Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik I

Elmar Langetepe

Online Motion Planning

MA INF 1314

Summersemester 2016

Manuscript: Elmar Langetepe

1.5 Constrained graph-exploration

We consider the problem of the exploration of an unknown graph $G = (V,E)$ starting from some fixed vertex $s \in V$. This means that we would like to visit all edges and vertices of G. First, we consider unit-weights which means that any visit of an edge has cost 1. Different from the previous section we consider a constrained version of the exploration, due to the following practical models. Let r denote the radius or depth of the graph w.r.t. s. I.e., r is the maximal length of a shortest path from s to some vertex $v \in V$. Let us first assume that r is known, but not the graph itself.

1. The agent is bounded by a tether of length $\ell = (1+\alpha)r$ (for example a cable constraint).
2. The agent has to return to the start after any $2(1+\alpha)r$ steps (for example an accumulator has to be recharged).
3. A large graph should be explored up to a given fixed depth d (for example for searching a close by target).

The above third variant will be applied to a searching heuristic with increasing depth, later. First, we show some simple simulation resutls. If an algorithm for the tether variant is known, one can also establish an accumulator strategy with some extra cost.

Lemma 1.21 *Given a tether variant strategy with tether length $l = (1+\alpha)r$ and overall cost T. For any $\beta > \alpha$ there is an accumulator-strategy with cost $\frac{1+\beta}{\beta-\alpha}T$*

Proof. We design the accumulator strategy by following the tether strategy. After any $2(\beta-\alpha)r$ steps we move back from the current vertex v to the start, recharge the agent and move back to v. Then we proceed with the next step of length $2(\beta-\alpha)r$ of the tether strategy path. In the tether strategy for any vertex v, we are never more than $(1+\alpha)r$ away from the start. That is $2(\beta-\alpha)r + 2(1+\alpha)r = 2(1+\beta)r$ always result in correct loops. The strategy is correct.

On the other hand, we have cost T for following the tether path and additional cost for moving back and force. We move back at most $\frac{T}{2(\beta-\alpha)r}$ times and require $2(1+\alpha)r$ steps for any movement. This gives total cost:

$$T + \frac{T}{2(\beta-\alpha)r} \cdot 2(1+\alpha)r \quad = \quad T\frac{\beta-\alpha+1+\alpha}{\beta-\alpha} = \frac{1+\beta}{\beta-\alpha}T\,.$$

□

Exercise 9 *Given an accumulator strategy S with accumulator size $2(1+\beta)r$ and overall cost T. For a given $\alpha > \beta$ design an efficient tether strategy that makes use of S so that the cost of the tether strategy is $f(\alpha,\beta)\cdot T$. Determine $f(\alpha,\beta)$ precisely.*

We can also consider the Offline-variant of the problem. In this case the graph is fully known. To the best of our knowledge the complexity of the Offline-variant (computing the best strategy) is not known. Since in the case that the tether is very long, the Hamiltonian-path problem appears to be part of the problem, the problem is assumed to be NP-hard.

If the optimal Offline-strategy is not known, we can design an Offline-strategy that approximates the optimal strategy. We consider the accumulation variant and assume that the accumulator has size $4r$.

Lemma 1.22 *Let us assume that an accumulator of size $4r$ is given. There is a simple Offline algorithm that explores a graph of depth r with no more than $6|E|$ steps.*

Proof. We consider the DFS walk among the edges of the graph which requires $2|E|$ steps. Now we split this overall path into pieces of size $2r$. Similarly to the simulation in the proof above we successively move to the start vertices of these subpaths, follow the DFS path for $2r$ steps and return to the start after that. In total the accumulator of size $4r$ is sufficient. Moving along the DFS path gives $2|E|$ steps. There are no more than $\left\lceil \frac{2|E|}{2r} \right\rceil$ sub-paths that require no more that $\left\lceil \frac{|E|}{r} \right\rceil 2r$ steps in total. We have $\left\lceil \frac{|E|}{r} \right\rceil 2r \leq \left(\frac{|E|}{r}+1\right) 2r \leq 2|E|+2r$ which shows that $4|E|+2r \leq 6|E|$ is sufficient. □

From now on we consider only the tether variant, for the accumulation variant similar results can be shown. A first simple idea is to take the tether length for the DFS walk into account.

Just performing DFS is not always possible. A BFS approach is always possible but results in too many exploration steps; see Figure 1.29. Therefore we apply DFS with the tether restriction as given in Algorithm 1.9. There is a backtracking step, if the tether is fully exhausted. We call this algorithm bDFS for bounded DFS.

