Online graph exploration on trees,unicyclic graphs and cactus graphs

Md.Mahedi Hasan Rigan
Student ID : 1705031

Asif Mustafa Hassan
Student ID : 1705041

July 28, 2021

Contents

Chapter 1

Introduction

This report briefly illustrates the well known problem Online graph exploration on different graphs.Online graph exploration is the problem of exploring all vertices of an undirected weighted graph that is initially unknown to the searcher.

Normally we work with graphs that we know the edges and vertices.These are known as offline graph.But in online graph we don't know the edges beforehand and we have to traverse all the vertices before starting to the start node. Basically this is similar to Travelling Salesman Problem(TSP).But the only difference between them is in TSP we know the edges value and make efficient choice to traverse the graph.

We consider a fixed graph scenario in which a connected undirected graph $G = (V, E)$with $n = |V|$vertices is explored. Each edge $e \epsilon E$ has a positive weight —e— and the graph contains a distinguished start node $s \epsilon V$from which the searcher begins its explo-ration.We assume that each vertex has an assigned unique identifier (ID).While arriving a vertex for the first time, the searcher obtains the weights of all edges incident to that vertex as well as the IDs of all adjacent vertices.

We will use competitive analysis to measure the performance of an online graph where competitive ratio is the ratio of an online problem solution to its corresponding offline problem solution.So in our case we will get the ratio using travelling salesman problem. We call an online exploration algorithm $c-$competitiveif it produces a tour no longer than c times the optimal (offline) tour for every instance.

The best known algorithms on general graphs are Near-est Neighbor (NN) and hierarchical DFS both with a competitive ratio of Θ $(logn)$. For NN this worst-case ratio is tight even on planar unit-weight (unweighted) graphs. In particular, no algorithm with constant competitive ratio is known on general graphs. On the other hand, the best known lower bound on the competitive ratio of an

online algorithm has recently been improved from 2.5 to 10/3.We prove that the tight lower bound of $\Theta(logn)$ on the competitive ratio of NN also holds on trees which improves the previous lower bound of $\Theta(logn/loglogn)$. We do so by modifying a graph construction Hurkens and Woeginger use to prove the lower bound on planar unit-weight graphs.

Chapter 2

Problem Solving Techniques

We are going to mainly focus on two algorithms to solve an online graph exploration on trees, unicyclic grah and cactus graph.These two algorithms are Nearest Neighbour Algorithm and Blocking Algorithm. Here we will discuss Nearest Neighbour Algorithm and Blocking Algorithm as both of them are relevent to our problem.

2.1 Nearest Neighbour Algorithm

The nearest neighbour algorithm was one of the first algorithms used to solve the travelling salesman problem approximately. In that problem, the salesman starts at a random city and repeatedly visits the nearest city until all have been visited. The algorithm quickly yields a short tour, but usually not the optimal one.These are the steps of the algorithm:

1. Initialize all vertices as unvisited.
2. Select an arbitrary vertex, set it as the current vertex u. Mark u as visited.
3. Find out the shortest edge connecting the current vertex u and an unvisited vertex v.
4. Set v as the current vertex u. Mark v as visited.
5. If all the vertices in the domain are visited, then terminate. Else, go to step 3.

The sequence of the visited vertices is the output of the algorithm.The nearest neighbour algorithm is easy to implement and executes quickly, but it can sometimes miss shorter routes which are easily noticed with human insight, due to its "greedy" nature. As a general guide, if the last few stages of the tour are comparable in length to the first stages, then the tour is reasonable; if they are much greater, then it is likely that much better tours exist. Another check is to use an algorithm such as the lower bound algorithm to estimate if this tour is good enough.

2.2 Blocking Algorithm

The algorithm Blockingis a generalization of DFS. It uses a blocking condition which, depending on a fixed blocking parameter $\partial \epsilon R$, determines when to delay the traversal of an edge, possibly forever.

Definition 2.2.1 (Boundary edge). During the exploration, we call an edge a boundary edge when one of its endpoints has been visited while the other has not.

Definition 2.2.2 (Blocking condition).A boundary edge $e = (u, v)$is blocked by another boundary edge $e' = (u', v')$ if eis shorter than e and the length of any shortest path from u to v' is at most $(1 + \partial)|e|$.

