

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, structural properties, proofs!
- Different models: grid-world/full vision!
- 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■

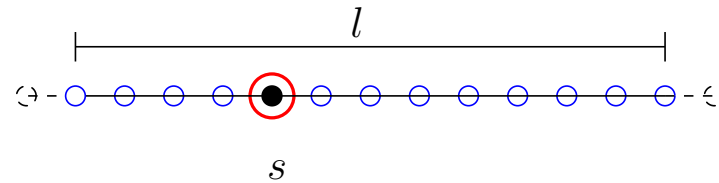
Theorem: Exploring an unknown graph requires roughly twice as many edge visits than the optimal exploration route for the known graph. DFS requires no more than twice as many edges. ■

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

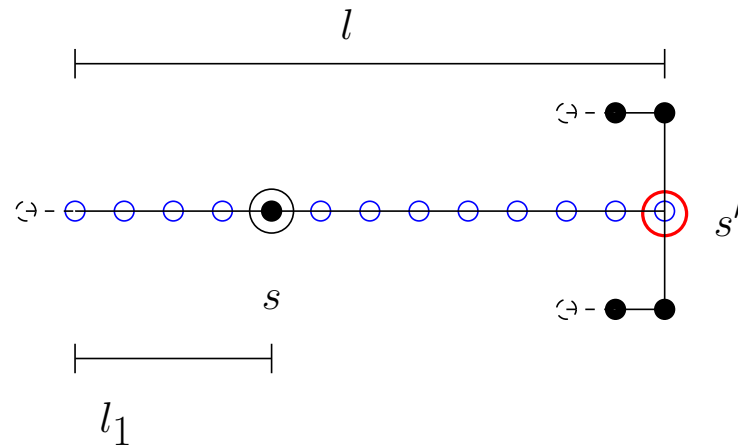
Graphexploration, Edge visits, Adversary

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

corridor, agent from s has explored l vertices

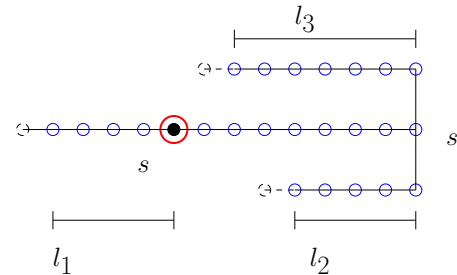


Now bifurcation at s'

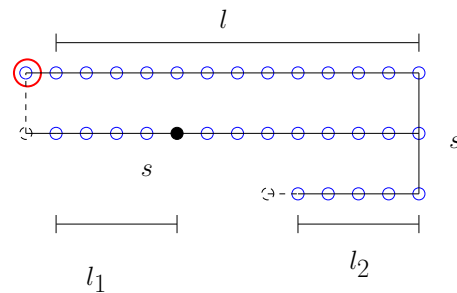


Graphexploration, Edge visits, Adversary

- Case 1: Agent returns to s



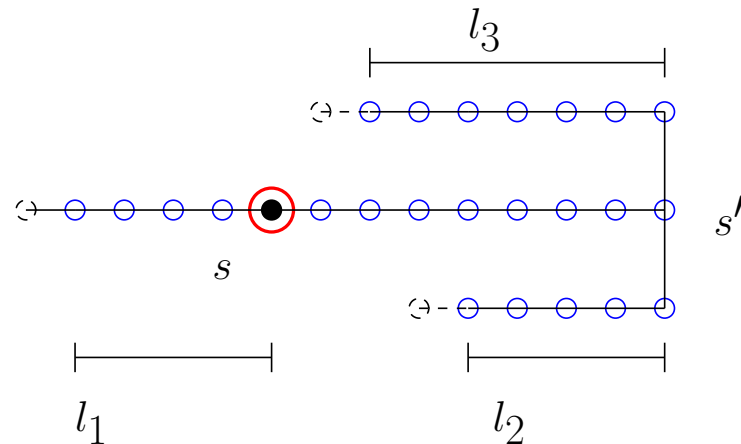
- Case 2: Agent reaches the end of the start corridor