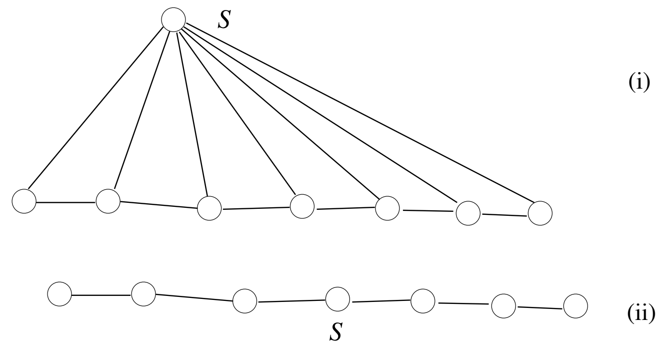

Figure 1.29: (i) A Graph with n vertices and with depth $r = 1$, pure DFS would require a tether of length $n-1$. (ii) A graph of depth n, BFS with a tether of length n requires $\Omega(n^2)$ steps.

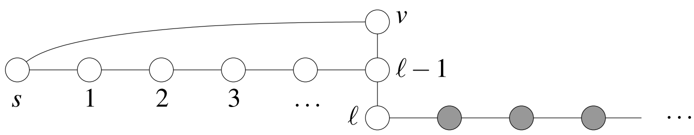

Figure 1.30: bDFS kann einige Knoten nicht erreichen.

Algorithm 1.9 boundedDFS

bDFS(v**,** ℓ **)**:
  if $(\ell = 0) \vee$ (no adjacent non-explored edges) **then**
    RETURN
  end if
  for all non-explored edge $(v,w) \in E$ **do**
    Move from v to w along (v,w).
    Mark (v,w) as *explored*.
    bDFS(w, $\ell - 1$).
    Move back from w to v along (v,w).
  end for

In general bDFS is not sufficient for the full exploration of a graph. For example in Figure 1.30 we have the problem that the dark-colored vertices cannot be reached, if the algorithm first chooses the path along the vertices $1,2,\ldots,\ell-1$, visits vertex l, v and s and winds back to the start s. The path from s over v is short enough but will not be considered by bDFS.

Therefore we would like to call bDFS from different sources. The aim is to achieve a constant competitive algorithm. In Algorithm 1.10 we maintain a set of (edge) disjoint trees $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ with root vertices $s_1, s_2, \ldots, s_k$, respectively. The trees still contain incomplete vertices where not all adjacent edges have been visited. We choose a tree T_i with the minimal distance from s to root s_i among all trees of $\mathcal{T}$. From this tree we prune subtrees T_{w_j} with root vertices w_j, so that w_j is a certain distance ($minDist = \frac{\alpha r}{4}$) away from s and T_{w_j} has a certain minimal depth (determined over $minDepth = \frac{\alpha r}{2}$). Those trees will be inserted into $\mathcal{T}$. The pruning forces the trees of $\mathcal{T}$ to have a minimum size, it is still worth visiting them.

After pruning, the rest of T_i will be explored by DFS and if an incomplete vertex will be found, we start bDFS with the current remaining tether length for the exploration of *new* edges. The newly explored edges and vertices build a graph G'. If G' has incomplete vertices, we construct a spanning tree T' with a root vertex s', where s' is the vertex in T' closest to s in the current overall explored graph G^*. T' will be inserted into $\mathcal{T}$. After the overall DFS (and bDFS) walk in T_i we delete all trees of $\mathcal{T}$ that are now fully explored. Some of the trees in $\mathcal{T}$ might have common vertices. We merge those trees and build a new spanning tree for them with a new root vertex.

A scheme of the algorithm is shown in Figure 1.31. We have done the prune step by values $(2,4)$. Otherwise, we have to build very large example graphs.

