RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT BONN
INSTITUT FÜR INFORMATIK I

Elmar Langetepe

Online Motion Planning

MA INF 1314

Summersemester 2016
Manuscript: Elmar Langetepe

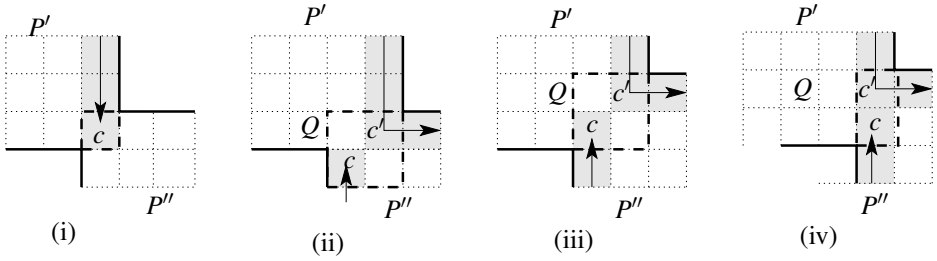


Figure 1.20: A gridpolygon P_i that is separated into components of type (I) or (II) at the split-cell. The rectangle Q is always inside P_i.

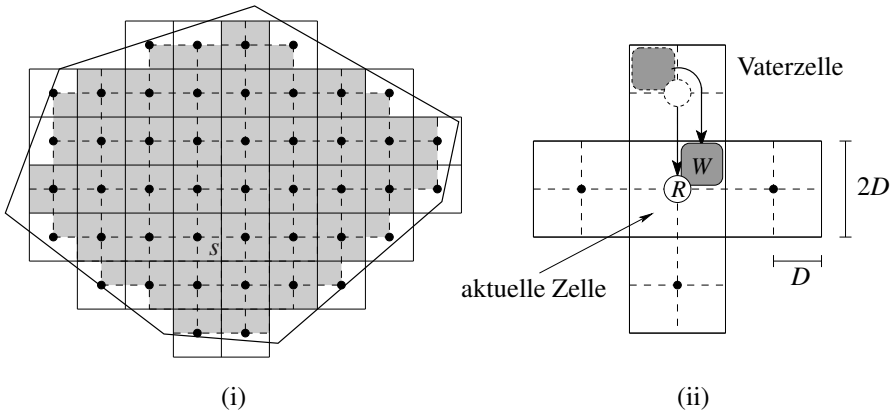


Figure 1.21: $2D$-cells and $D \times D$ sub-cells.

1.4.3 Exploration of general gridpolygons

For the more general exploration of gridpolygons we first slightly change the model[4]: We consider an agent that is located at the center of 4 cells of size $D \times D$. The tool for the exploration still has size $D \times D$ as before and moves freely around the agent. More precisely, we consider 4 **sub-cells** of size $D \times D$ and unify them to a $2D$-cell[5]; see Figure 1.21(i).

It can happen that for the $2D$-cell, not all sub-cells belong to the initial gridpolygon, since some of the sub-cells simply belong to the boundary. Such $2D$-cell are denoted as **partially occupied cells**.

In Figure 1.21(i) all cells intersected by the original polygonal segments are partially occupied (compare also reffigfigOnline/PolyToGrid on page 8). The agent is always located in the center of the $2D$-cell. Analogously to the SmartDFS model, the agents scans the four adjacent $2D$-cells. The tool moves freely around the agent, we would like the count the number of steps of the tool; see also Figure 1.21(ii).

The current cell of the agent is denoted as *current cell*. The *parent cell* of the agent is the cell where he is actually coming from. In the beginning we initially an arbitrary adjacent $2D$-cell as the parent cell.

The strategy "Spanning-Tree-Covering" (STC) constructs a spanning tree for all connected $2D$-cells that are also not occupied. The tool moves along the spanning tree by the Left-Hand-Rule. The construction can be done fully online. The $2D$-cells are detected by the Right-Hand-Rule. Obviously by this approach the tool exactly visits any cell at most once by following the spanning tree. Figure 1.22(i) shows an example for the efficient exploration of all non-occupied cells by $2D$-Spiral-STC. As mentioned before, for the start we can choose an arbitrary parent cell.

The disadvantage of $2D$-Spiral-STC is, that we do not visit sub-cells by that tool which actually lie in the connected component of the sub-cells. Now we relax the behaviour of $2D$-Spiral-STC. The strategy

[4] We will see later that the change was only done for the reason of a convenient analysis and description.

