COS 330: Great Ideas in Theoretical Computer Science

Fall 2025

Precept 9

My many selves

Learn

In lecture, we studied one way to model distributed computing. Let's review it and then see another example.

We think of a distributed system as an undirected graph $G = (V, E)$ (with $n = |V|$):

- Each node $u \in V$ is a processor running the *same* algorithm.
- Edges represent bidirectional communication links between processors.

In the 3-coloring example from lecture, the graph we considered was a path graph (or a cycle). In the fault-tolerant computing example, the graph was a complete graph.

Time proceeds in synchronous rounds $t = 0, 1, 2, \ldots$. In each round, every node u:

1. sends (possibly different) messages to each neighbor,
2. receives all messages sent to it in that round,
3. updates its local state according to some deterministic or randomized transition rule.

We usually assume:

- Each node has a unique identifier (UID), e.g. an integer in $\{1, \ldots, n^C\}$ for some constant C.
- Messages are $O(\log n)$ bits long (enough to send a UID or a constant number of them). We didn't emphasize this constraint much in lecture, but it's important in more advanced settings that we will not cover here.

Our metric of efficiency is the number of rounds until all nodes have produced their outputs - we assume that computation happens synchronously, so the number of rounds is the main bottleneck.

The MIS Problem on a Path

Let's see another short example. An *independent set* $I \subseteq V$ is a set of vertices with no edges between them. It is *maximal* if no strict superset of I is also independent (i.e., you can't add any more vertices without breaking independence).

We'll work with a path graph P_n in the distributed model, which means:

- The network is a path graph with n nodes.
- Each node u has a unique identifier $\mathrm{ID}(u)$.
- There is no global knowledge of n.

Our goal is for each node u to output a bit $\mathrm{MIS}(u) \in \{0, 1\}$ such that

$$I := \{u : \mathrm{MIS}(u) = 1\}$$

is a maximal independent set of the path.

High-level idea. We'll use a greedy approach: let "locally best" nodes (those with the highest IDs among their neighbors) join the MIS, then force their neighbors to stay out. We repeat this process locally until every node has decided.

We assign to each node one of three states that will evolve throughout the algorithm: active (node has not yet decided whether it is in the MIS), inMIS (node has decided to join the MIS), or outMIS (node has decided to stay out of the MIS), and update these states in synchronous rounds. In other words, each node u stores:

$$\text{state}(u) \in \{\text{active}, \text{inMIS}, \text{outMIS}\}.$$

Additionally, each node u knows its own ID $\mathrm{ID}(u)$.

Initially, all nodes are active. Messages are very simple: an active node sends its ID and state to its neighbors. In each synchronous round, every node u does:

1. Send $(\mathrm{ID}(u), \text{state}(u))$ to all neighbors and receive their current states and IDs.
2. If $\text{state}(u) = \text{active}$:
    - If u has an active neighbor that just became inMIS in this round, set $\text{state}(u) :=$ outMIS.
    - Else, if $\mathrm{ID}(u)$ is strictly larger than the ID of every *active* neighbor, then set $\text{state}(u) :=$ inMIS.
    - Otherwise, remain active.
3. If $\text{state}(u) \in \{\text{inMIS}, \text{outMIS}\}$, u keeps that state forever.

Nodes can stop as soon as they know that they are no longer active and that their neighbors' states will not change (e.g., after seeing that all neighbors are no longer active for one round), and they output $\mathrm{MIS}(u) = 1$ if $\text{state}(u) = \text{inMIS}$ and $\mathrm{MIS}(u) = 0$ otherwise.

Let's now prove that this algorithm always produces a valid MIS. To do so, we'll first show that the output set I is indeed an independent set, and then that it is maximal.

Independence. We need to show that no two adjacent nodes both join the MIS. Suppose for contradiction that two adjacent nodes u and v both end in state inMIS. Consider the first round when one of them (say u) joined the MIS. At that moment, v must have been active (since it later joins too). But u can only join if its ID is strictly larger than all active neighbors, including v. This means in the same round, v sees a neighbor join the MIS and must become outMIS. Once it becomes outMIS, v can never join the MIS, giving us a contradiction. Therefore, I is independent.

Maximality. We also need to show that we can't add any more nodes to I without breaking independence. Consider any node w that's not in the MIS at the end. It must be in state outMIS. By the algorithm's rules, a node only becomes outMIS if it has a neighbor that joined the MIS. So every node outside I has a neighbor in I, which means we can't add it to I without creating an edge between two nodes in the set. Therefore, I is maximal.

How many rounds does this algorithm take? Consider a path where the IDs increase from left to right: $1 - 2 - 3 - \cdots - n$. In the first round, only node n (the rightmost) joins the MIS. In the second round, node $n-1$ becomes inactive. Then node $n-2$ might join in round 2, node $n-3$ becomes inactive in round 3, and so on. The decision "propagates" along the path one edge at a time, taking $O(n)$ rounds total. So our algorithm has worst-case running time $O(n)$ on a path of length n.

It's possible to improve the above algorithm to run in $O(\log n)$ rounds on a path, by picking UIDs randomly, but analyzing that requires more advanced techniques, which we won't cover here.

Practice

Problem 1

Let's consider the same model of distributed computing as above, and study the coloring problem again, but now on different kinds of graphs. Assume:

- The network graph $G = (V, E)$ is a *tree* (connected and acyclic).
- Exactly one node is designated as the *root*: it has an input bit $\mathsf{root} = 1$, and all other nodes have $\mathsf{root} = 0$.

The goal is for every node u to output a color $\mathsf{col}(u) \in \{0, 1\}$ so that adjacent nodes always have different colors (a proper 2-coloring of the tree). We'll do that in two steps.

(a) Design a simple distributed algorithm that lets every node u learn its distance $\mathsf{dist}(u)$ from the root (the number of edges on the unique path from u to the root), and give a bound on the number of rounds in terms of the tree's diameter D[1].

You do not need to be very formal; a clear high-level description is enough.

---

[1] The *diameter* of a tree is the maximum distance between any two nodes in the tree.

(b) Suppose now that each node knows its distance $\mathsf{dist}(u)$ from the root.

Give a very simple local rule to choose a color $\mathsf{col}(u) \in \{0, 1\}$ from $\mathsf{dist}(u)$, and prove that your rule produces a proper 2-coloring of the tree. How many rounds does this 2-coloring algorithm take in total?