Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

FS 2022

Prof. R. Wattenhofer
Florian Grötschla
Andrei Constantinescu

Principles of Distributed Computing
Exercise 13: Sample Solution

1 GNNs for Algorithmic Problems

a) We want to update the states for every node $v \in V$ as follows:

$$h_v^{(t)} = \min\left(\{h_w^{(t-1)} + 1 \mid w \in N(v)\} \cup \{h_v^{(t-1)}\}\right)$$

This is a somewhat simplified version of the Bellman-Ford algorithm for unweighted graphs. To do so, we choose Aggregate to be the min function and Update to simply add 1 to the aggregated value.

$$\text{AGGREGATE}\left(\{\{{h_u}^{(t-1)} \mid u \in N(v)\}\}\right) = \min\left(\{\{{h_u}^{(t-1)} + 1 \mid u \in N(v)\}\}\right)$$

and

$$\text{UPDATE}\left({h_v}^{(t-1)}, {a_v}^{(t)}\right) = \min(a_v^{(t)}, h_v^{(t-1)}).$$

b) Consider a cycle where every node has the same initial state. As every node gets the exact same inputs for its state update function in every iteration k, it follows that for any k all nodes will be in the same state after k steps. This means that the GNN can either put all nodes or no nodes in the vertex cover, in both cases it is not a minimal vertex cover.

c) Consider a tree of size 2. The two leafs will always be in the same color class, so either both or none will end up in the vertex color. Similar to the argument in the previous question, this makes it impossible for the GNN to compute a minimal vertex cover.

d) The algorithmic idea that we use is to identify leaves in every iteration of the GNN and add their parents to the minimal vertex cover. We then remove the parents and leaves from the tree and repeat the same action. To encode this information we use the following representations for the states:

0 — no decision has been taken yet, node is indecisive;

1 — indecisive leaf;

2 — indecisive non-leaf;

3 — node is in the vertex cover;

4 — node is not in the vertex cover and does not take part in the computation anymore.

For now, we set $h_v^{(0)} = 0$ for all nodes $v \in V$ and choose:

$$\textsc{aggregate}\,(M) = \begin{cases} 1, & \text{if there are at most one 0 and at most one 2 in } M \\ 2, & \text{otherwise} \end{cases}$$

where $M = \{\{h_u{}^{(t-1)} \mid u \in N(v)\}\}$ and

$$\textsc{update}\,(\,h_v{}^{(t-1)},\, a_v{}^{(t)}\,) = \begin{cases} 0, & \text{if } h_v^{(t-1)} \in \{1,2\} \text{ and } a_v^{(t)} = 2 \\ 1, & \text{if } h_v^{(t-1)} = 0 \text{ and } a_v^{(t)} = 1 \\ 2, & \text{if } h_v^{(t-1)} = 0 \text{ and } a_v^{(t)} = 2 \\ 3, & \text{if } h_v^{(t-1)} = 3 \text{ or } h_v^{(t-1)} = 2 \text{ and } a_v^{(t)} = 1 \\ 4, & \text{if } h_v^{(t-1)} \in \{1,4\} \end{cases}$$

After enough iterations all nodes will be in either state 3 or 4, representing our minimal vertex cover. Until here, we did not use the color information and assumed that all initial states are 0. As we showed previously, this algorithm can not be correct. It breaks when being executed on a connected component of size 2. In this case we want to add exactly one of the two nodes to the vertex cover. We can identify this case and use the two-coloring as a tie-break as we know that both nodes have to be colored differently.

2 The Weisfeiler-Lehman Test for Trees

a) Consider a path of n vertices. Then, WL takes $\approx n/2$ rounds to finish.

b) Consider graphs $G_1 =$ the chain $1-2-3-4-5$ and $G_2 =$ the cycle $a-b-c-d-a$. Taking $t = 2$, we have that $s_3^{(2)} = s_a^{(2)}$, yet the 2-hop neighborhood of 3 is a path, while the 2-hop neighborhood of a is a cycle, so they are not isomorphic.

c) Let G be the path $1-2-3$. Then, the rooted trees $(G,1)$ and $(G,2)$ have the required property.

d) This is the more difficult part of the question, and solving it requires a bit of insight. To begin, note that $\textsc{relabel}$ is injective, so it is invertible on its codomain; let $\textsc{relabel}^{-1}$ be its inverse. What does this give us? Recall that for any node $w \in V$ and $t > 0$:

