Juho Hirvonen and Jukka Suomela:

Distributed Algorithms 2020

https://jukkasuomela.fi/da2020/ · October 1, 2023

Chapter 5
CONGEST Model: Bandwidth Limitations

In the previous chapter, we learned about the LOCAL model. We saw that with the help of unique identifiers, it is possible to gather the full information on a connected input graph in $O(\mathrm{diam}(G))$ rounds. To achieve this, we heavily abused the fact that we can send arbitrarily large messages. In this chapter we will see what can be done if we are only allowed to send small messages. With this restriction, we arrive at a model that is commonly known as the “CONGEST model”.

5.1 Definitions

Let A be a distributed algorithm that solves a problem Π on a graph family $\mathscr{F}$ in the LOCAL model. Assume that Msg_A is a countable set; without loss of generality, we can then assume that

$$\mathrm{Msg}_A = \mathbb{N},$$

that is, the messages are encoded as natural numbers. Now we say that A solves problem Π on graph family $\mathscr{F}$ in the CONGEST model if the following holds for some constant C: for any graph $G = (V, E) \in \mathscr{F}$, algorithm A only sends messages from the set $\{0, 1, \ldots, |V|^C\}$.

Put otherwise, we have the following *bandwidth restriction*: in each communication round, over each edge, we only send $O(\log n)$-bit messages, where n is the total number of nodes.

5.2 Examples

Assume that we have an algorithm A that is designed for the LOCAL model. Moreover, assume that during the execution of A on a graph $G = (V, E)$, in each communication round, we only need to send the following pieces of information over each edge:

- $O(1)$ node identifiers,
- $O(1)$ edges, encoded as a pair of node identifiers,
- $O(1)$ counters that take values from 0 to $\operatorname{diam}(G)$,
- $O(1)$ counters that take values from 0 to $|V|$,
- $O(1)$ counters that take values from 0 to $|E|$.

Now it is easy to see that we can encode all of this as a binary string with $O(\log n)$ bits. Hence A is not just an algorithm for the LOCAL model, but it is also an algorithm for the CONGEST model.

Many algorithms that we have encountered in this book so far are of the above form, and hence they are also CONGEST algorithms (see Exercise 5.1). However, there is a notable exception: the algorithm for gathering the entire network from Section 4.2. In this algorithm, we need to send messages of size up to $\Theta(n^2)$ bits:

- To encode the set of nodes, we may need up to $\Theta(n \log n)$ bits (a list of n identifiers, each of which is $\Theta(\log n)$ bits long).

- To encode the set of edges, we may need up to $\Theta(n^2)$ bits (the adjacency matrix).

While algorithms with a running time of $O(\operatorname{diam}(G))$ or $O(n)$ are trivial in the LOCAL model, this is no longer the case in the CONGEST model. Indeed, there are graph problems that *cannot* be solved in time $O(n)$ in the CONGEST model (see Exercise 5.6).

In this chapter, we will learn techniques that can be used to design efficient algorithms in the CONGEST model. We will use the all-pairs shortest path problem as the running example.

5.3 All-Pairs Shortest Path Problem

Throughout this chapter, we will assume that the input graph $G = (V, E)$ is connected, and as usual, we have $n = |V|$. In the *all-pairs shortest path* problem (APSP in brief), the goal is to find the distances between all pairs of nodes. More precisely, the local output of node $v \in V$ is

$$f(v) = \big\{(u, d) : u \in V,\ d = \operatorname{dist}_G(v, u)\big\}.$$

That is, v has to know the identities of all other nodes, as well as the shortest-path distance between itself and all other nodes.

Note that to represent the local output of a single node we need $\Theta(n \log n)$ bits, and just to transmit this information over a single edge we would need $\Theta(n)$ communication rounds. Indeed, we can prove that any algorithm that solves the APSP problem in the CONGEST model takes $\Omega(n)$ rounds—see Exercise 5.7.

In this chapter, we will present an optimal distributed algorithm for the APSP problem: it solves the problem in $O(n)$ rounds in the CONGEST model.

5.4 Single-Source Shortest Paths

As a warm-up, we will start with a much simpler problem. Assume that we have elected a leader $s \in V$, that is, there is precisely one node s with input 1 and all other nodes have input 0. We will design an algorithm such that each node $v \in V$ outputs

$$f(v) = \operatorname{dist}_G(s, v),$$

i.e., its shortest-path distance to leader s.

