Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Online Graph Exploration

Semester thesis

Simon Hungerbühler

simonhu@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Sebastian Brandt
Prof. Dr. Roger Wattenhofer

January 16, 2017

Acknowledgements

I thank my supervising tutor Sebastian Brandt for his assistance during this thesis. He took the time for our weekly meetings and provided helpful inputs when I had trouble. He also gave me a lot of support while writing and helped me to satisfy the academic standards.

Abstract

How should you act, if you wake up at an unknown place and you have to create a map of your environment for orientation? The problem is, that you only see the possible paths, but you do not know where they will end until you are there. What are good strategies to finish quickly? How much longer do you need, compared to someone who has a map?
First we consider unit-weight graphs. We prove a tight bound for the competitive ratio on unit-weight graphs. This bound is equal to 2.
Later we consider graphs with two different edge weights. Different natural algorithms have a competitive ratio of at least 4 on this graph class. The so called hierarchical depth search algorithm has a proven upper bound of 4 on graphs with two different edge weights. But potentially it has a better competitive ratio. We found a lower bound graph with a competitive ratio of 3.
Finally we provide an algorithm, which explores graphs with two different edge weights. This algorithm prevents some of the poor behavior of the others. It could have a better competitive ratio than 4.

Contents

Chapter 1

Introduction

In *online graph exploration* an agent has to explore an unknown input graph. At the beginning the agent does not know anything about the graph. Whenever the agent arrives at a new node all incident edges are learned. Based on the partial information, the agent has already received, he has to decide where to go next. The goal is to find a tour with as little as possible costs.
Online graph exploration is the online version of the traveling salesman problem. To measure how good an online algorithm performs, competitive analysis is used. The competitive ratio is the ratio between the costs of the online algorithm divided by the costs of the optimal solution of the offline problem.
One possible example could be a tour with a car through different cities. The online driver has no knowledge about the area and drives just after the direction signs. It learns the cities and theirs connecting roads step by step. The offline driver on the other hand gets a map in advance and has all information to plan an optimal tour.
The competitive ratio is at least 1, because you cannot be better than the optimal tour. But how much worse can you get? Is there a strategy with a bounded competitive ratio on all graphs? How does the number of edge weights influence the competitive ratio?
We take a look at graphs with one and two edge weights. We discuss the competitive ratios of existing algorithms and provide a new algorithm.

1.1 Related Work

One of the most intuitive strategies is called nearest neighbor or greedy. As the name implies, the explorer always chooses the nearest unexplored node as the next node to visit. Unfortunately this algorithm is only $\log n$ competitive [3]. Even on unit-weight graphs it is $\log n$ competitive [4].
There has been a lot of research on the topic of *online graph exploration*. New algorithms have been introduced over a long period of time, but there is still a large gap between the lower and upper bound of the competitive ratio. A lower bound of $5/2 - \varepsilon$ for every deterministic algorithm has been showed by Stefan

Dobrev et al. [1]. This is until now the highest lower bound graph to the best of our knowledge. Nicole Megow et al. found an algorithm with an upper bound on the competitive ratio of $2k$ [2]. With k the number of distinct weights. They generalized this result to arbitrary graphs by rounding every weight to the next power of 2 and achieved an upper bound for the competitive ratio of $\log n$ (n the number of nodes in the graph). This upper bound is achieved by an algorithm called hierarchical depth first search ($hDFS$). We analyse this algorithm and give a lower bound graph for any k with a lower bound on the competitive ratio of $2k - 1$. The basis lower bound graph consists of two different weights, but it is expandable to any number of weights.

Chapter 2

Model

There are different possible models for online graph exploration. We only consider undirected connected graphs $G = (V, E)$. Every edge $e \in E$ has a non-negative weight. The explorer has to visit all nodes. The tour begins at the start vertex $s \in V$. Whenever the explorer arrives at a new vertex, it learns all incident edges with their costs. But it does not know the identity of the vertex at the other end of the edge until both vertices are explored. Every vertex is distinguishable by its ID. The tour is finished, when all nodes have been visited and the explorer has returned to the start vertex. The total costs are the accumulated costs of every traversed edge.
The task is to find a tour with minimal cost. To rate the behavior of an online algorithm we use competitive analysis.

