Algorithms for Massive Data Science **Lecture date: March 25, 2019**
Instructor: Ben Moseley **Scribe: Rudy Zhou**

1 Introduction

Recall the generic partitioning approach for developing distributed algorithms:

- Partition the input evenly across all machines.
- Each machine constructs a coreset given its partition, and sends that coreset to a designated machine.
- The designated machine uses all the coresets to construct the final solution.

In this lecture we will apply this approach to the k-centers problem and the triangle counting problem.

2 k-Centers Clustering

Clustering is a data analysis technique in which we want to group data points together that are similar. We are given n data points along with some notion of similarity between data points. The goal of clustering is to the divide the data into k clusters such that points in the same cluster are similar. The way we will measure similarity is by defining an objective function. In this section, we will consider the k-centers objective function.

The formal definition of k-centers clustering is as follows:

Definition 1 (k-Centers Clustering) *As input we are given a set U of n points, a parameter $k \in \mathbb{N}$ (the number of centers to choose), and a distance metric d on U.*

The output is a set of k centers, say $C \subset U$, where $|C| = k$. We wish to output a set of k centers minimizing $\max_{u \in U} d_C(u)$, where $d_C(u) := \min_{c \in C} d(u, c)$.

2.1 Sequential Approximation Algorithm for k-Centers

Before considering the distributed setting, it will be useful to develop a sequential algorithm for k-centers. Consider the following greedy algorithm:

Algorithm 1 Sequential k-center

```
GreedyKC($U, k$)
1: $C = \emptyset$
2: Add any point $u \in U$ to $C$
3: while $|C| < k$ do
4:     $u = \texttt{argmax}_{v \in U}\ d_C(v)$
5:     $C = C \cup \{u\}$
6: end while
7: Output $C$
```

Lemma 1 GreedyKC *is a* 2*-approximation for the* k*-center problem.*

Proof: Let $C^* = \{c_1^*, \ldots c_k^*\}$ be the centers chosen by the optimal solution achieving objective value r^*.

Let V_i^* denote the points in c_i^*'s cluster in the optimal solution (so V_i^* is the set of points $u \in U$ such that $d(u, C^*) = d(u, c_i^*)$.)

We will use the following fact for the analysis of the algorithm. Fix some V_i^*. For any $u, v \in V_i^*$, by the triangle inequality, we have:

$$d(u,v) \leq d(u, c_i^*) + d(v, c_i^*) \leq 2r^*$$

Now we consider two cases. Suppose C has one point in each V_i^*, say c_i. Then for each $u \in V_i^*$, $d(u, C) \leq d(u, c_i) \leq 2r^*$, because $u, c_i \in V_i^*$. This holds for all V_i^*. Because $V_1^*, \ldots, V_k^*$ form a partition of U, we have a 2-approximation.

Otherwise, there exists some i such that C has two points, say $u, v \in V_i^* \cap C$. Without loss of generality, assume v was chosen after u. We have $d(u,v) \leq 2r^*$. By definition of the algorithm, once we add v to C, all points in U are distance at most $2r^*$ to C. □

2.2 MPC Approximation Algorithm for k-Centers

Now we will combine GreedyKC with our partition approach to give a MPC algorithm for k-centers.

Note that we assume that initially the data is arbitrarily partitioned across all m machines evenly.

Further, we assume that we have oracle access to the distance function d, so we do not need to store the distance function on the machines, and we assume that each point of U can be stored in $O(1)$ space.

Intuitively, our algorithm will be to run GreedyKC on each machine, and to use the resulting centers as a coreset for that partition. Then one machine will gather all the coresets and use them to compute the final clustering.

Algorithm 2 Distributed k-center

```
Distributed-KC($U, k$)
1: Partition $U$ into $m$ equal sized sets $U_1, \ldots, U_m$ where machine $i$ receives $U_i$.
2: Machine $i$ assigns $C_i$ = GreedyKC($U_i, k$)
3: All sets $C_i$ are assigned to machine 1
4: Machine 1 sets $C$ = GreedyKC($\cup_{i=1}^m C_i, k$)
5: Output $C$
```

Theorem 2 Distributed-KC *is a two round MPC algorithm which achieves a* 4*-approximation for the* k*-center problem which communicates* $O(km)$ *amount of data assuming the data is already partitioned across the machines. The algorithm uses* $O(\max\{n/m, mk\})$ *memory on each machine.*

The communication and memory requirements are clear, so it remains to show that Distributed-KC is a 4-approximation.

