Chapter 11

Hard Problems

This chapter is on "hard" problems in distributed computing. In sequential computing, there are NP-hard problems which are conjectured to take exponential time. Is there something similar in distributed computing? Using flooding/echo (Algorithms 11,12) from Chapter 3, everything so far was solvable basically in $\mathcal{O}(D)$ time, where D is the diameter of the network.

11.1 Diameter & APSP

But how do we compute the diameter itself!?! With flooding/echo, of course!

Algorithm 45 Naive Diameter Construction

all nodes compute their radius by synchronous flooding/echo
all nodes flood their radius on the constructed BFS tree
the maximum radius a node sees is the diameter

Remarks:

- Since all these phases only take $\mathcal{O}(D)$ time, nodes know the diameter in $\mathcal{O}(D)$ time, which is asymptotically optimal.
- However, there is a problem! Nodes are now involved in n parallel flooding/echo operations, thus a node may have to handle many and big messages in one single time step. Although this is not strictly illegal in the message passing model, it still feels like cheating! A natural question is whether we can do the same by just sending short messages in each round.
- In Definition 1.6 of Chapter 1 we postulated that nodes should send only messages of "reasonable" size. In this chapter we strengthen the definition a bit, and require that each message should have at most $\mathcal{O}(\log n)$ bits. This is generally enough to communicate a constant number of ID's or values to neighbors, but not enough to communicate everything a node knows!
- A simple way to avoid large messages is to split them into small messages that are sent using several rounds. This can cause that messages are

getting delayed in some nodes but not in others. The flooding might not use edges of a BFS tree anymore! These floodings might not compute correct distances anymore! On the other hand we know that the maximal message size in Algorithm 45 is $\mathcal{O}(n \log n)$. So we could just simulate each of these "big message" rounds by n "small message" rounds using small messages. This yields a runtime of $\mathcal{O}(nD)$ which is not desirable. A third possible approach is "starting each flooding/echo one after each other" and results in $\mathcal{O}(nD)$ in the worst case as well.

- So let us fix above algorithm! The key idea is to arrange the flooding-echo processes in a more organized way: Start the flooding processes in a certain order and prove that at any time, each node is only involved in one flooding. This is realized in Algorithm 46.

Definition 11.1. *(BFS_v) Performing a breadth first search at node v produces spanning tree BFS_v (see Chapter 3). This takes time $\mathcal{O}(D)$ using small messages.*

Remarks:

- A spanning tree of a graph G can be traversed in time $\mathcal{O}(n)$ by sending a pebble over an edge in each time slot.
- This can be done using e.g. a depth first search (DFS): Start at the root of a tree, recursively visit all nodes in the following way. If the current node still has an unvisited child, then the pebble always visits that child first. Return to the parent only when all children have been visited.
- Algorithm 46 works as follows: Given a graph G, first a leader l computes its BFS tree BFS_l. Then we send a pebble P to traverse tree BFS_l. Each time pebble P enters a node v for the first time, P waits one time slot, and then starts a breadth first search (BFS) – using edges in G – from v with the aim of computing the distances from v to all other nodes. Since we start a BFS_v from every node v, each node u learns its distance to all these nodes v during the according execution of BFS_v. There is no need for a echo-process at the end of BFS_u.

Algorithm 46 Computes APSP on G.

Assume we have a leader node l (if not, compute one first)
compute BFS_l of leader l
send a pebble P to traverse BFS_l in a DFS way;
while P traverses BFS_l **do**
  if P visits a new node v **then**
    wait one time slot; // avoid congestion
    start BFS_v from node v; // compute all distances to v
    // the depth of node u in BFS_v is $d(u,v)$
  end if
end while

Remarks:

- Having all distances is nice, but how do we get the diameter? Well, as before, each node could just floods its radius (its maximum distance) into the network. However, messages are small now and we need to modify this slightly. In each round a node only sends the maximal distance that it is aware of to its neighbors. After D rounds each node will know the maximum distance among all nodes.

