ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

FS 2020

Prof. Dr. Mohsen Ghaffari,
Prof. Dr. Roger Wattenhofer

Exam
Principles of Distributed Computing

Thursday, August 13, 2020
15:00 – 18:00

Do not open or turn until told to by the supervisor!

The exam lasts 180 minutes, and there is a total of 180 points. The maximal number of points for each question is indicated in parentheses. Your answers must be in English. Be sure to always justify (prove) your answers. Algorithms can be specified in high-level pseudocode or as a verbal description. You do not need to give every last detail, but the main aspects need to be there. Big-O notation is acceptable when giving algorithmic complexities. Please write legibly. If we cannot read your answers, we cannot grade them.

Please write down your name and Legi number (your student ID) in the following fields.

Name	Legi-Nr.

Exercise	**Achieved Points**	**Maximal Points**
1 - Multiple Choice		24
2 - Partial Coloring		25
3 - Labeling Elections		29
4 - Disjoint Walks		25
5 - Fault-Tolerant Sorting Networks		24
6 - Shared Graph		28
7 - Heaviest Connections in MST		25
Total		**180**

1 Multiple Choice (24 points)

Evaluate each of the following statements in terms of correctness. Indicate whether a statement is true or false by ticking the corresponding box. Each correct answer gives one point. Each wrong answer and each unanswered question gives 0 points.

A) [3] Consider the asynchronous model and the synchronous model.

	true	**false**
Any problem that can be solved in the synchronous model can be solved in the asynchronous model.	☐	☐
Any problem that can be solved in the asynchronous model can be solved in the synchronous model.	☐	☐
If a problem can be solved in both models, the time complexity of the optimal solution in the synchronous model is not worse than the time complexity of the optimal solution in the asynchronous model.	☐	☐

B) [3] Given an unoriented tree with constant degree, in the LOCAL model we can deterministically color it with a constant number of colors in...

	true	**false**
... $O(\log n)$ rounds.	☐	☐
... $O(\log^* n)$ rounds.	☐	☐
... $O(1)$ rounds.	☐	☐

C) [3] Given a cycle of n nodes, for the problem of deterministically computing a set of marked nodes, such that each two marked nodes have distance at least 10, and each unmarked node has a marked node within distance 100, the following bounds hold for the round complexity in the LOCAL model:

	true	**false**
$O(\log n)$.	☐	☐
$\Omega(\log n)$.	☐	☐
$\Omega(\log^* n)$.	☐	☐

D) [3] Given a synchronous network with n nodes and diameter D, and messages consist of 1 bit (can communicate with every neighbor in every round). Each node has a $\log n$ bit string. Detecting if all strings are equal takes ...

	true	**false**
(at least) $\Omega(n)$ time.	☐	☐
$\Omega(D)$ time.	☐	☐
$\Omega(D + \log n)$ time.	☐	☐

E) [3] The slotted aloha algorithm (Algorithm 13.1) ...

	true	**false**
elects a leader in $O(1)$ time with high probability.	☐	☐
also works in a uniform setting (i.e. if the nodes do not know n).	☐	☐
does not require collision detection.	☐	☐

F) [3] Consider the technique of introducing collision detection into a network through a specific leader node (Table 13.7).

	true	**false**
The method enables every node to have collision detection, except for the leader.	☐	☐
When at least 2 nodes submit in the same time slot, the listening nodes can decide whether the leader was one of the nodes that transmitted.	☐	☐
The method also works if there are two different nodes that execute the role of the leader simultaneously.	☐	☐

G) [3] Consider the Maximal Independent Set (MIS) problem and Luby's algorithm for solving it.

	true	**false**
In Luby's MIS algorithm, the probability of each node being removed in the first round is at least a positive constant.	☐	☐
A graph on n vertices with maximum degree Δ contains an independent set of size at least $\frac{n}{\Delta+1}$.	☐	☐
When running Luby's algorithm on a graph with constant degree, every node is in the resulting MIS with constant probability.	☐	☐

H) [3] Consider the following statements.

	true	**false**
There is a randomized distributed algorithm that computes the minimum cut in any graph in at most $n^{7/8}$ rounds, with high probability.	☐	☐
In any n-node weighted graph, we can compute a 2-approximation of the weight of the minimum spanning tree in poly(log n) rounds of the LOCAL model, with high probability.	☐	☐
For any n-node graph, there exists a network decomposition into clusters of strong-diameter $O(\log n)$, colored with $O(\log n)$ colors, so that neighboring clusters have different colors.	☐	☐

Solutions

A) [3] Consider the asynchronous model and the synchronous model.

