size $\pi/6$. This means the one segment if the triangle runs in parallel with the $\pi/6$. Consider the vertex X of the triangle on the $\pi/6$-cut. Either an online strategy visits the $\pi/6$-cut to the left or to the right of X. Both cases might include that exactly X is visited.

In the first case (visit to the left of X), the adversary present the $\pi/2$-cut as the true cut and the agent now moves toward O. The optimal path has length 1, whereas the strategy runs at least $2 \cdot \frac{1}{2\cos \pi/6} = 2 \cdot \frac{\sqrt{3}}{3} = \frac{2}{\sqrt{3}}$. In the latter case the (visit to the right of X), the adversary present the $\pi/6$-cut as the true cut, the ratio is at least $\frac{\frac{1}{\sqrt{3}}}{\sin \pi/6} = \frac{2}{\sqrt{3}}$. In both situations the same worst-case ratio is attained. □

We will now sketch the ideas for the competitive only exploration of a general polygon by a recursive subdivision of the reflex vertices in groups of left and right vertices and by a consequent successive exploration of the groups by half-circles.

By Algorithm 4.3 we explore a *single* right vertex. The strategy manages two lists of vertices. The *TargetList* contains right vertices that have been detected (but not explored) ordered in ccw-order along the boundary. Right vertices, that will be detected by *ExploreRightVertex* and that do not lie behind left vertices of the SPT, will be inserted into *TargetList* during the execution of *ExploreRightVertex*. It might happen that the goal vertex *Target* changes during the execution. In this sense *ExploreRightVertex* does not only explore a single right vertex, the target changes. The exploration is restricted to a set of right vertices that subsequently lie along the boundary such that no left reflex vertex occurs in between. The goal is to explore all vertices of the sequence. We consider the exploration as shown in Figure 4.17 and exemplify the usage of Algorithm 4.3.

The agent starts in s. We initialize *BasePoint* by s and *TargetList* contains only r_1. The target r_1 is visible. *Back* is also s. We follow the half-circle $\text{arc}(s, r_1)$ until the next right vertex r_2 is detected at e_1 (the first event). Since r_2 does not lie behind a left vertex and lies in ccw-order behind r_1 we insert r_2 into the target list *TargetList*. In cw-order r_2 lies in front of r_1 and r_2 is the first element of the target list. Therefore there is an update of *Target* and the agent now moves along the half-circle $\text{arc}(s, r_2)$. At

Algorithm 4.3 Exploration of a right vertex.

ExploreRightVertex(*TargetList*, *ToDoList* **):**

```
BasePoint := current position of the agent.
Target := first vertex of TargetList.
while Target is no longer visible do
  Move along shortest path from BasePoint toward Target.
end while
Back := last polygon vertex reached along the
  shortest path from BasePoint to the current position.
while Target is not fully explored do
  Move along halfcircle arc(Back, Target) in cw-order.
  Update TargetList, ToDoList, Target, Back during the task.

  // Special situations during the half-circle move:
  if the boundary of P blocks the move then
    Follow the boundary until the half-circle can be continued.
  end if
  if Target will get out of sight by a vertex then
    Move toward Target to the vertex, that blocks the sight.
  end if
end while
```

the second event e_2 the visibility to the current target gets blocked by ℓ_1, which is the second special situation. The agent moves toward the target r_2 to ℓ_1. At ℓ_1 we update $Back := \ell_1$, since we reached a polygon vertex. Now we move along the arc $\text{arc}(\ell_1, r_2)$. Close behind ℓ_1 the *BasePoint* s is no longer visible and still $Back := \ell_1$ remains true. At event e_3 the s gets visible again, we are no longer at a vertex and we set $Back := s$. Note that the *BasePoint* will be s all the time. At event e_4 the current *Back* point s vanishes again, we set $Back := r_1$ and run along the half-circle $\text{arc}(r_1, r_2)$. At event e_5 the next right reflex vertex r_3 in ccw-order is detected, inserted into *TargetList* and gets the new *Target*. Therefore we move along the half-circle $\text{arc}(\ell_1, r_3)$ until at e_6 the r_1 (*Back*) gets out of sight and we set $Back := r_2$ and continue with $\text{arc}(r_2, r_3)$. This movement is blocked betweem e_7 and e_8 from the boundary where the first special situation is used and the agent follows the boundary until picking up $\text{arc}(r_2, r_3)$ again. Finally, r_3 is fully explored. The vertex r_3 defines an essential cut that dominates the cuts of r_1 and r_2.

