RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT BONN
INSTITUT FÜR INFORMATIK I

Elmar Langetepe

Online Motion Planning

MA INF 1314

Summersemester 2016
Manuscript: Elmar Langetepe

Altogether, the algorithm makes $(4+\frac{8}{\alpha})|E|$ step whereas any optimal algorithm visits at least any edge once. □

In general we assume that α is a small constant with $0 < \alpha < 1$. The above proof works for any $\alpha > 0$. The cost of the algorithm for known depth r are within $O(|E|/\alpha)$. More precisely we can show that actually $O(|E|+|V|/\alpha)$ steps are made. For this we have a closer look at the cost. bDFS work on the edges only. The DFS walk work on trees where the number of vertices is the same as the number of edges. Some of these vertices appear in two trees, so by a factor of 2 we are on the save side. The movements from s to s_i are analysed over the size of spanning trees, where vertices and edges are also the same.

The cost $K_1(T_R)$ and $K_2(T_R)$ sum up to $\left(2+\frac{8}{\alpha}\right)2|V|$.

Altogether there is an $\Theta(|E|+|V|/\alpha)$ algorithm for the exploration of arbitrary graphs.

Corollary 1.25 *The CFS–Algorithm for the constrained graph-exploration of an unkown graph with known depth has optimal exploration cost* $\Theta(|E|+|V|/\alpha)$.

Now we have some possibilities for extensions. First, we assume that the depth of the graph is unknown in the beginning. Next we would like to consider weighted edges.

1.5.1 Restricted graph-exploration with unknown depth

Let is now assume that the radius, say R, of the unknown graph G is not known. From a practical point of view, spending some cable is costly and we would like to extend the tether only if it is necessary. A first simple idea is that we guess the depth, say r, and successively double its length until the algorithm finally explores the whole graph. Obviously, the repeated application of the CFS-algorithm runs in $O(\log r|E|)$ step. As shown above we can also refine the analysis of this approach. For any bDFS step we make use of the already visited edges and directly *jump* to incomplete vertices (now with larger tether length). Therefore the bDFS steps are still subsumed by $2|E|$ steps. But we still have to take the movements to the roots of the trees into account as well as the DFS movements on the new subtrees. Therefore we have the following result.

Corollary 1.26 *Applying the CFS–Algorithmus by successively doubling the current depth r gives an algorithm that explores an unknown graph G with unknown depth R with* $\Theta(|E|+(\log R)|V|/\alpha)$ *steps.*

We will now show that we can get rid of the log-factor by successively adjusting r appropriately. We only exchange two calls in the main procedure. In principle, instead of the known value r we successively make use of $r := d_{G^*}(s,s_i)$, which is the smallest distance from s to one of the roots of the trees in $\mathcal{T}$.

More precisely, we exchange prune(T_i, s_i, $\frac{\alpha r}{4}$, $\frac{\alpha r}{2}$) wird by prune(T_i, s_i, $\frac{\alpha d_{G^*}(s,s_i)}{4}$, $\frac{9\alpha d_{G^*}(s,s_i)}{16}$) and explore($\mathcal{T}$, T_i, s_i, $(1+\alpha)r$) by explore($\mathcal{T}$, T_i, s_i, $(1+\alpha)d_{G^*}(s,s_i)$). This means that the pruning-step is done with the values $\frac{\alpha d_{G^*}(s,s_i)}{4}$ and $\frac{9\alpha d_{G^*}(s,s_i)}{16}$ and the eplore-step is done with tether length $(1+\alpha)d_{G^*}(s,s_i)$.

In the beginning we have $d_{G^*}(s,s_i) = 0$, therefore we make use of some fixed constant c in the beginning and use $r := max(d_{G^*}(s,s_i),c)$. Let $d_{G^*}(s,T)$ denote the shortest distance from s to some vertex in T inside G^*.

