Models and Algorithms for Online Exploration and Search

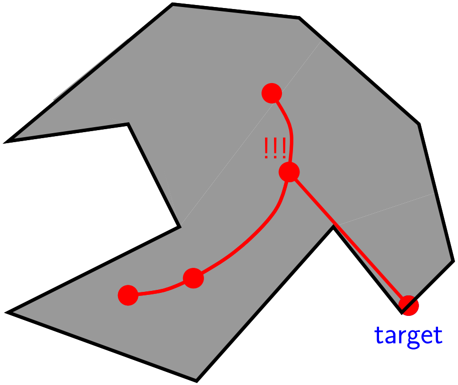
Tom Kamphans[1]

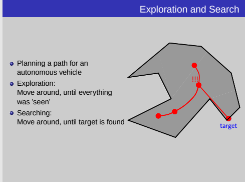
[1]University of Bonn, Computer Science I, Bonn, Germany.

April 04, 2006

Exploration and Search

- Planning a path for an autonomous vehicle
- Exploration:
  Move around, until everything was ’seen’
- Searching:
  Move around, until target is found





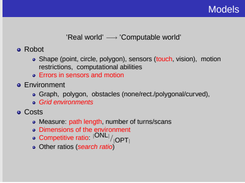
- Planning a path for an autonomous vehicle
- Exploration: Move around, until everything was 'seen'
- Searching: Move around, until target is found

First, I'd like to give a short introduction to the topics of this talk. *Exploration* and *search* are two problems that occur, for example, in robotics.
In both problems, we want to find a path for a mobile robot.
In the exploration problem, we want the robot to move around, until the whole environment is seen. In this example, the ROBOT (the red dot) is located inside THIS polygon. From its current position, it sees only THIS gray shaded region. Now, the robot has to move, for example like this, to see the other parts of the environment.
In the search problem, we move around, until we see a TARGET POINT, and then, we move directly towards the target.

Models

'Real world' $\longrightarrow$ 'Computable world'

- Robot
  - Shape (point, circle, polygon), sensors (touch, vision), motion restrictions, computational abilities
  - Errors in sensors and motion
- Environment
  - Graph, polygon, obstacles (none/rect./polygonal/curved),
  - *Grid environments*
- Costs
  - Measure: path length, number of turns/scans
  - Dimensions of the environment
  - Competitive ratio: $^{|\mathrm{ONL}|}/_{|\mathrm{OPT}|}$
  - Other ratios (*search ratio*)



Models

'Real world' $\longrightarrow$ 'Computable world'

- Robot
  - Shape (point, circle, polygon), sensors (touch, vision), motion restrictions, computational abilities
  - Errors in sensors and motion
- Environment
  - Graph, polygon, obstacles (none/rect./polygonal/curved),
  - *Grid environments*
- Costs
  - Measure: path length, number of turns/scans
  - Dimensions of the environment
  - Competitive ratio: $^{|ONL|}/_{|OPT|}$
  - Other ratios (*search ratio*)

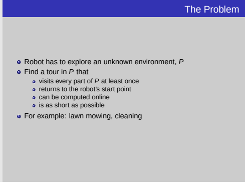
I guess, you all know what algorithms are good for, but maybe I should say some words about models. Well, we always use models to map the real world into a computable world. In our case, we have to model the robot itself: its shape and its sensors (is it blind or can it see points in a farther distance). There may be some motion restrictions (car-like robot); we have to account the robot's computitional abilities (map?).
For practical applications, it is also important to consider that no robot is error free.
Then, there are several possiblities to model the robot's environment. Very common is to move the robot in a graph or in a polygon; perhaps in the presence of obstacles of different complexities. But we may consider also special kinds of environments, for example, environments with a grid structure like a chessboard.
Last, we want to be able to compare different solutions for our problem, so we have to give a model for the costs. The first question is, what are the costs. Usually, we use length of the path travelled by the robot is as a cost measure, but we may use other measures like the number of turn or scans. We may give the cost depending on size of the environment or we may compare it to the optimal solution (if we are dealing with online algorithms). But we may use also other ratios: for example, the *search ratio*.

In the following, I will talk about the exploration of grid environments with a blind robot.

Further, we I will give some results on search ratios. In my thesis, I considered searching with error-prone robots, but I will omit these results here.

The Problem

- Robot has to explore an unknown environment, P
- Find a tour in P that
  - visits every part of P at least once
  - returns to the robot's start point
  - can be computed online
  - is as short as possible
- For example: lawn mowing, cleaning

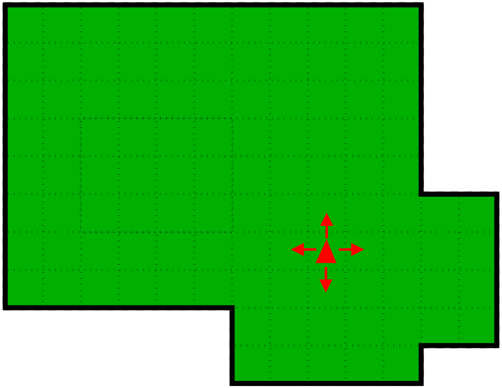

The Problem

- Robot has to explore an unknown environment, *P*
- Find a tour in *P* that
  - visits every part of *P* at least once
  - returns to the robot's start point
  - can be computed online
  - is as short as possible
- For example: lawn mowing, cleaning

We consider the problem of a robot that has to explore an environment that is unknown to the robot.

More precisely, we want to find a path that visits every part of the environment at least once and returns to the start point. The environment is unknown to the robot, so we want to compute our path online. We use the length of the robot's path as quality measure; thus, we want to keep the path as short as possible.

Environment and Robot

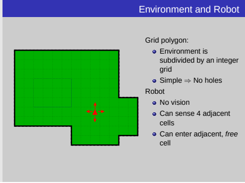


Grid polygon:

- Environment is subdivided by an integer grid
- Simple $\Rightarrow$ No holes

Robot

- No vision
- Can sense 4 adjacent cells
- Can enter adjacent, *free* cell



Environment and Robot

Grid polygon:

- Environment is subdivided by an integer grid
- Simple ⇒ No holes

Robot

- No vision
- Can sense 4 adjacent cells
- Can enter adjacent, *free* cell

We deal with a special kind of environments we call *grid polygons*, that is, we assume, that the environment is subdivided by an integer grid.
If there are no holes in the environment, we call the polygon *simple* polygon.

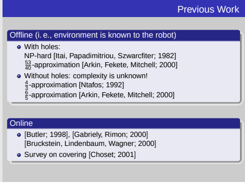
Our robot has no vision, but can sense the four cells that are adjacent to the current position. The robot can move from one cell to an adjacent cell that is part of the polygon.

