Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik I

Elmar Langetepe

Online Motion Planning

MA INF 1314

Summersemester 2016

Manuscript: Elmar Langetepe

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 (Algorithmus 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.

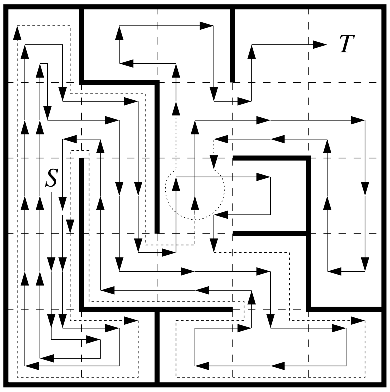


Figure 1.2: An example of the execution of Shannons Algorithm.

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

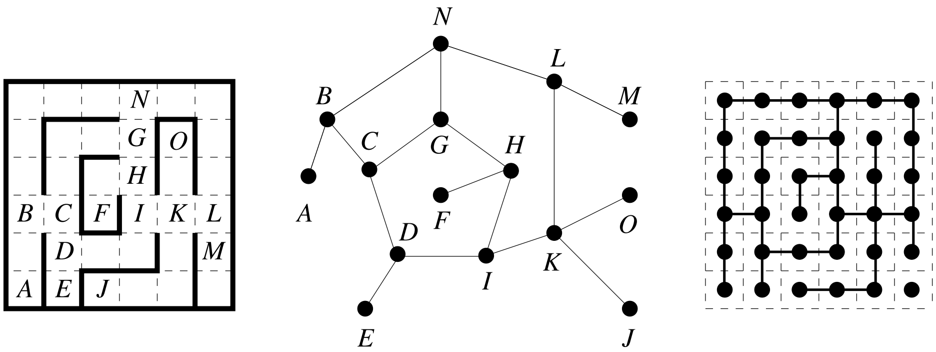
For a human a labyrinth consists of corridors and connection points. In this sense the environment for Shannons task can be considered to be a labyrinth. Obviously any such labyrinth can be modeled by a planar graph.[1] More precisely the environment for Shannons task is a grid graph. Figure 1.3 shows 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: $|S_{ROB}| \geq 2|S_{OPT}|-6$.

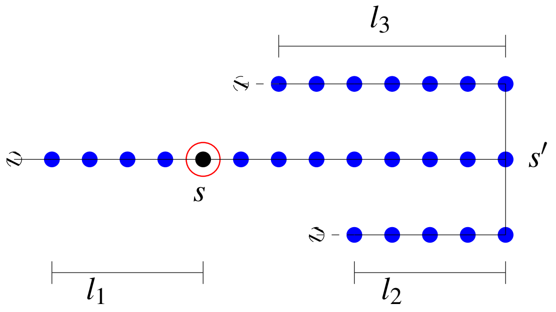


Figure 1.4: The agent return to s.

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 $|S_{ROB}| \geq 2|S_{OPT}|-4 > 2|S_{OPT}|-6$.

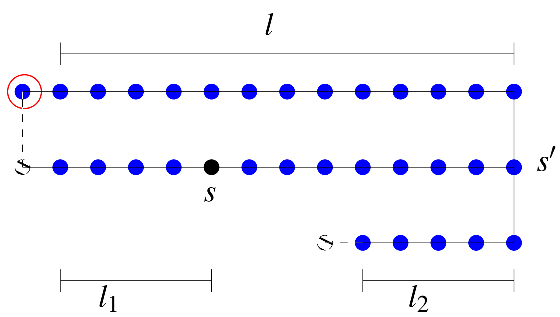


Figure 1.5: The agent has visited $\ell+1$ vertices in corridor 3.

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.

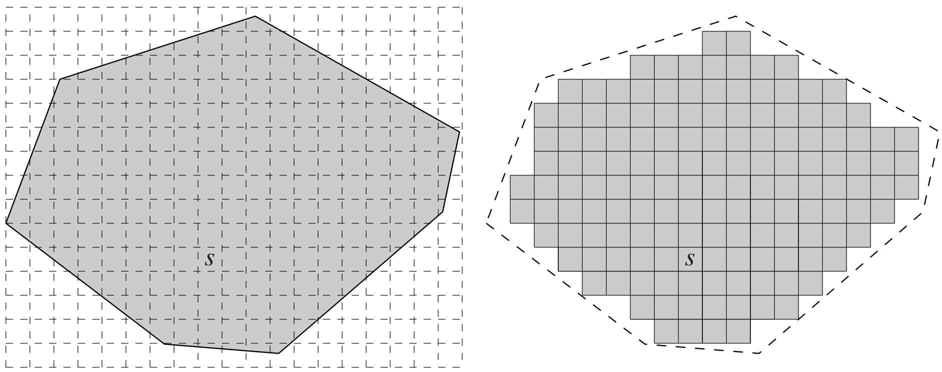


Figure 1.6: A polygon P and the gridpolygon $P_\square$ as a reasonable approximation.

