Distributed Algorithms

Abdul Rehman, Momin Ahmad Rizvi, Tse Yik Long

February 2026

Contents

This document provides, or attempts to provide, solutions to some of the problems from "Distributed Algorithms 2020" by Jukka Suomela (https://jukkasuomela.fi/). We aim to solve at least 3 questions from each chapter, covering the entire book.

1 Introduction

1.1 Exercise 1.2

Rewrite the greedy algorithm of Table 1.1 so that stopped nodes do not need to send messages. Be precise: explain your algorithm in detail so that you could easily implement it.

Solution

Consider the following updated version of the greedy 3-coloring algorithm:

Algorithm 1: Updated Greedy 3-Coloring Algorithm

Data: Vertices V with unique identifiers
Result: Valid 3-coloring of the path
Set $N \leftarrow \emptyset$ Repeat forever: Send message c to all neighbors;
Receive messages from all neighbors and store in M;
if $c \notin \{1, 2, 3\}$ *and* $c > \max M$ **then**
    Let $c \leftarrow \min((\{1, 2, 3\} \setminus M) \setminus N)$;
    Send c to all neighbors;
    terminate;
end
Send *null* to all neighbors;
Receive messages from all neighbors and store in M;
$N \leftarrow N \cup M$;

In this version, once a node determines its color it terminates and does not send any more messages. The main difference is that we maintain a set N of colors that have been assigned to neighbors, and we only consider colors that are not in N when determining our own color. This way, once a node has assigned itself a color, it will not interfere with the coloring decisions of its neighbors, and it can safely terminate without sending further messages.

1.2 Exercise 1.3

The fast 3-coloring algorithm from Section 1.4 finds a 3-coloring very fast in any directed path. Design an algorithm that is almost as fast and works in any path, even if the edges are not directed. You can assume that the range of identifiers is known.

Solution

We will first convert the undirected path into directed subgraphs. Then, we will run the fast 3-coloring algorithm on each directed subgraphs. Finally, we will combine the subgraphs to get a valid 3-coloring.

Algorithm 2: Generalized Fast 3-Coloring Algorithm

Data: Vertices V with unique identifiers
Result: Valid 3-coloring of the undirected path
For each node v, classify it either as local maxima, local minima, or intermediary based on its neighbors' identifiers;
Orient edges from lower to higher identifiers to create directed subgraphs;
Treating local maxima and minima as endpoints, run the fast 3-coloring algorithm on each directed subgraph;
For each local maxima, if its color conflicts with its neighbor (which would be a local minima), assign the highest color that does not conflict;
return *The assigned colors*

Correctness: Each subgraph is colored correctly by correctness of the fast 3-coloring algorithm. Local maxima and minima are handled to ensure no conflicts arise at their connections. Thus, the entire path is properly 3-colored.

Time Complexity: The classification step takes $O(1)$ time. The fast 3-coloring algorithm runs in $O(\log^* n)$ time on each directed subgraph. The final adjustment step also takes $O(1)$ time. Therefore, the overall time complexity is $O(\log^* n)$ - that is, it is dominated by the fast 3-coloring algorithm.

1.3 Exercise 1.4

The simple randomized 3-coloring algorithm finds a 3-coloring in time $O(\log n)$ with high probability, and it does not need any unique identifiers. Can you design a randomized algorithm that finds a 3-coloring in time $o(\log n)$ with high probability? You can assume that n is known.

Solution

The algorithm is simple: in each super-round, each node (independently) selects a random identifier from $[2^n]$ and run the algorithm in Section 1.4. Each node then checks with its neighbors to see if there are any conflicts. If there are no conflicts, the algorithm halts. Otherwise, the node repeats the process in the next super-round.

The correctness of the algorithm is trivial. To analyze the runtime, observe that

$$p = \mathbb{P}[\text{no identifier collision}] = \frac{\binom{m}{n} \cdot n!}{m^n} = \frac{m!/(m-n)!}{m^n} \geq \frac{(m-n)^n}{m^n} = \left(1 - \frac{n}{m}\right)^n \tag{1}$$

where m is the size of the pool of identifiers. As we have chosen m to be 2^n, we have

$$p \geq \left(1 - \frac{n}{2^n}\right)^n. \tag{2}$$

By Bernoulli's inequality,

$$1 - \frac{n^2}{2^n} \leq \left(1 - \frac{n}{2^n}\right)^n \leq 1. \tag{3}$$

By the Squeeze theorem, we know that $p \to 1$ as $n \to +\infty$. If all identifiers are distinct in a super-round, the number of rounds in the super-round is constant. We have thus shown that the algorithm halts in $O(1)$ time with high probability, which is $o(\log n)$.

2 Graph Theoretical Foundations

2.1 Exercise 2.6 (isomorphism)

Construct non-empty 3-regular connected graphs G and H such that G and H have the same number of nodes and G and H are not isomorphic. Just giving a construction is not sufficient – you have to prove that G and H are not isomorphic.

Solution

Consider the following two graphs G and H:

- G: Let G be the complete bipartite graph with bipartition u_1, u_2, u_3 and v_1, v_2, v_3. It is easy to verify that G is 3-regular as each vertex is adjacent to exactly 3 vertices in the other partition.
- H: Consider the graph obtained by taking a triangle (3-cycle) and attaching a pendant edge to each vertex of the triangle. Formally, let H have vertices $\{a, b, c\}$ forming a triangle, and vertices $\{d, e, f\}$ such that d is adjacent to a, e is adjacent to b, and f is adjacent to c. The three vertices d, e, f are themselves connected to form a triangle. It is easy to verify that H is also 3-regular as each vertex has degree 3.

The graphs G and H are not isomorphic. To see this, observe that G is bipartite while H is not. In G, we can partition the vertices into two sets such that no edges exist within the same set, while in H, any attempt to partition the vertices will fail due to the presence of the triangle formed by a, b, c and the triangle formed by d, e, f. This is because the triangle structure in H creates odd cycles, which cannot exist in a bipartite graph. Since isomorphisms preserve the property of being bipartite, G and H cannot be isomorphic.

2.2 Exercise 2.7 (matchings and edge domination)

Show that

(a) a maximal matching is a minimal edge dominating set,

(b) a minimal edge dominating set is not necessarily a maximal matching,

(c) a minimum maximal matching is a minimum edge dominating set,

(d) any maximal matching is a 2-approximation of a minimum edge dominating set.

Solution (a)

Let $G = (V, E)$ be a graph and M be a maximal matching in G. We first show that M is an edge dominating set. Assuming otherwise, there exists an edge $e = \{u, v\}$ that is not dominated by M. This means that e is not in M, and no edge in M is incident to either u or v. However, $M + e$ is a matching, contradicting the maximality of M.

Next, we suppose we could remove an edge $e = \{u, v\}$ from M and still have an edge dominating set. Without loss of generality, assume e is incident to $v \in G[M]$ and $\{v, w\} \in M$. However, this would mean that both e and $\{v, w\}$ are incident to v, contradicting the fact that M is a matching. We have thus shown that M is also minimal.

Solution (b)

Counterexample: Consider the graph G with vertices $V = \{a, b, c, d\}$ and edges $E = \{\{a, b\}, \{b, c\}, \{c, d\}, \{d, a\}\}$. Visually,

$$\begin{array}{ccc} a & - & b \\ | & & | \\ d & - & c \end{array} \tag{4}$$

$M = \{a, b\}, \{b, c\}$ is a minimal edge dominating set but is obviously not a matching.

Solution (c)

Let $G = (V, E)$ be a graph and D be a minimum edge dominating set in G. We will show a procedure which produces a maximal matching M such that $|M| = |D|$.

Let $M_0 = D$. For each $i \geq 1$, we check if there are $u, v, w \in V$ such that $\{u, v\} \in M_{i-1}$ and $\{v, w\} \in M_{i-1}$. If not, we stop and output M_{i-1}. Otherwise, we search for another edge $e = \{w, x\}$ such that e is not adjacent to any edge in $M_{i-1} - \{u, v\}$, which we use to replace $\{v, w\}$ to produce M_i.

We first show that M_i remains a minimum edge dominating set after the replacement. It is obvious that $|M_i| = |D|$ for all i. We only need to show that M_i dominates E. The case when $i = 0$ is given. For $i > 0$, assume otherwise. Note that M_i is formed by replacing $\{v, w\}$ of M_{i-1} with $e = \{w, x\}$. In other words, $V(M_{i-1}) \subseteq V(M_i)$. Therefore, M_{i-1} domainting E implies that M_i also dominates E. We have thus shown that M_i is a minimum edge dominating set for all i.

Next, we show that the search for e always succeeds. Assume otherwise. This would mean that every edge incident to w is adjacent to some edge in M_{i-1}. However, by construction, all edges incident to w are covered by $M_{i-1} - \{v, w\}$, so e is redundant and M_{i-1} is not minimal, which contradicts the above. As such, the search for e always succeeds.

We also need to show the procedure halts. Every time we replace an edge, say $\{v, w\}$, with another edge $e = \{w, x\}$ at iteration i, $\{v, w\}$ can never be in M_j for $j \geq i$. This is because the procedure never adds edges adjacent to M_j, and e is not adjacent to any edge in M_i. Since there are only finitely many edges, the procedure halts after finitely many iterations.

Finally, we show that the output is a maximal matching. Assume otherwise. Then there exists an edge $e = \{u, v\}$ such that $M + e$ is a matching. Since M is an edge dominating set, there exists an edge $f = \{v, w\} \in M$ that dominates e. However, this would mean that both e and f are incident to v, contradicting the fact that $M + e$ is a matching. We have thus shown that the output is a maximal matching.

Solution (d)

Let $G = (V, E)$ be a graph and M be a matching of G. Observe that the complement of M is also a matching, so $|M| \leq \frac{|E|}{2}$. If M is a maximal matching, it is also an edge dominating set. Therefore, $|M| \leq \frac{|E|}{2} \leq 2 \cdot \text{OPT}$, where OPT is the size of a minimum edge dominating set. We have thus shown that any maximal matching is a 2-approximation of a minimum edge dominating set.

2.3 Exercise 2.8 (Petersen 1891)

Show that any $2d$-regular graph $G = (V, E)$ has an orientation $H = (V, E')$ such that

$$\text{indegree}_H(v) = \text{outdegree}_H(v) = d$$

for all $v \in V$. Show that any $2d$-regular graph has a 2 -factorization.

Solution

We proceed by induction on d.
Base Case: For $d = 1$, a 2-regular graph is a collection of cycles. We can orient each cycle in one direction, ensuring that each vertex has an indegree and outdegree of 1.
Inductive Step: Assume the statement holds for some $d = k$. We need to show it holds for $d = k + 1$. Given a $2(k + 1)$-regular graph G, we can find a 2-factor (a collection of cycles covering all vertices). Remove this 2-factor from G, resulting in a $2k$-regular graph G'. By the inductive hypothesis, G' has an orientation where each vertex has indegree and outdegree of k. Now, we can orient the edges of the removed 2-factor such that each vertex's indegree and outdegree increase by 1. This gives us the desired orientation for G.
Thus, by induction, any $2d$-regular graph has the required orientation.
To show that any $2d$-regular graph has a 2-factorization, we can repeatedly find and remove 2-factors from the graph until no edges remain. Each removal reduces the degree of each vertex by 2, and since the graph is $2d$-regular, we can perform this process d times, resulting in a 2-factorization of the graph.