Offline (i. e., environment is known to the robot)

- With holes:
  NP-hard [Itai, Papadimitriou, Szwarcfiter; 1982]
  $\frac{53}{40}$-approximation [Arkin, Fekete, Mitchell; 2000]
- Without holes: complexity is unknown!
  $\frac{4}{3}$-approximation [Ntafos; 1992]
  $\frac{6}{5}$-approximation [Arkin, Fekete, Mitchell; 2000]

Online

- [Butler; 1998], [Gabriely, Rimon; 2000]
  [Bruckstein, Lindenbaum, Wagner; 2000]
- Survey on covering [Choset; 2001]



Previous Work

Offline (i. e., environment is known to the robot)

- With holes:
  NP-hard [Itai, Papadimitriou, Szwarcfiter; 1982]
  $\frac{53}{40}$-approximation [Arkin, Fekete, Mitchell; 2000]
- Without holes: complexity is unknown!
  $\frac{4}{3}$-approximation [Ntafos; 1992]
  $\frac{6}{5}$-approximation [Arkin, Fekete, Mitchell; 2000]

Online

- [Butler; 1998], [Gabriely, Rimon; 2000]
  [Bruckstein, Lindenbaum, Wagner; 2000]
- Survey on covering [Choset; 2001]

It is known, that the offline case where the environment is known to the robot is NP-hard for polygons with holes, and there is an approximation by Arkin et al.

For polygons without holes it is not known, whether this problem is NP-complete. However, there are also some approximations.

For the online case, there are some works in a more practical context, for example by Butler, Gabriely and Rimon, or Bruckstein and colleages. Also, there is a nice survey on covering algorithms by Howie Choset.

A Lower Bound

Theorem

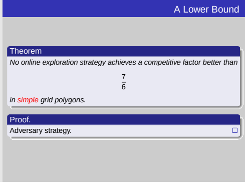
No online exploration strategy achieves a competitive factor better than

$$\frac{7}{6}$$

in simple grid polygons.

Proof.

Adversary strategy. $\square$


A Lower Bound

Theorem

No online exploration strategy achieves a competitive factor better than

$$\frac{7}{6}$$

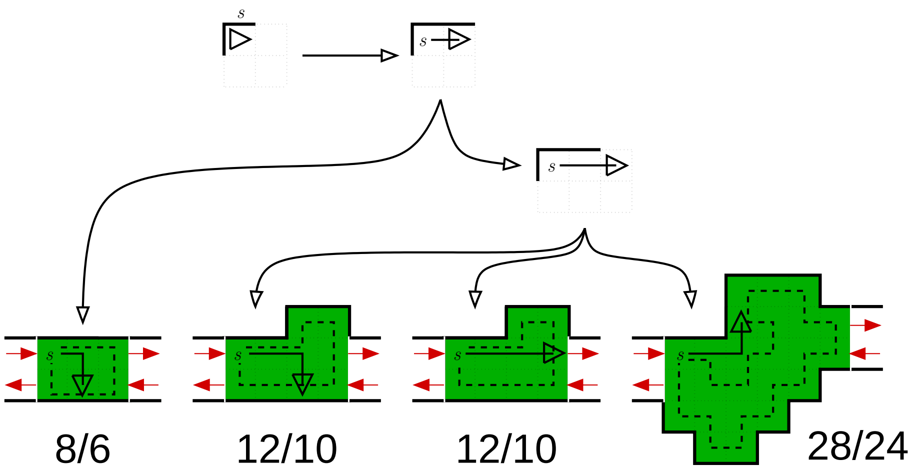
in simple grid polygons.

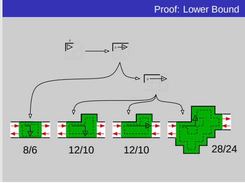
Proof.

Adversary strategy. □

First, let me give you a lower bound on the exploration problem in SIMPLE polygons: It can be shown that no online exploration strategy can achieve a factor better than 7/6. To show this, we give an adversary that that forces **every** online strategy to walk a path that is at least 7/6 times longer than the optimal path.

Proof: Lower Bound

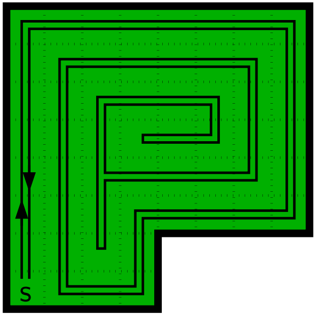




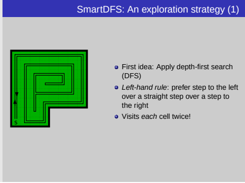
We assume, that the robot starts in a corner of the polygon and moves one step to the east. If the robot walks south, we can used a mirrored construction.
For the second step, the robot has two possibilities: It may walk to the south, or it may walk to the east.
In the first case, we close the polygon like this. Now, whatever the robot does, it needs at least 8 steps, whereas the optimal solution needs only 6 steps (the dotted lines show the optimal path).
In the second case, the robot has three possiblities: It can move S/E/N.
In the first two cases, we close the polygon like this, and get a ratio of $\frac{12}{10}$.
In the more interesting case, the robot continues to follow the wall and we close the polygon is this way. Now, the robot needs at least 28 steps, the optimal strategy needs 24 steps (wall following yields best case).

These blocks have limited size. To get a general lower bound, we have to construct polygons of arbitrary size. We can do this by repeating this construction: As soon as the robot leaves one block, it enters the start cell of the next block and the 'game' starts again.

SmartDFS: An exploration strategy (1)



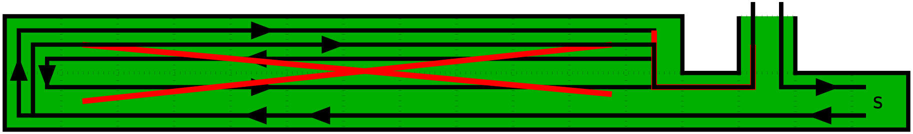
- First idea: Apply depth-first search (DFS)
- *Left-hand rule*: prefer step to the left over a straight step over a step to the right
- Visits *each* cell twice!


SmartDFS: An exploration strategy (1)

- First idea: Apply depth-first search (DFS)
- *Left-hand rule*: prefer step to the left over a straight step over a step to the right
- Visits *each* cell twice!

A first idea for the exploration is, to use a simple depth-first search. We use the *left-hand rule*, that is, we always keep the polygons boundary and the visited cells on the left side of the robot. Of course, DFS visits each cell twice.

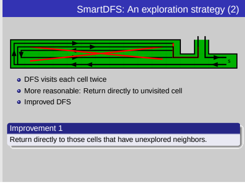
SmartDFS: An exploration strategy (2)



- DFS visits each cell twice
- More reasonable: Return directly to unvisited cell
- Improved DFS

Improvement 1

Return directly to those cells that have unexplored neighbors.