[5] In the following a cell always denotes a $2D$-cell.

Algorithm 1.6 2D-Spiral-STC

2DSPSTC(*parent*, *current* **)**:

Mark *current* as visited.

while *current* has unvisited neighbour cell **do**

- From *parent* search in ccw order for a neighbouring cell *free*, which is not marked as visited and is not partially occupied.
- Build the spanning tree edge from *current* to *free*.
- Move the tool by Left-Hand-Rule along the spanning tree edge to the first sub-cell of *free*.
- Call 2DSPSTC(*current*, *free*).

end while

if $current \neq s$ **then**

- Move by the Left-Hand-Rule along the spanning tree edge back from *current* to the first sub-cell of *parent*.

end if

Algorithm 1.7 SpiralSTC

SPSTC(*parent*, *current* **)**:

Mark *current* as visited.

while *current* has unvisited neighbour cell **do**

- From *parent* search in ccw order for the first neighbouring cell *free*.
- Build a spanning tree edgs from *current* to *free*.
- Move the tool along the spanning tree edge to the first sub-cell of *free*. The movement depends on the local situation. For double-sided edges the tool moves by Left-Hand-Rule along the edge. For single-sided edges the tool might change to the other (left) side of the spanning tree edge in order to avoid an occupied sub-cell for reaching the corresponding sub-cell.
- Call SPSTC(*current*, *free*).

end while

if $current \neq s$ **then**

- Move along the spanning tree edge back from *current* to the first possible sub-cell of *parent*. The movement depends on the type of the edge, as mentioned above.

end if

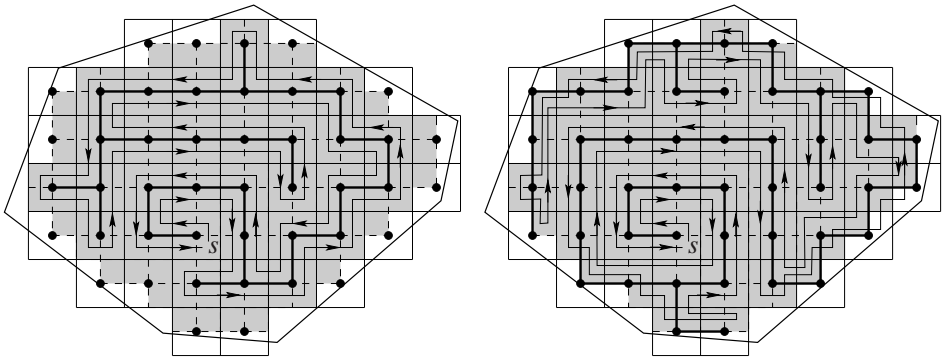


Figure 1.22: Examples for (i) 2D-Spiral-STC and (ii) Spiral-STC.

Spiral-STC (Algorithmus 1.7) also constructs a spanning tree in an online fashion. But we also insert a corresponding edge if a partially occupied 2D-cell contains sub-cells that are still reachable by the tool. In this case the tool cannot always move the the Left-Hand-Rule along the spanning tree edge. The tool has to avoid occupied sub-cells and visits some sub-cells more than once. For systematically analysing the corresponding additional sub-cell visits of the tool we make use of the following notion:

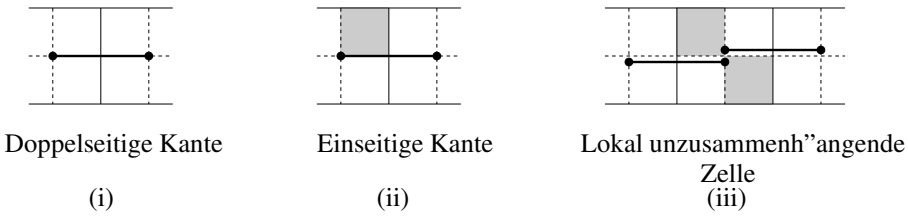


Figure 1.23: (i) Double-sided edge, (ii) one-sided edge, (iii) locally disconnected 2D-cell.

Definition 1.18 A spanning tree edge construced by STC in a gridpolygon P is denoted as
(i) **double-sided edge**, if all adjacent sub-cells belong to the gridpolygon P (Figure 1.23(i)),
(ii) **single-sided edge**, if at least one of the adjacent sub-cells is a boundary sub-cell of P (Figure 1.23(ii)).