3 Port Numbering Model

3.1 Exercise 3.4 (More than two colors)

Design a distributed algorithm that finds a maximal matching in k-colored graphs. You can assume that k is a known constant.

Solution

We generalize the algorithm for 2-colored graphs to work for k-colored graphs. The idea is to have k phases in each round instead of 2. In each phase i, the nodes colored i will recieve proposals from their neighbors and respond with acceptance (taking the minimum color and port number).

Algorithm 3: Generalized Maximal Matching Algorithm for k-colored Graphs

```
Data: Graph G = (V, E) with k-coloring
Result: Maximal matching in G
Initialize matched pairs M = ∅;
Intialize all nodes as UR;
Intialize counter c = 0;
while there exists an UR node do
    c = c + 1;
    for phase i from 1 to k − 1 do
        for each UR node u with color not i do
            Send proposal to all neighbors ;
        end
        for each UR node v with color i do
            if X(v) ≠ ∅ then
                M(v) ← min_{(u,p)∈X(v)}(color(u), p);
                Change state to MR;
            end
            if c > deg(v) then
                Change state to US;
            end
        end
    end
    Commence next round;
    for each MR node v do
        Send acceptance to M(v);
        Change state to MS;
    end
    for each UR node u with color not i do
        if received acceptance from neighbor v then
            Add (u, v) to M;
            Change state to MS;
        end
    end
end
return Matched pairs M
```

Analysis: Analogous to the 2-colored case, the algorithm halts in $O(kd)$ where d is maximum degree since in each $2(k-1)$ rounds, either a proposal is accepted, or rejected.
Assume that the output is not a maximal matching. Then there exists an unmatched edge (u, v) where both u and v are unmatched. However, since both u and v are unmatched, they must have proposed to each other at some point, leading to a contradiction.

3.2 Exercise 3.5 (analysis of vertex cover approximation)

Is the analysis of the minimum vertex cover 3-approximation algorithm tight? That is, is it possible to construct a network N such that the algorithm outputs a vertex cover that is exactly 3 times as large as the minimum vertex cover of the underlying graph of N?

Solution

Let the physical network N be the graph P_3 with vertices $V = \{a, b, c\}$ and edges $E = \{\{a, b\}, \{b, c\}\}$. Assign port numbers as follows: $p(a, 1) = b$, $p(b, 1) = a$, $p(b, 2) = c$, $p(c, 1) = b$. Next we construct the virtual network N' as described in the the chapter. The vertices of N' are $a_1, a_2, b_1, b_2, c_1, c_2$ where a_1, b_1, c_1 are white and a_2, b_2, c_2 are black. The edges of N' are a_1b_2, a_2b_1, b_1c_2, and b_2c_1.
The bipartite maximal matching algorithm can be made to output the matching $M' = \{a_1b_2, b_1c_2\}$. With the given port numbers and a deterministic tie-breaking rule (e.g., accepting the proposal from the smallest port), this matching is indeed produced: in round 1, a_1 proposes to b_2 (port 1), b_1 proposes to a_2 (port 1) and c_2 (port 2); b_2 accepts a_1; a_2 and c_2 each accept b_1. After round 1, a_1, b_2, b_1, c_2 are

matched. No unmatched white node remains, so the algorithm stops.
By the algorithm, a physical vertex v outputs 1 iff at least one of its copies v_1, v_2 is matched in M'. Here a_1, b_1, b_2, and c_2 are matched, so a, b, and c all output 1. Hence the algorithm returns $C = \{a, b, c\}$, of size 3.
The graph P_3 has a minimum vertex cover of size 1 (e.g., $\{b\}$). Therefore, $|C| = 3 = 3 \cdot 1 = 3 \cdot \tau(G)$, where $\tau(G)$ is the size of a minimum vertex cover. The ratio 3 is attained, so the analysis is tight.

3.3 Exercise 3.7 (composition)

Assume that algorithm A_1 solves problem Π_1 on family $\mathscr{F}$ given Π_0 in time T_1, and algorithm A_2 solves problem Π_2 on family $\mathscr{F}$ given Π_1 in time T_2. Is it always possible to design an algorithm A that solves problem Π_2 on family $\mathscr{F}$ given Π_0 in time $O(T_1 + T_2)$?

Solution

Yes. The basic idea is simple: we first run A_1 to solve Π_1 given Π_0, and then we run A_2 to solve Π_2 given Π_1. The problem arises when T_1 is not constant, since each node in the network may not know when A_1 has finished and A_2 can start. We can solve this problem by having each node switch to the next problem immediately after it finishes its part of A_1. Each node, no matter in A_1 or A_2, keeps track of all messages belonging to A_2 and replays one round in A_2 once it finishes its part in A_1 and the set of received messages is complete. We describe the algorithm in more detail below.

Let $N = (V, P, p)$ be the port-numbered network. Without loss of generality, we will assume that A_1 and A_2 use disjoint states and messages, and that $\text{Output}_{A_1} = \{\varnothing\}$. (This is possible per Exercise 3.3.) We define the state machine and transition function for each node as follows:

$$\text{States}_A = (\text{States}_{A_1} \cup \text{States}_{A_2}) \times \mathbb{Z}_{\geq 0} \times (\text{Msg}_{A_2} \cup \{\varnothing\})^{d \times \mathbb{N}}, \tag{5}$$

$$\text{Output}_A = \text{Output}_{A_1} \cup \text{Output}_{A_2}, \tag{6}$$

$$\text{Msg}_\text{A} = (\text{Msg}_{A_1} \cup \text{Msg}_{A_2}) \times \mathbb{Z}_{\geq 0}, \tag{7}$$

$$\text{init}_{A,d}(f) = (\text{init}_{A_1,d}(f), 0, \{\varnothing\}^{d \times \mathbb{N}}), \tag{8}$$

$$\text{send}_{A,d}((x, r, M)) = \begin{cases} (\text{send}_{A_1,d}(x), r) & \text{if } x \in \text{States}_{A_1}, \\ (\text{send}_{A_2,d}(x), r) & \text{if } x \in \text{States}_{A_2}, \end{cases} \tag{9}$$

$$\text{receive}_{A,d}((x, r, M), (m_i)_{i \in [d]}) = \begin{cases} (\text{receive}_{A_1,d}(x, \overline{m}), 0, M|m) & \text{if } x \in \text{States}_{A_1} \setminus \text{Output}_{A_1}, \\ (\text{init}_{A_2,d}(x), 1, M|m) & \text{if } x \in \text{Output}_{A_1}, \\ (x, r, M|m) & \text{if } x \in \text{States}_{A_2} \wedge \varnothing \in (M|m)[r], \\ (\text{receive}_{A_2,d}(x, (M|m[r])), r + 1) & \text{otherwise}, \end{cases} \tag{10}$$

where

$$\overline{m} = \begin{cases} m & \text{if } m \in \text{Msg}_{A_1} \\ \varnothing & \text{otherwise} \end{cases}, \; M|m = \left(\begin{cases} m[i][1] & \text{if } m[i][2] = r \\ M[r][i] & \text{otherwise} \end{cases} \right)_{i \in [d], r \in \mathbb{N}}. \tag{11}$$

The correctness of A is clear. We now show that its runtime is $O(T_1 + T_2)$. By construction, after $T_1 + 1$ rounds, all nodes are in States_{A_2}. Therefore, in the next round, the message history stored in every node is complete for $r = 1$, and thus all nodes will begin A_2.

4 LOCALModel: Unique identifiers

4.1 Exercise 4.1 (Applications)

Let Δ be a known constant, and let $\mathscr{F}$ be the family of graphs of maximum degree at most Δ. Design fast distributed algorithms that solve the following problems on in the LOCAL model.
(a) Maximal independent set.
(b) Maximal matching.
(c) Edge coloring with $O(\Delta)$ colors.
You can assume that all nodes get the value of n (the number of nodes) as input; also the parameter c in the identifier space is a known constant, so all nodes know the range of unique identifiers.

Solution

Maximal Independent Set

We keep on selecting the local maximum - refer to exercise 6.7 for analysis and correctness. Remark: Another idea is to use the fast Δ+1-coloring algorithm - however, that increases the runtime unnecessarily (refer to exercise 6.6).

Maximal Matching

We first run the fast $\Delta + 1$-coloring algorithm to get a proper coloring of the graph. Then in $\Delta + 1$ phases, for each color, we arbritrarily select a neighbouring node to match.
Correctness is clear from the brute force nature of the algorithm.
The fast coloring algorithm runs in $O(\Delta+\log^* n)$ time. Each phase can be implement in 2 communication rounds, hence the total runtime is $O(\Delta + \log^* n)$.

Edge Coloring with $O(\Delta)$ colors

We first form a new graph G' where each edge in G corresponds to a vertex in G', and two vertices in G' are adjacent if their corresponding edges in G share an endpoint. Note that the maximum degree of G' is at most $2\Delta - 2$ since each edge in G can be adjacent to at most $2\Delta - 2$ other edges (each endpoint can have at most $\Delta - 1$ other edges).
Then, we run the fast $\Delta+1$-coloring algorithm on G'. Since the maximum degree of G' is at most $2\Delta-2$, we can color G' with at most $2\Delta - 1$ colors, which is $O(\Delta)$.
Correctness follows from the fact that a proper vertex coloring of G' corresponds to a proper edge coloring of G. The runtime is $O(\Delta + \log^* n)$ due to the coloring step. We have $|E| \in O(n^2)$ vertices in G' and maximum degree at most $2\Delta - 2$, so the coloring algorithm runs in $O(\Delta + \log^* n)$ time.

4.2 Exercise 4.2 (vertex cover)

Let $\mathscr{F}$ consist of cycle graphs. We design a fast distributed algorithm that finds a 1.1-approximation of a minimum vertex cover on $\mathscr{F}$ in the LOCAL model.

Solution

We can use the following algorithm to achieve a 1.1-approximation for the minimum vertex cover on cycle graphs. The idea is to break the cycle into short paths and solve each path optimally.

Algorithm 4: 1.1-Approximation for Minimum Vertex Cover on Cycle Graphs

```
Data: Cycle graph C_n
Result: A vertex cover of size at most 1.1 times the minimum vertex cover
Fix an integer k ≥ 11;
for each vertex v do
    Gather information about all vertices within distance k along the cycle;
    Identify the vertex with the smallest identifier among these 2k + 1 vertices;
    if v is the vertex with the smallest identifier then
        Mark v as a cut vertex;
    end
end
for each vertex v do
    Learn the structure of the path it belongs to (between two cut vertices);
    Compute the optimal vertex cover for that path;
    Output 1 if v is selected in the cover, otherwise output 0;
end
```

Correctness and approximation: The union of the covers of the paths covers all edges, because every edge lies on exactly one path (cut vertices are not adjacent to each other, so the paths are edge-disjoint and cover the whole cycle). For a single path of length ℓ (number of edges), its minimum cover size is $\lceil \ell/2 \rceil$. Our local solution on that path gives exactly that optimum. Thus the total size of our cover is $\sum_i \lceil \ell_i/2 \rceil$ where ℓ_i are the edge counts of the paths. The optimum for the whole cycle, OPT,

satisfies

$$\text{OPT} \geq \sum_i \lceil \ell_i/2 \rceil - c$$

where the loss c is at most the number of cut vertices (each cut vertex may be counted in two adjacent paths, but we can bound the error). For a cycle, a standard averaging argument shows that the total cover produced is at most $(1 + 1/(k-1))\,\text{OPT}$. With $k \geq 11$, we have $1 + 1/(k-1) \leq 1.1$.