Lemma 1.27 *For the CFS–Algorithmus with unknown depth R we have the following properties:*

(i) Any incomplete vertex belongs to a tree in $\mathcal{T}$.

(ii) There is always an incomplete vertex $v \in V^*$ *with* $\mathrm{d}_{G^*}(s,v) \le r$, *until* $G^* \neq G$.

(iii) For the closest root s_i *we have:* $\mathrm{d}_{G^*}(s,s_i) \le r$.

(iv) For all trees $T \in \mathcal{T}$ *we have* $|T| \ge \frac{\max(d_{G^*}(s,T),c)\alpha}{4}$. *After pruning the remaining tree will be fully explored by DFS.*

(v) All trees ever considered in $\mathcal{T}$ *are (edge) disjoint.*

Proof. For the proof of (i),(ii),(iii) and (v) we apply the same arguments as in the proof of Lemma 1.23. It remains to show that (iv) holds. The main difference is that the size of a tree T is directly correlated to the distance from s to T, this is different from the previous argumentation.

Let us first show that the remaining tree T_i (after pruning) will be fully explored by DFS. For any vertex v in T_i we have $d_{T_i}(s_i,v) \leq \frac{9d_{G^*}(s,s_i)\alpha}{16}$, otherwise v has been cut of by pruning. Thus we have

$$(1+\alpha)d_{G^*}(s,s_i) - d_{G^*}(s,s_i) - d_{T_i}(s_i,v) \geq \frac{7d_{G^*}(s,s_i)\alpha}{16},$$

which shows that the tether is long enough T_i will be fully explored by DFS.

By induction over the number of pruning steps we will finally show: $\forall T \in \mathcal{T} : |T| \geq \frac{\max(d_{G^*}(s,T),c)\alpha}{4}$.

In the beginning we apply bDFS from the start with tether length c. Either we explore the whole graph or we have $|T| \geq (1+\alpha)c > \frac{\alpha c}{4}$ for the resulting spanning tree T. For simplicity we assume $d_{G^*}(s,T_i) > c$ from now on.

We would like to show that for any tree T_w, resulting from the pruning of some T_i, we have $|T_w| \geq \frac{d_{G^*}(s,T_w)\alpha}{4}$. Also the remaining tree T_i has this property.

For the remaining tree T_i (after pruning), we conclude $d_{G^*}(s,T_i) = d_{G^*}(s,s_i)$ and pruning guarantees $|T| \geq \frac{d_{G^*}(s,T)\alpha}{4}$. For a tree T_w pruned from T_i we have: $|T_w| \geq \frac{9d_{G^*}(s,s_i)\alpha}{16} - \frac{d_{G^*}(s,s_i)\alpha}{4} = 5\frac{d_{G^*}(s,s_i)\alpha}{16}$ by the pruning values. Additionally, we have $d_{G^*}(s,T_w) \leq d_{G^*}(s,s_i) + d_{G^*}(s_i,w) = (1+\frac{\alpha}{4})d_{G^*}(s,s_i)$, since the root w of T_w is exactly $\frac{\alpha d_{G^*}(s,s_i)}{4}$ steps away from s. Für $0 < \alpha < 1$ we conclude: $d_{G^*}(s,T_w) < \frac{5d_{G^*}(s,s_i)}{4}$ and together with the above inequality we have $|T_w| > \frac{d_{G^*}(s,T_w)\alpha}{4}$.

