Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik I

Christian Icking[1] **Tom Kamphans**[2]
Rolf Klein[2] **Elmar Langetepe**[2]

Exploring Grid Polygons Online

Technical Report 001
December 2005

[1]FernUniversität Hagen, Praktische Informatik VI, Feithstr. 142, D-58084 Hagen.
[2]Universität Bonn, Institut für Informatik, Abt. I, Römerstr. 164, D-53117 Bonn.

Abstract

We investigate the exploration problem of a short-sighted mobile robot moving in an unknown cellular room. To explore a cell, the robot must enter it. Once inside, the robot knows which of the 4 adjacent cells exist and which are boundary edges. The robot starts from a specified cell adjacent to the room's outer wall; it visits each cell, and returns to the start. Our interest is in a short exploration tour; that is, in keeping the number of multiple cell visits small. For abitrary environments containing no obstacles we provide a strategy producing tours of length

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

and for environments containing obstacles we provide a strategy, that is bound by

$$S \leq C + \frac{1}{2}E + 3H + W_{\mathrm{cw}} - 2,$$

where C denotes the number of cells—the area—, E denotes the number of boundary edges—the perimeter—, and H is the number of obstacles, and W_{cw} is a measure for the sinuosity of the given environment.

The strategies were implemented in a Java applet [13] that can be found in

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

Key words: Robot navigation, Online algorithms, competitive analysis, unknown environment, obstacles.

1 Introduction

Exploring an unknown environment and searching for a target in unknown position are among the basic tasks of autonomous mobile robots. Both problems have received a lot of attention in computational geometry and in robotics; see, for example, [2, 5, 7, 10, 14, 16, 17, 18, 21, 22, 26].

In general, one assumes that the robot is equipped with an ideal vision system that provides, in a continuous way, the full visibility polygon of the robot's current position. In practice, there are at least two reasons that limit the reach of this model.

- A realistic laser scanner can reliably detect objects only within a distance of up to a few meters. Hence, in a large environment the robot has to move towards more distant areas to explore them.
- Service robots like lawn mowers or cleaning devices need to get close to the parts of the environment they want to inspect and, possibly, to work on.

To accomodate these situations, we study in this paper the model of a rather short-sighted robot. We assume that the environment is given by a simple polygon, P. Its edges are either vertical or horizontal, and its vertices have integer coordinates. Thus, the interior of P consists of integer grid cells. An interior cell is either free or blocked.[1]

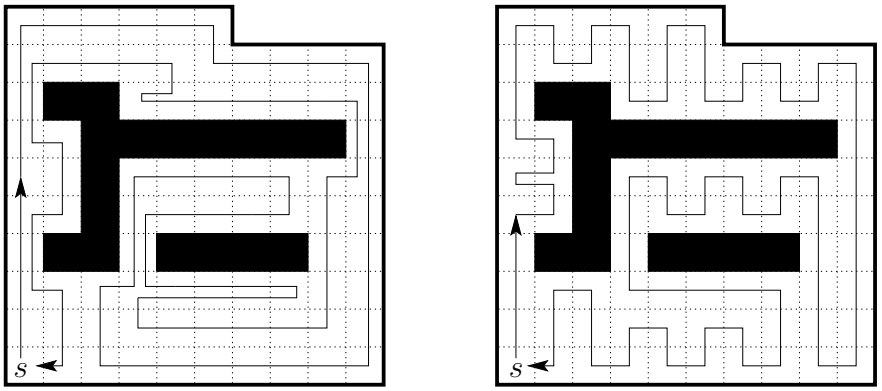
The robot starts from a free cell, s, adjacent to the polygon's boundary. From there it can enter one of the neighboring free cells, and so on. Once inside a cell, the robot knows which of its 4 neighbors are blocked and which are free. The robot's task is to visit each free cell inside P and to return to the start; see Figure 1 (i) for an example. This example shows a tour that visits each cell once and some cells even more often. Our interest is in a short exploration tour, so we would like to keep the number of excess cell visits small.

Even though our robot does not know its environment in advance, it is interesting to ask how short a tour can be found in the offline situation (i.e., when the environment is already known). This amounts to constructing a shortest traveling salesperson tour on the free cells.

If the polygonal environment contains obstacles, the problem of finding such a minimum length tour is known to be NP-hard, by work of Itai et al. [19]. There are $1+\varepsilon$ approximation schemes by Grigni et al. [12], Arora [4], and Mitchell [20], and a $\frac{53}{40}$ approximation by Arkin et al. [3].

In a simple polygon without obstacles, the complexity of constructing offline a minimum length tour seems to be open. Ntafos [24] and Arkin

[1] For convenience, we consider the exterior of the polygon to consist of blocked cells only.



(i) (ii)

Figure 1: (i) An example exploration tour. (ii) A shortest TSP tour for the same polygon.

et al. [3] have shown how to approximate the minimum length tour with factors of $\frac{4}{3}$ and $\frac{6}{5}$, respectively. Umans and Lenhart [27] have provided an $O(C^4)$ algorithm for deciding if there exists a Hamiltonian cycle (i.e., a tour that visits each of the C cells of a polygon *exactly* once). For the related problem of Hamiltonian paths (i.e., different start and end positions), Everett [9] has given a polynomial algorithm for certain grid graphs. Cho and Zelikovsky [15] studied *spanning closed trails* (a relaxation of Hamiltonian cyles) in grid graphs.[2]

In this paper, our interest is in the online version of the cell exploration problem. The task of exploring an grid polygon with holes was independently considered by Gabriely and Rimon [11]. They introduce a somehow artificial robot model by distinguishing between the robot and its tool. The tool has the size of one grid cell and moves from cell to cell. The robot moves between the midpoints of 2×2–blocks of cells. The robot constructs a spanning tree on the 2×2 cell-blocks and the tools explores the polygon keeping the spanning tree edges on its right side. Only if the tool's path is blocked by an obstacle, it switches to the other side of the spanning tree and keeps the spanning tree edges on it's left side. This model allows a smart analysis yielding an upper bound of $C + B$, where C denotes the number of cells and B the number of boundary cells. However, this bound is larger than $C + \frac{1}{2}E + 3H + W_{\text{cw}} - 2$ except for corridors of width one, in which both bounds are the same, this may justify our more complicated analysis of the strategy.

Another online task is the *piecemeal exploration*, where the robot has to interrupt the exploration every now and then so as to return to the start

[2]The grid graph corresponding to a grid polygon, P, consists of one node for every free cell in P. Two nodes are connected by an edge, if their corresponding cells are adjacent.

point, for example, to refuel. Piecemeal exploration of grid graphs was studied by Betke et al. [6] and Albers et al. [1]. Note that their objective is to visit every node *and* every edge, whereas we require a complete coverage of only the cells. Subdividing the robot's environment into grid cells is used also in the robotics community, see, for example, Moravec and Elfes [23], and Elfes [8].

Our paper is organized as follows: In Section 2 we give more detailled description of our robot and the environment. We analyze the competitive complexity in Section 3, showing lower bounds for both simple polygons and polygons with holes. In Section 4 we present an exploration strategy for simple polygons together with the analysis of the strategy and in Section 5 we present and analyze a strategy for polygons with holes.

2 Definitions

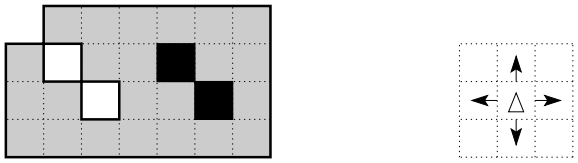


Figure 2: (i) Polygon with 23 cells, 38 edges and one(!) hole (black cells), (ii) the robot can determine which of the 4 adjacent cells are free, and enter an adjacent free cell.

Definition 1 A *cell* is a basic block in our environment, defined by a tuple $(x, y) \in \mathbb{N}^2$. A cell is either *free* and can be visited by the robot, or *blocked* (i.e., unaccessible for the robot).[3] We call two cells $c_1 = (x_1, y_1), c_2 = (x_2, y_2)$ *adjacent* or *neighboring*, if they share a common edge (i.e., if $|x_1 - x_2| + |y_1 - y_2| = 1$ holds), and *touching*, if they share a common edge or corner.

A *path*, Π, from a cell s to a cell t is a sequence of free cells $s = c_1, \ldots, c_n = t$ where c_i and c_{i+1} are adjacent for $i = 1, \ldots, n-1$. Let $|\Pi|$ denote the length of Π. We assume that the cells have unit size, so the length of the path is equal to the number of *steps* from cell to cell that the robot walks.

A *grid polygon*, P, is a connected set of free cells; that is, for every $c_1, c_2 \in P$ exists a path from c_1 to c_2 that lies completely in P.

We call a set of touching blocked cells that are completely surrounded by free cells an *obstacle* or *hole*, see Figure 2. Polygons without holes are called *simple polygons*.

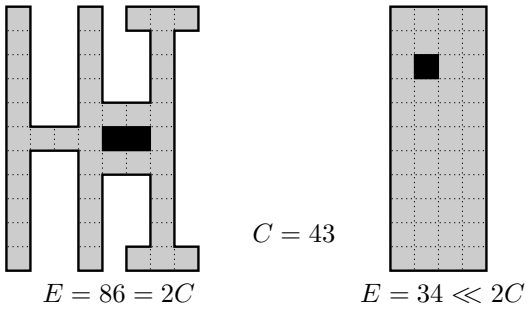


Figure 3: The perimeter, E, is used to distinguish between *thin* and *thick* environments.

We analyze the performance of an exploration strategy using some parameters of the grid polygon. In addition to the area, C, of a polygon we introduce the *perimeter*, E. C is the number of free cells and E is the total number of edges that appear between a free cell and a blocked cell, see, for example, Figure 2 or Figure 3. We use E to distinguish between thin and thick environments, see Section 3. In Section 5.2 we introduce another parameter, the sinuosity W_{cw}, to distinguish between straight and twisted polygons.

3 Competitive Complexity

We are interested in an online exploration. In this setting, the environment is not known to the robot in advance. Thus, the first question is whether the robot is still able to approximate the optimum solution up to a constant factor in this setting. There is a quick and rather simple answer to this question:

Theorem 2 *The competitive complexity of exploring an unknown cellular environment with obstacles is equal to 2.*

Proof. Even if the environment is unknown we can apply a simple depth-first search algorithm (DFS) to the grid graph. This results in a complete exploration in $2C-2$ steps. The shortest tour needs at least C steps to visit all cells and to return to s, so DFS is competitive with a factor of 2.

On the other hand, 2 is also a lower bound for the competitive factor of any strategy. To prove this, we construct a special grid polygon depending on the behavior of the strategy. The start position, s, is located in a long corridor of width 1. We fix a large number, Q, and observe how the strategy explores this corridor. Two cases occur.

[3] In the following, we sometimes use the terms *free cells* and *cells* synonymously.

Case 1: The robot eventually returns to s after walking at least Q and at most $2Q$ steps. At this time, we close the corridor with two unvisited cells, one at each end, see Figure 4(i). Let R be the number of cells visited so far. The robot has already walked at least $2R-2$ steps and needs another $2R$ steps to visit the two remaining cells and to return to s, whereas the shortest tour needs only $2R$ steps to accomplish this task.

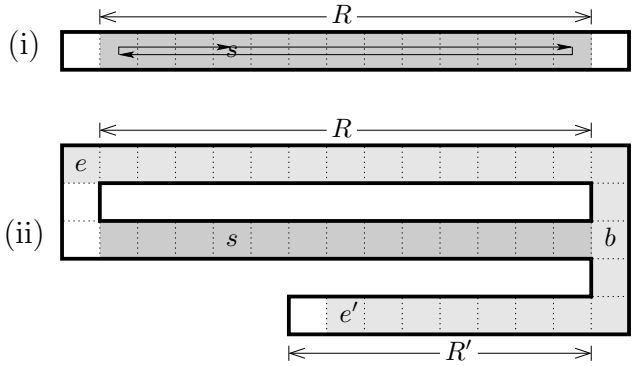


Figure 4: A lower bound of 2 for the exploration of grid polygons.

Case 2: In the remaining case the robot concentrates—more or less—on one end of the corridor. Let R be the number of cells visited after $2Q$ steps. Now, we add a bifurcation at a cell b immediately behind the farthest visited cell in the corridor, see Figure 4(ii). Two paths arise, which turn back and run parallel to the long corridor. If the robots returns to s before exploring one of the two paths an argument analogous to case 1 applies. Otherwise, one of the two paths will eventually be explored up to the cell e where it turns out that this corridor is connected to the other end of the first corridor. At this time, the other path is defined to be a dead end of length R', which closes just one cell behind the last visited cell e'.

From e the robot still has to walk to the other end of the corridor, to visit the dead end, and to return to s. Altogether, it will have walked at least four times the length of the corridor, R, plus four times the length of the dead end, R'. The optimal path needs only $2R + 2R'$, apart from a constant number of steps for the vertical segments.

In any case, the lower bound for the number of steps tends to 2 while Q goes to infinity. □

We cannot apply Theorem 2 to simple polygons, because we used a polygon with a hole to show the lower bound. The following lower bound holds for simple polygons.

Theorem 3 *Every strategy for the exploration of a simple grid polygon with C cells needs at least $\frac{7}{6}C$ steps.*

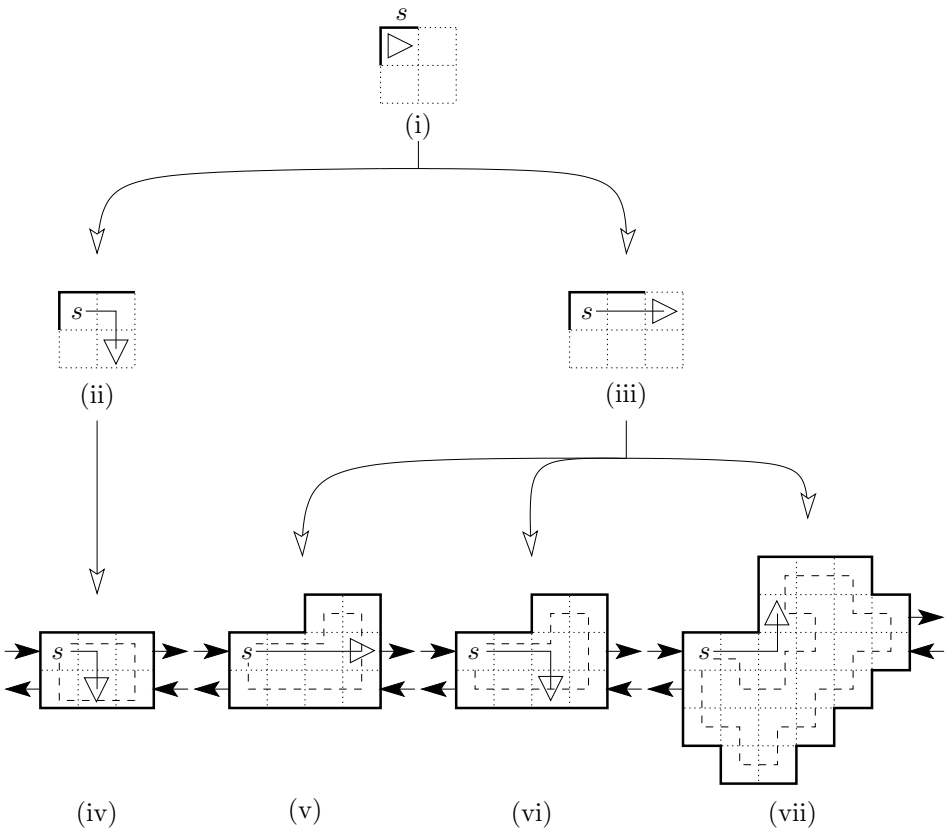
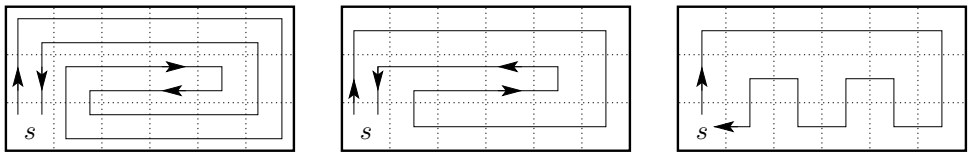


Figure 5: A lower bound for the exploration of simple polygons. The dashed lines show the optimal solution.

