Online Motion Planning

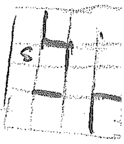
Example: Shannon mouse (Claude Shannon, 1950)

$a \times a$ grid environment; walls between cells allowed;

mouse: can move 1 step ↔↕ if no wall in the way

(no memory) can leave a mark $\in \{N, E, S, W\}$ in each cell (and update on later visits) initially: all marks = N

Start cell S, target cell T; T reachable from S (not (path definition!)



Question: Can mouse find T, starting from S ?

(more precisely: Does there exist a strategy A for mouse such that for each grid maze M, for each placement of S and T in M, mouse starting from S finds T ?

Yes! (Shannon build 5x5 maze)

Strategy MOUSE

```
WHILE T not found do
    leave current cell in clockwise-first free direction
                                              after mark;
    update mark of current cell to this direction
```

(→ Example)

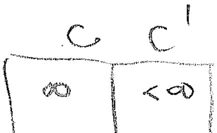
Theorem: Strategy MOUSE always finds the target

Proof: Remove T from maze. Claim: Mouse visits each cell (reachable from S) infinitely often

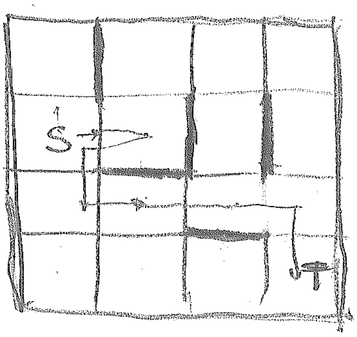
(MOUSE doesn't stop) → (both ≠ ∅)

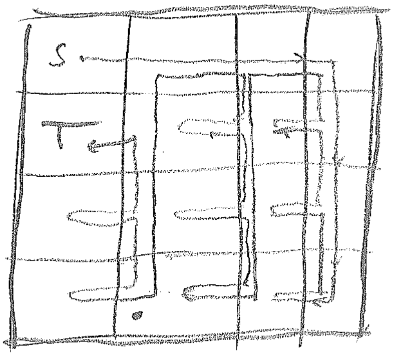
Proof: Otherwise, 2 classes of cells: infinitely often visited, finitely often visited

Two cells of different classes must be adjacent, without wall (path argument)



↳ on each visit, mark is turned by 90°
⇒ c' also visited ∞ often





MOUSE seems
not very efficient !
(many cells 3 times vis

Efficiency of search strategy:
Compare max length of S-to-T path found by A
to shortest S-to-T path possible (OPT)

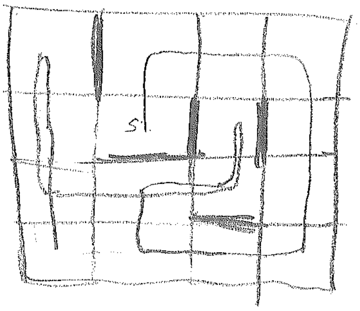
Efficiency of exploration strategy:
Compare max length of exploration path found by A
(a path that visits every cell)
to shortest exploration path possible. (OPT)

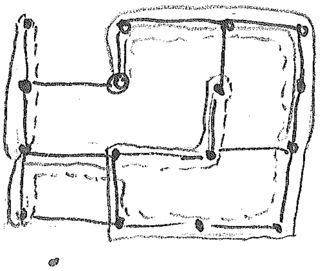
For exploration: Robot is Turing Machine,
can store partial map of part of environ
so far visited

Shannon's maze = cell graph
vertices: cells
edges: connections between adjacent cells
not separated by wall

simple exploration strategy: Depth First Search
(Charles Pierre Tremaux, 188

neighbors: N, E
ordered





Grid graph (?)

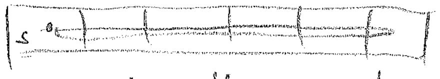
DFS

```
∀v:  H(v) := N(v)        (neighboring vertices in given order)

Q := {s};                (Q = stack,  s = start vertex)
mark s;

while Q ≠ ϕ do
    v := top(Q);
    if H(v) ≠ ϕ then
        v' := next(H(v))
        H(v) := H(v) \ {v'}
        if v' not marked then  push(Q, v')
                               mark v'
    else pop(Q, v)
```

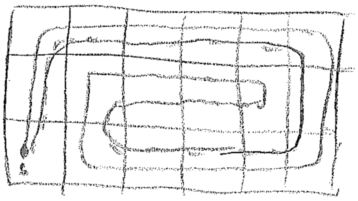
Observe:
Unless abstract DFS algo
robot must walk