Finally, we have to analyse the emerging spanning trees T_v, which will be constructed from the bDFS steps starting during the DFS walk in T_i. Such a tree T_v *starts* at some incomplete vertex v in T_i. We have $d_{G^*}(s_i,v) \leq \frac{9\alpha d_{G^*}(s,s_i)}{16}$, otherwise v would have been pruned and could not be a leaf of the rest of T_i any more. Thus we have $d_{G^*}(s,T_v) \leq d_{G^*}(s,s_i) + d_{G^*}(s_i,v) < \frac{25d_{G^*}(s,s_i)}{16}$ or $d_{G^*}(s,s_i) > \frac{16d_{G^*}(s,T')}{25}$. If T_v is fully explored, we are done, since the tree will be deleted. Assume that T_v still has incomplete vertices. As mentioned above we have $d_T(s_i,v) \leq \frac{9\alpha d_{G^*}(s,s_i)}{16}$. Starting from v there was a remaining tether length of $\frac{7\alpha d_{G^*}(s,s_i)}{16}$ for the construction of the incomplete T_v, which gives $|T_v| \geq \frac{7\alpha d_{G^*}(s,s_i)}{16}$. Application of $d_{G^*}(s,s_i) > \frac{16d_{G^*}(s,T_v)}{25}$ gives $|T_v| > \frac{7\alpha d_{G^*}(s,T_v)}{25} > \frac{d_{G^*}(s,T_v)\alpha}{4}$. Either we have explored everything behind v or the spanning tree T_v has size $|T_v| > \frac{d_{G^*}(s,T_v)\alpha}{4}$.

We have considered any emerging $T \in \mathcal{T}$! □

Theorem 1.28 *(Duncan, Kobourov, Kumar, 2001/2006)*
Applying the CFS–Algorithm with the adjustments above results in a correct restricted graph-exploration of an unknown graph with unknown depth. The algorithm is $(4+\frac{8}{\alpha})$*–competitive. [DKK06, DKK01]*

Proof. We apply the same analysis as in the proof of Theorem 1.24. For the analysis of the movements from s to the roots of the trees we make use of the correlation $|T_R| > \frac{d_{G^*}(s,T_R)\alpha}{4}$. □

For the number of steps we can also refine the analysis, analogously.

Corollary 1.29 *The above CFS–Algorithm for the restricted exploration of an unknown graph with unknown depth requires* $\Theta(|E| + |V|/\alpha)$ *exploration steps, which is optimal.*

Finally, we would like to argue that the usage of a look-ahead of αr is necessary for attaining linear optimal exploration cost (i.e., in comparison to $|E|$ and $|V|$. This can be shown for the accumulator variant as follows. First, it is clear that an accumulator of size $2r$ is not sufficient for exploring all edges. The graph in Figure 1.32 has depth 6, but exploring all edges requires an accumulator of size 13.

This means that an accumulator size $2r+1$ is necessary. We show that an accumulator of size $2r+d$ for constant d is not sufficient in the sense of performing no more than $C \cdot |E|$ exploration steps.

Lemma 1.30 *For the accumulator variant with accumulator size* $2r+d$ *for constant* d*, there are examples do that any algorithm attains at least* $\Omega\left(|E|^{\frac{3}{2}}\right)$ *exploration steps.*

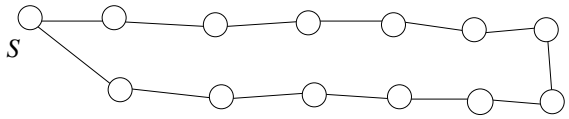

Figure 1.32: A graph of depth $r = 6$ that cannot be explored by an accumulator of size $2r$.

Proof. We consider the following example as given in Figure 1.33. Starting from s there is a path of length $\frac{n}{2}$ that visits a clique of size $\frac{n}{2}+1$. Moving forth and back along the path requires n steps, the depth of the graph is $\frac{n}{2}+1$. Exploration with accumulator size $n+2+d$ means that we have to visit the clique $\Omega\left(\frac{n^2}{d}\right)$ times since the clique has $\Omega\left(n^2\right)$ edges. This gives $\Omega\left(\frac{n^2}{d}\cdot n\right) = \Omega\left(n^3\right)$ exploration steps. The statement follows from $|E| \in \Theta(n^2)$. □