Since the current target *Target* is explored, the procedure Algorithm 4.3 ends. It might be the case that *TargetList* still contains non-explored reflex vertices. In general the procedure Algorithm 4.3 is part the procedure *ExploreRightGroup* that explores such a group of reflex vertices successively by Algorithm 4.3 with corresponding *BasePoint*s.

Algorithm 4.4 Exploration of a group of right vertices.

ExploreRightGroup(*TargetList*, *ToDoList* **):**

```
StagePoint := current position of the agent.
ToDoList := ∅
while TargetList is non-empty do
  ExploreRightVertex( TargetList, ToDoList ).
  For the current cut, move along the point on the cut that has the shortest distance (in P) to the
  StagePoint, update TargetList and ToDoList.
end while
Move along the shortest path (in P) back to StagePoint.
```

We exemplify *ExploreRightGroup* (Algorithm 4.4) and its interplay with *ExploreRightVertex* by Figure 4.18. Beginning at s as the current stage point and with the first single target r_1 in the target list

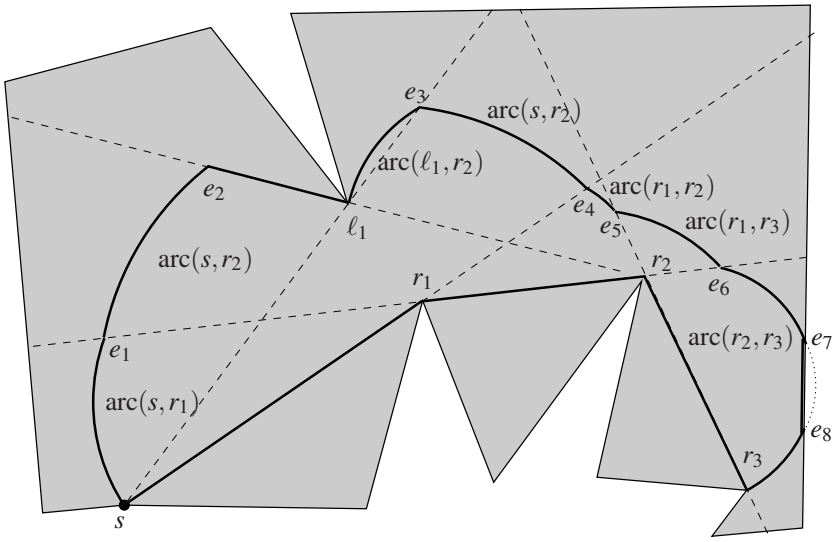


Figure 4.17: Exploration of a right vertex.

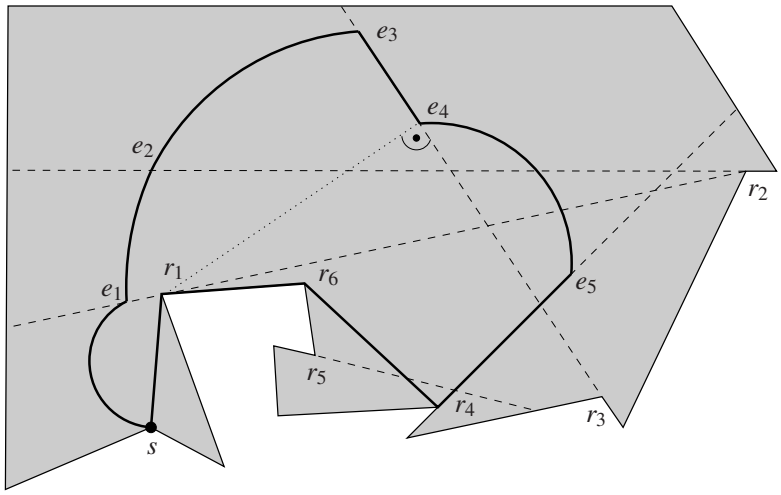


Figure 4.18: Exploration of a group of right vertices.