Clear DFS visits each cell twice.
(it walks around a tree)
(1 cell visit = 1 edge traversed in grid graph)

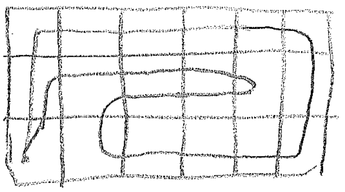
Observations



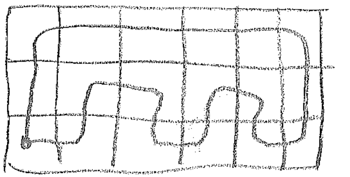
each cell must be visited twice (even OPT does it)



DFS



better

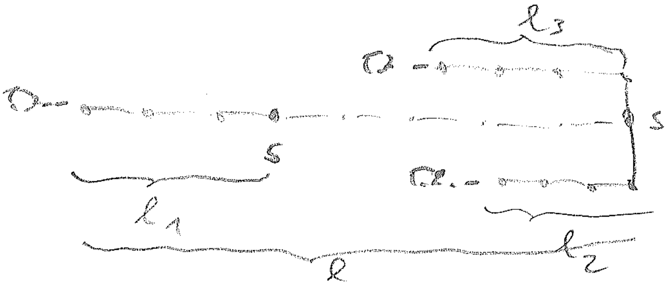


optimal

(Icking, Kamphans, Klein 00)

Theorem No on-line exploration strategy for grid graphs can always do better than twice the optimum solution.
off-line

Proof

(1) 

Let A be an exploration strategy, and l large.
start robot on chain at s
Once l vertices explored at s' : start 2 new chains from
and let their lengths l_2, l_3 grow.

Which event comes first?

(1) robot returns to s.

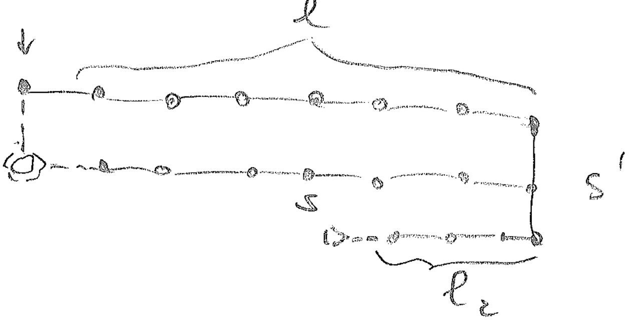
A so far: $\geq 2l_1 + (l-l_1) + 2l_2 + 2l_3 + (l-l_1) = 2(l+l_2+l_3)$

extend all chains by 1 vertex

A left with $\geq 2(l+l_2+l_3)+6 = \text{OPT}$

$\Rightarrow A \geq 2\,\text{OPT} - 6$

(2) robot has checked l+1 vertices in one of the new chain
say the upper.

(2) 

A so far $\geq 2l_1 + (l-l_1) + 2l_2 + l+1$.

Connect the upper/middle chains:

("cost" dro

A left with $\geq l+1 + 2(l_2+1) + l - l_1$

$\Rightarrow \quad A \geq 4l + 4l_2 + 4 = 4(l+l_2)+4$

$OPT = 2(l+1) + 2(l_2+1) \overset{\circ}{=} 2(l+l_2)+4$

$\Rightarrow \quad A \geq 2\,OPT - 4 > 2\,OPT - 6.$

In either case: $\frac{A}{OPT} \geq 2 - \frac{6}{OPT} \geq 2 - \varepsilon$ if, given ε <br> l is large

□

(Sleator, Tarjan '85)

<u>Definition</u> Π: problem, P∈Π instance

A: strategy for solving Π on-line

OPT: optimal strategy for solving Π off-line.

Then, A is called competitive with factor C :⟺

$\exists\, \alpha > 0 \quad \forall P \in \Pi: \quad cost(A(P)) \leq C \cdot cost(OPT(P)) + \alpha.$

like additive c
in Big-O n

<u>Corollary</u> DFS is 2-competitive for grid graph exploration, and no other strategy can achieve a smaller factor

$[\frac{4}{3}, \frac{17}{10}]$

Another example: Bin packing; on-line vs. off-line

First-Fit: all bins but 1 are at least half full

$\Rightarrow (m-1)\frac{H}{2} \leq \sum h_k \Rightarrow m-1 \leq 2\lceil\frac{\sum h_k}{H}\rceil = 2 \cdot$ min # bins by th
volume

<u>Now:</u> simple cellular environments (no holes) (only one outside wall, forms simple rectangular polygon)

(offline-complex: Open. ∃Hamilton? ∈ P)

Theorem $\frac{7}{6}$ is lower bound to competitive factor of exploration strategies for simple cellular environments.