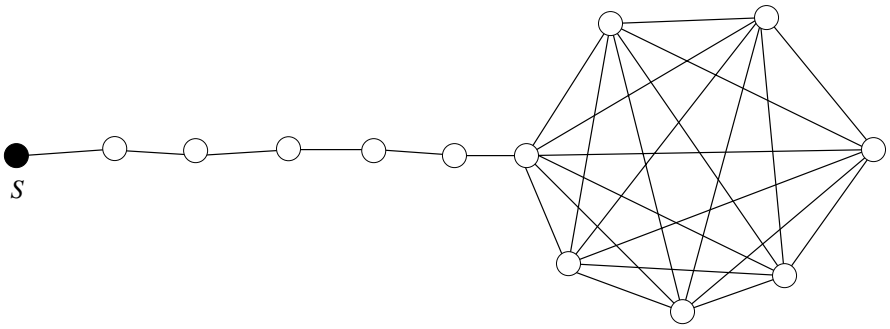

Figure 1.33: A graph with $n+1 = 13$ vertices. A path of length $\frac{n}{2}$ visits a clique of size $\frac{n}{2}+1$. Any accumulator strategy with accumulator size $n+2+d$ requires $\Omega(n^3)$ steps.

With a similar argument we conclude that an sub-linear extension of the accumulator, i.e., size $2r+o(r)$, is not sufficient for attaining a linear cost strategy. Let us briefly repaet the small-o notation. For real valued functions or series f and g we define $f \in o(g)$, if and only if $\lim_{r\to\infty} \frac{f(r)}{g(r)} \to 0$ holds. Therefore we conclude $r \in o(r^2)$, $c \in o(r)$ for any constant c and also $\frac{1}{r} \in o(1)$. By the above arguments and example we can show that $\Omega\left(\frac{n^3}{f(n)}\right)$ exploration steps are necessary for an accumulator of size $n+2+f(n)$. For $f(n) = n^{1-\varepsilon}$ (this means $f \in o(n)$) we have to perform $\Omega(|E|^{1+\varepsilon})$ exploration steps.

Note, that for the tether variant up to our knowledge there is no such statement that a tether of length $r+o(r)$ is necessary for attaining $O(|E|)$ exploration cost.

We have shown that we can explore any graph (online and offline) with at most $\Theta(|V|+|E|)$ exploration steps. These are the pure cost for the motion of the agent. In the literature this is also denoted as the *mechanical cost*; see also [DJMW91]. Besides, there are also some *computational cost*, for the planning and preparation of the strategy.

For example the computational cost of the CFS-Algorithm have to be analysed for the following tasks:

- Build the spanning trees
- Update the shortest paths to the trees of $\mathcal{T}$
- Merge the trees
- Detect fully explored trees
- Prune a tree
- Maintain the list $\mathcal{T}$
- Apply DFS/bDFS

For unit-lenght edges some of the above tasks can be done very efficiently. The overall approach can be easily extended to weighted graphs (positive edge weights).

Exercise 10 *Analyse the computational cost for the CFS-Algorithm in O−notation for $|E|$ and/or $|V|$.*

Exercise 11 *Show that the CFS-Algorithm approach also works for graphs with positive edges weights. How do we have to adjust the CFS-Algorithm?*

1.5.2 Mapping of an unknown graph

Finally, in this section we would like to show the influence of different capabilities of the agent. Up to now we assumed that an already visited vertex or edge will be recognized at the next visit. This means that we have marked any visited edge and vertex.

Let us now assume that the agent cannot mark parts of the environment. We do not have any *landmarks*. We still assume that we have enough storage for constructing the sub-graph detected so far.

The following model is taken from Dudek et al.; see[DJMW91]. The agent has no orientation and no compass. At any vertex the outgoing edges are presented in the same order. This order need not represent a planar embedding. If the agents visits the vertex from different incoming edges, the order will be consistent. This means that there is a fixed cyclic order, the relative presentation of the order stems from the edge where the agent currently comes from. Figure **??** shows an example of a relative order. By this order, the agent knows where he was coming from and can also return to this vertex. Since the storage is not limited, it is possible to remember a return path. Let us for example assume that the agent visits vertex v_2 by edge e_1 and then visits the second edge e_3 in ccw-order from e_1. If the agent moves back along e_3 to v_2, it already knows that it was recently coming from the first edge in ccw order, which is e_1. The agent can make use of this return path. If the agent visits a vertex in a forward step, it has no idea which of the vertices the visited vertex actually is.