Proof. We assume that the robot starts in a corner of the polygon, see Figure 5(i) where $\triangle$ denotes the robot's position. Let us assume, the strategy decides to walk one step to the east—if the strategy walks to the south we use a mirrored construction. For the second step, the strategy has two possibilities: Either it leaves the wall with a step to the south, see Figure 5(ii), or it continues to follow the wall with a further step to the east, see Figure 5(iii). In the first case, we close the polygon as shown in Figure 5(iv). The robot needs at least 8 steps to explore this polygon, but the optimal strategy needs only 6 steps yielding a factor of $\frac{8}{6} \approx 1.3$. In the second case we proceed as follows. If the robot leaves the boundary, we close the polygon as shown in Figure 5(v) and (vi). The robot needs 12 step, but 10 steps are sufficient. In the most interesting case, the robot still follows the wall, see Figure 5(vii). In this case, the robot will need at least 28 steps to explore this polygon, whereas an optimal strategy needs only 24 steps. This leaves us with a factor of $\frac{28}{24} = \frac{7}{6} \approx 1.16$.

We can easily extend this pattern to build polygons of arbitrary size by repeating the preceding construction several times using the *entry* and *exit*

cells denoted by the arrows in Figure 5(iv)–(vii). As soon as the robot leaves one block, it enters the start cell of the next block and the game starts again; that is, we build the next block depending on the robot's behavior. Note that this construction cannot lead to overlapping polygons or polygons with holes, because the polygon always extends to the same direction. □



DFS Improvement Optimal

Figure 6: DFS is not the best possible strategy.

Even though we have seen in Theorem 2 that the simple DFS strategy already achieves the optimal competitive factor in polygons with holes, DFS is not the best possible exploration strategy! There is no reason to visit *each* cell twice just because this is required in some special situations like dead ends of width 1. Instead, a strategy should make use of wider areas, see Figure 6.

We use the perimeter, E, to distinguish between thin environments that have many corridors of width 1, and thick environments that have wider areas, see Figure 3 on page 4. In the following sections we present strategies that explore grid polygons using no more than roughly $C + \frac{1}{2}E$ steps. Since all cells in the environment have to be visited, C is a lower bound on the number of steps that are needed to explore the whole polygon and to return to s.[4] Thus, $\approx \frac{1}{2}E$ is an upper bound for the number of additional cell visits. For thick environments, the value of E is in $O(\sqrt{C})$, so that the number of additional cell visits is substantially smaller than the number of free cells. Only for polygons that do not contain any 2×2 square of free cells, E achieves its maximum value of $2(C + 1)$, and our upper bound is equal to $2C - 2$, which is the cost of applying DFS. But in this case one cannot do better, because even the optimal offline strategy needs that number of steps. In other cases, our strategies are more efficient than DFS.

[4]More precisely, we need at least $C - 1$ steps to visit every cell, and at least 1 step to return to s.

4 Exploring Simple Polygons

We have seen in the previous section that a simple DFS traversal achieves a competitive factor of 2. Because the lower bound for simple grid polygons is substantially smaller, there may be a strategy that yields a better factor. Indeed, we can improve the DFS strategy. In this section, we give a precise description of DFS and present two improvements that lead to a $\frac{4}{3}$-competitive exploration strategy for simple polygons.

4.1 An Exploration Strategy

There are four possible directions—north, south, east and west—for the robot to move from one cell to an adjacent cell. We use the command *move(*dir*)* to execute the actual motion of the robot. The function *unexplored(*dir*)* returns true, if the cell in the given direction seen from the robot's current position is not yet visited, and false otherwise. For a given direction *dir*, *cw(*dir*)* denotes the direction turned 90° clockwise, *ccw(*dir*)* the direction turned 90° counterclockwise, and *reverse(*dir*)* the direction turned by 180°.

Using these basic commands, the simple DFS strategy can be implemented as shown in Algorithm 4.1. For every cell that is entered in direction *dir*, the robot tries to visit the adjacent cells in clockwise order, see the procedure *ExploreCell*. If the adjacent cell is still unexplored, the robot enters this cell, recursively calls *ExploreCell*, and walks back, see the procedure *ExploreStep*. Altogether, the polygon is explored following the *left-hand rule*: The robot proceeds from one unexplored cell to the next while the polygon's boundary or the explored cells are always to its left hand side.

Obviously, all cells are visited, because the graph is connected, and the whole path consists of $2C-2$ steps, because each cell—except for the start—is entered exactly once by the first *move* statement, and left exactly once by the second *move* statement in the procedure *ExploreStep*.

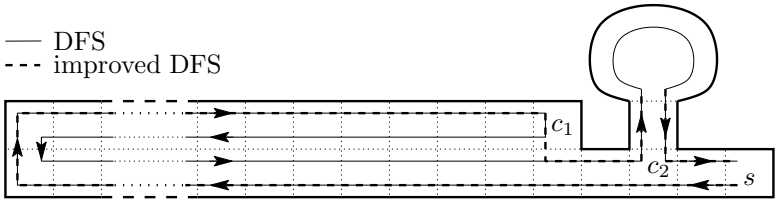


Figure 7: First improvement to DFS: Return directly to those cells that still have unexplored neighbors.

The first improvement to the simple DFS is to return directly to those cells that have unexplored neighbors. See, for example, Figure 7: After the robot has reached the cell c_1, DFS walks to c_2 through the completely

Algorithm 4.1 DFS

DFS(P**,** *start***):**

Choose direction *dir*, so that reverse(*dir*) points to a blocked 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

explored corridor of width 2. A more efficient return path walks on a shortest path from c_1 to c_2. Note that the robot can use for this shortest path only cells that are already known. With this modification, the robot's position might change between two calls of *ExploreStep*. Therefore, the procedure *ExploreCell* has to store the current position, and the robot has to walk on the shortest path to this cell, see the procedure *ExploreStep* in Algorithm 4.2. The function *unexplored(*cell, dir*)* returns true, if the cell in direction *dir* from *cell* is not yet visited.

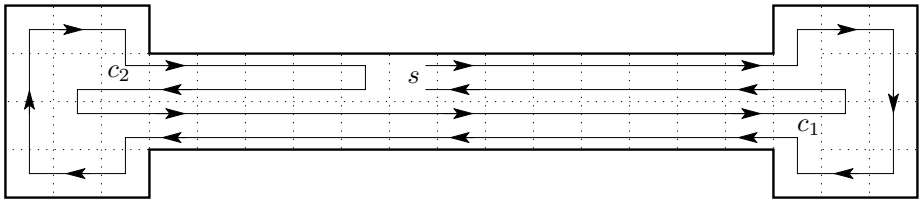


Figure 8: Second improvement to DFS: Detect polygon splits.

Now, observe the polygon shown in Figure 8. DFS completely surrounds the polygon, returns to c_2 and explores the left part of the polygon. After this, it walks to c_1 and explores the right part. Altogether, the robot walks four times through the narrow corridor. A more clever solution would explore the right part immediately after the first visit of c_1, and continue with the left part after this. This solution would walk only two times through the corridor in the middle! The cell c_1 has the property that the graph of unvis-

ited cells splits into two components after c_1 is explored. We call cells like this *split cells*. The second improvement to DFS is to recognize split cells and diverge from the left-hand rule when a split cell is detected. Essentially, we want to split the set of cells into several components, which are finished in the reversed order of their distances to the start cell. The detection and handling of split cells is specified in Section 4.2. Algorithm 4.2 resumes both improvements to DFS.

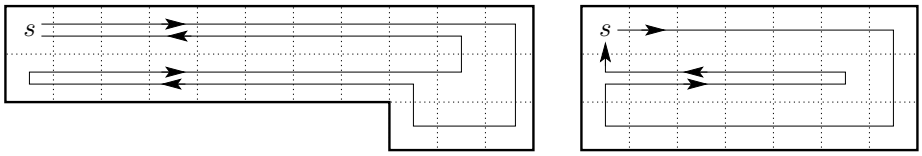


Figure 9: Straightforward strategies are not better than SmartDFS.

Note that the straightforward strategy *Visit all boundary cells and calculate the optimal offline path for the rest of the polygon* does not achieve a competitive factor better than 2. For example, in Figure 9(i) this strategy visits almost every boundary cell twice, whereas SmartDFS visits only one cell twice. Even if we extend the simple strategy to detect split cells while visiting the boundary cells, we can not achieve a factor better than $\frac{4}{3}$. A lower bound on the performace of this strategy is a corridor of width 3, see Figure 9(ii). Moreover, it is not known whether the offline strategy is NP-hard for simple polygons.

4.2 The Analysis of SmartDFS

SmartDFS explores the polygon in layers: Beginning with the cells along the boundary, SmartDFS proceeds towards the interior of P. Let us number the single layers:

Definition 4 Let P be a (simple) grid polygon. The boundary cells of P uniquely define the *first layer* of P. The polygon P without its first layer is called the *1-offset* of P. The ℓth layer and the ℓ-offset of P are defined successively, see Figure 10.

Note that the ℓ-offset of a polygon P is not necessarily connected. Although the preceding definition is independent from any strategy, SmartDFS can determine a cell's layer when the cell is visited for the first time. We can define the ℓ-offset in the same way for a polygon with holes, but the layer of a given cell can no longer be determined on the first visit in this case. The ℓ-offset has an important property:

Lemma 5 *The ℓ-offset of a simple grid polygon, P, has at least 8ℓ edges fewer than P.*

Algorithm 4.2 SmartDFS

SmartDFS(P**,** *start***):**

Choose direction *dir* for the robot, so that reverse(*dir*) points to a blocked cell;
ExploreCell(*dir*);
Walk on the shortest path to the start cell;

ExploreCell(dir**):**

Mark the current cell with the number of the current layer;
base := current position;
if not isSplitCell(*base*) **then**
  // Left-Hand Rule:
  ExploreStep(*base*, ccw(*dir*));
  ExploreStep(*base*, *dir*);
  ExploreStep(*base*, cw(*dir*));
else
  // choose different order, see page 14 ff
  Determine the types of the components using the layer numbers of the surrounding cells;
  if No component of type III exists **then**
    Use the left-hand rule, but omit the first possible step.
  else
    Visit the component of type III at last.
  end if
end if

ExploreStep($base$**,** dir**):**

if unexplored(*base*, *dir*) **then**
  Walk on shortest path using known cells to *base*;
  move(*dir*);
  ExploreCell(*dir*);
end if

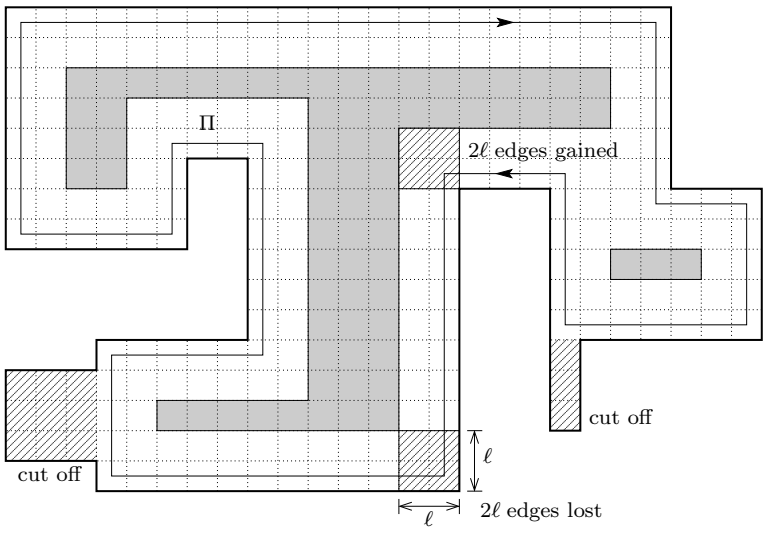


Figure 10: The 2-offset (shaded) of a grid polygon P.

Proof. First, we can cut off blind alleys that are narrower than 2ℓ, because those parts of P do not affect the ℓ-offset. We walk clockwise around the boundary cells of the remaining polygon, see Figure 10. For every left turn the offset gains at most 2ℓ edges and for every right turn the offset looses at least 2ℓ edges. O'Rourke [25] showed that #vertices $= 2 \cdot$#reflex vertices$+4$ holds for orthogonal polygons, so there are four more right turns than left turns. □

Definition 4 allows us to specify the detection and handling of a split cell in SmartDFS. We start with the handling of a split cell and defer split cell detection.

Let us consider the situation shown in Figure 11(i) to explain the handling of a split cell. SmartDFS has just met the first split cell, c, in the fourth layer of P. P divides into three parts:

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

where K_1 and K_2 denote the connected components of the set of unvisited cells. In this case it is reasonable to explore the component K_2 first, because the start cell s is closer to K_1; that is, we can extend K_1 with ℓ layers, such that the resulting polygon contains the start cell s.

