6.896 Sublinear Time Algorithms	September 23, 2004
Lecture 4	
Lecturer: Ronitt Rubinfeld	*Scribe: Seth Gilbert*

Overview

- Approximating the weight of the minimum spanning tree
  - *Assume from last lecture*: **approx_num_cc**(G, ϵ) approximates the number of connected components in an n node graph, G, with degree $\leq d$ in time $O(d/\epsilon^4)$.
  - *This time*: Determine the (approximate) weight of a minimum spanning tree (MST) for an n node graph, G, with degree $\leq d$.
  - *Plan*: Characterize minimum spanning tree in terms of number of connected components of certain subgraphs. Use **approx_num_cc** to estimate these values.
- Testing Bipartiteness
  - *Adjacency Matrix Model*: The adjacency matrix model is defined.
  - *Problem Statement*: The problem of testing for bipartitness is presented.
  - *Sketch*: We sketch the algorithm to test for bipartitness and its proof.

1 Approximating the weight of the minimum spanning tree

1.1 Problem Statement

Assume you are given:

- a graph, $G = (V, E)$, of n nodes and m edges with degree $\leq d$,
- G is connected,
- each edge $(i, j) \in E$ is assigned an integer weight $w_{ij} \in \{1, \ldots, w\}$, and
- G is represented as an adjacency list augmented with weights for each edge.

We define the minimum spanning tree (MST) weight as follows:

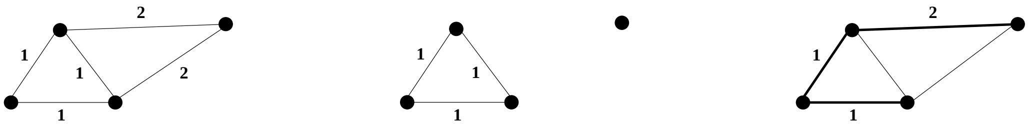
$$MST(G) = \min_{T \text{ spans } G} w(T) = \sum_{(i,j) \in T} w_{ij}$$

Our goal is to output $\widehat{MST}(G)$ such that:

$$(1 - \epsilon)MST(G) \leq \widehat{MST}(G) \leq (1 + \epsilon)MST(G)$$

Comments

- Notice that the resulting algorithm provides no indication of how to actually construct the minimum spanning tree.
- Notice that $n - 1 \leq MST(G)$, since each edge has weight at least 1, and there are $n - 1$ edges in a minimum spanning tree.
- Notice that $MST(G) \leq w \cdot (n - 1) < \infty$, since G is connected, each edge has weight at most w, and there are $n - 1$ edges in a minimum spanning tree.



(a) (b) (c)

Figure 1: Figure (a) is a simple example graph. Figure (b) represents the subgraph of (a) consisting of edges of weight 1. Figure (c) emphasizes a minimum spanning tree of weight 4.

1.2 Intuition

We want to relate the weight of a minimum spanning tree to the number of connected components in certain subgraphs of the original graph. As motivation, consider the following algorithm to find a minimum spanning tree:

1. Sort the edges by weight.
2. Start with $i = 1$.
3. Add as many edges as possible of weight i, while ensuring that no cycle is created.
4. If there are $n - 1$ edges, stop.
5. Let $i = i + 1$.
6. Go to step 3.

Notice that the weight 1 edges added in step 3 form a forest of connected components. If there are k such connected components, then the resulting spanning tree has $k - 1$ edges of weight > 1.

Consider as an example the example in Figure 1(a). Further, consider the subgraph $G^{(1)}$, the subgraph of G including only edges of weight ≤ 1, as depicted in Figure 1(b). Let $C^{(1)}$ be the number of connected components in $G^{(1)}$. In this case, $C^{(1)}$ is equal to 2.

The $MST(G)$ has $C^{(1)} - 1 = 1$ edge of weight 2. All the other edges in the minimum spanning tree have weight 1. That is:

$$\begin{aligned} MST(G) &= (2 \cdot C^{(1)} - 1) + (n - 1 - C^{(1)}) \\ &= (C^{(1)} - 1) + (n - 1) \\ &= n - 2 + C^{(1)} \\ &= 4 - 2 + 2 = 4 \end{aligned}$$

Notice that the minimum spanning tree, depicted in Figure 1(c) contains 2 edges of weight 1 and 1 edge of weight 2, as described.

1.3 Main Lemma

Define $G^{(i)}$ and $C^{(i)}$ as follows:

$$\begin{aligned} G^{(i)} &= \text{subgraph of } G \text{ including edges of weight } \leq i \\ C^{(i)} &= \text{number of connected components in } G^{(i)} \end{aligned}$$

Lemma 1 *For a graph, G with n nodes and edges with integer weights in $\{1, \ldots, w\}$:*

$$MST(G) = n - w + \sum_{i=1}^{w-1} C^{(i)}$$

Proof Let α_i be the number of edges of weight i in any minimum spanning tree of G. It is known that all minimum spanning trees have the same number of edges of weight i. Consider the following facts:

- The total number of edges is $n-1$.
$$\sum_{i=1}^{w} \alpha_i = n-1$$

- The number of edges of weight $> \ell$ is equal to the number of connected components in $G^{(\ell)}$ minus 1.
$$\sum_{i=\ell+1}^{w} \alpha_i = C^{(\ell)} - 1$$

