Final Project - Distributed Graph Algorithms - Spring 2022
On the paper: Time-Optimal Construction of Overlay Networks, by Thorsten Götte, Kristian Hinnenthal, Christian Scheideler, and Julian Werthmann

Lior Atias[*] Raz Ashkenazi[†]

01.08.2022

1 Summary

In this project, we talk about construction of overlay networks in optimal time. An overlay network is a network that is built on top of another network. For example, peer-to-peer networks which operate over the internet are overlay networks. We can think of the ids of nodes in the networks as IP addresses. The difficulty in building the overlay networks is that it might expand the diameter or the degrees of some nodes which can impair performance of the networks. So basically, we want to build the overlay networks quickly with low diameter and low degrees for each node.

1.1 The Model and Assumptions

The communication here works in synchronous rounds as learnt in class. The model that is used here is the NCC_0 model which is very similar to the CONGEST model as we learned in class and talks about networks, in which each node initially only knows the ids of its neighbours, but now new connections can be established by sending node ids. They assume each node can send a message in size of $O(\log n)$ bits, and can send and receive at most $O(\log n)$ messages in each round. The n nodes here are the participants of all networks. Each node has a unique id with size of $O(\log n)$. When node u knows the id of node v, it means that there is a direct edge from u to v and therefore node u can send messages to node v. A node's degree is the sum of his incoming edges with his outgoing edges, and the graph's degree is the maximum degree of any node in the graph, denoted by d in this paper. They refer a weakly-connected graph as a graph, either as directed or undirected, that has at least one path for each pair of nodes. When we refer to a graph as undirected, we assume it can be made bidirected by letting each node introduce itself to all of its neighbors, and it can be made easily in their algorithms.

[*]lioratias@campus.technion.ac.il, 316198894
[†]razas@campus.technion.ac.il, 311177034

1.2 Illustration

A simple example of overlay networks in directed distributed graphs: for a given path $P = u \to v \to w$, we can construct an overlay network by adding a new edge (u, w). This can be done if node v sends the id of node w to node u. From now, node u can send direct messages to node w and might save time in future rounds. In case of clique, every node knows the ids of all nodes in the graph. However, in a graph that isn't a clique, every node cannot send all the ids to all nodes for building a clique because of the limits in our model as mentioned before.

1.3 The Complexity Desired

Without limiting the message size, the worst case we have is a line of nodes, and the first node needs to know about the last node. There is an algorithm based on pointer jumping which takes $O(\log n)$ rounds for the first node to know about the last node in the line, and thus this is the lower bound. According to this paper, the best time achieved for all nodes to know about each other was $O(\log^{\frac{3}{2}} n)$ rounds [GHS19], but now in this paper it was found out that the best time is $O(\log n)$ rounds w.h.p.

1.4 The Algorithm

The general idea of the new purposed algorithm is to use constant length random walks, denoted by l in this paper: for any directed graph as input, we turn the graph into a specific weakly-connected graph, denoted as benign graph in this paper, and then each node u spreads unique messages that called tokens. The tokens contain the id of u, and each of them walk along the graph randomly and independently. After l rounds, every token t gets to an endpoint node v_t, and for some of these nodes that received the tokens we construct new edges between them and node u, i.e. with both directions. We repeat this process for $L = O(\log n)$ iterations, until we get an overlay network as a rooted tree with diameter in size $O(\log n)$ and constant degree Δ for each node in the graph, denoted as well-formed tree in this paper.

1.4.1 The Inputs

$\forall i \in [L]$, we denote $G_i = (V, E_i)$ as the current communication graph which created after i iterations of random walks in the algorithm. Thus, we denote G_0 as the input connected graph with degree d. Furthermore, the algorithm also get the parameters l, Δ and L as mentioned before as inputs. The last input parameter is Λ in size $O(\log n)$, and is particularly used to make the benign graph. All these inputs are known to all nodes in V.

1.4.2 Phase I: Make the Benign Graph

If we want runtime complexity of $O(\log n)$ rounds w.h.p, then the algorithm must work only with weakly-connected graphs and constant degree for each node, because of the methodology of random walks (which will be explained in the next section). Thus, for any sort of directed graph $G_0 = (V, E_0)$ as input, we want to make it as weak as possible, and in the same time we want to keep it connected, which is guaranteed for been connected if we keep the degree Δ in size of $\Omega(\log n)$. As revealed in this paper, there are 3 properties that were chosen carefully and are all mandatory to make the graph as such, and this sort of graph denoted as benign graph:

1. Every node in V has exactly Δ ingoing and Δ outgoing edges (including self-loops).
2. Every node in V has at least $\frac{\Delta}{2}$ self-loops.
3. Every cut $c(U, \overline{U})$ s.t. $U \subset V$ has at least Λ edges.

