

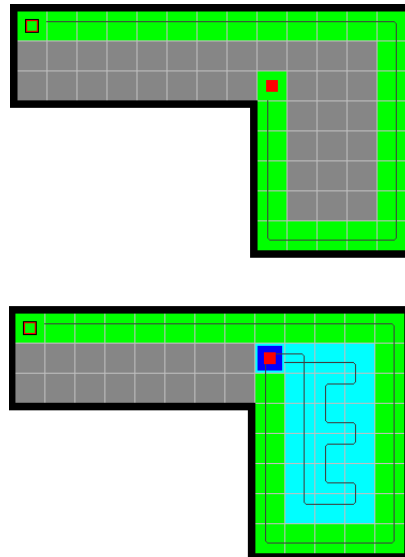
Online Motion Planning MA-INF 1314

Restricted Graphexploration

Elmar Langetepe
University of Bonn

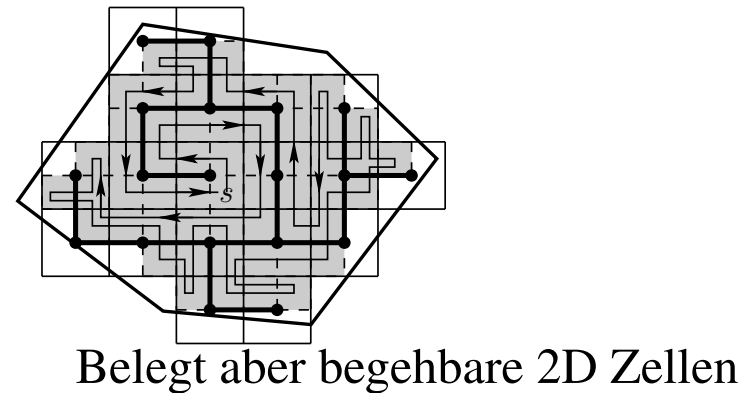
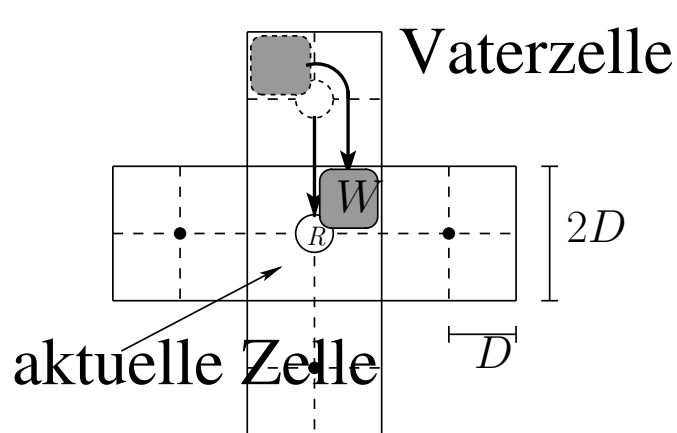
Extension: $\frac{5}{4}$ Algorithm

- Improvements for SmartDFS
- Deadlocks
- Local decomposition
- Run this part optimally
- Offline Problem



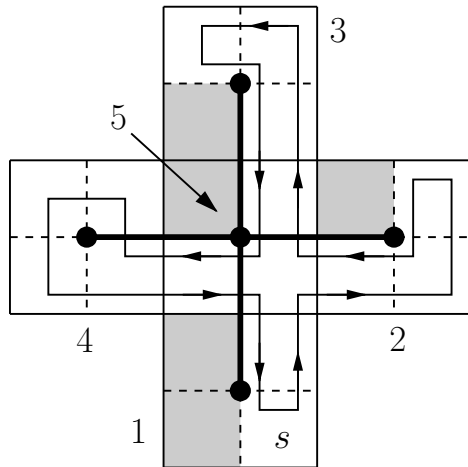
Repetition: Gridpolygons with holes!

- Modell 2D-cells, Spanning-Tree online construction
- SpiralSTC/ScanSTC: Detours along Spanning-Tree edge
- SpiralSTC equivalent to sub-cell-Modell!!
- Algorithmic formulation, recursively defined
- Strategy-Analysis: Locally!



Repetition: Local analysis!