- Close the corridors, present the scene

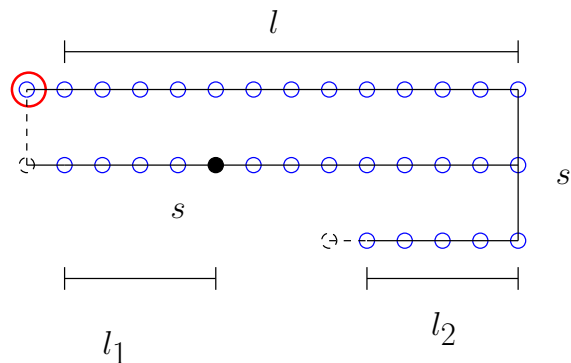
Adversary analysis: Case 1.

- Strategy S_{ROB} against S_{OPT} ■
- Case 1: Agent returns to s ■
- Up to now $|S_{\text{ROB}}| \geq 2l_1 + 2l_2 + 2l_3 + 2(l - l_1)$ ■ $= 2(l + l_2 + l_3)$ ■
- From now on: $2(l + l_2 + l_3) + 6$ ■
- Optimal: $|S_{\text{OPT}}| = 2(l + l_2 + l_3) + 6$ ■
- $|S_{\text{ROB}}| \geq (2 - \delta)|S_{\text{OPT}}|$ for $l \geq \lceil \frac{3}{\delta} \rceil$ ■



Adversary analysis: Case 2.

- Case 2: Agent reaches the end of start corridor
- Up to now: $|S_{\text{ROB}}| \geq 2l_1 + l - l_1 + 2l_2 + l + 1$
- From now on: $l + 1 + 2(l_2 + 1) + l - l_1$
- Sum: $|S_{\text{ROB}}| = 4(l + l_2) + 4$
- Optimal: $|S_{\text{OPT}}| = 2(l + 1 + l_2 + 1) = 2(l + l_2) + 4$
- $|S_{\text{ROB}}| \geq (2 - \delta)|S_{\text{OPT}}|$ für $l \geq \lceil \frac{3}{\delta} \rceil$



Remarks

Corollary : DFS for Online-Exploration of graphs is 2-competitive, there is no C -competitive strategy with $C < 2$.

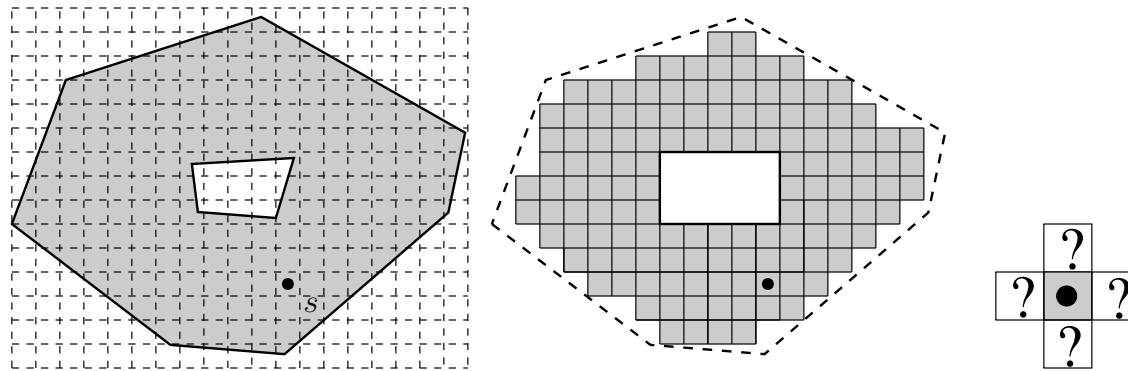
- DFS is optimal
- Additive constant: Start situation
- Example: Goal very close
- Lower bounds only with arbitrary large examples

Remarks!