SmartDFS: An exploration strategy (2)

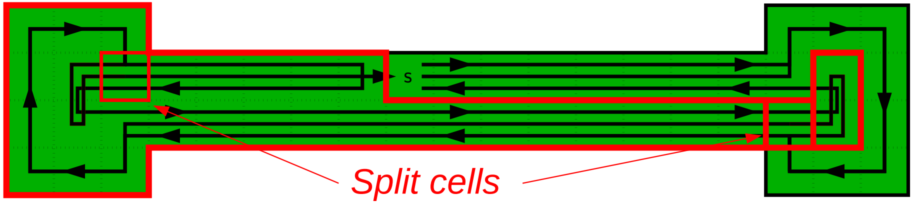
- DFS visits each cell twice
- More reasonable: Return directly to unvisited cell
- Improved DFS

Improvement 1
Return directly to those cells that have unexplored neighbors.

Visiting each cell twice is not very efficient, we can do better. In this example, DFS visits each cell in the long corridor twice. Of course, it is more reasonable to omit the second visit of the long corridor and walk directly to the unexplored cell $\otimes$ so we get a path like this.

So the first improvement to DFS is to return *directly* to those cells that have unexplored neighbors.

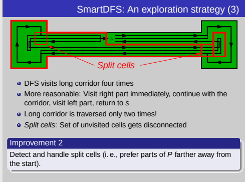
SmartDFS: An exploration strategy (3)



- DFS visits long corridor four times
- More reasonable: Visit right part immediately, continue with the corridor, visit left part, return to *s*
- Long corridor is traversed only two times!
- *Split cells*: Set of unvisited cells gets disconnected

Improvement 2

Detect and handle split cells (i. e., prefer parts of *P* farther away from the start).



SmartDFS: An exploration strategy (3)

- DFS visits long corridor four times
- More reasonable: Visit right part immediately, continue with the corridor, visit left part, return to *s*
- Long corridor is traversed only two times!
- *Split cells*: Set of unvisited cells gets disconnected

Improvement 2

Detect and handle split cells (i. e., prefer parts of *P* farther away from the start).

But we still can do better. In this case, even the improved version of DFS visits the long corridor in the middle four times.
It is more reasonable to explore the polygon up to this cell. Now, we do not follow the left-hand rule, but explore the right part first. Then, we continue with the corridor, visit the left part and return to the start. Now, we visit the corridor in the middle only two times!
We call cells where we do not follow the LHR *split cells*, because they have the property, that the unvisited cells split in two components after the cell is visited, as you can see in this example. Now, we have ONE block of unexplored cells, but as we enter this cell, we have one block on the right and one block on the left side.

Thus, the second improvement to DFS is to detect split and handle split cells. Basically, we want to deal with components that are farther away from the start first.

Performance of SmartDFS

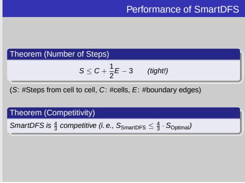
Theorem (Number of Steps)

$$S \leq C + \frac{1}{2}E - 3 \qquad \textit{(tight!)}$$

(S: #Steps from cell to cell, C: #cells, E: #boundary edges)

Theorem (Competitivity)

SmartDFS is $\frac{4}{3}$ competitive (i. e., $S_{\text{SmartDFS}} \leq \frac{4}{3} \cdot S_{\text{Optimal}}$)


Performance of SmartDFS

Theorem (Number of Steps)

$$S \le C + \frac{1}{2}E - 3 \qquad \textit{(tight!)}$$

(S: #Steps from cell to cell, C: #cells, E: #boundary edges)

Theorem (Competitivity)

SmartDFS is $\frac{4}{3}$ competitive (i. e., $S_{\text{SmartDFS}} \le \frac{4}{3} \cdot S_{\text{Optimal}}$)

We can give two performance results for our exploration strategy: First, the number of steps from cell to cell —and, in turn, the length of the exploration path—is bound by the number of cells plus half the number of edges minus three. And this bound is tight!

Using this upper bound on the number of steps, we can further show that SmartDFS is competitive with a factor of 4/3; that is, the path generated by our strategy is never longer than $\frac{4}{3}$ times the optimal solution.

Java Applet

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

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

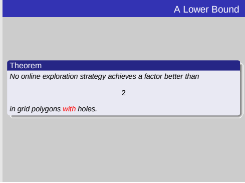
A Lower Bound

Theorem

No online exploration strategy achieves a factor better than

$$2$$

in grid polygons with holes.


A Lower Bound

Theorem

No online exploration strategy achieves a factor better than

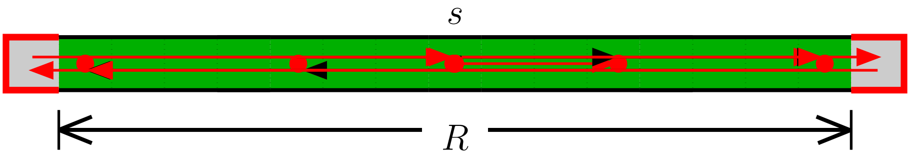
$$2$$

in grid polygons with holes.

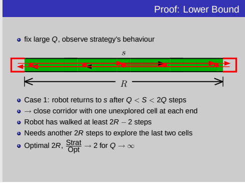
We can show that the exploration problem for polygons *with* holes is harder than for polygons *without* holes; that is, we have a lower bound of 2.

Proof: Lower Bound

- fix large Q, observe strategy's behaviour

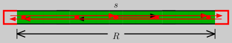


- Case 1: robot returns to s after $Q < S < 2Q$ steps
- $\rightarrow$ close corridor with one unexplored cell at each end
- Robot has walked at least $2R - 2$ steps
- Needs another $2R$ steps to explore the last two cells
- Optimal $2R$, $\frac{\text{Strat}}{\text{Opt}} \rightarrow 2$ for $Q \rightarrow \infty$



Proof: Lower Bound

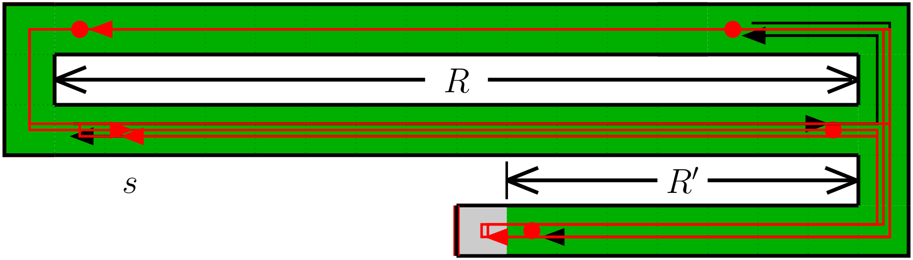
- fix large Q, observe strategy's behaviour



