SSSP Initialization

Ashish Kumar

September 2014

1 Basic Idea of the Data Structure

We intend to make a linear list in which any subtree of the original tree would occur contiguously in the list. So, to identify the descendants of a node, we just traverse that contiguous sub-part of the list.
For this, we would also need to know the length of the list. The mechanism we suggest has an added advantage of letting us know in O(1) time if a node is a descendant of a given node. The space complexity of the data structures is O(n), where n is the number of nodes in original tree.

2 Data Structures Used

We will maintain the following three structures –

1. A doubly linked list, containing the node number and an associated value. These values are in increasing order when traversing the linked list. The need for this value and how it is computed will be clear subsequently. This value will help us identify if a given node is a descendant of another node. The basic idea is to check if the value of the child node is within some range of the value of the parent node. This range value is obtained from the third data structure.
2. We will maintain an array of pointers, each pointing to a node. This would help us to locate a node in constant time if we know its index.
3. We will maintain an array of floating point numbers – which tell us the range for each node. This array will be indexed by node number.

3 How it helps in Initialization

The usual initialization process expects us to add all the Affected Nodes in the queue. Bellman ford is then run on this queue. We will describe the idea of edge insertion. Let us assume that inserting the new edge improve the shortest paths

The usual way to initialize it is to mark all the descendants of the node affected by the edge insertion, reduce their weight by the improvemnt in weight of the parent node, and then check all their outgoing edges, to see if the node [whose weight we just reduced] gives a better path to any of its immediate children. If yes, it is inserted in the queue. But, none of its children should be a descendant of the originally affected node.
Our method optimizes in the follwoing ways –

1. The original method to initialize has to linearly traverse the affected nodes and their out going edges because if done in parallel, it would require linear time to see if a node is a descendant of the originally affectd node. We can do this in constant time.
2. This enables us to make the initialization process parallel, and generate a task, where each task contains examining the out going edges of a descendent node [which are descendant of the originally affected nodes]

4 Initialize the Data Structure

The list would basically be a pre-order traversal of the tree. Another linear traversal of the tree would give the values to the nodes in increasing order of natural numbers. Another traversal of the list can be used to fill up the range-array, according to the asigned values.

5 Maintain the Data Structure

To maintain the data structure, if we change the parent of a node A from X to Y, then, in the forst list, we insert it just infront of Y: Y-¿A-¿B. Now, the value of A is 1/2*(val(Y)+val(B)) and its range is 1/2(new-val(A)+val(B)). Any children of A which we add subsequently are given the value within the range of A. To iterate over the children of A, we keep on iterating linearly as long as the values of the encountered nodes are within the specified range of A.