4.3 Exercise 4.4 (distance-2 coloring)

Let $G = (V, E)$ be a graph. A distance-2 coloring with k colors is a function $f : V \to \{1, 2, ..., k\}$ with the following property:

$$\text{dist}_G(u, v) \leq 2 \implies f(u) \neq f(v) \text{ for all nodes } u \neq v.$$

Let Δ be a known constant, and let F be the family of graphs of maximum degree at most Δ. Design a fast distributed algorithm that finds a distance-2 coloring with $O(\Delta^2)$ colors for any graph $G \in F$ in the LOCAL model.

You can assume that all nodes get the value of n (the number of nodes) as input; also the parameter c in the identifier space is a known constant, so all nodes know the range of unique identifiers.

Solution

By performing 2 iterations of the gathering algorithm, each node can learn the structure of its distance-2 neighborhood. Maintain a round counter in each node, and attach a "destination header" to each message indicating the intended recipient. In every even round, each node only distributes the messages it received to the intended recipients in its distance-1 neighborhood. Through the above procedures, we can run the coloring algorithm described in Theorem 4.8 on the square of G, $G^2 = (V, E^2)$, defined as

$$E^2 = \{\{u, v\} \mid \text{dist}_G(u, v) \leq 2, u \neq v\}, \tag{12}$$

in a factor of 2 of the original runtime. The correctness of the algorithm is clear since the degree of G^2 is at most Δ^2.

5 CONGEST Model: Bandwith Limitations

5.1 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 $O(diam(G))$.

Solution

We use the leader election algorithm to elect a leader node in $O(diam(G))$ time, followed by a breadth-first tree construction rooted at the leader (also $O(diam(G))$ time). Then, every node sends the number of edges it is incident to up the tree, and the leader sums these values and divides by 2 to get the total number of edges. Finally, the leader broadcasts the result back down the tree.

Algorithm 5: Edge Counting in CONGEST Model

Data: Connected graph $G = (V, E)$
Result: All nodes output $|E|$
Run leader election algorithm to elect a leader node ℓ;
Construct a BFS tree rooted at ℓ using breadth-first traversal;
for *each node v in post-order* **do**
  if *v is a leaf* **then**
    $\text{sum}(v) \leftarrow \deg(v)$;
  end
  else
    $\text{sum}(v) \leftarrow \deg(v) + \sum_{u \in \text{children}(v)} \text{sum}(u)$;
  end
end
Leader ℓ computes $|E| \leftarrow \text{sum}(\ell)/2$;
Broadcast $|E|$ down the tree to all nodes;
return *All nodes output $|E|$*

Correctness: Each edge is counted exactly twice (once from each endpoint), so the sum at the leader will be $2|E|$. Dividing by 2 gives the correct count of edges.
Time Complexity: The leader election and BFS tree construction each take $O(diam(G))$ time, as shown in this chapter. Then, the tree is traversed twice, which also takes $O(diam(G))$ time. It is also clear that we never transmit more than $O(\log n)$ bits in any message, since we only send degree counts and sums, which can be represented in $O(\log n)$ bits.

5.2 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 $O(diam(G))$:

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

Solution

We first run the leader election algorithm to elect a leader node ℓ in $O(diam(G))$ time, followed by a breadth-first tree construction rooted at the leader (also $O(diam(G))$ time).
Then, we perform a breadth-first traversal of the tree, assigning colors to nodes based on their distance from the leader. The leader is colored with color 0, its neighbors with color 1, their neighbors with color 0, and so on. If we encounter an edge that connects two nodes of the same color, we send a wave to indicate that the graph is not bipartite, and wave to set $ans = 0$ for all nodes. Otherwise, if we complete the traversal without finding such an edge, we wave to set $ans = 1$ for all nodes.

Algorithm 6: Bipartite Graph Detection in CONGEST Model

```
Data: Connected graph G = (V, E)
Result: All nodes output 1 if bipartite, otherwise output 0
Run leader election algorithm to elect a leader node ℓ;
Construct a BFS tree rooted at ℓ using breadth-first traversal;
Color the leader with color 0;
for each neighbor v of ℓ do
    Color v with color 1;
end
for each node u in BFS traversal do
    for each neighbor w of u do
        if color(u) == color(w) then
            Send wave to indicate graph is not bipartite;
            Set ans ← 0 for all nodes;
            return All nodes output 0
        end
        else
            Color w with the opposite color of u;
        end
    end
end
Send wave to set ans ← 1 for all nodes;
return All nodes output 1
```

Correctness: Correctness is clear, we are attempting to 2-color the graph, and if we find an edge that connects two nodes of the same color, then the graph is not bipartite. If we successfully color the graph with two colors, then it is bipartite.
Time Complexity: The leader election and BFS tree construction each take $O(diam(G))$ time. The BFS traversal also takes $O(diam(G))$ time, since we are traversing the tree. The wave to set the answer also takes $O(diam(G))$ time. Thus, the overall time complexity is $O(diam(G))$.
The messages we send only contain color information and wave signals, which can be represented in $O(\log n)$ bits, so we respect the bandwidth limitations of the CONGEST model.

5.3 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.

Solution

In a complete graph, each node has degree $n - 1$. Thus, each node can simply send all it's neighbours it's degree. In the next round, each node checks if all received degrees are $n - 1$. If so, it outputs 1, otherwise it outputs 0.

Algorithm 7: Complete Graph Detection in CONGEST Model

```
Data: Connected graph G = (V, E)
Result: All nodes output 1 if complete graph, otherwise output 0
for each node v do
    Send degree deg(v) to all neighbors;
end
for each node v do
    if all received degrees equal n − 1 and deg(v) = n − 1 then
        Output 1;
    end
    else
        Send wave to indicate graph is not complete;
        Output 0;
    end
end
```

Correctness: Trivial.

Time Complexity: We only send messages for 2 rounds, so the time complexity is $O(1)$. The messages we send only contain degree information and wave signals, which can be represented in $O(\log n)$ bits, so we respect the bandwidth limitations of the CONGEST model.

5.4 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.

5.4.1 Solution

We interpret "full information" as the ability to reconstruct the entire graph structure at each node. Suppose there exists an algorithm that gathers full information of the input graph G in cn rounds, under a bandwidth of $\lg(bn)$ bits per message, where $b, c > 0$ are constants. Consider the case when G is in the form $v - u \in (G - v)$, i.e. there exists a node v such that it is connected to $G - v$ by exactly one edge. Then, v only receives $\lg(bn)$ bits of information from $G - v$ in each round. Therefore, after cn rounds, there are only $(bn)^{cn}$ reachable states for v. However, there are $2^{\Theta(n^2)}$ possible configurations for $G - v$. Observe that

$$\lg((bn)^{cn}) = cn \lg(bn) = (\lg b)cn + cn \lg n \in O(n \log n) \in o(n^2). \tag{13}$$

In other words, by the pigeonhole principle, there exist two different configurations of $G - v$ that are indistinguishable to v after cn rounds, which means that v cannot reconstruct the full graph structure. This contradicts the assumption that the algorithm gathers full information in time $O(|V|)$.

6 Randomized Algorithms

6.1 Exercise 6.3 (large independent sets)

Design a randomized PN algorithm A with the following guarantee: in any graph $G = (V, E)$ of maximum degree Δ, algorithm A outputs an independent set I such that the expected size of I is $|V|/O(\Delta)$. The running time of the algorithm should be $O(1)$. You can assume that all nodes know Δ.

Solution

Each node v selects itself with probability $\frac{1}{2\Delta}$, and then sends its decision to all neighbors. If a node v selects itself and none of its neighbors select themselves, then v is added to the independent set I. Otherwise, v is not added to I.

Algorithm 8: Randomized Independent Set Algorithm

Data: Graph $G = (V, E)$ with maximum degree Δ
Result: Independent set I of expected size $|V|/O(\Delta)$
Initialize $I \leftarrow \emptyset$;
for *each node* v **do**
  $s(v) \leftarrow 1$ with probability $\frac{1}{2\Delta}$, and 0 otherwise;
  Send $s(v)$ to all neighbors;
end
for *each node* v **do**
  if $s(v) = 1$ *and* $s(u) = 0$ *for all neighbors* u *of* v **then**
    Add v to I;
  end
end
return I

Expected Size of I: For each node v, the probability that v is added to I is the probability that v selects itself and none of its neighbors select themselves. This is given by:

$$P(v \in I) = P(s(v) = 1) \cdot P(s(u) = 0 \text{ for all neighbors } u) = \frac{1}{2\Delta} \cdot \left(1 - \frac{1}{2\Delta}\right)^{deg(v)}.$$

Since $deg(v) \leq \Delta$, we have:

$$P(v \in I) \geq \frac{1}{2\Delta} \cdot \left(1 - \frac{1}{2\Delta}\right)^{\Delta} \geq \frac{1}{2\Delta} \cdot \frac{1}{e} = \frac{1}{2e\Delta}.$$

Thus, the expected size of I is:

$$\mathbb{E}[|I|] = \sum_{v \in V} P(v \in I) \geq \sum_{v \in V} \frac{1}{2e\Delta} = \frac{|V|}{2e\Delta} = \frac{|V|}{O(\Delta)}.$$

Runtime: The algorithm runs in $O(1)$ time since each node only needs to perform a constant number of operations and communicate with its neighbors once.

6.2 Exercise 6.4 (max cut problem)

Let $G = (V, E)$ be a graph. A cut is a function $f : V \to \{0, 1\}$. An edge $\{u, v\} \in E$ is a cut edge in f if $f(u) \neq f(v)$. The size of cut f is the number of cut edges, and a maximum cut is a cut of the largest possible size.

(a) Prove: If $G = (V, E)$ is a bipartite graph, then a maximum cut has $|E|$ edges.

(b) Prove: If $G = (V, E)$ has a cut with $|E|$ edges, then G is bipartite.

(c) Prove: For any $\alpha > 1/2$, there exists a graph $G = (V, E)$ in which the maximum cut has fewer than $\alpha|E|$ edges.

Solution

(a) If G is bipartite, there exists a partition of V into two disjoint sets V_1 and V_2 such that every edge in E connects a vertex in V_1 to a vertex in V_2. Define the cut function f as follows: $f(v) = 0$ if $v \in V_1$, and $f(v) = 1$ if $v \in V_2$. Then, for every edge $\{u, v\} \in E$, we have $f(u) \neq f(v)$, which means that every edge is a cut edge. Therefore, the size of the cut is $|E|$, which is the maximum possible size.