- Return to the target■
- No return: **Exercise**■
- Examples: Opt. Tour visits any edge twice■
- Visit only the vertices, DFS for vertices? **Exercise!**■

Exploration simple grid polygons

- Formal definition, Environment and Agent



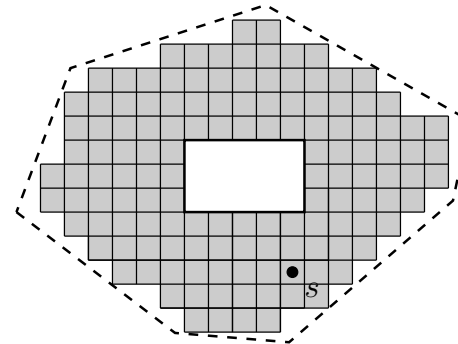
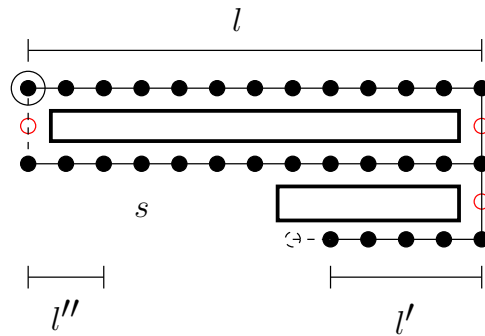
Def. 1.8:

- **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 $s = c_1, \dots, c_n = t$ so that c_i and c_{i+1} are adjacent.■
- **Gridpolygon** P , Set of path-connectes cells, i.e.
 $\forall c_i, c_j \in P : \exists \text{ path } \pi(c_i, c_j), \text{ that runs in } P.$ ■

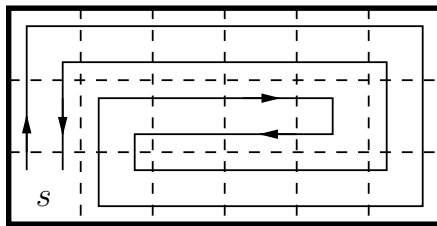
Gridpolygons

- 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

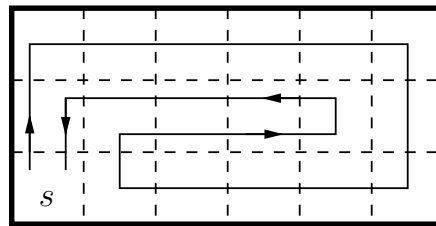


Simple gridpolygons

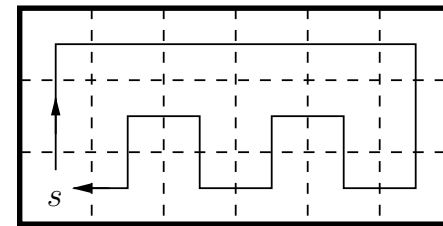
- Gridpolygons without holes
- Simple improvement vs. DFS
- Example!