- The number of edges of weight w is equal to the number of connected components in $G^{(w-1)}$ minus 1.
$$\alpha_w = C^{(w-1)} - 1$$

We therefore calculate the weight of the minimum spanning tree, multiplying the number of edges of weight i in the minimum spanning tree times i, as follows:

$$\begin{aligned}
MST(G) &= \sum_{i=1}^{w} i \cdot \alpha_i \\
&= \sum_{i=1}^{w} \alpha_i + \sum_{i=2}^{w} \alpha_i + \sum_{i=3}^{w} \alpha_i + \cdots + \sum_{i=w}^{w} \alpha_i \\
&= (n-1) + (C^{(1)} - 1) + (C^{(2)} - 1) + \cdots + (C^{(w-1)} - 1) \\
&= n - w + \sum_{i=1}^{w-1} C^{(i)}
\end{aligned}$$

■

1.4 MST Approximation Algorithm

[Chazelle, Rubinfeld, Trevisan 2001]

MST_approx(G, ϵ, w)
  `For` $i = 1$ `to` $w-1$ `do`
    $\widehat{C}^i =$ **approx_num_cc**($G^{(i)}, \epsilon/w$)

Output:

$$\widehat{MST}(G) = n - w + \sum_{i=1}^{w-1} \widehat{C}^i$$

Runtime:

$$O\left(\frac{dw^5}{\epsilon^4}\right)$$

Notice that the algorithm makes w calls to **approx_num_cc**, each of which has running time dw^4/ϵ^4. One detail: rather than extract $G^{(i)}$ from G, we simply modify **approx_num_cc** so that it ignores edges of weight $> i$.

Total additive error: $\leq w \cdot \frac{\epsilon}{w} n \leq \epsilon n$. Since $MST(G) \geq n - 1 \geq 1$, we conclude that:

$$\widehat{MST}(G) \leq MST(G) + \epsilon \leq MST(G)(1 + \epsilon)$$

Similarly, we conclude that:

$$MST(G)(1 - \epsilon) \leq MST(G) - \epsilon \leq \widehat{MST}(G)$$

This results in a ϵ approximation of the weight of the minimum spanning tree:

$$MST(G)(1 - \epsilon) \leq \widehat{MST}(G) \leq MST(G)(1 + \epsilon)$$

Comments

- It is possible to improve the running time to:

$$O\left(\frac{dw}{\epsilon^2} \log\left(\frac{dw}{\epsilon}\right)\right)$$

- It is possible to solve even if the weights are not integral, as long as the maximum weight is bounded by w.
- It is possible to solve if the average degree is d, even if the maximum degree is larger.
- There is a lower bound indicating that finding the approximate weight of the minimum spanning tree within a multiplicative factor requires:

$$\Omega\left(\frac{dw}{\epsilon^2}\right)$$

2 Testing Bipartiteness

2.1 Adjacency Matrix Model

- Developed by [Goldreich, Goldwasser, Ron 1997].
- Graphs on n nodes represented as an adjacency matrix.
- $f_G : V \times V \rightarrow \{0, 1\}$
- $f_G(u, v) = 1$ if $(u, v) \in V$, else 0
- $f_G(u, v) = f_G(v, u)$
- 1 step equals 1 query to f_G.
- Distance is defined as follows:

$$d(G_1, G_2) = \frac{|\{u, v \in V : f_{G_1}(u, v) \neq f_{G_2}(u, v)\}|}{n^2}$$

2.2 Problem Statement and Algorithm

Let P equal the class of bipartite graphs. That is, the class of graphs where there exists a partition (V_1, V_2) of the vertices V such that for all $u, v \in V_1$ or $u, v \in V_2$, $(u, v) \notin E$.

Equivalently, G is ϵ-far from bipartite if for all (V_1, V_2) partitions of V, there are $\geq \epsilon n^2$ violating edges. That is,

$$|\{(u,v) \in E : u, v \in V_1 \text{ or } u, v \in V_2\}| \geq \epsilon n^2$$

test-bipartite [GGR]

1. Pick m nodes at random:
$$m = \Theta\left(\frac{\log \frac{1}{\epsilon}}{\epsilon^2}\right)$$
2. For all pairs u, v in the sample, query (u, v).
3. Test subgraph for bipartiteness using BFS.
4. If bipartite, pass. Else, fail.

Runtime, in queries or time:

$$O\left(\frac{\log^2 \frac{1}{\epsilon}}{\epsilon^4}\right)$$

Theorem 2 *If the graph is bipartite, then the algorithm outputs pass. If the graph is ϵ-far from bipartite, then the algorithm outputs pass with probability* $2/3$.

Proof idea: It is clear that if the graph is bipartite, then the algorithm outputs pass, since all subgraphs of a bipartite graph are bipartite.

We therefore assume that the graph is ϵ-far from bipartite. We divide the sample into two parts, U and S. We consider an arbitrary partition of U, u_1 and u_2, and use this to induce a partition on the entire graph, C_1 and C_2. Any partition of U results in C_1 and C_2 having a significant number of violating edges. We can then see that as long as S is large enough, it will contain some of those violating edges. Thus the algorithm, when attempting to determine a bipartite partition of U and S together, will discover the violation and fail.