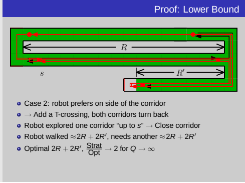
- Case 1: robot returns to s after $Q < S < 2Q$ steps
- $\rightarrow$ close corridor with one unexplored cell at each end
- Robot has walked at least $2R - 2$ steps
- Needs another $2R$ steps to explore the last two cells
- Optimal $2R$, $\frac{\text{Strat}}{\text{Opt}} \rightarrow 2$ for $Q \rightarrow \infty$

To show the lower bound, we fix a large number, Q, and observe the behaviour of the strategy in a corridor. The strategy may visit the left and the right parts of the corridor alternately, and in our first case, the robot returns back to the start after walking at least Q steps and at most $2Q$ steps. Now, we close the corridor with one unexplored cell on each side. So far, the robot has walked $2R - 2$ steps and needs another $2R$ steps to visit the last cells. The optimal strategy needs only $2R$ steps, and the ratio goes to 2 when Q goes to infinity. But how do we proceed, if the robot does NOT return to s?

Proof: Lower Bound



- Case 2: robot prefers on side of the corridor
- $\rightarrow$ Add a T-crossing, both corridors turn back
- Robot explored one corridor “up to s” $\rightarrow$ Close corridor
- Robot walked $\approx 2R + 2R'$, needs another $\approx 2R + 2R'$
- Optimal $2R + 2R'$, $\frac{\text{Strat}}{\text{Opt}} \rightarrow 2$ for $Q \rightarrow \infty$



Proof: Lower Bound

- Case 2: robot prefers on side of the corridor
- $\rightarrow$ Add a T-crossing, both corridors turn back
- Robot explored one corridor "up to s" $\rightarrow$ Close corridor
- Robot walked $\approx 2R + 2R'$, needs another $\approx 2R + 2R'$
- Optimal $2R + 2R'$, $\frac{\text{Strat}}{\text{Opt}} \rightarrow 2$ for $Q \rightarrow \infty$

In this case, the robot prefers one side of the corridor. After $2Q$ steps, we add a T-crossing to the corridor. Now, the robot may explore the new corridors alternately. Both corridors turn back and lead parallel along the first corridor.
Eventually, one of the corridors will be explored in the same length as the first corridor. Now, we close the polygon with a loop connecting THESE two corridors and one unexplored cell in the third corridor.
The robot has walked roughly $2R + 2R'$—apart from the vertical steps—and needs the same number to explore the last cell and go back to the start. The optimal strategy needs only $2R + 2R'$, so, again, the ratio goes to 2 for Q to infinity.

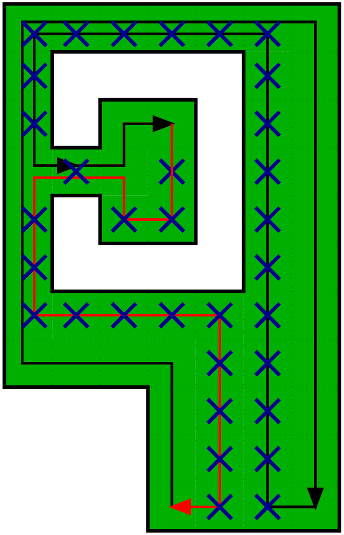
Strategy CellExplore

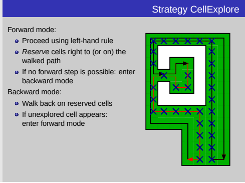
Forward mode:

- Proceed using left-hand rule
- *Reserve* cells right to (or on) the walked path
- If no forward step is possible: enter backward mode

Backward mode:

- Walk back on reserved cells
- If unexplored cell appears: enter forward mode



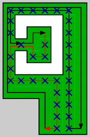


Forward mode:

- Proceed using left-hand rule
- *Reserve* cells right to (or on) the walked path
- If no forward step is possible: enter backward mode

Backward mode:

- Walk back on reserved cells
- If unexplored cell appears: enter forward mode



Our exploration strategy operates in two modes. In the forward mode we move following the left-hand rule while reserving cells for the path back. In this example, cells marked with a cross will be used for the return path. We proceed, until there is no free, unreserved cell adjacent to the robot's position, and then we switch to the backward mode.

In the backward mode, we simply walk back using the reserved cells. If we encounter unexplored parts of the parts of the polygon, we switch back to the forward mode, explore the new part recursively and then continue with the path back.

Performance of CellExplore

Theorem (Number of Steps)

CellExplore needs at most

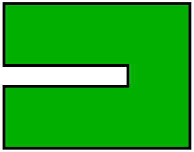
$$C + \frac{1}{2}E + 3H + W - 2$$

steps to explore a polygon. This bound is tight.

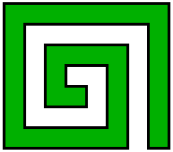
(C: #cells, E: #boundary edges, H: #holes, W: "sinuosity")

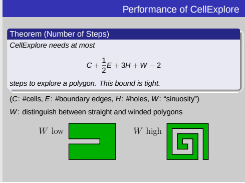
W: distinguish between straight and winded polygons

W low



W high





Theorem (Number of Steps)

CellExplore needs at most

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

steps to explore a polygon. This bound is tight.

(C: #cells, E: #boundary edges, H: #holes, W: “sinuosity”)

W: distinguish between straight and winded polygons

We can show, that the number of steps needed by our strategy is again bounded by the number of cells plus half the number of edges. Additionally, we need at most three steps for every obstacle, and, unfortunately, we need another parameter, W, which is used to diffentiate between straight and winded polygons such as spirals with many turns.

Java Applet

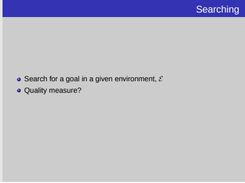
`http://www.geometrylab.de/Gridrobot/`

Java Applet

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

Searching

- Search for a goal in a given environment, $\mathcal{E}$
- Quality measure?



- Search for a goal in a given environment, $\mathcal{E}$
- Quality measure?

Now let’s consider the search for a goal. We want to find an appropriate quality measure for search tasks.