we start *ExploreRightVertex*. Analogously, to the above description at event e_1 the vertex r_2 is detected and $\text{arc}(s,r_2)$ is started. r_2 is the new *Target* Since the vertices r_6 and r_3 which are detected during the movement of $\text{arc}(s,r_2)$ up to e_2 do not lie behind (in ccw-order) r_2 they will not become new targets in the procedure *ExploreRightVertex*. In the current target list *TargetList* r_6 and r_3 lie behind r_2. In e_2 the current target vertex r_2 is fully explored and *ExploreRightVertex* ends here. Fortunately, w.r.t. the current *Back* point s the segment se_2 is orthogonal to the cut of r_2, the agent is located at the point on the cut with the smallest distance back to the *Back* point.

Now in *ExploreRightGroup* the procedure *ExploreRightVertex* is called up again with r_3, r_6 in the target list. This exploration ends at e_3. The vertices r_4 and r_5 are detected (and inserted in the target list) during the walk along $\text{arc}(s,r_3)$. Note that r_6 is deleted during an update of the list. r_6 was expored. The vertices r_4 and r_5 do not lie behind r_3 and therefore first r_2 is fully explored and *ExploreRightVertex* ends again.

In between the *Back* point has changed to r_1. Now w.r.t. the cut of r_3 at e_3 the agent is not located at a point on the cut of r_3 that has the shortest distance to the stage point s (and also to the current *Back* point). Therefore we move to such a point e_4 along the cut of r_3. This movement is part of the *ExploreRightGroup* procedure. Now *ExploreRightVertex* is applied with target r_4 and current back point

r_1 so that the arc arc(r_1, r_4) is used until this procedure end at e_5.

The current back point has changed to r_6 and the *ExploreRightGroup* forces the agent to slip along the cut of r_4 to move to the point closest to r_6 and s. The *TargetList* is updated in between and also r_5 is deleted out of *TargetList*. Now the *TargetList* is empty and in our case we return to the *StagePoint* which is s in this case.

The procedure *ExploreRightGroup* is used in the frame procedure Algorithm 4.5. This procedure builds up groups of left and right vertices which are explored in an alternating way. The usage of *ExploreRightGroup* goes into the depth in the sense that there is a list of stage points (*StagePoint*) (back points on the shortest path back to s) stored in the *ToDoList* that will be used as starting points for the procedure *ExploreRightGroup*. Analogous procedures for the exploration of left vertices and groups of left reflex vertices will be used.

The main procedure starts with the exploration of a right group from the start and returns to the start. After that all known left vertices are ordered along the boundary and the same group procedure is called for the left vertices from the start. Then the recursion starts by moving to the stage points and recall the procedures from there.

Algorithm 4.5 Exploration of simple polygons.

ExploreRightGroupRec(*TargetList* **):**
ExploreRightGroup(*TargetList*, *ToDoList*).
for all Vertex v in *ToDoList* **do**
Move along the shortest path to v.
NewTargetList := all detected left vertices,
which are successor of v in the SPT.
ExploreLeftGroupRec(*NewTargetList*).
end for

ExplorePolygon(P**,** s **):**
TargetList := right vertices visible from s, sorted in cw-order
along the boundary of P .
ExploreRightGroup(*TargetList*, *ToDoList*)
TargetList := detected left vertices, lying behind (in the SPT) the vertices
of *ToDoList* .
Additionally add all from s visible left vertices to *TargetList*.
Sort *TargetList* in ccw-order.
ExploreLeftGroupRec(*TargetList*).