(IKKL05)

Proof :

glue pieces together
(no extra cells)

proof by backwards induction

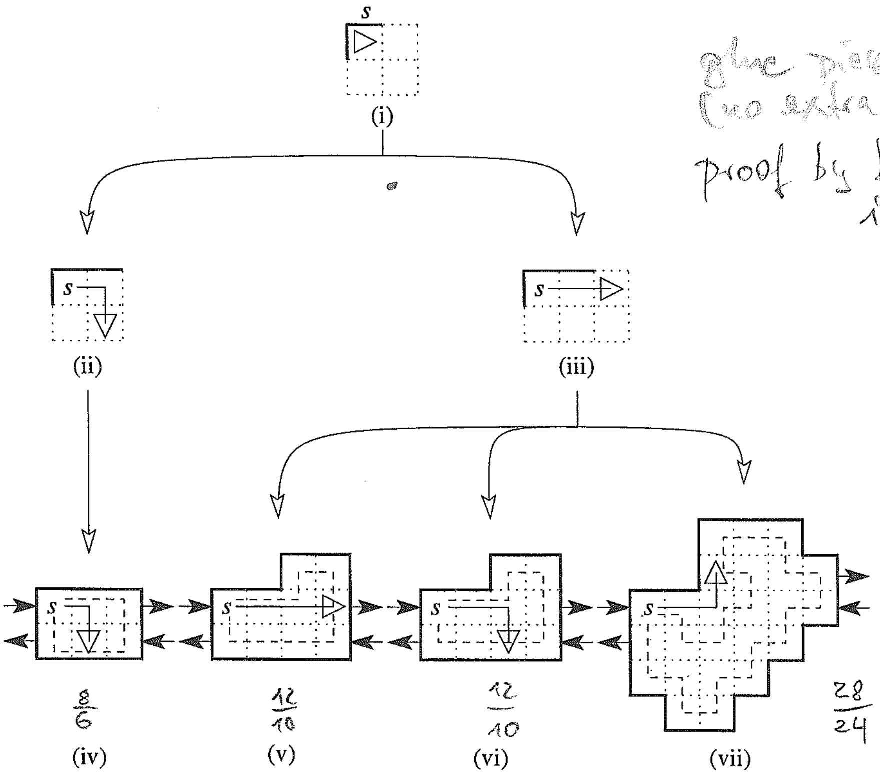


Abbildung 1.11: Untere Schranke für die Exploration einfacher Gitterpolygone.

Wir können dieses Schema benutzen, um Polygone beliebiger Größe zu konstruieren, indem wir die einzelnen Blöcke über die 'Eingangs-' und 'Ausgangszellen', die in Abbildung 1.11(iv)–(vii) mit Pfeilen gekennzeichnet sind, konkatenieren. Sobald der Roboter einen Block verlassen hat, beginnt das 'Spiel' von neuem. Dabei ist sichergestellt, daß wir kein Polygon mit Löchern oder ein überlappendes Polygon konstruieren, da wir das Polygon stets in dieselbe Richtung erweitern.

Wenn wir die Umgebung nicht beliebig erweitern könnten, und zum Beispiel nur eine Umgebung mit maximal D vielen Zellen erzeugen könnten, dann ist die Konstruktion der unteren Schranke nicht gültig. Jeder *vernünftige* Algorithmus könnte diese Umgebung optimal im Sinne von Definition 1.6 explorieren, da dann eine Konstante $\alpha \gg D$ existiert, mit $|S_{\mathrm{ALG}}| \leq |S_{\mathrm{OPT}}| + \alpha$. □

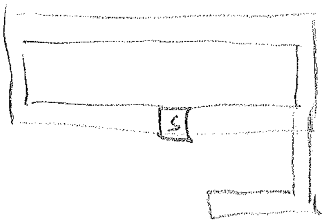
Betrachten wir zunächst die Erkundung eines Polygons mit DFS, Algorithmus 1.4. Der Roboter erkundet das Polygon mit der "Linke-Hand-Regel", d. h. er bevorzugt einen Schritt nach links einem Schritt geradeaus und bevorzugt einen Geradeausschritt einem Schritt nach rechts. Die aktuelle Bewegungsrichtung (Nord, Ost, Süd oder West) ist in der Variablen *dir* gespeichert und die Funktionen cw(*dir*), ccw(*dir*) und reverse(*dir*) berechnen die Richtung bei Drehung um