Lemma 11.2. *In Algorithm 46, at no time a node w is simultaneously active for both BFS$_u$ and BFS$_v$.*

Proof. Assume a BFS$_u$ is started at time t_u at node u. Then node w will be involved in BFS$_u$ at time $t_u + d(u,w)$. Now, consider a node v whose BFS$_v$ is started at time $t_v > t_u$. According to the algorithm this implies that the pebble visits v after u and took some time to travel from u to v. In particular, the time to get from u to v is at least $d(u,v)$, in addition at least node v is visited for the first time (which involves waiting at least one time slot), and we have $t_v \geq t_u + d(u,v) + 1$. Using this and the triangle inequality, we get that node w is involved in BFS$_v$ strictly after being involved in BFS$_u$ since $t_v + d(v,w) \geq (t_u + d(u,v) + 1) + d(v,w) \geq t_u + d(u,w) + 1 > t_u + d(u,w)$. □

Theorem 11.3. *Algorithm 46 computes APSP (all pairs shortest path) in time $\mathcal{O}(n)$.*

Proof. Since the previous lemma holds for any pair of vertices, no two BFS "interfere" with each other, i.e. all messages can be sent on time without congestion. Hence, all BFS stop at most D time slots after they were started. We conclude that the runtime of the algorithm is determined by the time $\mathcal{O}(D)$ we need to build tree BFS$_l$, plus the time $\mathcal{O}(n)$ that P needs to traverse BFS$_l$, plus the time $\mathcal{O}(D)$ needed by the last BFS that P initiated. Since $D \leq n$, this is all in $\mathcal{O}(n)$. □

Remarks:

- All of a sudden our algorithm needs $\mathcal{O}(n)$ time, and possibly $n \gg D$. We should be able to do better, right?!
- Unfortunately not! One can show that computing the diameter of a network needs $\Omega(n/\log n)$ time.
- On the other hand we can check fast whether a graph has diameter 1 or not: each node just checks whether its degree is $n-1$ and tells the result to its neighbors.

11.2 Lower Bound Graphs

We define a family $\mathcal{G}$ of graphs that we use to prove a lower bound on the rounds needed to compute the diameter. To simplify our analysis, we assume that $(n-2)$ can be divided by 8. We start by defining four sets of nodes, each consisting of $q = q(n) := (n-2)/4$ nodes. Throughout this chapter we write $[q]$ as a short version of $\{1, \ldots, q\}$ and define:

$$
\begin{array}{llll}
\mathbf{L_0} & := & \{l_i \mid i \in [q]\} & \text{// upper left in Figure 11.1} \\
\mathbf{L_1} & := & \{l'_i \mid i \in [q]\} & \text{// lower left} \\
\mathbf{R_0} & := & \{r_i \mid i \in [q]\} & \text{// upper right} \\
\mathbf{R_1} & := & \{r'_i \mid i \in [q]\} & \text{// lower right}
\end{array}
$$

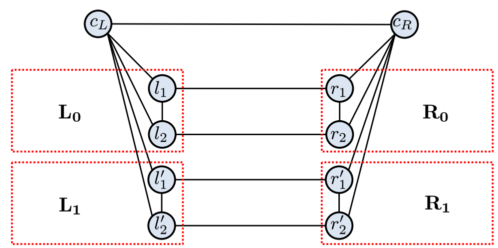

Figure 11.1: The above skeleton G' contains $n = 10$ nodes, such that $q = 2$.

We add node c_L and connect it to all nodes in $\mathbf{L_0}$ and $\mathbf{L_1}$. Then we add node c_R, connected to all nodes in $\mathbf{R_0}$ and $\mathbf{R_1}$. Furthermore, nodes c_L and c_R are connected by an edge. For $i \in [q]$ we connect l_i to r_i and l'_i to r'_i. Also we add edges such that nodes in $\mathbf{L_0}$ are a clique, nodes in $\mathbf{L_1}$ are a clique, nodes in $\mathbf{R_0}$ are a clique, and nodes in $\mathbf{R_1}$ are a clique. The resulting graph is called G'. Graph G' is the skeleton of any graph in family $\mathcal{G}$.

More formally skeleton $G' = (V', E')$ is:

$$V' := \mathbf{L_0} \cup \mathbf{L_1} \cup \mathbf{R_0} \cup \mathbf{R_1} \cup \{c_L, c_R\}$$

$$
\begin{array}{lll}
E' := & \bigcup_{v \in \mathbf{L_0} \cup \mathbf{L_1}} \{(v, c_L)\} & \text{// connections to } c_L \\
\cup & \bigcup_{v \in \mathbf{R_0} \cup \mathbf{R_1}} \{(v, c_R)\} & \text{// connections to } c_R \\
\cup & \bigcup_{i \in [q]} \{(l_i, r_i), (l'_i, r'_i)\} \cup \{(c_L, c_R)\} & \text{// connects left to right} \\
\cup & \bigcup_{S \in \{\mathbf{L_0}, \mathbf{L_1}, \mathbf{R_0}, \mathbf{R_1}\}} \bigcup_{u \neq v \in S} \{(u, v)\} & \text{// clique edges}
\end{array}
$$