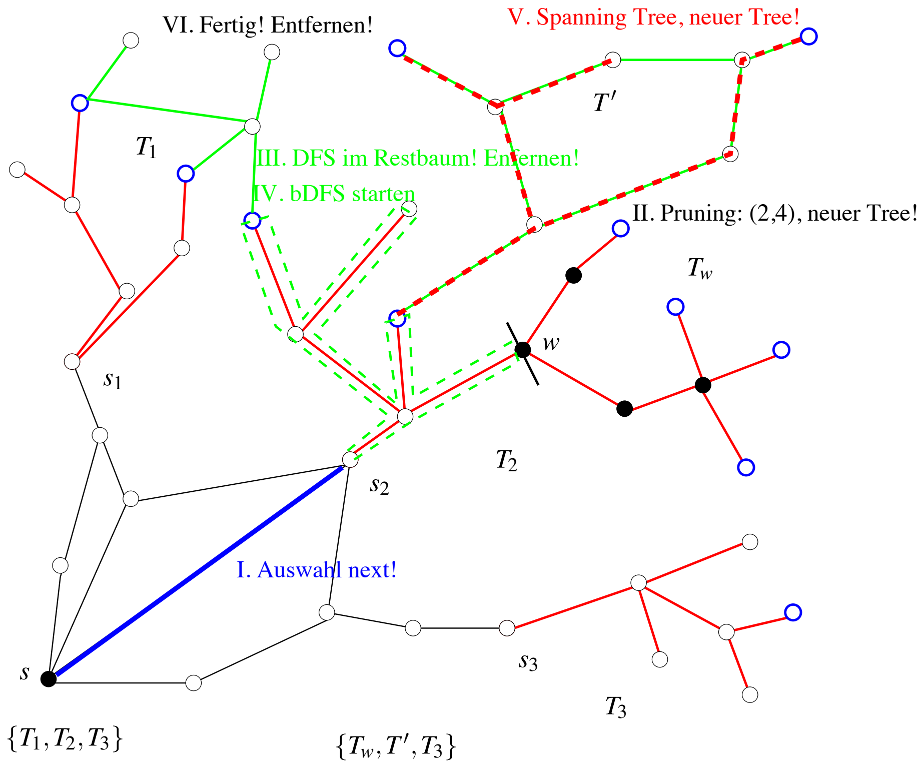

Figure 1.31: The algorithm maintains a set of disjoint trees $\mathcal{T} = \{T_1, T_2, T_3\}$ and choose the tree T_2 with minimal distance $d_{G^*}(s, s_i)$. After that the tree is pruned. Subtrees of distance 2 away from s_2 with vertices inside that have distance at least 4 from s_2 are cut-off. After that DFS starts on the rest of T_2 and starts bDFS on the incomplete vertices. Here some new graphs G' will be explored and we build spanning trees T' for them. Some trees in $\mathcal{T}$ get fully explored. T_w und T' are added to $\mathcal{T}$, the tree T_2 is deleted.

In the following let $\mathrm{d}_{G'}(v, w)$ denote the distance between vertices v and w in the subgraph or tree G'. $G^* = (V^*, E^*)$ denotes the currently known part of G.

The algorithm makes us of the following subdivision of vertices:

non-explored a vertex, which was never been visited before.

incomplete a vertex already visited before but some of the adjacent edges are still non-explored.

Algorithm 1.10 CFS

CFS(s**,** r**,** α **)**

$\mathcal{T} := \{\{s\}\}$.
repeat
$T_i :=$ closest subtree of $\mathcal{T}$ to s in G^*.
$s_i :=$ vertex of T_i closest to s n G^*.
$(T_i, \mathcal{T}_i) :=$ 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($\mathcal{T}, T_i, s_i, (1+\alpha)r$).
Delete fully explored trees from $\mathcal{T}$.
Merge the trees of $\mathcal{T}$ with common vertices.
Define a root vertex closest to s in G^*.
until $\mathcal{T} = \emptyset$

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

$v :=$ Wurzel von T.
$\mathcal{T}_i := \emptyset$.
for all $w \in T$ with $d_T(v,w) = minDist$ **do**
$T_w :=$ subtree of T with root w.
if maximale Distanz between v and a vertex in $T_w > minDepth$ **then**
// Cut-Off T_w from T ab:
$T := T \setminus T_w$.
$\mathcal{T}_i := \mathcal{T}_i \cup \{T_w\}$.
end if
end for
RETURN $(T, \mathcal{T}_i)$

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

Move from s to s_i along shortest path in G^*.
Explore T by DFS. If incomplete vertex occurs, do:
$\ell' :=$ remaining tether length.
bDFS(v, ℓ').
$E' :=$ set of newly explored edges.
$V' :=$ set of vertices of E'.
Calculate spanning tree T' for $G' = (V', E')$.
Define root vertex of T' closest to s in G^*
$\mathcal{T} := \mathcal{T} \cup \{T'\}$.
Move back from s_i to s.

explored a vertex, that was visited and all adjacent edges have been explored.

Additionally, for the bDFS walk we mark the edges as 'non-explored' or 'explored'.