Theorem 4.14 *(Hoffmann, Icking, Klein, Kriegel, 1998)*
The strategy PolyExplore explores an unknown simple polygon within a competitive ratio of 26.5 against the SWR. *[HIKK01]*

The ratio of 26.5 might appear to be huge, in fact it is an improvement of the ratios 133 (Hoffmann et al. [HIKK97]) or 2016 (Deng et al. [DKP91]) previously known. Indeed, the ratio is merely a result of the analysis. The best known lower bound for the strategy was given by an example where the ratio is roughly 5. The conjecture is that the ratio of the strategy is indeed close to 5, whereas a full proof can only be given for 26.5.

The online and the offline strategies given above can be easily restricted to a depth d. As mentioned before it suffices to ignore all reflex vertices with distance $> d$. This means that the approximation factors of 26.5 and 1 remain valid for the depth-restricted case. Note that the SWR for depth d might leave $P(d)$; see Figure 4.19.

For the online case we can make use of $\beta = 1$ and $C_\beta = 26.5$ for the exploration of $P(d)$, in the offline case we have $\beta = 1$ and $C_\beta = 1$. Application of Theorem 3.24 gives the following result:

Corollary 4.15 *The optimal search path and the optimal search ratio for general simple polygons can be approximated offline within a ratio of 8 and online within a ratio of 212.*

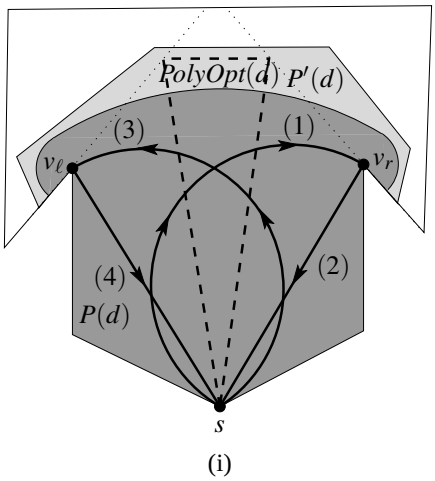


Figure 4.19: In this case SWR(d) leave the part $P(d)$. PolyExplore keeps inside $P(d)$.

4.4 Polygons with holes

In the previous section competitive strategies for the exploration of simple ploygons were presented. We would like to show that in a scene with polygonal obstacles such results cannot be obtained. We consider non-simple polygons which means that the polygon has holes (or obstacles) inside. These holes are non interescting and they are given as simple polygons itself.

The task of exploring a polygon with holes is much more complicated. At the first place the computation of the SWR is NP-hard. There is a simple reduction of the TSP problem by placing small obstacles around the corresponding point set. Furthermore, for simple polygons it can be shown that it suffices to explore the boundary. More precisely, if the boundary of a simple polygon P was *seen* along an exploration path, also any point inside P has been seen by the path. This is not true for polygons with holes as depicted in Figure 4.20. The path π *sees* the boundary of all obstacles and the outer boundary, but there is still a portion of the polygons that is not explored.

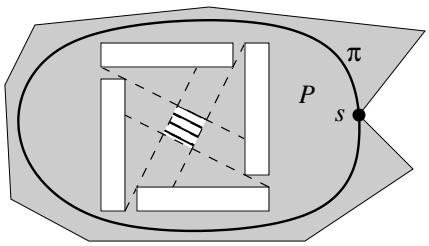


Figure 4.20: A polygon with holes. The path detects the full boundary but not all points inside P have been *seen*.

We can show that there is no strategy that explores any polygon with holes within a constant competitive ratio against the shortest exploration path.

