CIS 700: “algorithms for Big Data”

Lecture 10: Massively Parallel Algorithms

Slides at http://grigory.us/big-data-class.html

Grigory Yaroslavtsev

http://grigory.us
Computational Model

- **Input**: size $n$
- $M$ machines, space $S$ on each ($S = n^{1-\epsilon}$, $0 < \epsilon < 1$)
  - Constant overhead in total space: $M \cdot S = O(n)$
- **Output**: solution to a problem (often size $O(n)$)
  - Doesn’t fit on a single machine ($S \ll n$)

Input: size $n$ \implies \text{Output: size } O(n)$
Computational Model

• Computation/Communication in $R$ rounds:
  – Every machine performs a near-linear time computation $\Rightarrow$ Total running time $O(n^{1+o(1)}R)$
  – Every machine sends/receives at most $S$ bits of information $\Rightarrow$ Total communication $O(nR)$.

Goal: Minimize $R$.

Ideally: $R = \text{constant.}$

$M$ machines

$O(S^{1+o(1)})$ time

$\leq S$ bits

$S$ space
MapReduce-style computations

What I won’t discuss today

- PRAMs (shared memory, multiple processors) (see e.g. [Karloff, Suri, Vassilvitskii‘10])
  - Computing XOR requires $\tilde{\Omega}(\log n)$ rounds in CRCW PRAM
  - Can be done in $O(\log_s n)$ rounds of MapReduce
- Pregel-style systems, Distributed Hash Tables (see e.g. Ashish Goel’s class notes and papers)
- Lower-level implementation details (see e.g. Rajaraman-Leskovec-Ullman book)
Models of parallel computation

- **Bulk-Synchronous Parallel Model (BSP)** [Valiant, 90]
  
  **Pro**: Most general, generalizes all other models
  
  **Con**: Many parameters, hard to design algorithms

- **Massive Parallel Computation** [Feldman-Muthukrishnan-Sidiropoulos-Stein-Svitkina’07, Karloff-Suri-Vassilvitskii’10, Goodrich-Sitchinava-Zhang’11, …, Beame, Koutris, Suciu’13]
  
  **Pros**:
  
  - Inspired by *modern* systems (Hadoop, MapReduce, Dryad, …)
  - Few parameters, simple to design algorithms
  - **New algorithmic ideas**, robust to the exact model specification
  - **# Rounds** is an information-theoretic measure => can prove unconditional lower bounds
  - Between *linear sketching* and *streaming with sorting*
Sorting: Terasort

• Sorting \( n \) keys on \( M = O(n^{1-\alpha}) \) machines
  – Would like to partition keys uniformly into blocks: first \( n/M \), second \( n/M \), etc.
  – Sort the keys locally on each machine

• Build an approximate histogram:
  – Each machine takes a sample of size \( s \)
  – All \( M \times s \leq S = n^{\alpha} \) samples are sorted locally
  – Blocks are computed based on the samples

• By Chernoff bound \( M \times s = O\left(\frac{\log n}{\epsilon^2}\right) \) samples suffice to compute all block sizes with \( \pm \epsilon n \) error

• Take \( \epsilon = \frac{n^{\alpha-1}}{n} \): error \( O(S) \);
  \( M \times s = \widetilde{O}(n^{2-2\alpha}) = O(M^2) \leq \widetilde{O}(n^\alpha) \) for \( \alpha \geq 2/3 \)
Algorithms for Graphs

• **Dense graphs** vs. sparse graphs
  
  – **Dense:** $S \gg |V|$
    
    • Linear sketching: one round
    
    • “Filtering” (Output fits on a single machine) [Karloff, Suri Vassilvitskii, SODA’10; Ene, Im, Moseley, KDD’11; Lattanzi, Moseley, Suri, Vassilvitskii, SPAA’11; Suri, Vassilvitskii, WWW’11]
  
  – **Sparse:** $S \ll |V|$ (or $S \ll \text{solution size}$)
    
    Sparse graph problems appear hard (**Big open question:** connectivity in $o(\log n)$ rounds?)