Definition 2.1. The competitive ratio for an algorithm on one graph is defined as the ratio of the cost the online algorithm accumulates and the optimal costs achieved by an offline algorithm on this graph.

Definition 2.2. The general competitive ratio for an algorithm is the highest competitive ratio the algorithm has on any graph or a graph class.

Definition 2.3. The competitive ratio for a graph is the lowest competitive ratio an online algorithm achieves on it.

Definition 2.4. A boundary edge $e = (u, v)$ is an edge with one explored vertex u and an unexplored vertex v. An edge is considered explored only if both vertices are explored.

Chapter 3

Unit-Weight Graphs

A unit-weight graph has only one edge weight. Without loss of generality we assume that it is 1.
We prove a constant competitive ratio of 2 on unit-weight graphs.

Theorem 3.1. *The competitive ratio on unit-weight graphs is 2.*

Proof. On our proof of the lower bound we will construct a graph $G = (V, E)$ at which every online algorithm has at least a competitive ratio of 2.
The graph contains two special nodes. The start node and the branch node. This two nodes are connected via two paths of length l. From the branch node starts a new path. This path is not connected to anything but just ends after some nodes. The number of the nodes of this third path is determined by the behavior of the online algorithm. Figure 3.1 shows one possible state of the partially explored graph G.
Since an online algorithm has to explore all nodes, it must arrive at the branch vertex after a finite number of traversed edges. At the first arrival at the branch vertex the algorithm has traversed at least $l + 2 \cdot a$ edges. a is the number of explored nodes on that path starting from the start node and ending at the branch node which is not fully explored. If the algorithm just walked on one path yet, a might also be zero.
Now there are three not completely explored paths. One starting at the start node and two starting from the branch node. The online algorithm can explore all of these paths. Every time a new node on a path is explored an additional node is connected to this path. We call the the number of explored nodes on the path starting at the start node $\tilde{a}$. ($\tilde{a} \geq a$)
The number of explored nodes at the two other paths we call b_1 and b_2. Without loss of generality $b_1 \geq b_2$. As soon as $\tilde{a} + b_1 = l$ the path starting at the start node is connected with the longer of the two other paths. The number of nodes of the third path gets now defined as $b_2 + 1$. This means, that the explorer missed to visit the last node of the path. The graph is now completely constructed.
At this point the accumulated costs from the online algorithm are at least $l + 2 \cdot \tilde{a} + 2 \cdot b_2 + b_1$. The last two tasks of the online algorithm are to visit

the last unexplored node of the third path and then to return from there to the start vertex.
The total costs accumulated form the online algorithm are at least:
$l + 2 \cdot \tilde{a} + 2 \cdot b_2 + 2 \cdot b_1 + 2 \cdot (b_2 + 1) + l$
$= 4 \cdot l + 4 \cdot b_2 + 2$
The minimal costs to explore this graph are: $2 \cdot l + 2 \cdot b_2 + 2$
For every $\varepsilon > 0$, there exists an l, that the competitive ratio is equal to $2 - \varepsilon$.

We show a lower bound with two simple arguments: A depth first search (DFS) on a graph $G = (V, E)$ with n vertices traverses $2 \cdot (n - 1)$ edges. The optimal tour traverses at least n edges. For every vertex one. Therefore the upper bound is also 2. □

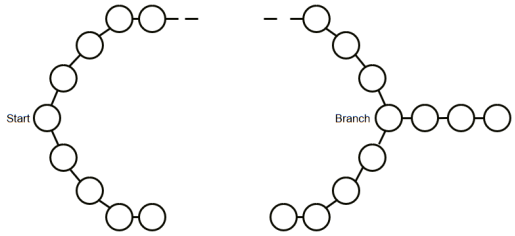

Figure 3.1: Lower bound graph for unit-weight graphs

Chapter 4

Graphs with Two Weights

With more weights things get more complicated. The lower bound of the competitive ratio increases. The authors of [1] proved a lower bound of $5/2 - \varepsilon$ for every deterministic online algorithm on arbitrary graphs.
What happens to the upper bound of the competitive ratio? The authors of [2] constructed an algorithm called hierarchical depth first search (hDFS) which is 2k-competitive (k equals the number of distinct edge weights). Until now there is no known algorithm with constant competitive ratio on arbitrary graphs. On the other hand there is no proof that it cannot exist.
In this chapter we focus on graphs with two different weights. First we show that *DFS* works bad on graphs with two different edge weights. Later we analyze an existing algorithm called hierarchical depth first search. In the end we present our own algorithm.

