1 Proof of NP-complete

To prove that the "Node-Labelling Decision Problem" is NP-complete, we need to show two things:

1. The problem is in NP:

   Given a certificate, which is a labeling of the nodes in the graph, we can easily verify in polynomial time whether the labeling is valid or not.

   Specifically, given an instance of the Node-Labelling Decision Problem with graph $G = (V, E)$, label set $K = 0, 1, \ldots, k-1$, and non-negative integer R, suppose we have a labeling function $c : V \to K$. To verify that this labeling is valid and satisfies the requirement that $|C(v, R)| = k$ for every vertex $v \in V$, we can perform the following steps:

   (a) Check if every label in K is assigned to at least one node in G. This can be done by iterating through all vertices $v \in V$ and maintaining a set of encountered labels. This step takes $O(|V|)$ time.

   (b) For each vertex $v \in V$, calculate the set $N(v, R)$, which contains all nodes within R hops from v. This can be done using a Breadth-First Search (BFS) starting from v, and stopping once we reach depth R. Since BFS has a time complexity of $O(|V| + |E|)$, this step takes $O(|V|(|V| + |E|))$ time in total for all vertices.

   (c) For each vertex $v \in V$, calculate the set $C(v, R)$, which contains the distinct labels within R hops from v. This can be done by iterating through $N(v, R)$ and maintaining a set of encountered labels. This step takes $O(|V|^2)$ time in total for all vertices.

   (d) Check if $|C(v, R)| = k$ for every vertex $v \in V$. This step takes $O(|V|)$ time.

   The overall time complexity for verifying the solution is $O(|V|^2 + |V|(|V| + |E|))$. Since this is polynomial in the size of the input, we conclude that the Node-Labelling Decision Problem is in NP.

2. The problem is NP-hard:

   We will show that the problem is NP-hard by reducing the 3-coloring problem (which is known to be NP-complete) to the Node-Labelling Decision Problem. Suppose we have an instance of the 3-coloring problem, which is an undirected graph $G = (V, E)$, and we want to determine if there exists a valid 3-coloring of the graph.

   Now we will construct a new instance $G' = (V', E')$ of the Node-Labelling Decision Problem as follows:

   (a) Initially, let $V' = V$. (This step takes $O(|V|)$ time to create the set V'.)

(b) For each edge $(u, v) \in E$, add a new vertex $w_{(u,v)}$ to V' and edges $(u, w_{(u,v)}), (v, w_{(u,v)})$ to E'. (This step takes $O(|E|)$ time to process all edges and create the new vertices and edges.)

(c) For each vertex u in V, add two new vertices u_1, u_2 to V', and add edges (u, u_1), (u, u_2), (u_1, u_2) to E'. (This step takes $O(|V|)$ time to process all vertices and create the new vertices and edges.)

(d) Set the set of labels $K = 0, 1, 2$, and set $R = 1$. (This step takes $O(1)$ time.)

The total time complexity of the reduction is $O(|V| + |E|)$, which is polynomial with respect to the size of the input graph $G = (V, E)$. Therefore, the reduction from the 3-coloring problem to the Node-Labelling Decision Problem is in polynomial time.

Now we will prove that G has a valid 3-coloring if and only if there exists a valid labeling for the Node-Labelling Decision Problem with the constructed G', K, and R.

($\Rightarrow$) Let $c : V \to K$ be a valid 3-coloring of G. We will extend this coloring to the vertices of G' as follows:

- For each vertex $v \in V$, we assign $c'(v) = c(v)$.
- For each edge $(u, v) \in E$, we assign $c'(w_{(u,v)}) = 3 - c(u) - c(v)$, which is the remaining color not used by u or v.
- For each vertex $u \in V$, we assign $c'(u_1) = (c(u) + 1) \mod 3$ and $c'(u_2) = (c(u) + 2) \mod 3$.

Now, we will show that this labeling c' is a valid solution for the Node-Labelling Decision Problem with G', K, and R.