Quality measure

- *Competitive ratio* for a strategy, $\mathcal{S}$:

$$C := \sup_{\mathcal{E}} \sup_{p \in \mathcal{E}} \frac{|\mathcal{S}(s,p)|}{|\mathrm{sp}(s,p)|}$$

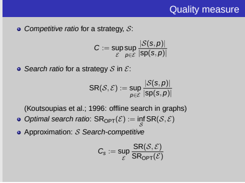
- *Search ratio* for a strategy $\mathcal{S}$ in $\mathcal{E}$:

$$\mathrm{SR}(\mathcal{S},\mathcal{E}) := \sup_{p \in \mathcal{E}} \frac{|\mathcal{S}(s,p)|}{|\mathrm{sp}(s,p)|}$$

(Koutsoupias et al.; 1996: offline search in graphs)

- *Optimal search ratio*: $\mathrm{SR}_{\mathrm{OPT}}(\mathcal{E}) := \inf_{\mathcal{S}} \mathrm{SR}(\mathcal{S},\mathcal{E})$
- Approximation: $\mathcal{S}$ *Search-competitive*

$$C_S := \sup_{\mathcal{E}} \frac{\mathrm{SR}(\mathcal{S},\mathcal{E})}{\mathrm{SR}_{\mathrm{OPT}}(\mathcal{E})}$$



Quality measure

- *Competitive ratio* for a strategy, $\mathcal{S}$:

$$C := \sup_{\mathcal{E}} \sup_{p \in \mathcal{E}} \frac{|\mathcal{S}(s,p)|}{|\text{sp}(s,p)|}$$

- *Search ratio* for a strategy $\mathcal{S}$ in $\mathcal{E}$:

$$\text{SR}(\mathcal{S},\mathcal{E}) := \sup_{p \in \mathcal{E}} \frac{|\mathcal{S}(s,p)|}{|\text{sp}(s,p)|}$$

  (Koutsoupias et al.; 1996: offline search in graphs)
- *Optimal search ratio*: $\text{SR}_{\text{OPT}}(\mathcal{E}) := \inf_{\mathcal{S}} \text{SR}(\mathcal{S},\mathcal{E})$
- Approximation: $\mathcal{S}$ *Search-competitive*

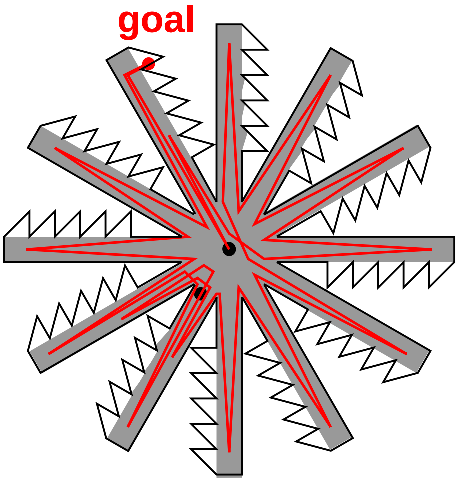
$$C_s := \sup_{\mathcal{E}} \frac{\text{SR}(\mathcal{S},\mathcal{E})}{\text{SR}_{\text{OPT}}(\mathcal{E})}$$

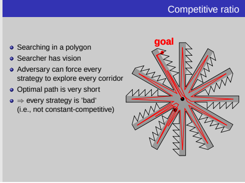
Of course, we may use the competitive ratio as a quality measure. That is, we compare the strategy to the shortest path and take the worst case ratio over all points in an environment over all environments. But there is a problem with this ratio.
We can use the search ratio to measure this qualtiy: We define the search ratio of a given strategy in a given environment as the worst case ratio of the strategy. Now, the *optimal* search ratio is the best achievable search ratio in the given environment.
Unfortunately, for many settings, **including polygons**, it is not known how to compute a path with optimal search ratio. Moreover, in an online setting we may have no chance to compute an optimal search path. But we can try to give an approximation. We call a strategy *search competitive* if it approximates the optimal search path within a constant factor.

Observe the difference to the competitive ratio! We no longer compare only to the shortest path to the goal, but to the best achievable worst case ratio. Keep in mind, that the ratio in the definition of search-competitive is in fact a ratio of ratios.

Competitive ratio

- Searching in a polygon
- Searcher has vision
- Adversary can force every strategy to explore every corridor
- Optimal path is very short
- $\Rightarrow$ every strategy is 'bad' (i.e., not constant-competitive)





Competitive ratio

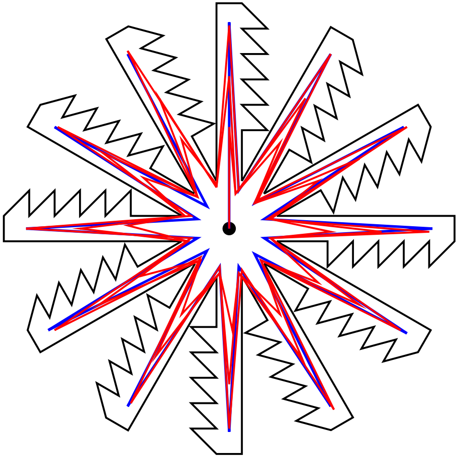
- Searching in a polygon
- Searcher has vision
- Adversary can force every strategy to explore every corridor
- Optimal path is very short
- ⇒ every strategy is 'bad' (i.e., not constant-competitive)

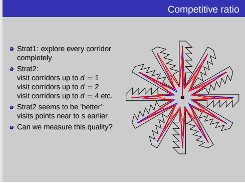
Let's consider, for example, searching in polygons. The searcher is equipped with a vision system. For example, from THIS point, the searcher can see the gray shaded region, and from THIS point, the searcher sees THIS region.

We can force any strategy to explore every corridor. No matter in which order the pockets are visited, we place the goal in the last visited pocket. Because the optimal path is very short, every strategy is considered bad in this framework.

Competitive ratio

- Strat1: explore every corridor completely
- Strat2:
  visit corridors up to $d = 1$
  visit corridors up to $d = 2$
  visit corridors up to $d = 4$ etc.
- Strat2 seems to be 'better': visits points near to *s* earlier
- Can we measure this quality?




Competitive ratio

- Strat1: explore every corridor completely
- Strat2:
  visit corridors up to $d = 1$
  visit corridors up to $d = 2$
  visit corridors up to $d = 4$ etc.
- Strat2 seems to be 'better': visits points near to s earlier
- Can we measure this quality?

Now, let us observe two strategies: the first one successively visits every corridor completely. The seconds strategy visits every corridor up to a certain distance, and then doubles this distance.
Intuitively, the second strategy seems to be better: points near to the start are visited early, so their ratios can't be that bad if the goal is near to the start.
On the other hand, we can 'afford' to visit points farther away after walking a longer path, because the optimal path to these points is also long.

Now, the question is, if we can measure this quality.