4.1 Depth First Search

DFS works pretty good at unit-weight graphs. It has a *CR* of 2 and we proved that there does not exist an algorithm with a $CR < 2$. But how good is *DFS* on graphs with two weights? Unfortunately the competitive ratio gets arbitrarily bad, if the difference of the two weights increases. Without loss of generality we assume the smaller weight to be 1. For the higher weight we use the symbol X.

Lemma 4.1. *A lower bound of the competitive ratio for* DFS *on graphs with two edge weights is* $X/2$.

Proof. We construct the lower bound graph as follows: It consists of three nodes arranged in a triangle (see Figure 4.1). The two edges ending at the start node have weight 1. The edge between the two other nodes has weight X. The costs for a *DFS* are $2 + 2 \cdot X$. The optimal costs depend on X. If $X < 2$ the optimal costs are $2 + X$. If $X > 2$ the optimal cost are 4. For large X the competitive

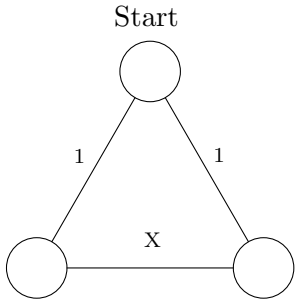

Figure 4.1: Lower bound graph for *DFS*

ratio is about $X/2$. The competitive ratio increases linearly with respect to X. □

Lemma 4.2. *An upper bound of the competitive ratio for* DFS *on graphs with two edge weights is* $2 \cdot X$

Proof. An upper bound for the *DFS* on graphs with two different edge weights follows from these two arguments: The worst thing *DFS* can do, is always take en edge of weight X. This leads to costs of $2 \cdot (n-1) \cdot X$. The optimal solution has at least costs of n. Hence the competitive ratio for *DFS* is always smaller then $\frac{2 \cdot (n-1) \cdot X}{n} \leq 2 \cdot X$ □

4.2 Hierarchical Depth First Search

Hierarchical depth first search *hDFS* [2] explores an input Graph $G = (V, E)$ in a DFS manner. But it only focuses on a subgraph $\tilde{G}(v, w)$ of G (with v the current position and w some weight). $\tilde{G}$ is composed of all nodes, which are accessible with edges of weight at least w from v. The weight w is chosen as the smallest value such that $\tilde{G}$ still contains boundary edges.
The algorithm does not consider the actual distance it takes to explore a new vertex but only the value of the highest edge weight it has to traverse. This fact leads to situations, where the algorithm walks very long distances because it always takes the smaller edge weight first.
We construct a lower bound graph $G = (V, E)$ where we exploit this behavior. The start vertex has 4 neighbors. All are connected to the start vertex with an edge of weight 1. Two neighbors are the starting points of two chains of length l. In these chains all nodes are connected to their neighbors with an edge of weight 1. After this two chains are built there are 4 leafs in the graph. The two neighbors from the start node, which have not been changed yet and the two endpoints of the chains. Then add to every leaf the lowest weight of all k weights

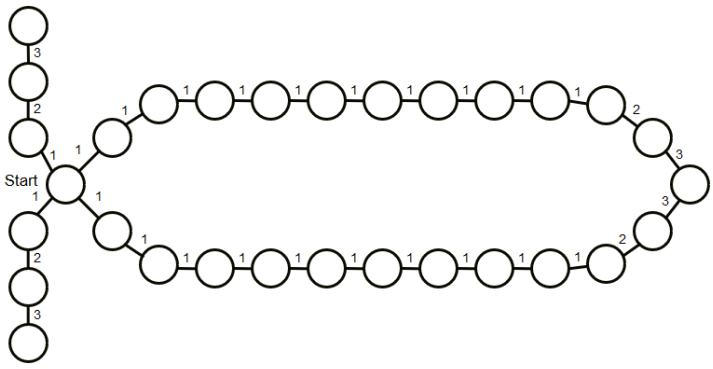

Figure 4.2: Lower bound graph for *hDFS*

you want to add to the graph. Repeat this step until no distinct weights are left. Finally merge the two end nodes of the chains of the length $l + k$. An example how this could look like with three weights is shown in Figure 4.2.

