Maintaining Subgraphs in Fully Distributed Data Streams

Rohan Garg
Purdue University, Department of Computer Science
rohang@purdue.edu

Abstract

As the number of computers increase, guarantees and algorithms for distributed computing models become a requirement rather than a desire. Furthermore, with the new paradigm of big data, we must adapt our distributed protocols to space and time limitations. In this paper, we give a deterministic, distributed protocol for maintaining a β-degree bounded subgraph H of a graph $G = (V, E)$ in a fully distributed system where the edges of G arrive in a stream $E = (e_1, e_2, \ldots, e_m)$. We consider adversarial streams but our algorithm can be extended to work on incidence streams.

2012 ACM Subject Classification Theory of computation → Design and Analysis of Algorithms

Keywords and phrases Distributed Computing, Graph Theory

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

Subgraphs are a widely used graph-theoretic concept used for many algorithms on graphs. A subgraph H of a graph $G = (V, E)$ is a graph on V with some subset of edges $E' \subseteq E$ where the subgraph H satisfies some useful property. Often, we want the subgraph H to reduce the size of the graph G so that we can simply work with H instead of G and gain some savings in the space and time-complexities of our algorithms. A simple example is given a connected, undirected graph $G = (V, E)$ with we can use the subgraph H where H is a spanning tree of G to have knowledge of a path from any node u to any other node v without storing all $O(|V|^2)$ edges of G. H will have exactly $(|V| - 1)$ edges.

In this paper, we consider a distributed computing model. In this model, we have machines or *nodes* that communicate via messages. As distributed systems become more commonplace, its important we design algorithms that function in distributed settings with low communication overhead and fault-tolerance guarantees. Distributed algorithms are hard to design and often harder to reason about. A distributed system designer must deal with inherent non-determinism due to the presence of many computing nodes and different execution traces.

On top of considering a distributed setting, we also focus on the case where the data is given in a stream. The *streaming* model where input arrives in a stream and we can only use $O(polylog(n))$ space has been motivated by the big data paradigm of computation in the 21st century. With software technology companies working on datasets in the size of trillions of samples, it no longer becomes feasible to store all the data on one machine. Similarly, the *semi-streaming* model for graph problems allows us $O(n \cdot polylog(n))$ space. This has been shown to be a rich area for graph streaming problems [4], [8].

1.1 Related Work

Recently, streaming settings were incorporated into the distributed computing model. Introduced by Cormode, the *Continuous Distributed Monitoring Model (CDMM)* describes a model for computing useful functions over data streams [3]. In this model, we have k sites $S_1, S_2, \ldots, S_k$ (or nodes) and one special node, C, called the coordinator node. Sites may not communicate with each other but all sites can communicate with the coordinator. Other works have considered distributed, parallel streams for two parties, Alice and Bob, and measure communication complexity to compute functions over the union of both data streams [6]. For more on distributed data streams, see [5].

Recently, Kane et. al studied the problem of counting arbitrary subgraphs in data streams. They give algorithms that apply to the distributed setting and also work in the turnstile setting [7].

1.2 Problem Definition

In this paper, we consider the problem of returning a degree bounded subgraph H of a given graph $G = (V, E)$. The bound on the max-degree of H will be a parameter β given to us. Additionally, the graph G will be given as a stream. We will know the vertex-set V before-hand but the edge-set E will be given as a stream $(e_1, e_2, \ldots e_m)$.

1.3 Our Contributions

We give a protocol for the fully, distributed streaming model that computes a β-degree bounded subgraph H of a given graph G where no node in H has degree larger than some given parameter β. The result can be summarized by the main theorem below:

Theorem 1: The algorithm *DualCommit* computes a β-degree bounded subgraph in the fully, distributed streaming model using no more than $2m$ messages with size $O(\log n)$ in $O(m)$ rounds.

2 Model

For our algorithms, we will consider the *fully, distributed streaming model.* We will focus on the case of graph streaming. This model can be defined more generally to include other forms of data streams. We'll first describe the distributed portion of the model and then we will describe the streaming portion and assumptions about the stream.