Theorem 4.16 *(Albers, Kursawe, Schuierer, 1999)*
Let A be an arbitrary online strategy for an agent with a vision system for the exploration of a polygon P with holes. Let n denote the overall number of vertices of P. we have *[AKS02]*

$$|\pi_A| \geq \Omega(\sqrt{n}) \cdot |\pi_{opt}|.$$

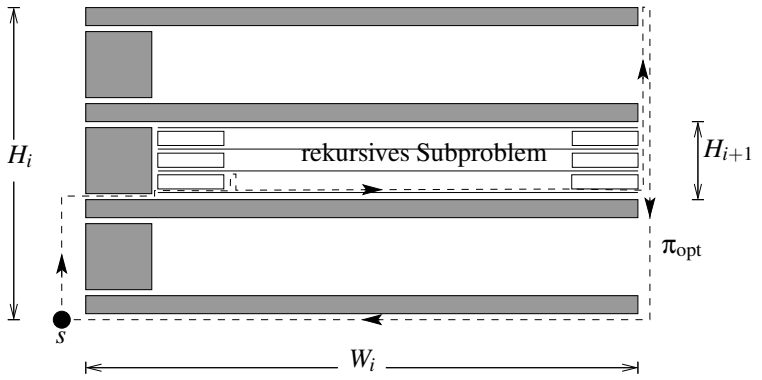


Figure 4.21: The lower bound construction for the exporation of a polygon with holes and a sketch of the optimal offline path π_{opt}.

Proof. We recursively construct a polygonal scene as shown in Figure 4.21. The starting scene consists of $k+1$ thin rectangles of length $W = 2k$ and arbitrarily small height, called spikes, and k rectangles of width 1 and height 1, the so-calles bases. The construction has height roughly $H_1 = k$. The agent starts at the lower left corner. Between a spike and a base there is an arbitrary thin corridor, so that the agent can move inside and have a look behind the base. Behind one of the bases the situation appears recursively, again with k spikes of length $W_i = 2k - i$ and k bases of width 1. The overall height is $H_i := \frac{1}{(2k)^{i-1}}$. The agent does not know whether the next sub-problem has the bases on the left or on the right side.

The construction will be repeated k times with values $H_{i+1} = \frac{H_i}{2k}$ and $W_{i+1} = W_i - 1$ for $i = 2, \ldots, k-1$, starting with $H_1 = k-1$ und $W_1 = 2k$. This means that we have k sub-problems, each nested behind the base of a previous one (up to the starting problem). Altogether, we have $k \times (2k+1)$ rectangles and $4k \times (2k+1) = n$ edges, with $k \in \Omega(\sqrt{n})$.

The strategy A has to see all points. In the first stage for finding the second block, the agent can either look behind the k bases from the left by moving distance 2k-1 or moves to the right (distance 2k) and then upwards. For both cases the next block will be presented at the last visit. In the first case the next base rectangles are located to the left, in the latter case the next base rectangles are located to the right. So the same situation occurs again. This means that the agent has to move at least k times distance k which gives $\Omega(k^2)$ in total. This means $|\pi_A| \in \Omega(k^2)$.

The optimal offline strategy directly moves to the base where the next recursive sub-problem is nested. Then the sub-problem is explored optimally with path length $2H_i$. Finally, the agent has to move to the right upper corner and moves back along the left side to look behind all bases; see Figure 4.21.

We have

$$\begin{aligned}
|\pi_{opt}| &= 2W_1 + 2\sum_{i=1}^{k} H_i \\
&= 2W_1 + 2H_1 \sum_{i=1}^{k} \frac{1}{(2k)^{i-1}} \\
&= 4k + 2k\left(\frac{\left(\frac{1}{2k}\right)^k - 1}{\left(\frac{1}{2k}\right) - 1}\right) = 4k + 2k\left(\frac{2k\left(1 - \left(\frac{1}{2k}\right)^k\right)}{2k-1}\right) \\
&\leq 8k\,.
\end{aligned}$$

This gives a ratio of $\Omega(k) = \Omega(\sqrt{n})$ which gives the bound $\Omega(\sqrt{n})$. □

Finally, by a simple trick we show that also the optimal search path cannot be approximated within a constant ration. The optimal search path for the above situation might decide to detect a point that has distance 1 from the start after $\Omega(k)$ steps, therefore the search ratio might be k.