	true	false
Any problem that can be solved in the synchronous model can be solved in the asynchronous model. *Reason: Assume that every node has a number; find the largest number among all nodes if the number of nodes is unknown.*		✓
Any problem that can be solved in the asynchronous model can be solved in the synchronous model. *Reason: The synchronous model can be considered as a special case of the asynchronous model.*	✓	
If a problem can be solved in both models, the time complexity of the optimal solution in the synchronous model is not worse than the time complexity of the optimal solution in the asynchronous model. *Reason: The synchronous model can be considered as a special case of the asynchronous model.*	✓	

B) [3] Given an unoriented tree with constant degree, in the LOCAL model we can deterministically color it with a constant number of colors in...

	true	false
... $O(\log n)$ rounds. *Reason: See the next point.*	✓	
... $O(\log^* n)$ rounds. *Reason: $\Delta + 1$ coloring of any constant degree graph can be done in $O(\log^* n)$ rounds.*	✓	
... $O(1)$ rounds. *Reason: 3-coloring a directed path, for which we know a $\Omega(\log^* n)$ lower bound, reduces to the problem with only a constant overhead.*		✓

C) [3] Given a cycle of n nodes, for the problem of deterministically computing a set of marked nodes, such that each two marked nodes have distance at least 10, and each unmarked node has a marked node within distance 100, the following bounds hold for the round complexity in the LOCAL model:

	true	false
$O(\log n)$. *Reason: Compute an MIS of G^{10}, which can even be done in $O(\log^* n)$ rounds (e.g. via coloring, note that both G and G^{10} have constant degree)*	✓	
$\Omega(\log n)$. *Reason: As mentioned, the problem can be solved in $O(\log^* n)$ rounds.*		✓
$\Omega(\log^* n)$. *Reason: 3-coloring a directed path, for which we know a $\Omega(\log^* n)$ lower bound, reduces to the problem with only a constant overhead.*	✓	

D) [3] Given a synchronous network with n nodes and diameter D, and messages consist of 1 bit (can communicate with every neighbor in every round). Each node has a $\log n$ bit string. Detecting if all strings are equal takes ...

	true	**false**
(at least) $\Omega(n)$ time. *Reason: $O(\log n)$ is possible in a clique.*		✓
$\Omega(D)$ time. *Reason: In case two nodes have different strings, the propagation of this information takes $\Omega(D)$.*	✓	
$\Omega(D + \log n)$ time. *Reason: If all nodes have the same string except one node, then detecting the difference takes $\Omega(\log n)$ and only after that nodes can be informed about this, which takes $\Omega(D)$.*	✓	

E) [3] The slotted aloha algorithm (Algorithm 13.1) ...

	true	**false**
elects a leader in $O(1)$ time with high probability. *Reason: It only works in $O(1)$ expected time, but not in $O(1)$ time with high probability.*		✓
also works in a uniform setting (i.e. if the nodes do not know n). *Reason: Each node transmits with probability $\frac{1}{n}$, so the algorithm requires a knowledge of n.*		✓
does not require collision detection. *Reason: The algorithm does not use CD.*	✓	

F) [3] Consider the technique of introducing collision detection into a network through a specific leader node (Table 13.7).

	true	**false**
The method enables every node to have collision detection, except for the leader. *Reason: The table shows that the follow-up round indeed lets the listening nodes distinguish between silence and a collision. However, the leader has to transmit in the follow-up round, so it cannot listen; as such, it cannot distinguish between these two cases.*	✓	
When at least 2 nodes submit in the same time slot, the listening nodes can decide whether the leader was one of the nodes that transmitted. *Reason: If at least 2 nodes transmit, then the listening nodes only hear a collision in both time slots. This allows them to deduce that at least 2 nodes have transmitted, but they have no way to know whether the leader was among these nodes.*		✓
The method also works if there are two different nodes that execute the role of the leader simultaneously. *Reason: In this case, nodes will always only hear a collision in the follow-up round, so this does not introduce CD into the network.*		✓

G) [3] Consider the Maximal Independent Set (MIS) problem and Luby's algorithm for solving it.

	true	**false**
In Luby's MIS algorithm, the probability of each node being removed in the first round is at least a positive constant. *Reason: Consider a tree, where we have a node v of degree $\log n$ surrounded by nodes of degree $n^{1/2}$. The probability of v joining the MIS is $\approx 1/\log n$, whereas v is removed by any of its neighbors with probability only $\approx \log n/n^{1/2}$.*		✓
A graph on n vertices with maximum degree Δ contains an independent set of size at least $\frac{n}{\Delta+1}$. *Reason: As seen in the second graded homework, even the first round of Luby's algorithm yields an independent set of this size.*	✓	
When running Luby's algorithm on a graph with constant degree, every node is in the resulting MIS with constant probability. *Reason: Every node is even included in the independent set after the first round, with constant probability.*	✓	