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

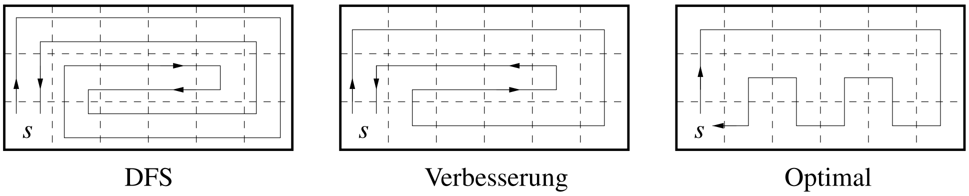


Figure 1.7: Ist DFS optimal?

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 $\frac{1}{2}E << C$; see also Figure 1.8.

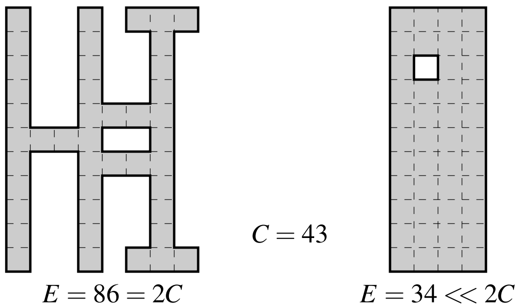


Figure 1.8: The number of boundary edges E in comparison to the number of cells C is a measure for the existence of *fleshy* or *skinny* parts.

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 $\frac{8}{6} \approx 1.3$.

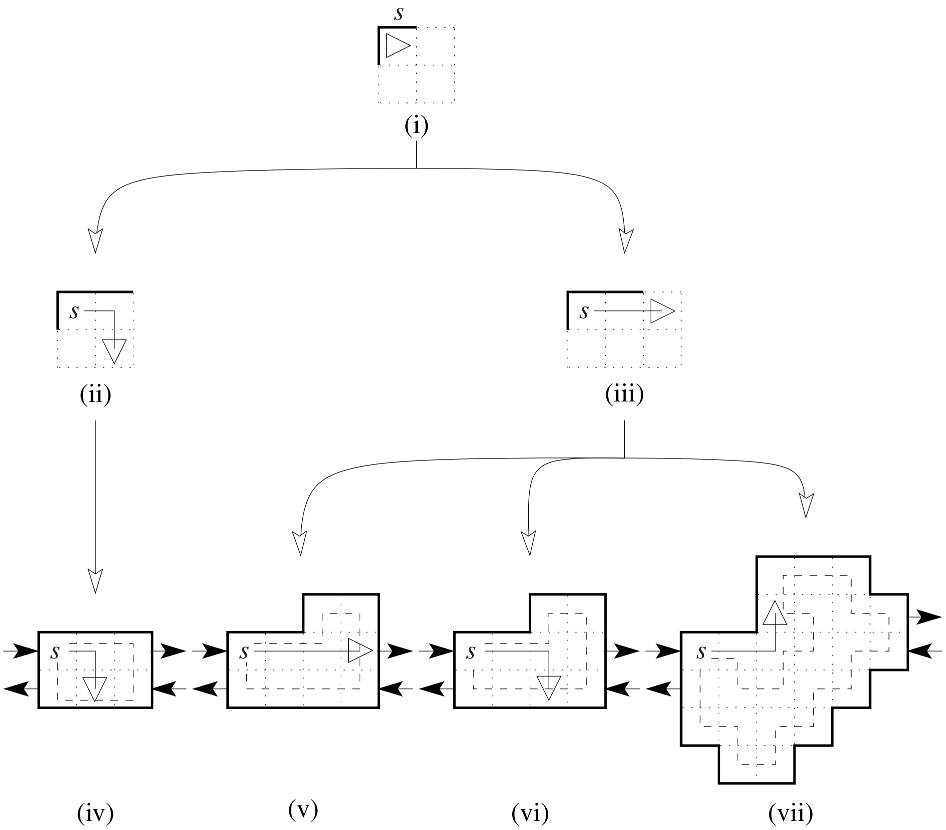


Figure 1.9: A lower bound construction for the exploration of simple gridpolygons.