The optimal tour has costs of: $2 \cdot l + 6 \cdot \sum_{i=1}^{k} w_i - w_k$

hDFS traverses the chains of length l two times for every weight, because the algorithm does not take higher boundary edges if it can reach other boundary edges with smaller weights. The graph has two additional chains which contain all edge weights for the reason that there is always an unexplored boundary edge near the start and the explorer has to return and cannot take two new nodes at once on the long chains of length l. For the highest edge weight the algorithm takes only one chain twice.

hDFS has at least costs of $2 \cdot (2 \cdot k - 1) \cdot l$. If l is large enough the competitive ratio of *hDFS* is $2 \cdot k - 1$ on these graphs.

4.3 Advanced Depth First Search

Advanced depth first search *aDFS* is our algorithm designed to explore graphs with only two different edge weights. As the name implies, it does a DFS but with certain exceptions.

aDFS avoids the disadvantages of *hDFS*. Our algorithm does not go back large distances, if there is a boundary edge of weight X. But what does large mean? When should you return and when should you take an edge with a large weight? We answer this question with the following consideration. Imagine a cycle with one edge of weight X and the other edges with weight 1. The explorer has to visit all nodes and then to return to the start. After which number of visited nodes should the explorer return if it is in front of the X edge?

If the cycle has less than X edges of weight 1 it is always better to go back than to explore the edge of weight X. If the cycle has more than X edges of weight

1 it is always better to traverse the edge of weight X and finish the tour in one round. Unfortunately the explorer does not know the total number of nodes in the cycle but knows the number of already visited nodes.

Definition 4.3. A boundary edge $e = (u, v)$ of weight X is called blocked, if there is a boundary edge $e' = (u', v')$ of weight 1 with a shortest path between the two explored nodes of these two boundary edges smaller than X.

Definition 4.4. A boundary edge $e = (v, u)$ of weight X is called unblocked, if there is no boundary edge $e' = (u', v')$ of weight 1 with a shortest path between the two explored nodes of these two boundary edges smaller than X.

Definition 4.5. A boundary edge $e = (v, u)$ of weight 1 is allways considered to be unblocked.

Lemma 4.6. *There exists a strategy with a competitive ratio of 2 on these cycles.*

Proof. The strategy is to perform a *DFS* but to ignore blocked boundary edges. If the agent has visited all nodes it returns to the start on the shortest path and does no backtracking.
There are two cases: The edge of weight X is blocked or unblocked when the agent arrives there.
If the edge of weight X is blocked when the agent arrives there, the agent returns to the start and then explores the other side of the cycle until it arrives at the other endpoint of the edge of weight X. Then the agent has visited all nodes and returns to the start. As the agent explored all nodes with an edge of weight 1 and traversed all this edges twice the costs cannot be higher than twice the optimal costs.
If the edge of weight X is unblocked the agent explores all nodes in one round. As the edge of weight X is unblocked there are more than X edges of weight 1 (n is bigger or equal than X). The costs accumulated by this strategy are smaller than two times the minimal costs. □

A first idea could be that the explorer is not allowed to depart further than X from a blocked boundary edge. This turns out to be a very bad idea. Figure 4.3 shows a lower bound graph with an arbitrarily bad competitive ratio for this constraint. We assume that there are a chains of edges of weight 1. This chains all have a length of X. The chain on the left side has length $X - 1$. On every node of this chain one node is connected with an edge weight of X.
Since the agent is not allowed to depart further than X from a blocked edge, the only possibility to unblock all edges of weight X, is to explore every round one additional edge on every chain of edges of weight 1. After $X - 1$ repetitions of this step all nodes of the graph are explored.

The optimal tour costs are: $2 \cdot X \cdot X + 2 \cdot (a+1) \cdot X$
The online algorithm has costs of: $2 \cdot X \cdot X + (a+1) \cdot \sum_{i=1}^{X} i$
$= 2 \cdot X \cdot X + (a+1) \cdot X \cdot (X+1)/2$
If a is chosen as X and X is large enough a lower bound of the competitive ratio of an algorithm with this constraint is about X.
Another problem of an algorithm with this repression are deadlocks. Two boundary edges of weight X are blocked by two different boundary edges of weight 1. The first boundary edge of weight 1 is too far away from the second boundary edge of weight X and vice versa. Therefore neither of them is unblocked and the algorithm gets stuck.

