Project on Algorithms

I. The "Node Labeling Problem"

Let $G = (V, E)$ be an undirected graph. For every node $v \in V$ and every non-negative integer $h \geq 0$, let $N(v, h) \subseteq V$ be the set of nodes in G that are at most h hops away from v (including v itself). That is, if we use $d(u, v)$ to denote the minimum distance from u to v (measured by the number of edges in the shortest path between them), then

$$N(v, h) \triangleq \{u \in V \mid d(u, v) \leq h\}.$$

Let $K = \{0, 1, \cdots, k-1\}$ be a set of k integers, where $k \leq |V|$. For each node $v \in V$, we will assign an integer $c(v) \in K$ to v, and call $c(v)$ the "label of v". (Different nodes may get the same label.) The set of distinct labels within h hops from v is denoted by $C(v, h)$, namely,

$$C(v, h) \triangleq \{c(u) \mid u \in N(v, h)\}.$$

Note that since different nodes may be assigned the same label, in general we have $|C(v, h)| \leq |N(v, h)|$.

A labelling of the nodes (namely, the assignment of one label to each node) is called "valid" if every label in K is assigned to at least one node in G. For every node $v \in V$, let $r(v)$ be the smallest integer such that $|C(v, r(v))| = k$, namely,

$$r(v) \triangleq \min\{h \mid |C(v, h)| = k\}.$$

In other words, $r(v)$ is the smallest integer such that the node v can find all the k distinct labels within its neighborhood of radius $r(v)$. Similarly, let $m(v)$ be the smallest integer such that $|N(v, m(v))| \geq k$, namely,

$$m(v) \triangleq \min\{h \mid |N(v, h)| \geq k\}.$$

In other words, $m(v)$ is the smallest integer such that the node v has at least k nodes in its neighborhood of radius $m(v)$ (including v itself). Since every node has exactly one label, we have

$$r(v) \geq m(v).$$

Let us first define a "Node-Labelling Decision Problem":

Node-Labelling Decision Problem

Input: We have the following inputs:

1) An undirected graph $G = (V, E)$.
2) A set of k labels $K = \{0, 1, \cdots, k-1\}$, where $k \leq |V|$.
3) A non-negative integer R.

Question: does there exist a labeling that assigns a label $c(v) \in K$ to every node $v \in V$, such that for every node $v \in V$ we have $|C(v, R)| = k$ (namely, every node can find all the k distinct labels within a neighborhood of radius R)?

Two examples of the Node-Labelling Decision Problem are shown in Fig. 1.

Let us now consider the node labeling problem as an optimization problem. We would like to assign labels to nodes such that for every node $v \in V$, $r(v)$ is as small as possible. Since $m(v)$ is a lower bound of $r(v)$, we would like the ratio $r(v)/m(v)$ to be as small as possible. So we define a "Node-Labelling Optimization Problem" as follows.

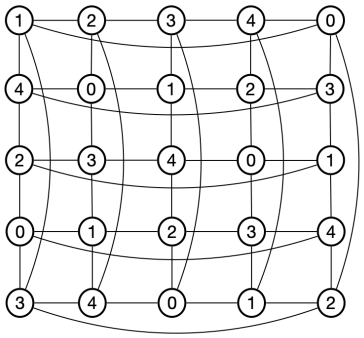
Node-Labelling Optimization Problem

Input: We have the following inputs:

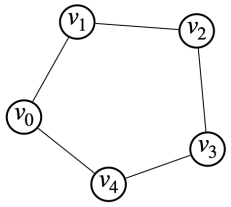
1) An undirected graph $G = (V, E)$.
2) A set of k labels $K = \{0, 1, \cdots, k-1\}$, where $k \leq |V|$.

Output: A valid labelling that assigns a label $c(v) \in K$ to every node $v \in V$, such that the maximum ratio

$$\max_{v \in V} \frac{r(v)}{m(v)}$$



(a) $k = 5,\ R = 1$



(b) $k = 3,\ R = 1$

Fig. 1. (a) G is a 5×5 torus with 25 nodes, $k = 5$ and $R = 1$. The answer is "yes" to the Node-Labelling Decision Problem. (b) G is a cycle with 5 nodes, $k = 3$ and $R = 1$. The answer is "no" to the Node-Labelling Decision Problem.

is minimized.