Our next goal we shall work towards

(Icking, Kamphans, Langetepe, TCS 05)

<u>Theorem</u> There exists a $\frac{4}{3}$-competitive strategy (called SmartDFS) for exploration of simple cellular environme

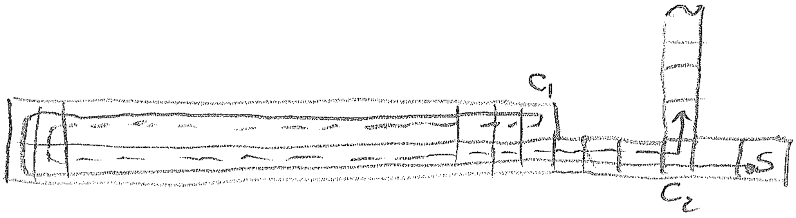
<u>Remarks</u> (i) no contradiction to lower bound 2 — this was for non-simple environments



(ii) gap $\left[\frac{7}{6}, \frac{8}{6}\right]$ is pretty small.

<u>Idea</u> What makes plain DFS behave poorly?

(A) When retracting, DFS always follows the paths used for advancing.

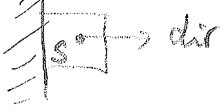


<u>Better</u>: go "directly from c_1 to c_2; continue exploration from there.

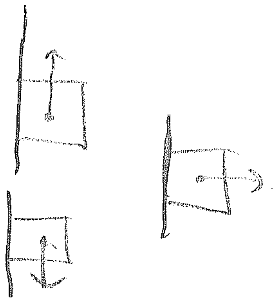
Pseudo Code : dir ∈ {N, E, S, W}, reverse(dir), cw(dir), ccw(dir)
start cell s on boundary given

<u>Improved DFS</u> :

```
pick direction dir such that reverse(dir) is blocked ;
exploreCell (dir);
go back to s on shortest path;
```



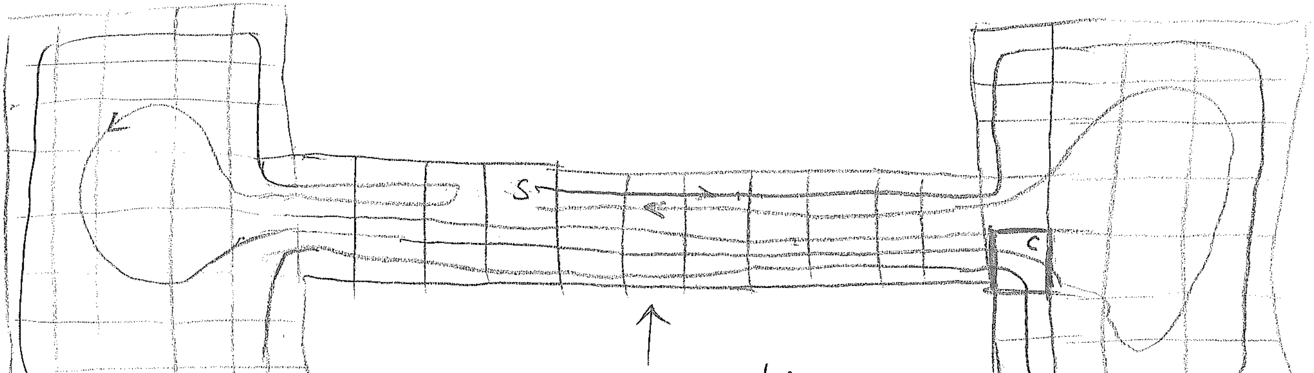
```
Explore Cell (dir):
base := current cell ;      (local variable)
ExploreStep (base, ccw (dir));
ExploreStep (base,  dir  );
ExploreStep (base,  cw (dir));
```



```
ExploreStep (base, dir):

if unexplored(base, dir) then
   go back to base on shortest path through
                  cells explored ;
   move(dir);
   ExploreCell(dir);
```

All cells of advancing path are "base", stored in different recursive calls to ExploreCell.
When current cell has no unexplored neighbors,
preceding base cells are inspected
until one is found that does have unexplored neighbors
Robot moves back to this cell.