To simplify our arguments, we partition G' into two parts: **Part L** is the subgraph induced by nodes $\mathbf{L_0} \cup \mathbf{L_1} \cup \{c_L\}$. **Part R** is the subgraph induced by nodes $\mathbf{R_0} \cup \mathbf{R_1} \cup \{c_R\}$.

Family $\mathcal{G}$ contains any graph G that is derived from G' by adding any combination of edges of the form (l_i, l'_j) resp. (r_i, r'_j) with $l_i \in \mathbf{L_0}$, $l'_j \in \mathbf{L_1}$, $r_i \in \mathbf{R_0}$, and $r'_j \in \mathbf{R_1}$.

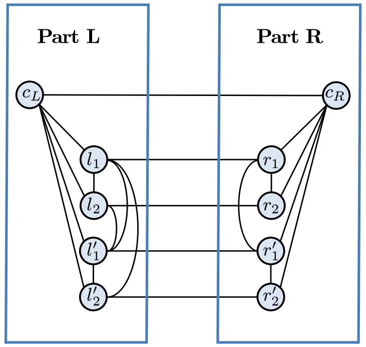

Figure 11.2: The above graph G has $n = 10$ and a member of family $\mathcal{G}$. What is the diameter of G?

Lemma 11.4. *The diameter of a graph $G = (V, E) \in \mathcal{G}$ is 2 if and only if: For each tuple (i, j) with $i, j \in [q]$, there is either edge (l_i, l'_j) or edge (r_i, r'_j) (or both edges) in E.*

Proof. Note that the distance between most pairs of nodes is at most 2. In particular, the radius of c_L resp. c_R is 2. Thanks to c_L resp. c_R the distance between, any two nodes within **Part L** resp. within **Part R** is at most 2. Because of the cliques $\mathbf{L_0}, \mathbf{L_1}, \mathbf{R_0}, \mathbf{R_1}$, distances between l_i and r_j resp. l'_i and r'_j is at most 2.

The only interesting case is between a node $l_i \in \mathbf{L_0}$ and node $r'_j \in \mathbf{R_1}$ (or, symmetrically, between $l'_j \in \mathbf{L_1}$ and node $r_i \in \mathbf{R_0}$). If either edge (l_i, l'_j) or edge (r_i, r'_j) is present, then this distance is 2, since the path (l_i, l'_j, r'_j) or the path (l_i, r_i, r'_j) exists. If neither of the two edges exist, then the neighborhood of l_i consists of $\{c_L, r_i\}$, all nodes in $\mathbf{L_0}$, and some nodes in $\mathbf{L_1} \setminus \{l'_j\}$, and the neighborhood of r'_j consists of $\{c_R, l'_j\}$, all nodes in $\mathbf{R_1}$, and some nodes in $\mathbf{R_0} \setminus \{r_i\}$ (see for example Figure 11.3 with $i = 1$ and $j = 2$.) Since the two neighborhoods do not share a common node, the distance between l_i and r'_j is (at least) 3. □

Remarks:

- Each part contains up to $q^2 \in \Theta(n^2)$ edges.
- There are $2q + 1 \in \Theta(n)$ edges connecting the left and the right part. Since in each round we can transmit $\mathcal{O}(\log n)$ bits over each edge (in each direction), the bandwidth between **Part L** and **Part R** is $\mathcal{O}(n \log n)$.

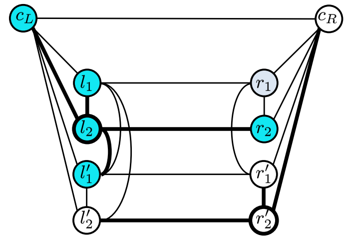

Figure 11.3: Nodes in the neighborhood of l_2 are cyan, the neighborhood of r'_2 is white. Since these neighborhoods do not intersect, the distance of these two nodes is $d(l_2, r'_2) > 2$. If e.g. edge (l_2, l'_2) was included, their distance was 2.