Given a deterministic algorithm for the "Node-Labelling Optimization Problem", if the algorithm has

$$\max_{v \in V} \frac{r(v)}{m(v)} \leq \rho$$

for all possible instances, we call the algorithm a "ρ-proximity algorithm", and call ρ the "proximity ratio" of the algorithm. (Here "proximity ratio" is similar to the concept of "approximation ratio" that we have learned in class.)

In this project, we will need to:

1) Prove that the "Node-Labelling Decision Problem" is NP-complete.
2) Design a polynomial-time algorithm for a special case of the "Node-Labelling Optimization Problem" – where the graph $G = (V, E)$ is a tree – with a "proximity ratio" that is as small as possible.

II. What to Submit

In this project, you need to submit three items: (1) a report for NP-completeness proof and algorithm design, (2) a program that implements your algorithm, (3) output of your program on test instances.

A. Submission Item One: a report for NP-completeness proof and algorithm design

First, you are to (1) prove that the "Node-Labelling Decision Problem" is NP-complete, (2) design an efficient algorithm for a special case of the "Node-Labelling Optimization Problem" where the graph $G = (V, E)$ is a tree. You then submit the above results as a report. The report should have the following elements:

1) A rigorous proof showing that the "Node-Labelling Decision Problem" is NP-complete.
2) A polynomial-time algorithm for a special case of the "Node-Labelling Optimization Problem" where the graph $G = (V, E)$ is a tree. It should have the following four elements:
   - The main idea of your algorithm.
   - The pseudo code of your algorithm.
   - A rigorous proof that proves your algorithm has an "proximity ratio" ρ, for a constant number ρ that you specify.
   - The analysis of time complexity for your algorithm, which shows that your algorithm has polynomial-time complexity. (Note that we do not require the time complexity to be the lowest possible. It is enough to just show that it is polynomial time.)

B. Submission Item Two: A Program That Implements Your Algorithm

You need to implement your algorithm (for the "Node-Labelling Optimization Problem" where the graph $G = (V, E)$ is a tree) using a commonly used programming language, such as Python, C++, Java, etc. We encourage you to use Python if possible, but other languages are fine, too.

Your program will take a list of instances of the "Node-Labelling Optimization Problem" as input, and return a list of solutions (corresponding to those instances) as its output.

To test your program, we will provide you with input instances as two lists (one containing the adjacency lists of trees, the other containing the number of labels k for each tree) in Python (after you have submitted the program), and ask you to submit the output solutions as a "list" in Python (which contains the labelling for each tree). (Details on the three files, including their formats, will be explain in the next subsection for "Submission Item Three".)

When you submit your program, you need to explain clearly how to run your code. If your programming language is not Python, then you also need to include an explanation on how to turn the provided "two lists" of "input instances" in Python to your programming language, and how to turn your "output solutions" in your programming language to a "list" in Python (following the formats specified in the next subsection).

C. Submission Item Three: Output of Your Program on Test Instances

To test your algorithm/program, we will provide you with three sets of "input instances", all in the Python language:

1) A set of instances of relatively small sizes.
2) A set of instances of medium sizes.
3) A set of instances of relatively large sizes.

Note that an algorithm is not only about correctness, but also about efficiency. When you run your algorithm on the three test sets, if its time complexity is high, then you may notice the difference in the actual running time for the three test sets.

To give you an idea what the three sets of input instances are like, we provide three "example" datasets (downloadable from our course webpage):

1) A small example dataset: it has two files
   - "Small_Examples_of_AdjLists_of_Trees"
   - "Small_Examples_of_k_values"
2) A medium example dataset: it has two files
   - "Medium_Examples_of_AdjLists_of_Trees"
   - "Medium_Examples_of_k_values"
3) A large example dataset: it has two files
   - "Large_Examples_of_AdjLists_of_Trees"
   - "Large_Examples_of_k_values"

Each of the above 6 files is a Python list, whose format will be explained below. You can download them by clicking on the links in our course webpage. After downloading each file, you can open it using pickle.load(open(filePath, 'rb')), where "filePath" is the path of your downloaded file. (Of course, these 6 files are different from the 6 files we will send you for testing your program. But they are similar.)

After you run your program on the three test sets of "input instances" (not the above three example sets), save your results as three Python "lists" in three files:

1) File "small_solutions" corresponding to the input instances of small sizes.
2) File "medium_solutions" corresponding to the input instances of medium sizes.
3) File "large_solutions" corresponding to the input instances of large sizes.

Now let's explain the formats of the "input instances" and "output solutions".