H) [3] Consider the following statements.

	true	**false**
There is a randomized distributed algorithm that computes the minimum cut in any graph in at most $n^{7/8}$ rounds, with high probability. *Reason: Computing even an approximation of the minimum cut requires $\Omega(D)$ rounds, which can be $\Omega(n)$.*		✓
In any n-node weighted graph, we can compute a 2-approximation of the weight of the minimum spanning tree in poly(log n) rounds of the LOCAL model, with high probability. *Reason: This task requires $\Omega(D)$ rounds.*		✓
For any n-node graph, there exists a network decomposition into clusters of strong-diameter $O(\log n)$, colored with $O(\log n)$ colors, so that neighboring clusters have different colors. *Reason: See exercises.*	✓	

2 Partial Coloring (25 points)

For this problem, we use the LOCAL model of distributed algorithms, where the network is abstracted as an n-node undirected graph $G = (V, E)$, where nodes have unique $O(\log n)$-bit identifiers, and per round each node can send an unbounded-size message to each neighbor.

Suppose that the network graph G is an unoriented tree (i.e., G is connected and has no cycle, but there is no notion of parents known to the nodes). Devise a deterministic algorithm that colors at least 90% of the nodes using at most 100 colors, where each node knows its own color. Your algorithm should have the best possible complexity, and any complexity $\Omega(\log n)$ or higher receives zero points.

Solutions

We will show an algorithm that needs $O(\log^* n)$ rounds. The algorithm has 10 phases. In the i-th phase, we first set aside all vertices that are already colored and define G_i as the subgraph formed by the remaining vertices. Moreover, let $H_i \subseteq G_i$ be the set of vertices with degree at most 2 in G_i. The algorithm colors the graph H_i in the i-th phase with three colors $3i, 3i+1, 3i+2$. Note that H_i is disjoint union of paths, so it can be done in $O(\log^* n)$ rounds by Linial's algorithm.

After this algorithm finishes, we have colored the graphs $H_1 \cup H_2 \cup \dots H_{10}$ with $30 \leq 100$ colors, in $O(\log^* n)$ rounds of the LOCAL model. We need to upper bound the number of uncolored nodes. This is done as follows: Each G_i is a forest, so it has at most $|G_i| - 1 \leq |G_i|$ edges. In other words, the sum of degrees of vertices in G_i is bounded by $2|G_i|$. On the other hand, each vertex of $G_i \setminus H_i = G_{i+1}$ has degree at least 3 in G_i. Hence, we have

$$3|G_{i+1}| \leq 2|G_i|$$

This implies that $|G_{11}| \leq \left(\frac{2}{3}\right)^{10} |G_1| \leq 0.1|G_1|$, as needed.

Finally, let us sketch why the achieved round complexity $O(\log^* n)$ is best possible (this was not needed to get points). For contradiction suppose there is an algorithm A that, in $o(\log^* n)$ rounds, outputs coloring with at most 100 colors for 90% of the nodes for any cycle on n nodes. We first construct n such cycles on n nodes, such that identifiers of one cycle are disjoint from identifiers of all the other cycles. On each cycle, there exists a configuration of identifiers such that the algorithm A does not color consecutive path of $\Omega(\log^* n)$ nodes; otherwise, we could color all nodes of that cycle with constant number of colors in $o(\log^* n)$ rounds which would contradict Linial's lower bound. Finally, we can take the path of length $\Omega(\log^* n)$ that A cannot color from each cycle and glue all of these paths in a new cycle. If we run A on this new instance, it will not color $1 - o(1)$ fraction of vertices, which finishes the lower bound.

3 Labeling Elections (29 points)

Let us study labeling schemes for *elections*. There are k candidates in the election. The voters are organized as a tree of n nodes, with $n \geq k$. Each voter/node u in the tree is voting for one candidate, i.e., the vote of each node u is in $\{1, \ldots, k\}$. Given two nodes u and v, we are interested in recovering the winning candidate on the path between u and v. The winning candidate is a candidate receiving most votes from the nodes in the path between u and v (included). If the winning candidate is not unique, but multiple candidates have most votes in the path, the decoder can return any of those candidates.

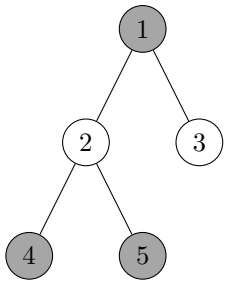

Figure 1: Example of an election on a tree with $k = 2$ candidates. The votes are shown as the colors of the nodes: Gray nodes vote for candidate 1, while white nodes vote for candidate 2. On the path between nodes 4 and 5, candidate 1 receives 2 votes (from 4 and 5) while candidate 2 only receives 1 vote (from 2). Thus, the decoder should return candidate 1 for this request. For a request between 4 and 3, the decoder can return candidate 1 or 2 as they both receive 2 votes.