(B) Set of unexplored cells may become disconnected
DFS handles components in wrong order

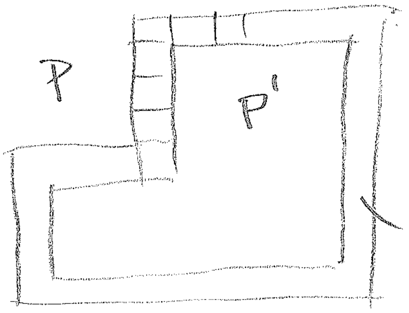


4 passes through corridor

- on visiting cell c, set of unvisited cells became disconnected → c = split cell
- better first finish right part before moving left
  → only 2 passes through corridor.

Closer examination requires some preparations.

Definition (i) boundary cell of P: adjacent, or diagonally adjacent, to blocked cell



(ii) 1-layer of P: all boundary cells

(iii) 1-Offset P' of P: $P \setminus$ 1-layer of P

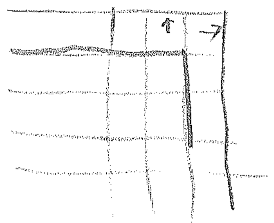
(iv) P^l by induct

Lemma: If $P^l \neq \phi$, it has $\leq \underbrace{E(P)}_{\#\text{edges of } P} - 8l$ many edges

Proof: By induction on l. Sufficient: $l = 1$

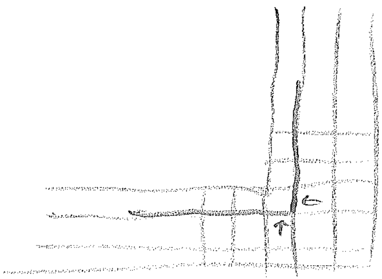
Clockwise walk around boundary of P: #right turns = #left tu

In right turn:



1-offset looses 2 edges

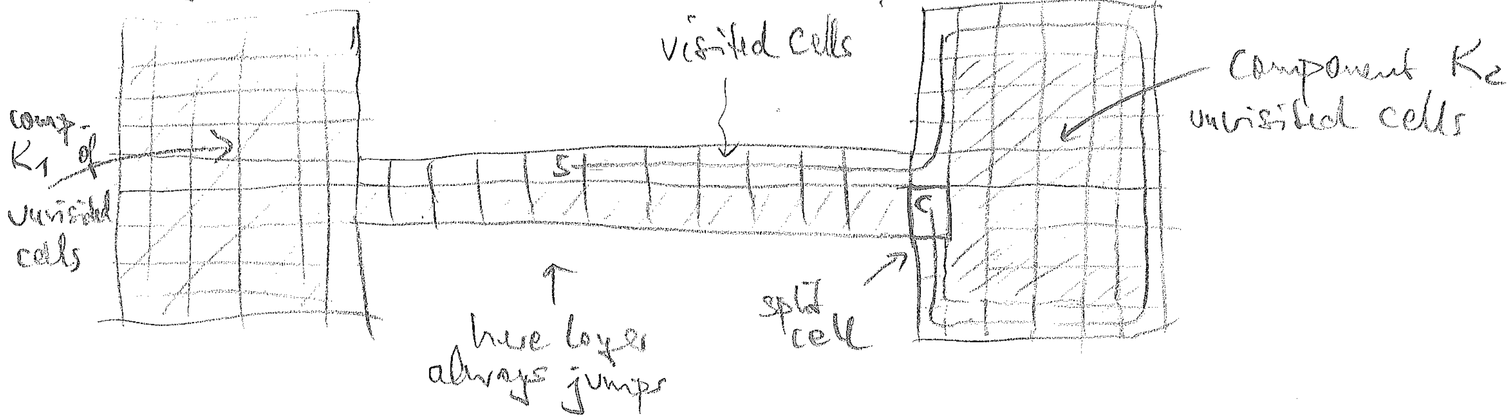
On left turn



it gains

□

Coming back to bad DFS example:


visited cells
component K_2 unvisited cells
comp. K_1 of unvisited cells
s
c
split cell
here layer always jumps

Observation split cell in layer l

K_2 fully surrounded by visited cells of layer l

K_1 only partially —"— (due to layer jump

=> deal with K_2 first !

there are 3 types of components: → (11)