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 Algorithmus 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

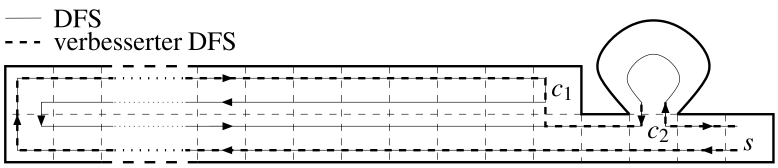


Figure 1.10: First simple improvement of DFS.

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

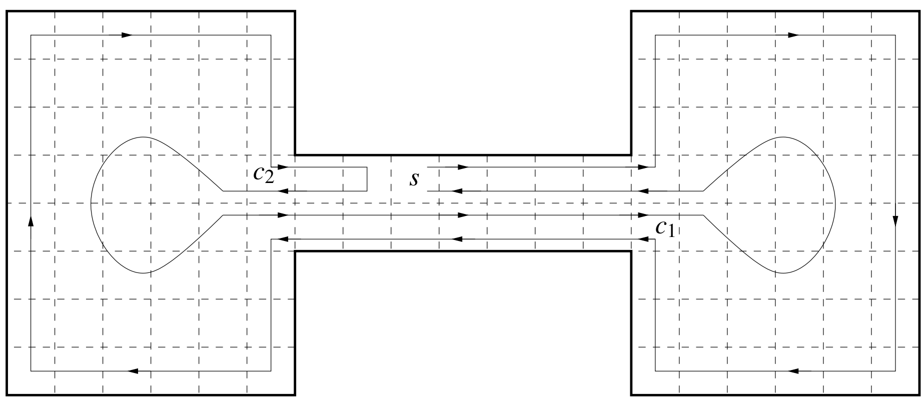


Figure 1.11: Second improvement of DFS.

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 Algorithmus 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.

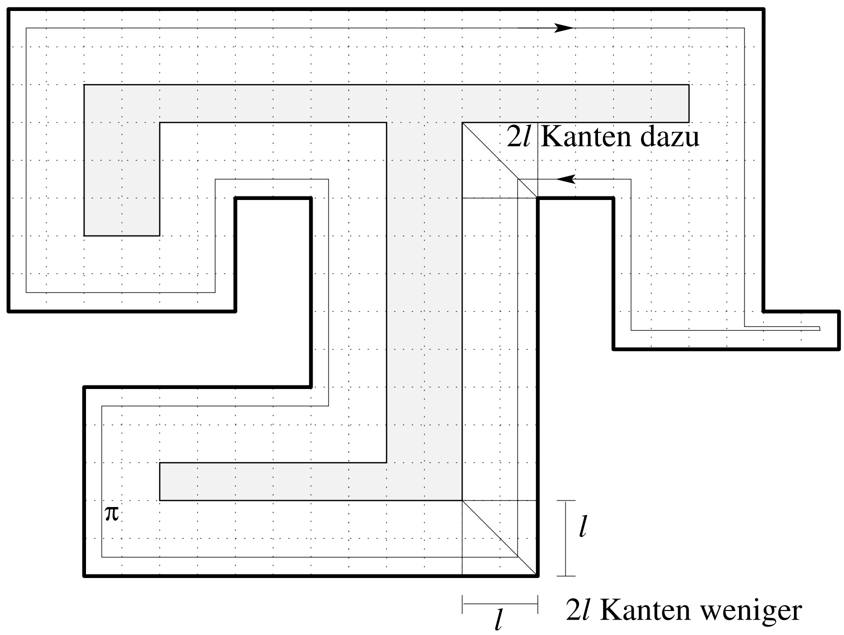


Figure 1.12: The ℓ-Offset of gridpolygon P.

