INF421 PROGRAMMING PROJECT
OPTIMAL TREE LABELING

January 31, 2023

Zuhong LIU and Xiaoxian YANG

ÉCOLE POLYTECHNIQUE

IP PARIS

CONTENTS

1 OVERVIEW OF THE PROBLEM

In this section, we will restate the problem. We have a tree denoted $T = (V, E)$ where V is the set of vertices and E is the set of edges.Let $V = V_1 \cup V_2$, where V_1 is the set of leaf vertices and V_2 is the set of non-leaf vertices. In the optimal labeling tree problem, every leaf vertex $v \in V_1$ is initially labeled with $L(v)$ which is a subset of the overall alphabet $L = \{A, B, C, ..., Z\}$ and every edge, noted as $e = (u, v)$, has a weight $w(e)$ as the Hamming distance between $L(u)$ and $L(v)$ that is:

$$w(e) = |\{X|X \in L(u) \text{ and } X \notin L(v)\} \cup \{X|X \notin L(u) \text{ and } X \in L(v)\}|$$

Our aim is to choose the optimal labels for the non-leaf vertices to minimize the total weight of all the edges in the tree. In mathematics languages, that is to say: Given an application $F : V_1 \to P(L)$ that assigns labels to all leaves of T, where P(L) is the set of all subsets of the library L, we hope to construct an application $F : V \to P(L)$ such that:

$$\begin{aligned} W(T) = \min & \sum_{e \in E} w(e) \\ & s.t.\ G(v) = F(v), \qquad \forall v \in V_1 \end{aligned} \tag{1}$$

2 THE DESCRIPTION AND ANALYSIS OF THE OUR ALGORITHMIC SOLUTIONS

2.1 Description of our solution

In this section, we will describe the main idea of algorithm. We solve this problem by first considering a sub-problem that each vertices have only two possible labels 0 and 1. Then we decompose the original L-labeling tree (L is the overall alphabet) into several disjoint 0-1 labeling tree and go through each one in order to obtain the final solution.

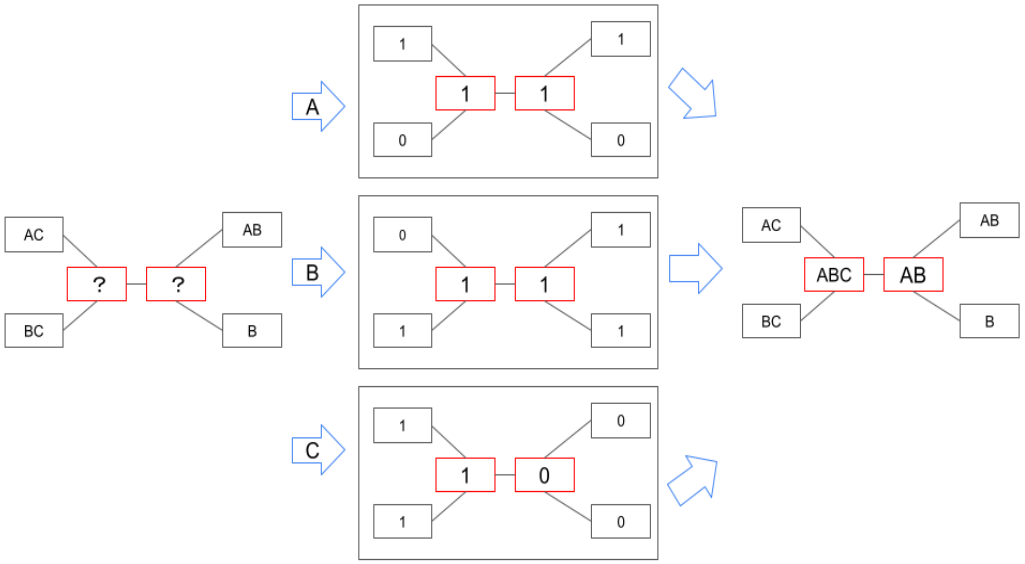

Figure 1: Framework of our algorithm illustrated on the given example

2.1.1 • Sub-problem of the big problem

Given the library of label $L = \{l_1, l_2, l_3, ...l_n\}$ while n is the length of the library. For each vertex v, its label $L(v)$ can be converted correspondingly into a one-hot vector h_v, while $h_v(i) = \mathbf{1}_{l_i \in L(v)}$ for $1 \leq i \leq n$ We notice that a Hamming distance between two neighboring vertices u and v is a superposition of n edges' weights whose values depend only on one letter as the label:

$$w(e) = \sum_{i=1}^{n} \mathbf{1}_{h_u(i) \neq h_v(i)}$$

where $e = (u, v)$

Due to the independence of each single labeling tree and linear additivity of the weight function for each edge e, we can calculate the minimum of the total weight of each single labeling tree and then sum them up.