Lemma 1.23 *The following properties hold during the execution of the CFS–Algorithm:*
(i) Any incomplete vertex belongs to a tree in $\mathcal{T}$.
(ii) Until $G^* \neq G$, there is always an incomplete vertex $v \in V^*$ so that $d_{G^*}(s,v) \leq r$.
(iii) For any chosen root vertex s_i: $d_{G^*}(s,s_i) \leq r$.
(iv) After pruning T_i is fully explored by DFS. All trees $T \in \mathcal{T}$ have size $|T| \geq \frac{\alpha r}{4}$.
(v) All trees $T \in \mathcal{T}$ are disjoint (w.r.t. edges).

Proof.
(i) Follows directly from the construction of the trees by bDFS and Pruning. No incomplete vertex is missing.
(ii) Assume that for all $v \in V^*$ we have $\mathrm{d}_{G^*}(s,v) > r$ and let v be an incomplete vertex of V^*. In G there is a shortest path $P(s,v)$ from s to v with length $\leq r$. Along $P(s,v)$ there is a first vertex w that does not belong to G^*. Thus its predecessor w' along $P(s,v)$ belongs to V^* and is incomplete. We have $\mathrm{d}_{G^*}(s,w') \leq r$.
(iii) Follows from (ii), the root of a corresponding tree T is always the vertex of T closest to s.
(iv) We show the property by successively considering the upcoming trees. Or by induction on the number of pruning steps. In the beginning the algorithm starts with bDFS at the root s. Either, the graph will be fully explored and we are done, or bDFS have exhausted the tether of length $(1+\alpha)r$ and have visited more than $(1+\alpha)r$ edges. The single spanning tree T has size $|T| \geq (1+\alpha)r > \frac{\alpha r}{4}$. Let us assume that the condition holds for the trees inside $\mathcal{T}$ and the next pruning step happens. Now by the next iteration we are choosing tree T_i with root s_i closest to s among all trees in $\mathcal{T}$. After that we prune T_i. The rest of T_i has still size $|T_i| \geq \frac{\alpha r}{4}$ since we cut off subtrees T_w with distance $\geq \frac{\alpha r}{2}$ away from s_i. For a corresponding subtree T_w we conclude $|T_w| \geq \frac{\alpha r}{2} - \frac{\alpha r}{4} = \frac{\alpha r}{4}$ since there is a vertex inside T_w that is at least distance $\frac{\alpha r}{2}$ away from s. Now consider the remaining DFS/bDFS combination on (the rest of) T_i. The distance from s to s_i is at most r.
Any incomplete vertex in the current T_i has at most distance $\frac{\alpha r}{2}$ from s_i otherwise this vertex would be part of a tree T_w that has to be considered in the pruning step. This means that at any incomplete vertex there is a rest tether of length $\frac{\alpha r}{2}$ which can be used for the bDFS part. If the exploration results in another spanning tree T' with incomplete vertices, this tree has size at least $\frac{\alpha r}{2}$.
Finally fully explored trees are deleted from $\mathcal{T}$ which is not critical. Additionally, some other trees might be merged and still have incomplete vertices. These trees only grow.

□

Finally, we show:

Theorem 1.24 *(Duncan, Kobourov, Kumar, 2001/2006)*
The CFS–Algorithm for the constrained graph-exploration of an unknown graph with known depth is $(4+\frac{8}{\alpha})$–competitive. *[DKK06, DKK01]*

Proof. We split the cost for any appearing subtree T_R. Let $K_1(T_R)$ denote the cost for moving from s to s_i in G^*. Let $K_2(T_R)$ denote the cost of DFS for T_R and let $K_3(T_R)$ denote the cost for the bDFS exploration done for the incomplete vertices starting at T_R. The trees are edge disjoint.

The total cost is a sum of the cost for any T_R. We have

$\sum_{T_R} K_3(T_R) \leq 2 \cdot |E|$, since bDFS only visits non-explored edges (twice).

$\sum_{T_R} K_2(T_R) = \sum_{T_R} 2 \cdot |T_R| \leq 2 \cdot |E|$, the cost for all DFS walks.

For $K_1(T_R)$ we have $K_1(T_R) = 2 \cdot \mathrm{d}_{G^*}(s,s_i) \leq 2r$. The complexity of any T_R is at least $\frac{\alpha r}{4}$ which gives $|T_R| \geq \frac{\alpha r}{4}$ for the number of edges. We conclude $r \leq \frac{4|T_R|}{\alpha}$ and

$$\sum_{T_R} K_1(T_R) \leq \sum_{T_R} 2r \leq \frac{8}{\alpha} \sum_{T_R} |T_R| \leq \frac{8}{\alpha}|E|$$

Index

P

Q

R

S

T

W