(b) Partition V by f and let $V_i = \{v \in V \mid f(v) = i\}$ for $i = 0, 1$. Since every edge is a cut edge, there are no edges between vertices in the same set V_i. Therefore, G is bipartite with partition (V_0, V_1).

(c) Consider $K_{2n} = (V, E)$ for some $n \in \mathbb{N}$. Obviously, $|V| = 2n$ and $|E| = \binom{2n}{2} = 2n^2 - n$. Using elementary calculus, we know that the max cut of K_{2n} is n^2. Since

$$\lim_{n \to +\infty} \alpha(n) = \frac{n^2}{2n^2 - n} = \frac{1}{2}, \tag{14}$$

for any $\alpha > 1/2$, we can find n large enough such that $\alpha(n) < \alpha$. Then, the maximum cut of K_{2n} has fewer than $\alpha|E|$ edges.

6.3 Exercise 6.5 (max cut algorithm)

Design a randomized PN algorithm A with the following guarantee: in any graph $G = (V, E)$, algorithm A outputs a cut f such that the expected size of cut f is at least $|E|/2$. The running time of the algorithm should be $O(1)$.

Note that the analysis of algorithm A also implies that for any graph there exists a cut with at least $|E|/2$.

Solution

In the first (and only) round, each node v independently and uniformly at random selects a value $f(v) \in \{0, 1\}$. The expected size of the cut f is given by

$$\mathbb{E}\left[\sum_{u,v \in E} \mathbb{1}[f(u) \neq f(v)]\right] = \sum_{u,v \in E} P(f(u) \neq f(v)) = \sum_{u,v \in E} \frac{1}{2} = \frac{|E|}{2}. \tag{15}$$

To find a cut with at least $|E|/2$ edges, use the method of conditional expectation.

6.4 Exercise 6.6 (Maximal Independent Sets)

Design a randomized PN algorithm that finds a maximal independent set in time $O(\Delta + \log n)$ with high probability.

Solution

We first run the randomized coloring algorithm to get a $(\Delta + 1)$-coloring of the graph. Then, we process each color in order, adding all feasible nodes to the independent set.

Algorithm 9: Randomized Maximal Independent Set Algorithm

```
Data: Graph G = (V, E) with (Δ + 1)-coloring
Result: Maximal independent set I
Initialize I ← ∅;
Run the randomized coloring algorithm to get a (Δ + 1)-coloring of the graph;
for each color c from 1 to Δ + 1 do
    if node v has color c then
        if node v did not recieve a termination signal then
            Add v to I;
            Send termination signal to all neighbors of v;
        end
    end
end
return I
```

Correctness: It is clear that we never add any adjacent pair. To see that I is maximal, assume for contradiction that there exists a node u that is not in I and is not adjacent to any node in I. Then, u must have proposed to be added to I at some point, and since it was not added, it must have received a termination signal from one of its neighbors, which is a contradiction.

Runtime: The randomized coloring algorithm runs in $O(\log n)$ time. Processing each color only takes 1 rounds, and there are $\Delta + 1$ colors, so the total time complexity is $O(\Delta + \log n)$ - as required.

6.5 Exercise 6.7 (Maximal Independent Set Quickly)

Design a randomized distributed algorithm that finds a maximal independent set in time $O(\log n)$ with high probability.

Solution

Each node v chooses a random number $v \in [0, 1]$. Then, each non-terminated node keeps on selecting a local maximum.

Algorithm 10: Randomized Maximal Matching Algorithm

```
Data: Graph G = (V, E)
Result: Maximal matching M
Initialize M ← ∅;
Initialize all nodes as active;
while there exists an active node do
    for each active node v do
        r(v) ← random number in [0, 1];
        Send r(v) to all neighbors;
    end
    for each active node v do
        if r(v) > r(u) for all active neighbors u then
            Add v to M;
            Terminate v and all it's neighbours;
        end
    end
end
return M
```

Correctness: It is clear that we never add any adjacent pair to M. To see that M is maximal, assume there exists v such that v is not selected and none of it's neighbours are selected. It follows that node v is active (since it did not generate or recieve a termination signal), which is a contradiction since all nodes must terminate to return M.

Runtime: Similar to the analysis of the randomized coloring algorithm, we show that any node has at least a constant probability of being terminated. Similar analysis follows that with high probability, the algorithm takes $O(\log n)$ time.

Consider any node u. Then either u is a local maximum, or it is not - condition on this:

$$P(u \text{ is terminated}) = P(u \text{ is local maximum}) + P(u \text{ is not local maximum and has a local maximum neighbor})$$
$$= \frac{1}{deg(u)+1} + \sum_{n \in N(u)} \frac{1}{deg(n)} \geq \frac{1}{deg(u)+1} + \frac{deg(u)}{\Delta}$$

There are two cases: either $deg(u) \in \theta(\Delta)$, or $deg(u) \in o(\Delta)$. In the first case,

$$P(u \text{ is terminated}) \geq \frac{1}{deg(u)+1} + \frac{deg(u)}{\Delta} \geq \frac{deg(u)}{\Delta} \in \theta(1).$$

In the second case,

$$P(u \text{ is terminated}) \geq \frac{1}{deg(u)+1} + \frac{deg(u)}{\Delta} \geq \frac{1}{deg(u)+1} \in \theta(1).$$

Thus, in either case, there exists a constant c such that $P(u \text{ is terminated}) \geq c$. This is what we needed to show. □

7 Covering Maps

7.1 Exercise 7.2 (homogeneity)

Assume that G is homogeneous and it contains a node of degree at least two. Give several examples of graph problems that cannot be solved with any deterministic PN-algorithm in any family of graphs that contains G.

Solution

Let G be a homogeneous graph that contains a node of degree at least two. By definition of homogeneity, there exist port-numbered networks N and N' such that:

- N is simple and has G as its underlying graph,

- N' consists of a single node,
- there is a covering map $\varphi : N \to N'$.

By Theorem 7.1 (covering maps preserve indistinguishability), any deterministic PN-algorithm A must produce identical outputs at all nodes of N. Hence, for any input f, the output is a constant function:

$$g(v) = c \quad \text{for all } v \in V.$$

Therefore, any graph problem for which a constant labeling is not a feasible solution on G cannot be solved by a deterministic PN-algorithm in any graph family that contains G.

Examples of such problems:

- Maximal Independent Set (MIS).
- Maximal Matching.
- Vertex Coloring (proper coloring).
- Weak Coloring.
- Minimum Dominating Set (or any constant-factor approximation).
- Edge Coloring.
- 2-Approximation (or better) of Vertex Cover.

7.2 Exercise 7.4 (complete graphs)

Recall that 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 $\{u, v\} \in E$. Show that

- any $2k$-regular graph is homogeneous,
- any complete graph with $2k$ nodes has a 1-factorization,
- any complete graph is homogeneous.

Solution

- Let G be a $2k$-regular graph. We construct a port-numbered network N with underlying graph G as follows: for each node v, we pair up its $2k$ incident edges arbitrarily and assign port numbers so that the two edges in each pair have the same port number. This creates a perfect matching of the edges at each node, and thus a covering map to a single-node network can be defined by mapping all nodes of G to the single node in N'. Hence, G is homogeneous.
- A complete graph with $2k$ nodes has a perfect matching (1-factor) because it is regular of degree $2k - 1$. By repeatedly removing perfect matchings, we can decompose the complete graph into $2k - 1$ perfect matchings, which gives us a 1-factorization.
- Since any complete graph with an even number of nodes has a 1-factorization, we can construct a port-numbered network for any complete graph by assigning port numbers according to the 1-factorization. This allows us to define a covering map to a single-node network, making any complete graph homogeneous.

7.3 Exercise 7.5 (dominating sets)

Let $\Delta \in \{2, 3, ...\}$, let $\varepsilon > 0$, and let $\mathscr{F}$ consist of all graphs of maximum degree at most Δ. Show that it is possible to find a $(\Delta + 1)$-approximation of a minimum dominating set in constant time in family $\mathscr{F}$ with a deterministic PN-algorithm. Show that it is not possible to find a $(\Delta + 1 - \varepsilon)$-approximation with a deterministic PN-algorithm.

Solution

- **Upper bound (algorithm).** The trivial dominating set consisting of all vertices is a $(\Delta + 1)$-approximation. Indeed, every vertex in any dominating set can dominate at most its closed neighbourhood of size at most $\Delta + 1$, so for the size $\gamma(G)$ of a minimum dominating set we have

$$n \leq (\Delta + 1)\,\gamma(G).$$

  Therefore the set V has size at most $(\Delta + 1)\gamma(G)$, and so it is a $(\Delta + 1)$-approximation. This set can be produced in 0 rounds (each node simply decides to include itself), hence a deterministic PN-algorithm computes a $(\Delta + 1)$-approximation in constant time.

- **Lower bound (indistinguishability).** Fix any constant running time t and any deterministic PN-algorithm A that runs in t rounds. We give a family of graphs of maximum degree Δ on which A cannot achieve approximation ratio $\Delta + 1 - \varepsilon$ for any $\varepsilon > 0$.
  Let k be an arbitrary positive integer and consider the disjoint union of k copies of the complete graph $K_{\Delta+1}$. In each copy, label the vertices by elements of $\mathbb{Z}_{\Delta+1}$. For a copy, define port numbers so that for distinct vertices $i, j \in \mathbb{Z}_{\Delta+1}$ the port at i leading to j is $(j - i) \bmod (\Delta + 1)$ (mapped to $\{0, \ldots, \Delta - 1\}$). This port-numbering is translation-invariant inside each clique, so every vertex in every copy has an identical radius-t neighbourhood (the labelled, port-numbered t-ball looks the same at every vertex).
  Since A is deterministic and sees the same t-neighbourhood at every vertex, all vertices must make the same output decision. The only uniform choices that yield a dominating set are the whole vertex set or the empty set; the empty set is not dominating, so A must output the whole vertex set. Thus A's output has size $k(\Delta + 1)$, while an optimal dominating set has size k (one vertex per clique). The approximation ratio on these instances is

$$\frac{k(\Delta + 1)}{k} = \Delta + 1.$$

  As k was arbitrary, this shows that no constant-time deterministic PN-algorithm can guarantee an approximation ratio strictly better than $\Delta + 1$ (in particular not $\Delta + 1 - \varepsilon$ for any $\varepsilon > 0$).

7.4 Exercise 7.9 (k-fold covers)

Let $N = (V, P, p)$ and $N' = (V', P', p')$ be simple port-numbered networks such that the underlying graphs of N and N' are connected, and assume that $\varphi : V \to V'$ is a covering map from N to N'. Prove that there exists a positive integer k such that the following holds: $|V| = k|V'|$ and for each node $v' \in V'$ we have $|\varphi^{-1}(v')| = k$. Show that the claim does not necessarily hold if the underlying graphs are not connected.