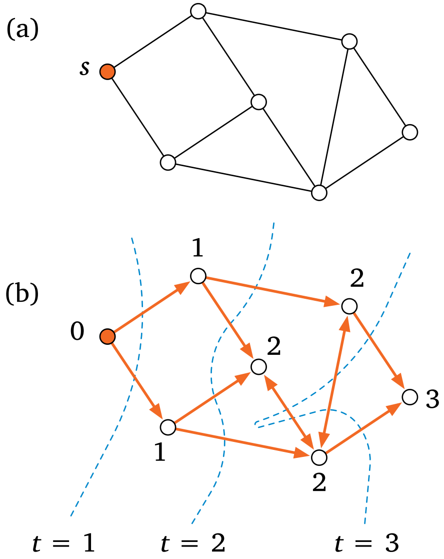

Figure 5.1: (a) Graph G and leader s. (b) Execution of algorithm Wave on graph G. The arrows denote ‘*wave*’ messages, and the dotted lines indicate the communication round during which these messages were sent.

The algorithm proceeds as follows. In the first round, the leader will send message ‘*wave*’ to all neighbors, switch to state 0, and stop. In round i, each node v proceeds as follows: if v has not stopped, and if it receives message ‘*wave*’ from some ports, it will send message ‘*wave*’ to all other ports, switch to state i, and stop; otherwise it does nothing. See Figure 5.1.

The analysis of the algorithm is simple. By induction, all nodes at distance i from s will receive message ‘*wave*’ from at least one port in round i, and they will hence output the correct value i. The running time of the algorithm is $O(\text{diam}(G))$ rounds in the CONGEST model.

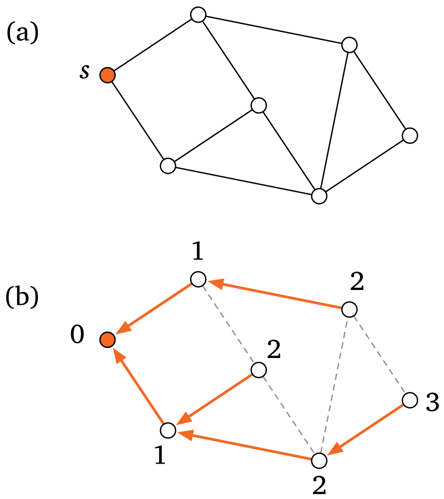

Figure 5.2: (a) Graph G and leader s. (b) BFS tree T (arrows) and distance labels $d(v)$ (numbers).

5.5 Breadth-First Search Tree

Algorithm Wave finds the shortest-path distances from a single source s. Now we will do something slightly more demanding: calculate not just the distances but also the shortest paths.

More precisely, our goal is to construct a *breadth-first search tree* (BFS tree) T rooted at s. This is a spanning subgraph $T = (V, E')$ of G such that T is a tree, and for each node $v \in V$, the shortest path from s to v in tree T is also a shortest path from s to v in graph G. We will also label each node $v \in V$ with a *distance label* $d(v)$, so that for each node $v \in V$ we have

$$d(v) = \mathrm{dist}_T(s, v) = \mathrm{dist}_G(s, v).$$

See Figure 5.2 for an illustration. We will interpret T as a directed graph, so that each edge is of form (u, v), where $d(u) > d(v)$, that is, the edges point towards the root s.

There is a simple centralized algorithm that constructs the BFS tree and distance labels: breadth-first search. We start with an empty tree and unlabeled nodes. First we label the leader s with $d(s) = 0$. Then in step $i = 0, 1, \ldots,$ we visit each node u with distance label $d(u) = i$, and check each neighbor v of u. If we have not labeled v yet, we will label it with $d(v) = i + 1$, and add the edge (v, u) to the BFS tree. This way all nodes that are at distance i from s in G will be labeled with the distance label i, and they will also be at distance i from s in T.

We can implement the same idea as a distributed algorithm in the CONGEST model. We will call this algorithm BFS. In the algorithm, each node v maintains the following variables:

- $d(v)$: distance to the root.
- $p(v)$: pointer to the parent of node v in tree T (port number).
- $C(v)$: the set of children of node v in tree T (port numbers).
- $a(v)$: acknowledgment—set to 1 when the subtree rooted at v has been constructed.

Here $a(v) = 1$ denotes a stopping state. When the algorithm stops, variables $d(v)$ will be distance labels, tree T is encoded in variables $p(v)$ and $C(v)$, and all nodes will have $a(v) = 1$.

