

# Project of Algorithms on Node Labeling

Kevin Lei

August 2, 2024

## 1 Introduction

In this project we discuss the “Node Labeling Problem”, in which we attempt to label the nodes of a graph with unique labels from a set of labels. We have the following definitions:

- Let  $G = (V, E)$  be an undirected graph.
- Let  $d(u, v)$  be the distance between nodes  $u$  and  $v$ .
- For all nodes  $v \in V$ , let  $N(v, h) \subseteq V$  be the set of nodes that are at most  $h$  hops away from  $v$ .
- Let  $K = \{0, 1, \dots, k-1\}$  be the set of  $k$  integers, where  $k \leq |V|$ .
- For all  $v \in V$ , let  $c(v) \in K$  be the label of node  $v$ , where different nodes may have the same label.
- Let  $C(v, h)$  be the set of labels of nodes in  $N(v, h)$ .
- A labeling of the nodes is *valid* if every label in  $K$  is used at least once.
- Let  $r(v)$  be the smallest integer such that the node  $v$  has all the labels in  $K$  in  $N(v, r(v))$ .
- Let  $m(v)$  be the smallest integer such that the node  $v$  has at least  $k$  nodes in  $N(v, m(v))$ .

Formally, our relevant sets and values can be defined as follows:

$$\begin{aligned} N(v, h) &\triangleq \{u \in V \mid d(u, v) \leq h\} \\ C(v, h) &\triangleq \{c(u) \mid u \in N(v, h)\} \\ r(v) &\triangleq \min\{h \mid |C(v, h)| = k\} \\ m(v) &\triangleq \min\{h \mid |N(v, h)| \geq k\}. \end{aligned}$$

Note that in general, we have  $|C(v, h)| \leq |N(v, h)|$ , since the labels of nodes in  $N(v, h)$  are not necessarily distinct, and  $r(v) \geq m(v)$ , since there must be at least one node per label but not necessarily one label per node.

The Node-Labeling Decision Problem is defined as follows:

Given:

- An undirected graph  $G = (V, E)$
- A set of  $k \leq |V|$  labels  $K = \{0, 1, \dots, k-1\}$
- A nonnegative integer  $R$ ,

does there exist a labeling  $c(v)$  for all  $v \in V$  such that  $|C(v, R)| = k$  for all  $v \in V$ ?

Now consider this as an optimization problem. The Node-Labeling Optimization Problem is defined as follows:

Given:

- An undirected graph  $G = (V, E)$
- A set of  $k \leq |V|$  labels  $K = \{0, 1, \dots, k-1\}$ ,

find a valid labeling for all the nodes such that  $\max_{v \in V} \frac{r(v)}{m(v)}$  is minimized.

In the case of the optimization problem, if an algorithm that solves it has  $\max_{v \in V} \frac{r(v)}{m(v)} \leq \rho$  for all possible instances, then we say that the algorithm has a *proximity ratio* of  $\rho$ , and the algorithm is a  $\rho$ -proximity algorithm.

First, we will prove that the Node-Labeling Decision Problem is NP-Complete. Then, we will present a polynomial-time algorithm for the Node-Labeling Optimization Problem where the graph is a tree, analyze the proximity ratio of the algorithm, and finally analyze the runtime complexity of the algorithm.

## 2 NP-Completeness Proof

**Theorem 1.** *The Node-Labeling Decision Problem is NP-Complete.*

*Proof.* A problem is NP-Complete if it is in NP and every problem in NP can be reduced to it in polynomial time. We will show the former by presenting a polynomial time algorithm to verify a solution to the Node-Labeling Decision Problem, and the latter by reducing k-coloring to the Node-Labeling Decision Problem.

First, consider the following algorithm to verify a solution to the Node-Labeling Decision Problem:

---

**Algorithm 1:** Verify a Solution to the Node-Labeling Decision Problem

---

**Input:** An undirected graph  $G = (V, E)$ , a set of  $k \leq |V|$  labels  $K = \{0, 1, \dots, k-1\}$ , a nonnegative integer  $R$ , and a labeling  $c : V \rightarrow K$

**Output:** True if the labeling is valid and  $|C(v, R)| = k$  for all  $v \in V$ , False otherwise

```

for  $v \in V$  do
     $l = \emptyset$ 
    if  $\neg \text{BFS}(v, 0, l)$  then
        return False
    end
end
return True

```

---

**Algorithm 2:** BFS

---

**Input:** A node  $v$ , current depth  $d$ , set of labels seen  $l$

**Output:** True if all  $k$  labels are seen within depth  $R$ , False otherwise

```

if  $d > R$  then
    return False
end
 $l = l \cup \{c(v)\}$  if  $|l| = k$  then
    return True
end
for each neighbor  $u$  of  $v$  do
    if  $\text{BFS}(u, d+1, l)$  then
        return True
    end
end
return False

```

---

This algorithm works by performing a breadth-first search from each node  $v$  in the graph, and checking if all  $k$  labels are seen within depth  $R$ . If a depth of  $R$  is reached without seeing all  $k$  labels, the algorithm returns False. Otherwise, the algorithm returns True. The algorithm runs in  $O(|V| \cdot (|V| + |E|))$  time, which is polynomial in the size of the input. Thus, the Node-Labeling Decision Problem is in NP.

Now we perform a reduction from the 3-coloring problem to the Node-Labeling Decision Problem. Let  $G' = (V', E')$  be an instance of the 3-coloring problem. We can construct an instance of the node labeling problem ( $G = (V, E), K, R$ ) as follows:

- $V = V'$
- $E = E'$
- $K = \{0, 1, 2\}$

- $R = 2$

This transformation can be done in polynomial time, since we only copy the graph and set  $K$  and  $R$  to constant values. We claim that  $G'$  is 3-colorable if and only if  $G$  has a valid labeling such that  $|C(v, R)| = 3$  for all  $v \in V$ .