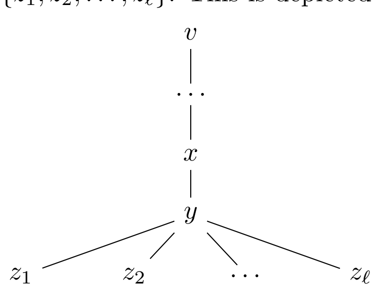
$$s_w^{(t)} = \textsc{relabel}(s_w^{(t-1)}, \{\{s_u^{(t-1)} \mid u \in N(w)\}\})$$

so, by taking inverses:

$$\textsc{relabel}^{-1}(s_w^{(t)}) = (s_w^{(t-1)}, \{\{s_u^{(t-1)} \mid u \in N(w)\}\})$$

This means that whenever $s_w^{(t)}$ is known, the value $s_w^{(t-1)}$ and the multiset $\{\{s_u^{(t-1)} \mid u \in N(w)\}\})$ can be uniquely determined/recovered.

The rest of the solution relies on an observation, which we outline in what follows. Assume we root the whole tree in node v, and consider two nodes $x, y \in V$ in the tree, such that x is the parent of y with respect to the root v. Furthermore, assume that $N(y) = \{x\} \cup \{z_1, z_2, \ldots, z_\ell\}$. This is depicted below.



Now, assume that we are know the values $s_x^{(k)}$ and $s_y^{(k-1)}$ for some $k \geq 2$. If so, by taking the inverse on $s_x^{(k)}$, we can recover the value $s_x^{(k-1)}$. By taking the inverse again, on $s_x^{(k-1)}$, we can also recover $s_x^{(k-2)}$. Moreover, by taking the inverse on $s_y^{(k-1)}$, we can recover the multiset $\{\{s_u^{(k-2)} \mid u \in N(y)\}\}$. Now, since $N(y) = \{x\} \cup \{z_1, z_2, \ldots, z_\ell\}$, we have that

$$\{\{s_u^{(k-2)} \mid u \in N(y)\}\} = \{\{s_x^{(k-2)}\}\} \cup \{\{s_{z_i}^{(k-2)} \mid 1 \leq i \leq \ell\}\}$$

from which we get that:

$$\{\{s_{z_i}^{(k-2)} \mid 1 \leq i \leq \ell\}\} = \{\{s_u^{(k-2)} \mid u \in N(y)\}\} \setminus \{\{s_x^{(k-2)}\}\}$$

Since we were able to uniquely recover the value $s_x^{(k-2)}$, this means that, by taking the set difference above, we are also able to uniquely recover $\{\{s_{z_i}^{(k-2)} \mid 1 \leq i \leq \ell\}\}$. Why is this useful? Well, starting with $s_x^{(k)}$ and $s_y^{(k-1)}$ we were able to uniquely determine the multiset $\{\{s_u^{(k-2)} \mid u \text{ is a child of } y\}\}$. Intuitively, this reasoning can then be repeated for each of $z_1, \ldots z_\ell$ with parent y to find the $s^{(k-3)}$ values of their children, and so on. Observe how we used node indices $v, x, y, z_1, \ldots, z_\ell \in V$ in explaining the approach, but, in reality, all operations performed do not require knowledge of the node indices, only the assumption that they exist being enough. We make this formal in Algorithm 1. The algorithm is recursive, it takes as arguments k and two values s_p and s, which, as a precondition, are required to correspond to some values $s_x^{(k)}, s_y^{(k-1)}$ in the original tree, such that x is the parent of y when the tree is rooted in v. The return value is an isomorphic copy of the subtree of y with respect to root v, truncated to depth $k-1$. The case where $y = v$ is special, since v does not have a parent, so in the implementation we use the special value $\perp$ to signify a missing value. In order to recover the t-hop neighborhood of v, the initial call has to be RecoverSubtree$(t+1, \perp, s_v^{(t)})$. To help the exposition, in comments on the right hand side we wrote the algorithm again, only this time with node indices. A fully formal proof of correctness would, in fact, carefully leverage the connection with the version with node identities through code invariants. The details are technical and out of the scope of this exercise.

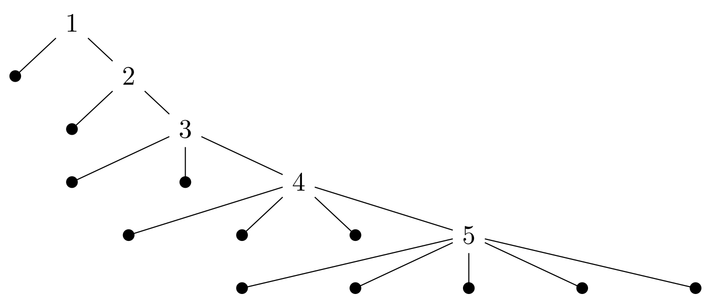
Algorithm 1 Recovering a subtree, truncated to depth $k-1$.