More generally, we want to divide our polygon P into two parts, P_1 and P_2, so that each of them is an extension of the two components. Both polygons overlap in the area around the split cell c. At least one of these polygons contains the start cell. If only one of the polygons contains s, we want our strategy to explore this part at last, expecting that in this part the path from the last visited cell back to s is the shorter than in the other

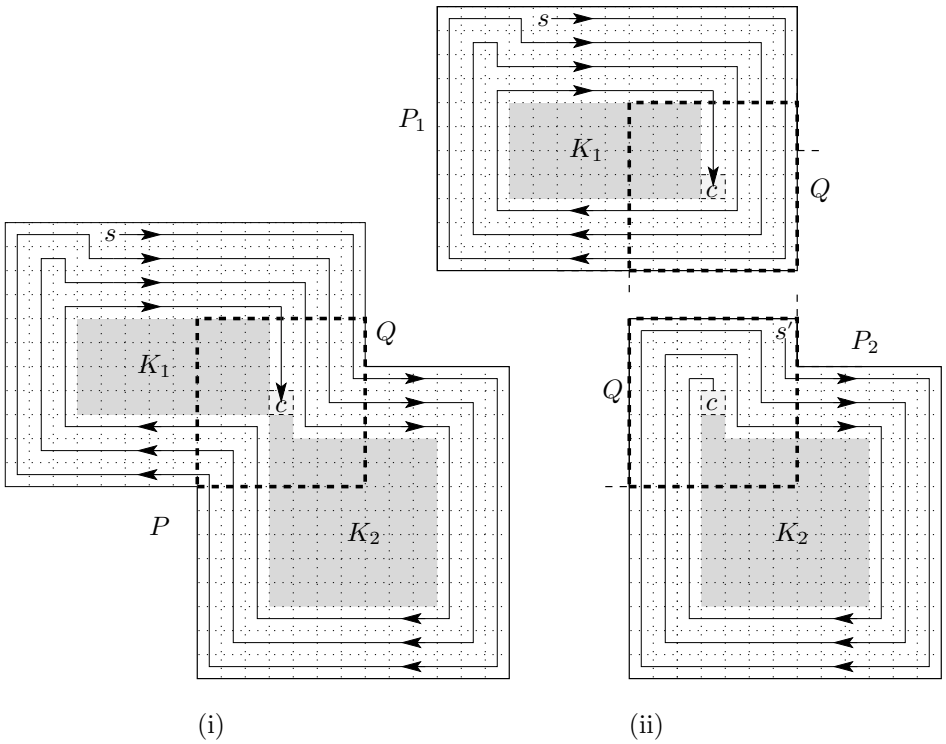


Figure 11: A decomposition of P at the split cell c and its handling in SmartDFS.

part. Vice versa, if there is a polygon that does *not* contain s, we explore the corresponding component first. In Figure 11, SmartDFS recursively enters K_2, returns to the split cell c, and explores the component K_1 next.

In the preceding example, there is only one split cell in P, but in general there will be a sequence of split cells, $c_1, \ldots, c_k$. In this case, we apply the handling of split cells in a recursive way; that is, if a split cell $c_{i+1}, 1 \leq i < k$, is detected in one of the two components occurring at c_i we proceed the same way as described earlier. Only the role of the start cell is now played by the preceding split cell c_i. In the following, the term *start cell* always refers to the start cell of the current component; that is, either to s or to the previously detected split cell. Further, it may occur that three components arise at a split cell, see Figure 14(i) on page 15. We handle this case as two successive polygon splits occurring at the same split cell.

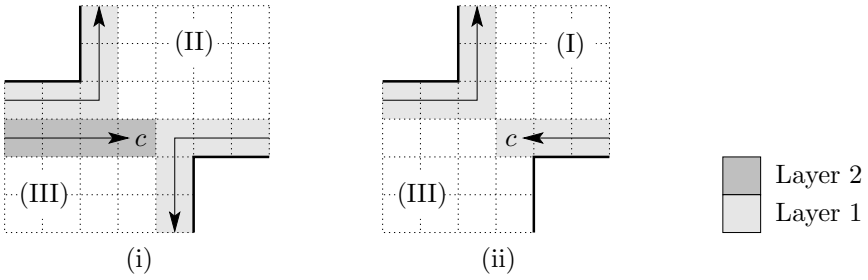


Figure 12: Several types of components.

Visiting Order

We use the layer numbers to decide which component we have to visit at last. Whenever a split cell occurs in layer ℓ, every component is one of the following types, see Figure 12:

I. K_i is *completely* surrounded by layer ℓ[5]
II. K_i is *not* surrounded by layer ℓ
III. K_i is *partially* surrounded by layer ℓ

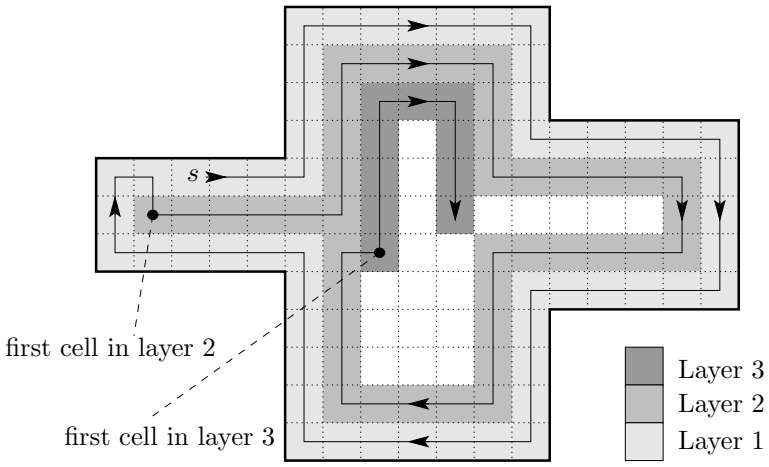


Figure 13: Switching the current layer.

There are two cases, in which SmartDFS switches from a layer $\ell - 1$ to layer ℓ. Either it reaches the first cell of layer $\ell - 1$ in the current component and thus passes the start cell—see, for example, the switch from layer 1 to layer 2 in Figure 13—, or it hits another cell of layer $\ell - 1$ but no polygon split occurs, such as the switch from layer 2 to layer 3 in in Figure 13. In the second case, the considered start cell must be located in a narrow passage that is completely explored; otherwise, the strategy would be able to reach

[5]More precisely, the part of layer ℓ that surrounds K_i is completely visited. For convenience, we use the slightly sloppy, but shorter form.

the first cell of layer $\ell - 1$ as in the first case. In both cases the part of P surrounding a component of type III contains the first cell of the current layer ℓ as well as the start cell. Therefore, it is reasonable to explore the component of type III at last.

There are two cases, in which no component of type III exists when a split cell is detected:

1. The part of the polygon that contains the preceding start cell is explored completely, see for example Figure 14(i). In this case the order of the components makes no difference.[6]

2. Both components are completely surrounded by a layer, because the polygon split and the switch from one layer to the next occurs within the same cell, see Figure 14(ii). A step that follows the left-hand rule will move towards the start cell, so we just omit this step. More precisely, if the the robot can walk to the left, we prefer a step forward to a step to the right. If the robot cannot walk to the left but straight forward, we proceed with a step to the right.

We proceed with the rule in case 2 whenever there is no component of type III, because the order in case 1 does not make a difference.

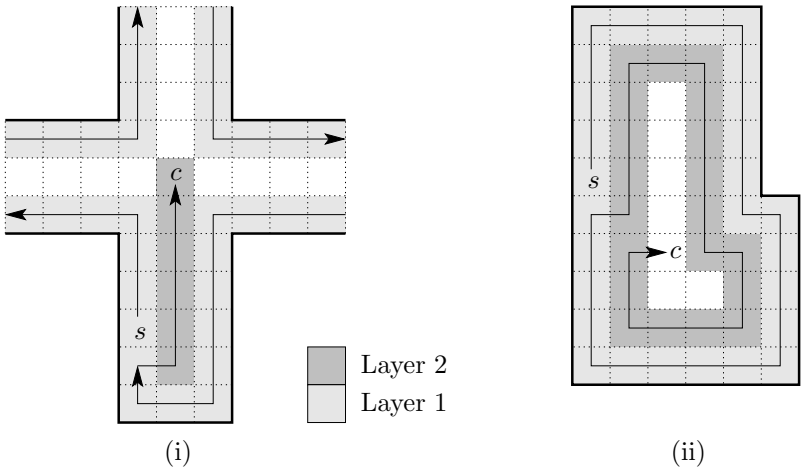


Figure 14: No component of type III exists.

[6]In Figure 14(i) we gain two steps, if we explore the part left to the splitcell at last and do not return to the split cell after this part is completely explored, but return immediately to the start cell. But decisions like this require facts of much more global type than we consider up to now. However, for the analysis of our strategy and the upper bound shortcuts like this do not matter.

An Upper Bound on the Number of Steps

For the analysis of our strategy we consider two polygons, P_1 and P_2, as follows. Let Q be the square of width $2q+1$ around c with

$$q := \begin{cases} \ell, & \text{if } K_2 \text{ is of type I} \\ \ell - 1, & \text{if } K_2 \text{ is of type II} \end{cases},$$

where K_2 denotes the component that is explored first, and ℓ denotes the layer in which the split cell was found. We choose $P_2 \subset P \cup Q$ such that $K_2 \cup \{c\}$ is the q-offset of P_2, and $P_1 := ((P \backslash P_2) \cup Q) \cap P$, see Figure 11. The intersection with P is necessary, because Q may exceed the boundary of P. Note that at least P_1 contains the preceding start cell. There is an arbitrary number of polygons P_2, such that $K_2 \cup \{c\}$ is the q-offset of P_2, because blind alleys of P_2 that are not wider than $2q$ do not affect the q-offset. To ensure a unique choice of P_1 and P_2, we require that both P_1 and P_2 are connected, and both $P \cup Q = P_1 \cup P_2$ and $P_1 \cap P_2 \subseteq Q$ are satisfied.

The choice of P_1, P_2 and Q ensures that the robot's path in $P_1 \backslash Q$ and in $P_2 \backslash Q$ do not change compared to the path in P. The parts of the robot's path that lead from P_1 to P_2 and from P_2 to P_1 are fully contained in the square Q. Just the parts inside Q are bended to connect the appropriate paths inside P_1 and P_2, see Figure 11 and Figure 15.

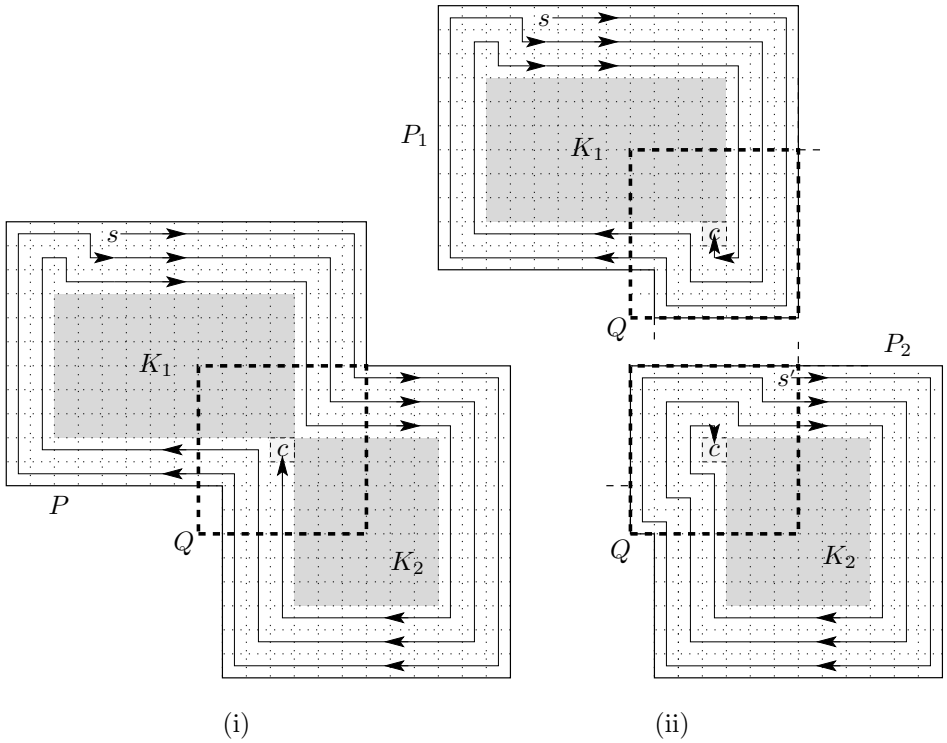


Figure 15: The component K_2 is of type I. The square Q may exceed P.

In Figure 11, K_1 is of type III and K_2 is of type II. A component of type I occurs, if we detect a split cell as shown in Figure 15. Note that Q may exceed P, but P_1 and P_2 are still well-defined.

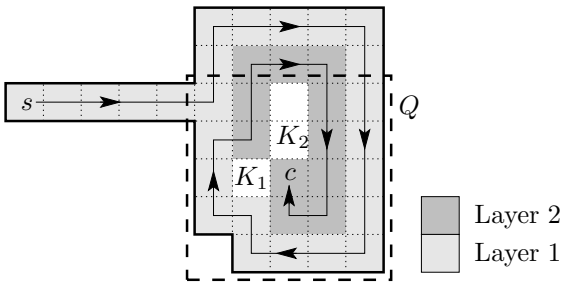


Figure 16: The order of components is not necessarily optimal.

Remark that we do not guarantee that the path from the last visited cell back to the corresponding start cell is the shortest possible path. See, for example, Figure 16: A split cell is met in layer 2. Following the preceding rule, SmartDFS enters K_2 first, returns to c, explores K_1, and returns to s. A path that visits K_1 first and moves from the upper cell in K_2 to s is slightly shorter. A case like this may occur if the first cell of the current layer lies in Q. However, we guarantee that there is only one return path in $P_1 \backslash Q$ and in $P_2 \backslash Q$; that is, only one path leads from the last visited cell back to the preceding start cell causing double visits of cells.

We want to visit every cell in the polygon and to return to s. Every strategy needs at least $C(P)$ steps to fulfill this task, where $C(P)$ denotes the number of cells in P. Thus, we can split the overall length of the exploration path, Π, into two parts, $C(P)$ and $\text{excess}(P)$, with $|\Pi| = C(P) + \text{excess}(P)$. $C(P)$ is a lower bound on the number of steps that are needed for the exploration task, whereas $\text{excess}(P)$ is the number of additional cell visits.

Because SmartDFS recursively explores $K_2 \cup \{c\}$, we want to apply the upper bound inductively to the component $K_2 \cup \{c\}$. If we explore P_1 with SmartDFS until c is met, the set of unvisited cells of P_1 is equal to K_1, because the path outside Q do not change. Thus, we can apply our bound inductively to P_1, too. The following lemma gives us the relation between the path lengths in P and the path lengths in the two components.

