Online Motion Planning MA-INF 1314
Graph-Exploration

Elmar Langetepe
University of Bonn

Repetition!

- Categories: Algorithms, Geometric Algorithms
- Difference: Online/Offline, Example SWR!
- Correctness, performance, struktural properties, proofs!
- Different models: grid-world/full visionl
- Labyrinth, Labyrinth with grid-structure!
- Shannon: Labyrinth with grid-structure 5×5
- Simple label algorithm
- Correctness: Formal proof
- Efficiency: Competitive Analysis!

Competitive analysis, competitive ratio

Definition 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 $K_{\text{OPT}}(P)$ 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 $P \in \Pi$

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

holds.

Rep.: Efficient Algorithm Graph-Exploration

- Explore a graph, visit all edges (and vertices)
- Vertex: All outgoing edges are visible
- Visited edges are visible
- Strategy: Online-DFS for edges, visits any edge twice

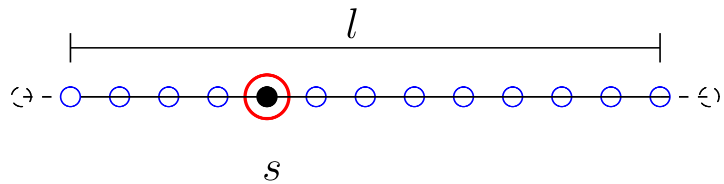
Theorem: Exploring an unknown graph requires roughly twice as many edge visits than the optimal exploration route for the known graph. DFS requires no more that twice as many edges.

Formal proof! Second part is already clear! Lower bound by worst-case adversary strategy

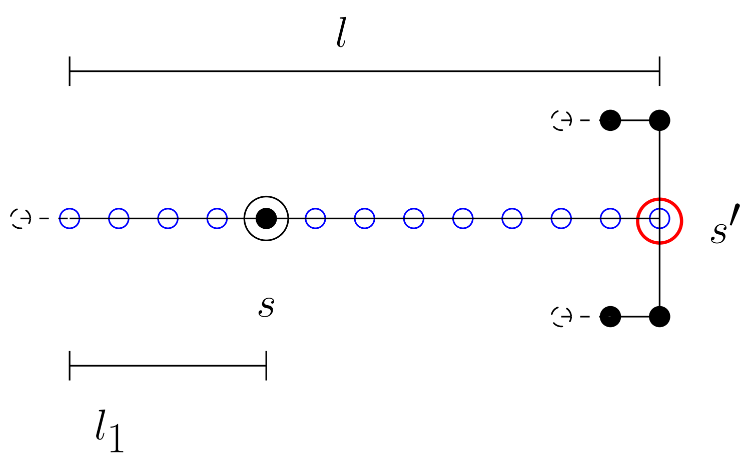
Graphexploration, Edge visits, Adversary

Adversary: $2 - \delta$ worse than the optimum

corridor, agent from s has explored l vertices



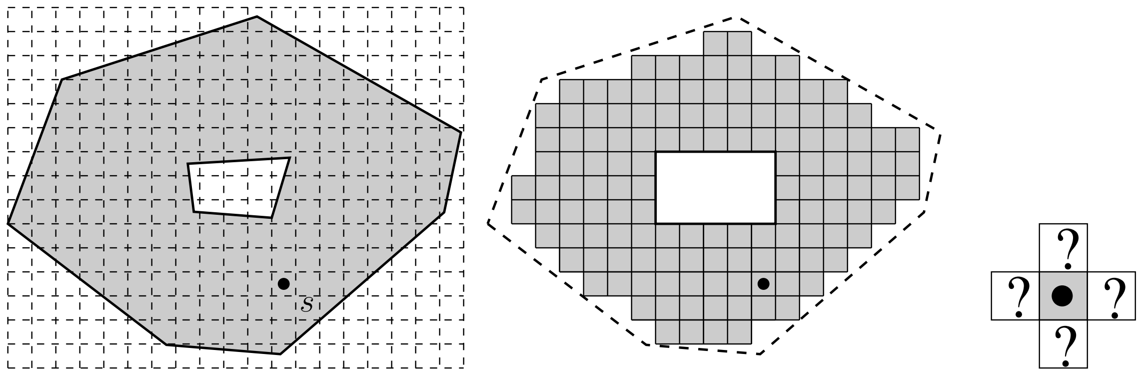
Now bifurcation at s'



and so on $\cdots$

Exploration simple grid polygons

- Formal definition, Environment and Agent



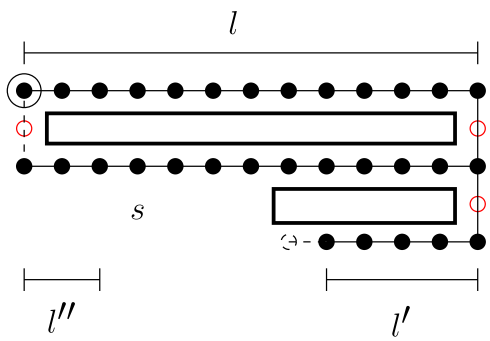
Def. 1.8:

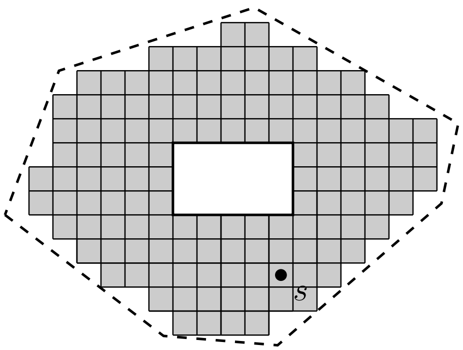
- cell c Tupel $(x, y) \in \mathbb{N}^2$.

- Cells $c_1 = (x_1, y_1), c_2 = (x_2, y_2)$ are adjacent, $:\Leftrightarrow |x_1 - x_2| + |y_1 - y_2| = 1$. For any cell c there are 4 adjacent cells.
- Two cells $c_1 = (x_1, y_1), c_2 = (x_2, y_2)$, $c_1 \neq c_2$ are diagonally adjacent, $:\Leftrightarrow |x_1 - x_2| \leq 1 \wedge |y_1 - y_2| \leq 1$. For any cell, 8 cells are diagonally adjacent.
- Path $\pi(s,t)$ from s to t is a sequence $s = c_1, \ldots, c_n = t$ so that c_i and c_{i+1} are adjacent.
- Gridpolygon P, Set of path-connectes cells, i.e. $\forall c_i, c_j \in P : \exists$ path $\pi(c_i, c_j)$, that runs in P.

Gridpolygons

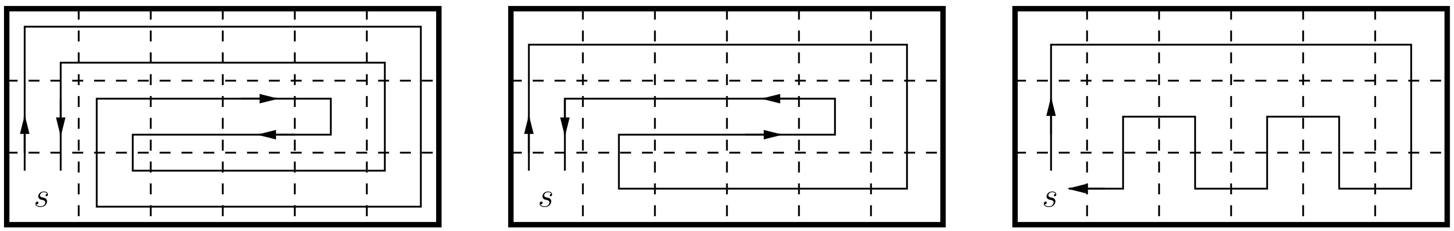
- Similar to Graph-exploration? Compare DFS, OPT!
- Gridpolygons: DFS for vertices, $2(C-1)$ steps!
- Lower Bound 2? Yes, but gridpolygon with holes
- *Simple* gridpolygons (without holes): Lower bound/strategy





Simple gridpolygons

- Gridpolygons without holes
- Simple improvement vs. DFS
- Example!



