University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
Z. Parsaeian, G. Schmid

Distributed Graph Algorithms
Sample Solution Exercise Sheet 7

Motivation. The *Sinkless Orientation* problem plays a central role in the theory of distributed graph algorithms in the LOCAL model. One way to argue for its importance is to show that it is *easier* than other fundamental problems: if Sinkless Orientation already requires many rounds, then other problems that reduce to it must be hard as well. Another way is to say that other problems are easy, if we know how to solve Sinkless Orientation fast. We will see one example each.

Δ-coloring in low degree graphs

We compare the complexity of 5-coloring and 4-coloring graphs of maximum degree at most 4, and show how the hardness of 4-coloring can be explained via a reduction to Sinkless Orientation. Notice that one is in essence a $\Delta + 1$ coloring, while the other is Δ coloring.

Exercise 1: 5-Coloring Degree-4 Graphs is Easy

Let $G = (V, E)$ be a graph of maximum degree at most 4.

Show that there exists a deterministic distributed algorithm that computes a proper 5-coloring of G in $O(\log^* n)$ rounds in the LOCAL model.

Sample Solution

We first apply Linial's algorithm to compute a proper $O(\Delta^2)$-coloring of the graph. Since $\Delta \leq 4$, this yields an $O(4^2) = O(1)$-coloring in $O(\log^* n)$ rounds in the LOCAL model.
Next, we iteratively eliminate all colors greater than 5. Consider an iteration corresponding to some color $i > 5$. Each node v with color i has at most 4 neighbors and is therefore adjacent to at most 4 distinct colors. Consequently, at least one color in the set $\{1, 2, 3, 4, 5\}$ is not used by any neighbor of v. Node v can thus safely recolor itself with this unused color.
We claim that the coloring remains proper after every iteration. When a node v recolors itself, it explicitly chooses a color different from all colors used by its neighbors, and hence remains properly colored as long as its neighbors do not recolor simultaneously. Since the initial coloring is proper, no two adjacent nodes share the same color, and therefore no neighbor of v participates in the recoloring step for color i. Thus, no conflicts are introduced.
Each recoloring iteration takes $O(1)$ rounds, as the number of colors involved is constant. Hence, the total running time of the procedure is dominated by the initial coloring step and is therefore $O(\log^* n)$ rounds.

Exercise 2: Why 4-Coloring Degree-4 Graphs is Hard

In contrast to Exercise 1, we now argue that 4-coloring graphs with maximum degree at most 4 is a much harder problem. We do this by relating it to the Sinkless Orientation problem.

To do so, we solve the following two exercises.

Exercise 2.1: From Vertex Coloring to Edge Coloring

Let G be a tree of maximum degree at most 3. Suppose there exists a deterministic distributed algorithm that computes a proper 4-vertex-coloring of any graph of maximum degree at most 4 in T rounds.

Show that this implies the existence of a deterministic $O(T)$-round distributed algorithm that computes a proper 4-edge-coloring of any tree of maximum degree at most 3.

Sample Solution

We use the given algorithm to compute a vertex coloring of the line graph $L(G)$ of G. To this end, we simulate the execution of the algorithm on $L(G)$ within the LOCAL model on G.
Each vertex of $L(G)$ corresponds to an edge of G. We assign the simulation of each edge $e = \{u, v\}$ to the endpoint with the smaller identifier (or at random in randomised models). Two edges of G are adjacent in $L(G)$ if and only if they share a common endpoint in G. Consequently, two simulated vertices of $L(G)$ that are adjacent are handled by two nodes at distance at most 2 in G. It follows that one round of communication in $L(G)$ can be simulated in at most two rounds in the LOCAL model on G.
By definition of the line graph, the maximum degree of $L(G)$ is

$$\Delta_{L(G)} = 2\Delta_G - 2 = 4.$$

A proper vertex coloring of $L(G)$ therefore yields a proper edge coloring of G using 4 colors. Thus, given a T-round algorithm for vertex coloring $L(G)$, we obtain a 4-edge-coloring of G in $2T = O(T)$ rounds.

