MSF Project

Jesper Öberg, Alexander Willemsen, Erik Stolpe

December 5, 2022

Kattis submission ID: 9988215
(Passes all test cases with 1623 points)

1 Introduction

In most cases, a graph algorithm will always be $\Omega(|V| + |E|)$, since one would have to process the whole input information. However, when considering approximation algorithms we might not need the full information of the graph to make a good estimate. In this project we have developed a sub-linear graph algorithm for estimating the weight of a minimum spanning forest (MSF) of a graph with weighted edges. The MSF is a generalization of a minimum spanning tree for disconnected graphs. It is the union of the minimum spanning trees for each connected component of a graph.

2 Algorithm

Our algorithm is based on an existing algorithm for estimating the weight of a minimum spanning tree[CS10], which we have expanded to work for a minimum spanning forest. That is, we have generalized the algorithm for disconnected graphs. The implemented algorithm transforms the problem of estimating the weight of the tree into estimating the number of connected components in the graph when removing edges that exceed a certain weight. By considering the amount of components of different weights, we can approximate the number of edges of each weight necessary to span the whole graph. The formula is given by

$$MST = N - W + \sum_{i=1}^{W-1} c_i$$

where N is the number of vertices, W is the maximum weight among edges, and c_i is the number of connected components in the graph when discarding all edges of weight greater than i. The formula is explained in more detail in Section 4.1.

To solve this transformed problem, we need a way to estimate the different c_i. We do this by repeatedly propagating BFS (Breadth-first search) on the graph starting from randomly chosen vertices. When estimating components c_i, we naturally only consider edges of weight at most i. We also use a random cap X on our BFS that changes with every search. The distribution of this cap is carefully chosen to make our estimator of c_i unbiased. (More details in Section 4.2.) If a BFS exhausts all nodes within the component before reaching the cap we consider this a success, otherwise a fail. Finally, we calculate the estimator

$$\hat{c_i} = N\frac{s_{success}}{s}$$

where $s_{success}$ and s are the number of succeeded and total number of BFS attempts. The main tuning parameter for modulating the accuracy of the estimated total weight in this algorithm is s.

In order to solve the MSF problem, we adapt the formula for MST to work for disconnected graphs in the following way,

$$MSF = MST - W(c_W - 1)$$

where the subtracted term emerges from the derivation of the MST formula when considering disconnected graphs. Note that this term is always 0 in a connected graph since $c_W - 1 = 0$, and that this is not the case for disconnected graphs since $c_W > 1$. A handwavy explanation for this term is that it is the weight of edges that are needed to connect the components spanned by edges of maximum weight; but there are no such edges in our disconnected graph since the components spanned by edges of maximum weight are disconnected. Further analysis on why this term emerges is presented in Section 4.1.

By using the BFS methodology to calculate the estimators for each c_i, and using the above MSF formula; we are able to calculate a (1 ± 0.1)-approximation of the weight of a MSF given a sufficiently large sample size s.

3 Work progress

The following bullet list gives a rough overview of the work done during the project. We will further outline the implementation and testing done for each point in the rest of the section.

- Implemented base algorithm[CS10] for connected graphs only
- Expanded to disconnected graphs (MSF instead of MST)
- Experimenting with number of samples needed to reach accuracy
- Implemented memory for Kattis queries to reuse in BFS
- Reusing start nodes for BFS

3.1 Original algorithm

In order to implement the original algorithm three main functions were needed. The main function was for estimating the number of components c_i. This function uses a BFS search, and needs to be able to query nodes from Kattis. These three were separated into one function each; querying information about nodes from Kattis, doing the BFS search, and estimating number of components. The implementation for querying nodes from Kattis is based on the provided code from Canvas, and the implementation for estimating c_i was largely inspired from the corresponding pseudocode in [CS10]. This implementation passed the first 6 test cases on Kattis. In these attempts, the number of BFS per c_i was set to a constant 3000 to make sure the algorithm worked.