- If we transmit the information of the $\Theta(n^2)$ edges in a naive way with a bandwidth of $\mathcal{O}(n \log n)$, we need $\Omega(n/\log n)$ time. But maybe we can do better?!? Can an algorithm be smarter and only send the information that is really necessary to tell whether the diameter is 2?
- It turns out that any algorithm needs $\Omega(n/\log n)$ rounds, since the information that is really necessary to tell that the diameter is larger than 2 contains basically $\Theta(n^2)$ bits.

11.3 Communication Complexity

To prove the last remark formally, we can use arguments from two-party communication complexity. This area essentially deals with a basic version of distributed computation: two parties are given some input each and want to solve a task on this input.

We consider two students (Alice and Bob) at two different universities connected by a communication channel (e.g. via email) and we assume this channel to be reliable. Now Alice and Bob want to check whether they received the same problem set for homework (we assume their professors are lazy and wrote it on the black board instead of putting a nicely prepared document online.) Do Alice and Bob really need to type the whole problem set into their emails? In a more formal way: Alice receives an n-bit string x and Bob another n-bit string y, and the goal is for both of them to compute the equality function.

Definition 11.5. *(Equality.) We define the equality function* EQ *to be:*

$$\mathrm{EQ}(x, y) := \begin{cases} 1 & : x = y \\ 0 & : x \neq y \,. \end{cases}$$

Remarks:

- In a more general setting, Alice and Bob are interested in computing a certain function $f : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}$ with the least amount of communication between them. Of course they can always succeed by having Alice send her whole n-bit string to Bob, who then computes the function, but the idea here is to find clever ways of calculating f with less than n bits of communication. We measure how clever they can be as follows:

Definition 11.6. *(Communication complexity CC.) The communication complexity of protocol A for function f is $CC(A, f) :=$ minimum number of bits exchanged between Alice and Bob in the worst case when using A. The communication complexity of f is $CC(f) := \min\{CC(A, f) \mid A \text{ solves } f\}$. That is the minimal number of bits that the best protocol needs to send in the worst case.*

Definition 11.7. *For a given function f, we define a $2^k \times 2^k$ matrix M^f representing f. That is $M^f_{x,y} := f(x, y)$.*

Example 11.8. *For* EQ*, in case $k = 3$, matrix M^{EQ} looks like this:*

EQ	000	001	010	011	100	101	110	111	$\leftarrow x$
000	1	0	0	0	0	0	0	0	
001	0	1	0	0	0	0	0	0	
010	0	0	1	0	0	0	0	0	
011	0	0	0	1	0	0	0	0	
100	0	0	0	0	1	0	0	0	
101	0	0	0	0	0	1	0	0	
110	0	0	0	0	0	0	1	0	
111	0	0	0	0	0	0	0	1	
$\uparrow y$									

As a next step we define a (combinatorial) monochromatic rectangle. These are "submatrices" of M^f which contain the same entry.

Definition 11.9. *(monochromatic rectangle.) A set $R \subseteq \{0,1\}^k \times \{0,1\}^k$ is called a monochromatic rectangle, if*

- *whenever $(x_1, y_1) \in R$ and $(x_2, y_2) \in R$ then $(x_1, y_2) \in R$.*
- *there is a fixed z such that $f(x, y) = z$ for all $(x, y) \in R$.*

Example 11.10. *The first three of the following rectangles are monochromatic, the last one is not:*

$R_1 = \{011\} \times \{011\}$	*Example 11.8:*	*light gray*
$R_2 = \{011, 100, 101, 110\} \times \{000, 001\}$	*Example 11.8:*	*gray*
$R_3 = \{000, 001, 101\} \times \{011, 100, 110, 111\}$	*Example 11.8:*	*dark gray*
$R_4 = \{000, 001\} \times \{000, 001\}$	*Example 11.8:*	*boxed*

Each time Alice and Bob exchange a bit, they can eliminate columns/rows of the matrix M^f and a combinatorial rectangle is left. They can stop communicating when this remaining rectangle is monochromatic. Informally speaking, a fooling set can be used to fool a protocol that wants to be lazy: if the fooling

set is large, there will be many maximal monochromatic rectangles (maximal in the sense that they cannot be extended while staying monochromatic). Since by communicating one bit the set of possible monochromatic rectangles does not shrink too much, we can expect that it takes long time until a monochromatic rectangle is found in the worst case.