Lemma 6 *Let P be a simple grid polygon. Let the robot visit the first split cell, c, which splits the unvisited cells of P into two components K_1 and K_2, where K_2 is of type I or II. With the preceding notations we have*

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

Proof. The strategy SmartDFS has reached the split cell c and explores $K_2 \cup \{c\}$ with start cell c first. Because c is the first split cell, there is

no excess in $P_2 \backslash (K_2 \cup \{c\})$ and it suffices to consider excess$(K_2 \cup \{c\})$ for this part of the polygon. After $K_2 \cup \{c\}$ is finished, the robot returns to c and explores K_1. For this part we take excess(P_1) into account. Finally, we add one single step, because the split cell c is visited twice: once, when SmartDFS detects the split and once more after the exploration of $K_2 \cup \{c\}$ is finished. Altogether, the given bound is achieved. □

c is the first split cell in P, so $K_2 \cup \{c\}$ is the q-offset of P_2 and we can apply Lemma 5 to bound the number of boundary edges of $K_2 \cup \{c\}$ by the number of boundary edges of P_2. The following lemma allows us to charge the number of edges in P_1 and P_2 against the number of edges in P and Q.

Lemma 7 *Let P be a simple grid polygon, and let P_1, P_2 and Q be defined as earlier. The number of edges satisfy the equation*

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

Proof. Obviously, two arbitrary polygons P_1 and P_2 always satisfy

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

Let $Q' := P_1 \cap P_2$. Note that Q' is not necessarily the same as Q, see, for example, Figure 15. With $P_1 \cup P_2 = P \cup Q$ we have

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

The latter equation holds because $Q' = P \cap Q$. □

Finally, we need an upper bound for the length of a path inside a grid polygon.

Lemma 8 *Let Π be the shortest path between two cells in a grid polygon P. The length of Π is bounded by*

$$|\Pi| \leq \frac{1}{2} E(P) - 2 \; .$$

Proof. W.l.o.g. we can assume that the start cell, s, and the target cell, t, of Π belong to the first layer of P, because we are searching for an upper bound for the shortest path between two arbitrary cells.

Observe the path Π_L from s to t in the first layer that follows the boundary of P clockwise and the path Π_R that follows the boundary counterclockwise. The number of edges along these paths is at least four greater than

the number of cells visited by Π_L and Π_R using an argument similar to the proof of Lemma 5. Therefore we have:

$$|\Pi_L| + |\Pi_R| \leq E(P) - 4.$$

In the worst case, both paths have the same length, so $|\Pi(s,t)| = |\Pi_L| = |\Pi_R|$ holds. With this we have

$$2 \cdot |\Pi(s,t)| \leq E(P) - 4 \implies |\Pi(s,t)| \leq \frac{1}{2}E(P) - 2.$$

□

Now, we are able to show our main theorem:

Theorem 9 *Let P be a simple grid polygon with C cells and E edges. P can be explored with*

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

steps. This bound is tight.

Proof. C is the number of cells and thus a lower bound on the number of steps that are needed to explore the polygon P. We show by an induction on the number of components that $\text{excess}(P) \leq \frac{1}{2}E(P) - 3$ holds.

For the induction base we consider a polygon without any split cell: SmartDFS visits each cell and returns on the shortest path to the start cell. Because there is no polygon split, all cells of P can be visited by a path of length $C - 1$. By Lemma 8 the shortest path back to the start cell is not longer than $\frac{1}{2}E - 2$; thus, $\text{excess}(P) \leq \frac{1}{2}E(P) - 3$ holds.

Now, we assume that there is more than one component during the application of SmartDFS. Let c be the first split cell detected in P. When SmartDFS reaches c, two new components, K_1 and K_2, occur. We consider the two polygons P_1 and P_2 defined as earlier, using the square Q around c.

W.l.o.g. we assume that K_2 is recursively explored first with c as start cell. After K_2 is completely explored, SmartDFS proceeds with the remaining polygon. As shown in Lemma 6 we have

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

Now, we apply the induction hypothesis to P_1 and $K_2 \cup \{c\}$ and get

$$\text{excess}(P) \leq \frac{1}{2}E(P_1) - 3 + \frac{1}{2}E(K_2 \cup \{c\}) - 3 + 1 \ .$$

By applying Lemma 5 to the q-offset $K_2 \cup \{c\}$ of P_2 we achieve

$$\begin{aligned} \text{excess}(P) &\leq \frac{1}{2}E(P_1) - 3 + \frac{1}{2}(E(P_2) - 8q) - 3 + 1 \\ &= \frac{1}{2}(E(P_1) + E(P_2)) - 4q - 5 \ . \end{aligned}$$

From Lemma 7 we conclude $E(P_1) + E(P_2) \leq E(P) + 4(2q + 1)$. Thus, we get excess$(P) \leq \frac{1}{2}E(P) - 3$.

In Section 3 we have already seen that the bound is exactly achieved in polygons that do not contain any 2×2-square of free cells. □

Competitive Factor

So far we have shown an upper bound on the number of steps needed to explore a polygon that depends on the number of cells and edges in the polygon. Now, we want to analyze SmartDFS in the competitive framework.

Corridors of width 1 or 2 play a crucial role in the following, so we refer to them as *narrow passages*. More precisely, a cell, c, belongs to a narrow passage, if c can be removed without changing the layer number of any other cell.

It is easy to see that narrow passages are explored optimally: In corridors of width 1 both SmartDFS and the optimal strategy visit every cell twice, and in the other case both strategies visit every cell exactly once.

We need two lemmata to show a competitive factor for SmartDFS. The first one gives us a relation between the number of cells and the number of edges for a special class of polygons.

Lemma 10 *For a simple grid polygon, P, with $C(P)$ cells and $E(P)$ edges, and without any narrow passage or split cells in the first layer, we have*

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

Proof. Consider a simple polygon, P. We successively remove a row or column of at least three boundary cells, maintaining our assumption that the polygon has no narrow passages or split cells in the first layer. These assumptions ensure that we can always find such a row or column (i.e., if we cannot find such a row or column, the polygon has a narrow passage or a split cell in the first layer). Thus, we remove at least three cells and at most two edges. This decomposition ends with a 3×3 block of cells that fulfills $E = \frac{2}{3}C + 6$. Now, we reverse our decomposition; that is, we successively add all rows and columns until we end up with P. In every step, we add at least three cells and at most two edges. Thus, $E \leq \frac{2}{3}C + 6$ is fulfilled in every step. □

For the same class of polygons, we can show that SmartDFS behaves slightly better than the bound in Theorem 9.

Lemma 11 *A simple grid polygon, P, with $C(P)$ cells and $E(P)$ edges, and without any narrow passage or split cells in the first layer can be explored using no more steps than*

$$S(P) \leq C(P) + \frac{1}{2}E(P) - 5\,.$$

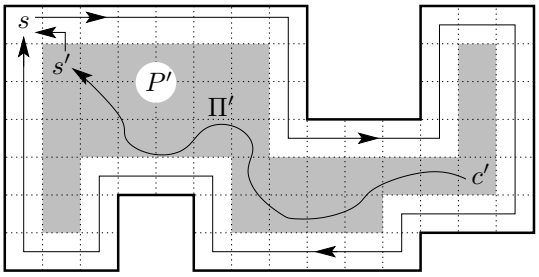


Figure 17: For polygons without narrow passages or split cells in the first layer, the last explored cell, c', lies in the 1-offset, P' (shaded).

Proof. In Theorem 9 we have seen that $S(P) \leq C(P) + \frac{1}{2}E(P) - 3$ holds. To show this theorem, we used Lemma 8 on page 18 as an upper bound for the shortest path back from the last explored cell to the start cell. Lemma 8 bounds the shortest path from a cell, c, in the first layer of P to the cell c' that maximizes the distance to c inside P; thus, c' is located in the first layer of P, too.

Because P has neither narrow passages nor split cells in the first layer, we can explore the first layer of P completely before we visit another layer, see Figure 17. Therefore, the last explored cell, c', of P is located in the 1-offset of P. Let P' denote the 1-offset of P, and s' the first visited cell in P'. Remark that s and s' are at least touching each other, so the length of a shortest path from s' to s is at most 2. Now, the shortest path, Π, from c' to s in P is bounded by a shortest path, Π', from c' to s' in P' and a shortest path from s' to s:

$$|\Pi| \leq |\Pi'| + 2\,.$$

The path Π', in turn, is bounded using Lemma 8 by

$$|\Pi'| \leq \frac{1}{2}E(P') - 2\,.$$

By Lemma 5 (page 10), $E(P') \leq E(P) - 8$ holds, and altogether we get

$$|\Pi| \leq \frac{1}{2}E(P) - 4\,,$$

which is two steps shorter than stated in Lemma 8. □

Now, we can prove the following

Theorem 12 *The strategy* SmartDFS *is* $\frac{4}{3}$*-competitive.*

Proof. Let P be a simple grid polygon. In the first stage, we remove all narrow passages from P and get a sequence of (sub-)polygons P_i, $i = 1, \ldots, k$, without narrow passages. For every P_i, $i = 1, \ldots, k-1$, the optimal

strategy in P explores the part of P that corresponds to P_i up to the narrow passage that connects P_i with P_{i+1}, enters P_{i+1}, and fully explores every P_j with $j \geq i$. Then it returns to P_i and continues with the exploration of P_i. Further, we already know that narrow passages are explored optimally. This allows us to consider every P_i separately without changing the competitive factor of P.

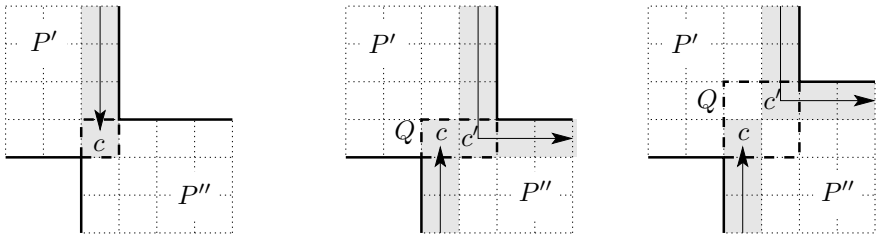
Now, we observe a (sub-)polygon P_i. We show by induction on the number of split cells in the first layer that $S(P_i) \leq \frac{4}{3}C(P_i) - 2$ holds. Note that this is exactly achieved in polygons of size $3 \times m$, m even, see Figure 18.

SmartDFS optimal strategy

Figure 18: In a corridor of width 3 and even length, $S(P) = \frac{4}{3}\, S_{\mathrm{Opt}}(P) - 2$ holds.

If P_i has no split cell in the first layer (induction base), we can apply Lemma 11 and Lemma 10:

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



(i) (ii) (iii)

Figure 19: Three cases of split cells, (i) component of type II, (ii) and (iii) component of type I.

Two cases occur if we meet a split cell, c, in the first layer, see Figure 19. In the first case, the new component was never visited before (component of type II, see page 14). Here, we define $Q := \{c\}$. The second case occurs, because the robot meets a cell, c', that is in the first layer and touches the current cell, c, see for example Figure 19(ii) and (iii). Let Q be the smallest rectangle that contains both c and c'.

Similar to the proof of Theorem 9, we split the polygon P_i into two parts, both including Q. Let P'' denote the part that includes the component of type II or III, P' the other part. For $|Q| = 1$, see Figure 19(i), we conclude $S(P_i) = S(P') + S(P'')$ and $C(P_i) = C(P') + C(P'') - 1$. Applying the induction hypothesis to P' and P'' yields

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

For $|Q| \in \{\,2, 4\,\}$ we gain some steps by merging the polygons. If we consider P' and P'' separately, we count the steps from c' to c—or vice versa—in both polygons, but in P_i the path from c' to c is replaced by the exploration path in P''. Thus, we have $S(P_i) = S(P') + S(P'') - |Q|$ and $C(P_i) = C(P') + C(P'') - |Q|$. This yields

$$\begin{aligned} S(P_i) &= S(P') + S(P'') - |Q| \\ &\leq \frac{4}{3}\,C(P') - 2 + \frac{4}{3}\,C(P'') - 2 - |Q| \\ &= \frac{4}{3}\,C(P_i) + \frac{1}{3}\,(|Q| - 6) - 2 \quad < \quad \frac{4}{3}\,C(P_i) - 2\,. \end{aligned}$$

The optimal strategy needs at least C steps, which, altogether, yields a competitive factor of $\frac{4}{3}$. □

Split Cell Detection

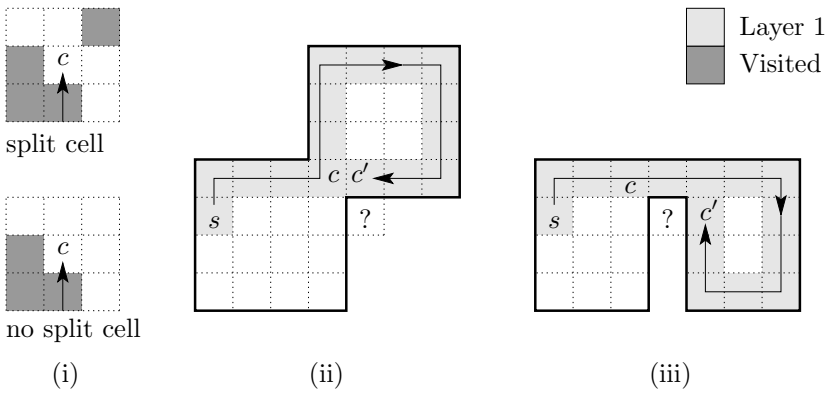


Figure 20: (i) Detecting a split cell, (ii) and (iii) a polygon split occurs in layer 1.

Finally, we describe the *detection* of a split cell. In the first instance, let us assume that the robot already moves in a layer $\ell > 1$. In Section 4.1 we defined that a split cell divides the graph of unexplored cells into two parts when the split cell is visited. Because the polygon is simple, we can determine a global split using a local criterion. We observe the eight cells surrounding the current robot's position. If there is more than one connected set of visited cells in this block, the current robot position is obviously a split cell, see Figure 20(i). Remark that we can execute this test although the robot's sensors do not allow us to access all eight cells around the current position. We are interested only in visited cells, so we can use the robot's map of visited cells for our test in layers $\ell > 1$. Unfortunately, this test method fails in layer 1, because the robot does not know the "layer 0", the polygon walls. However, we want to visit the component that has no visited cell in the current layer (type II) first; therefore, a step that follows the left-hand rule is correct. The strategy behaves correctly, although we do not report the splitcell explicitly. See, for example, Figure 20(ii) and (iii): In both cases the polygon split cannot be detected in c, because the cell marked with '?' is not known at this time. The split will be identified and handled correctly in c'.

5 Exploring Polygons with Holes

In an environment with obstacles (*holes*) it is not obvious how to detect and to handle split cells. When a polygon split is detected, the robot may be far away from the split cell, because it had no chance to recognize the split before reaching its current position. For example, in Figure 21(i) the robot has surrounded one single obstacle and c is a split cell, whereas in (ii) there are two obstacles and c is no split cell. Both situations cannot be distinguished until the cell c' is reached. So we use a different strategy to explore environments with obstacles.