DFS | Verbesserung | Optimal

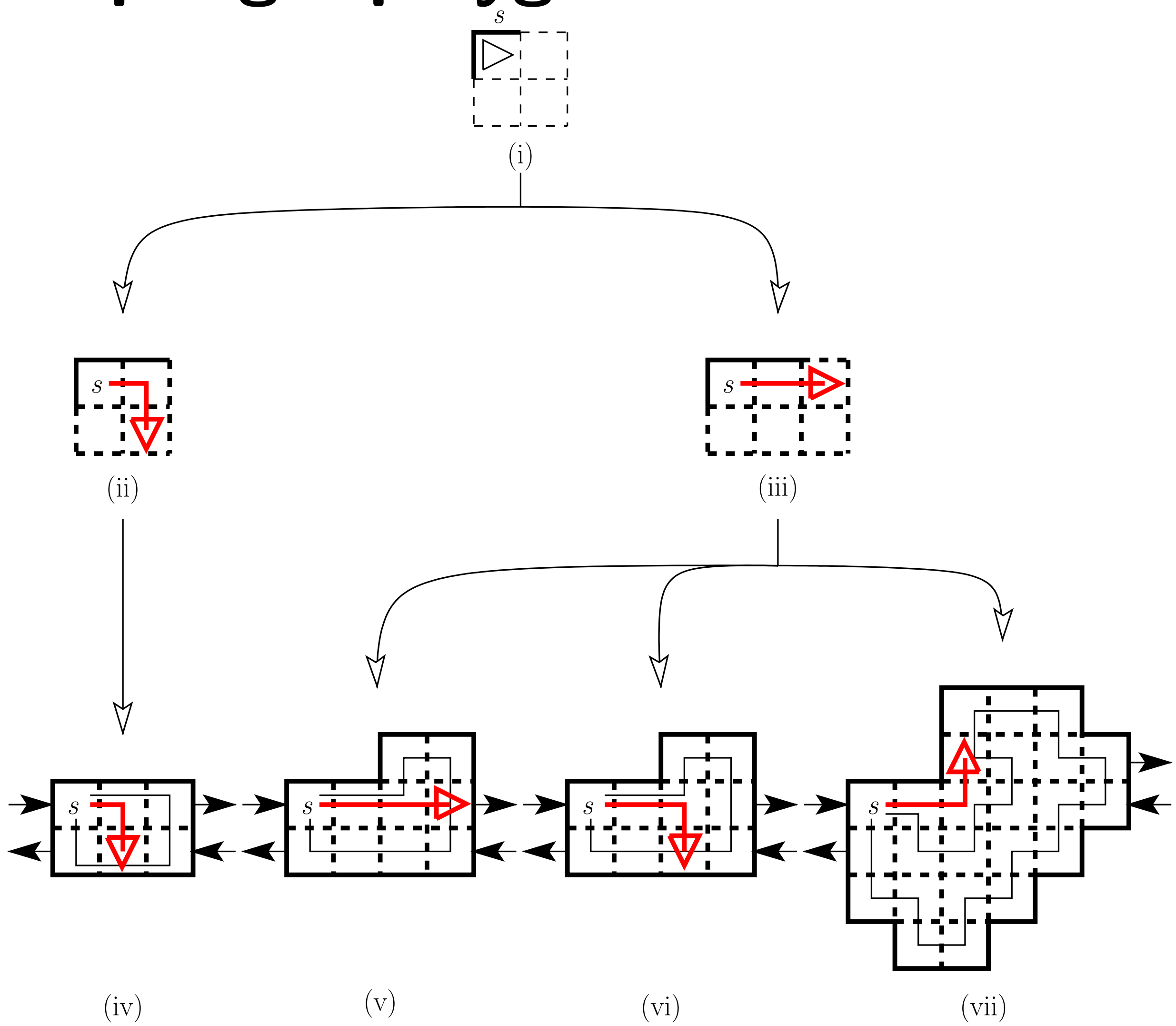
Simple gridpolygons: Lower bound!

Theorem: Any strategy for the exploration of simple gridpolygons with C cells requires at least $\frac{7}{6}C$ number of steps.

Proof: By adversary strategy

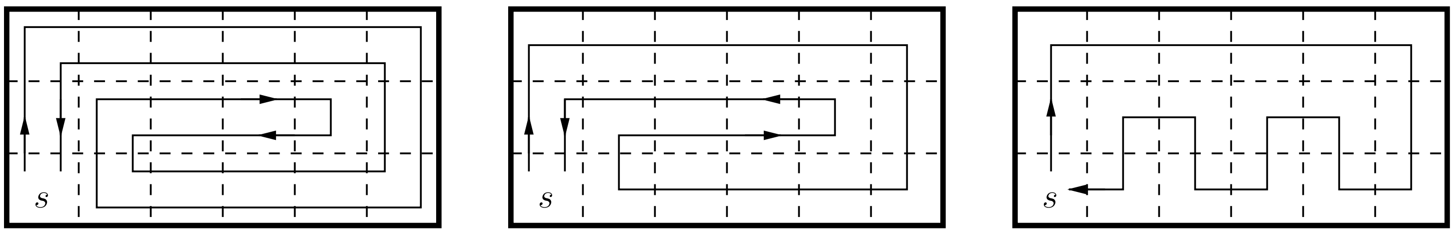
Not better than: $\frac{7}{6}$ competitive

Simple gridpolygons: Lower bound!



Simple gridpolygons: Improve DFS!

- *Fleshy* Environments: Better than 2?
- Visit only the vertices!
- Dependency from the boundary edges, E?
- Smart DFS!
- 1. Number of steps: $C + \frac{1}{2}E - 3$
- 2. $\frac{4}{3}$ kompetitiv



DFS Verbesserung Optimal

Formal description: DFS

DFS:

Choose Dir *dir*, so that reverse(*dir*) is boundary cell;
ExploreCell(*dir*);

ExploreCell(dir):