Exercise 2.2: From Edge Coloring to Sinkless Orientation

Assume there is an a deterministic $O(T)$-round distributed algorithm that computes a proper 4-edge-coloring of any tree of maximum degree at most 3.
Let $G = (V, E)$ be a 2-colored tree of maximum degree at most 3. (So we have the same setup as in the $\Omega(\log n)$ Sinkless Orientation lower bound of the lecture.)

Show that the Sinkless Orientation problem on G can be solved in $O(T)$ rounds.

Sample Solution

We first compute a proper 4-edge-coloring of G using the assumed algorithm. Observe that every node of degree 3 is incident to three edges of pairwise distinct colors.
We now define an orientation of the edges based on this coloring. Each white node applies the following rule: edges of colors 1 and 2 are oriented *towards* the white node, while edges of colors 3 and 4 are oriented *away from* the white node. Since the graph is bipartite, every edge is incident to exactly one white node. Consequently, each edge receives a unique orientation, and all edges are oriented.
We say that a white node is *happy* if it is incident to at least one edge of color 3 or 4, as such an edge is oriented away from the node. If a white node has degree 3, then its three incident edges must all have different colors. As there are only four colors in total, at least one of these edges must have color 3 or 4. Hence, every white node of degree 3 is happy.
An analogous argument applies to black nodes, with the roles of colors $1, 2$ and $3, 4$ exchanged. Thus, every node of degree at least 3 has at least one outgoing edge, and the resulting orientation is sinkless.

Sink-and Sourceless Orientation

In the problem of Sink-and Sourceless Orientation (SSO), we require an orientation of the edges, such that each node of degree at least 3 has both an incoming and an outgoing edge. In other words, no node is a sink (in the same way as in Sinkless Orientation), but also no node is a source (all edges are outgoing). Clearly the problem is at least as hard as Sinkless Orientation (any solution to Sink-and Sourceless Orientation is a solution to Sinkless Orientation). We will now show, that we can solve Sink-and Sourceless Orientation just as fast as Sinkless Orientation.

Exercise 3: SSO is easy for large degrees

Consider the problem SSO_6, where only nodes of degree at least 6 have to output a correct solution.

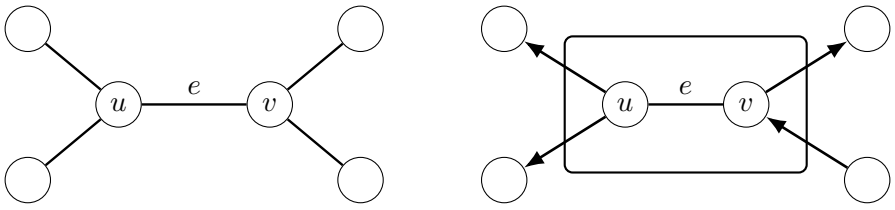
Show that if we are given a T round algorithm for SO, we can solve SSO_6 in T rounds.

Sample Solution

For every node v of degree at least 6, we conceptually split v into two distinct nodes v_1 and v_2. We distribute the edges incident to v evenly between v_1 and v_2, such that each of v_1 and v_2 has degree at least 3. Note that no edge is introduced between v_1 and v_2.
We then simulate the Sinkless Orientation algorithm on the graph obtained by this node-splitting operation. As a result, we obtain an orientation of the edges in which both v_1 and v_2 have at least one outgoing edge.
Next, we reverse the direction of one outgoing edge incident to v_1. After this modification, the original node v has at least one outgoing edge (since v_2 still has an outgoing edge) and at least one incoming edge (due to the flipped edge from v_1). Thus, v satisfies the requirements of the SSO_6 problem.
Since this argument applies to every node of degree at least 6, we obtain a valid solution to the SSO_6 problem. (Note that no edge is flipped twice, since it is only outgoing for one endpoint.)