Solution

Take any $v' \in V'$ and let $k = |\varphi^{-1}(v')|$. Let u' be any neighbor of v' such that $p'(v', i) = (u, j)$ for some i, j. Since φ is a covering map, for each $v \in \varphi^{-1}(v')$, there exists a unique neighbor u of v such that $p(v, i) = (u, j)$ and $\varphi(u) = u'$. This implies that $|\varphi^{-1}(u')| = k$. By repeating this argument, we can show that for any node $w' \in V'$, we have $|\varphi^{-1}(w')| = k$. Since the underlying graph of N' is connected, we can reach any node from v' by a path, and thus the above argument applies to all nodes in V'. Therefore, we have $|V| = k|V'|$ and for each node $v' \in V'$ we have $|\varphi^{-1}(v')| = k$.

For a counterexample when the underlying graphs are not connected, consider the case when N' consists of two nodes u' and v' without any connections. Let N consist of three nodes u_1, u_2 and v, also without any connections. If we set $\varphi(u_1) = \varphi(u_2) = u'$ and $\varphi(v) = v'$, then φ is a covering map from N to N', but clearly the claim does not hold.

8 Local Neighborhoods

8.1 Exercse 8.1 (Edge Coloring)

In this exercise, the graph family consists of path graphs.

1. Show that it is possible to find a 2-edge coloring in time $O(n)$ with deterministic PN-algorithms.

2. Show that it is not possible to find a 2-edge coloring in time $o(n)$ with deterministic PN-algorithms.

3. Show that it is not possible to find a 2-edge coloring in time $o(n)$ with deterministic LOCAL-algorithms.

Solution

We consider the family of path graphs.

1. **A deterministic PN-algorithm that finds a 2-edge coloring in time $O(n)$.** Let $G = (V, E)$ be a path on n nodes. We construct a simple algorithm:

   - Pick an arbitrary node v_0 as a reference (this will be implicitly determined by the port numbering).
   - Simulate a wave that traverses the path from one endpoint to the other.
   - Each node determines its distance (parity) from the endpoint.

   Thus each node computes its distance from the closest endpoint. This takes $O(n)$ rounds. Now color each edge $\{u, v\}$ by:

   $$\text{color}(\{u, v\}) = \begin{cases} 0 & \text{if } \min(d(u), d(v)) \text{ is even,} \\ 1 & \text{otherwise.} \end{cases}$$

   This produces an alternating coloring along the path, hence a valid 2-edge coloring. Therefore, a deterministic PN-algorithm exists with time complexity $O(n)$.

2. **It is not possible to find a 2-edge coloring in time $o(n)$ with deterministic PN-algorithms.** We use the given theorem (indistinguishability via isomorphic neighborhoods). Assume, for contradiction, that there exists a deterministic PN-algorithm A that finds a 2-edge coloring in time $t(n) = o(n)$. Consider a sufficiently long path G with n nodes, where $n \gg t(n)$. Pick an edge $e = \{u, v\}$ that is far from both endpoints:

   $$\text{dist}(u, \text{endpoints}) > t(n), \quad \text{dist}(v, \text{endpoints}) > t(n).$$

   Then the radius-$t(n)$ neighborhoods of u and v are isomorphic (both look like infinite paths locally). By the theorem, their local views are identical, hence:

   $$x_{t(n)}(u) = x_{t(n)}(v).$$

   This implies that the two endpoints of edge e produce identical outputs, so the edge must be colored the same way from both sides. However, since all such edges in the middle of the path have identical neighborhoods, the algorithm must assign the same color pattern everywhere in the interior. This leads to a contradiction: in a valid 2-edge coloring of a path, adjacent edges must alternate colors. A locally indistinguishable algorithm cannot consistently enforce this alternation without seeing a boundary (endpoint). Thus, no deterministic PN-algorithm can solve 2-edge coloring in time $o(n)$.

3. **It is not possible to find a 2-edge coloring in time $o(n)$ with deterministic LOCAL-algorithms.** The argument is similar, but even stronger. In the LOCAL model, nodes may have unique identifiers, but in time $t(n) = o(n)$ they can only see their radius-$t(n)$ neighborhoods. Consider again a sufficiently long path and two adjacent edges in the middle, far from endpoints. We can assign identifiers so that the radius-$t(n)$ neighborhoods of these edges are isomorphic (this is always possible by choosing IDs consistently). By the same indistinguishability theorem, the algorithm must behave identically on these neighborhoods. Hence, it cannot distinguish positions along the path and enforce a globally consistent alternating pattern. But a valid 2-edge coloring requires:

   $$\text{adjacent edges have different colors.}$$

   Without global coordination (which requires $\Omega(n)$ time), the algorithm cannot ensure consistent alternation.Therefore, no deterministic LOCAL-algorithm can find a 2-edge coloring in time $o(n)$.

8.2 Exercise 8.2 (maximal matching)

In this exercise, the graph family consists of path graphs.

1. Show that it is possible to find a maximal matching in time $O(n)$ with deterministic PN-algorithms.
2. Show that it is not possible to find a maximal matching in time $o(n)$ with deterministic PN-algorithms.
3. Show that it is possible to find a maximal matching in time $o(n)$ with deterministic LOCAL-algorithms.

Solution

1. **Deterministic PN-algorithm with running time $O(n)$.** Choose an arbitrary endpoint (the port numbering implicitly gives a consistent orientation, e.g., port 1 points towards the left end). Perform a wave that propagates from the left endpoint to the right; each node learns its distance d from the left endpoint (this takes $O(n)$ rounds). Then include the edge $\{u, v\}$ in the matching if the smaller of the two distances is even. This produces a matching that is maximal: every node is either incident to a chosen edge or adjacent to a chosen edge. Hence a deterministic PN-algorithm exists with time complexity $O(n)$.

2. **Lower bound for deterministic PN-algorithms.** Assume, for contradiction, that there is a deterministic PN-algorithm A that finds a maximal matching on any path in time $t(n) = o(n)$. Consider a sufficiently long path G with n nodes, where the port numbers are chosen consistently (e.g., port 1 towards the left end). Pick two adjacent edges $e_1 = \{u, v\}$ and $e_2 = \{v, w\}$ in the middle of the path such that the distance from u and w to the nearest endpoint is larger than $t(n)$. Then the radius-$t(n)$ neighborhoods of u and w are isomorphic (they both look like an infinite path with the same port pattern). By Theorem 8.1, the states of u and w after $t(n)$ rounds are identical; similarly, the state of v is the same in both neighbourhoods. Consequently, the algorithm must decide the same status (either "in the matching" or "not in the matching") for both edges e_1 and e_2. However, in any maximal matching of a path, two adjacent edges cannot both be in the matching (they share a vertex) and they cannot both be out of the matching (otherwise the middle vertex v would be free and both incident edges could be added). Hence a contradiction. Therefore $t(n)$ cannot be $o(n)$; any deterministic PN-algorithm requires $\Omega(n)$ rounds.

3. **Deterministic LOCAL-algorithm with running time $o(n)$.** In the LOCAL model every node has a unique identifier. We first compute a proper 2-coloring of the vertices of the path using the Cole–Vishkin algorithm; this takes $O(\log^* n)$ rounds. After the 2-coloring, each node knows its own colour (say black or white) and the colours of its neighbours. Now we construct a maximal matching in constant additional rounds as follows:

   - Each node sends its identifier to its neighbours.
   - For every edge, orient it from the endpoint with the larger identifier to the endpoint with the smaller identifier.
   - Every node that has at least one outgoing edge selects the outgoing edge that leads to the neighbour with the largest identifier among all smaller neighbours, and sends a proposal along that edge.
   - A node that receives one or more proposals accepts the proposal coming from the neighbour with the largest identifier (identifiers are unique, so there is no tie).

   The set of accepted edges forms a matching, and it is maximal because any edge that is not selected would have both endpoints free, which cannot happen due to the orientation and acceptance rule. The whole procedure runs in $O(\log^* n) = o(n)$ rounds. Hence a deterministic LOCAL-algorithm can find a maximal matching in time $o(n)$ (in fact in $O(\log^* n)$).

8.3 Exercise 8.3 (optimization)

In this exercise, the graph family $\mathscr{F}$ consists of path graphs. Can we solve the following problems with deterministic PN-algorithms? If yes, how fast? Can we solve them any faster in the LOCAL model?

1. Minimum vertex cover.
2. Minimum dominating set.
3. Maximum independent set.

Solution

1. **Minimum vertex cover.** Yes, a deterministic PN-algorithm can compute a minimum vertex cover in time $O(n)$. Using the port numbering, we can orient the path and propagate a dynamic programming wave from one endpoint to the other. For instance, each node computes two values: the size of a minimum vertex cover of the already processed prefix when the node is taken or not taken. After $O(n)$ rounds the information reaches the other endpoint, and the optimal value can be extracted. In the LOCAL model, any algorithm that outputs the exact minimum vertex cover must have time at least $\Omega(n)$. Indeed, the solution depends on the parity of the total number of vertices (or the exact positions of endpoints), which cannot be determined in $o(n)$ rounds because the diameter of a path is $n - 1$.
2. **Minimum dominating set.** Similarly, a deterministic PN-algorithm can compute a minimum dominating set in time $O(n)$ by a wave algorithm that propagates appropriate DP information (states representing whether the current node or its neighbour is already dominated). In the LOCAL model, the same $\Omega(n)$ lower bound applies: the exact domination number of a path (which is $\lceil n/3 \rceil$) depends on n modulo 3, and learning the global size or the endpoints requires $\Omega(n)$ rounds.
3. **Maximum independent set.** The same reasoning holds. A deterministic PN-algorithm can compute a maximum independent set in $O(n)$ rounds via a DP wave (e.g., standard linear-time algorithm for paths). In the LOCAL model, the problem is essentially equivalent to minimum vertex cover (by complementation) and also requires $\Omega(n)$ rounds. No deterministic LOCAL-algorithm can solve any of these three problems in $o(n)$ time because they are global optimisation tasks that need knowledge of the whole path.

9 Round Elimination

9.1 Exercise 9.5 (Solving Weak 3-labeling)

Present a 2-round deterministic PN-algorithm for solving weak 3-labeling in $(3, 2)$-biregular trees.

Solution

The main difficulty is to break symmetry so that the two edges incident to each passive node are treated differently. In the first round, each passive node deterministically selects one of its two incident edges (using port numbers). This creates an asymmetry visible to the adjacent active nodes. In the second round, each active node assigns labels to its three incident edges based on which of them were selected by their passive endpoints. Since each passive node selects exactly one of its edges, the two edges incident to it will be assigned different labels.

Algorithm 11: Weak 3-Labeling in $(3, 2)$-biregular trees

Round 1 (Passive → Active):;
forall *passive nodes* u **do**
  Let the two incident ports be $\{1, 2\}$;
  Select the edge corresponding to port 1;
  forall *neighbors* v **do**
    if v *is connected via port 1* **then**
      send message `selected` to v;
    end
    else
      send message `not-selected` to v;
    end
  end
end
Round 2 (Active nodes compute labels):;
forall *active nodes* v **do**
  Let e_1, e_2, e_3 be the incident edges;
  Let $S \subseteq \{e_1, e_2, e_3\}$ be edges marked `selected`;
  Assign distinct labels from $\{1, 2, 3\}$ to e_1, e_2, e_3 such that edges in S receive the smallest labels (in arbitrary order), and remaining edges receive the remaining labels;
end

Correctness: We prove that the algorithm produces a valid weak 3-labeling.