DFS



Verbesserung



Optimal

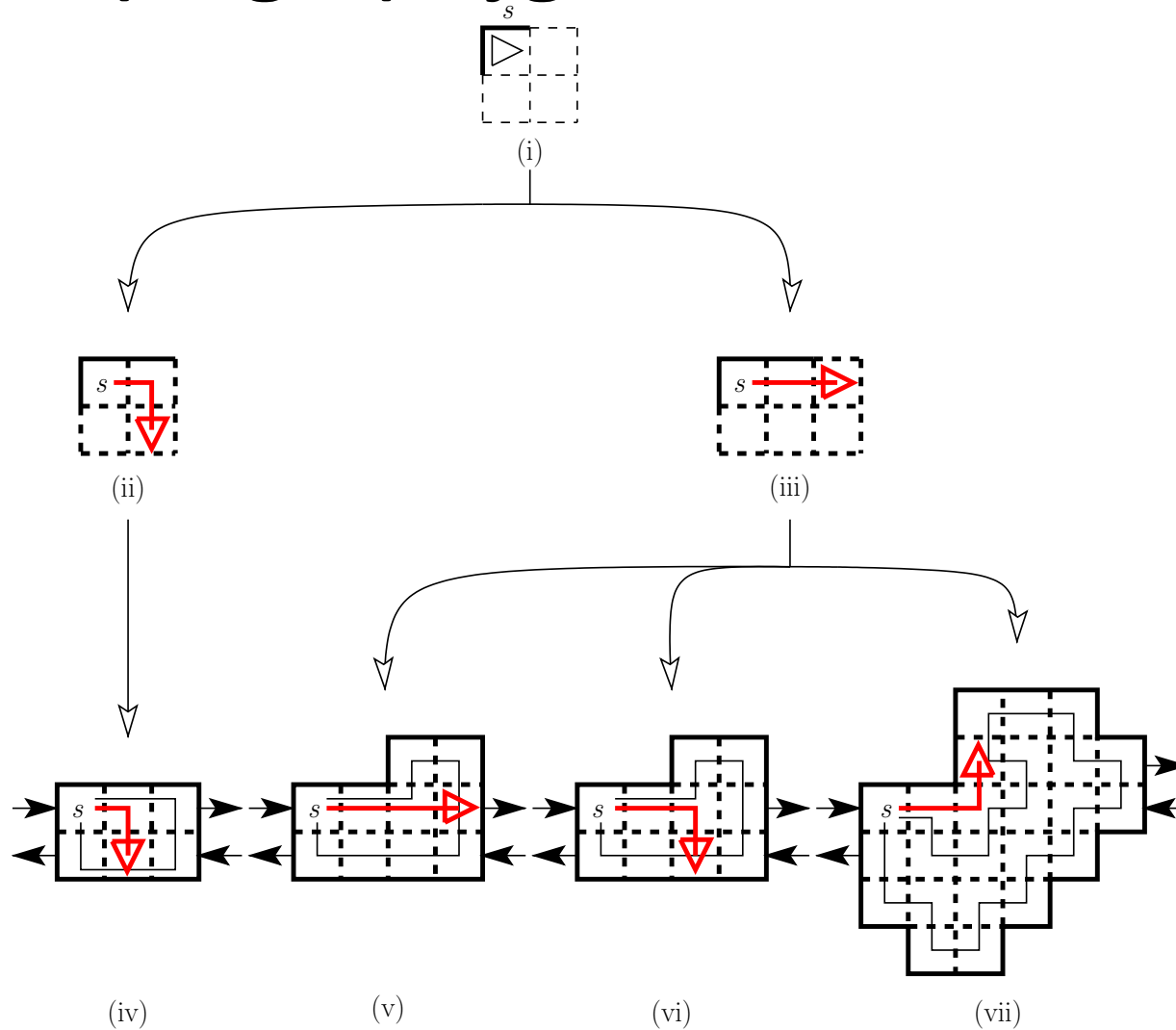
Simple gridpolygons: Lower bound!

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■

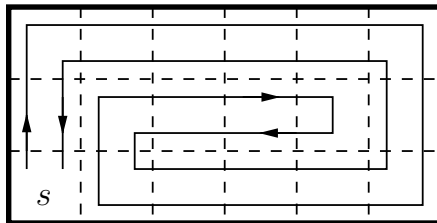
Not better than: $\frac{7}{6}$ competitive■

Simple gridpolygons: Lower bound!

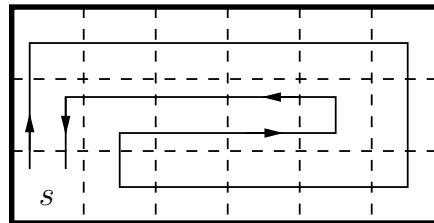


Simple gridpolygons: Improve DFS!

