CENG567: Homework #2

Yiğit Sever

November 15, 2020

1 Checking Consistency of Judgements

Given the collection of n butterflies and a potential judgement between every pair (or not if the judgement is ambiguous), we have a graph $G = (V, E)$ with $n = |V|$ vertices and $|E| = m \leq \frac{n(n-1)}{2}$ edges, with every edge $(i, j) \in E$ labelled either "same" or "different", assuming (i, j) is not judged again as (j, i). At the end of our algorithm, the vertices should be *consistently* labelled as either `A` and `B` or our algorithm should be able to prove that G cannot be labelled as so.

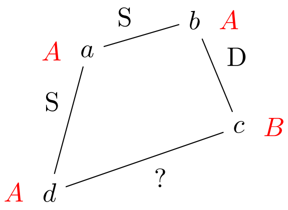
A modified graph traversal using either BFS or DFS will work. Here we will modify the graph traversal given on *page 42* on our 3rd lecture slides that uses BFS. The input of the algorithm is any node $s \in E$. If, due to ambiguous (i.e. missing) nodes, the graph is not connected, the algorithm should be run on a new undiscovered node until every connected component is discovered.

Algorithm 1: Modified graph traversal algorithm so solve judgement consistency checking problem

```
function consistency_check(s: node)
    Data: K = data structure of discovered nodes
    Result: boolean = whether G is consistent or not
    label s as A                    /* the opposite label is B */
    put s in K
    while K is not empty do
        take a node v from K
        if v is not marked "explored" then
            mark v "explored"
            for each edge (v, w) incident to v do
                if w is labelled then
                    if the label of w is not consistent with the label
                      of v with respect to the judgement (v, w) then
                        terminate the algorithm; G is inconsistent
                    end if
                else
                    if (v, w) is labelled "same" then
                        label w with the label of v
                    else  /* (v, w) is labelled "different" */
                        label w with the opposite label of v
                    end if
                end if
                put w in K
            end for
        end if
    end while
    the spanned connected component is consistent
end
```

With the assumption that accessing the labels (u, w) takes $\mathcal{O}(1)$ time this algorithm has the same running time as BFS; $\mathcal{O}(m + n)$.

To give a short proof, consider the scenario below;



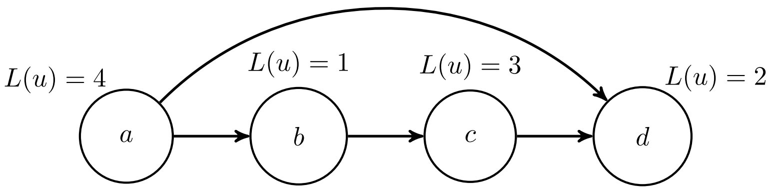
Consider the situation where we started at node a with the label `A`, labelled b with `A` due to "same" (a, b) edge and labelled d with `A` due to "same" (a, d) edge. Node c will be labelled with `B` due to "different" (b, c) edge. Now, depending on the judgement on (or lack thereof) (d, c) edge, the graph will be either consistent (if "different") or inconsistent (if "same"). Since the modified BFS above visits every node at least once and considers every edge at least once (property of BFS), an inconsistent path will be discovered or none will occur, meaning a consistent graph.

2 Reachability

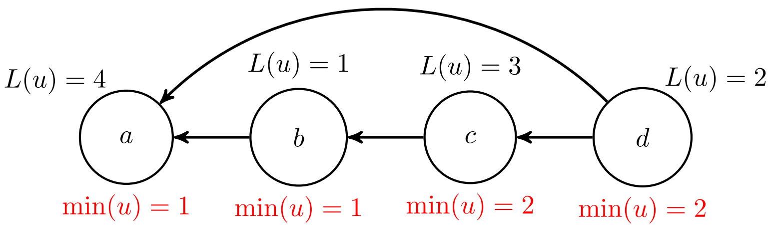
The intuition for this question can be illustrated as follows;

If G is strongly connected then the answer is trivial; every vertex have the same $\min(u) = 1$.

For a directed acyclic graph, consider the scenario below;



We can assign the $\min(u)$ values on this directed acyclic graph by reversing the direction of the edges and traversing the graph, keeping a minimum $\min(u)$ encountered so far for the nodes;



Starting from the "root" `d`, the $\min(u)$ is 2, which is assigned to the immediate neighbours of `d`, `c` and `a`. When the traversal *reaches* `b`, `b` sets the current $\min(u)$ to 1 and `a`'s $\min(u)$ value is updated to 1 as well.

With the intuition out of the way, we will generalize the problem. First, compute all strongly connected components (SCCs) of G by using (Tarjan 1972) per *page 72* of the 3rd lecture notes in $\mathcal{O}(E + V)$ time. Instead of labelling the SCCs with their root node, we will initially label all nodes of the SCC F' with the $\min(u)$ of the connected component; $\min(a)$ of $a \in F' = \min\{L(w) : w \in F'\}$.

Then, by ignoring the tree edges, *shrink* the graph G such that $E' = \{(v, w) \mid v \in F', w \in F''\}$, leaving only cross links behind. This step takes another $\mathcal{O}(E + V)$ time.

Now run the topological sort algorithm presented in *page 84* of the 3rd lecture notes. This operation is yet again $\mathcal{O}(E + V)$.

We now have a scenario like the one illustrated above. Reverse the direction of the edges on the graph that have been output by the topological sort and starting from the new root node, traverse the graph downwards and update the $\min(u)$ of every SCC in $\mathcal{O}(m + n)$ time as follows;

Algorithm 2: Updating $\min(u)$ of the SCCs

Data: G' = topological sorted G with reversed edges
Result: $\min(u)$ for all vertices $u \in V$
minnest $\longleftarrow$ min(root)
while *traversing G' downwards with current node v* **do** // $m + n$
  if $\min(v) < minnest$ **then**
    minnest $\longleftarrow \min(v)$
  else
    label v as *minnest*
  end if
end while

Finally, update the $\min(u)$ of every node $w \in G$ to match the cross link edges of the SCC they belong to. This operation is yet another $\mathcal{O}(m + n)$ traversal.

References

Tarjan, R. (1972). "Depth-First Search and Linear Graph Algorithms". In: *SIAM J. Comput.* DOI: 10.1137/0201010.