Advanced Algorithms

Erik Betzholtz, erikbet@kth.se
Pothuganti Srilekha, potsri@kth.se
Melissa Rebecca Mazura, mazura@kth.se
Raahitya Botta, raahitya@kth.se

Kattis ID: 12445088

1 Introduction

For this project, our goal was to develop a sub-linear time algorithm for estimating the weight of a minimum-spanning forest (MSF) with a maximum a relative error of 0.1 in either direction. The normal precise graph algorithms for minimum-spanning tree (MST) weight calculation need a lot more time as they take in the whole graph as input, therefore making them inefficient and not ideal for all problems. The use of sub-linear algorithms makes it easier as they takes a fraction of the input and provide an approximate solution, which is very useful when we have a large amount of input data. So in our case, to implement the sub-linear algorithm for MSF, we initially studied how it's done on MST[1] and further adapted it onto minimum-spanning forests.

2 Minimum spanning tree to forest

2.1 MST

Before moving on to forest, let's study minimum spanning trees first. They are considered one of the most basic problems in graph theory but finding the exact weight of an MST requires knowledge of all the nodes in the graph. It is, however, possible to calculate an approximate value of the weight of the MST by utilizing connected components, containing only edges of certain weights. Such an algorithm starts by assuming the graph to be an undirected graph G=(V,E) with edge weights ranging from $\{1, ..., W\}$. The aim of the algorithm is to approximate the MST for a graph, given the number of vertices n, the maximum edge weight W and the number of vertices allowed to access q. For this we can utilize a formula for determining the weight of an MST presented in[1]:

$$\text{MST} = n - W + \sum_{i=1}^{W-1} c^{(i)}$$

Here, $c^{(i)}$ is the number of connected components of sub-graph $G^{(i)} = (V, E^{(i)})$ having at most i as edge weights.

The formula is based on the fact that a graph with $c^{(i)}$ connected components of weight at most i will have an MST which connects these components with edges of weight greater than i. As such, we can deduce the number edges in the MST with weight higher than i through $c^{(i)} - 1$ and the number of edges of weight i or less through $n - 1 - (c^{(i)} - 1)$.

To use this formula in an approximation-algorithm where we do not have access to all the vertices, we can instead estimate $c^{(i)}$ as $\hat{c}^{(i)}$ though the following method[1]:

Algorithm 1 ApproxConnectedComponents(G, s)

Choose s vertices v_1 to v_s at random and uniformly
for $i = 1$ to s **do**
    choose X such that $Pr[X \geq k] = 1/k$
    run BFS starting at v_i until
    (1) the whole connected component containing v_i is explored or
    (2) X vertices have been explored.
    if BFS has stopped at (1) **then**
        $b_i = 1$
    else
        $b_i = o$
    end if
end for
Output $\hat{c} = n/s \sum_{i=1}^{s} b_i$

The basic process of the algorithm is to compute the likelihood of a given node from a sample being in a connected component of a limited size, decided by a random distribution. The likelihood can then be multiplied by the number of nodes in order to estimate the number of such connected components in the whole graph.

2.2 MSF

To adapt the above MST algorithm to work for MSF, we made two observations. Firstly, since using the approximation of connected components with weight W-1 for an MST yields an approximaiton of how many connected components there are in the graph that are not the whole MST, then, using the approximation of connected components with weight W for an MSF should yield an approximation

of how many trees it consists of. Secondly, since running the approximation of an MST on an MSF yields an approximate weight, calculated as if the separate trees were connected with edges of weight W. Therefore we must subtract one W for each tree in the MSF, giving us the formula:

$$\text{MSF} = n - (W * Trees) + \sum_{i=1}^{W-1} c^{(i)}$$

Where $Trees = c^{(W)}$

The pseudocodes used in our MSF algorithms for finding approx MSF weight, approx connected components, and BFS are given in section 5:

3 Process

3.1 Implementation

The initial strategy consisted of studying and implementing the algorithm provided in the given paper[1]. This proved to be more difficult than just translating the pseudo-code as it was provided at a general level and our specific implementations might be more extensive.

Most complications with the implementation of the program arose with the BFS function which was omitted from the pseudo-code in the paper. The first issue related to which nodes should be considered part of a connected component. At first the intuition was that only edges with weight equal to exactly the current weight should be considered part of the connected component. This was, however, later corrected to include edges with weight equal to or less than the current weight in accordance with the paper.

The BFS algorithm also had some unexpected results when it came to the return statement as, for example, returning a value based on if the node count had exceeded the threshold gave different results than returning based on of the queue was empty. This might have had to do with cases where the component is exactly as big as the given X and so technically both conditions were fulfilled. This was also adjusted to be in exact accordance with the paper. The final implementation can be found in section 5.