- First, all labels in K are used in G', so the labeling is valid.
- Second, we need to show that for each vertex $v \in V'$, we have $|C(v, R)| = 3$. Since $R = 1$, we only need to consider the 1-hop neighborhood of each vertex.
  - For each vertex $v \in V$, its 1-hop neighbors are v_1, v_2, and all the $w_{(u,v)}$ where $(u, v) \in E$. Since $c'(v_1) \neq c'(v_2)$ and $c'(v) \neq c'(v_1), c'(v_2)$, we have $|C(v, 1)| = 3$.
  - For each vertex $w_{(u,v)}$, its 1-hop neighbors are u and v. Since $c'(u) \neq c'(v)$ and $c'(w_{(u,v)}) \neq c'(u), c'(v)$, we have $|C(w_{(u,v)}, 1)| = 3$.
  - For each vertex u_1, its 1-hop neighbors are u and u_2. Since $c'(u) \neq c'(u_2)$ and $c'(u_1) \neq c'(u), c'(u_2)$, we have $|C(u_1, 1)| = 3$.
  - Similarly, for each vertex u_2, we have $|C(u_2, 1)| = 3$.

Thus, if G has a valid 3-coloring, there exists a valid labeling for the Node-Labelling Decision Problem with the constructed G', K, and R.

($\Leftarrow$) Now, let $c' : V' \to K$ be a valid labeling for the Node-Labelling Decision Problem with G', K, and R. We will use this labeling to construct a valid 3-coloring for G.

Define a coloring $c : V \to K$ such that for each vertex $v \in V$, we assign $c(v) = c'(v)$. We will show that c is a valid 3-coloring for G.

We need to show that for each edge $(u, v) \in E$, we have $c(u) \neq c(v)$. Recall that we added a vertex $w_{(u,v)}$ in G', and edges $(u, w_{(u,v)})$, $(v, w_{(u,v)})$ in E'. Since c' is a valid labeling for the Node-Labelling Decision Problem with $R = 1$, we have $|C(w_{(u,v)}, 1)| = 3$. Therefore, $c'(w_{(u,v)})$, $c'(u)$ and $c'(v)$ must be the three distinct labels. This implies that $c(u) = c'(u) \neq c'(v) = c(v)$.

Thus, if there exists a valid labeling for the Node-Labelling Decision Problem with the constructed G', K, and R, then G has a valid 3-coloring.

In summary, we have shown that G has a valid 3-coloring if and only if there exists a valid labeling for the Node-Labelling Decision Problem with the constructed G', K, and R. This reduction can be done in polynomial time. Hence, the Node-Labelling Decision Problem is NP-hard.

Since we have shown that the Node-Labelling Decision Problem is both in NP and NP-hard, we conclude that the problem is NP-complete.

2 Main idea of algorithm

First calculate its $m(u)$ for each node u, and then try to distribute all k labels in its $m(u)$-hop neighbors. The specific method is: count the labels that have not yet appeared in the $m(u)$-hop neighborhood, and then label the neighbors that have not been labeled with labels that have not yet appeared.

If it is found that after all neighbors are labeled, k labels still cannot be distributed in the $m(u)$-hop neighborhood, it means that some labels are repeated in its neighbors. Then we will check these repeated labels, and try to modify the labels of those nodes that have been marked with labels that have appeared many times to labels that have not yet appeared.

In order to avoid the algorithm running time becoming exponential or falling into an infinite loop, we set a step size. In particular, we check that the neighbor range of each node u is within $m(u) + step$. Therefore, when the *step* has not exceeded 1, we still only check the $m(u)$-hop neighbors, aiming to make $r(u)/m(u) = 1$ for each node.

As the step increases, we will allow distribution of these k labels within $(m(u) + 1)$-hop, $(m(u) + 2)$-hop, ..., neighbors. In this way, we guarantee that the algorithm will not search for labels in exponential time, and ensure that the algorithm will return a solution (which may be the optimal solution or a suboptimal solution) in polynomial time.