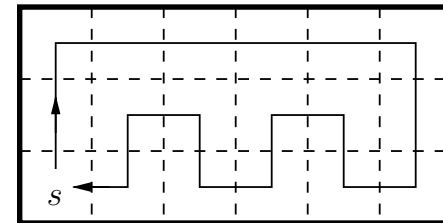
- *Fleshy* Environments: Better than 2? ■
- Visit only the vertices! ■
- Dependency from the boundary edges, E ? ■
- **Smart DFS!** ■
- 1. Number of steps: $C + \frac{1}{2}E - 3$ ■
- 2. $\frac{4}{3}$ kompetitiv ■



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

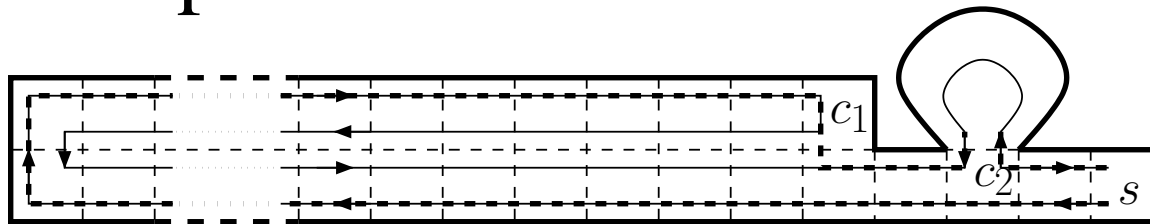
ExploreStep(*dir*):

```
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! ■