So now, we just need to solve the sub-problem.Since a vertex is either labeled with or without a letter (we can note the label in a binary way: 1 stands for labeled while 0 stands for unlabeled), we can then sightly modify the Hamming weighty to simplify the notation in the sub-problem:

$$w(e) = \mathbf{1}_{x_u \neq x_v}, \quad x_u, x_v \in \{0, 1\}$$

Then we will describe how our algorithm works in the sub-problem A.We introduce a notation $w_i(v)$ which represents the total weight of the sub-tree T_v whose root is the non-leaf vertex v when v is labeled with $i \in \{0, 1\}$.

$$\begin{aligned} w_i(v) &\overset{not.}{=} w_{x_v=i}(v) \\ &= \sum_{u \ child \ of \ v} W(T_u) + \sum_{\substack{u \ child \ of \ v \\ e=(u,v) \in E}} w(e) \\ &= \sum_{u \ child \ of \ v} (w_0(u)\mathbf{1}_{x_u=0} + w_1(u)\mathbf{1}_{x_u=1}) + \sum_{\substack{u \ child \ of \ v \\ e=(u,v) \in E}} w(e) \end{aligned} \tag{2}$$

For the reason of consistency, we suppose $w_i(u) = \begin{cases} 0 & x_u = i \\ \infty & x_u \neq i \end{cases}$, for each leaf vertex u.

2.1.2 • Description of algorithm

Now we begin to explain our algorithm. First, we randomly select a non-leaf vertex r as the root of the entire tree T. For each sub problem l_j, based on formula 2, we obtain that the optimal total weight of the vertex v, $w_i(v)$ can be expressed as following:

$$w_i(v) = \sum_{u \ child \ of \ v} \min(w_i(u), 1 + w_{1-i}(u)), \qquad i \in \{0, 1\}$$

Since that the children of v may still be non-leaf vertices, we need to continue calculating $w_i(u)$ and $w_{1-i}(u)$ by checking the children of u. To realize this step, we can iteratively calculate $w_0(v)$ and $w_1(v)$ in a bottom-up way and store their values as done in all dynamic programming algorithms. Then we obtain that $min(w_0(r), w_1(r))$ is the solution for the sub problem l_j. Finally, we obtain the global minimum by summing up the results of all sub-problems.

However, we cannot obtain labels for each vertex only through this bottom-up process. To achieve this goal, we revisit each vertex through a top-down broad first search to determine whether the vertex is labeled as 0 or 1. The pseudo-code 1 of our algorithm is presented as follows.

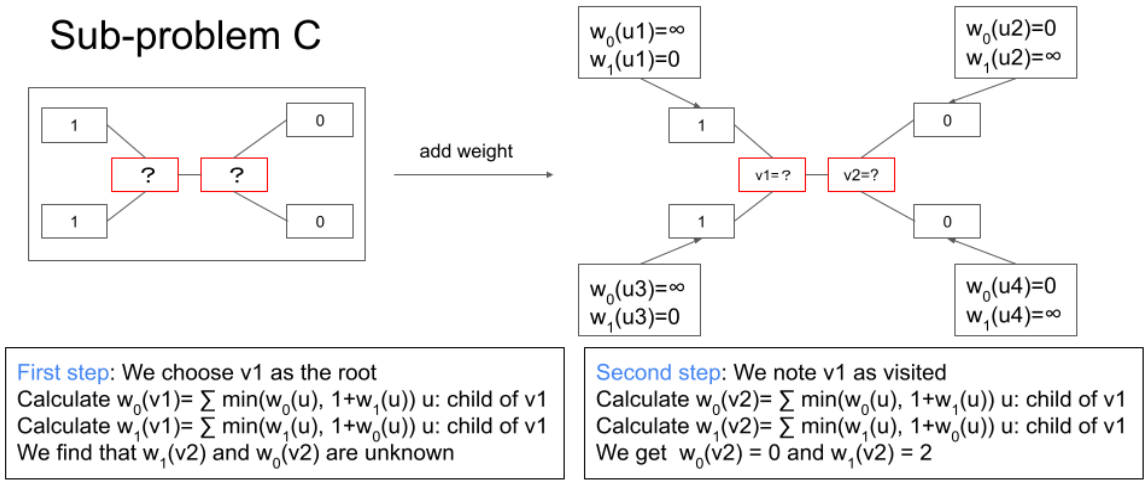

Figure 2: A illustration to show how to solve sub-problem C on the given example

Algorithm 1 Optimal Tree Labeling

Input: A tree with labeled leaf vertices T
Output: The optimal total weight $W(T)$