To avoid this situation we shift the start k steps away from the block construction as shown in Figure 4.22. Now any non-visible point has distance at least k. An optimal exploration path has length at

most 10k and gives a constant approximation of the search ratio (which is a constant). As shown above any online strategy will detect the last point distance at most 4k away after at least $\Omega(k^2)$ steps. Thus the search ratio is in $\Omega(k)$.

Corollary 4.17 *For polygons with holes there is no strategy that approximates the optimal search path and the search ratio within by a constant factor.*

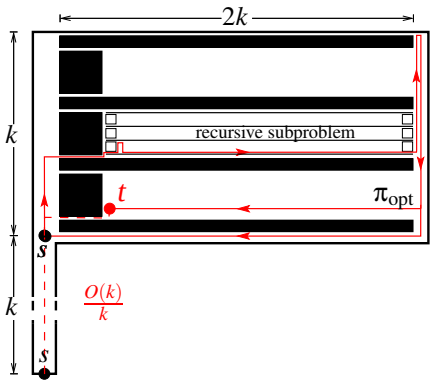


Figure 4.22: Shifting the start point away means that any invisible point has distance $\Theta(k)$, this gives a constant search ratio for the best offline exploration path.

Bibliography

[Ad80] H. Abelson and A. A. diSessa. *Turtle Geometry*. MIT Press, Cambridge, 1980.

[AFM00] E. M. Arkin, S. P. Fekete, and J. S. B. Mitchell. Approximation algorithms for lawn mowing and milling. *Comput. Geom. Theory Appl.*, 17:25–50, 2000.

[AG03] Steve Alpern and Shmuel Gal. *The Theory of Search Games and Rendezvous*. Kluwer Academic Publications, 2003.

[AKS02] Susanne Albers, Klaus Kursawe, and Sven Schuierer. Exploring unknown environments with obstacles. *Algorithmica*, 32:123–143, 2002.

[BRS94] Margrit Betke, Ronald L. Rivest, and Mona Singh. Piecemeal learning of an unknown environment. Technical Report A.I. Memo No. 1474, Massachusetts Institute of Technology, March 1994.

[BSMM00] Ilja N. Bronstein, Konstantin A. Semendjajew, Gerhard Musiol, and Heiner Mühlig. *Taschenbuch der Mathematik*. Verlag Harry Deutsch, Frankfurt am Main, 5th edition, 2000.

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

[CN86] W. Chin and S. Ntafos. Optimum watchman routes. In *Proc. 2nd Annu. ACM Sympos. Comput. Geom.*, pages 24–33, 1986.

[CN88] W. Chin and S. Ntafos. Optimum watchman routes. *Inform. Process. Lett.*, 28:39–44, 1988.

[DELM03] Moshe Dror, Alon Efrat, Anna Lubiw, and Joseph S. B. Mitchell. Touring a sequence of polygons. In *Proc. 35th Annu. ACM Sympos. Theory Comput.*, pages 473–482, 2003.

[DHN97] G. Das, P. Heffernan, and G. Narasimhan. LR-visibility in polygons. *Comput. Geom. Theory Appl.*, 7:37–57, 1997.

[DJMW91] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robotic exploration as graph construction. *Transactions on Robotics and Automation*, 7:859–865, 1991.

[DKK01] Christian A. Duncan, Stephen G. Kobourov, and V. S. Anil Kumar. Optimal constrained graph exploration. In *Proc. 12th ACM-SIAM Symp. Discr. Algo.*, pages 307–314, 2001.

[DKK06] Christian A. Duncan, Stephen G. Kobourov, and V. S. Anil Kumar. Optimal constrained graph exploration. *ACM Trans. Algor.*, 2:380–402, 2006.

[DKP91] X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 298–303, 1991.

[DKP98] Xiaotie Deng, Tiko Kameda, and Christos Papadimitriou. How to learn an unknown environment I: The rectilinear case. *J. ACM*, 45(2):215–245, 1998.

[EFK+06] Andrea Eubeler, Rudolf Fleischer, Tom Kamphans, Rolf Klein, Elmar Langetepe, and Gerhard Trippen. Competitive online searching for a ray in the plane. In Sándor Fekete, Rudolf Fleischer, Rolf Klein, and Alejandro López-Ortiz, editors, *Robot Navigation*, number 06421 in Dagstuhl Seminar Proceedings, 2006.