3.2 Expanding to MSF

To adapt the original algorithm to work for the minimum spanning forest problem, our main approach was to closely investigate how the full algorithm for estimating a MST worked in [CS10]. We arrived at the conclusion that we would not need to alter anything related to estimating the number of components, since how that was done was not related to calculating the actual minimum weight. Thus, examining their presented MST formula closer seemed like a good start. Our intuition was that a MSF should have smaller weight than a MST if we consider a connected graph for the MST, and then remove some edges to make it disconnected for the MSF. The weights of the edges we removed would not be in the MSF, but they would be in the MST.

In the referenced paper, the authors give a short example to motivate the formula presented, but no more. To understand it, we tried larger examples and looked for patterns within those. After some work, it became evident that the formula stated in the paper was a simplified version of another formula, which could only be simplified in that way assuming the graph was connected. Thus, when removing that assumption, we arrived at a formula that generalized to both MST and MSF. The formula and corresponding analysis is presented in Section 4.1. This version of the algorithm was able to pass all the first 12, mandatory test cases, with s set to some high constant value ≥ 1000.

3.3 Tuning parameters and optimizations

In our attempt to reduce the number of Kattis queries while keeping the MSF approximation within the required bounds, we tried a variety of optimization strategies.

We spent the majority of our time tuning the number of samples for estimating each c_i. Since we started at $s \geq 1000$ for safety, we reduced it to ≈ 600 per c_i, for which some of our Kattis submissions managed to pass all 32 test cases (there was some variance), with a score of about 1400. Considering the MSF formula, we noted that the forest term $W(c_W - 1)$ could possibly account for a large part of the estimated value. Thus, we increased the number of queries for estimating c_W, and reduced it further for all other c_i. With this approach, we managed to decrease s to approximately 400, while keeping it higher for c_W, at 700. Again, with some variance, with this we managed to pass all test cases with a score varying between 1500 and 1600. At this point, we started looking at alternatives to manual tuning for setting the value of s. One such approach we tried was to linearly decrease the number of queries used per c_i with increasing i since we reasoned that there would be more components of smaller size, which would attribute more to the total weight and thus we should estimate those more carefully. After tuning the linear decrease we managed to score 1623, passing all test cases. With this linear decrease, s would start at 500 for c_1 and decrease linearly to 200 for c_{W-1}, for c_W we kept a sample size of 700.

Some minor attempts were made to scale the number of queries to the actual "maxQueries" provided by Kattis. By performing a statistical analysis of the expected value of X we estimated the numbers of node queries used for each BFS. E[X] was in fact unbounded when Y is unform on (0, 1), but by changing the interval of Y to (1e-10, 1) the expected value becomes E[X] ≈ 20. However, the experimentation ended up with the approximations being too poor to pass all test cases. When using values proportional to (but greather than) maxQueries we still had to go quite far over the value to actually pass all test cases, not giving score increases compared to the previous methods.

Two optimizations we made were to cache the results of Kattis queries, and reduce the variance of X (the exploration cap on the BFS search). Caching queries turned out to not have much of an impact on our score. By testing it on the provided cycle graph, we realized the number of vertices in Kattis test cases were probably too large for the caching to almost ever be used. We were simply touching so few of the total nodes that the chance of touching the same one twice was very minor. Tuning the variance of X didn't seem to have much of an effect either. The variance was tuned by essentially limiting the maximum value of X.

Finally, trials were ran at reusing explored nodes for future BFS attempts. The idea was to build up large components with BFS at large weights, and then being able to reuse their bounded lower weight components for estimated c_i:s at lower values of i. This method seemed promising, but was not really explored enough to yield any results prior to the end of the project period, though.