Is it possible to build a map of the graph and to locate oneself inside the graph? The offline input is a triple $G = (V,E,S)$, where by S for any vertex the cyclic local order of the edges is given.

First, it is easy to see that without further capabilities, one can not fully detect a given graph. Figure 1.34 shows two different regular graphs of fixed degree 3. For an agent the information on any vertex is exactly the same. It is not possible to distinguish between the two variants. At least one marker is necessary.

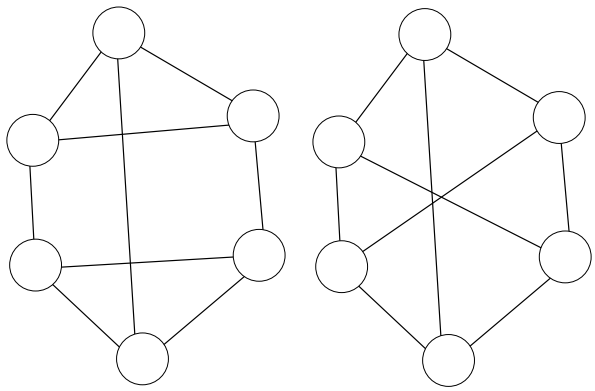

Figure 1.34: Two different regular graphs of degree 3, an agent cannot distinguish them without a marker.

Corollary 1.31 *Let $G = (E,V,S)$ be a graph with local cyclic edge order- Without a marker an online agent cannot build a correct map of the graph.*

Exercise 12 *Give a formal argument that the graphs in Figure 1.34 are different. Which class of graphs can be correctly detected by an online agent without a marker?*

A single marker (or pebble) is sufficient as shown by Dudek et al. [DJMW91]. We describe the corresponding *Marker*-Algorithm. The algorithm maintains the current known graph G^* and a list L of non-determined (seen but not correctly detected) edges. In the beginning the starting vertex is known and its outgoing edges belong to L. They are given in the cyclic order.

In the main step, the algorithm choose an edge e of L starting at a detected vertex b and moves to a vertex u along the edges $e = (b,u)$. Now the agent sets the marker on u, moves back to b along e and searches for the pebble in G^*.

Case 1: The pebble was not found in G^*. In this case we add the edge $e = (b,u)$ to G^* w.r.t. the cyclic order. All outgoing edges of u different from e will be inserted into the list L of non-determined edges.

Case 2: The marker has been found at some vertex $v \in G^*$. If there is more than one non-determined outgoing edges at $v = u$, we cannot precisely detect e. Therefore we take the marker, move back to b, place the pebble there, move back to v again and successively check the non-determined edges. Finally, we will detect the edge e and add it to G^* by the local order.

The above algorithm is simple and correct. By construction in any step an additional edge will be correctly detected. The number of exploration steps is restricted by $O(|E| \times |V|)$ and the same holds for the computational cost. We assume that the graph is not a multigraph and has no loop edges (v,v). Besides, we assumed that any edge has unit-length.

Theorem 1.32 *(Dudek, Jenkin, Milios, Wilkes, 1991)*
Let $G = (E,V,S)$ be a graph with given cyclic local order of the edges. By the use of one marker it is possible to fully detect the structure of the graph by online navigation with $O(|E| \times |V|)$ exploration steps and also overall $O(|E| \times |V|)$ computational cost.

Proof.