3 Pseudo code

Algorithm 1 Greedy Labeling Algorithm

```
procedure GreedyLabeling(tree, k, m_values)
    nodes ← sorted(range(len(tree)), key=lambda x: m_values[x])
    labels ← [None] * len(tree)
    step ← 0
    while True do
        for node in nodes do
            if labels[node] = None then
                available_labels ← set(range(k))
                distances ← bfs(tree, node)
                neighbors ← [x if distances[x] ≤ m_values[node]]
                for neighbor in neighbors do
                    if labels[neighbor] ∈ available_labels then
                        available_labels −= {labels[neighbor]}
                    end if
                end for
                if available_labels then
                    for neighbor in neighbors do
                        if labels[neighbor] = None then
                            labels[neighbor] ← available_labels.pop()
                        end if
                    end for
                end if
                unique_labels ← set()
                if available_labels then
                    for neighbor in neighbors do
                        if labels[neighbor] ∈ unique_labels then
                            labels[neighbor] ← available_labels.pop()
                        end if
                        unique_labels ← unique_labels ∪ {labels[neighbor]}
                    end for
                end if
            end if
        end for
        count ← 0
        for node in nodes do
            missing_labels ← set(range(k))
            distances ← bfs(tree, node)
            neighbors ← [x if distances[x] ≤ m_values[node] + step]
            for neighbor in neighbors do
                if labels[neighbor] ∈ missing_labels then
                    missing_labels −= {labels[neighbor]}
                end if
            end for
            if ¬missing_labels then
                count ← count + 1
            end if
        end for
        if count = len(nodes) then
            break
        else
            step ← step + 0.001
        end if
    end while
    return labels
end procedure
```

4 Proof of proximity ratio

Theorem 1. *The given Greedy Labeling Algorithm has a proximity ratio of* $\rho = 2$.

Proof. Let's consider the algorithm's process of assigning labels. The algorithm iterates through the nodes sorted by their $m(v)$ values and assigns labels according to the available labels in the neighborhood of radius $m(v)$ or the neighborhood of radius $(m(v) + \text{step})$. During this process, the algorithm assigns labels to nodes that have not been assigned yet.

Let's analyze the worst-case scenario. In the worst case, when the algorithm assigns labels to nodes, there will be one label missing from the neighborhood of radius $m(v)$. In this case, the algorithm will look into the neighborhood of radius $(m(v) + \text{step})$ and continue assigning labels. In the worst case, there will still be one label missing from the neighborhood of radius $(m(v) + \text{step})$. Therefore, the maximum value of $r(v)$ will be at most $(m(v) + \text{step})$.

Now, let's consider the lower bound of $r(v)$, which is $m(v)$. Since $r(v)$ is at most $(m(v) + \text{step})$, the ratio $\frac{r(v)}{m(v)}$ is bounded by 2 in the worst case. In other words, $\frac{r(v)}{m(v)} \leq 2$.

Hence, the given Greedy Labeling Algorithm has a proximity ratio of $\rho = 2$. □

5 Analysis of Time Complexity

- Firstly we need do BFS from each node, to get the distances between it and other nodes, which takes $O(|V| * (|V| + |E|))$, that is, $O(|V|^2)$ since $|E| = |V| - 1$.
- Computing m value of each node costs $O(|V|^2)$, since we need check the *distances* array of each node.
- Sorting nodes by the m values costs $O(|V| \log |V|)$.
- For each node, we try to distribute all k labels within its neighbors, which takes $O(|V|)$ since we need traverse its neighbors. There are $|V|$ nodes, thus this first for loop in while loop takes at most $O(|V|^2)$.
- The second for loop in while loop takes at most $O(|V|^2)$ too, since it traverses neighbors to check whether all k labels are distributed within $m + step$ hop.
- Therefore, every time we execute the content of the while loop body, it takes $O(|V|^2)$.
- Since *step* will grow by 0.001 each time we execute the content of the while loop body, after $1000 * |V|$ times, in the second for loop of while loop, we will check whether all k labels are distributed within $m + |V|$ hop, which

is definitely true for each node. Thus, after at most $1000 * |V|$ times we execute the content of the while loop body, we will get $count = |V|$, and quit the while loop.

Overall, the total time complexity is $O(1000|V|^3)$. And if the increment of *step* is set to c (c is a constant), the time complexity will become $O(\frac{1}{c}|V|^3)$.