( $\Rightarrow$ ) Assume that  $G'$  is 3-colorable. Then there exists some valid 3-coloring  $c' : V' \rightarrow \{0, 1, 2\}$  of  $G'$ . Let  $c : V \rightarrow K$  be the labeling of  $G$  such that  $c(v) = c'(v)$  for all  $v \in V'$ . Since  $c'$  is a valid 3-coloring, it uses all the colors, so  $c$  will also be a valid labeling of  $G$ . Then, for all  $v \in V$ ,  $N(v, 2)$  must contain at least 3 nodes, so in order for the labeling to be valid, at least 3 distinct labels must be used. Thus,  $|C(v, 2)| = 3$  for all  $v \in V$ .

( $\Leftarrow$ ) Assume that there exists a valid labeling  $c : V \rightarrow K$  of  $G$  such that  $|C(v, 2)| = 3$  for all  $v \in V$ . We can use the same coloring  $c'$  of  $G'$  such that  $c'(v) = c(v)$  for all  $v \in V'$ . For all edges  $(u, v) \in E'$ , we have that  $u \in N(v, 2)$  and  $v \in N(u, 2)$ . Since  $|C(u, 2)| = 3$  and  $|C(v, 2)| = 3$ , we have that  $c'(u) \neq c'(v)$ , or else one of them would only see 2 distinct labels in 2 hops. Thus,  $c'$  is a valid 3-coloring of  $G'$ .

Now we have shown that the Node-Labeling Decision Problem is in NP and that the 3-coloring problem can be reduced to it in polynomial time. Therefore, the Node-Labeling Decision Problem is also NP-hard, and thus NP-Complete.  $\square$

### 3 Approximation Algorithm

Here we discuss an algorithm to solve the Node-Labeling *Optimization* Problem when the input graph is a tree. This is a random heuristic greedy algorithm that starts with a random valid labeling at first, and then iteratively improves the labeling by identifying the “worst” node each time and changing its label such that label diversity in its neighborhood is maximized. Due to the nature of the algorithm, it is not guaranteed to find the optimal solution every time, so it will restart if it gets stuck in local optima. The algorithm is as follows:

## 4 Pseudocode

---

**Algorithm 3:** Node Labeling Algorithm for Trees

---

**Input:** A tree  $T = (V, E)$ , number of labels  $k$

**Output:** A labeling  $c : V \rightarrow \{0, 1, \dots, k-1\}$  that minimizes  $\max_{v \in V} \frac{r(v)}{m(v)}$

```

for  $i = 1$  to  $|V| \cdot k$  do
    Initialize  $c$  to random valid labeling;
     $\alpha = 0$ ;
     $\beta = \infty$ ;
     $\gamma = c$ ;
    for  $j = 1$  to  $|V| \cdot k$  do
        if  $\max_{v \in V} \frac{r(v)}{m(v)} = 1$  then
            return  $c$ ;
        end
        if  $\max_{v \in V} \frac{r(v)}{m(v)} \geq \beta$  then
             $\alpha = \alpha + 1$ ;
        end
        else
             $\alpha = 0$ ;
             $\beta = \max_{v \in V} \frac{r(v)}{m(v)}$ ;
             $\gamma = c$ ;
        end
        if  $\alpha \geq |V|$  then
            Reset  $c(v)$  to random valid labeling;
             $\alpha = 0$ ;
             $\beta = \infty$ ;
            continue;
        end
         $w = \text{random node in } \arg \max_{v \in V} \frac{r(v)}{m(v)}$ ;
         $M = N(w, \lceil \max_{v \in V} \frac{r(v)}{m(v)} \rceil)$ ;
         $S = \{v \in N \mid c(v) = \text{most common label in } M\}$ ;
         $c(\arg \max_{v \in S} \frac{r(v)}{m(v)}) = \text{least common label in } M$ ;
    end
end
return  $\gamma$ ;

```

---

## 5 Proximity Ratio Analysis

Let  $T = (V, E)$  be the input tree with  $n$  nodes,  $k$  labels, and a diameter of  $D$ . First, the lower bound for  $\rho$  is 1, since there must be at least one node per

label. The algorithm will always produce a valid labeling, since it starts with a random valid labeling and only changes the most common labels to the least common labels in some neighborhood. Consider the path of the diameter of the tree. Since the diameter is the longest path in the tree, and we have a valid labeling, it must be that  $C(v, D) = K$  for all  $v \in V$ . Thus, an upper bound for  $\rho$  is  $D$ , and this algorithm is a  $D$ -proximity algorithm for all instances of the problem where the input tree has a diameter of  $D$ .

## 6 Runtime Complexity Analysis

This algorithm contains an outer loop for each “attempt” and an inner loop for each “improvement” of the solution. The outer loop is executed at most  $|V| \cdot k$  times, and the inner loop is executed at most  $|V| \cdot k$  times. Inside the inner loop, we have the following tasks:

- Calculate proximity ratios for each node. This takes  $O(|V|)$  time for each node, so  $O(|V|^2)$  time in total.
- Find the node with the highest proximity ratio. This takes  $O(|V|)$  time.
- Find the neighborhood of the node. This takes  $O(|V|)$  time.
- Find the most common label in the neighborhood. This takes  $O(|V|)$  time.
- Find the nodes with the most common label. This takes  $O(|V|)$  time.
- Find the least common label in the neighborhood. This takes  $O(|V|)$  time.
- Update the label of the node. This takes  $O(1)$  time.

Thus, the algorithm runs in  $O(|V|^4 k^2)$  time in the worst case.