Double-sided edges are handled in the same way as in the 2D-Spiral-STC strategy. Single-sided edges impose a detour for the tool, some sub-cells will be visited more than once since the tool changes to the other side of the spanning tree edge. For the analysis we will consider the corresponding cases systematically. A special case occurs, if the situation imposes two spanning tree edges for the same cell from different directions. The cell is locally disconnected in this case; see Figure 1.23(iii). This 2D-cell will be visited twice from different directions. For simplicity we internally double the corresponding vertex and the spanning tree has exactly one incoming edge for any vertex. For the analysis we have to take care that we count the cell only once. An example of the execution of Spiral-STC is shown in Figure 1.22(ii).

By the preference rule for the 2D-cells the Spiral-STC constructs spanning trees with many windings. This is not always intended, especially for lawn-mowing or vacuum-cleaning a tool should try to avoid so many turns. The number of turns might also be part of the cost model. The Scan-STC variant has a fixed given preference for vertical or horizontal edges. We would like to make local decision for the construction of spanning tree edges. In our examples we prefer a vertical scan of the gridpolygon. For this we extend the sensor model and allow to have information about all diagonally adjacent 2D-cells of a current cell.

The idea is that the construction of a horizontal edge will be postponed, if it is clear that we can also reach the 2D-cell by another vertical spanning tree edge. To keep the rule simple we only look ahead as

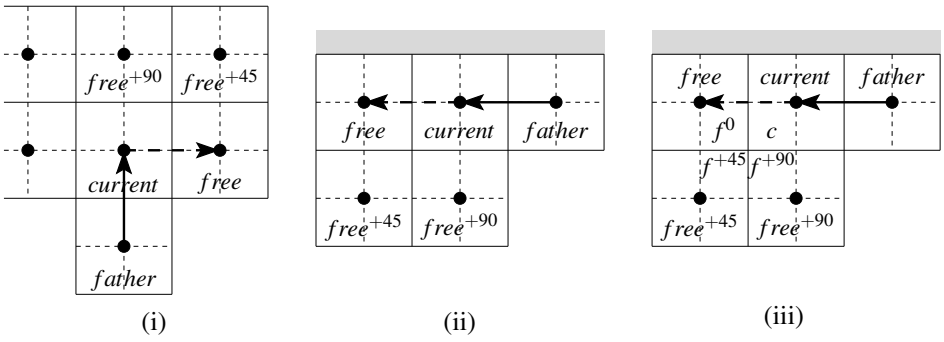


Figure 1.24: Avoid horizontal edges with the Scan-STC.

indicated in Figure 1.24 (i) and (ii). Here we currently would like to build a horizontal edge. The agent is located at cell *current* and is looking (in ccw order starting from *father*) for the first free cell *free*. If there is a counterclockwise path from *free* over $free^{+45}$ and $free^{+90}$ back to the current cell, we change the preference and build a spanning tree edge to $free^{+90}$. Here $free^{+45}$ lies on the sam row as *free* and is the the next cell in ccw order from *free*. $free^{+90}$ is the next cell in ccw order from $free^{+45}$ in the same column as *current*.

If the full turn exists, the cell *free* will also be reached from $free^{+45}$ be a vertical edges and $free^{+45}$ can be reached from $free^{+90}$. Note that we have extended the sensor model in this case and also have information about diagonally adjacent edges.

Analogously, we can also consider partially occupied $2D$-cells and apply the same idea. For the corresponding avoidance rule we consider the sub-cells c, f^0, f^{+45} and f^{+90} instead if the cells *current*, *free*, $free^{+45}$ und $free^{+90}$; see Figure 1.24(iii).

By the above idea we could define a strategy $2D$-Scan-STC that corresponds to $2D$-Spiral-STC. We skip this step and directly define a Scan-STC Algorithm that makes use of the sub-cells c, f^0, f^{+45} and f^{+90} by the same arguments. If f^{+45} and f^{+90} are also free, we will reach f^0 from f^{+45} and in turn f^{+45} from f^{+90}. Algorithmus 1.8 summarizes this behaviour.

