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 $K = \{0, 1, \ldots, k-1\}$ 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 $K = \{0, 1, \ldots, k-1\}$
- 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 $K = \{0, 1, \ldots, k-1\}$,

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 ρ, and the algorithm is a ρ-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 $K = \{0, 1, \ldots, k-1\}$, a nonnegative integer R, and a labeling $c : V \to K$

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 $\neg$ *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 *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 $c' : V' \to \{0, 1, 2\}$ of G'. Let $c : V \to K$ 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 $c : V \to K$ 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.

4 Pseudocode

```
Input: Adjacency list of a tree adj_list, number of labels k
Output: A labeling of the nodes
LabelNodes(adj_list, k, max_attempts) best_labeling ← null;
best_max_ratio ← ∞;
for i ← 1 to max_attempts do
    initial_labels ← InitialLabeling(adj_list, k);
    final_labeling ← ImproveLabeling(adj_list, k, initial_labels);
    ratios ← CalculateProximityRatios(adj_list, k, final_labeling);
    max_ratio ← max(ratios);
    if max_ratio < best_max_ratio then
        best_labeling ← final_labeling;
        best_max_ratio ← max_ratio;
    end
    if max_ratio = 1.0 then
        return final_labeling;
    end
end
return best_labeling;
InitialLabeling(adj_list, k) n ← |adj_list|;
labeling ← [−1, . . . , −1] ;                                  // Length n
for i ← 0 to n − 1 do
    if labeling[i] = −1 then
        available_labels ← {0, . . . , k − 1} \ {labeling[j] : j ∈
          adj_list[i], labeling[j] ≠ −1};
        labeling[i] ← random choice from available_labels (or from
          {0, . . . , k − 1} if available_labels is empty);
    end
end
return labeling;
ImproveLabeling(adj_list, k, labeling) n ← |adj_list|;
for i ← 1 to n · k do
    ratios ← CalculateProximityRatios(adj_list, k, labeling);
    max_ratio ← max(ratios);
    if max_ratio = 1.0 then
        break;
    end
    worst_nodes ← {v : ratios[v] = max_ratio};
    worst_node ← random choice from worst_nodes;
    distances ← BFS(adj_list, worst_node);
    if max_ratio = ∞ then
        neighborhood ← keys of distances;
    else
        neighborhood ← {node : distances[node] ≤ ⌈max_ratio⌉};
    end
    label_counts ← count occurrences of each label in neighborhood;
    most_common_label ←_l label_counts[l];
    least_common_label ←_l label_counts[l];
    nodes_with_most_common ← {v ∈ neighborhood : labeling[v] =
      most_common_label};
    node_to_swap ←_{v∈nodes_with_most_common} ratios[v];
    labeling[node_to_swap] ← least_common_label;
end
return labeling;
BFS(adj_list, start) distances ← {start : 0};
queue ← [(start, 0)];
while queue is not empty do
    node, dist ← dequeue from queue;
```

5 Proximity Ratio Analysis

6 Runtime Complexity Analysis