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

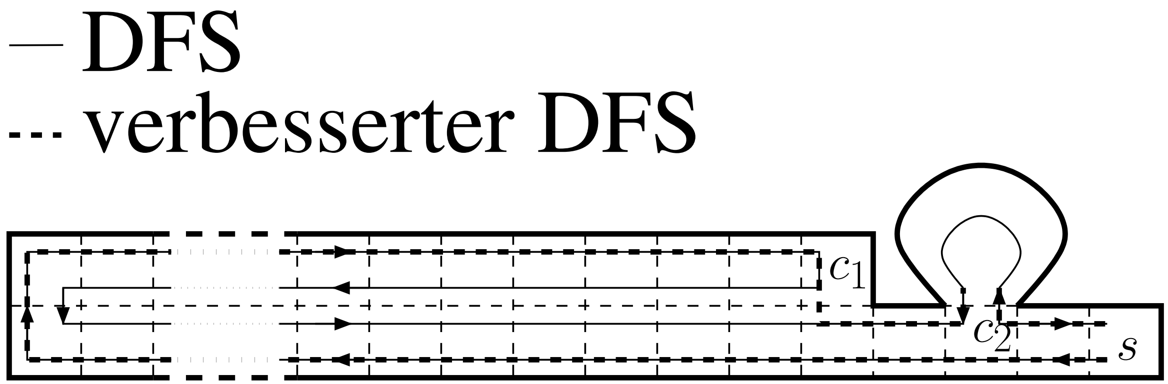
Formal description: DFS

ExploreStep(dir):

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

First Improvement for DFS

First idea: Move along the shortest path to the next *free* cell!



Smart DFS: 1. Improvement

DFS:

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

ExploreCell(dir):

base := aktuelle Position;
Left-Hand-Rule DFS:
ExploreStep(*base*, ccw(*dir*));
ExploreStep(*base*, *dir*);
ExploreStep(*base*, cw(*dir*));

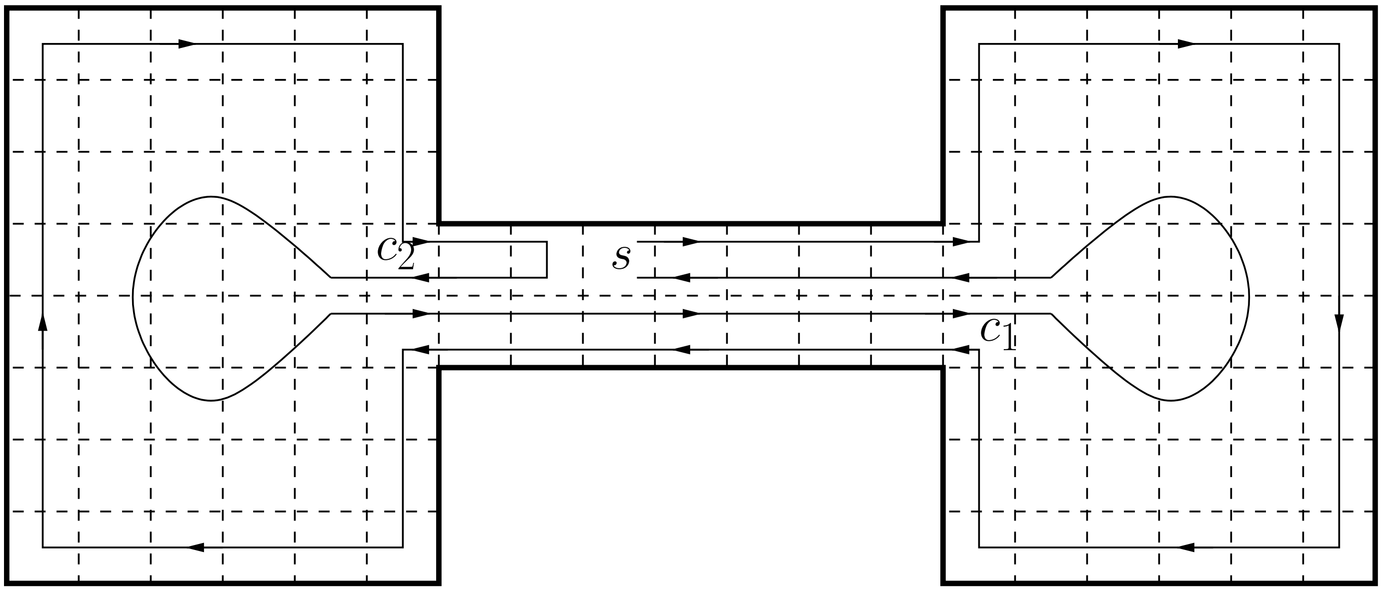
Smart DFS: 1. Improvement

ExploreStep(*base*, *dir*):