We will assume the commonly used CONGEST model of distributed systems. In the CONGEST model, we have n nodes in the system. The system is fault-free, message-passing, and asynchronous. In the asynchronous model, messages sent between machines will take a finite but potentially arbitrarily long amount of time to reach. We assume a fully connected underlying communication graph where edges represent bidirectional communication links. We assume that each edge delivers messages in FIFO order. Initially, nodes are in some *idle* state and a node is awakened either by the receipt of a message from another node or the receipt of a new edge in the stream that it is an endpoint of. Each node is only responsible for computing its part of the output. In this case, that means each node must know which edges incident on it are in the subgraph H. The model is message-passing so nodes communicate to each other via messages on the communication network. We assume that nodes suffer no faults during the algorithm and that the communication network also does not fail. All nodes have a unique identifier (UID) and that the size of each UID is no more than $c \log(n)$ bits for some constant c. In the CONGEST model, every message is restricted to size $\Theta(\log n)$. In a closely related model called the LOCAL model, message size is unbounded.

For the streaming model, we assume that the edges of $G = (V, E)$ arrive in a stream $E = (e_1, e_2, \ldots, e_m)$ and on edge $e = (u, v)$ only nodes u and v are notified of edge e. The stream ends with an "end-stream" token and all nodes are notified of the end of the stream.

3 The *DualCommit* Algorithm

In this section, we describe the *DualCommit* Algorithm to compute a β-degree bounded subgraph H of a graph $G = (V, E)$ where the edges arrive in a stream $E = (e_1, e_2, \dots, e_m)$.

At a high level, the algorithm is rather simple. Initially, all vertices have no knowledge of any edges. Each node will store its own degree value and maintain that its degree is never larger than β. On the arrival of edge $e = (u, v)$ with $UID_u < UID_v$, both u and v are notified of edge e. On the arrival of the edge, only u, with the smaller UID will check its local degree value. If $degree(u) < \beta$ then u sends a *commit* message to v. A *commit* from a vertex u to its neighbor v signifies that u is committed to discarding any other edges it receives and will not commit to other nodes in the meantime until it receives a response from v. In this case, the commit message itself will serve as a confirmation message. We will use a "no" message for when a node needs to tell a node that either it is committed or its degree is too high and it cannot handle another edge. For example, if u sends a commit message, after some wait, v will receive that commit message. At this point, v can add edge $e = (u, v)$ to its set of incident edges H_v that will be in H or can respond with "no". After u receives v's message, they both can go back to receiving edges and committing to other nodes. The *DualCommit* Algorithm is run by each node and is given in Algorithm 1 .

▶ **Lemma 1.** *The DualCommit Algorithm terminates in $O(m)$ rounds.*

Proof. There is no wait operation and so this algorithm makes progress per edge received in the stream. On the arrival of edge $e = (u, v)$ either one of the two endpoints is interested in keeping the edge e or neither endpoint can keep edge e. If neither endpoint can keep the edge, then both discard the edge and there is one less edge in the stream to handle. Lets consider the case where exactly one endpoint is interested in keeping the edge. Say on edge $e = (u, v)$, $deg(u) < \beta$ and so node u is interested in keeping edge e but node v is not. Then node u will commit to node v and the stream progresses normally as is. As edges come, either node v responds to the commit message or node u waits and discards edges and other commit messages in the meantime. Lastly, consider the case where both endpoints are interested in keeping the edge $e = (u, v)$ that has just arrived. In this case, both nodes commit to each other and so both nodes know they can safely add this edge. Again, this edge is handled and the stream progresses. No case prevents the stream from progressing and so after all m edges have arrived, the end-of-stream token will terminate the algorithm. ◀

▶ **Lemma 2.** *If some edge $e = (u, v)$ is in H_u, then it is also in H_v.*

Proof. The only time an edge $e = (u, v)$ is added to either H_u or H_v is when both nodes have sent a commit message to each other. At this point both nodes have degree less than β. So, both endpoints are interested in adding the edge to their respective edge-sets and both nodes will. ◀