[FKK+04] Rudolf Fleischer, Tom Kamphans, Rolf Klein, Elmar Langetepe, and Gerhard Trippen. Competitive online approximation of the optimal search ratio. In *Proc. 12th Annu. European Sympos. Algorithms*, volume 3221 of *Lecture Notes Comput. Sci.*, pages 335–346. Springer-Verlag, 2004.

[Gal80] Shmuel Gal. *Search Games*, volume 149 of *Mathematics in Science and Engeneering*. Academic Press, New York, 1980.

[GKP98] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics*. Addison-Wesley, 1998.

[GR03] Yoav Gabriely and Elon Rimon. Competitive on-line coverage of grid environments by a mobile robot. *Comput. Geom. Theory Appl.*, 24:197–224, 2003.

[HIKK97] Frank Hoffmann, Christian Icking, Rolf Klein, and Klaus Kriegel. A competitive strategy for learning a polygon. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pages 166–174, 1997.

[HIKK01] Frank Hoffmann, Christian Icking, Rolf Klein, and Klaus Kriegel. The polygon exploration problem. *SIAM J. Comput.*, 31:577–600, 2001.

[HIKL99] Christoph Hipke, Christian Icking, Rolf Klein, and Elmar Langetepe. How to find a point on a line within a fixed distance. *Discrete Appl. Math.*, 93:67–73, 1999.

[IKKL00a] Christian Icking, Thomas Kamphans, Rolf Klein, and Elmar Langetepe. Exploring an unknown cellular environment. In *Abstracts 16th European Workshop Comput. Geom.*, pages 140–143. Ben-Gurion University of the Negev, 2000.

[IKKL00b] Christian Icking, Thomas Kamphans, Rolf Klein, and Elmar Langetepe. Exploring an unknown cellular environment. Unpublished Manuscript, FernUniversität Hagen, 2000.

[IKKL05] Christian Icking, Tom Kamphans, Rolf Klein, and Elmar Langetepe. Exploring simple grid polygons. In *11th Internat. Comput. Combin. Conf.*, volume 3595 of *Lecture Notes Comput. Sci.*, pages 524–533. Springer, 2005.

[IKL97] Christian Icking, Rolf Klein, and Elmar Langetepe. Searching for the kernel of a polygon: A competitive strategy using self-approaching curves. Technical Report 211, Department of Computer Science, FernUniversität Hagen, Germany, 1997.

[IKL99] Christian Icking, Rolf Klein, and Elmar Langetepe. An optimal competitive strategy for walking in streets. In *Proc. 16th Sympos. Theoret. Aspects Comput. Sci.*, volume 1563 of *Lecture Notes Comput. Sci.*, pages 110–120. Springer-Verlag, 1999.

[IKL+04] Christian Icking, Rolf Klein, Elmar Langetepe, Sven Schuierer, and Ines Semrau. An optimal competitive strategy for walking in streets. *SIAM J. Comput.*, 33:462–486, 2004.

[IKM94] Christian Icking, Rolf Klein, and Lihong Ma. An optimal competitive strategy for looking around a corner. Technical Report 167, Department of Computer Science, FernUniversität Hagen, Germany, 1994.

[IPS82] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM J. Comput.*, 11:676–686, 1982.

[KL03] Tom Kamphans and Elmar Langetepe. The Pledge algorithm reconsidered under errors in sensors and motion. In *Proc. of the 1th Workshop on Approximation and Online Algorithms*, volume 2909 of *Lecture Notes Comput. Sci.*, pages 165–178. Springer, 2003.

[Kle91] Rolf Klein. Walking an unknown street with bounded detour. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 304–313, 1991.

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

[KPY96] Elias Koutsoupias, Christos H. Papadimitriou, and Mihalis Yannakakis. Searching a fixed graph. In *Proc. 23th Internat. Colloq. Automata Lang. Program.*, volume 1099 of *Lecture Notes Comput. Sci.*, pages 280–289. Springer, 1996.