A) [8] If the tree is a path, give a labeling scheme using at most $\mathcal{O}(k \log n)$ bits for each label.

B) [8] For any tree, give a labeling scheme using at most $\mathcal{O}(k \log^2 n)$ bits for each label.

If k is small we are happy with our solution of **A)**. But if k is large ($k \to n$), the labels of **A)** are too long. We want to improve this.

C) [13] If the tree is a path, give a labeling scheme using at most $\mathcal{O}(\sqrt{n} \log n)$ bits for each label.

Solutions

A) Let us fix an arbitrary endpoint of the path and call it the starting node. For each node u, we store in its label the candidate x_u it is voting for. Also, we store the number of times every candidate c appears between u and the starting node (both nodes included). We need $\log n$ bits for each candidate and $\log k$ bits for u's candidate, so the labels will have $k \log n + \log k \in \mathcal{O}(k \log n)$ bits in total. Given the labels of two nodes, we can recover the number of time each candidate appears on the path between them and deduce the answer from it.

B) We can adapt the Heavy-Light decomposition algorithm seen in the lecture notes (Chapter 14, page 78). We define the candidate of an edge as the candidate voted by the parent of the edge. Each node has the heavy-light labeling, with the following modifications. For each light edge, nodes also store the candidate of the edge, using $\log k$ additional bits. For each heavy path, instead of storing the length of the path, nodes store the number of time each candidates appear on the path using $k \log n$ bits. As there are at most $\log n$ light edges in every path to the root, the label of each node contains at most $k \log n$ bits. Finally, we add to this label the candidate that the node is voting for. Given two labels, we can find the node in the tree where the heavy-light path diverges and recover the sets of colors in between the two nodes. Again, we can deduce the answer out of it.

C) For each candidate having more than $\sqrt{n}$ votes (let us call them *heavy candidates*), we create a counter as in the first question. This takes $\mathcal{O}(\sqrt{n} \log n)$ bits, as there could be as many as $\sqrt{n}$ such candidates.

To deal with the other candidates (let us call them *light candidates*), we will design another labelling scheme. Again, let us fix an arbitrary endpoint of the path as the starting node of the path. For each node u, we define $I_u(x)$ as the index of the first node after u such that some light candidate receives at least x votes between those two nodes. We define this (unique) candidate $C_u(x)$. If there is no such candidate receiving x votes, then we set these two values to -1. We can now create u's label by concatenating $I_u(x)$ and $C_u(x)$ for $x = 1, \ldots, \sqrt{n}$. As no candidate appears more than $\sqrt{n}$ times, we only need to store at most $\sqrt{n}$ pairs. We also store in the label the position of the node in the path, using $\log n$ bits. In total, the labelling of these light candidates also takes $\mathcal{O}(\sqrt{n} \log n)$ bits.

From these two labels, you can recover the heavy and light candidates receiving most of the votes (among their category), and compare them to finally select the one receiving most votes.

4 Disjoint Walks (25 points)

For this problem, we use both the LOCAL and CONGEST model of distributed algorithms, where the network is abstracted as an n-node undirected graph $G = (V, E, w)$, where nodes have unique $O(\log n)$-bit identifiers. In the LOCAL model, per round each node can send an unbounded-size message to each neighbor, whereas in the CONGEST model, per round each node can send one $O(\log n)$-bit message to each neighbor.

Each node $v \in V$ is associated with a walk of length three: v is given three nodes $v_1, v_2, v_3 \in V$, such that (v, v_1, v_2, v_3) is a walk in G, i.e. $\{v, v_1\}, \{v_1, v_2\}, \{v_2, v_3\} \in E$. Note, that the nodes v, v_1, v_2, v_3 are not necessarily distinct.

The task is to design an algorithm that computes a maximal set S of walks, such that there are no two walks (v_0, v_1, v_2, v_3) and (u_0, u_1, u_2, u_3) in S that share an edge, i.e. there are no i, j such that $\{v_i, v_{i+1}\} = \{u_j, u_{j+1}\}$. Such a set S is maximal if we cannot add a walk, without it sharing an edge with a walk already in S. For the output, each node v should know whether or not its walk has been added to S.

A) [13] Devise a deterministic algorithm that solves this problem in poly$(\log n)$ rounds in the LOCAL model.

B) [12] Devise a randomized algorithm that solves this problem with high probability in $O(\log n)$ rounds in the CONGEST model.