Definition 11.11. *(fooling set.) A set $S \subset \{0,1\}^k \times \{0,1\}^k$ fools f if there is a fixed z such that*

- *$f(x, y) = z$ for each $(x, y) \in S$*
- *For any $(x_1, y_1) \neq (x_2, y_2) \in S$, the rectangle $\{x_1, x_2\} \times \{y_1, y_2\}$ is not monochromatic: Either $f(x_1, y_2) \neq z$, $f(x_2, y_1) \neq z$ or both $\neq z$.*

Example 11.12. *Consider $S = \{(000, 000), (001, 001)\}$. Take a look at the non-monochromatic rectangle R_4 in Example 11.10. Verify that S is indeed a fooling set for* EQ*!*

Remarks:

- Can you find a larger fooling set for *EQ*?
- We assume that Alice and Bob take turns in sending a bit. This results in 2 possible actions (send 0/1) per round and in 2^t *action patterns* during a sequence of t rounds.

Lemma 11.13. *If S is a fooling set for f, then $CC(f) = \Omega(\log |S|)$.*

Proof. For simplicity we assume that $|S|$ is a power of 2. We prove the statement via contradiction: fix a protocol A and assume that it needs $t := \log_2(|S|/2)$ rounds in the worst case. Then there are $2^t = 2^{\log_2(|S|/2)} = |S|/2$ possible action patterns. On the other hand there are $|S| = 2^{\log_2 |S|}$ elements in S and we conclude that at least two elements (let's call them $(x_1, y_1), (x_2, y_2)$) in S cause the same action pattern P. Naturally, the action pattern on the alternative inputs $(x_1, y_2), (x_2, y_1)$ will be P as well: in the first round Alice and Bob have no information on the other party's string and send the same bit that was sent in P. Based on this, they determine the second bit to be exchanged, which will be the same as the second one in P for a similar reason. This continues for all t rounds. We conclude that after t rounds, Alice does not know whether Bob's input is y_1 or y_2 and Bob does not know whether Alice's input is x_1 or x_2. By the definition of fooling set, either

- $f(x_1, y_2) \neq f(x_1, y_1)$ in which case Alice (with input x_1) does not know the solution yet,

or

- $f(x_2, y_1) \neq f(x_1, y_1)$ in which case Bob (with input y_1) does not know the solution yet.

This contradicts the assumption that A leads to a correct decision for all inputs after t rounds. Therefore at least $t + 1$ rounds are necessary, which is

$$t + 1 = \log_2(|S|/2) + 1 = \frac{\log |S|}{\log 2} \in \Omega(\log |S|).$$

□

Theorem 11.14. $CC(\mathrm{EQ}) = \Omega(k)$.

Proof. The set $S := \{(x,x) \mid x \in \{0,1\}^k\}$ fools EQ and has size 2^k. Now apply Lemma 11.13. □

Definition 11.15. *Denote the negation of a string z by $\overline{z}$ and by $x \circ y$ the concatenation of strings x and y.*

Lemma 11.16. *Let x, y be k-bit strings. Then $x \neq y$ if and only if there is an index $i \in [2k]$ such that the i^{th} bit of $x \circ \overline{x}$ and the i^{th} bit of $\overline{y} \circ y$ are both 0.*

Proof. If $x \neq y$, there is an $j \in [k]$ such that x and y differ in the j^{th} bit. Therefore either the j^{th} bit of both x and $\overline{y}$ is 0, or the j^{th} bit of $\overline{x}$ and y is 0. For this reason, there is an $i \in [2k]$ such that $x \circ \overline{x}$ and $\overline{y} \circ y$ are both 0 at position i.

If $x = y$, then for any $i \in [2k]$ it is always the case that either the i^{th} bit of $x \circ \overline{x}$ is 1 or the i^{th} bit of $\overline{y} \circ y$ (which is the negation of $x \circ \overline{x}$ in this case) is 1. □

Remarks:

- With these insights we get back to the problem of computing the diameter of a graph and relate this problem to EQ.

Definition 11.17. *Using the parameter q defined before, we define a bijective map between all pairs x, y of q^2-bit strings and the graphs in $\mathcal{G}$: each pair of strings x, y is mapped to graph $G_{x,y} \in \mathcal{G}$ that is derived from skeleton G' by adding*

- *edge (l_i, l'_j) to **Part L** if and only if the $(j + q \cdot i)^{th}$ bit of x is 1.*
- *edge (r_i, r'_j) to **Part R** if and only if the $(j + q \cdot i)^{th}$ bit of y is 1.*

Remarks:

- Clearly, **Part L** of $G_{x,y}$ depends on x only and **Part R** depends on y only.