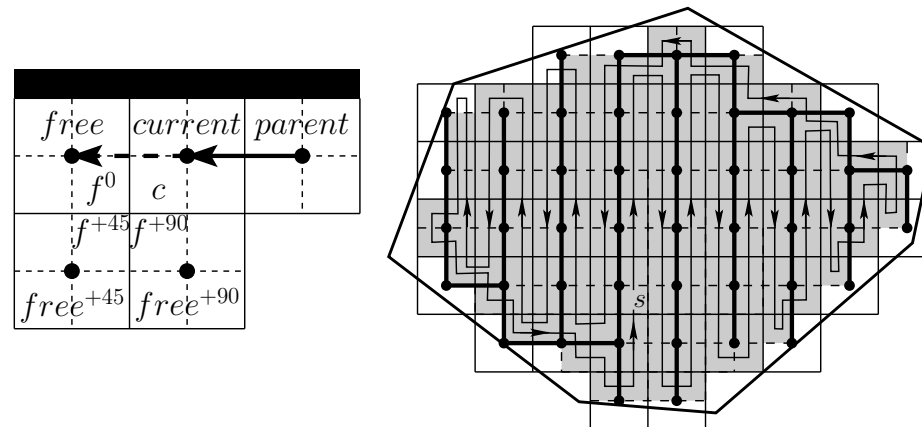
- Count the boundary cells
- Local analysis, multiple visits of cells, charge 2D cell
- Inner-cell (responsibility), Intra-cell
- Systematically: Boundary D -cells \geq inner+intra
- Theorem: $C + K$ (tight!)



Cell	Intra	Inner	Full	Bd.-cells
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

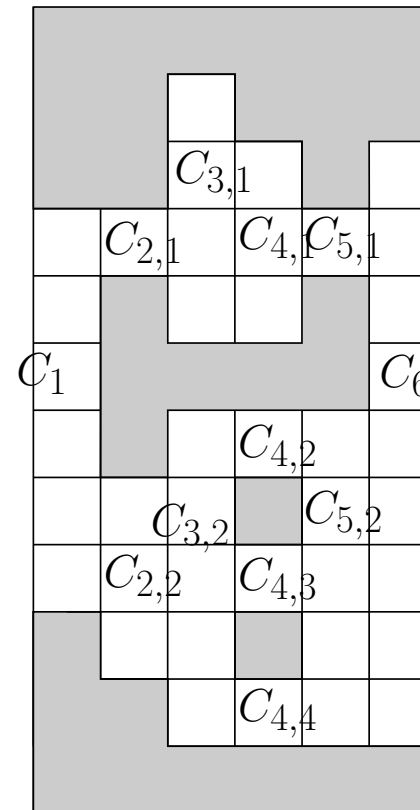
Repetition: Less rotations for the tool

- Avoid spiral-like paths
- ● Move in columns
- Also for the general case/path should exist
- Scan also diagonally adjacent 2D cells
- ScanSTC Algorithm
- Also for the Backtracking step

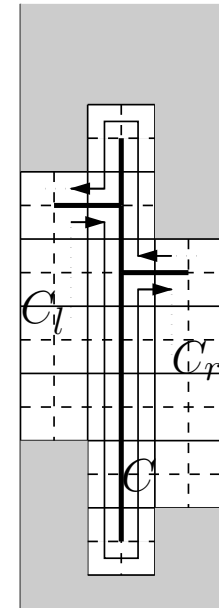


Analysis of 2D-ScanSTC

- Columns connectivity
- From Left to Right X nach Y
- Sum up the Differences: Overall Z
- Connectivity changes



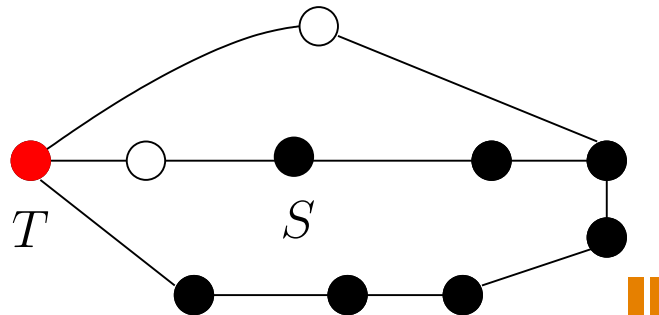
(i)