Quality measure

- *Competitive ratio* for a strategy, $\mathcal{S}$:

$$C := \sup_{\mathcal{E}} \sup_{p \in \mathcal{E}} \frac{|\mathcal{S}(s,p)|}{|\mathrm{sp}(s,p)|}$$

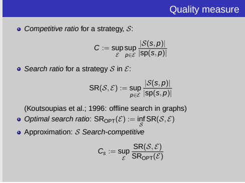
- *Search ratio* for a strategy $\mathcal{S}$ in $\mathcal{E}$:

$$\mathrm{SR}(\mathcal{S},\mathcal{E}) := \sup_{p \in \mathcal{E}} \frac{|\mathcal{S}(s,p)|}{|\mathrm{sp}(s,p)|}$$

  (Koutsoupias et al.; 1996: offline search in graphs)

- *Optimal search ratio*: $\mathrm{SR}_{\mathrm{OPT}}(\mathcal{E}) := \inf_{\mathcal{S}} \mathrm{SR}(\mathcal{S},\mathcal{E})$
- Approximation: $\mathcal{S}$ *Search-competitive*

$$C_S := \sup_{\mathcal{E}} \frac{\mathrm{SR}(\mathcal{S},\mathcal{E})}{\mathrm{SR}_{\mathrm{OPT}}(\mathcal{E})}$$



Quality measure

- *Competitive ratio* for a strategy, $\mathcal{S}$:

$$C := \sup_{\mathcal{E}} \sup_{p \in \mathcal{E}} \frac{|\mathcal{S}(s,p)|}{|\mathrm{sp}(s,p)|}$$

- *Search ratio* for a strategy $\mathcal{S}$ in $\mathcal{E}$:

$$\mathrm{SR}(\mathcal{S},\mathcal{E}) := \sup_{p \in \mathcal{E}} \frac{|\mathcal{S}(s,p)|}{|\mathrm{sp}(s,p)|}$$

(Koutsoupias et al.; 1996: offline search in graphs)

- *Optimal search ratio*: $\mathrm{SR}_{\mathrm{OPT}}(\mathcal{E}) := \inf_{\mathcal{S}} \mathrm{SR}(\mathcal{S},\mathcal{E})$
- Approximation: $\mathcal{S}$ *Search-competitive*

$$C_s := \sup_{\mathcal{E}} \frac{\mathrm{SR}(\mathcal{S},\mathcal{E})}{\mathrm{SR}_{\mathrm{OPT}}(\mathcal{E})}$$

Of course, we may use the competitive ratio as a quality measure. That is, we compare the strategy to the shortest path and take the worst case ratio over all points in an environment over all environments. But there is a problem with this ratio.
We can use the search ratio to measure this qualtiy: We define the search ratio of a given strategy in a given environment as the worst case ratio of the strategy. Now, the *optimal* search ratio is the best achievable search ratio in the given environment.
Unfortunately, for many settings, **including polygons**, it is not known how to compute a path with optimal search ratio. Moreover, in an online setting we may have no chance to compute an optimal search path. But we can try to give an approximation. We call a strategy *search competitive* if it approximates the optimal search path within a constant factor.

Observe the difference to the competitive ratio! We no longer compare only to the shortest path to the goal, but to the best achievable worst case ratio. Keep in mind, that the ratio in the definition of search-competitive is in fact a ratio of ratios.

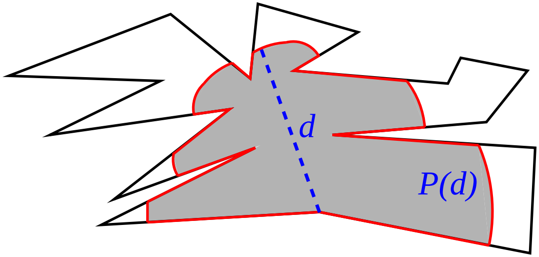
Depth-Restrictable Exploration

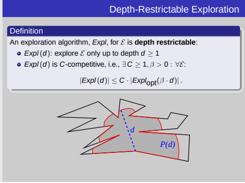
Definition

An exploration algorithm, *Expl*, for $\mathcal{E}$ is **depth restrictable**:

- $Expl(d)$: explore $\mathcal{E}$ only up to depth $d \geq 1$
- $Expl(d)$ is C-competitive, i.e., $\exists\, C \geq 1, \beta > 0 : \forall \mathcal{E}$:

$$|Expl(d)| \leq C \cdot |Expl_{\text{opt}}(\beta \cdot d)|\,.$$




Depth-Restrictable Exploration

Definition

An exploration algorithm, *Expl*, for $\mathcal{E}$ is **depth restrictable**:

- $Expl\,(d)$: explore $\mathcal{E}$ only up to depth $d \geq 1$
- $Expl\,(d)$ is C-competitive, i.e., $\exists C \geq 1, \beta > 0 : \forall \mathcal{E}$:

$$|Expl\,(d)| \leq C \cdot |Expl_{\text{opt}}(\beta \cdot d)|\,.$$

The main idea to the design of search-competitive strategies is to use exploration strategies, more precisely, depth-restricted exploration strategies.

We call an exploration strategy *depth-restrictable*, if we can modify the strategy to explore the enviroment only up to a given depth and the modified strategy is still competitive. But we allow the depth-restricted strategy to explore a little bit more than required; and compare it to the optimal strategy in a larger environment. So we have this factor β HERE.

The factor β makes it sometimes easier to find appropriate exploration algorithms; for example in graphs.

Approximation Framework

Approximation Strategy

Use **Doubling paradigm**: call $Expl\,(2^i)$, $i = 1, 2, 3, \ldots$.

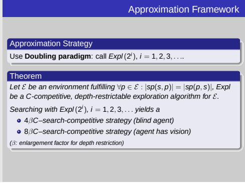
Theorem

Let $\mathcal{E}$ be an environment fulfilling $\forall p \in \mathcal{E} : |sp(s, p)| = |sp(p, s)|$, Expl be a C-competitive, depth-restrictable exploration algorithm for $\mathcal{E}$.

Searching with $Expl\,(2^i)$, $i = 1, 2, 3, \ldots$ yields a

- *$4\beta C$–search-competitive strategy (blind agent)*
- *$8\beta C$–search-competitive strategy (agent has vision)*

(β: enlargement factor for depth restriction)



Approximation Framework

Approximation Strategy

Use **Doubling paradigm**: call *Expl* (2^i), $i = 1, 2, 3, \ldots$.

Theorem

Let $\mathcal{E}$ be an environment fulfilling $\forall p \in \mathcal{E} : |sp(s, p)| = |sp(p, s)|$, Expl be a C-competitive, depth-restrictable exploration algorithm for $\mathcal{E}$.

Searching with Expl (2^i), $i = 1, 2, 3, \ldots$ yields a

- *$4\beta C$–search-competitive strategy (blind agent)*
- *$8\beta C$–search-competitive strategy (agent has vision)*

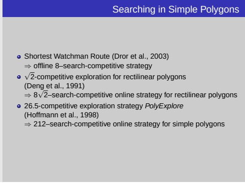
(β: enlargement factor for depth restriction)

Now, if we have found a depth-restrictable exploration strategy, we can simply use the doubling paradigm, that is, we explore the environment up a certain depth and double the exploration depth in every step.
We can show that idea yields indeed a search-competitive strategy; with a factor of $4\beta C$ for blind agents and $8\beta C$ for agents with vision. C is the competitive factor of the exploration strategy, and β is the enlargement factor from the depth restriction.