3.2 Decisions

Additionally there were some parts of the algorithm which were possible to decide based on our implementation.

One such example was the number of vertices s to be used in the connected components estimation which was initially set to match the number of nodes allowed to be studied q provided in the problem instance. However, since the connected components approximation will run once for each weight and each

node will undergo a BFS, the total number of nodes explored would be closer to $q \times w \times X$. To compensate for this, s was set to q/w which increased the speed of the program, made no critical difference in the accuracy of the approximation and likely contributed to a better score.

Another unspecified variable was X which was to be chosen from a distribution such that $Pr[X \geq k] = 1/k$. This was initially quite difficult to understand how to implement but when introduced to the fact that with picking X randomly and uniformly from [0,1], leads to $Pr[X \leq t] = t$, the problem was made easier. What follows is that $Pr[X \geq t] = 1/t$ which means we can select X from $1/random(0, 1)$. For practical reasons we also limited X since $1/random(0, 1)$ could be a very large number and if a graph contained a very large connected component this could imply searching a lot of vertices. This upper limitation was initially set to 100 but was later changed to 10 as no severe negative impact was seen in the estimations. When later running the algorithm adapted for MSF and likely larger graphs, a higher upper bound of 50 was experienced to give better accuracy.

4 Time complexity

The time complexity for our specific MST approximation algorithm can be calculated to have a worst case time complexity $O((W - 1) * (q/W) * 50 * 2 * D)$, where D is the maximum degree of the vertices in the graph, W is the maximum edge weight, q is the limit on the number of vertices to be explored and the constant 50 comes from the greatest value that X can assume.

We then adapted the code from the MST to an MSF which includes changes in our algorithm in the form of adding multiple trees into the equation. This requires us to estimate the number of trees in the graph by approximating the connected components using the max weight W. Subsequently we now run the connected components approximation for 1 to W instead of 1 to $W - 1$, giving us the time complexity $O((W) * (q/W) * 50 * 2 * D) = O(q * 100 * D) = O(qD)$.

In conclusion, the time complexity of our adapted MSF algorithm almost remains the same as the MST one. The differences are the presence of multiple trees which have to be approximated though an additional round of connected components approximation.

References

[1] Artur Czumaj and Christian Sohler. Estimating the weight of metric minimum spanning trees in sublinear time. *SIAM Journal on Computing*, 39(3):904–922, 2009.

5 Appendix

Algorithm 2 ApproxMSTWeight(G, ε)

for $i = 1$ to $W - 1$ **do**
    Compute estimator $\hat{c}^{(i)}$ for $c^{(i)}$
    Output MST $= n - W + \sum_{i=1}^{W-1} \hat{c}^{(i)}$
end for

Algorithm 3 msfApproxWeight(w, n, q)

Input: Maximum weight, Number of vertices, Number of explorable vertices
Output: The estimated weight of MSF

for $i = 1$ to $w - 1$ **do**
    ccsum += ccapprox(i, w, n, q)
end for
trees = ccapprox(w, w, n, q)
if $trees < 1$ **then**:
    trees = 1
end if
Return MSF $= n - (W * trees) + ccsum$

Algorithm 4 ccapprox(currentW, w, n, q)

Input: Current weight, Maximum weight, Number of vertices, Number of explorable vertices
Output: The estimated number of connected components with with edgeweights at most currentW

$b\_sum \leftarrow 0$
$lim \leftarrow 1$
if $w > o$ **then**:
    $lim \leftarrow q/w$
end if
for $i = 0$ to $lim - 1$ **do**
    $x \leftarrow \lfloor 1.0/\text{random}(0, 1) \rfloor$
    if $x > 50$ **then**
        $x \leftarrow 50$
    end if
    $b\_sum \leftarrow b\_sum +$ bfs(randomInteger$(0, n - 1)$, currentW$, x)$
end for
return $\frac{n}{\text{lim}} \times b\_sum$

Algorithm 5 Breadth-First Search (BFS)

```
1: function BFS(root, w, x)
2:     node_queue ← Queue
3:     node_queue.push(root)
4:     seen_nodes ← [root]
5:     node_count ← 0
6:     while (node_queue NOT empty) ∧ (node_count < x) do
7:         n ← node_queue.pop()
8:         node_count ← node_count + 1
9:         neighbours ← get_neighbours(n)
10:        for each node in neighbours do
11:            if (node.weight ≤ w) ∧ (node ∉ seen_nodes) then
12:                seen_nodes.append(node)
13:                node_queue.push(node)
14:            end if
15:        end for
16:    end while
17:    if node_queue is empty then
18:        return 1
19:    else
20:        return 0
21:    end if
22: end function
```