Initially, we set $d(v) \leftarrow \perp$, $p(v) \leftarrow \perp$, $C(v) \leftarrow \perp$, and $a(v) \leftarrow 0$ for each node v, except for the root which has $d(s) = 0$. We will grow tree T from s by iterating the following steps:

- Each node v with $d(v) \neq \perp$ and $C(v) = \perp$ will send a *proposal* with value $d(v)$ to all neighbors.
- If a node u with $d(u) = \perp$ receives some proposals with value j, it will *accept* one of them and *reject* all other proposals. It will set $p(u)$ to point to the node whose proposal it accepted, and it will set $d(u) \leftarrow j + 1$.
- Each node v that sent some proposals will set $C(v)$ to be the set of neighbors that accepted proposals.

This way T will grow towards the leaf nodes. Once we reach a leaf node, we will send acknowledgments back towards the root:

- Each node v with $a(v) = 1$ and $p(v) \neq \perp$ will send an *acknowledgment* to port $p(v)$.
- Each node v with $a(v) = 0$ and $C(v) \neq \perp$ will set $a(v) \leftarrow 1$ when it has received acknowledgments from each port of $C(v)$. In particular, if a node has $C(v) = \emptyset$, it can set $a(v) \leftarrow 1$ without waiting for any acknowledgments.

It is straightforward to verify that the algorithm works correctly and constructs a BFS tree in $O(\operatorname{diam}(G))$ rounds in the CONGEST model.

Note that the acknowledgments would not be strictly necessary in order to construct the tree. However, they will be very helpful in the next section when we use algorithm BFS as a subroutine.

5.6 Leader Election

Algorithm BFS constructs a BFS tree rooted at a single leader, assuming that we have already elected a leader. Now we will show how to elect a leader. Surprisingly, we can use algorithm BFS to do it!

We will design an algorithm Leader that finds the node with the smallest identifier; this node will be the leader. The basic idea is very simple:

(a) We modify algorithm BFS so that we can run multiple copies of it in parallel, with different root nodes. We augment the messages with the identity of the root node, and each node keeps track of the variables d, p, C, and a separately for each possible root.

(b) Then we pretend that all nodes are leaders and start running BFS. In essence, we will run n copies of BFS in parallel, and hence we will construct n BFS trees, one rooted at each node. We will denote by BFS_v the BFS process rooted at node $v \in V$, and we will write T_v for the output of this process.

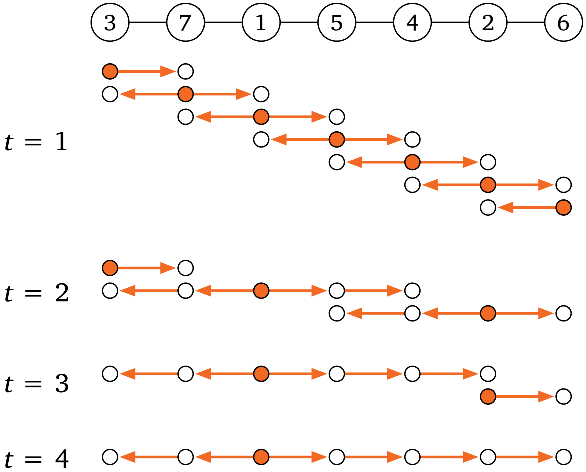

Figure 5.3: Leader election. Each node v will launch a process BFS_v that attempts to construct a BFS tree T_v rooted at v. Other nodes will happily follow BFS_v if v is the smallest leader they have seen so far; otherwise they will start to ignore messages related to BFS_v. Eventually, precisely one of the processes will complete successfully, while all other process will get stuck at some point. In this example, node 1 will be the leader, as it has the smallest identifier. Process BFS_2 will never succeed, as node 1 (as well as all other nodes that are aware of node 1) will ignore all messages related to BFS_2. Node 1 is the only root that will receive acknowledgments from every child.

However, there are two problems: First, it is not yet obvious how all this would help with leader election. Second, we cannot implement this idea directly in the CONGEST model—nodes would need to send up to n distinct messages per communication round, one per each BFS process, and there is not enough bandwidth for all those messages.

Fortunately, we can solve both of these issues very easily; see Figure 5.3:

(c) Each node will only send messages related to the tree that has the *smallest identifier as the root*. More precisely, for each node v, let $U(v) \subseteq V$ denote the set of nodes u such that v has received messages related to process BFS_u, and let $\ell(v) = \min U(v)$ be the smallest of these nodes. Then v will ignore messages related to process BFS_u for all $u \neq \ell(v)$, and it will only send messages related to process $\mathsf{BFS}_{\ell(v)}$.

We make the following observations:

- In each round, each node will only send messages related to at most one BFS process. Hence we have solved the second problem —this algorithm can be implemented in the CONGEST model.