(ii)

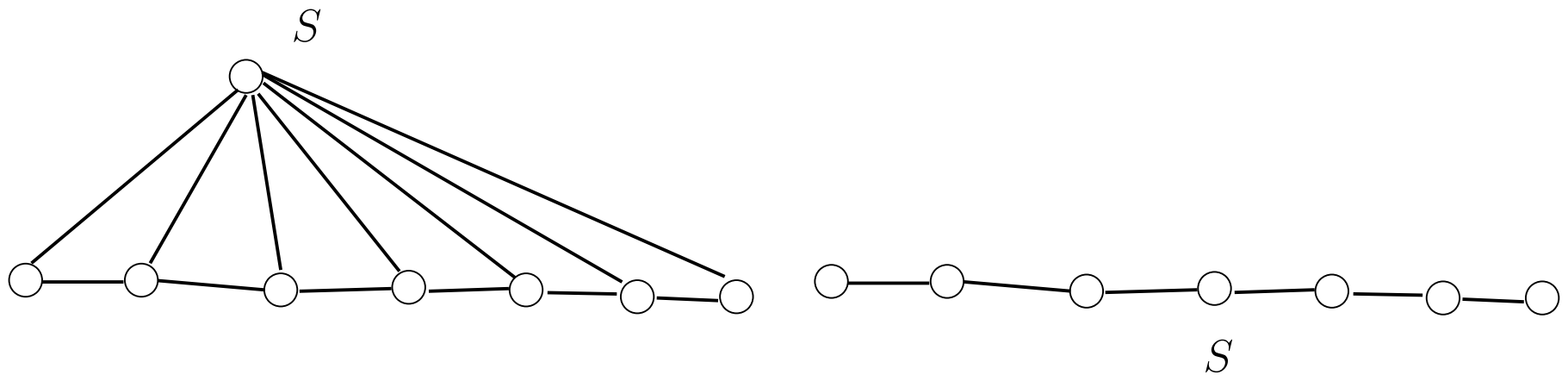
Online graph exploration!

- Graph G : Visit all edges and vertices
- ● DFS 2 competitive, optimal
- Searching \Rightarrow Not too much into the depth
- Restricted exploration, tether/accum. (applications)



Restricted online graph exploration

- Tether of length k
- Graph G : Depth k , longest shortest path to start
- Pure DFS: $k = 1$ but tether length n is required
- BFS: $k \approx n/2$ but $\Omega(n^2)$ visits for n edges



Modell: Restricted (online) graph exploration

1. Tethered agent $l = (1 + \alpha)r$ (cable).■
2. Agent returns to start after $2(1 + \alpha)r$ steps (recharge accumulator)■
3. Large graph, explore up to depth d , flexible d ■
 - All vertices r steps away, depth r (radius)■
 - All edges length 1 (weights, exercise)■
 - Small look-ahead α necessary■
 - First variant, reduction for the others (Lemma/Exercise)■