The list whose filename ends with "_AdjLists_of_Trees" is a list of N elements, where N is the number of trees (namely, N is the number of instances here). For $i = 0, 1, \cdots, N-1$, the i-th element is the adjacent list of the i-th tree. Let the i-th element be denoted by Adj_list. Then Adj_list is a list of n elements, where n is the number of nodes in the i-th tree. (The n nodes in the i-th tree are indexed by 0, 1, $\cdots$, $n-1$. Note that the value of n changes from tree to tree.) For $j = 0, 1, ..., n-1$, Adj_list[j] is a list of integers between 0 and $n-1$, which are the indices of the nodes that are adjacent to node j in the i-th tree. That is, if Adj_list[j][r] = k for some r, then node k is adjacent to node j in the i-th tree.

The list whose filename ends with "_k_values" is a list of N positive integers, where N is the number of trees (namely, N is the number of instances here). For $i = 0, 1, \cdots, N-1$, its i-th integer is the number of labels k for the i-th tree (namely, for the i-th instance).

The above two lists (one for adjacency lists, one for the number of labels k) both belong to the "input instances". There are N instances in total. So for $i = 0, 1, \cdots, N-1$, the i-th element of the first list and the i-th element in the second list together form the i-th input instance.

The "output solutions" is a "list" (in Python) of the following format: it is a list of N elements. For $i = 0, 1, \cdots, N-1$, its i-th element is a list that stores the labels of the nodes in the i-th tree. More specifically, let label_of_node denote the i-th element, let n be the number of nodes in the i-th tree, and let k be the number of labels for the i-th tree. (Note that the values of n and k change from tree to tree.) Then label_of_node is a list of n integers. For $j = 0, 1, \cdots, n-1$, label_of_node[j] is an integer between 0 and $k-1$, which is the label assigned to node j in the i-th tree. Remember that to make the labelling "valid", each of the k labels should be assigned to at least one node in the i-th tree.

We give three small "example" files for the "input instances" and "output solutions". Their filenames are "Examples_of_AdjLists_of_Trees", "Examples_of_k_values" and "Examples_of_labelling", and you can download them by clicking the links in the course webpage. (As before, after you have downloaded a file, you can use pickle.load(open(filePath, 'rb')) to open it.) The files "Examples_of_AdjLists_of_Trees" and "Examples_of_k_values" contain $N = 10$ sample instances, and the file "Examples_of_labelling" contains 10 corresponding solutions (which are not necessarily optimal solutions). Here is an example:

- Among the 10 instances here, Instance 0 has the following details. Its tree $G = (V, E)$ has $n = 12$ nodes, and there are $k = 6$ distinct labels. The adjacency list is as follows:

	neighbors
node 0	1, 2
node 1	0, 3, 4
node 2	0, 5, 6
node 3	1, 7, 8, 9
node 4	1, 10, 11
node 5	2
node 6	2
node 7	3
node 8	3
node 9	3
node 10	4
node 11	4

The tree and its labelling are as shown in Fig. 2. (The number beside each node is its label.) The maximum ratio $\max_{v \in V} \frac{r(v)}{m(v)}$ for this tree is 1.5. (For example, as can be seen in the figure, $r(v_0) = 3$ and $m(v_0) = 2$, so $\frac{r(v_0)}{m(v_0)} = \frac{3}{2} = 1.5$.) It means that if the labelling shown here is generated by an algorithm, the "proximity ratio" of the algorithm is at least 1.5.

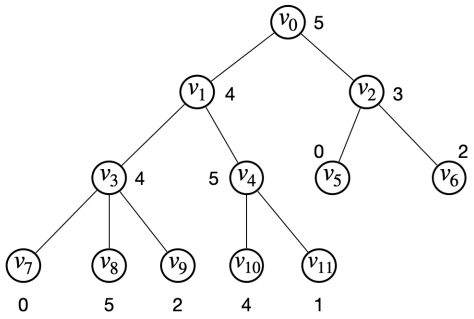


Fig. 2. The 0th instance among the 10 example instances.

III. How to Grade Project

The project has a total of 100 points. If any of the three items below is not submitted on time, the whole project will receive no point.

Assuming that all the three items are submitted on time, the project will be graded as follows.

A. Submission Item One: A Report on Proof of NP-completeness and Algorithm Design

The algorithms report has 35 points:

- 15 points for the proof showing that the "Node-Labelling Decision Problem" is NP-complete. The proof should be rigorous and clear.