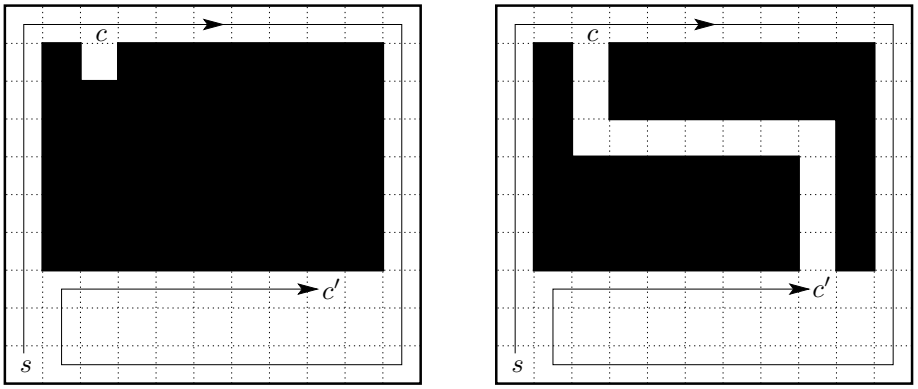


Figure 21: In an environment with obstacles, the robot may detect a split on a position far away from the splitcell, (i) c was a split cell, (ii) c was no split cell.

5.1 An Exploration Strategy

The basic idea of our strategy, CellExplore, is to *reserve* all cells right to the covered path for the way back. As in SmartDFS we use the left-hand rule; that is, the robot proceeds keeping the polygon's boundary or the reserved cells on its left side. CellExplore uses two modes. In the forward mode the robot enters undiscovered parts of the polygon, and in the backward mode the robot leaves known parts, see Algorithm 5.1 on page 27. We require that the robot starts with its back to a wall.

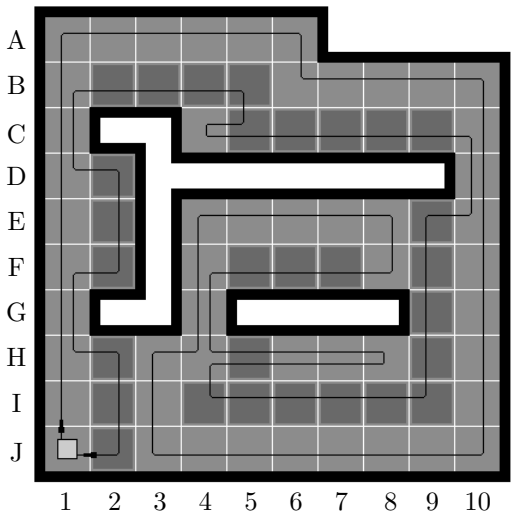


Figure 22: Example of an exploration tour produced by CellExplore (Screenshot using [13]; the white cells are holes, dark gray cells are reserved).

Figure 22 shows an example of an exploration tour. The robot starts in J1 and explores the polygon in the forward mode until F8 is reached. There, the robot switches to the backward mode and explores the reserved cells F7–F5. The path from F5 to H5 is blocked by the hole in G5, so the robot walks on the cells F4–H4 which have been visited already in the forward mode. In H5 the robot discovers the unreserved and unexplored cell H6, switches back to the forward mode and explores the cells H6–H8. Note that no cells can be reserved in this case, because the cells I6–I8 have already been reserved during the exploration of J8–J6. Therefore, the robot walks the same path back to H5 and continues the return path in the backward mode. In the forward mode, the robot could not reserve a cell from H3, so we move via H4 to I4 and proceed with the return path in the backward mode. The cells D9, C2 and G2 are blocked, so the robot has to circumvent these cells using visited cells. In C5 another unreserved and unexplored cell is discovered, so we switch to the forward mode and visit C4.

Algorithm 5.1 CellExplore

Forward mode:

- The polygon is explored following the left-hand rule: For every entered cell the robot tries to extend its path to an adjacent, unexplored, and unreserved cell, preferring a step to the left[7] over a straight step over a step to the right.
- All unexplored and unreserved cells right to the covered path are reserved for the return path by pushing them onto a stack.[8] If no cell right to the robot's current position can be reserved—because there is a hole or the corresponding cell is already reserved or explored—the robot's position is pushed onto the stack for the return path.
- Whenever no step in the forward mode is possible, the strategy enters the backward mode.

Backward mode:

- The robot walks back along the reserved return path.
- Whenever an unexplored and unreserved cell appears adjacent to the current position, the forward mode is entered again.

A straightforward improvement to the strategy CellExplore is to use in the backward mode the shortest path—on the cells known so far—to the first cell on the stack that is unexplored or has unexplored neighbors instead of walking back using every reserved cell, see the first improvement of DFS. From a practical point of view, this improvement is very reasonable, because the performance of the strategy increases in most environments. Unfortunately, the return path (i.e., the path walked in the backward mode) is no longer determined by a local configuration of cells. Instead, we need a global view, which complicates the analysis of this strategy. However, there are polygons that force this strategy to walk exactly the same return path as CellExplore without any optimization, see Figure 23, so this idea does not

[7] A "step to the left" or "turn left" means that the robot turns 90° counterclockwise and moves one cell forward. Analogously with "step to the right" or "turn right".

[8] If the robot turns left, we reserve three cells: right hand, straight forward, and forward-right to the robot's position. Note that we store the markers only in the robot's memory. This allows us to reserve cells that only touch the current cell, even if we are not able to determine whether these cells are free or blocked.

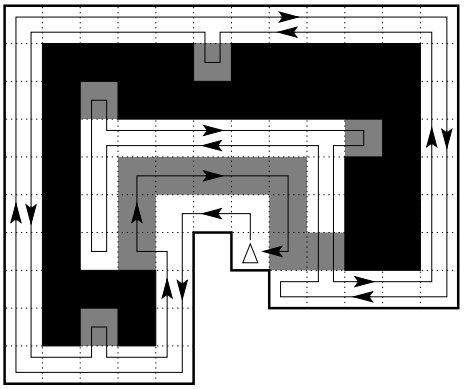
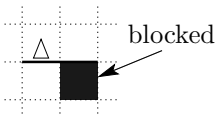


Figure 23: A polygon with $C = 69, \frac{E}{2} = 52, H = 1, W_{\text{cw}} = 2, S = 124 = C + \frac{1}{2}E + 3H + W_{\text{cw}} - 2$. The return path in this polygon cannot be shortened.

improve the worst case performance, and the upper bound for the number of steps is the same as in Theorem 18.

5.2 The Analysis of CellExplore



We analyze CellExplore in three steps: First, we analyze the single steps of the strategy with a local view and with one assumption concerning the robot's initial position, see the figure on the right. This results in a bound that depends on the number of left turns that are needed to explore the polygon. Then we discard the assumption, and finally we consider some global arguments to replace the number of left turns with parameters of the polygon.
To analyze our strategy CellExplore we use the following observations:

- CellExplore introduces a dissection of all cells into cells that are explored in the forward mode, denoted by $\mathcal{F}$, and cells that are explored in the backward mode, $\mathcal{B}$: $\mathcal{C} = \mathcal{F} \dot{\cup} \mathcal{B}$.
- All cells that are explored in the forward mode can be uniquely classified by the way the robot leaves them: either it makes a step to the left, a step forward or a step to the right. $\mathcal{F} = \mathcal{F}_L \dot{\cup} \mathcal{F}_F \dot{\cup} \mathcal{F}_R$.
- CellExplore defines a mapping $\varphi : \mathcal{B} \longrightarrow \mathcal{F}$, which assigns to every cell $c \in \mathcal{B}$ one cell $d \in \mathcal{F}$, so that c was reserved while d was visited.
- There exists a subset $\mathcal{D} \subseteq \mathcal{B}$ of the cells that have an unexplored and unreserved neighbor and the strategy switches from backward mode to forward mode. We call these cells *division cells*.

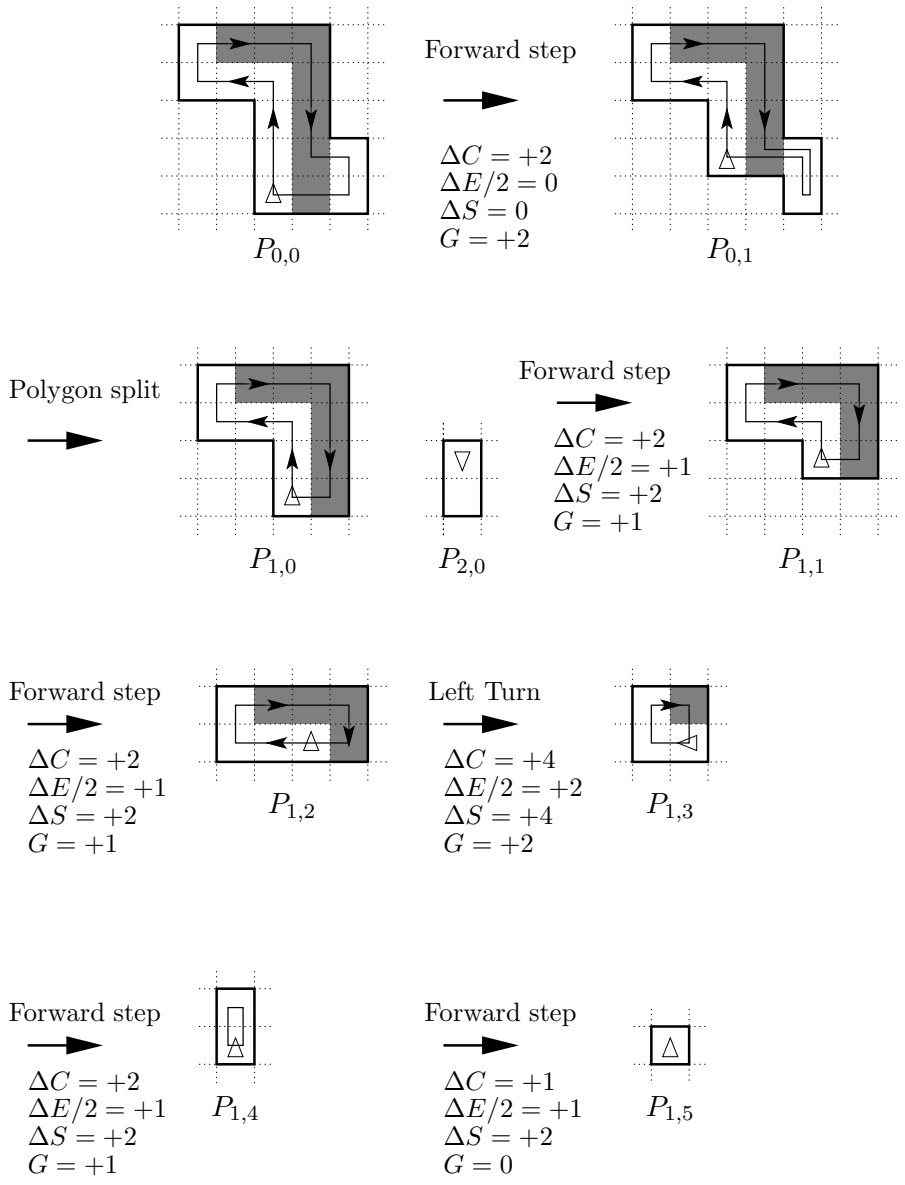


Figure 24: Decomposing a polygon. $\triangle$ denotes the start cell and the initial direction. $\Delta C, \Delta E$ and ΔS denote the differences in the number of cells, edges, and steps, respectively. G denotes the balance.

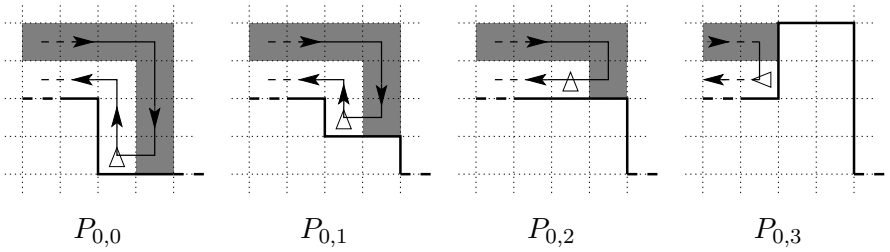


Figure 25: Decomposing a polygon. The shaded part shows the reserved cells.

We will analyze CellExplore by an induction over the cells in $\mathcal{F}$. Starting with the given polygon, P, and the given start cell, s, we can define a sequence $P_{k,i}$ of polygons ($P_{0,0} := P$) with start cells $s_{k,i}$ as follows: $P_{k,i+1}$ arises from $P_{k,i}$ by removing the start cell $s_{k,i}$ and all cells that are reserved in the first step in $P_{k,i}$ (i.e., every cell c with $\varphi(c) = s_{k,i}$). The start cell $s_{k,i+1}$ in the new polygon $P_{k,i+1}$ is the cell that the robot enters with its first step in $P_{k,i}$, see Figure 25. The reserved cells in this and all following figures are shown shaded; $\triangle$ denotes the start cell.

There is nothing to consider when the strategy enters the backward mode, because we remove all cells that are explored in the backward mode together with the forward cells. But what happens, if a division cell occurs; that is, the strategy switches from the backward mode to the forward mode?

Lemma 13 *If one of the cells reserved in the first step in $P_{k,i}$ is a division cell, then $P_{k,i}$ is split by removing $s_{k,i}$ and all cells that are reserved in this step (i.e., all cells c with $\varphi(c) = s_{k,i}$) into two or more not-connected components.*

Proof. Consider two components, P_1 and P_2, that are connected in P by some cells $\mathcal{X} \subset \mathcal{B}$. Let c_j be the first cell in $\mathcal{X}$ that is discovered on the return path, thus, P_2 is entered via c_j. In our successive decomposition, $\varphi(c_j)$ is the start cell of a polygon $P_{k,i}$. In $P_{k,i}$ all cells explored before $\varphi(c_j)$ are already removed. If there would be another connection, c_ℓ, between P_1 and P_2 at this time, c_ℓ would be discovered before c_j on the return path and P_2 would have been entered via c_ℓ in contradiction to our assumption that P_2 is entered via c_j. Thus, c_j must be the last cell that connects P_1 and P_2. □

If one or more of the cells to be removed are division cells, the polygon $P_{k,i}$ is divided into subpolygons $P_{k+1,0}, P_{k+2,0}, \ldots$ that are analyzed separately, see Figure 26. See Figure 24 for a more comprehensive example for the successive decomposition of a polygon.

Now, we are able give a first bound for the number of steps that CellExplore uses to explore a polygon.

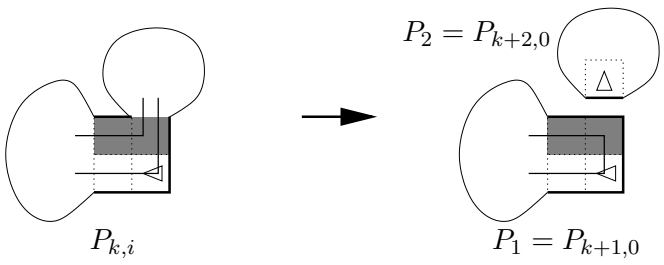


Figure 26: Handling of division cells.