- Let $s = \min V$ be the node with the smallest identifier. When messages related to BFS_s reach a node v, it will set $\ell(v) = s$ and never change it again. Hence all nodes will follow process BFS_s from start to end, and thanks to the acknowledgments, node s will eventually know that we have successfully constructed a BFS tree T_s rooted at it.

- Let $u \neq \min V$ be any other node. Now there is at least one node, s, that will ignore all messages related to process BFS_u. Hence BFS_u will never finish; node u will never receive the acknowledgments related to tree T_u from all neighbors.

That is, we now have an algorithm with the following properties: after $O(\operatorname{diam}(G))$ rounds, there is precisely one node s that knows that it is the unique node $s = \min V$. To finish the leader election process, node s will inform all other nodes that leader election is over; node s will output 1 and all other nodes will output 0 and stop.

5.7 All-Pairs Shortest Paths

Now we are ready to design algorithm APSP that solves the all-pairs shortest path problem (APSP) in time $O(n)$.

We already know how to find the shortest-path distances from a single source; this is efficiently solved with algorithm Wave. Just like we did with the BFS algorithm, we can also augment Wave with the root identifier and hence have a separate process Wave_v for each possible root $v \in V$. If we could somehow run all these processes in parallel, then each node would receive a wave from every other node, and hence each node would learn the distance to every other node, which is precisely what we need to do in the APSP problem. However, it is not obvious how to achieve a good performance in the CONGEST model:

- If we try to run all Wave_v processes simultaneously in parallel, we may need to send messages related to several waves simultaneously over a single edge, and there is not enough bandwidth to do that.

- If we try to run all Wave_v processes sequentially, it will take a lot of time: the running time would be $O(n \operatorname{diam}(G))$ instead of $O(n)$.

The solution is to *pipeline* the Wave_v processes so that we can have many of them running simultaneously in parallel, without congestion. In essence, we want to have multiple wavefronts active simultaneously so that they never collide with each other.

To achieve this, we start with the leader election and the construction of a BFS tree rooted at the leader; let s be the leader, and let T_s be the BFS tree. Then we do a *depth-first traversal* of T_s. This is a walk w_s in T_s that starts at s, ends at s, and traverses each edge precisely twice; see Figure 5.4.

More concretely, we move a *token* along walk w_s. We move the token *slowly*: we always spend 2 communication rounds before we move the token to an adjacent node. Whenever the token reaches a new node v that we have not encountered previously during the walk, we launch process Wave_v. This is sufficient to avoid all congestion!

The key observation here is that the token moves slower than the waves. The waves move at speed 1 edge per round (along the edges of G), while the token moves at speed 0.5 edges per round (along the edges of T_s, which is a subgraph of G). This guarantees that two waves never collide. To see this, consider two waves Wave_u and Wave_v, so that Wave_u was launched before Wave_v. Let $d = \operatorname{dist}_G(u, v)$. Then it will take

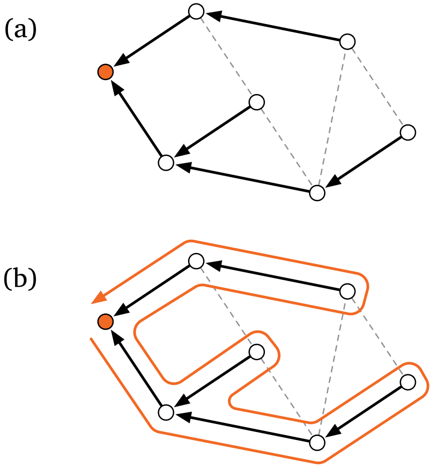

Figure 5.4: (a) BFS tree T_s rooted at s. (b) A depth-first traversal w_s of T_s.

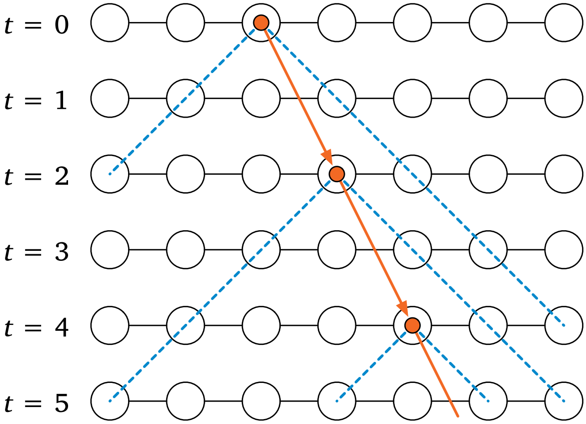

Figure 5.5: Algorithm APSP: the token walks along the BFS tree at speed 0.5 (thick arrows), while each Wave_v moves along the original graph at speed 1 (dashed lines). The waves are strictly nested: if Wave_v was triggered after Wave_u, it will never catch up with Wave_u.

at least $2d$ rounds to move the token from u to v, but only d rounds for Wave_u to reach node v. Hence Wave_u was already past v before we triggered Wave_v, and Wave_v will never catch up with Wave_u as both of them travel at the same speed. See Figure 5.5 for an illustration.