Let $G^* = (V^*,E^*,S^*)$ be the current graph during the execution of the Marker-Algorithm. Setting the marker has cost $O(1)$, searching for the marker in G^* can be done by DFS by $O(|V^*|)$ steps. Moving back and force along a path can be done in $O(|V^*|)$ steps as well. The traversal cost are considered for any edge, which gives $O(|E| \times |V|)$ steps in total.

For unit-edge length the computational cost are precisely the same for any edges we have to compute the shortest paths between two vertices. The effort is bounded by $O(|V^*|)$. This gives $O(|E| \times |V|)$. □

Exercise 13 *Explain why the cyclic order of the edges is necessary for the above Marker-Algorithm. Where is it used during the execution of the algorithm?*

Exercise 14 *Analyse the mechanical and the compuational cost of the marker algorithm for graphs with positive edge weights.*

Bibliography

[AFM00] E. M. Arkin, S. P. Fekete, and J. S. B. Mitchell. Approximation algorithms for lawn mowing and milling. *Comput. Geom. Theory Appl.*, 17:25–50, 2000.

[AKS02] Susanne Albers, Klaus Kursawe, and Sven Schuierer. Exploring unknown environments with obstacles. *Algorithmica*, 32:123–143, 2002.

[BRS94] Margrit Betke, Ronald L. Rivest, and Mona Singh. Piecemeal learning of an unknown environment. Technical Report A.I. Memo No. 1474, Massachusetts Institute of Technology, March 1994.

[DJMW91] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robotic exploration as graph construction. *Transactions on Robotics and Automation*, 7:859–865, 1991.

[DKK01] Christian A. Duncan, Stephen G. Kobourov, and V. S. Anil Kumar. Optimal constrained graph exploration. In *Proc. 12th ACM-SIAM Symp. Discr. Algo.*, pages 307–314, 2001.

[DKK06] Christian A. Duncan, Stephen G. Kobourov, and V. S. Anil Kumar. Optimal constrained graph exploration. *ACM Trans. Algor.*, 2:380–402, 2006.

[GR03] Yoav Gabriely and Elon Rimon. Competitive on-line coverage of grid environments by a mobile robot. *Comput. Geom. Theory Appl.*, 24:197–224, 2003.

[IKKL00a] Christian Icking, Thomas Kamphans, Rolf Klein, and Elmar Langetepe. Exploring an unknown cellular environment. In *Abstracts 16th European Workshop Comput. Geom.*, pages 140–143. Ben-Gurion University of the Negev, 2000.

[IKKL00b] Christian Icking, Thomas Kamphans, Rolf Klein, and Elmar Langetepe. Exploring an unknown cellular environment. Unpublished Manuscript, FernUniversität Hagen, 2000.

[IKKL05] Christian Icking, Tom Kamphans, Rolf Klein, and Elmar Langetepe. Exploring simple grid polygons. In *11th Internat. Comput. Combin. Conf.*, volume 3595 of *Lecture Notes Comput. Sci.*, pages 524–533. Springer, 2005.

[IPS82] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM J. Comput.*, 11:676–686, 1982.

[Lee61] C. Y. Lee. An algorithm for path connections and its application. *IRE Trans. on Electronic Computers*, EC-10:346–365, 1961.

[Sha52] Claude E. Shannon. Presentation of a maze solving machine. In H. von Foerster, M. Mead, and H. L. Teuber, editors, *Cybernetics: Circular, Causal and Feedback Mechanisms in Biological and Social Systems, Transactions Eighth Conference, 1951*, pages 169–181, New York, 1952. Josiah Macy Jr. Foundation. Reprint in [Sha93].

[Sha93] Claude E. Shannon. Presentation of a maze solving machine. In Neil J. A. Sloane and Aaron D. Wyner, editors, *Claude Shannon: Collected Papers*, volume PC-03319. IEEE Press, 1993.

[Sut69] Ivan E. Sutherland. A method for solving arbitrary wall mazes by computer. *IEEE Trans. on Computers*, 18(12):1092–1097, 1969.

Index

P

Q

R

S

T

W