Now, to make the input graph into benign graph, if we assume that $2d\Lambda \leq \Delta = O(\log n)$, and we can assume that because d is constant, then the input graph can be turned benign in 2 steps:

1. All edges are copied Λ times to obtain the desired minimum cut (property 3), and then we get that each node has at most $d\Lambda$ edges to other nodes.
2. Each node adds self-loops until its degree gets to Δ. By the time it adds all the self loops needed, it will have $\frac{\Delta}{2}$ self-loops because of the assumption $2d\Lambda \leq \Delta \Rightarrow d\Lambda \leq \frac{\Delta}{2}$.

1.4.3 Phase II: Use the Random Walks

After getting the benign graph, we want to establish new connections with L iterations using random walks. In each iteration $i \in [L]$, every node $v \in V$ creates $\frac{\Delta}{8}$ tokens, which all include id(v), and send them to walk across the graph $G_i = (V, E_i)$ randomly in each step, for l steps independently on the previous steps, i.e. a token can visit the same edge more than once in a single iteration. In distribution graphs perspective, when a node $u \in V$ receives a new token t, he chooses which of his neighbors $w \in N(u)$ should get t from him, and the decision is made uniformly at random among $N(u)$ with outgoing edges toward them. After l iterations of random walks, let $v' \in V$. If v' received less than $\frac{3\Delta}{8}$ tokens, then it means v' is about to communicate directly with the tokens' origins, and thus he sends its own id back to all tokens' origins to create edges between them. Else, v' received at least $\frac{3\Delta}{8}$ tokens and thus he chooses $\frac{3\Delta}{8}$ tokens randomly among all the received tokens to communicate with them directly by the same way as sending back his id to the tokens' origins. In constant number of rounds, for each token t' that v' received, v' can establish a new edge to the origin of t' easily by looking at the id stored in t'. By the end of iteration i, each node in V adds enough self-loops until its degree gets to Δ.

1.4.4 Phase III: Establish the Overlay Network

After L iterations of random walks we get the final graph $G_L = (V, E_L)$, which has diameter in size $O(\log n)$ thanks to the short random walks, and constant degree Δ for each node as promised from the last stage in each iteration. Finally, we want to establish the overlay network as a rooted tree from this graph. To obtain this, the algorithm first runs BFS on the graph, starting from the node with the lowest id, to get a rooted tree with diameter and degree $O(\log n)$. Then, we want to transform this tree to a well-formed tree, i.e. what we missing is constant degree Δ for each node. This is basically done by arranging each node's children as a path, and then the node keeps only an edge to one of them. With this action we get constant degree Δ for each node in the graph, and thus we get the desired well-formed tree, which finalizes the whole algorithm.

2 Related Work

There were many attempts over the years to find the best algorithms to construct overly networks in optimal time. In all previous researches they used supernodes as the main idea, but some with different rules, which basically are big sets of nodes which work in coordination. This idea is pretty basic method but implementing it can be quit difficult, which is why the random walks idea was also a breakthrough, in addition to discovering the best optimal time to this day. After reviewing the supernodes methodology, we will also review briefly how the algorithm in this paper solves graph problems in the hybrid networks, which was also found out in this paper.

2.1 The Supernodes Methodology

The supernodes work in phases, and in each phase we have some supernodes of a certain size. At first, they try to find all the other supernodes' ids, and according to the relevant algorithm they decide to merge with some of them to bigger supernodes, and if we repeat for enough number of phases then we get the topology we want for the overlay network.

2.1.1 The First Algorithm

It was probably first to found out about fast construction of overlay networks with supernodes by Angluin, Aspnes, Chen, Wu and Yin in 2005 [AAC+05].

References

[AAC+05] Dana Angluin, James Aspnes, Jiang Chen, Yinghua Wu, and Yitong Yin. Fast construction of overlay networks. *Proc. of the 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, page 145–154, 2005.

[GHS19] Thorste Götte, Kristian Hinnenthal, and Christian Scheideler. Faster construction of overlay networks. *International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, page 262–276, 2019.