Sometimes no partially encircled component exists →

SmartDFS : as ImprovedDFS, except for

```
ExploreCell(dir):
mark current cell by layer number;          ←
base := current cell;
if not Splitcell(base) then                 ←
    ExploreStep(base, ccw(dir));
    ExploreStep(base, dir);
    ExploreStep(base, cw(dir));
else visit components in appropriate order,          ←
     exploring partially surrounded component last
```

Analysis of SmartDFS (sketch of essential cases)

Suppose split cell c found at layer l

K_2 a component not partially surrounded, hence visited fi

Example → (13) : K_2 not surrounded by layer $l=4$

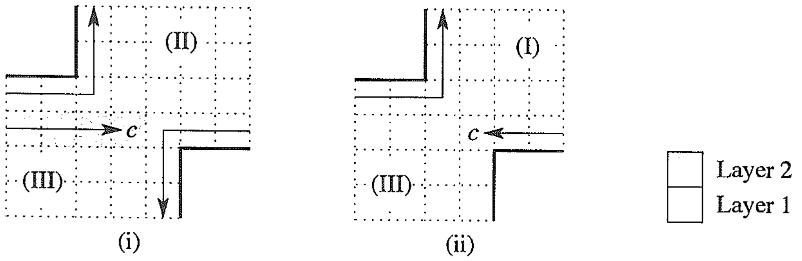
Center square Q of size $2l+1 = 7$ at c

(II) von Layer 2 gar nicht,
(III) partiell umrandet

hier c
in Layer 2

hier c in layer 1

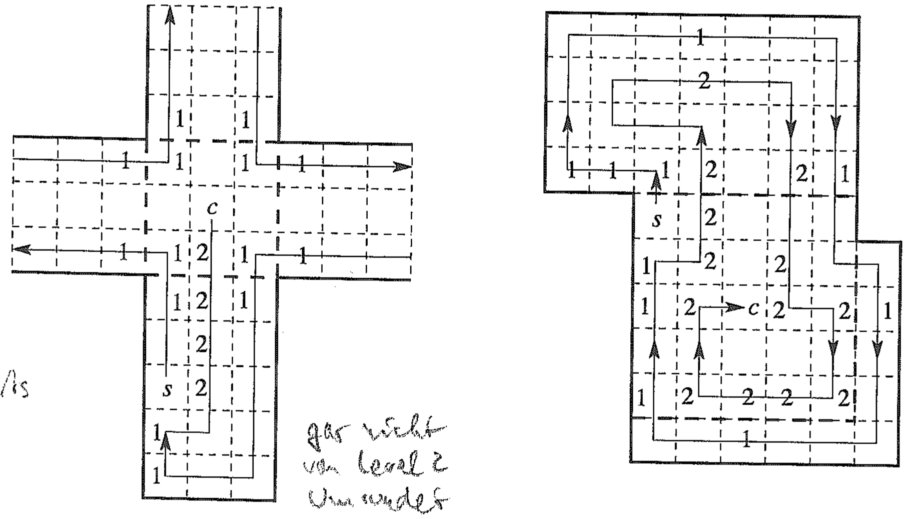
(I) von layer 1 fully
(III) partially surrounded



(i) (ii)

Abbildung 1.16: Drei Arten von Komponenten.

(1



(i)

(ii)

Layer 2 starts at s

Here go right (
explore lower comp

(Komplett von Level 2 umrundet,
aber c ist Level 3

Abbildung 1.17: Keine Komponente vom Typ (iii) existiert.

III

Falls es keine Komponente vom Typ (III) gibt, so liegt einer der folgenden Fälle vor:

(a) Es gibt keine Komponente, die den Startpunkt enthält, weil der Teil des Polygons mit dem Startpunkt bereits vollständig bearbeitet ist, wie z. B. in Abbildung 1.17(i). In diesem Fall ist die Reihenfolge der Bearbeitung der übrigen Komponenten beliebig[7].

? für die c auf K^ℓ liegt ?

Nebenbei zeigt die Abbildung, daß an einer Splitzelle auch mehrere Polygonzerfälle auftreten können. Diese werden nacheinander behandelt, so als würden sie an verschiedenen Zellen stattfinden.

(b) Beide Komponenten sind deswegen komplett umrundet, weil der Komponentenzerfall genau beim Wechsel von einem Layer ℓ zum nächsten Layer $\ell+1$ stattfindet, wie in Abbildung 1.17(ii) gezeigt. In allen anderen Fällen ist mindestens die Zelle, von der aus der Roboter die Splitzelle betreten hat, mit der Layernummer des aktuellen Layers markiert. Insgesamt kann wie im Folgenden beschrieben verfahren werden. Der Layer ℓ wurde also mit der Zelle abgeschlossen, von der aus der Roboter zur Splitzelle gelangt ist. Also kann der Teil des Polygons, von dem aus die Splitzelle betreten wurde, nicht

? s.o.