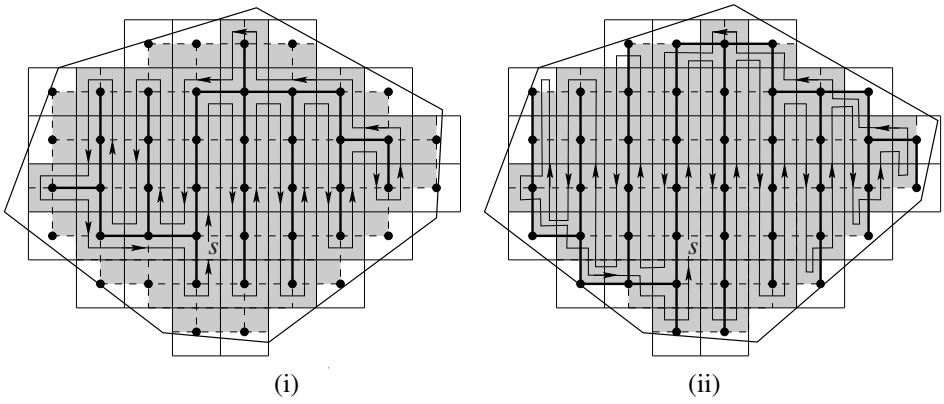


Figure 1.25: Examplle for (i) $2D$-Scan-STC, (ii) Scan-STC.

Theorem 1.19 *(Gabriely, Rimon, 2000)*
Let P be a gridpolygon with C sub-cells. Let K be the number of all sub-cells, which are diagonally adjacent to an occupied (boundary) sub-cell. [6]*. The gridpolygons P will be explored by Spiral-STC and Scan-STC in time $O(C)$ and space $O(C)$. The number of exploration steps,S, for the tool is bounded by: [GR03]*

$$S \leq C + K.$$

[6] K can be estimated by the number of sub-cells in the first layer of P.

Algorithm 1.8 ScanSTC

SCSTC(*parent*, *current* **):**

Mark *current* as visited.

while *current* has unvisited neighbouring cell **do**

• From *parent* search in ccw order for the fisrt non-visited neighbouring cell *free*.

if Spanning tree edge from *current* to *free* is horizontal and sub-cells f^{+45} and f^{+90} are free **then** $free := free^{+90}$.

end if

• Build a spanning tree edge from *current* to *free*.

• Move the tool along the spanning tree edge to the first sub-cell of *free*. The movement depends on the local situation. For double-sided edges the tool moves by Left-Hand-Rule along the edge. For single-sided edges the tool might change to the other (left) side of the spanning tree edge in order to avoid an occupied sub-cell and reach the corresponding sub-cell.

• Call SCSTC(*current*, *free*) auf.

end while

if $current \neq s$ **then**

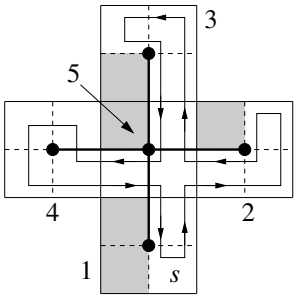
• Move along the spanning tree edge from *current* back to the first possible sub-cell of *parent*. The movement depends on the type of the edge, as mentioned above.

end if

Proof.

Correctness:

Both algorithms construct a spanning tree by DFS such that any 2*D*-cell which has reachable *D* sub-cells will be visited. The tool moves along the spanning tree on both sides – as long as the path is not blocked – and visits all sub-cells that are *touched* by the spanning tree.



Zelle	"Ubergr.	Intern	Gesamt	Randzellen
1	0	1	1	2
2	1	2	3	3
3	1	2	3	3
4	1	1	2	2
5	1	2	3	3

Figure 1.26: Estimating the double visits of sub-cells by STC locally.

Path length:

The number of steps for the tool is essential the sum of the visited sub-cells *C*. If the tool changes to the left side of a spanning tree a detour has to be made and some sub-cells will be visited more than once. Beyond *C* we simply count the number of sub-cells that are visited more than once and locallly charge the sub-cells of a 2*D*-cell for these visits.

We differentiate between *inner* double visits and *intra* double visits. The latter one occur during the movement inside a 2*D*-cell if a sub-cell is visited again. The former one occur if we leave a 2*D* cell *c* along the spanning tree to a neighbouring cell and the corresponding sub-cell was visited before. For this double visit we also charge the 2*D* cell *c*, since it was responsible for the detour.

Any 2*D*-cell *c* is visited for the first time by an incoming spanning tree edge. The inner-cell double visit will occur only if the cell *c* is left again along this edge. Figure 1.26 shows an example for counting inner and intro double visits. For cell 1 sub-cell *s* is visited twice, an intra double visit. The sub-cell above *s* is also visited twice, but by the movement back for 5 to 1 along the spanning tree edge. Therefore 2*D*-cell 5 is charged for this by an inner double visit.