(1) Well-defined labeling. Each active node has exactly 3 incident edges and assigns the labels $\{1, 2, 3\}$ bijectively to them. Hence every edge receives exactly one label.

(2) Passive node constraint. Let u be any passive node with two incident edges e and e'. By construction in Round 1, u selects exactly one of these edges, say e, and marks the other e' as not selected.

Let v and v' be the active endpoints of e and e' respectively. Then:

- Edge e is marked `selected` at v,
- Edge e' is marked `not-selected` at v'.

At each active node, edges marked `selected` are assigned labels from a set disjoint from those assigned to non-selected edges (since labels $\{1, 2, 3\}$ are assigned injectively and selected edges are given priority in label assignment).

Therefore, e and e' must receive different labels. Hence, the two edges incident to u are labeled differently.

(3) Termination and complexity. The algorithm consists of exactly two communication rounds:

- Round 1: passive nodes send selection information,
- Round 2: active nodes compute labels locally.

Thus the algorithm runs in 2 rounds.

9.2 Exercise 9.4 (iterated round elimination)

Fill in the missing details in Section 9.2.6 to show that formula (9.1) is a correct definition of the active configurations for problem Π_2 (i.e., it contains all possible configurations and only them).

Solution

We work with $\Sigma_1 = \{1, 2, 3\}$ after the simplification described in the text. Recall that $\Pi_2 = \mathrm{re}(\Pi_1)$ where Π_1 is the output problem of weak 3-labeling. For Π_2 the active nodes have degree 3 (since we swap the degrees again) and the alphabet is $\Sigma_2 = \{\, X \subseteq \Sigma_1 \mid X \neq \varnothing \,\}$. A triple (X, Y, Z) of non-empty subsets of $\{1, 2, 3\}$ belongs to A_2 iff

$$\forall x \in X,\ y \in Y,\ z \in Z\ :\ (x, y, z) \in P_1.$$

We already know that P_1 consists of all triples over Σ_1 except $(1,1,1)$, $(2,2,2)$ and $(3,3,3)$. Hence the condition is equivalent to:

$$\text{there is no } c \in \{1,2,3\} \text{ such that } c \in X,\ c \in Y,\ c \in Z.$$

In other words

$$X \cap Y \cap Z = \varnothing. \tag{1}$$

We now show that (1) holds exactly for the triples listed in (9.1). The four families in (9.1) are:

$$\begin{aligned}
\mathcal{F}_1 &: X \subseteq \{1,2\},\ Y \subseteq \{1,3\},\ Z \subseteq \{2,3\},\\
\mathcal{F}_2 &: X \subseteq \{1\},\ Y \subseteq \{2,3\},\ Z \subseteq \{1,2,3\},\\
\mathcal{F}_3 &: X \subseteq \{2\},\ Y \subseteq \{1,3\},\ Z \subseteq \{1,2,3\},\\
\mathcal{F}_4 &: X \subseteq \{3\},\ Y \subseteq \{1,2\},\ Z \subseteq \{1,2,3\},
\end{aligned}$$

where all subsets are understood to be non-empty.

From (9.1) to (1). For $\mathcal{F}_1$ we have $3 \notin X$, $2 \notin Y$, $1 \notin Z$, so no element belongs to all three sets. For $\mathcal{F}_2$, $X = \{1\}$ and $Y \subseteq \{2,3\}$ imply $2,3 \notin X$ and $1 \notin Y$, hence $X \cap Y \cap Z = \varnothing$. The same argument applies to $\mathcal{F}_3$ and $\mathcal{F}_4$ by symmetry. Thus every triple from (9.1) satisfies (1).

From (1) to (9.1). Assume X, Y, Z are non-empty subsets of $\{1,2,3\}$ with $X \cap Y \cap Z = \varnothing$. Consider the three elements $1,2,3$. For each $c \in \{1,2,3\}$ at least one of X, Y, Z does not contain c. Define $U = \{c \notin X\}$, $V = \{c \notin Y\}$, $W = \{c \notin Z\}$. Then $U \cup V \cup W = \{1,2,3\}$. One verifies by a case analysis (omitted for brevity) that any triple satisfying (1) must belong to at least one of the families $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3, \mathcal{F}_4$. The essential observation is that the four families together cover all possibilities where no element belongs to all three sets.
Therefore every triple with $X \cap Y \cap Z = \varnothing$ is listed in (9.1).
Combining both directions, we conclude that formula (9.1) correctly defines the active configurations for Π_2.

9.3 Exercise 9.6 (sinkless orientation)

Consider the sinkless orientation problem Π on $(3,3)$-biregular trees from Section 9.1.2. Compute the output problems $\mathrm{re}(\Pi)$ and $\mathrm{re}(\mathrm{re}(\Pi))$; include a justification for your results.

Solution

We fix a bipartition of the $(3,3)$-biregular tree into active and passive nodes (both have degree 3). The label set is $\Sigma_0 = \{0,1\}$ where 0 means "oriented from active to passive" and 1 means "oriented from passive to active". The local constraints are:

- At an active node: at least one incident edge carries label 0 (outgoing). Hence the active configuration set is
$$A_0 = \big\{[a_1,a_2,a_3] \in \Sigma_0^3 \mid \text{at least one } a_i = 0\big\}.$$
- At a passive node: at least one incident edge carries label 1 (outgoing from the passive side). Hence the passive configuration set is
$$P_0 = \big\{[b_1,b_2,b_3] \in \Sigma_0^3 \mid \text{at least one } b_i = 1\big\}.$$

First output problem $\Pi_1 = \mathrm{re}(\Pi)$. We swap the roles of active and passive nodes; now active nodes have degree 3 (formerly passive) and passive nodes have degree 3 (formerly active). The alphabet Σ_1 consists of all non-empty subsets of Σ_0:

$$\Sigma_1 = \big\{\{0\},\ \{1\},\ \{0,1\}\big\}.$$

Active configurations A_1: an active node v (degree 3) outputs a multiset $[X_1, X_2, X_3]$ with $X_i \in \Sigma_1$. The condition is: for every choice $x_1 \in X_1$, $x_2 \in X_2$, $x_3 \in X_3$, the triple (x_1, x_2, x_3) must belong to

P_0 (the old passive condition). P_0 contains all triples except $[0,0,0]$. Thus we must avoid the situation where we can pick $x_i = 0$ for all i. This happens exactly when $0 \in X_1$, $0 \in X_2$ and $0 \in X_3$ simultaneously. Hence the condition is: not ($0 \in X_1$ and $0 \in X_2$ and $0 \in X_3$). Equivalently, at least one X_i does **not** contain 0, i.e. at least one X_i equals $\{1\}$. Therefore

$$A_1 = \big\{[X_1, X_2, X_3] \mid X_i \in \Sigma_1, \text{ and } \{1\} \in \{X_1, X_2, X_3\}\big\}.$$

Passive configurations P_1: a passive node (degree 3) outputs a multiset $[Y_1, Y_2, Y_3]$ with $Y_i \in \Sigma_1$. The condition is: there exist $y_1 \in Y_1$, $y_2 \in Y_2$, $y_3 \in Y_3$ such that $(y_1, y_2, y_3) \in A_0$. A_0 contains all triples except $[1,1,1]$. Thus we need to avoid the situation where every possible choice yields only $(1,1,1)$. That would happen only if $Y_1 = Y_2 = Y_3 = \{1\}$ (because then the only choice is $(1,1,1) \notin A_0$). Hence

$$P_1 = \Sigma_1^3 \setminus \big\{[\{1\}, \{1\}, \{1\}]\big\}.$$

Second output problem $\Pi_2 = \mathrm{re}(\Pi_1)$. We swap roles again, returning to $(3,3)$-biregular trees with the original active/passive partition. Now Σ_2 consists of all non-empty subsets of Σ_1. Let us rename the elements of Σ_1 for brevity:

$$a = \{0\}, \qquad b = \{1\}, \qquad c = \{0,1\}.$$

Then $\Sigma_1 = \{a, b, c\}$ and

$$\Sigma_2 = \big\{\{a\}, \{b\}, \{c\}, \{a,b\}, \{a,c\}, \{b,c\}, \{a,b,c\}\big\}.$$

Active configurations A_2: an active node outputs $[X_1, X_2, X_3]$ with $X_i \in \Sigma_2$. The condition: for every choice $x_1 \in X_1$, $x_2 \in X_2$, $x_3 \in X_3$, we must have $(x_1, x_2, x_3) \in P_1$. P_1 is the set of all triples over Σ_1 except (b,b,b). Thus the only forbidden triple is (b,b,b). Therefore we must avoid the possibility of picking b from every X_i. That is, it must not be the case that $b \in X_1$, $b \in X_2$ and $b \in X_3$ simultaneously. Equivalently,

$$A_2 = \big\{[X_1, X_2, X_3] \mid X_i \in \Sigma_2, \text{ and } (\exists i)\ b \notin X_i\big\}.$$

Passive configurations P_2: a passive node outputs $[Y_1, Y_2, Y_3]$ with $Y_i \in \Sigma_2$. The condition: there exist $y_1 \in Y_1$, $y_2 \in Y_2$, $y_3 \in Y_3$ such that $(y_1, y_2, y_3) \in A_1$. Recall A_1 consists of all triples over Σ_1 that contain at least one b. Hence we need to be able to pick at least one b among the chosen elements. This is possible iff **not** all Y_i are subsets of $\Sigma_1 \setminus \{b\} = \{a, c\}$. In other words, at least one Y_i contains b. Thus

$$P_2 = \big\{[Y_1, Y_2, Y_3] \mid Y_i \in \Sigma_2, \text{ and } (\exists i)\ b \in Y_i\big\}.$$

We have therefore fully described $\mathrm{re}(\Pi)$ and $\mathrm{re}(\mathrm{re}(\Pi))$.

10 Sinkless Orientation

10.1 Exercise 10.1 (0-Round Solvability)

Prove that the following problems are not 0-round solvable.

1. $(d+1)$-edge coloring in (d,d)-biregular trees (see Section 9.1.2).
2. Maximal matching in (d,d)-biregular trees (see Section 9.1.2).
3. $\prod_1$, the output problem of sinkless orientation, in $(3,3)$-biregular trees (see Section 10.4.2).

Solution

In a 0-round deterministic PN-algorithm, nodes do not communicate. Hence, each node must decide its output solely based on its own local port numbering. In particular:

- All nodes of the same degree behave identically.
- The algorithm corresponds to fixing one *active configuration*, i.e., assigning an output label to each port $1, 2, \ldots, d$.

Thus, the behavior of the algorithm is fully determined by how it labels the ports of a node of degree d. We will show that for any such fixed assignment, there exists a port numbering of the graph such that at some node, the incident edges do not satisfy any valid *passive configuration*. This yields a contradiction.

(a) $(d+1)$-edge coloring in (d,d)-biregular trees.

In a valid $(d+1)$-edge coloring:

- Each node must assign distinct colors to its d incident edges.