4 Correctness

4.1 Formula for MST and MSF

As stated in Section 2, our algorithm for estimating the weight of a minimum spanning forest is based on the existing algorithm presented by Czumaj and Sohler [CS10]. In the paper, they reason shortly about the formula for the MST of a connected graph G based on estimators of the number of components c_i, in subgraphs spanned by edges of weights up to i, denoted G_i. They arrive at the following formula,

$$MST = N - W + \sum_{i=1}^{W-1} c_i, \tag{1}$$

where N is the number of vertices in G, and W is the maximum edge weight. To be able to understand how to alter the formula to calculate the weight of a minimum spanning forest of a disconnected graph, it is crucial to understand how it was derived to calculate the weight of a

minimum spanning tree in a connected graph, first.

We begin with an example, let G be a connected graph with a maximum edge weight of $W = 4$. The idea is to count the number of edges of each weight that need to be in the graph for it to be connected. First, we need to know how many edges of weight 1 are needed to connect the vertices of the graph. This is $N - 1 - (c_1 - 1)$, where $N - 1$ is the number of edges of weight at least 1 needed to connect the vertices, and $(c_1 - 1)$ is the number of edges with more than weight 1 needed to connect the components in G_1, thus we are left with exactly the number of edges of weight 1 that needs to be included in the MST. The number of edges of weight 2 needed in the MST follow the same pattern, $c_1 - 1 - (c_2 - 1)$. That is, the number of edges of weight at least 2 required to connect the components in G_1 is $c_1 - 1$, and the number of edges of more than weight 2 required to connect the components in G_2 is $(c_2 - 1)$. Thus, we get the number of edges of exactly weight 2 required in the MST. We continue in this fashion; the number of edges required of weight 3 is $c_2 - 1 - (c_3 - 1)$, and the number of edges of weight 4 is $c_3 - 1 - (c_4 - 1) = c_3 - 1$ since $c_4 = 1$. Finally, we get the weight of the MST of G as,

$$1(N-1-(c_1-1))+2(c_1-1-(c_2-1))+3(c_2-1-(c_3-1))+4(c_3-1-(c_4-1)) = N-4+c_1+c_2+c_3.$$

Considering that $N = c_0$ (the number of components in a graph spanned by edges of weight at most 0 is the number of vertices), this can be written as

$$MST = \sum_{i=1}^{W} = i(c_{i-1} - 1 - (c_i - 1)). \tag{2}$$

Furthermore, this can be simplified to the form in (1) by noting that the ones cancel out for all $i \neq W$, since $c_W = 1$ in a connected graph. Additionally, only one occurrence of each c_i remains (except for c_W which disappears).

Each term in (2) can be interpreted as "$i \times$ the exact number of edges of weight i, that connect as many components in $G_{(i-1)}$ as possible". Where, as in the example, $c_{i-1} - 1$ is the exact number of edges of weight $\geq i$, that must be used to connect the components in $G_{(i-1)}$, and $c_i - 1$ is the exact number of edges of weight $> i$, that must be used to connect the components in G_i. Note that in total, these are the edges of each weight, that must be in the MST for it to be connected.

Finally, let's consider what happens when G is a disconnected graph. Note that the formula in (2) simplifies to the formula in (1) only when $c_W = 1$. But, in a disconnected graph, the number of components spanned by edges of maximum weight is > 1. Thus, it can not be simplified in the same way. In fact, the formula in (2) works for both MST and MSF. If $c_W = 1$ it simplifies to the formula in (1), and otherwise it calculates the weight of a MSF. However, it can be simplified to,

$$MSF = N - W + \sum_{i=1}^{W-1} c_i - W(c_W - 1) \tag{3}$$

$$= N - c_W \times W + \sum_{i=1}^{W-1} c_i. \tag{4}$$

To conceptually understand why the MST formula can be simplified to (1), and why the MSF formula can not, we need to consider the last term in (2), when $i = W$. In a connected graph G, the subgraph G_{W-1} will only be missing edges of weight W. To connect the components of G_{W-1} we need $c_{W-1} - 1$ edges, all of those must have weight W, and since we know G is a connected graph, we know those edges exist. Thus, we arrive at $W(c_{W-1} - 1)$ weight added to the MST. However, in a disconnected graph U, with k connected components, when we add the final edges of weight W to connect the components in U_{W-1}, all components can not be connected. In fact, there are exactly k components that did not get connected! So, the edges of weight W that are "supposed" to connect these components do not exist. Thus, we remove a weight equal to $W(k-1)$ from the MSF.