Algorithm 1: The exploration algorithm Blocking $_\partial$ (G, y)as in

Input:A partially explored graph G, and a vertex y of G that is explored for the first time.;

while *there is an unblocked boundary edge* $e = (u, v)$, *with* u *explored and* v *unexplored, such that* $u = y$ *or such that* e *had previously been blocked by some edge* (u', y) **do**

  walk a shortest known path from y to u ;

  traverse $e = (u, v)$;

  Blocking $_\partial$ (G, v);

  walk a shortest known path from v to y;

end

On planar graphs, Blocking $_\partial$ is $2(2+\partial)(1+2/\partial)$-competitive for $\partial > 0$ and in particular 16-competitive for $\partial = 2$[4]. Like in that proof we charge the costs of the algorithms actions to the edges of the explored graph. Let Bbe the cost of Blocking, i.e. the sum of charges to all edges. For each iteration of the while loop, the costs of the movements described in the Lines 2, 3 and 5 are charged to the edge traversed in Line 3. Note that only unblocked boundary edges are charged this way and, in particular, every edge will be charged at most once. Moreover, the following holds.

Chapter 3

Online graph exploration on trees

We recursively define graphs G_k for $k \geq 1$ containing three distinguished vertices l_k, r_k and m_k. The graph G_1 simply consists of the two unit length edges $l_1 r_1$and $r_1 m_1$. For $k \geq 2$, we construct G_k by placing two copies G_{k-1} and G_{k-1} of G_{k-1}next to each other and, in the middle, adding a new vertex m_k. To connect the components, we add an edge of length k between $r'_{k-1}$and l''_{k-1}as well as a unit weight edge between $l''_{k-1}$and m_k.

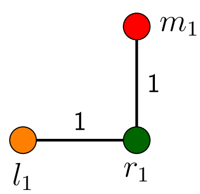

Figure 3.1: G_1

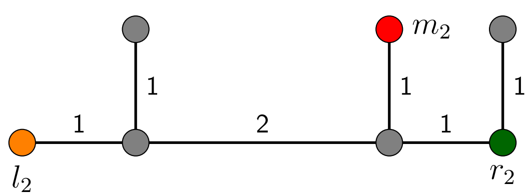

Figure 3.2: G_2

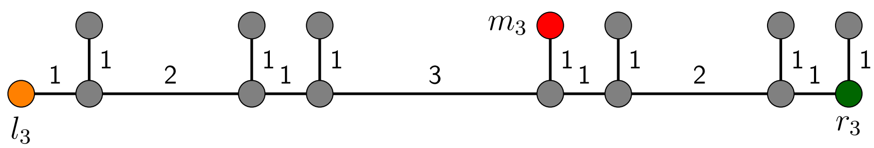

Figure 3.3: G_3

Lemma 3.0.1. *For $k \geq 1$, consider a graph G that contains G_k as a subgraph. Furthermore, assume that edges between G_k and $G - G_k$ are either incident to l_k and have a length of at least 1 or are incident to r_k and have a length of at least $k + 1$. Then there exists a partial NN tour exploring all of G_k that starts in l_k, finishes in m_k and has a length of*

$$(k + 1) * 2^k - 2$$

Our goal was to show that the tighter bounds of competitive ratio of NN also holds on the family of the tree.

- The upper bound follows directly from the general case.
- We show that: The lower bound on trees remains same if we follow NN approach.

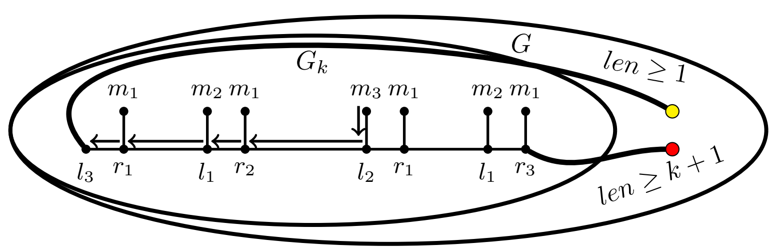

Figure 3.4: m_k to l_k

Calculating the Competitive Ratio lower bound:

- Traversing graph G_k when G_k is a part of larger graph G with some conditions, the shortest path length from l_k to m_k is $(k + 1) * 2^k - 2$.
- We can use Lemma 1 for only G_k as we did not need to go outside G_k in Lemma 1.
- Finally complete exploration by returning to l_k from m_k.

l_k to m_k needs length: $(k + 1) * 2^k - 2$ m_k to l_k needs length:

$$\begin{aligned} Ret(G_k) &= 1 + k + p_{k-1} \\ &= 1 + k + (2^k - (k - 1) - 2) \\ &= 1 + k + 2^k - k + 1 - 2 = 2^k \end{aligned}$$

Theorem 3.0.2. *The competitive ratio of NN on trees is* $\Theta(logn)$.