Lemma 11.18. *Let x and y be $\frac{q^2}{2}$-bit strings given to Alice and Bob*[1]*. Then graph $G := G_{x \circ \overline{x}, \overline{y} \circ y} \in \mathcal{G}$ has diameter 2 if and only if $x = y$.*

Proof. By Lemma 11.16 and the construction of G, there is neither edge (l_i, l'_j) nor edge (r_i, r'_j) in E if and only if $x \neq y$. Applying Lemma 11.4 yields: G has diameter 2 if and only if $x = y$. □

Theorem 11.19. *Any distributed algorithm A that decides whether a graph G has diameter 2 might need $\Omega\left(\frac{n}{\log n} + D\right)$ time.*

Proof. Computing D for sure needs time $\Omega(D)$. It remains to prove $\Omega\left(\frac{n}{\log n}\right)$. Assume there is a distributed algorithm A that decides whether the diameter of a graph is 2 in time $o(n/\log n)$. When Alice and Bob are given $\frac{q^2}{2}$-bit inputs x and y, they can simulate A to decide whether $x = y$ as follows: Alice constructs

[1] Thats why we need that $n - 2$ can be divided by 8.

Part L of $G_{x \circ \overline{x}, \overline{y} \circ y}$ and Bob constructs **Part R**. As we remarked, both parts are independent of each other such that **Part L** can be constructed by Alice without knowing y and **Part R** can be constructed by Bob without knowing x. Furthermore, $G_{x \circ \overline{x}, \overline{y} \circ y}$ has diameter 2 if and only if $x = y$ (Lemma 11.18.)

Now Alice and Bob simulate the distributed algorithm A round by round: In the first round, they determine which messages the nodes in their part of G would send. Then they use their communication channel to exchange all $2(2q+1) \in \Theta(n)$ messages that would be sent over edges between **Part L** and **Part R** in this round while executing A on G. Based on this Alice and Bob determine which messages would be sent in round two and so on. For each round simulated by Alice and Bob, they need to communicate $\Theta(n \log n)$ bits: possibly $\Theta(\log n)$ bits for each of $\Theta(n)$ messages. Since A makes a decision after $o(n/\log n)$ rounds, this yields a total communication of $o(n^2)$ bits. On the other hand, Lemma 11.14 states that to decide whether x equals y, Alice and Bob need to communicate at least $\Omega\left(\frac{q^2}{2}\right) = \Omega(n^2)$ bits. A contradiction. □

Remarks:

- Until now we only considered deterministic algorithms. Can one do better using randomness?

Algorithm 47 Randomized evaluation of EQ.

1: Alice and Bob use public randomness. That is they both have access to the same random bit string $z \in \{0,1\}^k$
2: Alice sends bit $a := \sum_{i \in [k]} x_i \cdot z_i \mod 2$ to Bob
3: Bob sends bit $b := \sum_{i \in [k]} y_i \cdot z_i \mod 2$ to Alice
4: **if** $a \neq b$ **then**
5: we know $x \neq y$
6: **end if**

Lemma 11.20. *If $x \neq y$, Algorithm 47 discovers $x \neq y$ with probability at least 1/2.*

Proof. Note that if $x = y$ we have $a = b$ for sure.

If $x \neq y$, Algorithm 47 may not reveal inequality. For instance, for $k = 2$, if $x = 01$, $y = 10$ and $z = 11$ we get $a = b = 1$. In general, let I be the set of indices where $x_i \neq y_i$, i.e. $I := \{i \in [k] \mid x_i \neq y_i\}$. Since $x \neq y$, we know that $|I| > 0$. We have

$$|a - b| \equiv \sum_{i \in I} z_i \pmod 2,$$

and since all z_i with $i \in I$ are random, we get that $a \neq b$ with probability at least 1/2. □

Remarks:

- By excluding the vector $z = 0^k$ we can even get a discovery probability strictly larger than 1/2.

- Repeating the Algorithm 47 with different random strings z, the error probability can be reduced arbitrarily.

- Does this imply that there is a fast randomized algorithm to determine the diameter? Unfortunately not!

- Sometimes public randomness is not available, but private randomness is. Here Alice has her own random string and Bob has his own random string. A modified version of Algorithm 47 also works with private randomness at the cost of the runtime.