Restricted graph exploration: Simulation

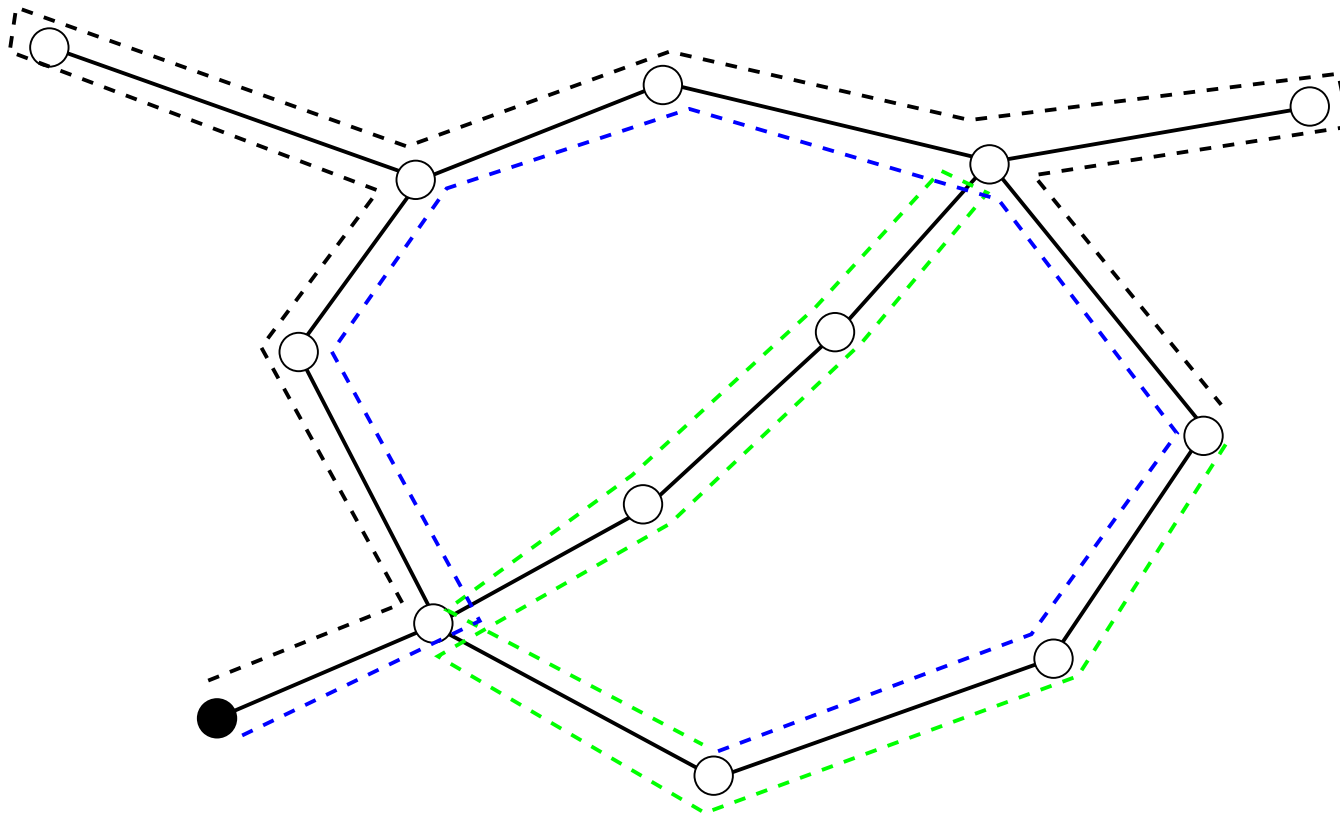
Lemma For any $\beta > \alpha$ a solution for the accumulation-variant with accumulator size $2(1 + \beta)r$ can be attained from the solution of the tethered-variant with tether length $l = (1 + \alpha)r$. The cost decrease by a factor of $\frac{1+\beta}{\beta-\alpha}$.

Proof: Blackboard!!!

Offline Algorithmus: Accumulator-variant

- Offline: Graph is fully known
- Assume: $4r$ Accumulator
- Complexity, (NP-hard ?) unknown! Approximation $O(|E|)!$
- Algorithm: DFS $2|E|$ steps
- Cut into pieces of length $2r$, subpaths
- Starting segment in distance r
- Visit from start, explore subpath, move back!

Example offline!



$$\left\lceil \frac{2|E|}{2r} \right\rceil \times 2r + 2|E| \leq 6|E| \quad \text{Example: } r = 5$$

Offline Algorithm: Accumulator-variant

Lemma A simple Accumulator-Offline Algorithm visits at most $6|E|$ edges. ■

- Reach any subpath-start with step-length $2r$ ■
- Explore all subpath: $2|E|$ ■
- $\left\lceil \frac{2|E|}{2r} \right\rceil$ subpaths in total ■
- Reaching by $\left\lceil \frac{|E|}{r} \right\rceil 2r$ steps ■
- $\left\lceil \frac{|E|}{r} \right\rceil 2r \leq \left(\frac{|E|}{r} + 1 \right) 2r \leq 2|E| + 2r$ ■
- $4|E| + 2r \leq 6|E|$ ■

Online: Tethered graphexploration

- Tether variant (cable), reductions for others (Lemma/Exercise)■
- First idea, DFS (edges) until tether is fully used, then backtracking■
- bDFS, bounded DFS■
- Nice try, is not enough!■