Proof: Total length for complete exploration on Online Tree G_k:

$$\begin{aligned} NN(G_k) &= Ret(G_k) + (k+1) * 2^k - 2 \\ &= 2^k + (k+1) * 2^k - 2 \\ &= (k+2) * 2^k - 2 \end{aligned}$$

The NN approach on particular Online Tree family has length:

$$(k+2) * 2^k - 2$$

The optimal solution on this offline Tree family should have length:

$$\begin{aligned} OPT(G_k) = 2 * w_k \\ &= 6 * 2^k - 2k - 6 \end{aligned}$$

G_k has vertices:

$$\begin{aligned} n_k &= 2^{k+1} - 1 \\ 2^{k+1} &= n_k + 1 \\ log_2 2^{k+1} &= log_2(n_k + 1) \\ k + 1 &= log_2(n_k + 1) \\ k &= log_2(n_k + 1) - 1 \end{aligned}$$

So, Lower bound:

$$\begin{aligned} c &= \frac{NN(G_k)}{OPT(G_k)} \\ &= \frac{(k+2) * 2^k - 2}{6 * 2^k - 2k - 6} \\ &\geq \frac{k+2}{6} \\ &= \frac{\log_2(n+1) + 1}{6} \quad \square \end{aligned}$$

Chapter 4

Conclusion

Online explorations are common in practical life. Moreover, online tree explorations help robots determine their path in unknown situations and much more. It is extremely important yet does not seem useful from a theoretical point of view. Although, this problem looks quite similar to TSP(Travelling Salesman Problem), it can be solved in polynomial time upon defining the environment family. A large number of tree family can be defined for polynomial online exploration domain. We can think of a future where most of the graphs will be online explorable in polynomial time, enhancing our robots and much more.

Bibliography

[1] K. Pruhs B. Kalyanasundaram. *Constructing competitive tours from local information*. Theor. Comput. Sci. 130(1), 1994, pp. 125–138.

[2] G. Woeginger C. Hurkens. *On the nearest neighbor rule for the traveling salesman problem*. Oper. Res. Lett. 32, 2004, pp. 1–4. URL: https://doi.org/10.1016/S0167-6377(03)00093-2.

[3] P.M. Levis D.J. Rosenkrantz R.E. Stearns. *An analysis of several heuristicsfor the traveling salesman problem*. SIAM J. Comput. 6(3), 1977, pp. 563–581.

[4] R. Wattenhofer M. Herlihy S. Tirthapura. *Competitive concurrent distributed queuing*. Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing, PODC '01, As-sociation for Computing Machinery, New York, NY, USA, 2001, pp. 127–133.

[5] P. Schweitzer N. Megow K. Mehlhorn. *Online graph exploration: new results on old and new algorithms*. Theor. Comput. Sci. 463, 2012, pp. 62–72.

[6] G. Trippen R. Fleischer. *Exploring an unknown graph efficiently*. G.S. Brodal, S. Leonardi (Eds.), Algorithms – ESA 2005, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 11–22.

[7] M.R. Henzinger S. Albers. *Exploring unknown environments*. SIAM J. Comput. 29(4), 2000, pp. 1164–1188. URL: https://doi.org/10.1137/S009753979732428X.

[8] E. Markou S. Dobrev R. Královi ˇc. *Online graph exploration with ad-vice*. G. Even, M.M. Halldórsson (Eds.), Structural Information and Communication Complexity, Springer Berlin Heidelberg, Berlin, Hei-delberg2, 2012, pp. 267–278.

[9] Y. Okabe S. Miyazaki N. Morimoto. *The online graph explo-ration problem on restricted graphs*. IEICE Trans. 92-D(9), 2009, pp. 1620–1627.

[10] C.H. Papadimitriou X. Deng. *Exploring an unknown graph*. Pro-ceedings of the 31st Annual Symposium on Foundations of Computer Science, SFCS '90, vol.1, IEEE Computer Society, Washington, DC, USA, 1990, pp. 355–361.