Algorithm aDFS(G,u)
Input: A partially explored graph G, a vertex $u \in V$

```
1:  if there is a boundary edge (u,v) of weight 1 do
2:  traverse (u,v)
3:  aDFS(G',v) //G' partially explored graph when the explorer moved to v
4:  else if there is an unblocked boundary edge (u,v) of weight X do
5:  traverse (u,v)
6:  aDFS(G',v) //G' partially explored graph when the explorer moved to v
7:  else if all nodes are visited and u is the start node do
8:  break
9:  else do
10: plan a shortest tour through all vertices which have an unblocked boundary
11: edge and then back to the start
12: traverse (u,v) // v first vertex of this tour
13: aDFS(G',v) //G' partially explored graph when the explorer moved to v
14: end if
```

The solution is to cancel this rule. The final specification of advanced depth first search *aDFS* is formally written above. Whenever the explorer is at a node with a boundary edge of weight 1 it takes this edge. If the explorer is at a node with an unblocked boundary edge of weight X and no boundary edge of weight 1 *aDFS* also traverses this edge of weight X. If there are no unblocked boundary edges at the current node the algorithm plans a shortest tour through all nodes with an unblocked boundary edge and then back to the start. Advanced depth first search just visits the first node of this shortest tour. At this node is at least one unblocked boundary edge, which will be traversed next.
For the new specification we have not found any lower bound graph with a competitive ratio larger than 2. On any cycle with two different edge weights *aDFS* has at least a competitive ratio of 2. The graphs with two different edge weights

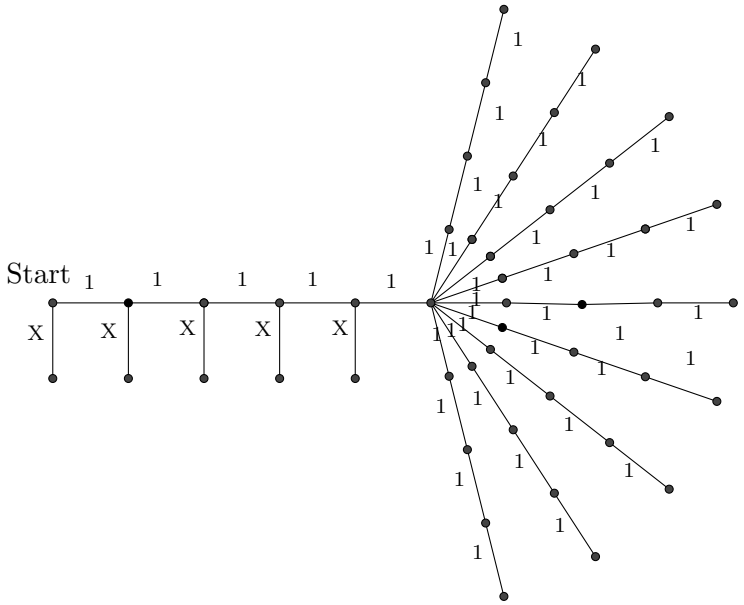

Figure 4.3: Lower bound graph

on which *hDFS* has a competitive ratio of 3 *aDFS* has a competitive ratio of less than 2. Also with the graphs on which the greedy algorithm has a competitive ratio of $\log n$ *aDFS* is still below 2.
We have no proof for an upper bound of the competitive ratio for *aDFS* due to the fact that the behavior of the tour planning is not really predictable. But it seems that *aDFS* performs better on graphs with two different weights than the others do.

Bibliography

[1] Stefan Dobrev, R.K., Markou, E.: Online graph exploration with advice. In: Structural information and communication complexity. (2012)

[2] Nicole Megow, K.M., Schweitzer, P.: Online graph exploration: New results on old and new algorithms. In: Automata, Languages and Programming

[3] Daniel Rosenkrantz, R.S., Lewis, P.: An analysis of several heuristics for the traveling salesman problem. In: SIAM journal on computing. (1977)

[4] Cor Hurkens, G.W.: On the nearest neighbor rule for the traveling salesman problem. In: Operations research letters. (January 2004)