Lemma 3 Distributed-KC *is a* 4*-approximation for* k*-centers.*

Proof: Let r^* be the optimal objective value.

We first show that for all $i = 1, \ldots, m$, $d_{C_i}(u) \le 2r^*$ for all $u \in U_i$.

Assume for contradiction this does not hold, so there exists i and some $u \in U_i$ with $d_{C_i}(u) > 2r^*$. By definition of GreedyKC, it must be the case that for any $v, v' \in C_i$, $d(v, v') \ge d_{C_i}(u) > 2r^*$, because otherwise we would have added u to C_i.

It follows, $\{u\} \cup C_i$ is a set of $k + 1$ points where each inter-point distance is at least $2r^*$. By the pidgeonhole principle, two points from this set, say $v, v' \in \{u\} \cup C_i$ must be assigned to the same center in the optimal solution, say c^*, so:

$$d(v, v') \le d(v, c^*) + d(v', c^*) \le 2r^*$$

This is a contradiction.

Note that an analogous proof as the previous claim shows that $d_C(u) \le 2r^*$ for any $u \in \bigcup_{i=1}^{m} C_i$.

We combine these two results to show that Distributed-KC is a 4-approximation.

Consider any $u \in U$. If $u \in C_i$ for some i, then by the second claim, $d_C(U) \le 2r^*$.

Otherwise, $u \notin \bigcup_{i=1}^{m} C_i$, so $u \in U_i$ for some i. By the first claim there exists some $c \in C_i$ with $d(u, c) \le 2r^*$, and by the second claim, $d_C(c) \le 2r^*$. Thus we conclude:

$$d_C(u) \le d(u, c) + d_C(c) \le 4r^*$$

□

3 Triangle Counting

Now we will apply the partition approach to another problem: triangle counting.

Definition 2 (Triangle Counting Problem) *The input is an undirected graph $G = (V, E)$ with n vertices and m edges. We want to output the number of triangles (the number of 3-cycles) in G.*

Observe that a naive partition algorithm will not work for this problem, because if the vertices of a triangle are split across multiple machines, then such a triangle will never be counted by an individual machine.

Triangle counting arises in social network analysis, where it is used to compute the *clustering coefficient* of vertices in a graph. The clustering coefficient of a vertex v is defined by:

$$\frac{\#\{(u, v) \in neighbors(v) \mid uv \in E\}}{\binom{deg(v)}{2}}$$

The clustering coefficient can be used to detect "tightly nit" communities (high clustering coefficient) and "structural holes" (low clustering coefficient.)

3.1 Warm Up: Sequential Algorithms for Triangle Counting

A first idea could be to enumerate all subsets of 3 vertices, say $\{u, v, w\} \in \binom{V}{3}$ and check if u, v, w form a triangle. This takes $\Theta(n^3)$ time. Observe that this runtime is "optimal" because in the complete graph, there are $\Theta(n^3)$ triangles, and in order to count them all, we really do need to check all subsets.

However, can we do better if the graph is sparse, which is often the case in practical instances?

Consider the following idea: For each $u \in V$, iterate over all pairs $(v, w) \in neighbors(v)$ and check if u, v, w form a triangle. This takes $\Theta(\sum\limits_{v \in V} deg(v)^2)$ time.

Observe that if all vertices in G have constant degree, then this algorithm takes $\Theta(n)$. We will not formalize this here, but usually the degree distributions of social networks have heavy tails, which gives a runtime of $\Theta(n^2)$.

We can improve on this algorithm even further by defining a total ordering the vertices by their degrees (ties are broken arbitrarily but consistently.) More precisely, for each $u \in V$, only enumerate pairs $v, w \in neighbors(u)$ such that $deg(v), deg(w) > deg(u)$. It can be shown that this algorithm runs in time $\Theta(m^{\frac{3}{2}})$.