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

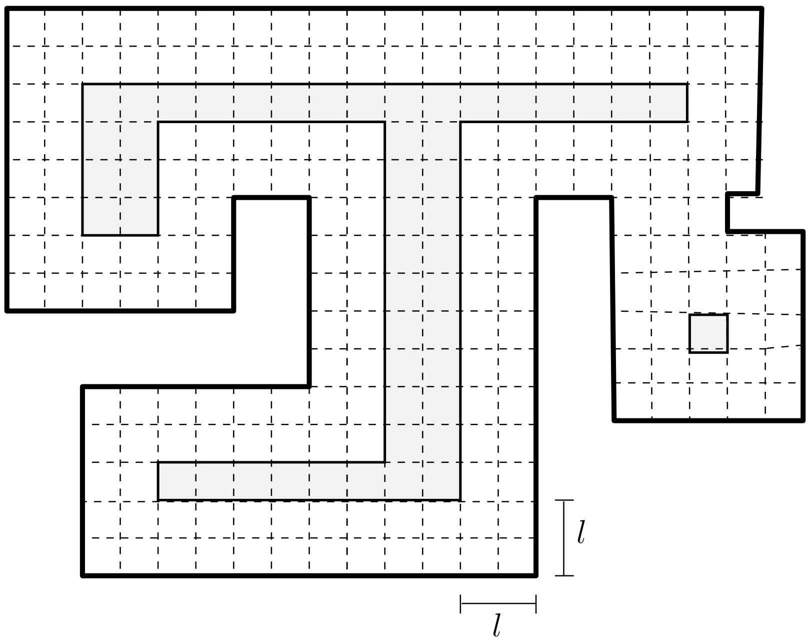
Smart DFS: 2.Improvement!

Second idea: Split into different areas happens: Work on the part where the starting point is not inside! Farther away!



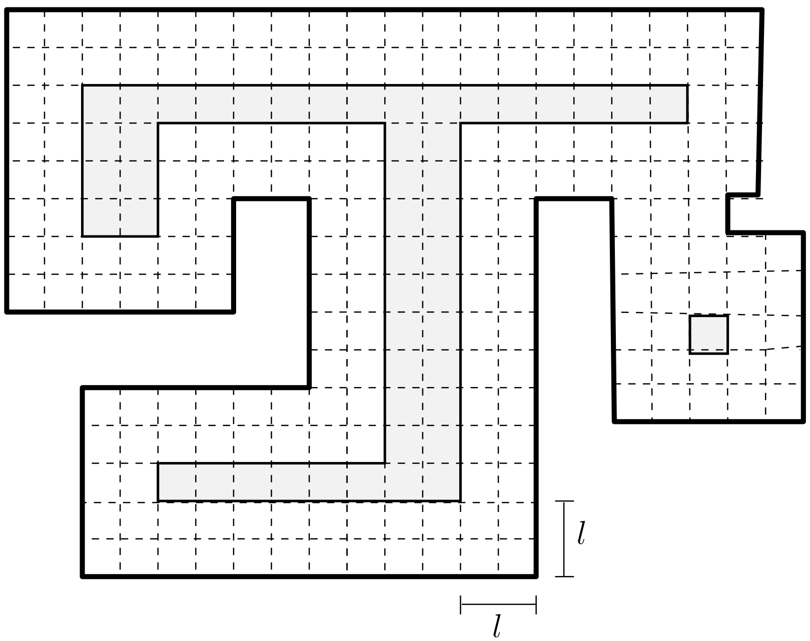
Definition Offset and Layer

- l-Layer and l-Offset of P defined recursively
- Cells along the boundary of P: 1-Layer
- P' after removing 1-Layers: 1-Offset
- Cells along the boundary of the 1-Offsets: 2-Layer
- 1-Offset after removing of 2-Layers: 2-Offset
- Go on recursively



l-Offset

- Need not be connected!
- Also defined for general gridpolygons
- Independent from any strategy



Smart DFS: 2. Improvement!

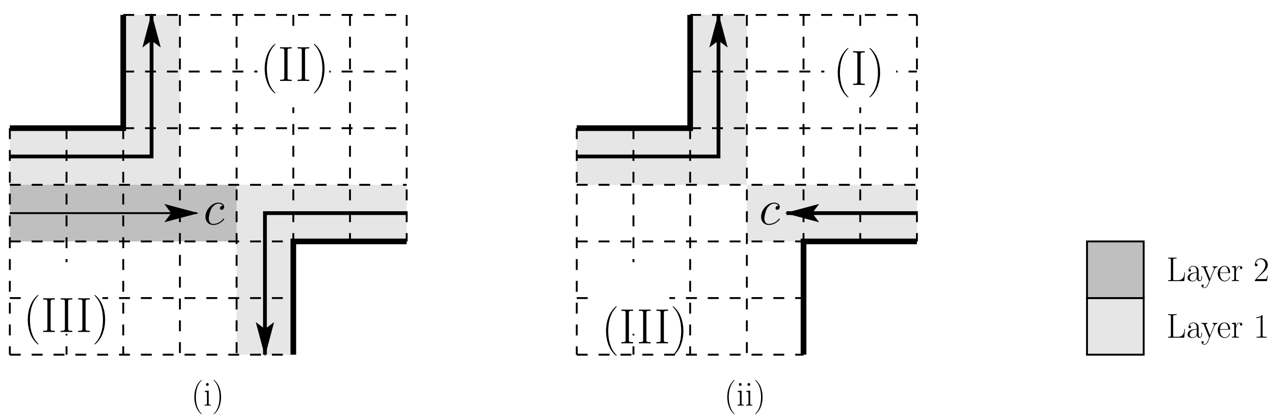
- Split into different part! When does it happen?
- Splitcell occurs in Layer l: How to proceed?
- Where is the starting point?

