26 Graph Sparsificiation

Big graphs, like any big data, can be difficult to process. How can we represent them in a way that takes up far less space, but still allows similar analysis.

For many years, it was assumed that really large graphs would remain very sparse so that $|E| \leq c|V|$ for some constant like 20. That is, as the number of nodes grew the number of edges would grow linearly. However, in practice it is usually more like $|E| = |V|^{1+c}$ for a small constant like $c = 0.1$ (but sometimes as high as $c = 0.5$). That is the number of edges, grows polynomially faster than the number of vertices.

Can we reduce to some set E' such that $|E'| = |V| \log |V|$?

26.1 Edge Sampling

The simplest way to sparsify a graph is to sample some of its edges. However, uniformly sampling the edges (so each is equally likely to be chosen) can cause problems. Some edges are much more important than others. For instance if you want to preserve the graph distance or connectedness, then some edges that bridge two communities are very important for these properties. These properties are then integrally related to the strength of various graph clusterings. Next we see how to bias the probability we sampled edges to preserve this.

Let the *degree* of a vertex $v_i \in V$ be d_i. Now sample each edge (i, j) with probability

$$p_{i,j} = \min\{1, t/\min\{d_i, d_j\}\}.$$

Then re-weight the edges that are sampled to weight $1/p_{i,j}$. This keeps the expected weight of all edges (and all edges around any vertex, and actually each individual edges) the same as in the original graph.

This sampling-probability process generalizes to weighted graphs. If the edges originally have weight $w_{i,j}$, then set $d_i = \sum_{e_{i,j}} w_{i,j}$.

Importantly, this keeps all edges of nodes with degree at most t, since they are sampled with probability 1. All other edges are kept proportional to t/d_i for d_i being the min degree endpoint.

Lemma 26.1.1. *The output of this procedure has* $\mathbb{E}[|E|] < t|V|$.

Proof. To see this, assign each edge to the node with smaller degree. Then for each node, we expect to select at most t of edges assigned to it. □

What should the value t be? Setting $t = (1/\varepsilon^2) \log |V|$ preserves any "cut" of the graph within ε. That is given a decomposition of the vertices V into two sets S, T, then recall the *cut* $C(S, T)$ is the subset of edges that have one vertex in each of S or T. Normalize all edge weights w_e so that $\sum_{e \in E} w_e = 1$. Then for any cut, the resulting graph (with weights w'_e for all $e \in E$) has

$$\left| \sum_{e \in C(S,T)} w_e - \sum_{e \in C(S,T)} w'_e \right| \leq \varepsilon,$$

for any S, T partition of V. Note in the sparsified graph H, many of the edge weights w'_e will be 0, while others may be much larger. This cut property is a key property needed for spectral clustering and in finding communities.

The analysis of this result is similar in spirit to column sampling **L16**.

Laplacian. We next want to generalize this notion of the graph cut property. The key tool we need is the Laplacian. Recall that for a graph G with adjacent matrix A_G and degree (diagonal) matrix D_G that the (unnormalized) *Laplacian* matrix is $L_G = D_G - A_G$.

Now in particular we want a sparse graph H such that

$$\|L_G - L_H\|_2 \leq \varepsilon.$$

or equivalently for all $x \in \mathbb{R}^{|V|}$

$$(1-\varepsilon)x^T L_G x \leq x^T L_H x \leq (1+\varepsilon)x^T L_G x.$$

Note how similar this is to the above claimed result about preserving graph cuts (think of x as being a direction you project the matrix onto and then cut, like in Spectral clustering), but only when x is in $\{0,1\}^{|V|}$. So this goal is much more general (and powerful).

The above sampling procedure does not (as far as is known) preserve this Laplacian property.

26.1.1 Effective Resistance

To attain this Laplacian property we use ideas from ideas from circuits! Let $R_{\mathsf{eff}}(e)$ is the *effective resistance* between end points $e = (u, v)$. That is, consider each edge has resistance described by its weight. Then if one puts one unit of current at u and retrieves it from v then the measured resistance is the the *effective resistance* $R_{\mathsf{eff}}(u, v)$.