This works for arbitrary kinds of environments, we require only that THIS condition holds.

Searching in Simple Polygons

- Shortest Watchman Route (Dror et al., 2003)
  $\Rightarrow$ offline 8–search-competitive strategy
- $\sqrt{2}$-competitive exploration for rectilinear polygons (Deng et al., 1991)
  $\Rightarrow$ $8\sqrt{2}$–search-competitive online strategy for rectilinear polygons
- 26.5-competitive exploration strategy *PolyExplore* (Hoffmann et al., 1998)
  $\Rightarrow$ 212–search-competitive online strategy for simple polygons



- Shortest Watchman Route (Dror et al., 2003)
  ⇒ offline 8–search-competitive strategy
- $\sqrt{2}$-competitive exploration for rectilinear polygons
  (Deng et al., 1991)
  ⇒ $8\sqrt{2}$–search-competitive online strategy for rectilinear polygons
- 26.5-competitive exploration strategy *PolyExplore*
  (Hoffmann et al., 1998)
  ⇒ 212–search-competitive online strategy for simple polygons

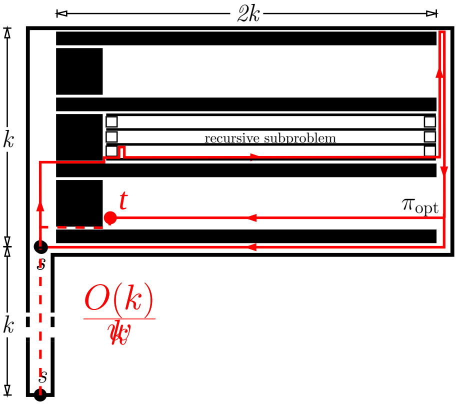
For example, we can apply our approximation framework to simple polygons. For the offline case, we get an 8-approximation using an algorithm that computes the shortest watchman route.
In the online setting, we can use the strategy by Deng, Kameda and Papadimitriou to get a search-competitive strategy for rectilinear polygons.

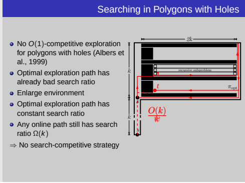
And for general simple polygons, we can use PolyExplore by Hoffmann, Icking, Klein, and Kriegel.

Searching in Polygons with Holes

- No $O(1)$-competitive exploration for polygons with holes (Albers et al., 1999)
- Optimal exploration path has already bad search ratio
- Enlarge environment
- Optimal exploration path has constant search ratio
- Any online path still has search ratio $\Omega(k)$

$\Rightarrow$ No search-competitive strategy




Searching in Polygons with Holes

- No $O(1)$-competitive exploration for polygons with holes (Albers et al., 1999)
- Optimal exploration path has already bad search ratio
- Enlarge environment
- Optimal exploration path has constant search ratio
- Any online path still has search ratio $\Omega(k)$

$\Rightarrow$ No search-competitive strategy

It was shown by Albers, Kursawe and Schuierer that there is no exploration strategy with a constant competitive factor for polygons WITH holes.
The scene for the lower bound looks like this, but the details are not import here. Unfortunately, we cannot use this scene as a lower bound for searching, because the optimal exploration path has already a bad search ratio. To solve this problem, we can enlarge the scene like this. Now, the optimal exploration has a constant search ratio, while any online exploration has a search ratio in $\Omega(k)$, so there is no search-competitive strategy for polygons with holes.

Again, the details are not important here. The more interesting part is that we can GENERALIZE this idea!

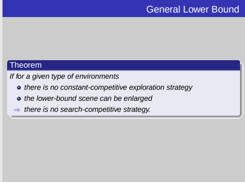
General Lower Bound

Theorem

If for a given type of environments

- *there is no constant-competitive exploration strategy*
- *the lower-bound scene can be enlarged*

$\Rightarrow$ *there is no search-competitive strategy.*



Theorem

If for a given type of environments

- *there is no constant-competitive exploration strategy*
- *the lower-bound scene can be enlarged*

⇒ *there is no search-competitive strategy.*

So, we can show this theorem: If there is no constant-competitive exploration and we can enlarge the scene used to prove the lower bound, then there is no search-competitive strategy.

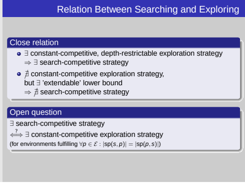
Relation Between Searching and Exploring

Close relation

- $\exists$ constant-competitive, depth-restrictable exploration strategy $\Rightarrow \exists$ search-competitive strategy
- $\nexists$ constant-competitive exploration strategy,
  but $\exists$ 'extendable' lower bound
  $\Rightarrow \nexists$ search-competitive strategy

Open question

$\exists$ search-competitive strategy
$\overset{?}{\Longleftrightarrow} \exists$ constant-competitive exploration strategy
(for environments fulfilling $\forall p \in \mathcal{E} : |\mathrm{sp}(s,p)| = |\mathrm{sp}(p,s)|$)



Relation Between Searching and Exploring

Close relation

- $\exists$ constant-competitive, depth-restrictable exploration strategy
  $\Rightarrow \exists$ search-competitive strategy
- $\nexists$ constant-competitive exploration strategy,
  but $\exists$ 'extendable' lower bound
  $\Rightarrow \nexists$ search-competitive strategy

Open question

$\exists$ search-competitive strategy
$\overset{?}{\Longleftrightarrow} \exists$ constant-competitive exploration strategy
(for environments fulfilling $\forall p \in \mathcal{E} : |\mathrm{sp}(s, p)| = |\mathrm{sp}(p, s)|$)

Altogether, we have seen a very close relation between searching and exploring: If there is a constant-competitive, depth-restrictable exploration, then there is a search competitive strategy and if there is no constant-competitive exploration but an extendible lower bound, then there is no constant-search competitive strategy.

Now, it is an open question, if there is an even closer relation; that is: is there a search competitive strategy if and only if there is a constant-competitive exploration?