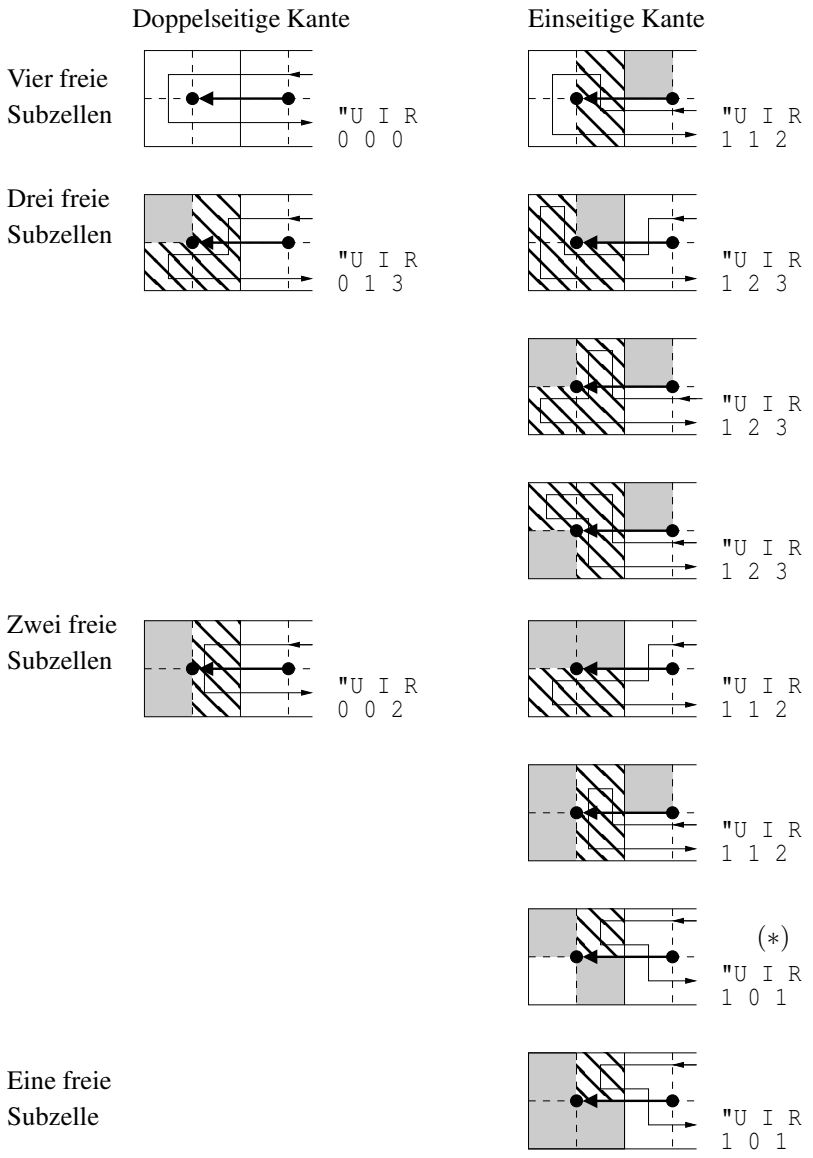


Figure 1.27: Analysis of STC, all possible cases.

The table of2D-cells Figure 1.26 shows the number of inner and intra double visits for any 2D-cell. We charge the 2D-cells for these double visits. On the other hand, for any 2D-cell we also count the number of sub-cells that are diagonally adjacent to a boundary sub-cell. The corresponding boundary sub-cell need not lie inside the 2D-cell itself. Surprisingly, the sum of inner and intra double visits does never exceed the number sub-cells with diagonally adjacent neighbours. This is also given in the table of Figure 1.26.

For a full systematic proof we refer to Figure 1.27. Any 2D-cell c is visited by some spanning tree edge for the first time and the inner double visits can only occur on this edge. Therefore it is sufficient to consider the 2D-cell without other outgoing spanning tree edges. For any intra detours only sub-cells of the current cell are responsible. For the inner detour only the parent cell was responsible.

We distinguish between double sided and single sided edges and between the number of boundary sub-cells inside the corresponding 2D-cell c. We always count inner and intra double visits and compare the sum to the number of sub-cells adjacent to boundary sub-cells.

For all reasonable cases the sum of double visits is always covered locally by the number sub-cells adjacent to boundary sub-cells. The case marked with $(*)$ is a bit tricky. The corresponding 2D cell might also be visited by another spanning tree edge. This is not critical because there is only 1 double

visit in this case for each sub-case. They can be handled separately.