- One can prove an $\Omega(n/\log n)$ lower bound for any randomized distributed algorithm that computes the diameter. To do so one considers the disjointness function *DISJ* instead of equality. Here, Alice is given a subset $X \subseteq [k]$ and and Bob is given a subset $Y \subseteq [k]$ and they need to determine whether $Y \cap X = \emptyset$. (X and Y can be represented by k-bit strings x, y.) The reduction is similar as the one presented above but uses graph $G_{\overline{x},\overline{y}}$ instead of $G_{x\circ\overline{x},\overline{y}\circ y}$. However, the lower bound for the randomized communication complexity of *DISJ* is more involved than the lower bound for $CC(EQ)$.

- Since one can compute the diameter given a solution for APSP, an $\Omega(n/\log n)$ lower bound for APSP is implied. As such, our simple Algorithm 46 is almost optimal!

- Many prominent functions allow for a low communication complexity. For instance, $CC(PARITY) = 2$. What is the Hamming distance (number of different entries) of two strings? It is known that $CC(HAM \geq d) = \Omega(d)$. Also, CC(decide whether "$HAM \geq k/2 + \sqrt{k}$" or "$HAM \leq k/2 - \sqrt{k}$") $= \Omega(k)$, even when using randomness. This problem is known as the Gap-Hamming-Distance.

- Lower bounds in communication complexity have many applications. Apart from getting lower bounds in distributed computing, one can also get lower bounds regarding circuit depth or query times for static data structures.

- In the distributed setting with limited bandwidth we showed that computing the diameter has about the same complexity as computing all pairs shortest paths. In contrast, in sequential computing, it is a major open problem whether the diameter can be computed faster than all pairs shortest paths. No nontrivial lower bounds are known, only that $\Omega(n^2)$ steps are needed – partly due to the fact that there can be n^2 edges/distances in a graph. On the other hand the currently best algorithm uses fast matrix multiplication and terminates after $\mathcal{O}(n^{2.3727})$ steps.

11.4 Distributed Complexity Theory

We conclude this chapter with a short overview on the main complexity classes of distributed message passing algorithms. Given a network with n nodes and diameter D, we managed to establish a rich selection of upper and lower bounds regarding how much time it takes to solve or approximate a problem. Currently we know five main distributed complexity classes:

- Strictly *local* problems can be solved in constant $\mathcal{O}(1)$ time, e.g. a constant approximation of a dominating set in a planar graph.

- Just a little bit slower are problems that can be solved in *log-star* $\mathcal{O}(\log^* n)$ time, e.g. many combinatorial optimization problems in special graph classes such as growth bounded graphs. 3-coloring a ring takes $\mathcal{O}(\log^* n)$.

- A large body of problems is *polylogarithmic* (or *pseudo-local*), in the sense that they seem to be strictly local but are not, as they need $\mathcal{O}(\text{polylog } n)$ time, e.g. the maximal independent set problem.

- There are problems which are *global* and need $\mathcal{O}(D)$ time, e.g. to count the number of nodes in the network.

- Finally there are problems which need *polynomial* $\mathcal{O}(\text{poly } n)$ time, even if the diameter D is a constant, e.g. computing the diameter of the network.

Chapter Notes

The linear time algorithm for computing the diameter was discovered independently by [HW12, PRT12]. The presented matching lower bound is by Frischknecht et al. [FHW12], extending techniques by [DHK+11].

Due to its importance in network design, shortest path-problems in general and the APSP problem in particular were among the earliest studied problems in distributed computing. Developed algorithms were immediately used e.g. as early as in 1969 in the ARPANET (see [Lyn96], p.506). Routing messages via shortest paths were extensively discussed to be beneficial in [Taj77, MS79, MRR80, SS80, CM82] and in many other papers. It is not surprising that there is plenty of literature dealing with algorithms for distributed APSP, but most of them focused on secondary targets such as trading time for message complexity. E.g. papers [AR78, Tou80, Che82] obtain a communication complexity of roughly $\mathcal{O}(n \cdot m)$ bits/messages and still require superlinear runtime. Also a lot of effort was spent to obtain fast sequential algorithms for various versions of computing APSP or related problems such as the diameter problem, e.g. [CW90, AGM91, AMGN92, Sei95, SZ99, BVW08]. These algorithms are based on fast matrix multiplication such that currently the best runtime is $\mathcal{O}(n^{2.3727})$ due to [Wil12].