Lemma 14 *Let us assume that the cell behind and right hand to the robot's position*[9] *is blocked. The number of steps, S, used to explore a polygon with C cells, E edges and H holes, is bounded by*

$$S \leq\ C + \frac{1}{2}E + H + 2L - 3,$$

where L denotes the number of the robot's left turns.

Proof. We observe the differences in the number of steps, cells, edges and holes between $P_{k,i}$ and $P_{k,i+1}$, and assume by induction that our upper bound for the length of the exploration tour holds for $P_{k,i+1}$ and for the separated subpolygons $P_{k+j,0}$. Therefore, we have to show that the limit is still not exceeded if we add the removed cells and merge the subpolygons. We want to show that the following inequation is satisfied in every step:

$$S \leq\ C + \frac{1}{2}E + H + 2L - 3.$$

Let G denote the "profit" made by CellExplore; that is, the difference between the actual number of steps and the upper bound. With G, the preceding inequation is equivalent to

$$C + \frac{1}{2}E + H + 2L - G - S - 3 = 0.$$

We have to consider three main cases: the division cells, the cells contained in $\mathcal{F}_L$ (left turns), and those contained in $\mathcal{F}_F \cup \mathcal{F}_R$ (forward steps and right turns). There is no need to consider right turns explicitly, because steps to the right can be handled as a sequence of forward steps, see Figure 27. The successive decomposition ends with one single cell ($C = 1, E = 4, H = 0$) for which $S = 0 = C + \frac{1}{2}E + H + 2L - 3$ holds (Induction base).

[9] In other words, the cell southeast to the robot's current position if the current direction is *north*.

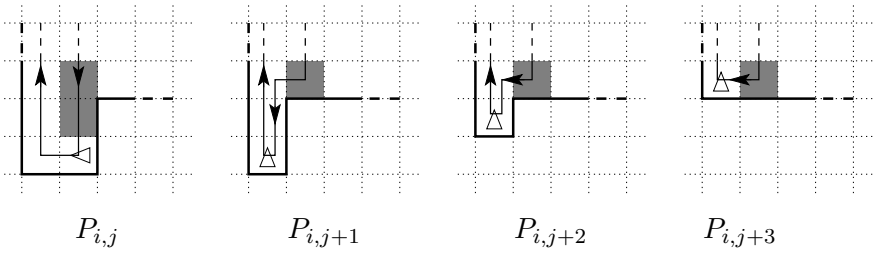


Figure 27: Decomposing a step to the right into several forward steps.

Division cells

If one of the cells that are reserved in the first step in $P_{k,i}$ is a division cell, $P_{k,i}$ is split into two polygons $P_{k+1,0}$ (P_1 for short) and $P_{k+2,0}$ (P_2 for short), see Lemma 13. We assume by induction that our upper bound is achieved in both polygons:

$$C_i + \frac{1}{2}E_i + H_i + 2L_i - G_i - S_i - 3 = 0, \quad i \in \{1, 2\}.$$

For the merge of P_1 and P_2 into one polygon P, we can state the following:

$$\begin{aligned} S &= S_1 + S_2 + \Delta S \\ E &= E_1 + E_2 + \Delta E \\ C &= C_1 + C_2 \\ H &= H_1 + H_2 \\ L &= L_1 + L_2 \\ G &= G_1 + G_2 + G_S, \end{aligned}$$

where G_S denotes the profit made by merging the polygons.

We want to show that our bound is achieved in P:

$$\begin{aligned} & C + \frac{1}{2}E + H + 2L - G - S - 3 \\ = \;& C_1 + C_2 + \frac{1}{2}(E_1 + E_2 + \Delta E) + H_1 + H_2 + 2L_1 \\ & + 2L_2 - G_1 - G_2 - G_S - S_1 - S_2 - \Delta S - 3 \\ = \;& \underbrace{C_1 + \frac{1}{2}E_1 + H_1 + 2L_1 - G_1 - S_1 - 3}_{=0} \\ & + \underbrace{C_2 + \frac{1}{2}E_2 + H_2 + 2L_2 - G_2 - S_2 - 3}_{=0} \\ & + \frac{1}{2}\Delta E - G_S - \Delta S + 3 \end{aligned}$$

$$
\begin{aligned}
&= 0 \\
\iff \quad G_S &= \frac{1}{2}\Delta E - \Delta S + 3
\end{aligned}
$$

Thus, for every polygon split we have to observe ΔE and ΔS. If G_S is positive, we gain some steps by merging the polygons, if G_S is negative, the merge incurs some costs.

The different configurations for a polygon split can be assigned to several classes, depending on the number of common edges between P_1 and P_2 and the way, the robot returns from P_2 to P_1—more precisely the distance between the step from P_1 to P_2 and the step from P_2 to P_1, compare for example Figure 30(3a) and (3b). Figure 30(i) and (ii) show some instances of one class. Because the actual values for ΔE and ΔS depend on only these two parameters, we do not list every possible polygon split in Figure 30, but some instances of every class.

The balances of the polygon splits are shown in table 1. Two cases have a negative balance, so we need one more argument to show that these cases do not incur any costs. Observe that after splitting $P_{k,i}$ the polygon P_1 starts with a cell from $\mathcal{F}_L$ in the cases (1b)–(5). Removing this block of 4 cells, we gain +2, see the first line of Figure 32. This covers the costs for the polygon split in the cases (4a) and (5a).

Forward steps

We have to consider several cases of forward steps, see Figure 31. The table lists the differences in the number of steps (ΔS), cells (ΔC), edges (ΔE) and holes (ΔH) if we *add* the considered cells to $P_{k,i+1}$. The last column shows $G = \Delta C + \frac{1}{2}\Delta E + \Delta H - \Delta S$. After removing the observed block of cells, the remaining polygon must still be connected; otherwise, we would have to consider a polygon split first. Thus, there are some cases, in which ΔH must be greater than zero.

It turns out that all cases have a positive balance, except those that violate the assumption in Lemma 14 that the cell behind and right hand to the robot's position is blocked, see the cases marked with (*) in Figure 31. Notice that the configurations shown in Figure 28 are left turns instead of forward steps!

To show that we have analyzed all possible cell configurations for a step forward, we use the following observations. When the robot makes a step forward, we know the following: Both behind and left hand to the robot are walls (otherwise it would have turned left), in front of the robot is no wall (otherwise it could not make a step forward). Right hand to the robot may or may not be a wall. In the latter case, we have three edges of interest that may or may not be walls, yielding $2^3 = 8$ cases, which can be easily enumerated. Two special cases occur by taking into consideration that the robot may enter the

observed block of cells from the upper cell or from the right cell, see for example Figure 31(5a) and (5b).

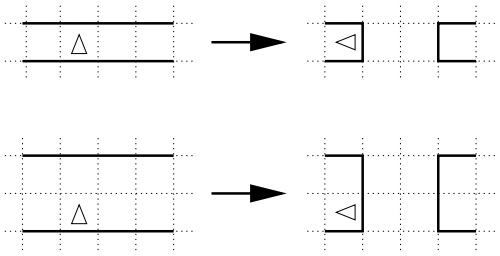
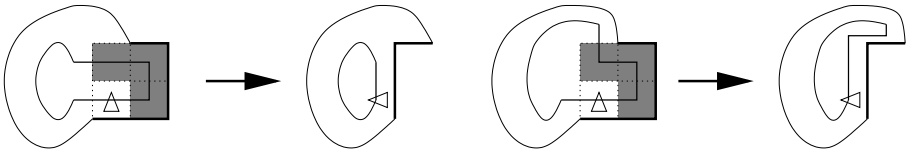


Figure 28: Configurations that are steps to the left instead of forward steps.

Steps to the left

Possible cases of left turns are shown in the figures 32–35. As in the previous case, the last column shows the balance. Again, we observe that all cases with a negative balance of -3 are a violation of our assumption that the cell behind and right hand to the robot's position is blocked. The negative balances of -2 are compensated by the addend $2L$ in our bound.



(i) Block is entered from left $\Delta S = 4$

(ii) Block is entered from upper cell $\Delta S = 2$

Figure 29: Another class of left turns.

When counting the number of steps, we assumed that the robot enters the block of cells from the same direction as it left the block (from the left as shown in the figures). The robot may enter the block from the upper cell as shown in Figure 29, but this would only increase the balance.

The completeness of the cases can be shown with the same argument as in the previous case: We know that there is a wall behind the robot and left hand to the robot is no wall. Examining a block of four cells for a left turn, we have six edges that may or may not be walls, and we have three cells that may or may not be holes, yielding corresponding configurations of edges. □

	ΔE	ΔS	G_S
(1)	-2	2	0
(2)	-4	0	1
(3a)	-6	0	0
(3b)	-6	-2	2
(4a)	-8	0	-1
(4b)	-8	-2	1
(4c)	-8	-4	3
(5a)	-10	0	-2
(5b)	-10	-2	0

Table 1: Balances of polygon splits

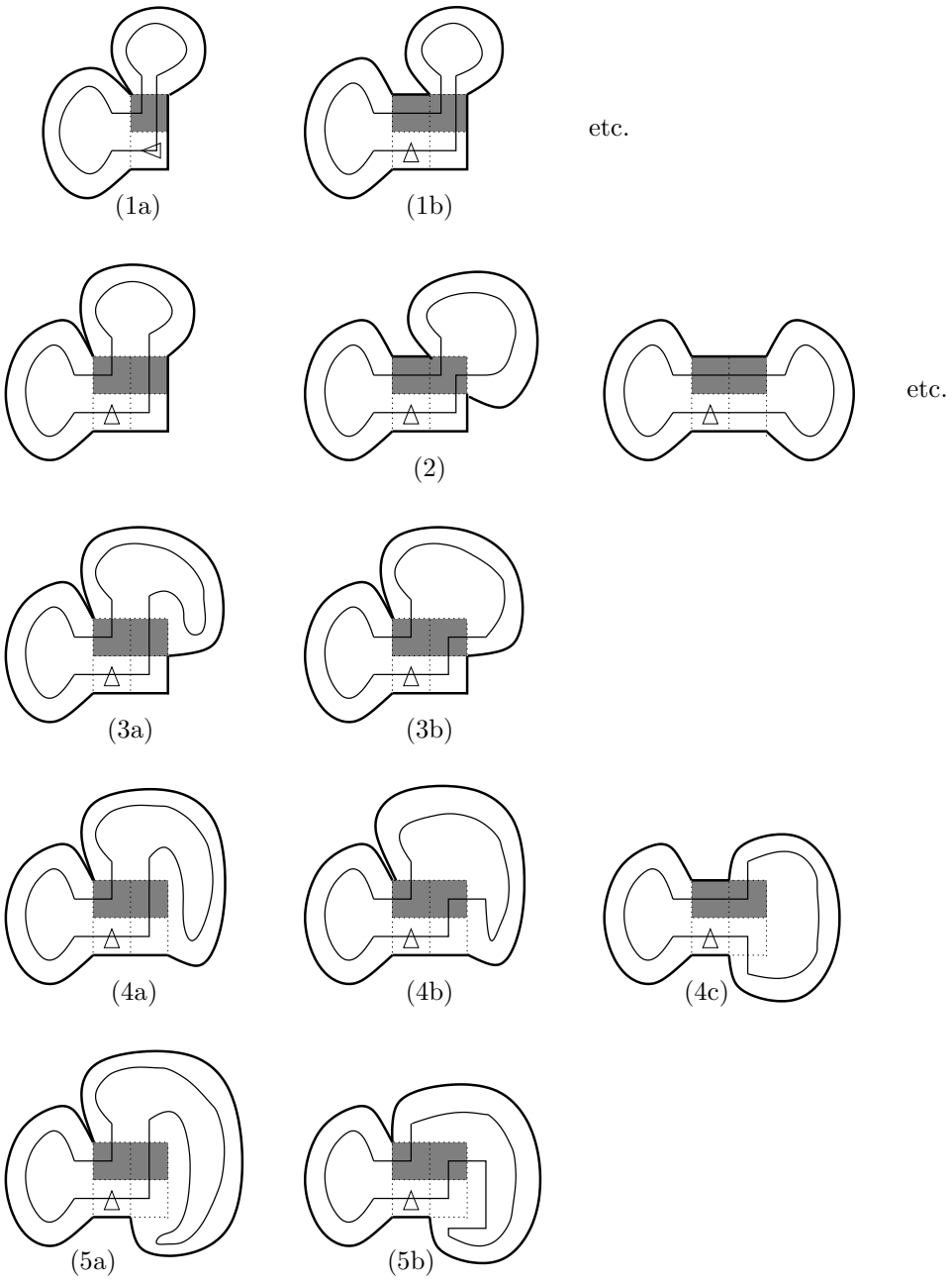
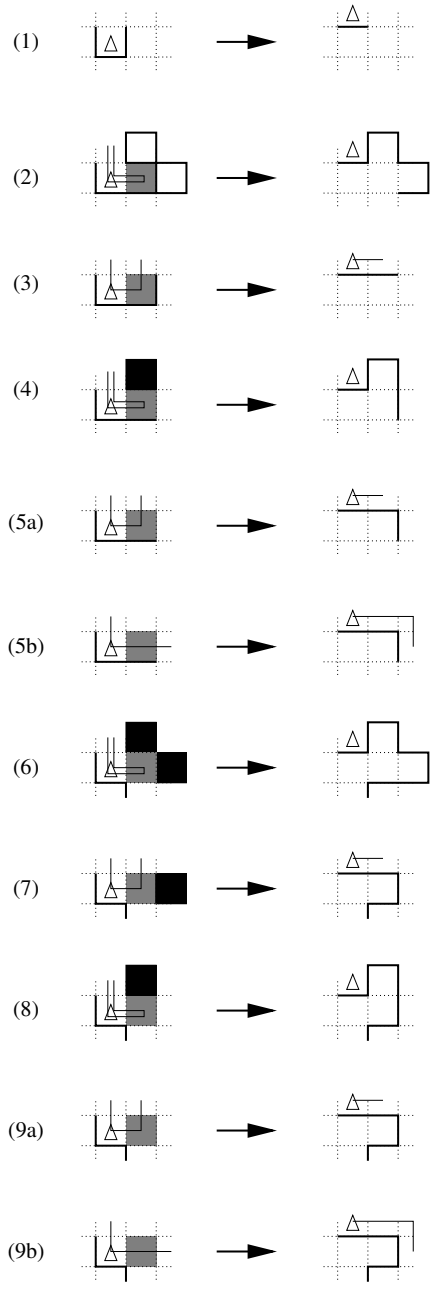
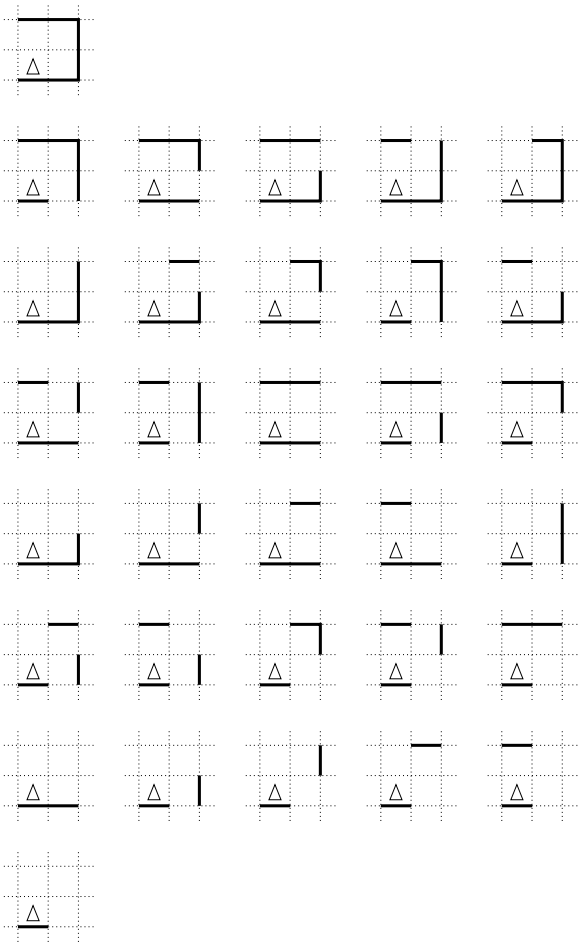


Figure 30: Some possible cases of polygon splits.