Solutions

A) Think of a graph G' on the same vertex set where two vertices u and v share an edge if and only if the respective paths p_u and p_v from the problem statement share an edge. The nodes of G can simulate any algorithm in the LOCAL model on G', with constant slowdown.

We are asked to output a maximal independent set in G'. We have seen in the lecture that this can be done with a deterministic algorithm in $\text{poly}(\log n)$ rounds of the LOCAL model.

B) As in the previous subtask, we will simulate an algorithm that finds a maximal independent set in the graph G'. This time it will be Luby's algorithm. In each round of Luby's algorithm, each node u chooses a random number r_u from $[0, 1]$, and it is put in the independent set if r_u is maximal in the neighborhood of u. We need to show how, in the CONGEST model, each node u can get to know whether r_u is maximal in the neighbourhood of u in the graph G', as this is the independent set S the Luby's algorithm takes away in this round. We will now show how to find a certain independent set S' such that $S \subseteq S'$. Taking away S' instead of S does not hurt the analysis of Luby's algorithm.

The set S' is found by the following algorithm: Each node u sends the number r_u along the path p_u. This means it is sent from $u = u_0$ to u_1, then to u_2 in the next round, and finally to u_3. Whenever some node v wants to send values $r_{u_1}, r_{u_2}, \ldots$ along the same edge, it sends only the largest value. Also, each node v for each edge vw remembers the largest value r_u that was sent through vw in either direction.

After the messages were sent, the surviving ones are sent back along the same path. On the way back, each such message collects the largest number $r_{u'}$ that was sent through the edges of p_u. If this number is not larger than r_u, we put u in S'. This algorithm can be implemented in constant number of CONGEST rounds.

We need to prove 1) that each u, such that r_u is maximal among neighbors of u in G', is in S' and 2) S' is an independent set. To see item 1), note that the message sent from u through p_u and back does not survive only if there is some $p_{u'}$ with larger $r_{u'}$ intersecting p_u. Similarly, if the message survives but is not put to S' in the end, it is for the same reason. To see item 2), consider two vertices u and v with paths p_u and p_v overlapping in edge e; say, $r_u > r_v$. If both messages sent from u and v survive and return back, the vertex v is not put in S', since the maximum number r_w that was sent through some edge of p_v is at least $r_u > r_v$. This finishes the proof.

5 Fault-Tolerant Sorting Networks (24 points)

A) [6] Show that: Given an odd/even sorting network, adding another comparator anywhere does not destroy the sorting property. (Note that this question only gives 6 points, so feel free to re-use arguments from the lecture.)

You can use the statement of question A) to answer the following questions.

B) [12] Given the sorting network of width 4 and 10 comparators in Figure 2. Assume we delete one comparator and consider the remaining network with 9 comparators. For which of the 10 comparators is the remaining sorting network still correct after deletion?

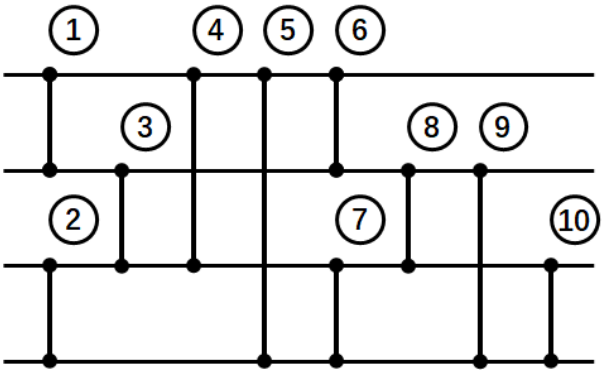

Figure 2: A Sorting Network

C) [6] Design a sorting network with at most 10 comparators, such that no matter which comparator is deleted, the remaining sorting network is still correct.

Solutions

A) Let us assume two sorting networks N and N' have the same input 0-1 sequence, $S = S'$. Before the adding comparator, $S = S'$. We show that that the k^{th} rightmost 1 in S' is more right than or equal to the k^{th} rightmost 1 in S after each comparator. By 0-1 Lemma, N' is also a correct sorting network. We prove this by induction. It is always correct when $k = 1$. Assume it is correct of the $k - 1^{th}$ rightmost 1. The k^{th} rightmost 1 in S' cannot move at comparator j only if the $k - 1^{th}$ rightmost 1 in S' is exactly its neighbor. Because the $k - 1^{th}$ rightmost 1 in S' is more right than or equal to the $k - 1^{th}$ rightmost 1 in S, the k^{th} rightmost 1 in S cannot exceed the one in S'. Therefore, the statement is always correct.

B) The network is correct if any of comparators 1, 2, 3, 7 is removed. Adding any comparator at the front does not destroy the sorting property (Exercise 8.1.c). Comparators 4, 5, 6, 8, 9, 10 constitue a select sort network.