Consider graph with (u, v), (u, a), (a, v), each of strength 1. Then $R_{\mathsf{eff}}(u, v) = 1/(1/2 + 1/1) = 2/3$.

We now change our sampling scheme slightly to sample edges with probability p_e, a value that is proportional to $R_{\mathsf{eff}}(e)$. And weight the selected edges as $1/p_e$; the rest get weight 0 (or are ignored). Again we take $O((1/\varepsilon^2)|V| \log |V|)$ edges total (with replacement and add weights).

There are some recent papers (2011) that improve the runtime to about $O(|V| \log |V| \log(1/\varepsilon))$. The idea is as follows:

- Sample several times. Each additional time only keep edges which were not in original sample. Through the use of *Rayleighs Monotonicity Property* this can be shown to only improve the bound.
- Remove degree $1, 2$ nodes to create a graph G_2 (contracting edges). Construct "rough" approximation H_2 of G_2. Again remove degree $1, 2$ nodes to create graph G_3. Repeat for $\log |V|$ rounds.
- This process can be sped up with a series of subtle tricks to get the stated bound.

Aside: My impression is that these techniques are not currently practical, but many believe it is on the very of being practical and is a good direction to work on. It may be in the next 5 or so years. But, always be wary of $O(1/\varepsilon^2)$ factors – they are usually large and annoying.

26.2 Spanners

We are now back to a metric space! Start with a metric $\mathbf{d}_G$ for all $a, b \in V$. Often $\mathbf{d}_G(a, b)$ is the shortest Euclidean path in Euclidean graph where all nodes V are embedded in $\mathbb{R}^d$, but we can only walk on edges. *Aside: Road networks use very similar models to this–these approaches give a first approximation to the very fast routing algorithms used in Google Maps and Bing Maps.* So also assume $V \subset \mathbb{R}^d$. Typically $d = 2, 3$ (but we only really need that V has low *doubling dimension*). Sometimes we start with G being a complete graph.

Specifically, for $G = (V, E)$ then $\mathbf{d}_G(a, b) = \|a - b\|$ if $(a, b) \in E$ otherwise

$$\mathbf{d}_G(a, b) = \min_{c \in N(a)} \|a - c\| + \mathbf{d}_G(c, b)$$

where $N(a)$ is the set of all *neighbors* of a, that is all nodes connected by an edge to a.

We say that H is a t-*spanner* of G if for all a, b we have

$$1 \leq \mathbf{d}_H(a,b)/\mathbf{d}_G(a,b) \leq 1+t.$$

The goal is to find a graph H such that has:

- small number of edges
- total weight (length) of all edges is small
- maximum degree of any one node is small

There are several algorithms. We will outline a few:

- **Greedy:** Start with no edges. Sort pairs by distance (small to large). If the next edge in list (a, b) has too much error (more than $1 + t$), then add edge to H.
- **Cone-Based:** Around each point divide space into $k \geq 6$ cones. Each cone defines a set of directions. Find the closest point in each cone and connect. If we set angle of cone to be $\theta = 2\pi/k$ radians, then is a t-spanner with $t \leq 1/(1 - \sin(\theta/2))$.
- **WSPD:** A t-*well-separated pair decomposition (WSPD)* is a set of pairs $\{(A, B)\}$ such that (i) $A, B \subset V$, (ii) each $(a, b) \in E$ in exactly one pair. (iii)

$$\min_{a\in A, b\in B} \mathbf{d}(a,b) > t \max\{\max_{a_1,a_2\in A} \mathbf{d}(a_1,a_2), \max_{b_1,b_2\in B} \mathbf{d}(b_1,b_2)\}.$$

  Amazingly we can create an s-WSPD of size $O(t^d|V|)$ (independent of E) in time $O(|V| \log |V| + t^d|V|)$ for points in $\mathbb{R}^d$. And its not that hard. It uses a *compressed quad tree*, which is basically a quad tree that compresses nodes with only 1 non-empty child. It is built recursively from top down, dividing a big cell into 2^d parts. If two cells satisfy the above t-property (iii), then make a pair. Otherwise, recursively check on all pairs of split(A) and split(B).

  Finally, to get a t-spanner, select one (any one) representative point from each of the sets in a pair, and connect across pairs.