Algorithm 1 The DualCommit Algorithm for a node u.

```
Result: $H_u$ such that $|H_u| \leq \beta$ edges incident to node $u$ in $H$.
Input: $G = (V, E = (e_1, e_2, \dots, e_m), \beta \in \{1 \dots |V| - 1\}$;
$deg(u) \leftarrow 0$;
$committed \leftarrow false$;
$currentMatch \leftarrow NULL$;
$H_u \leftarrow \emptyset$;
On: awakening input
if (end of stream token arrived) then
    exit;
end
if ($committed = true$) then
    if (edge $e = (u, v)$ with $UID_u < UID_v$ arrived) then
        discard edge $e$;
    end
    if (commit message from $v \neq currentMatch$ arrived) then
        send "no" message to $v$;
    end
    if ("no" message from $currentMatch$ arrived) then
        discard edge $e$;
        $committed \leftarrow false$;
        $currentMatch \leftarrow NULL$;
    end
    if ("commit" message from $currentMatch$ arrived then
        $H_u \leftarrow H_u \cup \{u, v\}$;
        $deg(u) \leftarrow deg(u) + 1$:
        $committed \leftarrow false$;
        $currentMatch \leftarrow NULL$;
    end
else
    if (edge $e = (u, v)$ with $UID_u < UID_v$ arrived) then
        if (($deg(u) < \beta$)) then
            $u$ sends "commit" message to $v$;
            $committed \leftarrow true$;
            $currentMatch \leftarrow v$;
        end
    end
    if (commit message from $v$ arrived) then
        if (($deg(u) < \beta$)) then
            $u$ sends "commit" message to $v$;
            $H_u \leftarrow H_u \cup \{u, v\}$;
            $deg(u) \leftarrow deg(u) + 1$:
        end
    end
end
```

▶ **Lemma 3.** *There does not exist a node u such that $|H_u| > \beta$.*

Proof. The edge-set of a particular node u, H_u, is only made larger when $deg(u) < \beta$. In particular, the algorithm maintains this lemma as an invariant throughout its execution. ◀

▶ **Lemma 4.** *The DualCommit algorithm uses no more than $2m$ messages.*

Proof. If node u with smaller UID cannot accept an edge, then it will simply discard the edge. This results in no messages sent. If node u with smaller UID can accept an edge $e = (u, v)$, then either v will respond with a "no" message or a confirmatory "commit" message. Either way, this will result in two messages sent. Thus, the algorithm sends at most two messages per edge and so the total number of messages sent is $\leq 2m$. ◀

▶ **Lemma 5.** *The message size is bounded by $O(\log n)$.*

Proof. The message needs to carry at most the UID of the sender, UID of the recipient, a bit for the "no" value, and a bit for the "commit" value. This is bounded by $O(\log n)$. ◀

4 Conclusions and Future Works

In this paper, we presented a deterministic distributed protocol for computing a β-degree bounded subgraph in a fully, distributed streaming model. This algorithm is a stepping stone towards computing more complex subgraphs with desirable properties. We suspect that the $(\frac{2}{3} - \epsilon)$-approximate matching algorithm in random order streams of Bernstein et. al [2] can be extended to this fully distributed streaming model with some communication overhead.

Recently it was shown that we can do better than $\frac{2}{3}$-approximate matching in random order streams by Assadi et. al. [1]. Seeing if this algorithm, amongst many other streaming algorithms, can be simulated/implemented in this fully distributed streaming setting is an open work. We made strong assumptions about the system being fault free and the underlying communication network being fully connected. Can crash-tolerant, Byzantine fault-tolerant, etc. distributed protocols be designed for this algorithm? Additionally, even though we show that the resulting subgraph is a β-degree bounded subgraph, we cannot make any lower bound claims on each node's incident edge set. If we switch to a synchronous model of computation or make different assumptions, can we make better guarantees?

5 Acknowledgements

The author would like to thank Elena Grigorescu, Raymond Song, Young-San Lin and Tamalika Mukherjee for many fruitful discussions on the topic.

References

1 Sepehr Assadi and Soheil Behnezhad. Beating two-thirds for random-order streaming matching. *CoRR*, abs/2102.07011, 2021. URL: `https://arxiv.org/abs/2102.07011`, `arXiv:2102.07011`.

2 Aaron Bernstein. Improved bounds for matching in random-order streams. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 12:1–12:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.ICALP.2020.12`.

3 Graham Cormode. The continuous distributed monitoring model. *SIGMOD record*, 42(1):5–14, 2013.

4 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2):207–216, December 2005. `doi:10.1016/j.tcs.2005.09.013`.

5 Minos Garofalakis. *Distributed Data Streams*, pages 883–890. Springer US, Boston, MA, 2009. doi:10.1007/978-0-387-39940-9_137.

6 Phillip B. Gibbons and Srikanta Tirthapura. Estimating simple functions on the union of data streams. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '01, page 281–291, New York, NY, USA, 2001. Association for Computing Machinery. URL: https://doi-org.ezproxy.lib.purdue.edu/10.1145/378580.378687, doi:10.1145/378580.378687.

7 Daniel M. Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming*, pages 598–609, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

8 Andrew McGregor. Graph stream algorithms: A survey. *SIGMOD Rec.*, 43(1):9–20, May 2014. doi:10.1145/2627692.2627694.