[7] In der Abbildung könnte man zwei Schritte einsparen, wenn der Roboter zuletzt den nach Westen zeigenden Arm des Polygons bearbeitet und von der zuletzt erkundeten Zelle direkt zum Start zurückkehrt. Um dies entscheiden zu können, müssen aber globale Informationen in Betracht gezogen werden. Für die Analyse unserer Strategie und die obere Schranke der Schrittanzahl spielt diese Abkürzung jedoch keine Rolle.

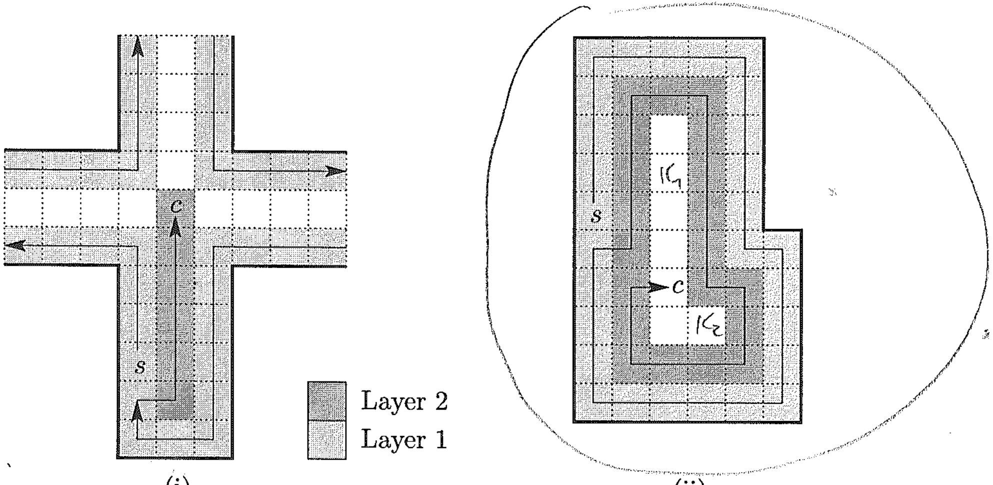
the first cell of layer $\ell - 1$ as in the first case. In both cases the part of P surrounding a component of type III contains the first cell of the current layer ℓ as well as the start cell. Therefore, it is reasonable to explore the component of type III at last.

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.



(i)

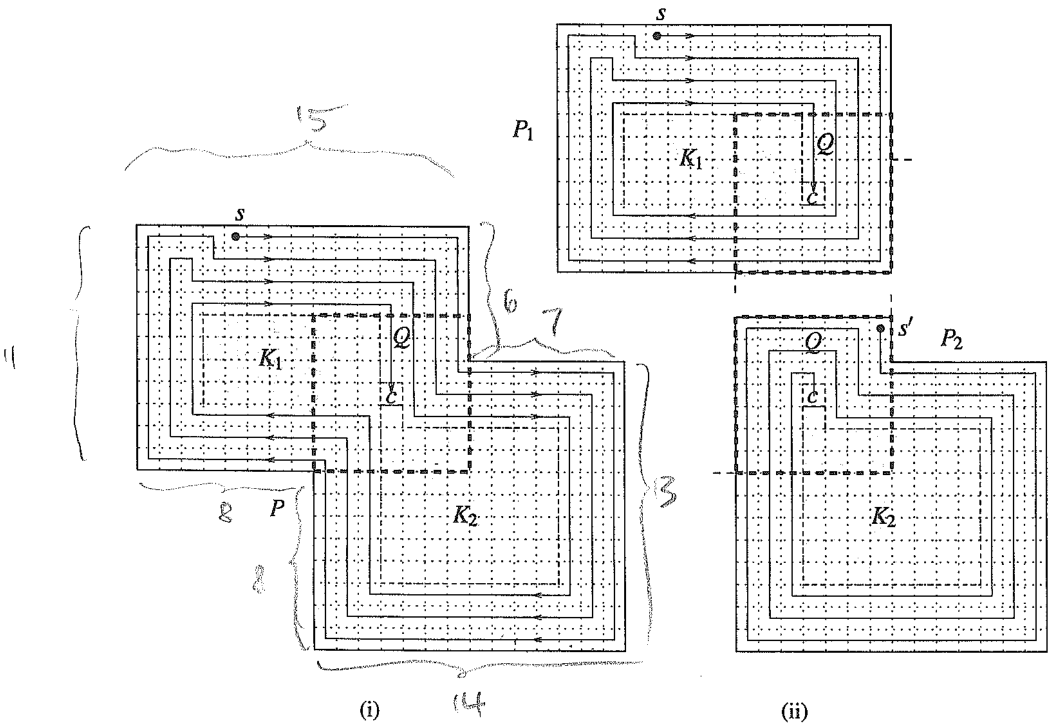
(ii)