VS.
Algorithm for Connectivity

• Version of Boruvka’s algorithm
• Repeat $O(\log n)$ times:
  – Each component chooses a neighboring component
  – All pairs of chosen components get merged
• How to avoid chaining?
• If the graph of components is bipartite and only one side gets to choose then no chaining
• Randomly assign components to the sides
Algorithm for Connectivity: Setup

Data: \( N \) edges of an undirected graph.

Notation:
• For \( v \in V \) let \( \pi(v) \) be its id in the data
• \( \Gamma(S) \equiv \) set of neighbors of a subset of vertices \( S \subseteq V \).

Labels:
• Algorithms assigns a label \( \ell(v) \) to each \( v \).
• Let \( L_v \subseteq V \) be the set of vertices with the label \( \ell(v) \) (invariant: subset of the connected component containing \( v \)).

Active vertices:
• Some vertices will be called active.
• Every set \( L_v \) will have exactly one active vertex.
Algorithm for Connectivity

• Mark every vertex as active and let $\ell(v) = \pi(v)$.
• For phases $i = 1, 2, \ldots, O(\log N)$ do:
  – Call each active vertex a leader with probability $1/2$. If $v$ is a leader, mark all vertices in $L_v$ as leaders.
  – For every active non-leader vertex $w$, find the smallest leader (with respect to $\pi$) vertex $w^* \in \Gamma(L_w)$.
  – If $w^*$ is not empty, mark $w$ passive and relabel each vertex with label $w$ by $w^*$.
• Output the set of CCs, where vertices having the same label according to $\ell$ are in the same component.
Algorithm for Connectivity: Analysis

- If $\ell(u) = \ell(v)$ then $u$ and $v$ are in the same CC.
- Unique labels w.h.p after $O(\log N)$ phases.
- For every CC # active vertices reduces by a constant factor in every phase.
  - Half of the active vertices declared as non-leaders.
  - Fix an active non-leader vertex $v$.
  - If at least two different labels in the CC of $v$ then there is an edge $(v', u)$ such that $\ell(v) = \ell(v')$ and $\ell(v') \neq \ell(u)$.
  - $u$ marked as a leader with probability $1/2$; in expectation half of the active non-leader vertices will change their label.
- Overall, expect $1/4$ of labels to disappear.
- By Chernoff after $O(\log N)$ phases # of active labels in every connected component will drop to one w.h.p.
Algorithm for Connectivity: Implementation Details

• Distributed data structure of size $O(|V|)$ to maintain labels, ids, leader/non-leader status, etc.
  – $O(1)$ rounds per stage to update the data structure

• Edges stored locally with all auxiliary info
  – Between stages: use distributed data structure to update local info on edges

• For every active non-leader vertex $w$, find the smallest leader (w.r.t $\pi$) vertex $w^* \in \Gamma(L_w)$
  – Each (non-leader, leader) edges sends an update to the distributed data structure

• Much faster with Distributed Hash Table Service (DHT) [Kiveris, Lattanzi, Mirrokni, Rastogi, Vassilvitskii’14]
Applications

• Using same reductions as in streaming:
  – Bipartiteness
  – k-connectivity
  – Cut-sparsification
Approximating Geometric Problems in Parallel Models

Geometric graph (implicit):
Euclidean distances between $n$ points in $\mathbb{R}^d$

Already have solutions for old NP-hard problems (Traveling Salesman, Steiner Tree, etc.)

• Minimum Spanning Tree (clustering, vision)
• Minimum Cost Bichromatic Matching (vision)
Geometric Graph Problems

Combinatorial problems on graphs in $\mathbb{R}^d$

**Polynomial time ("easy")**
- Minimum Spanning Tree
- Earth-Mover Distance = Min Weight Bi-chromatic Matching

**NP-hard ("hard")**
- Steiner Tree
- Traveling Salesman
- Clustering (k-medians, facility location, etc.)

Need new theory!