Dealing with low degree nodes The problem is trivial on nodes of degree 0,1,2 and we have have shown above that we can deal with degree ≥ 6 nodes easily. So the problem are nodes of degree $3, 4$ and 5. Our solution is to *merge* low degree nodes to obtain higher degree nodes. For example take two adjacent nodes u, v of degree 3, by acting as if they were just a single node (so contracting the edge between them), we have created a node of degree 4.

Before contraction **After contraction with SSO solution**



Exercise 4: Merging Nodes

Show that if we have a solution to SSO in the graph, where we treat the nodes u, v as a single node (that now has degree 4), we can always orient the edge $e = \{u, v\}$ to make both u and v happy.

Sample Solution

Assume that we are given a solution to the SSO problem in the graph in which the two adjacent nodes u and v are merged into a single node w of degree 4. By definition of SSO, the node w has at least one incoming and at least one outgoing edge.

We distinguish two cases.

Case 1: The incoming edge of w is incident to u and the outgoing edge of w is incident to v (or vice versa). In this case, we orient the edge $e = \{u, v\}$ from u to v. Then u has an outgoing edge and v has an incoming edge, and both nodes are happy.

Case 2: Both the incoming and the outgoing edge of w are incident to the same node, say u. Then u is already happy. The node v has degree 2 and therefore is incident to at least one edge. We orient the edge $e = \{u, v\}$ such that v receives either an incoming or an outgoing edge, making v happy as well.

Clusters: We now generalise this to more than just contracting a single edge. For any graph $G = (V, E)$, we say that $C \subset V$ is a cluster of V, if

- For any $u, v \in C$, there exists a path from u to v that is entirely contained in C of length $O(1)$.

We say that the degree of C is the number of edges with one endpoint in C and one endpoint in $V \setminus C$. We denote by E_C the set of edges contained in C, that is,

$$E_C = \big\{\{u, v\} \in E \mid u, v \in C\big\}.$$

If our clusters have degree at least 6, then, using Exercise 3, we can act as if it was a degree ≥ 6 node and obtain an orientation of the edges outside the cluster with an incoming and an outgoing edge.

Exercise 5: Clusters with Incoming and Outgoing Edges

Given a cluster C of G, assume that all edges outside of C are oriented, and that this orientation constitutes a valid solution for SSO, for the graph in which the cluster C is contracted to a single node. In particular, all edges incident to C are oriented, with the guarantee that at least one such edge is oriented towards C and at least one is oriented away from C.
Note that E_C are the only edges in the graph that are not yet oriented.

Show that there is an orientation of E_C, such that all nodes in C have an incoming and an outgoing edge.

Hint: Start with a path (fully contained in C) that connects the incoming edge e_{inc} to the outgoing edge e_{out}.

Sample Solution

We first consistently orient the edges in a direct path P from e_{inc} to e_{out}, if both are at the same node, we do nothing and P just consists of that single node. Note that by our definition of clusters, such a path P must exist and after orienting the path, all nodes in the path are happy.
It remains to orient the edges incident to nodes in $C \setminus P$. For this, consider the distance of a node $v \in C$ to the path P, defined as the length of the shortest path from v to any node in P that is fully contained in C.
To fix the directions, we have each node $u \notin P$ of the cluster, pick one shortest path towards P. As a result each node u has a corresponding "upwards edge" e_u that goes towards P. Notice that each node u picks a unique e_u, that is not picked by any other node in the cluster.
We now work our way from the borders of the cluster towards P. Take a node v at the border of a cluster, since v is a border node it must have at least one edge with a node outside of the cluster. If this edge is oriented away from v, v orients e_v towards itself. If instead the edge is oriented towards v, we orient the edge e_v away from v. Either way, v will be happy.

Now take an unprocessed node v that is adjacent to at least one already processed node u. If the edge $\{u, v\} = e_u$, then that edge is already oriented, if not we orient it arbitrarily. Thus v has at least one oriented edge that is not e_v, so we may orient e_v in the same way as for the vorder nodes, to make v happy.
We can use the same argument to obtain an orientation of all of the e_v edges (and a few other previously unoriented edges) to make every node in $C \setminus P$ happy. Finally, we orient all unoriented edges in C arbitrarily.