4.2 Estimation of c_i

Let us consider some weight i. When only considering edges with edge weights at most i, the graph will have c_i connected components. We estimate this number using BFS search with a variable

cap sampled from the random variable X, which we choose it according to

$$X = \frac{1}{Y}$$
$$Y = uniform(0, 1)$$

We will prove that this choice of X leads to an unbiased estimator $\hat{c_i}$ of the real value c_i. That is, $\mathrm{E}[\hat{c_i}] = c_i$.

Lemma 1 *$Pr[X \geq k] = \frac{1}{k}$ for all non-negative k*

Proof: Y is uniformly chosen between 0 and 1. This means that the probability density function of Y is

$$PDF_Y(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } 0 < x < 1 \\ 0 & \text{if } x \geq 1 \end{cases}$$

$$\Rightarrow Pr[Y \leq t] = \int_{-\infty}^{t} PDF_y(x)dx = \begin{cases} 0 & \text{if } t < 0 \\ t & \text{if } 0 \leq t \leq 1 \\ 1 & \text{if } t > 1 \end{cases}$$

We now consider the the variable $\mathrm{X} = \frac{1}{Y}$.

$$Pr[X \geq k] = Pr[\frac{1}{Y} \geq k] = Pr[Y \leq \frac{1}{k}] = \frac{1}{k}$$

Where the last equality holds because $0 \leq \frac{1}{k} \leq 1$ for all non-negative values k.

Lemma 2 *$E[\hat{c_i}] = c_i$*

Proof: Let b_i be a variable corresponding to the outcome of a specific BFS, starting at node u_i.

$$b_i = \begin{cases} 0 & \text{if BFS failed to succeed} \\ 1 & \text{if BFS succeeded} \end{cases}$$

Since we initialise our BFS in a random node of the graph, the chance of picking a node in a specific component is $\frac{|C|}{N}$ where |C| is the number of nodes in that component. The chance of fully exploring the component we picked is equal to the chance that our cap is greater or equal to the number of nodes in the component, which by Lemma 1 is

$$Pr[X \geq |C|] = \frac{1}{|C|}$$

Note that the chance of successfully exploring any component is actually equal! As the components get bigger, the chance of picking a node in the component increases, but the chance of our cap being large enough to finish searching the whole component decreases correspondingly. When summing over all possible components c_i, the expected value of b_i then becomes

$$E[b_i] = \sum_{components} Pr[u_i \in C]Pr[X \geq |C|] = \sum_{components} \frac{|C|}{N}\frac{1}{|C|} = \sum_{components} \frac{1}{N} = \frac{c_i}{N}.$$

Finally, recalling our formula for $\hat{c_i}$,

$$\hat{c_i} = N\frac{s_{success}}{s}$$
$$E[s_{success}] = E[\sum_i b_i] = sE[b_i] = s\frac{c_i}{N}$$

$$E[\hat{c_i}] = E[N\frac{s_{success}}{s}] = \frac{N}{s}E[s_{success}] = c_i$$

Which is exactly what we wanted to prove. This means our estimation of the components c_i are unbiased.

5 Time complexity

In the paper were the original MST algorithm is presented [CS10] they derive the time complexity to $O(D \cdot W^3 \log n/\epsilon^2)$, where D is the maximum degree of the input graph, and ϵ is the approximation error. Since we are simply doing one additional iteration which estimates c_W, and D is bounded by some constant (as given by Kattis), the time complexity of our algorithm is $O(W^3 \log n)$ given that we estimate with a fixed $\epsilon = 0.1$ error.

References

[CS10] Artur Czumaj and Christian Sohler. *Sublinear-time Algorithms*, pages 41–64. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.