The network is correct if any of comparators 4, 5, 9, 10 is removed. Comparators 1, 2, 3, 6, 7, 8 constitute an entire odd/even sorting network.

The network is correct if comparator 8 is removed. After comparators 4, 5, 6, the minimum number will be located on the first wire and the maximum number will be located on the last wire after comparators 9 and 10. Therefore, we only argue that the output cannot be $O = (0, 1, 0, 1)$. The sequence must be either $(0, 1, 0, 1)$ or $(0, 1, 1, 0)$ after comparator 3, which is impossible.

If comparator 6 is removed, it does not sort $I = (1, 1, 1, 0)$, because $O = (1, 0, 1, 1)$.

C) The example of the sorting network with 6 layers, 9 comparators is shown in Figure 3. Based on A), no matter which comparator is removed, there is a valid odd/even sorting network for four wires.

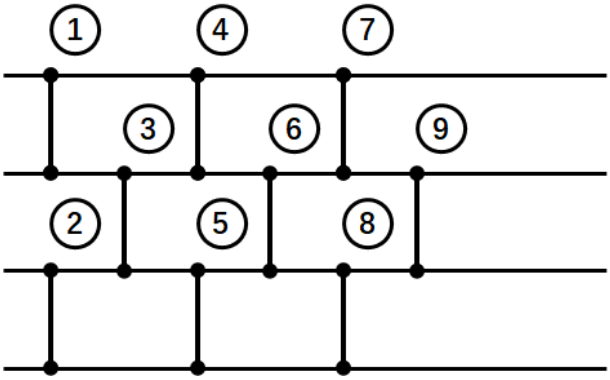

Figure 3: A Sorting Network

6 Shared Graph (28 points)

Alice and Bob share the knowledge of a graph $G = (V, E)$ with n vertices. Alice and Bob both know all the vertices of G, and the unique vertex IDs from 1 to n. Alice knows a subset of the edges $A \subseteq E$ and Bob knows a subset of the edges $B \subseteq E$, such that $A \cup B = E$. A and B need **not** be disjoint, so there might be edges that are known to both Alice and Bob. Alice and Bob want to communicate as little as possible. Help Alice and Bob to:

A) [4] Check whether G is complete (a clique $G = K_n$) using $\mathcal{O}(n^2)$ bits of communication.

B) [8] Give a lower bound for the number of bits of communication required for deciding whether G is complete. (Note that the result of **A)** is tight; only a lower bound proof of $\Omega(n^2)$ receives all points, but weaker lower bound arguments may get some partial points.)

C) [4] Can Alice and Bob improve upon the $\mathcal{O}(n^2)$ communication if they know that A and B are disjoint in advance?

D) [12] Give a lower bound for the number of bits of communication required for deciding whether G is triangle-free (no cliques of 3 nodes exist). (Only a lower bound of $\Omega(n^2)$ will receive all points, but weaker lower bound arguments may get some partial points.)

Solutions

A) Note that Bob can send all of his edges to Alice by encoding them in an array of length $n(n-1)/2$, where each element is a bit denoting whether Bob knows the edge or not (given some common ordering on the potential edges). Alice can then solve the problem, for example by calculating OR. The array contains $\frac{n(n-1)}{2}$ bits, so communication is $\mathcal{O}(n^2)$.

B) For full marks we give a reduction from **DISJ**. Let $X, Y \subseteq \{0, 1, ..., m-1\}$ be an instance of **DISJ**. We construct an instance of shared graph completeness as follows.

- Let $n = \lceil \sqrt{2m} \rceil + 1$. We have n nodes $\{0, 1, ..., n-1\}$ and $n(n-1)/2 \geq m$ potential edges
- Let the potential edges be labeled in lexicographic order, $\{(0,1), (0,2), ..., (0,n-1), (1,2), ...\}$, such that edge $e_i = (\lfloor \frac{i}{n} \rfloor, i \mod n)$
- Alice knows all edges e_i, where $X[i] = 0$
- Bob knows all edges e_i, where $Y[i] = 0$
- Alice and Bob both know all edges e_i for $i \geq m$

X and Y are not disjoint if and only if the graph is not complete. If for some i, $X[i] = Y[i] = 1$, then edge e_i is missing so G is not complete. On the other hand if X and Y are disjoint, i.e. there is no such i, then for all i at least one of Alice and Bob has the edge e_i. The communication complexity of DISJ is $\Omega(m)$. So at least $\Omega(m) = \Omega(n^2)$ bits of communication are required.