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.*

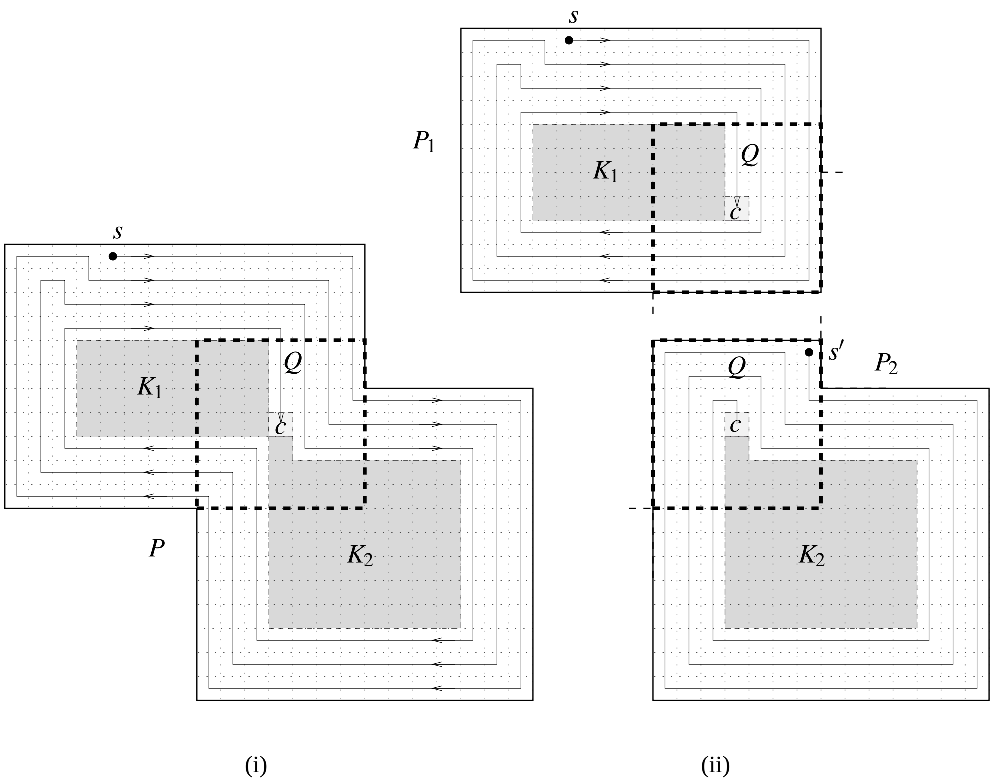


Figure 1.13: Decomposition at a split-cell.

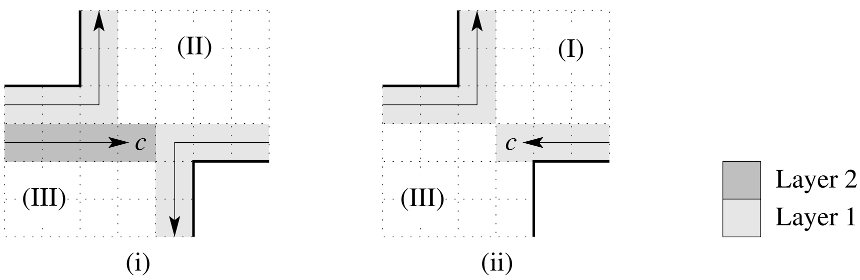


Figure 1.14: Three types of components.

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.

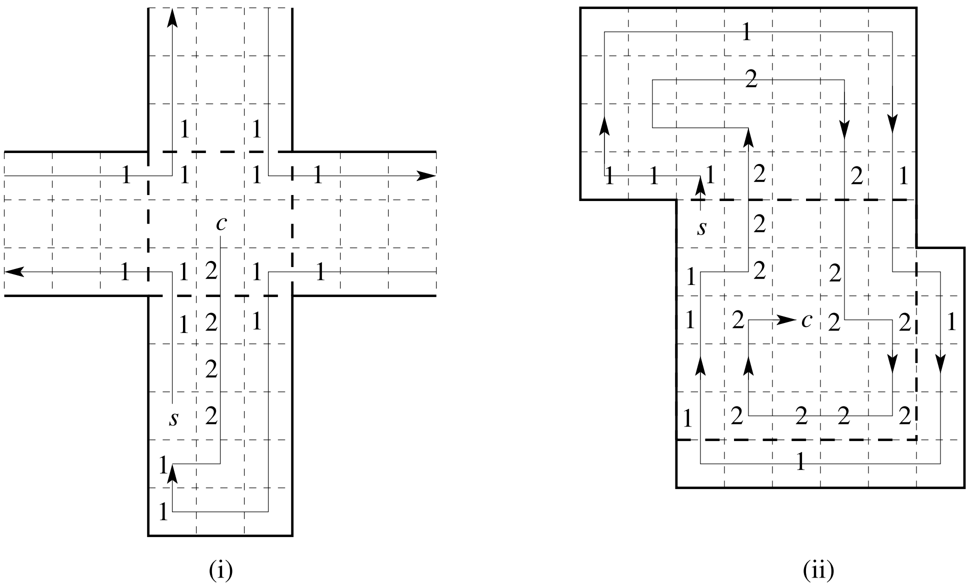


Figure 1.15: Special cases: No component of typ (III) exists.

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 $excess(P)\le \frac{1}{2}E-3$. 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:

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

□

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 Algorithmus 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.