In a 0-round algorithm, each node assigns colors to its ports:

$$(1, 2, \ldots, d) \mapsto (c_1, c_2, \ldots, c_d).$$

Now consider an edge $\{u, v\}$:

- At node u, suppose this edge is connected to port i and gets color c_i.
- At node v, suppose the same edge is connected to port j and gets color c_j.

Since the algorithm is fixed, c_i and c_j are predetermined. By choosing a port numbering such that $c_i \neq c_j$, we obtain a conflict: the same edge receives two different colors. Hence, no 0-round algorithm can guarantee a valid $(d+1)$-edge coloring.

(b) Maximal matching in (d,d)-biregular trees.

In a maximal matching:

- Each node is incident to at most one matched edge.
- Every unmatched edge must have at least one endpoint already matched.

A 0-round algorithm assigns to each port whether the corresponding edge is in the matching.
Consider any fixed assignment $(x_1, \ldots, x_d)$ where $x_i \in \{0, 1\}$.
We construct a port numbering as follows:

- At some node v, arrange the incident edges so that multiple ports with $x_i = 1$ correspond to distinct edges.

Then node v is incident to more than one matched edge, violating the matching constraint. Hence, no 0-round algorithm can produce a maximal matching.

(c) Π_1 (sinkless orientation) in $(3,3)$-biregular trees.

In sinkless orientation:

- Each edge must be oriented.
- Each node must have at least one outgoing edge.

A 0-round algorithm assigns a direction to each port:

$$(1, 2, 3) \mapsto (\text{in/out}, \text{in/out}, \text{in/out}).$$

Consider any such assignment. We construct a port numbering such that:

- At some node v, all incident edges are oriented towards v.

This is always possible by matching ports so that whenever a node assigns "out" to a port, the other endpoint assigns "in" to the corresponding port.
By pairing ports across edges, we ensure that for node v, all edges are incoming. Thus, v is a sink, violating the requirement.
Hence, no 0-round algorithm can solve sinkless orientation.

10.2 Exercise 10.2 (higher degrees)

Generalize the sinkless orientation problem from $(3,3)$-biregular trees to $(10,10)$-biregular trees. Apply round elimination and show that you get again a fixed point.

Solution

The sinkless orientation problem on $(10, 10)$-biregular trees is defined as follows:

- The graph is a $(10, 10)$-biregular tree, where active and passive nodes both have degree 10.
- The label set is $\Sigma_0 = \{0, 1\}$, where 0 means "oriented from active to passive" and 1 means "oriented from passive to active".
- The local constraints are:
  - At an active node: at least one incident edge must be labeled 0 (outgoing).
  - At a passive node: at least one incident edge must be labeled 1 (outgoing from the passive side).

First output problem $\Pi_1 = \mathrm{re}(\Pi)$. We swap the roles of active and passive nodes; now active nodes have degree 10 (formerly passive) and passive nodes have degree 10 (formerly active). The alphabet Σ_1 consists of all non-empty subsets of Σ_0:

$$\Sigma_1 = \big\{\{0\},\ \{1\},\ \{0,1\}\big\}.$$

Active configurations A_1: an active node v (degree 10) outputs a multiset $[X_1, X_2, \ldots, X_{10}]$ with $X_i \in \Sigma_1$. The condition is: for every choice $x_1 \in X_1$, $x_2 \in X_2$, $\ldots$, $x_{10} \in X_{10}$, the tuple $(x_1, x_2, \ldots, x_{10})$ must belong to P_0 (the old passive condition). P_0 contains all tuples except the one where all entries are 0. Thus we must avoid the situation where we can pick 0 from every X_i. This happens exactly when $0 \in X_i$ for all i. Hence the condition is: not ($0 \in X_1$ and $0 \in X_2$ and $\ldots$ and $0 \in X_{10}$). Equivalently, at least one X_i does **not** contain 0, i.e. at least one X_i equals $\{1\}$. Therefore

$$A_1 = \big\{[X_1, X_2, \ldots, X_{10}] \mid X_i \in \Sigma_1, \text{ and } \{1\} \in \{X_1, X_2, \ldots, X_{10}\}\big\}.$$

Passive configurations P_1: a passive node (degree 10) outputs a multiset $[Y_1, Y_2, \ldots, Y_{10}]$ with $Y_i \in \Sigma_1$. The condition is: there exist $y_1 \in Y_1$, $y_2 \in Y_2$, $\ldots$, $y_{10} \in Y_{10}]$ such that $(y_1, y_2, \ldots, y_{10}) \in A_0$. A_0 contains all tuples except the one where all entries are 1. Thus we need to avoid the situation where every possible choice yields only the tuple of all 1s. That would happen only if $Y_i = \{1\}$ for all i (because then the only choice is the tuple of all 1s, which is not in A_0). Hence

$$P_1 = \Sigma_1^{10} \setminus \big\{[\{1\}, \{1\}, \ldots, \{1\}]\big\}.$$

Second output problem $\Pi_2 = \mathrm{re}(\Pi_1)$. We swap roles again, returning to $(10, 10)$-biregular trees with the original active/passive partition. Now Σ_2 consists of all non-empty subsets of Σ_1. Let us rename the elements of Σ_1 for brevity:

$$a = \{0\}, \qquad b = \{1\}, \qquad c = \{0, 1\}.$$

Then $\Sigma_1 = \{a, b, c\}$ and

$$\Sigma_2 = \big\{\{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\big\}.$$

Active configurations A_2: an active node outputs $[X_1, X_2, \ldots, X_{10}]$ with $X_i \in \Sigma_2$. The condition: for every choice $x_1 \in X_1$, $x_2 \in X_2$, $\ldots$, $x_{10} \in X_{10}$, we must have $(x_1, x_2, \ldots, x_{10}) \in P_1$. P_1 is the set of all tuples over Σ_1 except the one where all entries are b. Thus the only forbidden tuple is the one where all entries are b. Therefore we must avoid the possibility of picking b from every X_i. That is, it must not be the case that $b \in X_i$ for all i. Equivalently,

$$A_2 = \big\{[X_1, X_2, \ldots, X_{10}] \mid X_i \in \Sigma_2, \text{ and } (\exists i)\ b \notin X_i\big\}.$$

Passive configurations P_2: a passive node outputs $[Y_1, Y_2, \ldots, Y_{10}]$ with $Y_i \in \Sigma_2$. The condition: there exist $y_1 \in Y_1$, $y_2 \in Y_2$, $\ldots$, $y_{10} \in Y_{10}]$ such that $(y_1, y_2, \ldots, y_{10}) \in A_1$. Recall A_1 consists of all tuples over Σ_1 that contain at least one b. Hence we need to be able to pick at least one b among the chosen elements. This is possible iff **not** all Y_i are subsets of $\Sigma_1 \setminus \{b\} = \{a, c\}$. In other words, at least one Y_i contains b. Thus

$$P_2 = \big\{[Y_1, Y_2, \ldots, Y_{10}] \mid Y_i \in \Sigma_2, \text{ and } (\exists i)\ b \in Y_i\big\}.$$

We have therefore fully described $\mathrm{re}(\Pi)$ and $\mathrm{re}(\mathrm{re}(\Pi))$.
Comparing Π_2 with Π_1, we see that they are essentially the same problem, just with a more complex alphabet. The conditions on the configurations are structurally identical, indicating that we have reached a fixed point in the round elimination process. Thus, $\mathrm{re}(\mathrm{re}(\Pi))$ is essentially the same problem as $\mathrm{re}(\Pi)$, confirming that we have a fixed point.

10.3 Exercise 10.3 (non-bipartite sinkless orientation)

Define non-bipartite sinkless orientation as the following problem $\Pi = (\Sigma, A, P)$ on $(3,2)$-biregular trees:

- $\Sigma = \{O, I\}$,
- $A = \{[O, x, y] \mid x, y \in \Sigma\}$,
- $P = \{[I, O]\}$.

Prove that applying round elimination to Π leads to a period-2 point, that is, to a problem Π' such that $\Pi' = \mathrm{re}(\mathrm{re}(\Pi'))$.

Solution

We start with the problem $\Pi = (\Sigma, A, P)$ defined as follows:

- $\Sigma = \{O, I\}$,
- $A = \{[O, x, y] \mid x, y \in \Sigma\}$,
- $P = \{[I, O]\}$.

First output problem $\Pi_1 = \mathrm{re}(\Pi)$. We swap the roles of active and passive nodes; now active nodes have degree 2 (formerly passive) and passive nodes have degree 3 (formerly active). The alphabet Σ_1 consists of all non-empty subsets of Σ:

$$\Sigma_1 = \{\{O\}, \{I\}, \{O, I\}\}.$$

Active configurations A_1: an active node v (degree 2) outputs a multiset $[X_1, X_2]$ with $X_i \in \Sigma_1$. The condition is: for every choice $x_1 \in X_1$, $x_2 \in X_2$, the tuple (x_1, x_2) must belong to P (the old passive condition). P contains only the tuple $[I, O]$. Thus we must ensure that for every choice of x_1 and x_2, we get $[I, O]$. This is only possible if $X_1 = \{I\}$ and $X_2 = \{O\}$. Therefore

$$A_1 = \{[\{I\}, \{O\}]\}.$$

Passive configurations P_1: a passive node (degree 3) outputs a multiset $[Y_1, Y_2, Y_3]$ with $Y_i \in \Sigma_1$. The condition is: there exist $y_1 \in Y_1$, $y_2 \in Y_2$, $y_3 \in Y_3$ such that $(y_1, y_2, y_3) \in A$. Recall A consists of all triples where the first element is O. Thus we need to be able to pick O as the first element. This is possible iff at least one Y_i contains O. Hence

$$P_1 = \{[Y_1, Y_2, Y_3] \mid Y_i \in \Sigma_1, \text{ and } (\exists i) O \in Y_i\}.$$

Second output problem $\Pi_2 = \mathrm{re}(\Pi_1)$. We swap roles again, returning to $(3,2)$-biregular trees with the original active/passive partition. Now Σ_2 consists of all non-empty subsets of Σ_1. Let us rename the elements of Σ_1 for brevity:

$$a = \{O\}, \qquad b = \{I\}, \qquad c = \{O, I\}.$$

Then $\Sigma_1 = \{a, b, c\}$ and

$$\Sigma_2 = \{\{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}.$$

Active configurations A_2: an active node outputs $[X_1, X_2]$ with $X_i \in \Sigma_2$. The condition: for every choice $x_1 \in X_1$, $x_2 \in X_2$, we must have $(x_1, x_2) \in P_1$. Recall P_1 consists of all pairs where at least one element contains O. Thus we need to ensure that for every choice of x_1 and x_2, at least one of them contains O. This is only possible if at least one of X_1 or X_2 contains a or c (since those are the subsets that contain O). Hence

$$A_2 = \{[X_1, X_2] \mid X_i \in \Sigma_2, \text{ and } (\exists i) X_i \in \{a, c\}\}.$$

Passive configurations P_2: a passive node outputs $[Y_1, Y_2, Y_3]$ with $Y_i \in \Sigma_2$. The condition: there exist $y_1 \in Y_1$, $y_2 \in Y_2$, $y_3 \in Y_3$ such that $(y_1, y_2, y_3) \in A_1$. Recall A_1 consists of the single pair $[\{I\}, \{O\}]$. Thus we need to be able to pick $y_1 = \{I\}$ and $y_2 = \{O\}$ simultaneously. This is possible iff Y_1 contains b (which is $\{I\}$) and Y_2 contains a (which is $\{O\}$). Hence

$$P_2 = \{[Y_1, Y_2, Y_3] \mid Y_i \in \Sigma_2, \text{ and } b \in Y_1 \text{ and } a \in Y_2\}.$$

Comparing Π_2 with Π, we see that they are essentially the same problem, just with a more complex alphabet. The conditions on the configurations are structurally identical, indicating that we have reached a period-2 point in the round elimination process. Thus, $\mathrm{re}(\mathrm{re}(\Pi_2))$ is essentially the same problem as Π_2, confirming that we have a period-2 point.