The problem sets in which one needs to distinguish diameter 2 from 4 are inspired by a combinatorial $(\times, 3/2)$-approximation in a sequential setting by Aingworth et. al. [ACIM99]. The main idea behind this approximation is to distinguish diameter 2 from 4. This part was transferred to the distributed setting in [HW12].

Two-party communication complexity was introduced by Andy Yao in [Yao79]. Later, Yao received the Turing Award. A nice introduction to communication complexity covering techniques such as fooling-sets is the book by Nisan and Kushilevitz [KN97].

This chapter was written in collaboration with Stephan Holzer.

Bibliography

[ACIM99] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast Estimation of Diameter and Shortest Paths (Without Matrix Multiplication). *SIAM Journal on Computing (SICOMP)*, 28(4):1167–1181, 1999.

[AGM91] N. Alon, Z. Galil, and O. Margalit. On the exponent of the all pairs shortest path problem. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 569–575, 1991.

[AMGN92] N. Alon, O. Margalit, Z. Galilt, and M. Naor. Witnesses for Boolean Matrix Multiplication and for Shortest Paths. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 417–426. IEEE Computer Society, 1992.

[AR78] J.M. Abram and IB Rhodes. A decentralized shortest path algorithm. In *Proceedings of the 16th Allerton Conference on Communication, Control and Computing (Allerton)*, pages 271–277, 1978.

[BVW08] G.E. Blelloch, V. Vassilevska, and R. Williams. A New Combinatorial Approach for Sparse Graph Problems. In *Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part I (ICALP)*, pages 108–120. Springer-Verlag, 2008.

[Che82] C.C. Chen. A distributed algorithm for shortest paths. *IEEE Transactions on Computers (TC)*, 100(9):898–899, 1982.

[CM82] K.M. Chandy and J. Misra. Distributed computation on graphs: Shortest path algorithms. *Communications of the ACM (CACM)*, 25(11):833–837, 1982.

[CW90] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of symbolic computation (JSC)*, 9(3):251–280, 1990.

[DHK+11] A. Das Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer. Distributed Verification and Hardness of Distributed Approximation. *Proceedings of the 43rd annual ACM Symposium on Theory of Computing (STOC)*, 2011.

[FHW12] S. Frischknecht, S. Holzer, and R. Wattenhofer. Networks Cannot Compute Their Diameter in Sublinear Time. In *Proceedings of the 23rd annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1150–1162, January 2012.

[HW12] Stephan Holzer and Roger Wattenhofer. Optimal Distributed All Pairs Shortest Paths and Applications. In *PODC*, page to appear, 2012.

[KN97] E. Kushilevitz and N. Nisan. *Communication complexity*. Cambridge University Press, 1997.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[MRR80] J. McQuillan, I. Richer, and E. Rosen. The new routing algorithm for the ARPANET. *IEEE Transactions on Communications (TC)*, 28(5):711–719, 1980.

[MS79] P. Merlin and A. Segall. A failsafe distributed routing protocol. *IEEE Transactions on Communications (TC)*, 27(9):1280–1287, 1979.

[PRT12] David Peleg, Liam Roditty, and Elad Tal. Distributed Algorithms for Network Diameter and Girth. In *ICALP*, page to appear, 2012.

[Sei95] R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences (JCSS)*, 51(3):400–403, 1995.

[SS80] M. Schwartz and T. Stern. Routing techniques used in computer communication networks. *IEEE Transactions on Communications (TC)*, 28(4):539–552, 1980.

[SZ99] A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 605–614. IEEE, 1999.

[Taj77] W.D. Tajibnapis. A correctness proof of a topology information maintenance protocol for a distributed computer network. *Communications of the ACM (CACM)*, 20(7):477–485, 1977.

[Tou80] S. Toueg. An all-pairs shortest-paths distributed algorithm. *Tech. Rep. RC 8327, IBM TJ Watson Research Center, Yorktown Heights, NY 10598, USA*, 1980.

[Wil12] V.V. Williams. Multiplying Matrices Faster Than Coppersmith-Winograd. *Proceedings of the 44th annual ACM Symposium on Theory of Computing (STOC)*, 2012.

[Yao79] A.C.C. Yao. Some complexity questions related to distributive computing. In *Proceedings of the 11th annual ACM symposium on Theory of computing (STOC)*, pages 209–213. ACM, 1979.