(I) Component K_i *fully* enclosed by Layer l.

(II) Component K_i *not* visited by Layer l

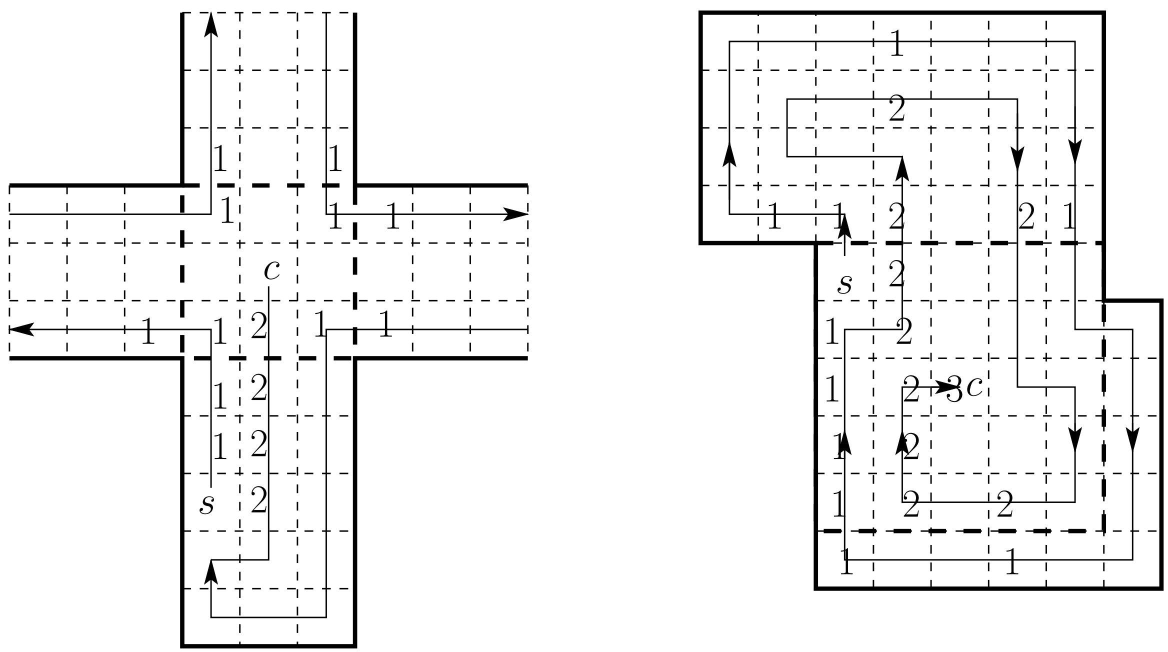
(III) Component K_i *partially* enclosed by Layer l.

- Visit component of type (III) last! Starting point!



Smart DFS: 2. Improvement!

- Special cases: There is no component of Typ (III)
- One step: Right Hand Rule!



(i) (ii)

Smart DFS: 2. Improvement!

ExploreCell(dir):

Mark current cell with the layer number;
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 preference:
  Calculate the type of components by the layer numbers of cells
  if There is no component of type (III) **then**
    Do one step by Right-Hand-Rule;
  else
    Visit the component of type (III) zuletzt.
  end if
end if

Smart DFS

- Strategy is well-defined!
- Next: Analysis of the strategy!
- Number of steps: $C + \frac{1}{2}E - 3$
- Attention: This is not a komp. Ratio!
- Better than DFS in fleshy environments, case sensitive
- Analysis over the split cells, recursion!

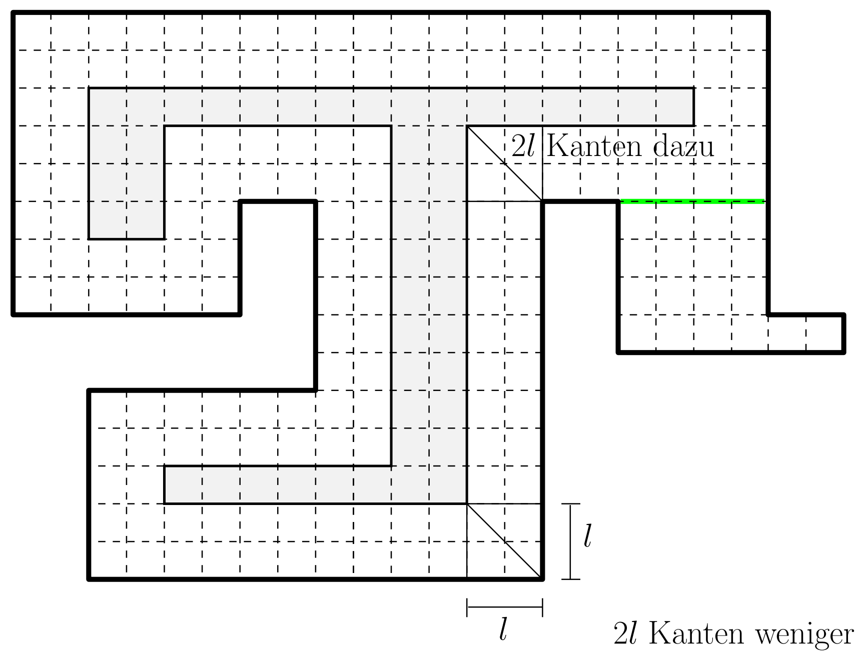
Edgelemma!