11 Hardness of Coloring

11.1 Exercise 11.1 (Randomized 2 coloring)

Prove that solving 2-coloring paths in the randomized PN-model requires $\Omega(n)$ rounds.

Solution

In a 2-coloring, each node outputs a color in $\{0, 1\}$ such that adjacent nodes have different colors. A randomized PN-algorithm may use random bits, but nodes do not have unique identifiers.
Assume there exists a randomized PN-algorithm A that solves 2-coloring in $t(n) = o(n)$ rounds with success probability at least $2/3$ on every path of size n.
Consider a path G with n nodes, where $n \gg t(n)$. Pick an edge $\{u, v\}$ such that both endpoints are far from the ends:

$$\text{dist}(u, \text{endpoints}) > t(n), \quad \text{dist}(v, \text{endpoints}) > t(n).$$

Then the radius-$t(n)$ neighborhoods of u and v are isomorphic as port-numbered networks. Hence, the distributions of their outputs are identical:

$$\Pr[g(u) = 0] = \Pr[g(v) = 0], \quad \Pr[g(u) = 1] = \Pr[g(v) = 1].$$

Let $X = g(u)$ and $Y = g(v)$. Since X and Y have identical distributions, we have:

$$\Pr[X \neq Y] = 1 - \Pr[X = Y].$$

The maximum probability that $X \neq Y$ can be is $1/2$, which occurs when X and Y are independent unbiased bits.
Thus,

$$\Pr[g(u) \neq g(v)] \leq \tfrac{1}{2}.$$

Therefore, for each such edge,

$$\Pr[\text{edge } \{u, v\} \text{ is properly colored}] \leq \tfrac{1}{2}.$$

There are $\Theta(n)$ edges in the middle of the path whose endpoints have isomorphic neighborhoods. Let E' be this set of edges, with $|E'| = \Theta(n)$. For the algorithm to succeed, all edges in E' must be properly colored.
Hence,

$$\Pr[\text{all edges in } E' \text{ are properly colored}] \leq (1/2)^{|E'|}.$$

Since $|E'| = \Theta(n)$, this probability is exponentially small in n.
This contradicts the assumption that the algorithm succeeds with probability at least $2/3$. Hence, no randomized PN-algorithm can solve 2-coloring in $o(n)$ rounds.

11.2 Exercise 11.2 (coloring grids)

Prove that 5-coloring 2-dimensional grids in the deterministic LOCAL model requires $\Omega(\log^* n)$ rounds.

Solution

A 2-dimensional grid G of size $n \times n$ contains a Hamiltonian path P of length n^2 (e.g., a snake that visits every node). If there exists a deterministic LOCAL algorithm A that 5-colors G in $T(n)$ rounds, then the same algorithm, when restricted to the subgraph P, yields a 5-coloring of the path P in at most $T(n)$ rounds (the algorithm may use the whole grid, but its output on P is a proper 5-coloring of P).
It is a classic result (Linial) that any deterministic LOCAL algorithm that properly colors a path with a constant number of colors requires $\Omega(\log^* N)$ rounds, where N is the number of nodes on the path. Here $N = n^2$, so $\log^* N = \log^* n + O(1)$. Hence $T(n)$ must be $\Omega(\log^* n)$.
Therefore, 5-coloring a 2-dimensional grid in the deterministic LOCAL model needs $\Omega(\log^* n)$ rounds.

11.3 Exercise 11.3 (more colors)

Prove that cycles cannot be colored with $O(\log^* n)$ colors in $o(\log^* n)$ rounds in the deterministic LOCAL model.

Solution

Assume for contradiction that there exists a deterministic LOCAL algorithm A that, for every n, colors every n-node cycle with at most $c(n) = O(\log^* n)$ colors and runs in $t(n) = o(\log^* n)$ rounds.
Consider a cycle C_n of length n, and pick an arbitrary edge e. Remove e to obtain a path P_n of n nodes (the endpoints of the path are the former endpoints of e). The coloring produced by A on C_n induces a proper $c(n)$-coloring of P_n (the two endpoints are not adjacent, so the coloring remains proper).
Now we analyze the algorithm on P_n. In the deterministic LOCAL model, after t rounds the output of a node depends only on its t-neighborhood, which on a path is a contiguous segment of length $2t + 1$ (including the node itself). The number of possible color patterns on such a segment is at most $c(n)^{2t+1}$, because each of the $2t + 1$ positions can receive one of at most $c(n)$ colors.
If $t(n) = o(\log^* n)$, then for sufficiently large n we have

$$c(n)^{2t(n)+1} = O(\log^* n)^{o(\log^* n)} = n^{o(1)} < n.$$

Thus the number of distinct color patterns that can appear on segments of length $2t+1$ is strictly smaller than the length of the path P_n. By the pigeonhole principle, there exist two disjoint segments of length $2t + 1$ (at distance at least $2t + 1$ apart) that have identical color patterns. Because the algorithm is deterministic and the neighborhoods are isomorphic, the coloring repeats periodically from that point onward. This periodic repetition forces a conflict at the boundary where the pattern would need to match a different color, contradicting the properness of the coloring (a standard argument in the proof of Linial's lower bound).
Hence no such algorithm A can exist; any deterministic LOCAL algorithm that colors a cycle with $O(\log^* n)$ colors must use $\Omega(\log^* n)$ rounds.

12 Conclusions

12.1 Exercise 12.3 (Randomized Constant Time)

Show that if a locally verifiable problem $\prod$ can be solved in constant time in cycles with a randomized LOCAL-algorithm, then it can be solved in constant time with a deterministic LOCAL-algorithm.

Solution

Let Π be a locally verifiable problem that can be solved in $O(1)$ rounds by a randomized LOCAL-algorithm A on cycles. Let r be the running time of A. Then each node decides its output based only on its radius-r neighborhood and its random bits.
Step 1: Constant success probability on a fixed instance.
Since A runs in constant time, its behavior does not depend on the size of the input. Hence, there exists a cycle N such that the success probability of A on N is some constant $p < 1$.
Step 2: Boosting failure probability.
Construct a new graph N' by taking k disjoint copies of N, where k is arbitrarily large. Since Π is locally verifiable and A runs in r rounds, the executions of A on different copies of N are independent. Therefore, the probability that A succeeds on all copies is

$$p^k.$$

As $k \to \infty$, we have $p^k \to 0$. Hence, the success probability of A on N' can be made arbitrarily small. Thus, a randomized algorithm for Π must succeed with probability at least a fixed constant (e.g., $2/3$) on every input. However, for sufficiently large k, the success probability on N' is less than $2/3$, which is a contradiction.
Step 4: Derandomization.
Hence, it must be that $p = 1$, i.e., A succeeds with probability 1 on every cycle. This means that for each radius-r neighborhood, there exists a choice of random bits that produces a correct output. Since there are only finitely many radius-r neighborhoods in cycles, we can fix these choices deterministically. Thus, we obtain a deterministic LOCAL-algorithm that runs in $O(1)$ rounds and solves Π.
Any locally verifiable problem that can be solved in constant time by a randomized LOCAL-algorithm on cycles can also be solved in constant time by a deterministic LOCAL-algorithm.

12.2 Exercise 12.2 (hardness of weak coloring)

Consider the weak c-coloring problem, as defined in Exercise 12.1.

1. Show that weak 2-coloring can be solved in cycles in $O(\log^* n)$ rounds.
2. Show that weak c-coloring, for any $c = O(1)$, requires $\Omega(\log^* n)$ rounds in cycles.

Solution

(a) Upper bound for weak 2-coloring. We first run Linial's 3-coloring algorithm on the cycle. This algorithm runs in $O(\log^* n)$ rounds and produces a proper 3-coloring of the cycle, i.e., every node receives a color from $\{1, 2, 3\}$ and adjacent nodes have different colors. Afterwards, each node applies the deterministic local mapping $f : \{1, 2, 3\} \to \{0, 1\}$ defined by

$$f(1) = 0, \quad f(2) = 1, \quad f(3) = 1.$$

We claim that the resulting 2-coloring is a valid weak 2-coloring. Consider any node v with color $c(v)$. Its two neighbors u and w have colors different from $c(v)$ (by properness of the 3-coloring). We examine three cases:

- If $c(v) = 1$, then $f(c(v)) = 0$. The neighbors have colors in $\{2, 3\}$, and $f(2) = f(3) = 1$. Hence both neighbors get color 1, which is different from 0, so the condition holds.
- If $c(v) = 2$, then $f(c(v)) = 1$. Its neighbors are in $\{1, 3\}$. $f(1) = 0$, $f(3) = 1$. Thus at least one neighbor (the one with color 1) receives $0 \neq 1$, satisfying the condition.
- If $c(v) = 3$, symmetric to the case $c(v) = 2$.

Thus every node has at least one neighbor of a different color. The whole procedure takes $O(\log^* n)$ rounds, proving the upper bound.

(b) Lower bound for weak c-coloring. Assume, for contradiction, that there exists a deterministic LOCAL algorithm A that solves weak c-coloring on all cycles in $T(n) = o(\log^* n)$ rounds. We apply the round elimination technique (Lemma 11.1 and Lemma 11.2) iteratively. Because the problem is non-trivial (the triple (x, x, x) is forbidden for every $x \in \Sigma$), the same proof as for 3-coloring (Section 11.3.2) shows that after $T(n)$ rounds we obtain a 0-round randomized algorithm for weak c-coloring with local failure probability $q < 1/c$, provided $T(n) \leq \log^* n - O(1)$ (which holds for sufficiently large n).

By Exercise 12.3, any constant-time randomized algorithm for a locally verifiable problem on cycles can be derandomized to a constant-time deterministic algorithm. Hence there exists a 0-round deterministic algorithm for weak c-coloring on cycles. However, a 0-round deterministic algorithm must assign the same color to every node (all nodes are indistinguishable), resulting in a constant coloring. The constant coloring violates the weak coloring condition because for any node both neighbors have the same color as the node itself, i.e., the triple (x, x, x) occurs. This is a contradiction.

Therefore no such algorithm A exists; any deterministic algorithm for weak c-coloring on cycles requires $\Omega(\log^* n)$ rounds.