- - DFS
 - ... improved DFS



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;

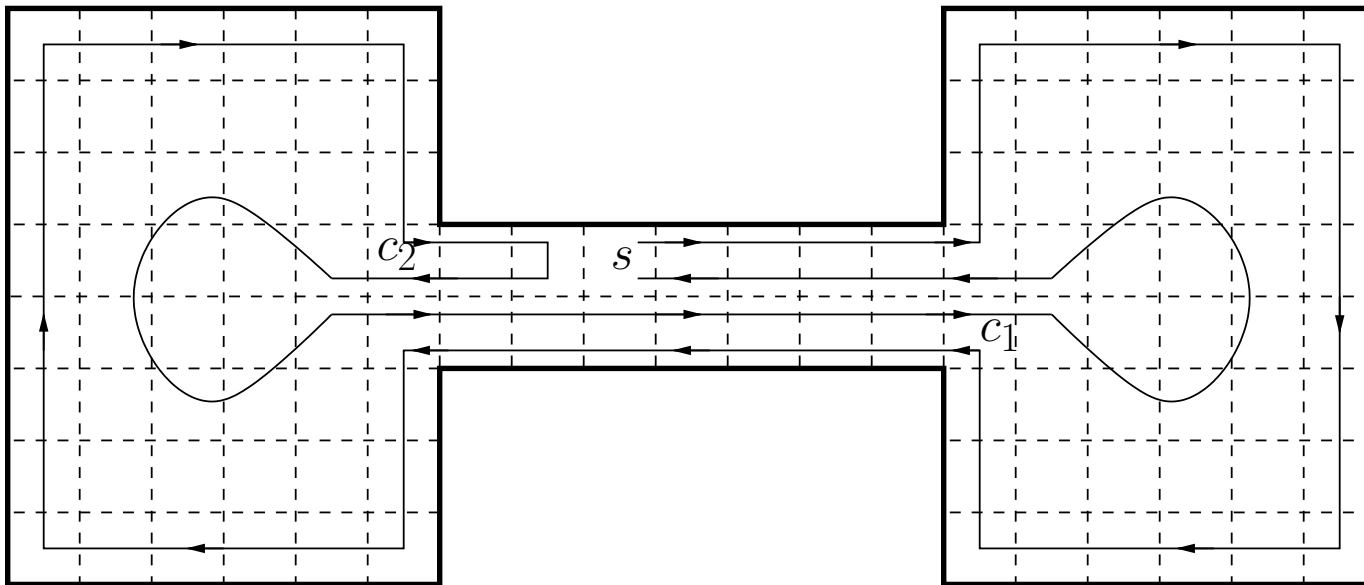
 move(*dir*);

 ExploreCell(*dir*);

end if

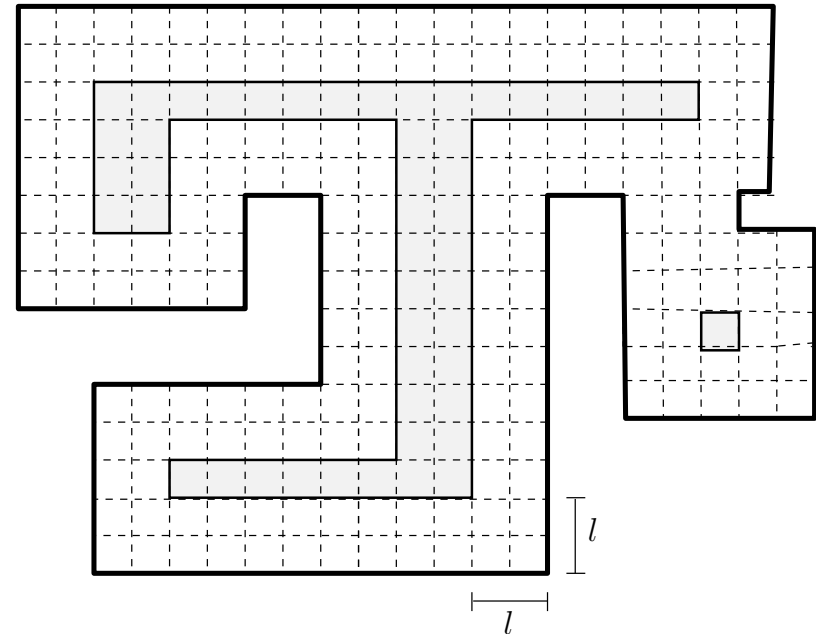
Smart DFS: 2.Improvement!

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



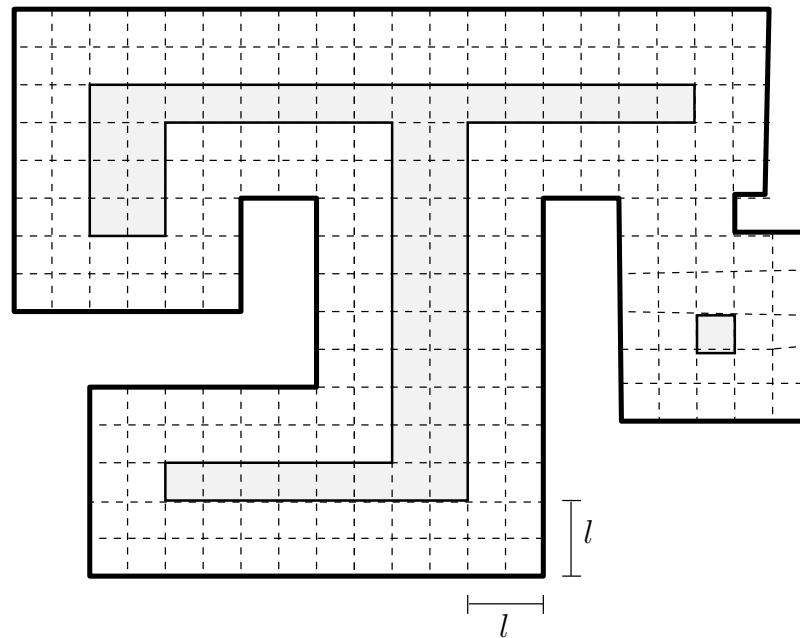
Definition Offset and Layer