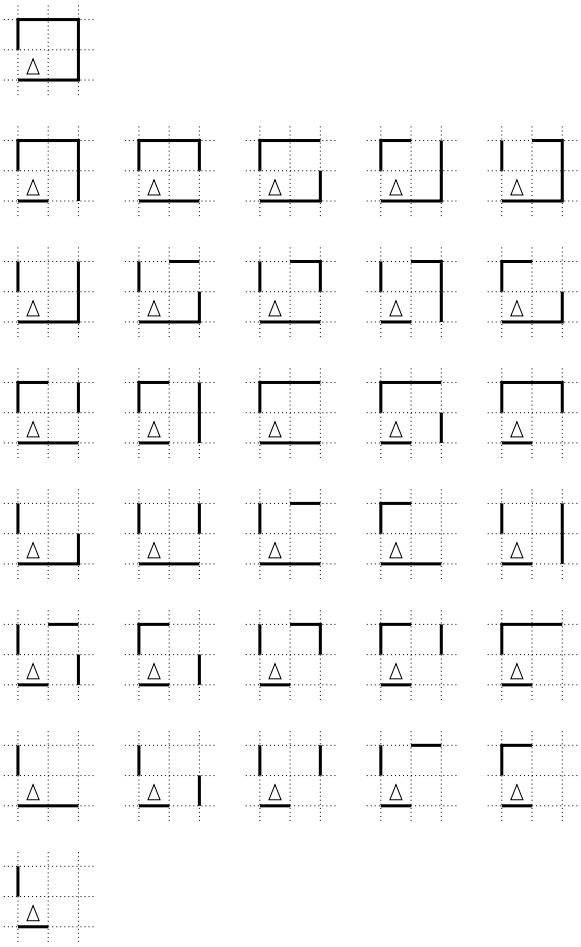
	ΔS	ΔC	$\Delta E/2$	ΔH	G	
(1)	+2	+1	+1	0	0	
(2)	+4	+2	+2	0	0	
(3)	+2 +2	+2 +2	+1 +1	0 +1	+1 +2	
(4)	+4	+2	+1	+1	0	
(5a)	+2 +2	+2 +2	0 0	0 +1	0 +1	
(5b)	0	+2	0	0	+2	
(6)	+4	+2	+1	+1	0	(*)
(7)	+2	+2	0	+1	+1	(*)
(8)	+4	+2	0	+1	−1	(*)
(9a)	+2 +2	+2 +2	−1 −1	0 +1	−1 0	(*)
(9b)	0	+2	−1	0	+1	(*)

Figure 31: Cell configurations for forward steps.



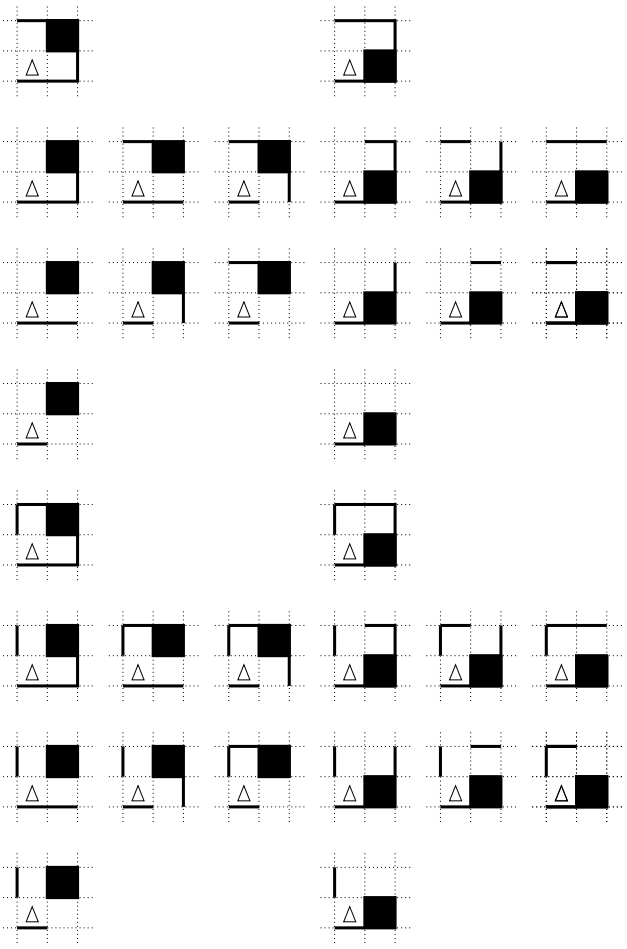
ΔS	ΔC	$\Delta E/2$	ΔH	G
+4	+4	+2	0	+2
+4	+4	+1	0	+1
+4	+4	+1	+1	+2
+4	+4	0	0	0
+4	+4	0	+1	+1
+4	+4	0	+2	+2
+4	+4	−1	0	−1
+4	+4	−1	+1	0
+4	+4	−1	+2	+1
+4	+4	−2	0	−2
+4	+4	−2	+1	−1
+4	+4	−2	+2	0
+4	+4	−2	+3	+1
+4	+4	−3	0	−3
+4	+4	−3	+1	−2
+4	+4	−3	+2	−1
+4	+4	−3	+3	0

Figure 32: Possible configurations for steps to the left (1).



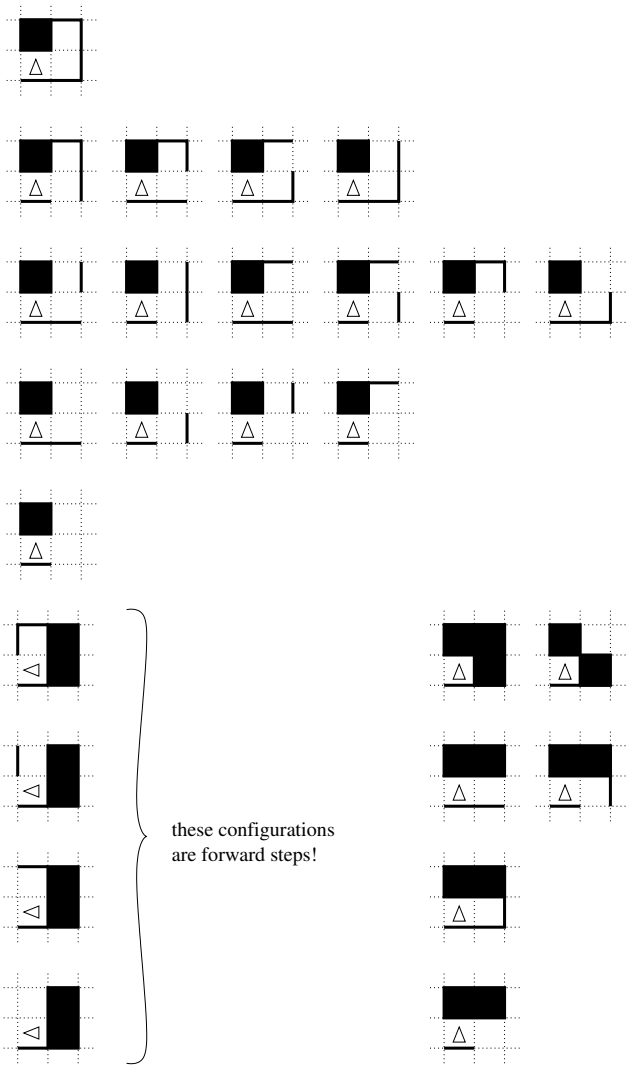
ΔS	ΔC	$\Delta E/2$	ΔH	G
+6	+4	+3	0	+1
+6	+4	+2	+1	+1
+6	+4	+1	+1	0
+6	+4	+1	+2	+1
+6	+4	0	+1	−1
+6	+4	0	+2	0
+6	+4	0	+3	+1
+6	+4	−1	+1	−2
+6	+4	−1	+2	−1
+6	+4	−1	+3	0
+6	+4	−2	+1	−3
+6	+4	−2	+2	−2
+6	+4	−2	+3	−1

Figure 33: Possible configurations for steps to the left (2).



ΔS	ΔC	$\Delta E/2$	ΔH	G
+4	+3	+2	0	+1
+4	+3	+1	0	0
+4	+3	+1	+1	+1
+4	+3	0	0	−1
+4	+3	0	+1	0
+4	+3	0	+2	+1
+4	+3	−1	0	−2
+4	+3	−1	+1	−1
+4	+3	−1	+2	0
+4	+3	−1	+3	+1
+6	+3	+3	0	0
+6	+3	+2	+1	0
+6	+3	+1	+1	−1
+6	+3	+1	+2	0
+6	+3	0	+1	−2
+6	+3	0	+2	−1
+6	+3	0	+3	0

Figure 34: Possible configurations for steps to the left (3).



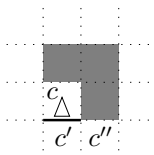
ΔS	ΔC	$\Delta E/2$	ΔH	G
+6	+3	+3	0	0
+6	+3	+2	+1	0
+6	+3	+1	+1	−1
+6	+3	+1	+2	0
+6	+3	0	+1	−2
+6	+3	0	+2	−1
+6	+3	0	+3	0
+6	+3	−1	+1	−3
+6	+3	−1	+2	−2
+6	+3	−1	+3	−1
+2	+1	+1	0	0
+4	+2	+1	+1	0
+4	+2	+2	0	0
+4	+2	0	+1	−1

Figure 35: Possible configurations for steps to the left (4).

Next, we want to discard the assumption we made in Lemma 14.

Lemma 15 *The assumption that the cell behind and right hand to the robot's position is blocked can be violated only in the robot's initial position; that is, in $P_{0,0}$ and not in any $P_{k,i}$ with $k + i > 0$.*

Proof.



Consider a robot located in some cell c with no wall behind and right hand to its position, and w.l.o.g. *dir* ='north'. If this is not the robot's initial position, but the position $s_{k,i}$ of a polygon $P_{k,i}$ occurring in the successive decomposition of the polygon, the robot must have entered the cell c from the cell c' below c in the polygon $P_{k,i-1}$. If the cell c'' right to c' is not a hole in $P_{k,i-1}$, c'' would be a reserved cell, and, thus, it would be removed together with c' in the step from $P_{k,i-1}$ to $P_{k,i}$. Consequently, when the robot has reached c, there would be a hole behind and right hand to its current position. □

Lemma 16 *The number of steps, S, used to explore a polygon with C cells, E edges and H holes, is bounded by*

$$S \leq \; C + \frac{1}{2}E + H + 2L - 2,$$

where L denotes the number of the robot's left turns.

Proof. Lemma 15 shows that the assumption in Lemma 14 can be violated only in the robot's initial position. On the other hand, we have seen in the proof of Lemma 14 that all cases that violate the assumption incur the costs of just one additional step. □

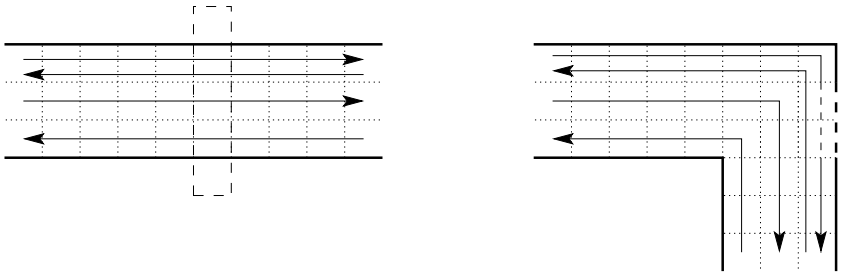


Figure 36: Corridors of odd width.

Our bound still depends on the number of left turns the robot makes while exploring the polygon. To give a bound that does not depend on the strategy, we introduce another property of grid polygons, a measure

to distinguish rather flat polygons from winded polygons; let us call it the *sinuosity* of P. Our motivation for introducing this property is the following observation: The robot may walk $n+1$ times through a corridor of width n, n odd, see Figure 36(i). The costs for this extra walk are covered by the corridor walls; more precisely, for every double visit we charge *two* polygon edges and get the addend $\frac{1}{2}E$ in the upper bound. If there is a left turn in the corridor, there are not enough boundary edges for balancing the extra walk. Figure 36(ii) shows a corridor of width 3 with a left turn. The steps shown with dashed lines cannot be assigned to edges, so we have to count the edges shown with dashed lines to balance the number of steps. We define two types of sinuosities, the clockwise and the counterclockwise sinuosity. Because CellExplore follows the left-hand rule, our strategy depends on the clockwise sinuosity. A similar strategy that follows the right-hand rule would depend on the counterclockwise sinuosity.

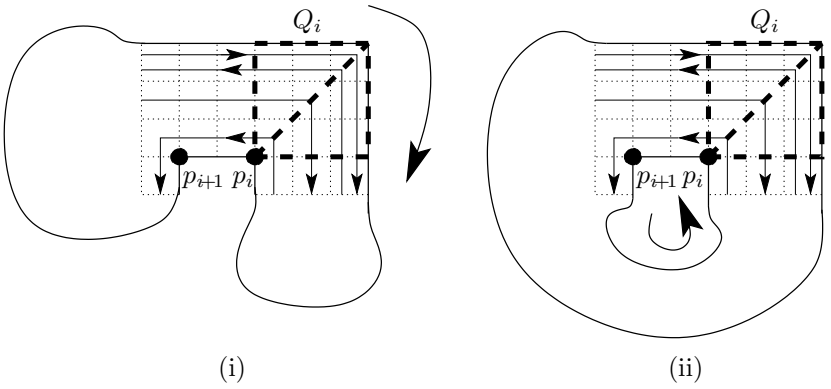


Figure 37: Contributions to W_{cw} by (i) the outer boundary, (ii) inner boundaries.