[Lan00] Elmar Langetepe. *Design and Analysis of Strategies for Autonomous Systems in Motion Planning*. PhD thesis, Department of Computer Science, FernUniversität Hagen, 2000.

[LB99] Sharon Laubach and Joel Burdick. RoverBug: Long range navigation for mars rovers. In Peter Corke and James Trevelyan, editors, *Proc. 6th Int. Symp. Experimental Robotics*, volume 250 of *Lecture Notes in Control and Information Sciences*, pages 339–348. Springer, 1999.

[Lee61] C. Y. Lee. An algorithm for path connections and its application. *IRE Trans. on Electronic Computers*, EC-10:346–365, 1961.

[LOS96] Alejandro López-Ortiz and Sven Schuierer. Walking streets faster. In *Proc. 5th Scand. Workshop Algorithm Theory*, volume 1097 of *Lecture Notes Comput. Sci.*, pages 345–356. Springer-Verlag, 1996.

[LS87] V. J. Lumelsky and A. A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[Sch01] S. Schuierer. Lower bounds in on-line geometric searching. *Comput. Geom. Theory Appl.*, 18:37–53, 2001.

[Sha52] Claude E. Shannon. Presentation of a maze solving machine. In H. von Foerster, M. Mead, and H. L. Teuber, editors, *Cybernetics: Circular, Causal and Feedback Mechanisms in Biological and Social Systems, Transactions Eighth Conference, 1951*, pages 169–181, New York, 1952. Josiah Macy Jr. Foundation. Reprint in [Sha93].

[Sha93] Claude E. Shannon. Presentation of a maze solving machine. In Neil J. A. Sloane and Aaron D. Wyner, editors, *Claude Shannon: Collected Papers*, volume PC-03319. IEEE Press, 1993.

[SM92] A. Sankaranarayanan and I. Masuda. A new algorithm for robot curvefollowing amidst unknown obstacles, and a generalization of maze-searching. In *Proc. 1992 IEEE Internat. Conf. on Robotics and Automation*, pages 2487–2494, 1992.

[SS99] Sven Schuierer and Ines Semrau. An optimal strategy for searching in unknown streets. In *Proc. 16th Sympos. Theoret. Aspects Comput. Sci.*, volume 1563 of *Lecture Notes Comput. Sci.*, pages 121–131. Springer-Verlag, 1999.

[Sut69] Ivan E. Sutherland. A method for solving arbitrary wall mazes by computer. *IEEE Trans. on Computers*, 18(12):1092–1097, 1969.

[SV90a] A. Sankaranarayanan and M. Vidyasagar. A new path planning algorithm for a point object amidst unknown obstacles in a plane. In *Proc. 1990 IEEE Internat. Conf. on Robotics and Automation*, pages 1930–1936, 1990.

[SV90b] A. Sankaranarayanan and M. Vidyasagar. Path planning for moving a point object amidst unknown obstacles in a plane: A new algorithm and a general theory for algorithm developments. In *Proceedings of 1990 IEEE Conf. on Decision and Control*, pages 1111–1119, 1990.

[SV91] A. Sankaranarayanan and M. Vidyasagar. Path planning for moving a point object amidst unknown obstacles in a plane: The universal lower bound on the worst case path lengths and a classification of algorithms. In *Proc. 1991 IEEE Internat. Conf. on Robotics and Automation*, pages 1734–1741, 1991.

[THL98] L. H. Tseng, P. Heffernan, and D. T. Lee. Two-guard walkability of simple polygons. *Internat. J. Comput. Geom. Appl.*, 8(1):85–116, 1998.

[Wal86] Wolfgang Walter. *Gewöhnliche Differentialgleichungen*. Springer, 1986.

[Web07] Maximilian Weber. Online suche auf beschränkten sternen. Diplomarbeit, Rheinische Friedrich-Wilhelms-Universität Bonn, 2007.

Index

T

U

V

W

Y