Running time and space requirement
The tool performs at most $C+K \leq 2C$ steps. Any movement is computed locally in $O(1)$ time. The corresponding overall information required does not exceed $O(C)$. □

Finally, we consider the Scan-variants of the STC-Algorithms. We would like to give a rough estimate for the efficiency in avoiding horizontal edges by 2D-Scan-STC.

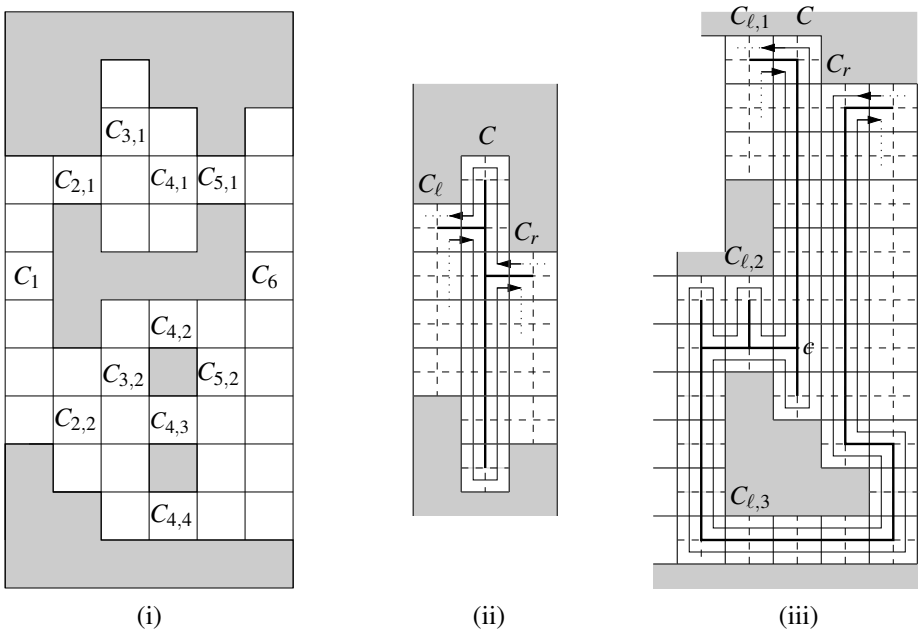


Figure 1.28: (i) Columns and the change of connectivity, (ii) Columns without changes, (iii) Difficult online situation.

We consider columns of the gridpolygon and from left to right we count the change of the connectivity from a column to its neighbour on the right. For example on Figure 1.28(i) there is a numbering of the columns and the number of different vertical components of the columns. From left to right we sum up all differences in the number of components of a column to its neighbour. In Figure 1.28(i) column C_1 has one component and in column C_2 this component split into two components $C_{2,1}$ and $C_{2,2}$. This gives a difference of 1. The components $C_{5,1}$ and $C_{5,2}$ of column C_5 run together in C_6 to a single component. This also is a change of 1 in the difference. Of course also many parts might be involved. We count the changes of any component separately. Let Z denote the sum of all these local changes.
The number Z is a measure for the additional horizontal edges of the spanning tree of Scan-STC against an optimal number of spanning tree edges:

Theorem 1.20 *(Gabriely, Rimon, 2000)*
Let P be a gridpolygon. Let H_{Opt} denote the minimal number of spanning tree edges among all 2D spanning trees of P. Let Z be the above number of connectivity changes for the columns of the 2D-cells.

2D-Scan-STC constructs a spanning tree with at most

$$H_{\text{STC}} \leq H_{\text{Opt}} + Z + 1$$

horizontal edges. *[GR03]*

Proof.(Sketch)
If there is no change in a 2D column, the optimal spanning tree and 2D-Scan-STC will visit and leave the column only once; compare Figure 1.28(ii). The main problem is that by 2D-Scan-STC a connected component of a column will be left by the spanning tree to the same side more than once. This can only happen, if there are changes in the connectivity; see Figure 1.28(iii). □

Concluding remarks

Arkin, Fekete and Mitchell gave some approximation results for the offline exploration of gridpolygons; see [AFM00]. Betke, Rivest und Singh considered a variant of the exploration problem. They introduced the following piecemeal-condition: The agent has to explore an environment with rectangular obstacles and has to return to the start from time to time (charging an accumulator); see [BRS94]. A strategy for this problem for general grid-environments stems from Albers, Kursawe und Schuierer [AKS02].