explore

K_2 fi

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.



(i) (ii)

Different?

Here, $\ell = 4$ and $K_2 \cup \{c\} = P_2^{\ell-1}$

Abbildung 1.15: Zerlegung eines Gitterpolygons an einer Splitzelle, und die Polygone P_1 und P_2.

Beim Antreffen einer Splitzelle müssen wir die Komponente vom Typ (III) zuletzt bearbeiten, da nur eine Komponente vom Typ (III) den Startpunkt enthalten kann.

K_2: not surrounded by Layer of c ($\ell = 4$)

consider square Q and subpolygons P_1, P_2

Q = interface between P_1, P_2

cut and re-arrange robot's path inside Q such that robot doesn't leave P_1 before reaching c

Def:
$E(R)$ = # edges of polygon R
$C(R)$ = # cells of polygon R
$S(R)$ = # steps made by SmartDFS in exploring R
excess(R) = # extra steps beyond necessary cell number

$$\Rightarrow \quad S(R) = C(R) + \text{excess}(R)$$

(need to upper bound excess(P) !)

<u>Lemma 1</u>

$$\text{excess}(P) \le \underbrace{\text{excess}(P_1)}_{\substack{\text{no excursion} \\ \text{to } P_2}} + \underbrace{\text{excess}(K_2 \cup \{c\})}_{\substack{c = \text{first split} \\ \Rightarrow \text{no excess} \\ \text{in } P_2 \setminus (K_2 \cup \{c\})}} + \underbrace{1}_{\substack{c \text{ visi} \\ \text{twice} \\ \text{2nd v} \\ \text{is exce} \\ \text{for P}}}$$

<u>Lemma 2</u> For any two cells p, q in R there is a path of length $\le \frac{1}{2} E(R) - 2$ connecting them

$\boxed{\text{Ü}}$

<u>Lemma 3</u> $E(P) + E(Q) = E(P_1) + E(P_2)$ clear in example.

Now we can prove important intermediate result:

<u>Theorem</u> $S(P) \le C(P) + \frac{1}{2} E(P) - 3$ for SmartDFS.

<u>Proof</u> by induction of number of components generated.

induction hypothesis: no split cell

$\Rightarrow$ no excess $\Rightarrow$ SmartDFS visits all $C(P)$ cells by path of length $C(P)-1$

returns to s by shortest path of length $\le \frac{1}{2}E(P)-2$ (Lemma 2)

induction step

Let c be first split cell, as in example, detected in P

$$\text{excess}(P) \underset{\text{Lemma 1}}{\le} \underbrace{\text{excess}(P_1)}_{\underset{\text{ind.hyp.}}{\le} \frac{1}{2}E(P_1)-3} + \underbrace{\text{excess}(K_2 \cup \{c\})}_{\underset{\text{ind.hyp}}{\le} \frac{1}{2}\underbrace{E(K_2\cup\{c\})}_{\le E(P_2)-8(l-1)}-3} + 1$$

since $K_2 \cup \{c\} = P_2^{l-1}$

$$\le \frac{1}{2}\big(\underbrace{E(P_1)+E(P_2)}\big) - 4(l-1) - 5$$

$$\underset{\text{Lemma 3}}{=} E(P) + \underbrace{E(Q)}_{4(2l-1)}$$

by def. of Q

$$= \frac{1}{2}E(P) - 3.$$

Hence, $S(P) = C(P) + \text{excess}(P) \le C(P) + \frac{1}{2}E(P) - 3.$ □

Nice bound, in particular if $E(P)$ is small, but not yet a competitive factor. Need to keep on!

Observation: SmartDFS works nicely in corridors of width 1 o

=: narrow passages

(formally: all cells whose removal does not change layers of remaining cells)