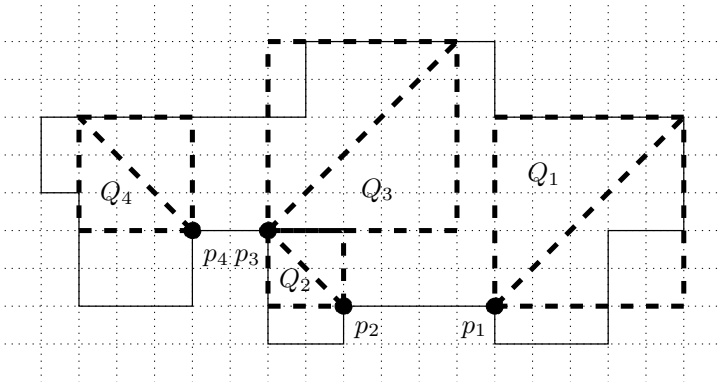


Figure 38: Reflex vertices p_i and the corresponding squares Q_i. $W_{\text{cw}} = q'_1 + q'_3 = 8, W_{\text{ccw}} = q'_4 = 2$.

Definition 17 Let the *clockwise sinuosity*, W_{cw}, and the *counterclockwise sinuosity*, W_{ccw}, of a grid polygon P be defined as follows: We trace the boundary of P—the outer boundary clockwise, the boundaries of the holes inside P counterclockwise—, and consider every pair, p_i and p_{i+1}, of consecutive reflex vertices, see Figure 37.

We trace the angular bisector between the two edges incident to p_i inside P until it hits the boundary of P. The resulting line segment defines the diagonal of a square, Q_i, see Figure 38.[10] Let q_i be the width of Q_i, analogously with q_{i+1}.

Because the robot needs some further steps only in odd corridors, we count only odd squares:

$$q'_i := \begin{cases} q_i - 1, & \text{if } q_i \text{ is odd} \\ 0, & \text{if } q_i \text{ is even} \end{cases}.$$

The need for additional edges may not only be caused by reflex vertices, but also by the start cell, see Figure 39(ii). Thus, we consider the squares $Q_{s_{\text{cw}}}$ and $Q_{s_{\text{ccw}}}$ from the start cell in clockwise and counterclockwise direction, respectively. Let $q'_{s_{\text{cw}}}$ and $q'_{s_{\text{ccw}}}$ be defined analogously to q'_i. Now, we define the clockwise sinuosity W_{cw} and the counterclockwise sinuosity W_{ccw} as

$$W_{\text{cw}} := q'_{s_{\text{ccw}}} + \sum_{i \geq 1} q'_{2i-1}, \quad \text{and} \quad W_{\text{ccw}} := q'_{s_{\text{cw}}} + \sum_{i \geq 1} q'_{2i}.$$

Figure 39 shows two examples for the definition of W_{cw}. Note that in (i) only one reflex vertex contributes to W_{cw}, and every edge we count here is needed.

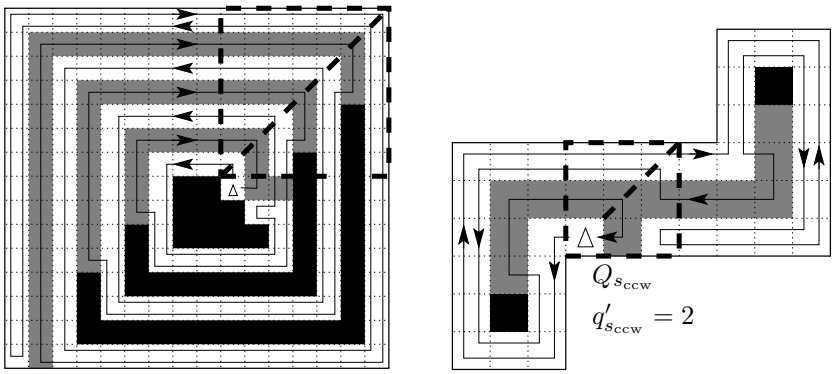


Figure 39: Examples for the definition of W_{cw}: (i) A polygon with $C = 193, \frac{E}{2} = 78, H = 3, W_{\text{cw}} = 6, S = 284$ (the bound for S is exactly achieved), (ii) the start cell contributes to W_{cw}, too ($C = 46, \frac{E}{2} = 23, H = 2, W_{\text{cw}} = 2, S = 74$).

[10] We can construct Q_i by "blowing up" a square around the cell in P that touches the boundary of P in p_i until the corner of Q_i opposite to p_i hits the polygon's boundary.

With the definition of W_{cw}, we can give our final result:

Theorem 18 *Let P be a grid polygon with C cells, E edges, H holes, and clockwise sinuosity W_{cw}. CellExplore explores P using no more than*

$$S \leq C + \frac{1}{2}E + W_{\text{cw}} + 3H - 2$$

steps. This bound is tight.

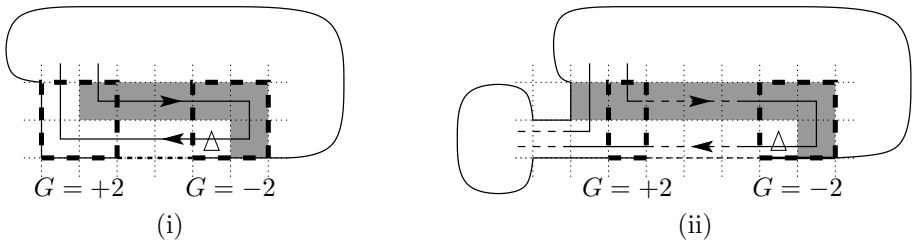


Figure 40: Left turn followed by (i) a right turn and (ii) a reduction.

Proof. We need some global arguments to charge the costs for a left turn to properties of P. So let us examine, which configurations may follow a left turn (after some forward steps):

- A right turn follows the left turn, see Figure 40(i). We gain +2 steps per right turn, so the possible costs of −2 for this left turn are covered.
- An obstacle follows the left turn. We can charge the obstacle with the costs for the left turn and get a factor of 3 for the number of obstacles. Every obstacle is charged at most once, because when the successive decomposition reaches the obstacle for the first time, the obstacle disappears; that is, the hole merges with the outer boundary.
- A reduction follows the left turn, see Figure 40(ii). Later in this section, we show that a reduction covers the costs of a left turn.
- Another left turn follows the observed left turn. In this case, there is no other property of P to be charged with this costs but the sinuosity W_{cw}, this follows directly from the definition of W_{cw}.

In the case of a reduction following a left turn we observe the number, d, of forward steps between the left turn and the reduction, as well as the cell marked with b, and—if $d \geq 1$—the cell marked with a in Figure 41(i). If b is blocked, b is either part of an obstacle inside the polygon or it is outside the polygon. In the first case, we charge the obstacle with the costs of the left turn as described earlier. In the second case we have a polygon split that

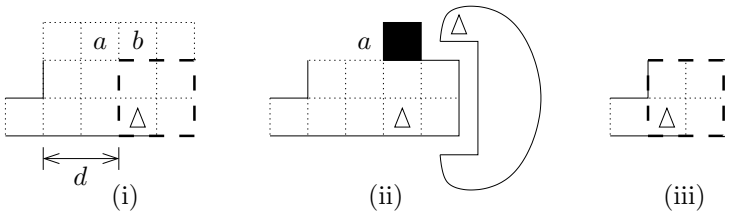


Figure 41: (i) A reduction follows a left turn, (ii) the left turn causes a polygon split, (iii) no forward steps between the left turn and the reduction ($d = 0$).

leaves us with a left turn that incurs no costs, see Figure 41(ii) and the first line of Figure 32. The same holds for the cell marked with a if there is at least one forward step between the left turn and the reduction (i.e., $d \geq 1$). Therefore, we assume that a and b are free cells in the following.

If the reduction follows immediately after the left turn ($d = 0$), see Figure 41(iii), we have one of the left turns shown in Figure 33, Figure 35 or the lower half of Figure 34. In any of these cases we have either a positive balance or we meet an obstacle ($\Delta H > 0$) and charge the costs for the left turn to the obstacle as earlier.

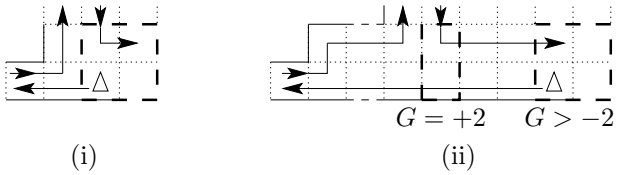


Figure 42: If a and b are free cells we gain 2. (i) $d = 1$, (ii) $d \geq 1$.

If there is one forward step between the left turn and the reduction ($d = 1$), the robot enters the 2×2 block of cells of the left turn not from the same side as it left it, because a and b are polygon cells. In Figure 42(i) the robot leaves the block to the left but enters it from above. This situation is described in Figure 29 on page 34 and reduces the costs for the left turn by 2, so the balance is either zero or positive. If there is more than one forward step ($d > 1$), we have either the same situation as in the preceding case, or the reduction from a corridor of width ≥ 3 to a corridor of width ≤ 2 shifts to the left, see Figure 42(ii), and eventually we reach a forward step as shown in Figure 31(5b) that gains $+2$ and covers the costs for the left turn.

Altogether, we are able to charge the costs for every left turn to other properties, which proves our bound.

Figure 39 and Figure 43 show nontrivial examples (i.e., $H \neq 0$ and $W_{\text{cw}} \neq 0$) for polygons in which the bound is exactly achieved. □

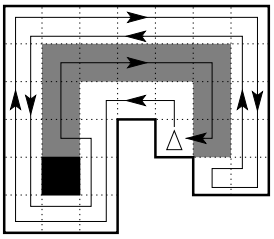


Figure 43: Polygon with $C = 34, \frac{E}{2} = 17, H = 1, W_{\text{cw}} = 2, S = 54 = C + \frac{1}{2}E + 3H + W_{\text{cw}} - 2$.

6 Summary

We considered the exploration of grid polygons. For simple polygons we have shown a lower bound of $\frac{7}{6}$ and presented a strategy, SmartDFS, that explores simple polygons with C cells and E edges using no more than $C + \frac{1}{2}E - 3$ steps from cell to cell. Using this upper bound, we were able to show that SmartDFS is in fact $\frac{4}{3}$-competitive, leaving a gap of only $\frac{1}{6}$ between the upper and the lower bound.

On the other hand, the competitive complexity for the exploration of grid polygons with holes is 2. A simple DFS exploration already achieves this competitive factor, but, nevertheless, DFS is not the best possible strategy, because is it not necessary to visit *each* cell twice. Therefore, we developed the strategy CellExplore that takes advantage of wider areas in the polygon, and thus corrects the weakness in the DFS strategy.

Interesting open questions are, how SmartDFS and CellExplore can be generalized to higher dimensions and other cell types than squares, e.g. triangular or hexagonal cells.

References

[1] S. Albers, K. Kursawe, and S. Schuierer. Exploring unknown environments with obstacles. *Algorithmica*, 32:123–143, 2002.

[2] S. Alpern and S. Gal. *The Theory of Search Games and Rendezvous.* Kluwer Academic Publications, 2003.

[3] E. M. Arkin, S. P. Fekete, and J. S. B. Mitchell. Approximation algorithms for lawn mowing and milling. Technical report, Mathematisches Institut, Universität zu Köln, 1997.

[4] S. Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In *Proc. 37th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 2–11, 1996.

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

[6] M. Betke, R. L. Rivest, and M. Singh. Piecemeal learning of an unknown environment. *Machine Learning*, 18(2–3):231–254, 1995.

[7] X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment I: The rectilinear case. *J. ACM*, 45(2):215–245, 1998.

[8] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *IEEE Computer*, 22(6):46–57, 1989.

[9] H. Everett. Hamiltonian paths in non-rectangular grid graphs. Report 86-1, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, 1986.

[10] A. Fiat and G. Woeginger, editors. *On-line Algorithms: The State of the Art*, volume 1442 of *Lecture Notes Comput. Sci.* Springer-Verlag, 1998.

[11] Y. Gabriely and E. Rimon. Competitive on-line coverage of grid environments by a mobile robot. *Comput. Geom. Theory Appl.*, 24:197–224, 2003.

[12] M. Grigni, E. Koutsoupias, and C. H. Papadimitriou. An approximation scheme for planar graph TSP. In *Proc. 36th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 640–645, 1995.

[13] U. Handel, C. Icking, T. Kamphans, E. Langetepe, and W. Meiswinkel. Gridrobot — an environment for simulating exploration strategies in unknown cellular areas. Java Applet, 2000. http://www.geometrylab.de/Gridrobot/.

[14] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. The polygon exploration problem. *SIAM J. Comput.*, 31:577–600, 2001.

[15] C. Hwan-Gue and A. Zelikovsky. Spanning closed trail and Hamiltonian cycle in grid graphs. In *Proc. 6th Annu. Internat. Sympos. Algorithms Comput.*, volume 1004 of *Lecture Notes Comput. Sci.*, pages 342–351. Springer-Verlag, 1995.

[16] C. Icking and R. Klein. Searching for the kernel of a polygon: A competitive strategy. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 258–266, 1995.

[17] C. Icking, R. Klein, and E. 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.

[18] C. Icking, R. Klein, E. Langetepe, S. Schuierer, and I. Semrau. An optimal competitive strategy for walking in streets. *SIAM J. Comput.*, 33:462–486, 2004.

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

[20] J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple new method for the geometric k-MST problem. In *Proc. 7th ACM-SIAM Sympos. Discrete Algorithms*, pages 402–408, 1996.

[21] J. S. B. Mitchell. Shortest paths and networks. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 24, pages 445–466. CRC Press LLC, Boca Raton, FL, 1997.

[22] J. S. B. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 633–701. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[23] H. P. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 116–121, 1985.

[24] S. Ntafos. Watchman routes under limited visibility. *Comput. Geom. Theory Appl.*, 1(3):149–170, 1992.

[25] J. O'Rourke. *Art Gallery Theorems and Algorithms*. The International Series of Monographs on Computer Science. Oxford University Press, New York, NY, 1987.

[26] N. S. V. Rao, S. Kareti, W. Shi, and S. S. Iyengar. Robot navigation in unknown terrains: introductory survey of non-heuristic algorithms. Technical Report ORNL/TM-12410, Oak Ridge National Laboratory, 1993.

[27] C. Umans and W. Lenhart. Hamiltonian cycles in solid grid graphs. In *Proc. 38th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 496–507, 1997.