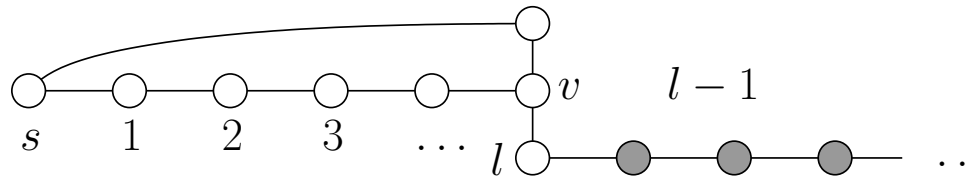
Method: Bounded DFS

bDFS(v, l):

- **if** $(l = 0) \vee$ (all outgoing edges are explored) **then**
 RETURN
end if
- for all** non-explored edge $(v, w) \in E$ **do**
 Move from v to w by (v, w) .
 Mark (v, w) as *explored*
 bDFS($w, l - 1$).
 Move back from w to v by (v, w) .
end for

Bounded DFS

- Example unit-length edge
- Problem: Not all edges will be reached
- Edge to v is marked, End!
- Only bDFS is not enough



CFS Algorithm: Mark the vertices

non-explored vertices, never visited.■

incomplete visited vertices, but there are non-explored **edges** starting at v .■

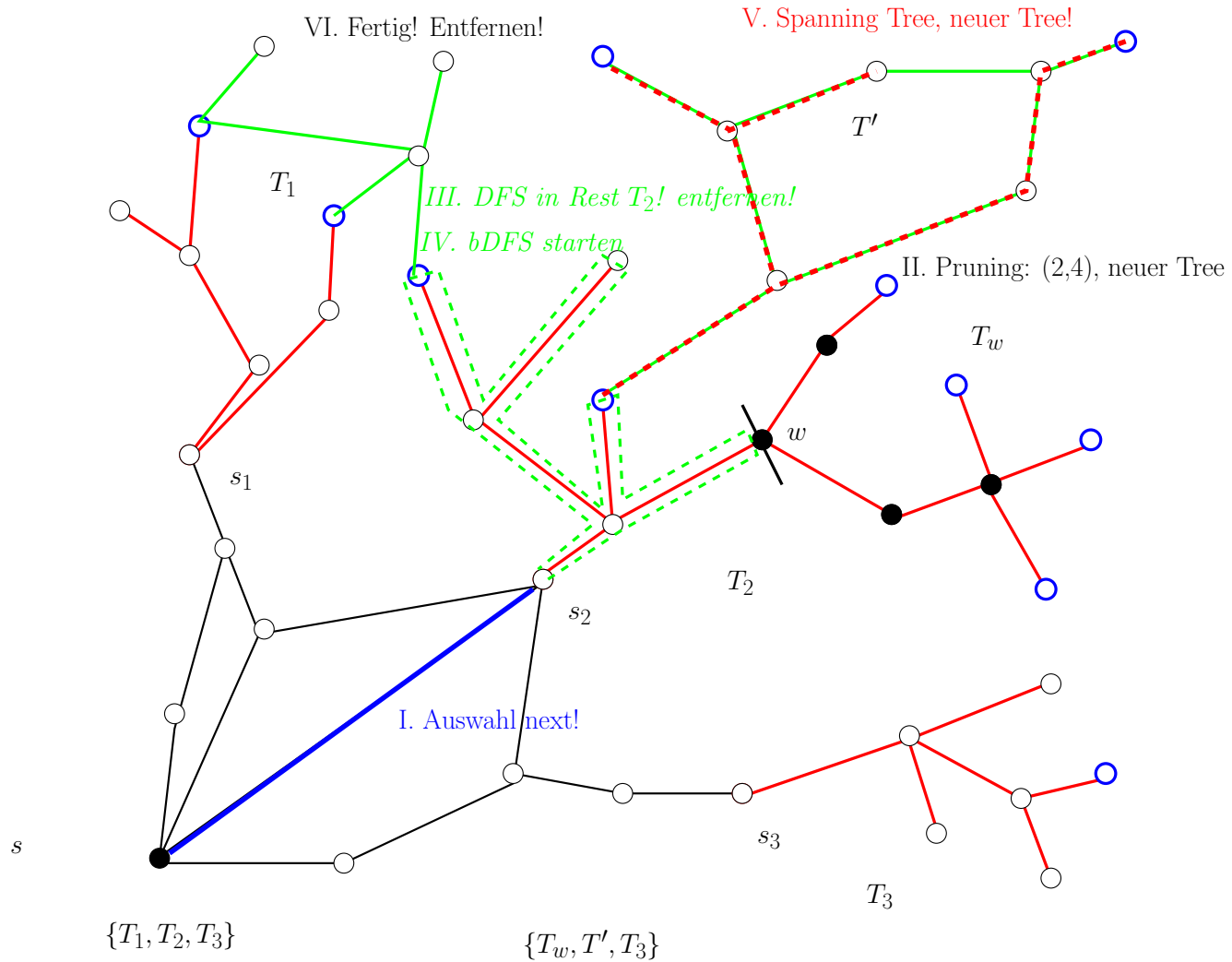
explored vertices, all incident **edges** have been explored.■