Hence we have an algorithm APSP that is able to trigger all Wave_v processes in $O(n)$ time, without collisions, and each of them completes $O(\operatorname{diam}(G))$ rounds after it was launched. Overall, it takes $O(n)$ rounds for all nodes to learn distances to all other nodes. Finally, the leader can inform everyone else when it is safe to stop and announce the local outputs (e.g., with the help of another wave).

5.8 Quiz

Consider the algorithm in Section 5.7 in a tree. Assume your tree T has 6 nodes, numbered from 1 to 6, and you have already elected node 1 as the leader. You have also already constructed the BFS tree rooted at the leader, which in this case is the original tree T. You would like to know how long it takes in the worst case to run the rest of the algorithms, i.e., let the token walk in tree T and let the waves propagate throughout the tree. Your task is to construct a worst-case example of a tree T so that the process takes as long as possible (i.e., you maximize the time until the final node learns about its distance to some other node).

5.9 Exercises

Exercise 5.1 (prior algorithms). In Chapters 3 and 4 we have seen examples of algorithms that were designed for the PN and LOCAL models. Many of these algorithms use only small messages—they can be used directly in the CONGEST model. Give at least four concrete examples of such algorithms, and prove that they indeed use only small messages.

Exercise 5.2 (edge counting). The *edge counting* problem is defined as follows: each node has to output the value $|E|$, i.e., it has to indicate how many edges there are in the graph.

Assume that the input graph is connected. Design an algorithm that solves the edge counting problem in the CONGEST model in time $O(\text{diam}(G))$.

Exercise 5.3 (detecting bipartite graphs)**.** Assume that the input graph is connected. Design an algorithm that solves the following problem in the CONGEST model in time $O(\text{diam}(G))$:

- If the input graph is bipartite, all nodes output 1.
- Otherwise all nodes output 0.

Exercise 5.4 (detecting complete graphs)**.** We say that a graph $G = (V, E)$ is *complete* if for all nodes $u, v \in V$, $u \neq v$, there is an edge $\{u, v\} \in E$.

Assume that the input graph is connected. Design an algorithm that solves the following problem in the CONGEST model in time $O(1)$:

- If the input graph is a complete graph, all nodes output 1.
- Otherwise all nodes output 0.

Exercise 5.5 (gathering)**.** Assume that the input graph is connected. In Section 4.2 we saw how to gather full information on the input graph in time $O(\text{diam}(G))$ in the LOCAL model. Design an algorithm that solves the problem in time $O(|E|)$ in the CONGEST model.

⋆ **Exercise 5.6** (gathering lower bounds)**.** Assume that the input graph is connected. Prove that there is no algorithm that gathers full information on the input graph in time $O(|V|)$ in the CONGEST model.

▹ *hint A*

⋆ **Exercise 5.7** (APSP lower bounds)**.** Assume that the input graph is connected. Prove that there is no algorithm that solves the APSP problem in time $o(|V|)$ in the CONGEST model.

5.10 Bibliographic Notes

The name CONGEST is from Peleg's [2] book. Algorithm APSP is due to Holzer and Wattenhofer [1]—surprisingly, it was published only as recently as in 2012.

5.11 Bibliography

[1] Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *Proc. 31st Annual ACM Symposium on Principles of Distributed Computing (PODC 2012)*, 2012. doi:10.1145/2332432.2332504.

[2] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2000.

5.12 Hints

A. To reach a contradiction, assume that A is an algorithm that solves the problem. For each n, let $\mathscr{F}(n)$ consists of all graphs with the following properties: there are n nodes with unique identifiers $1, 2, \dots, n$, the graph is connected, and the degree of node 1 is 1. Then compare the following two quantities as a function of n:

   (a) $f(n) =$ how many different graphs there are in family $\mathscr{F}(n)$.

   (b) $g(n) =$ how many different message sequences node number 1 may receive during the execution of algorithm A if we run it on any graph $G \in \mathscr{F}(n)$.

   Argue that for a sufficiently large n, we will have $f(n) > g(n)$. Then there are at least two different graphs $G_1, G_2 \in \mathscr{F}(n)$ such that node 1 receives the same information when we run A on either of these graphs.