Summary

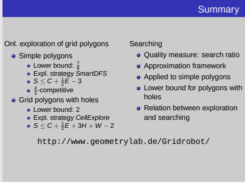
Onl. exploration of grid polygons

- Simple polygons
  - Lower bound: $\frac{7}{6}$
  - Expl. strategy *SmartDFS*
  - $S \leq C + \frac{1}{2}E - 3$
  - $\frac{4}{3}$-competitive
- Grid polygons with holes
  - Lower bound: 2
  - Expl. strategy *CellExplore*
  - $S \leq C + \frac{1}{2}E + 3H + W - 2$

Searching

- Quality measure: search ratio
- Approximation framework
- Applied to simple polygons
- Lower bound for polygons with holes
- Relation between exploration and searching

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



Summary

Onl. exploration of grid polygons

- Simple polygons
  - Lower bound: $\frac{7}{6}$
  - Expl. strategy *SmartDFS*
  - $S \leq C + \frac{1}{2}E - 3$
  - $\frac{4}{3}$-competitive
- Grid polygons with holes
  - Lower bound: 2
  - Expl. strategy *CellExplore*
  - $S \leq C + \frac{1}{2}E + 3H + W - 2$

Searching

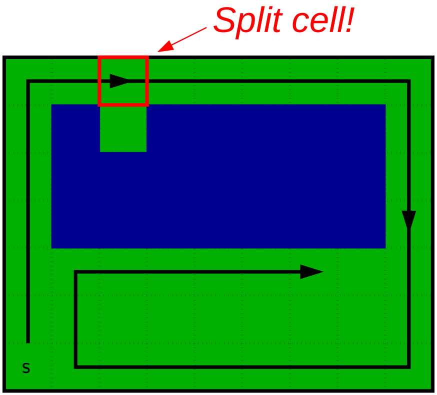
- Quality measure: search ratio
- Approximation framework
- Applied to simple polygons
- Lower bound for polygons with holes
- Relation between exploration and searching

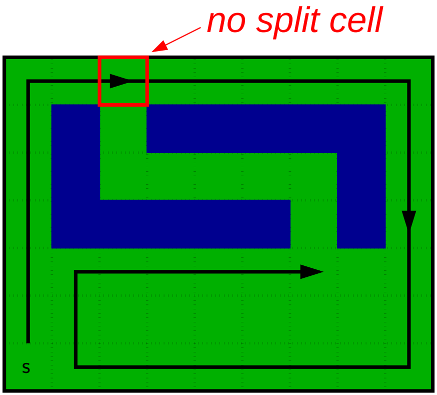
`http://www.geometrylab.de/Gridrobot/`

Here is a summary of the results...
More important, we have seen that the search ratio gives a strong relation between exploration and searching.

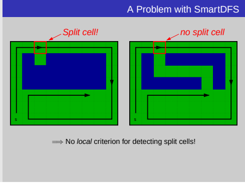
Thank you!

A Problem with SmartDFS





$\Longrightarrow$ No *local* criterion for detecting split cells!



A Problem with SmartDFS

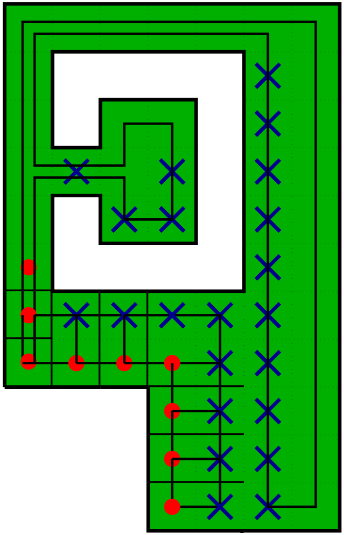
$\Longrightarrow$ No *local* criterion for detecting split cells!

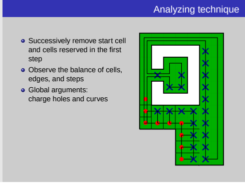
In the presence of holes, we have a problem with SmartDFS, that we can see in this example. We explore the polygon up to THIS cell. Now, we cannot determine, whether this cell is a split cell or not, because we cannot distinguish both cases so far. Thus, we have to proceed with the exploration, until we reach THIS cell. Now, we are able to distinguish both cases and see that THIS was indeed a split cell, whereas THIS was no one.

So, the problem is that there is no longer a *local* criterion for detecting split cells.

Analyzing technique

- Successively remove start cell and cells reserved in the first step
- Observe the balance of cells, edges, and steps
- Global arguments:
  charge holes and curves





Analyzing technique

- Successively remove start cell and cells reserved in the first step
- Observe the balance of cells, edges, and steps
- Global arguments: charge holes and curves

x

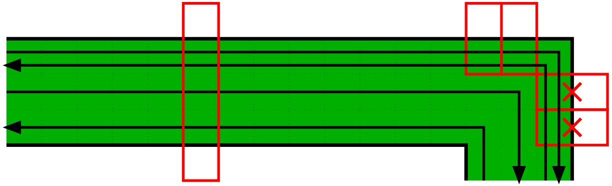
Theorem (Number of Steps)

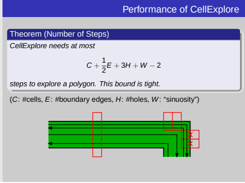
CellExplore needs at most

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

steps to explore a polygon. This bound is tight.

(C: #cells, E: #boundary edges, H: #holes, W: “sinuosity”)





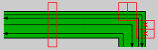
Theorem (Number of Steps)

CellExplore needs at most

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

steps to explore a polygon. This bound is tight.

(C: #cells, E: #boundary edges, H: #holes, W: “sinuosity”)



We can show, that the number of steps needed by our strategy is mainly bounded by the number of cells and half the number of edges. Additionally, we need at most three steps for every obstacle, and, unfortunately, we need another parameter, W, which is not very intuitive. It is used to diffentiate between straight and winded polygons such as spirals with may turns. To get an idea for what we count in W observe a corridor of width 3: CellExplore visits this corridor 4 times. We charge 3 visits to the cells. In the straight part, we can charge the additional visit to the *two* boundary edges like shown HERE. In a bend like here, we do not have *two* boundary edges for every doubly visited cell, so we have to count THESE two cells in W.

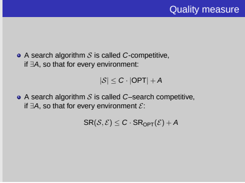
Quality measure

- A search algorithm $\mathcal{S}$ is called C-competitive, if $\exists A$, so that for every environment:

$$|\mathcal{S}| \leq C \cdot |\mathrm{OPT}| + A$$

- A search algorithm $\mathcal{S}$ is called C–search competitive, if $\exists A$, so that for every environment $\mathcal{E}$:

$$\mathrm{SR}(\mathcal{S}, \mathcal{E}) \leq C \cdot \mathrm{SR}_{\mathrm{OPT}}(\mathcal{E}) + A$$



Quality measure

- A search algorithm $\mathcal{S}$ is called C-competitive, if $\exists A$, so that for every environment:

$$|\mathcal{S}| \leq C \cdot |\mathrm{OPT}| + A$$

- A search algorithm $\mathcal{S}$ is called C–search competitive, if $\exists A$, so that for every environment $\mathcal{E}$:

$$\mathrm{SR}(\mathcal{S},\mathcal{E}) \leq C \cdot \mathrm{SR}_{\mathrm{OPT}}(\mathcal{E}) + A$$

x