CFS Algorithm

- Start bDFS at different sources ■
- Set of (edge) disjoint **trees** $\mathcal{T} = \{ T_1, T_2, \dots, T_k \}$ ■
- Root vertices s_1, s_2, \dots, s_k ■
- Choose T_i with s_i closest to s , move to s_i ■
- Pruning of T_i : Build T_{w_j} with root w_j if: ■
 1. $d_{T_i}(s_i, w_j) \geq \text{minDist} = \frac{\alpha r}{4}$ ■
 2. $\text{Depth}(T_{w_j}) \geq \text{minDepth} - \text{minDist} = \frac{\alpha r}{4}$ ■
- Add all T_{w_j} to \mathcal{T} ! Remove T_i from \mathcal{T} ■
- Explore T_i without T_{w_j} from s_i by DFS and ■
- start bDFS at the incomplete vertices ■

- Graph G' of new vertices and edges ■
- Build a spanning tree T' of G' ■
- Choose root s' with minimal distance to s ■
- Add all these trees to \mathcal{T} ■
- Special case: Trees in \mathcal{T} gets fully explored ■
- Trees in \mathcal{T} with common edges are joined ■
- Merging: Build spanning tree with new root ■

CFS Algorithm, Example



CFS Algorithm

CFS(s, r, α)

■ $\mathcal{T} := \{ \{s\} \}$.

repeat

$T_i :=$ tree in G^* closest to s .

$s_i :=$ root of T_i (closest vertex to s).

$(T_i, \mathcal{T}_i) :=$ **prune**($T_i, s_i, \frac{\alpha r}{4}, \frac{\alpha r}{2}$).

$\mathcal{T} := \mathcal{T} \setminus \{T_i\} \cup \mathcal{T}_i$.

explore($\mathcal{T}, T_i, s_i, (1 + \alpha)r$).

Remove all fully explored trees from \mathcal{T} .

Merge all trees in \mathcal{T} with common vertices.

Calculate spanning tree/root for merged trees.

until $\mathcal{T} = \emptyset$

CFS Algorithmus: Pruning!

prune($T, v, minDist, minDepth$)

■ $v :=$ Root of T .
for all $w \in T$ such that $d_T(v, w) = minDist$ **do**
 $T_w :=$ subtree of T with root w .
 if max. distance from v and vertex in $T_w > minDepth$ **then**
 // Cut-Off T_w from T :
 $T := T \setminus T_w$.
 $\mathcal{T}_i := \mathcal{T}_i \cup \{T_w\}$.
 end if
end for
RETURN (T, \mathcal{T}_i)

CFS Algorithmus: Explore!

explore(\mathcal{T} , T , s_i , l)

- Move from s to s_i along shortest (known) path.
Explore T by DFS. If incomplete vertex v is visited:
 - $l' :=$ remaining tether length.
 - bDFS(v , l').
 - $E' :=$ newly explore edges.
 - $V' :=$ vertices from in E' (plus v).
 - Build spanning tree T' of $G' = (V', E')$.
 - $\mathcal{T} := \mathcal{T} \cup \{T\}$.
- Move back from s_i to s .

CFS Algorithmus: Example!!

- $G^* = (V^*, E^*)$ Graph of the explored edges and and vertices
- (successively extended)
- Set \mathcal{T}
- Pruning
- Explore (DFS/bDFS)