Note that all of these algorithms have worst-case runtime $\Theta(n^3)$ in the case of the complete graph, but the latter two have more nuanced runtimes that give much better performance in certain cases.

3.2 MPC Algorithm for Triangle Counting using Overlapping Partitions

We solve the problem of the naive partitioning algorithm missing some triangles by constructing overlapping partitions. We will combine this with the idea of the third sequential algorithm (defining an order on our objects to avoid redundant enumeration.) These two ingredients will give us our MPC algorithm for triangle counting.

More precisely, define a partition of V into ℓ equal sized groups, say $V_1 \cup \dots \cup V_\ell = V$. Then for each (i, j, k) with $1 \le i < j < k \le \ell$, let $W_{ijk} = V_i \cup V_j \cup V_k$. Further, we define $G_{ijk} = (W_{ijk}, E[W_{ijk}])$ be the subgraph of G induced by W_{ijk}. The following lemma is immediate.

Lemma 4 *Every triangle in G is in some G_{ijk}. The number of edges in each G_{ijk} is $O((\frac{n}{\ell})^2)$.*

Thus the high-level of our algorithm is as follows: Suppose we have a machine for each $1 \le i < j < k \le \ell$, say m_{ijk}. For each m_{ijk}, we need to send it the requisite edges to construct the subgraph G_{ijk}. Then each m_{ijk} counts the number of triangles in G_{ijk}, and sends that count to a designated machine. The designated machine then sums up all counts and outputs the result.

Observe that a triangle can be present in multiple G_{ijk}'s. We avoid over-counting by ordering the G_{ijk}'s lexicographically by (i, j, k). Then we count a triangle *only in the lexicographically first partition containing it.*

Example: Suppose a triangle has all three of its vertices in V_2. Then this triangle is contained in G_{ijk} for all (i, j, k) satisfying $2 \in \{i, j, k\}$. However, the lexicographically first G_{ijk} that contains this triangle in G_{123}, so we will only count it there.

Algorithm 3 Distributed Triangle Counting

```
TRIANGLECOUNT($G$).
1: Suppose $G$ is represented by its edge set $E$. Define the random partition $V_1 \cup \cdots \cup V_\ell = V$ such that each $v \in V$ is independently and uniformly assigned to one of $V_1, \ldots, V_\ell$.
2: For each $uv \in E$ (regardless of what machine it is on), send a copy of $uv$ to every $m_{ijk}$ with $u, v \in V_i \cup V_j \cup V_k$.
3: Each $m_{ijk}$ counts the number of triangles on its machine and sends the resulting count to a designated machine.
4: The designated machine sums up and outputs the total count.
```

Lemma 5 *The expected size of any $E[W_{ijk}]$ is $O(\frac{m}{\ell^2})$.*

Proof: Fix any edge $uv \in E$. We have $Pr(u \in W_{ijk}), Pr(v \in W_{ijk}) = \frac{3}{\ell}$. These events are independent, so $Pr(e \in E[W_{ijk}]) = Pr(u, v \in W_{ijk}) = \frac{9}{\ell^2}$.

By linearity expectation, $\mathbb{E}[|E[W_{ijk}]|] = \sum_{e \in E} Pr(e \in E[W_{ijk}]) = \frac{9m}{\ell^2}$. □

Thus in expectation (and with high probability using Chernoff bounds) each m_{ijk} requires $O(\frac{m}{\ell^2})$ memory and there are $O(\ell^3)$ machines.

We summarize the performance of TRIANGLECOUNT with the following theorem:

Theorem 6 *Let $\ell \leq \sqrt{m}$. Then* TRIANGLECOUNT *uses $O(\ell^3)$ machines, and with high probability, the total work done over all machines in $O(m^{\frac{3}{2}})$.*

Proof: With high probability, each m_{ijk} stores a subgraph with $O(\frac{m}{\ell^2})$ edges. Triangle counting on each machine takes $O(\frac{m^{\frac{3}{2}}}{\ell^3})$ using the third sequential algorithm. Summing over all $O(\ell^3)$ machines gives total work $O(m^{\frac{3}{2}})$. □

Thus we have given a MPC algorithm for triangle counting that does the same amount of work as the sequential algorithm.