If we cannot obtain such a cluster with a large enough degree, we will instead find a cluster with a short cycle.

Exercise 6: Clusters with Cycles

1. Let $C \subset V$ be some cluster of G, that contains a cycle that is completely contained in C.
2. Assume that all edges outside of C are oriented (in an completely arbitrary way).

Show that there is an orientation of E_C, such that all nodes in C have an incoming and an outgoing edge.

Sample Solution

We do the same argument as in the previous task, but instead of starting from a consistently oriented path P, we instead take the cycle K (completely contained in C) and consistently orient it. Now all nodes of K are already happy and so we may use the edges twoards K to make the remaining nodes happy in the same way as in the previous task.

Exercise 7: Getting good clusters

We now have to obtain suitable clusters for all nodes with degree that is too small.
Let $G = (V, E)$ be an arbitrary graph and let $G' = (V', E')$ be the subgraph of G induced by all nodes of degree 3, 4, or 5 in G.
Your task is to design an $O(\log^* n)$ algorithm that partitions the node set V' into sets $C_1, \ldots, C_k, I$ such that

1. $V' = C_1 \cup \ldots \cup C_k \cup I$ and $C_1, \ldots, C_k, I$ are mutually disjoint.
2. The set I contains all isolated nodes.
3. Each set C_i is a cluster that either has degree ≥ 6, or contains a cycle.

Give a description of such an algorithm and argue that it always produces valid clusters of G'.

Hint: You may try to first solve a different $O(\log^* n)$ problem on G' and then use that to solve the task.
Hint: If you can't increase the degree to ≥ 6 immediately, increase it to just ≥ 4 and try running the procedure again.

Sample Solution

We first compute a 6-coloring of G' in $O(\log^* n)$ rounds. We then iterate through the color classes. Each node v with color i that is not yet part of a cluster joins a cluster with an arbitrary neighbor u. More precisely, u may have multiple nodes $v_1, v_2, \ldots,$ that want to join u. If u is already in a cluster C_u, then all of the $v_1, \ldots$ simply join C_u. If instead, u is not yet in a cluster, we create a new cluster containing just $u, v_1, \ldots$.

At the end, each node v that is not isolated will be in some cluster C_v. We argue that either the degree of $C_v \geq 4$, or C_v contains a cycle. Observe that the sum over the degree of nodes in C_v is at least $3|C_v|$, but each edge entirely contained in the cluster is counted twice. If C_v contains no cycles, then the number of edges between nodes in C_v is exactly $|C_v| - 1$. Consequently, the number of edges with only one endpoint in C_v is at least

$$3|C_v| - 2(|C_v| - 1) = |C_v| + 2 \geq 4.$$

As a result, if we contract each cluster into a sinlge node, the resulting graph G'' contains no nodes of degree smaller than 4. So by running the procedure a second time, we obtain clusters with degree $4|C_v| - 2(|C_v| - 1) = 2|C_v| + 2 \geq 6$ as desired.

Exercise 8: Computing SSO

We now have everything we need to solve SSO.
Combine the results from Exercises 3,5,6,7 to solve the problem of SSO in $O(\log^* n + T_{SO})$, where T_{SO} is the time to solve Sinkless Orientation.
Hint 1: You still have to deal with the isolated nodes, that are not put into some cluster in Exercise 7. What can you say about nodes that are isolated in G'?

Sample Solution

First compute a clustering as in Exercise 7. Each node that is isolated in G' either has at least on neigbhor of degree at least 6, or only neighbors of degree ≤ 2. If an isolated node as at least one large degree neighbor, it simply joins a cluster with that high degree neighbor, the resulting cluster must have degree ≥ 6. If all neighbors have degree at most 2, a solution can be trivially obtained by orienting an arbitrary edge towards the isolated node and another away.
We then use Exercise 3 to solve SSO_6 on the graph, on which all clusters are contracted. By Exercises 5,6 we can turn that solution into a solution of the original graph, so that all nodes that were in clusters are also happy. We finally orient any remaining unoriented edges (of small degree nodes).