- 15 points for proving that your algorithm for the "Node-Labelling Optimization Problem" with $G = (V, E)$ being a tree has an "proximity ratio" of ρ, where ρ is a constant finite number that you specify. The proof should be rigorous and clear.
  Note: for the three test sets, we will compute three values given your submitted labelling solutions, which are denoted by A_{small}, A_{medium} and A_{large}, respectively (whose definitions are presented below). They are all lower bounds to the proximity ratio of your algorithm. Therefore the ρ value that you specify here should satisfy $\rho \geq \max\{A_{small}, A_{medium}, A_{large}\}$. (It is easy for you to find the values of A_{small}, A_{medium} and A_{large}.) So if $\rho < A_{small}$, or if $\rho < A_{medium}$, or if $\rho < A_{large}$, all the 15 points here will be deducted.
- 2 points for the "main idea" and "pseudo code" of your algorithm. They should be correct and clear.
- 3 points for the "time-complexity analysis". It should not only be correct and clear, but should also show that your algorithm achieves polynomial-time complexity. (But note that we do not require the time complexity to be optimally low. Just having polynomial time complexity is enough here.)

B. Submission Item Two: A Program That Implements Your Algorithm

The program has 5 points.

If your program has bugs or does not have a clear interface – which means we have to revise your code or spend time trying to understand how to run your code – some points will be deducted.

C. Submission Item Three: Output of Your Program on Test Instances

This part has 60 points.

After you have submitted your "algorithm report" (submission item one) and "program" (submission item two), we will post 6 files for the 3 test sets: (1) file "Test_Set_Small_AdjLists_of_Trees" and file "Test_Set_Small_of_k_values" as the small test set; (2) file "Test_Set_Medium_AdjLists_of_Trees" and file "Test_Set_Medium_of_k_values" as the medium test set; (3) file "Test_Set_Large_AdjLists_of_Trees" and file "Test_Set_Large_of_k_values" as the large test set. They contain instances of small, medium and large sizes, respectively. You need to run your submitted program (the exact program that you have submitted as Submission Item Two) on the three sets of instances, save your results in three files "small_solutions", "medium_solutions", and "large_solutions" and submit them. Those three files are your "Submission Item Three", and they have 60 points:

1) For the "small_solutions", let A_{small} denote the maximum value of the ratio $\frac{r(v)}{m(v)}$ over all the nodes in all the trees of the small test set, namely,
$$A_{small} = \max_{G=(V,E)} \max_{v \in V} \frac{r(v)}{m(v)}.$$
Then the points you get here is $\frac{20}{A_{small}}$.
2) For the "medium_solutions", let A_{medium} denote the maximum value of the ratio $\frac{r(v)}{m(v)}$ over all the nodes in all the trees of the medium test set, namely,
$$A_{medium} = \max_{G=(V,E)} \max_{v \in V} \frac{r(v)}{m(v)}.$$
Then the points you get here is $\frac{20}{A_{medium}}$.
3) For the "large_solutions", let A_{large} denote the maximum value of the ratio $\frac{r(v)}{m(v)}$ over all the nodes in all the trees of the large test set, namely,
$$A_{large} = \max_{G=(V,E)} \max_{v \in V} \frac{r(v)}{m(v)}.$$
Then the points you get here is $\frac{20}{A_{large}}$.

Note that to compute the maximum ratios A_{small}, A_{medium} and A_{large}, the submitted solutions need to be valid. So if any submitted solution is not valid, the corresponding maximum ratio cannot be computed, and consequently that submitted solution will receive 0 point. Here are some errors that would make the submitted solution invalid:

1) If the number of labellings in the submitted solution is not equal to the number of given trees, it is an error.
2) If the number of labels assigned by the submitted solution is not equal to the number of nodes in the corresponding tree, it is an error.
3) If any assigned label is less than 0 or more than $k - 1$ (where k is the number of distinct labels), it is an error.
4) If some label is not used to label any node in some tree, it is an error.

Please also note that the program you use to generate the solutions here should be exactly the same program that you have submitted as "Submission Item Two". If any modification is detected (e.g., if we find that some solution submitted here is not the same as what the already-submitted program would produce), the project will receive 0 point.

IV. Hint on the NP-completeness Proof

Here is a hint on how to prove the NP-completeness of the "Node-Labelling Decision Problem": reduce the 3-coloring problem (which is known to be NP-complete) to the problem here.