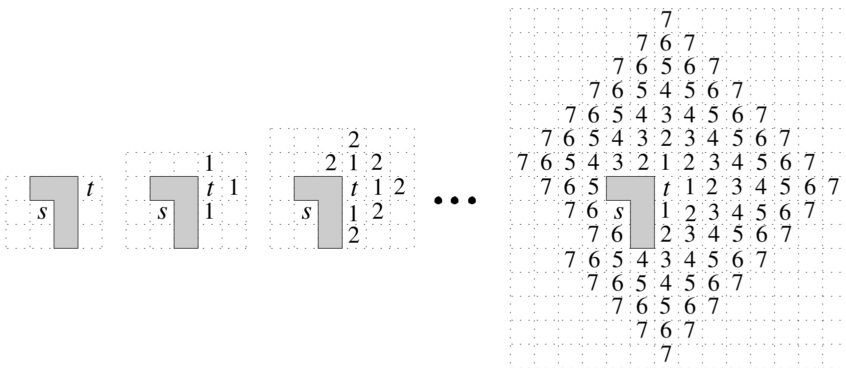


Figure 1.16: Wave-Propagation.

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.

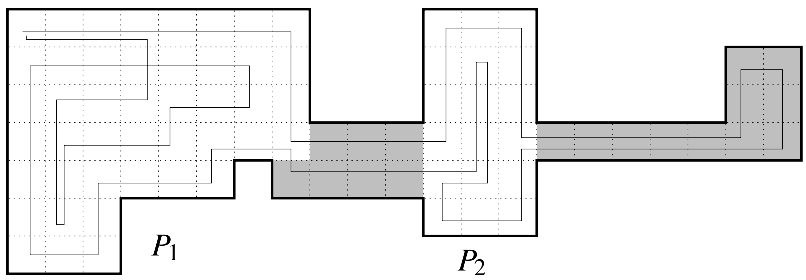


Figure 1.17: SmartDFS is optimal in narrow passages.

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.*

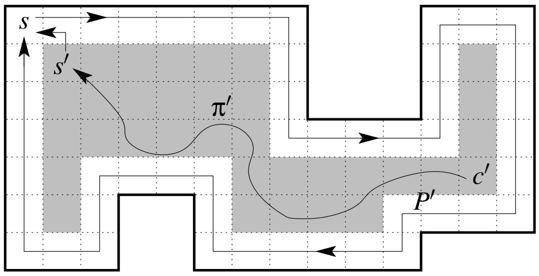


Figure 1.18: A simple gridpolygon without narrow passages and no split-cell in the first layer has the property $E(P) \leq \frac{2}{3}C(P)+6$. After the first coil SmartDFS starts in the 1-Offset P'. The return path to c' from an arbitrary point in P' is shorter than $\frac{1}{2}E(P)/2-2$.

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.

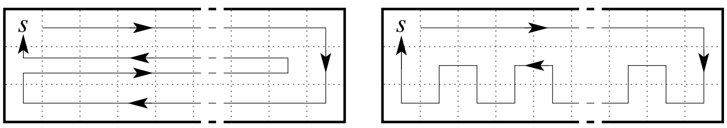
Theorem 1.17 *(Icking, Kamphans, Klein, Langetepe, 2005) The SmartDFS strategie for the exploration of simple gridpolygons is $\frac{4}{3}$-competitive!* *[?]*

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.



SmartDFS optimal strategy

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

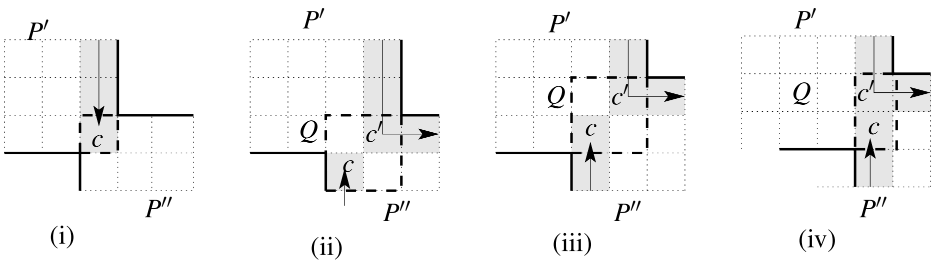


Figure 1.20: A gridpolygon P_i that is separated into components of type (I) or (II) at the split-cell. The rectangle Q is always inside P_i.

Bibliography

[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.

[IPS82] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM J. Comput.*, 11:676–686, 1982.

[Lee61] C. Y. Lee. An algorithm for path connections and its application. *IRE Trans. on Electronic Computers*, EC-10:346–365, 1961.

[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.

[Sut69] Ivan E. Sutherland. A method for solving arbitrary wall mazes by computer. *IEEE Trans. on Computers*, 18(12):1092–1097, 1969.

Index