Lemma: The l-Offset of a simple gridpolygon P has *at least* $8l$ edges less than P.

$8l$ edges less

Proof: Surround the l-Offset in CW order

- Assume: Remains connected
- Left curve: l-Offset wins $2l$ edges.
- Right curve: l-Offset looses $2l$ edges.
- Altogether 4 more right curves than left curves (Turning angle 2π!)
- Disconnection improves the result
- l-Offset has at least $8l$ edges less



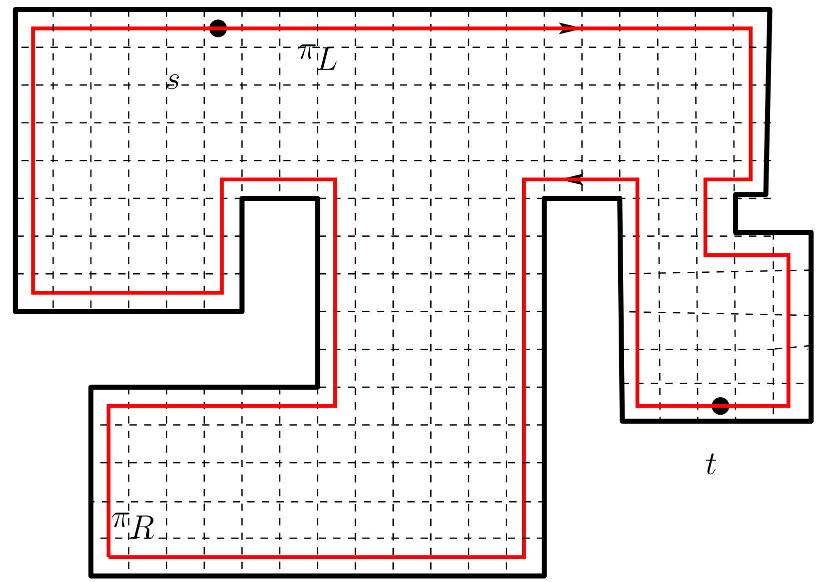
Distancelemma!

Lemma: The shortest path between to cells s and t of a simple gridpolygon P with $E(P)$edges has at most $\frac{1}{2}E(P) - 2$ steps.

Distancelemma! $\pi \leq \frac{1}{2}E(P) - 2$

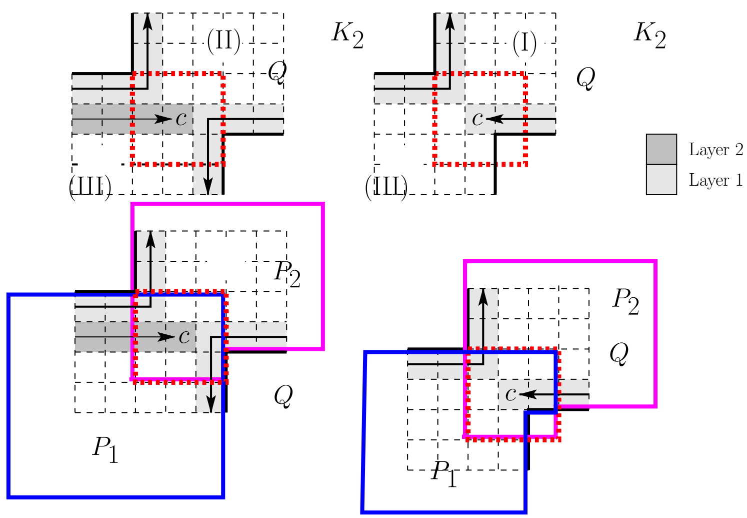
Beweis:

- s and t in 1-Layer, otherwise move them to the boundary
- Along the boundary (left) π_L, (right) π_R
- Roundtrip: Count edges!
- Roundtrip: At least 4 edges more than cells/steps
- Let π be shortest patp
- $|\pi_L| + |\pi_R| = E(P) - 4 \Rightarrow \pi \leq \frac{1}{2}E(P) - 2$