procedure RecoverSubtree(k, s_p, s) $\quad\triangleright$ RecoverSubtree$(k, x, y, s_x^{(k)}, s_y^{(k-1)})$
$n \leftarrow$ **new** node.
$T \leftarrow$ rooted tree consisting of just node n.
if $k = 1$ **then**
  return T
end if
$(-, S_c) \leftarrow \text{RELABEL}^{-1}(s)$ $\quad\triangleright$ $(-, \{\{s_u^{(k-2)} \mid u \in N(y)\}\}) \leftarrow \text{RELABEL}^{-1}(s_y^{(k-1)})$
if $s_p \neq \perp$ **then** $\quad\triangleright$ **if** $y \neq v$ **then**
  $(s_p^{-1}, -) \leftarrow \text{RELABEL}^{-1}(s_p)$ $\quad\triangleright$ $(s_x^{(k-1)}, -) \leftarrow \text{RELABEL}^{-1}(s_x^{(k)})$
  $(s_p^{-2}, -) \leftarrow \text{RELABEL}^{-1}(s_p^{-1})$ $\quad\triangleright$ $(s_x^{(k-2)}, -) \leftarrow \text{RELABEL}^{-1}(s_x^{(k-1)})$
  $S_c \leftarrow S_c \setminus \{\{s_p^{-2}\}\}$ $\quad\triangleright$ $\{\{s_{z_i}^{(k-2)} \mid 1 \leq i \leq \ell\}\} = \{\{s_u^{(k-2)} \mid u \in N(y)\}\} \setminus \{\{s_x^{(k-2)}\}\}$
end if
for $s_c \in S_c$ **do** $\quad\triangleright$ **for** $i = 1$ to ℓ **do**
  $T' \leftarrow$ RecoverSubtree$(k-1, s, s_c)$ $\quad\triangleright$ $T_i \leftarrow$ RecoverSubtree$(k-1, y, z_i, s_y^{(k-1)}, s_{z_i}^{(k-2)})$
  Make n the parent of the root of T'. $\quad\triangleright$ Make n the parent of the root of T_i.
end for
return T

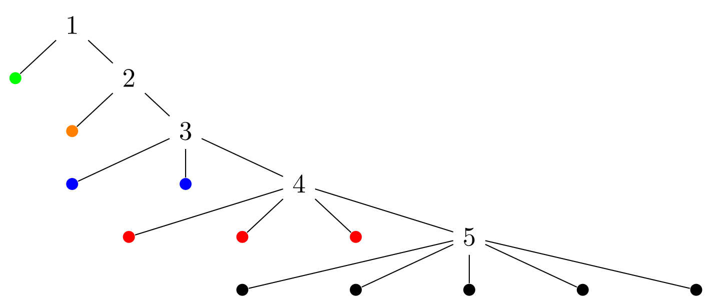
As a side note, an efficient implementation of this algorithm requires that RELABEL^{-1} can be computed efficiently (which is true for some complying RELABEL functions), but this is not required for our proof, since we only need to prove that the neighborhood is uniquely determined, not that it can be computed efficiently.

e) By part **d)**, for any $v \in V$ the value $s_v^{(t)}$ uniquely determines the t-hop neighborhood of v up to isomorphism. If $t \geq \Delta$, then this neighborhood will be the whole tree, no matter which $v \in V$ was used. If, instead, $t \geq \Delta/2$, then for some nodes $v \in V$ the t-hop neighborhood might not be the whole tree. However, if v is the midpoint of a diameter of G (if there are two midpoints either will work), then the t-hop neighborhood will be the whole tree. As a side note, such a node can be identified without knowledge of G by finding the t-hop neighborhood of all nodes and taking the node for which this neighborhood consists of the most nodes.

f) We will show a stronger statement: there are infinitely many trees for which WL finishes in $O(1)$ rounds, while their diameter is $\omega(1)$. Consider the following tree. Start with a chain of $k \geq 5$ nodes: $1 - 2 - \ldots - k$ and add extra leaves to it according to the list $w = [1, 1, 2, 3, 4, \ldots, k-3, k-2, k]$: for each node $1 \leq v \leq k$ link $w[k]$ additional leaves to v. For $k = 5$, this construction would look as follows:



The diameter $\Delta = k + 2$ is given by any path of the form $\bullet - 1 - 2 - \ldots - k - \bullet$. In the first round, WL assigns labels based on the degrees, so let us observe them: all $\bullet$ vertices are leaves, so they have degree 1, while for $1 \leq v \leq k$ the degree of v is $v + 1$. This means that after one iteration the only vertices to get equal labels will be the $\bullet$ vertices. After a second iteration, the $\bullet$ vertices will further partition themselves based on the label of their father. Pictorially, the new partition after two iterations looks as follows for $k = 5$:



In the third iteration the partition will not change, so WL finishes in two iterations, or, more precisely for our implementation, it finishes at $t = 3 < \Delta/2$.

g) Note that, by taking $\textsc{relabel}^{-1}$, we can construct a mapping $f : \{s_v^{(k)} \mid v \in V\} \to \{s_v^{(k-1)} \mid v \in V\}$, mapping $s_v^{(k)} \mapsto s_v^{(k-1)}$. Clearly, f is surjective, by definition. To show f is also injective, assume $v, v' \in V$ are such that $s_v^{(k)} \neq s_{v'}^{(k)}$, and yet $s_v^{(k-1)} = s_{v'}^{(k-1)}$. This means that the color partition has been refined at iteration $k - 1$. This can not be the case, since we assumed that $k \geq t$, and t is such that $(s_v^{(t)})_{v \in V}$ and $(s_v^{(t-1)})_{v \in V}$ induce the same color partition, so $(s_v^{(k)})_{v \in V}$ and $(s_v^{(k-1)})_{v \in V}$ also do so. These being said, f is a bijection between $\{s_v^{(k)} \mid v \in V\}$ and $\{s_v^{(k-1)} \mid v \in V\}$.

Knowing this, consider an arbitrary value $s_v^{(k)}$ and take $\textsc{relabel}^{-1}(s_v^{(k)})$. This gives us $s_v^{(k-1)}$ and the multiset $\{\{s_u^{(k-1)} \mid u \in N(v)\}\}$. By applying f^{-1} to all elements of this multiset we get the multiset $\{\{s_u^{(k)} \mid u \in N(v)\}\}$. If we now compute $\textsc{relabel}(s_v^{(k)}, \{\{s_u^{(k)} \mid u \in N(v)\}\})$ this will by definition give us $s_v^{(k+1)}$. Since the value $s_v^{(k)}$ considered was arbitrary, by doing so for all such values in $\{\{s_v^{(k)} \mid v \in V\}\}$ we can exactly determine the multiset $\{\{s_v^{(k+1)} \mid v \in V\}\}$, as required.

Note how, again, our argument hinged on the fact that RELABEL can be inverted. Taking this inverse does not have to be computationally tractable in order for the proof to work. Moreover, note how explicit knowledge of t is not required for our argument.

h) Consider a single tree $G(V, E)$, and let $\{\{s_v^{(t)} \mid v \in V\}\}$ be the values returned by WL when run on G. For brevity, let $n = |V|$ and introduce the notation $S_k = \{\{s_v^{(k)} \mid v \in V\}\}$ for $k \geq t$. As such, S_t is the return value of WL. By repeatedly using part **g)**, we can uniquely determine $S_{t+1}, S_{t+2}, \ldots, S_{n-1}$. Now, since we know S_{n-1} and it holds that $n - 1 \geq \Delta$, by part **e)** any value $s_v^{(n-1)} \in S_{n-1}$ is enough to completely determine the tree G up to isomorphism. From here it follows that WL is a complete isomorphism test for the class of trees, as required.

However, there is a subtlety still left to prove: here we made our argument assuming that the value of t is known. Therefore, we still need to do a preprocessing step which computes t before beginning with the rest of the argument. If we make the simplyfing assumption that 0 is not in the image of RELABEL, then this can be done by just taking any $s_v^{(t)} \in S_t$ and taking inverses $s_v^{(t)} \to s_v^{(t-1)} \to \ldots$ until $s_v^{(0)} = 0$ is reached, counting the number of steps required. Without this assumption, it might be that 0 is falsely reached before taking the inverse t times, so one has to work slightly harder. In particular, we will start with the whole set $S_t = \{\{s_v^{(t)} \mid v \in V\}\}$ and invert all elements in it simultaneously: $S_t \to S_{t-1} \to \ldots$, stopping when we reach $S_0 = \{\{0\}\} \times n$. We will now show that $S_i \neq \{\{0\}\} \times n$ for $1 \leq i \leq t$. To do so, assume for a contradiction that $S_i = S_0$ for some $1 \leq i \leq t$. This can only be the case if the partition did not refine in the first step: $S_0 = S_1$, also meaning that $t = 1$. In general, the partition does not refine in the first step of WL if and only if the graph is regular. Since G is a tree, this means that $n \leq 2$. However, for $n \leq 2$ WL can trivially distinguish trees of size n anyway (since there are only two such trees: one with $n = 1$ and one with $n = 2$).