C) With A and B disjoint, Alice and Bob can simply count the number of edges in A and B respectively. Alice sends her total to Bob. Bob now checks whether the total number of edges equals $\frac{n(n-1)}{2}$. If yes, then complete, else not complete. Communication complexity is $O(log(n^2)) = O(log(n))$.

D) For full marks we give a reduction from **DISJ**. Let $X, Y \subseteq \{0, 2, ..., m-1\}$ be an instance of **DISJ**. We construct an instance of shared graph triangle-freeness as follows.

- Let $n = \lceil \sqrt{m} \rceil$. We have n nodes $\{a_0, a_1, ..., a_{n-1}\}$ and n nodes $\{b_0, b_1, ..., b_{n-1}\}$ with edges $E_{ab} = \{(a_i, b_i) \mid i = 0, ..., n-1\}$
- We also add n nodes $\{c_0, c_1, ..., c_{n-1}\}$
- We add edges $E_a = \{(a_r, c_q) \mid \forall i \in X\}$, where $r = i \mod n$ and $q = \lfloor \frac{i}{n} \rfloor$
- We add edges $E_b = \{(b_r, c_q) \mid \forall i \in Y\}$, where $r = i \mod n$ and $q = \lfloor \frac{i}{n} \rfloor$
- Alice knows edges $A = E_a \cup E_{ab}$
- Bob knows edges $B = E_b \cup E_{ab}$

See Figure 4 for an illustration.

The graph is triangle-free if and only if E_a and E_b are "correspondingly" disjoint. With "correspondingly" disjoint we mean that there is no r and q such that both $(a_r, c_q) \in E_a$ and $(b_r, c_q) \in E_b$. And E_a and E_b are "correspondingly" disjoint if and only if X and Y are disjoint. The graph has $3n$ nodes, but $m = n^2$ potential edges in E_a and E_b. The communication complexity of DISJ is $\Omega(m)$. So at least $\Omega(m) = \Omega(n^2)$ bits of communication are required.

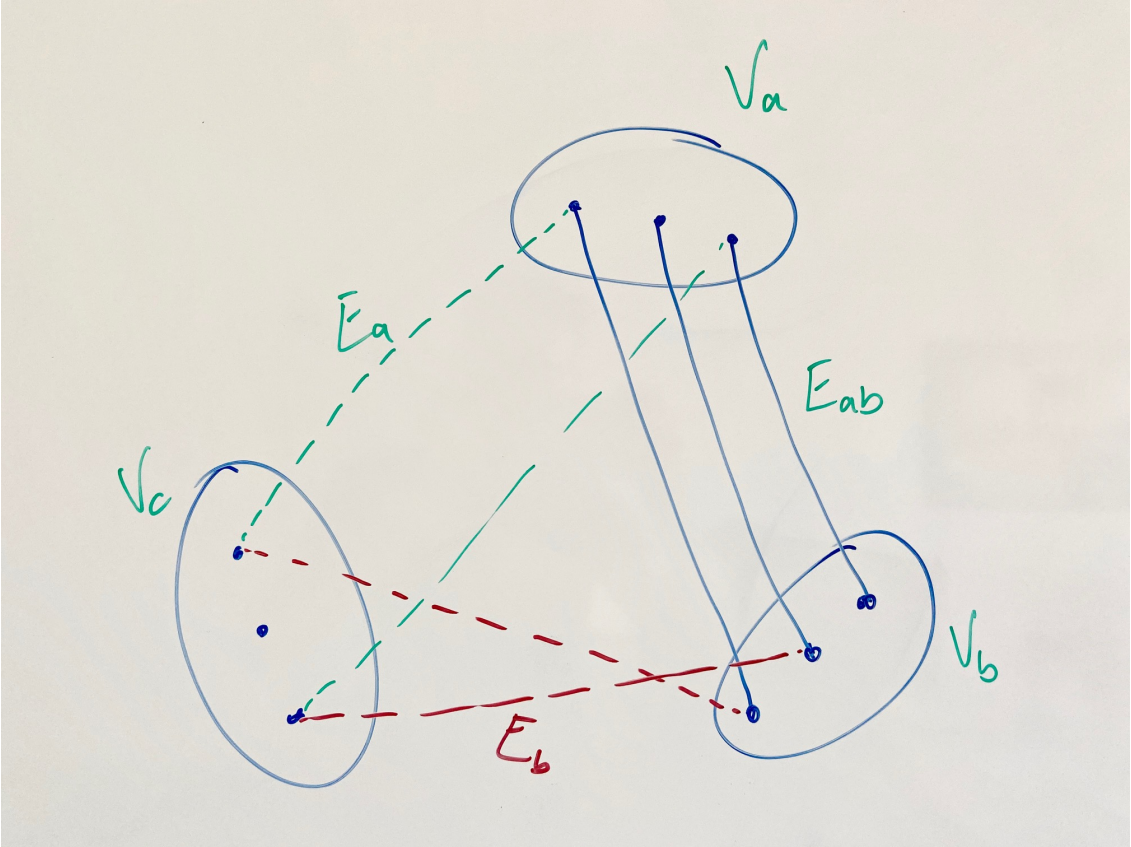

Figure 4: Lower bound construction for communication complexity of shared graph triangle freeness