function Update(v)
  $visited[v] = True$
  for $u \in N(v)$ **do**
    if $visited[u] = False$ **then**
      Update(u)
    end if
  end for
end function
function Label(v)
  let l be the label of v
  for $u \in N(v)$ **do**
    if $labeled[u] = False$ **then**
      $labeled[u] = True$
      u is labeled as l or $1 - l$ based on the order of two terms
      Label(u)
    end if
  end for
end function
function solve( )
  Initialize $w_i(v)$,for all leaf vertices and flags like *labeled* and *visited* for all vertices
  Randomly choose a non-leaf vertex r as the root
  $sum = 0$
  for each sub-problem **do**
    Update(r)
    $sum+ = \min w_i(r)$
    r is labeled as $\arg\min_{i \in \{0,1\}} w_i(r)$
    Label(r)
  end for
  return *sum*
end function

2.2 The analysis of the algorithm

2.2.1 • Termination

We implement our algorithm using recursion in the process of weight updating and labeling. As shown in the pseudo-codewe switch the value of flags when we do certain operations (Update or Label). Therefore,the algorithm absolutely terminates after all vertices have been visited and labeled.

2.2.2 • Correctness

We prove the correctness of a single sub-problem by induction which could be generalized to other sub-problem. We first define $w_(x_v^*)(v)$ as the minimum of total edge weight of the sub-tree T_v.

Lemma 1 *For the 0-1 sub-problem of labeling tree, $\forall v \in V$, for another possible x_v, $w_(x_v)(v) \geq w_(x_v^*)(v)$. Proof: For all leaf vertices, it's trivial that $w_(x_v)(v) \geq w_(x_v^*)(v) = 0$.For a non-leaf vertex v, we suppose by*

induction that the proposition is valid for each child u. Then we have:

$$w_{x_v}(v) = \sum_{\substack{u \ child \ of \ v \\ x_u = x_v}} w_{x_u}(u) + \sum_{\substack{u \ child \ of \ v \\ x_u \neq x_v}} (1 + w_{x_u}(u)) \tag{3}$$

$$\geq \sum_{\substack{u \ child \ of \ v \\ x_u^* = x_v}} w_{x_u^*}(u) + \sum_{\substack{u \ child \ of \ v \\ x_u^* \neq x_v}} (1 + w_{x_u^*}(u))$$

$$\geq \sum_{\substack{u \ child \ of \ v \\ x_u^* = x_v}} \min(w_{x_u^*}(u), 1 + w_{1-x_u^*}(u)) + \sum_{\substack{u \ child \ of \ v \\ x_u^* \neq x_v}} \min(w_{x_u^*}(u), 1 + w_{1-x_u^*}(u))$$

$$= w_{x_v^*}(v) \tag{4}$$

Using Lemma 1, we can prove that $w_{x_r}(r)$ is the optimal weight for T. Thus, the correctness of the overall algorithm is proved.

2.2.3 • Complexity

Now we integrate the 26 sub-problems to the overall problem by simply adding them together. Therefore, the time complexity of the whole program should be $|L| \times O(subproblem)$, where L is the size of the alphabet. In our algorithm we use adjacency list as the main data structure to store the graph so that we transverse each node once and compute the weight when i is either 0 or 1 for the updating process and revisit each node once for the labeling process in $O(1)$ operations. Hence, theoretically, the time complexity should be $O(|L||V|)$.

2.3 Experience and Final Result

We provide the final minimum that we found for each case in Table 2.3, Due to the large number of vertices of each case, we cannot give a clear visualization for all labels in the tree in our report. See more details in our code implementation https://github.com/ZuhongLIU/INF421.

	case0	case1	case2	case3	case4	case5	case6
$\|V\|$	100	2000	3000	4000	5000	6000	30000
Value	24	1682	6936	12927	3360	24971	128297
Time (ms)	35	93	130	172	194	223	937

Table 1: Final results of our Algorithm

After testing on the 8 cases given on Moodle, we draw an image in Figure 2.3 to show the relation between running time and number of vertices which is coincided with our theory.

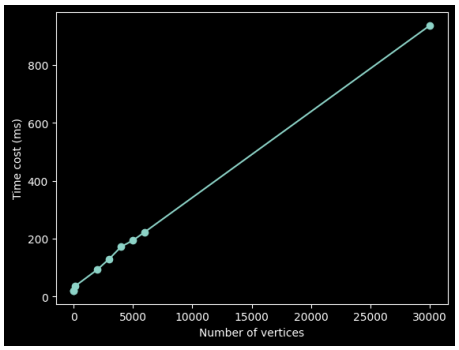

Figure 3: running time and number of vertices