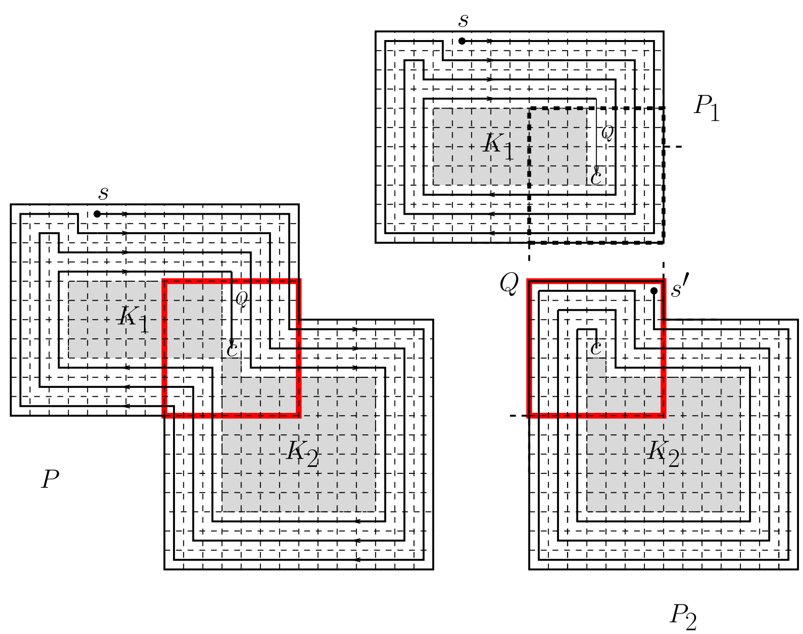
Decomposition of P at split-cell

- Decomposition Rectangle Q: $2q + 1$
- Cases: K_2 of type I) $(q = l)$ or vom type II) $(q = l - 1)$
- P_2, such that $K_2 \cup \{c\}$ is the q-Offset of P_2
- $P_1 := ((P \backslash P_2) \cup Q) \cap P$ Intersection with P for the movements



Decomposition of P

- Decomposition Rectangle Q: $2q+1$
- P_2, such that $K_2 \cup \{c\}$ is the q-Offset of P_2
- $P_1 := ((P \backslash P_2) \cup Q) \cap P$
- Path remains guilty!

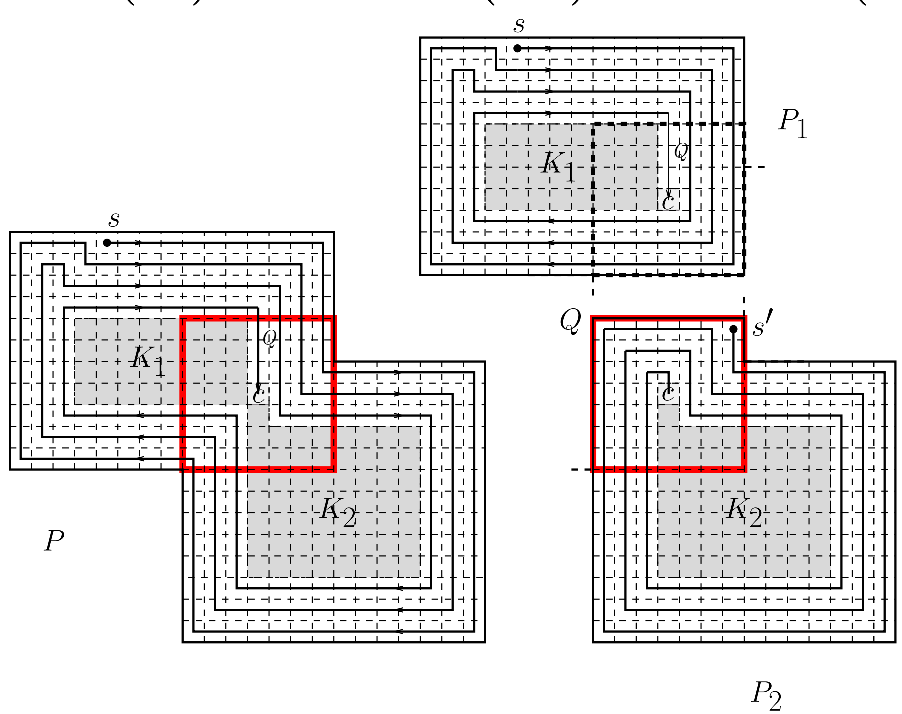


Analysis: Visity beyond the cells

- Any cell is visited once
- Number of steps $S(P)$: Visit cells plus additional visits
- $S(P) := C(P) + \mathit{excess}(P)$
- Calculate $\mathit{excess}(P)$

Excesslemma

Lemma: P gridpolygon and c a split-cell, such that P splits into K_1 and K_2 (for the first time). Let K_2 be the component, that is visited first. We have: $\mathit{excess}(P) \leq \mathit{excess}(P_1) + \mathit{excess}(K_2 \cup \{c\}) + 1.$



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

- Explore $K_2 \cup \{c\}$ after c by SmartDFS, return to c
- Gives: max. *excess*$(K_2 \cup \{c\})$ since $P_2 \backslash (K_2 \cup \{c\})$ optimal
- c twice: plus 1
- Then move to P_1: Maximal *excess*(P_1)