- 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



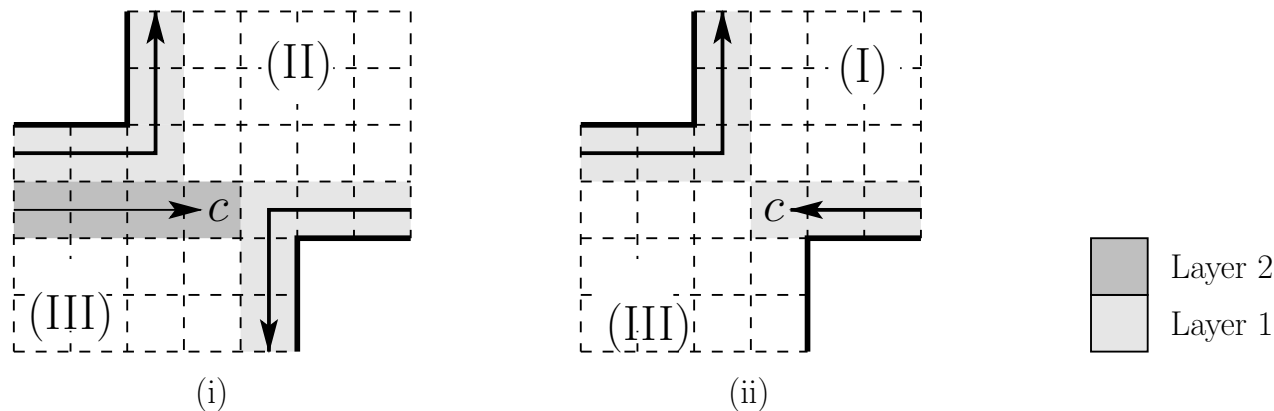
l -Offset

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



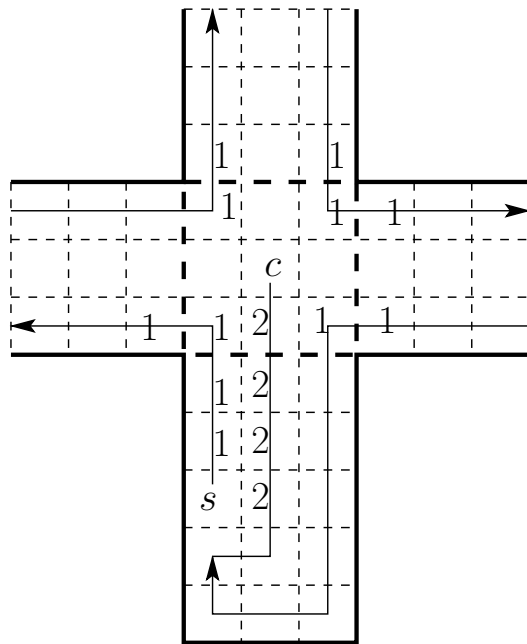
Smart DFS: 2. Improvement!

- 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! ■

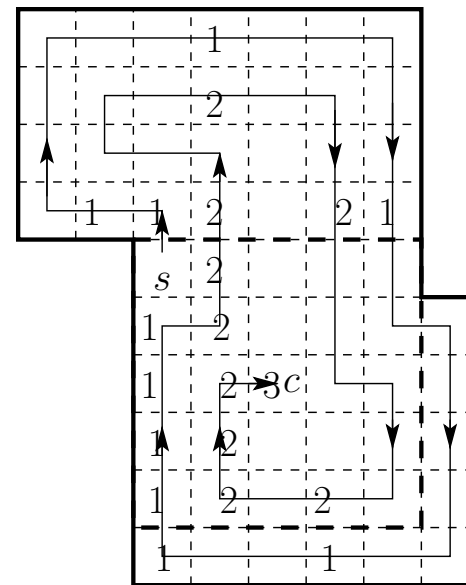


Smart DFS: 2. Improvement!

- 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!■