7 Heaviest Connections in MST (25 points)

For this problem, we use the CONGEST model of distributed algorithms, where the network is abstracted as an n-node undirected graph $G = (V, E, w)$, with diameter D, where nodes have unique $O(\log n)$-bit identifiers, and per round each node can send one $O(\log n)$-bit message to each neighbor. Moreover, assume that each edge $e = \{u, v\}$ has a unique weight $w(e) \in \{1, \ldots, n^{10}\}$, which is known to its endpoint nodes u and v.

Let T be the minimum weight spanning tree of G. The task is to design an algorithm so that, each two neighboring nodes w and w' learn the heaviest weight edge on the unique path in the tree T that connects w and w'. This should happen for all neighboring nodes w and w' simultaneously.

A) [15] Devise a deterministic algorithm that solves this problem in $O(d_T)$ rounds where d_T denotes the diameter of tree T, i.e., the maximum distance in the tree T between any two nodes.

B) [10] Devise a randomized algorithm that solves this problem in $O((D + \sqrt{n}) \cdot \text{poly}(\log n))$ rounds with high probability.

Solutions

A) We solve the problem as follows. First, we find the node of G with the smallest identifier in $O(D) = O(d_T)$ rounds as we have seen in the lecture. This enables us to root the tree in additional $O(d_T)$ rounds. Then, in the next $O(d_T)$ rounds, each node broadcasts its ID and the weight of an edge to its parent to all its children. Although all n nodes start broadcasting this information at once, two broadcasts started by two different vertices u, u' do not interfere, so in $O(d_T)$ rounds each node u knows the weights of the edges on the path from u to the root. Finally, in $O(d_T)$ rounds, for each edge uv in G, u sends the information about the path from u to the root to v and vice versa. The vertex v (and u, respectively) has then enough information to compute the length of the heaviest edge on the path from u to v, as needed.

B) We solve the problem as follows. First, we mark each edge of T with probability $1/\sqrt{n}$. One can see by Chernoff bound that the number of marked edges is $\Theta(\sqrt{n})$, with high probability. We condition on this event being true from now on. Hence, we splitted the tree in $\Theta(\sqrt{n})$ subtrees. Moreover, there are $\binom{n}{2} = O(n^2)$ paths in the tree. For each such path, it is the case by Chernoff bound that if its length is at least a sufficiently high constant multiple of $\sqrt{n}$, at least one of its edges was taken with high probability. We condition on all these events being true from now on. In other words, with high probability, we split T in $\Theta(\sqrt{n})$ connected components, each one with diameter $O(\sqrt{n})$. We call these components *fragments* from now on.

We first compute the answer for each e such that both endpoints are in the same fragment. This is done by rooting each fragment and simulating the algorithm from the previous subsection for each fragment in parallel.

For edges with endpoints in different fragments, the algorithm is somewhat more involved. First, for each fragment F_i we define its skeleton $S_i \subseteq F_i$ as its subtree that we get as union of all paths in F_i that connect two vertices $u, v \in F_i$ such that both u and v are incident with some marked edge. In other words, the skeleton S_i contains only the edges of F_i that are important to compute the answer for edges, whose both endpoints do not lie in F_i. Moreover, we do not need to know the weight of all edges of F_i: for each two vertices $u, v \in F_i$ with degree strictly bigger than 2 that are connected by a path such that all vertices on that path between u and v have degree 2, we compress this path into a single edge of weight equal to the maximum weight on the previous path. With that, we get a new graph S_i' whose size is proportional to the number of marked edges incident to F_i. The total size of all these graphs S_i' is hence $O(\sqrt{n})$; they can be computed and made public in $\tilde{O}(\sqrt{n} + D)$ rounds by routing from the lecture.

Going back to the original problem, to compute the answer for each edge $e = uv$ with endpoints u, v in different fragments, similarly to the previous subtask, the vertices u and v exchange the information about the path from u (or v, respectively) to the root of the fragment of u (or v, respectively). This information is computed for each node in $O(\sqrt{n})$ rounds in parallel and exchanged in $O(\sqrt{n})$ rounds. Together with the public information about the compressed skeletons of all fragments, this is enough information for each endpoint of e to deduce the maximum weight on the path between u and v on T, as needed.