Arora-Mitchell-style "Divide and Conquer", easy to implement in Massively Parallel Computational Models, but bad running time
MST: Single Linkage Clustering

- [Zahn’71] **Clustering** via MST (Single-linkage):
  - $k$ clusters: remove $k-1$ longest edges from MST
- Maximizes **minimum** intercluster distance

[Kleinberg, Tardos]
Earth-Mover Distance

- Computer vision: compare two pictures of moving objects (stars, MRI scans)
Large geometric graphs

• Graph algorithms: **Dense graphs** vs. sparse graphs
  – **Dense**: $S \gg |V|$
  – **Sparse**: $S \ll |V|.$

• **Our setting:**
  – Dense graphs, sparsely represented: $O(n)$ space
  – Output doesn’t fit on one machine ($S \ll n$)

• **Today:** $(1 + \epsilon)$-approximate MST
  – $d = 2$ (easy to generalize)
  – $R = \log_S n = O(1)$ rounds ($S = n^{\Omega(1)}$)
**$O(\log n)$-MST in $R = O(\log n)$ rounds**

- Assume points have integer coordinates $[0, \ldots, \Delta]$, where $\Delta = O(n^2)$.

**Impose an $O(\log n)$-depth quadtree**

**Bottom-up:** For each cell in the quadtree:
- Compute optimum MSTs in subcells
- Use only one representative from each cell on the next level

**Wrong representative:**
$O(1)$-approximation per level
**εL-nets**

- **εL-net** for a cell C with side length \( L \):
  Collection \( S \) of vertices in C, every vertex is at distance \( \leq \varepsilon L \) from some vertex in \( S \). (Fact: Can efficiently compute \( \varepsilon \)-net of size \( O \left( \frac{1}{\varepsilon^2} \right) \))

**Bottom-up:** For each cell in the quadtree
  - Compute optimum MSTs in subcells
  - Use \( \varepsilon L \)-net from each cell on the next level

- **Idea:** Pay only \( O(\varepsilon L) \) for an edge cut by cell with side \( L \)
- Randomly shift the quadtree: \( \Pr[\text{cut edge of length } \ell \text{ by } L] = \ell / L \) — charge errors
  O(1)-approximation per level

---

Wrong representative

<table>
<thead>
<tr>
<th>L</th>
<th>L</th>
</tr>
</thead>
</table>
Randomly shifted quadtree

- Top cell shifted by a random vector in $[0, L]^2$

**Impose a randomly shifted quadtree** (top cell length $2\Delta$)

Bottom-up: For each cell in the quadtree
- Compute optimum MSTs in subcells
- Use $\epsilon L$-net from each cell on the next level

Pay 5 instead of 4

$\Pr[\text{Bad Cut}] = \Omega(1)$
(1 + \(\epsilon\))-MST in \(\mathbb{R}\) = \(O(\log n)\) rounds

**Idea:** Only use short edges inside the cells

Impose a **randomly shifted** quadtree (top cell length \(\frac{2A}{\epsilon}\))

Bottom-up: For each node (cell) in the quadtree
- compute optimum Minimum Spanning Forests in subcells, using edges of length \(\leq \epsilon L\)
- Use only \(\epsilon^2 L\)-net from each cell on the next level

\[L = \Omega\left(\frac{1}{\epsilon}\right)\]

\[\Pr[\text{Bad Cut}] = O(\epsilon)\]
(1 + \(\epsilon\))-MST in \(\mathbb{R}\) = \(O(1)\) rounds

- \(O(\log n)\) rounds => \(O(\log_S n) = O(1)\) rounds
  - Flatten the tree: \((\sqrt{M} \times \sqrt{M})\)-grids instead of \((2x2)\) grids at each level.

\[
\begin{array}{c}
\text{\(\sqrt{M} = n^{\Omega(1)}\)}
\end{array}
\]

Impose a **randomly shifted** \((\sqrt{M} \times \sqrt{M})\)-tree

Bottom-up: For each node (cell) in the tree
- compute optimum MSTs in subcells via edges of length \(\leq \epsilon L\)
- Use only \(\epsilon^2 L\)-net from each cell on the next level
(1 + \(\epsilon\))-MST in \(\mathbf{R} = O(1)\) rounds

**Theorem:** Let \(l = \# \) levels in a random tree \(P\)

\[
\mathbb{E}_P[\text{ALG}] \leq (1 + O(\epsilon ld))\text{OPT}
\]

**Proof (sketch):**

- \(\Delta_P(u, v)\) = cell length, which first partitions \((u, v)\)
- **New weights:** \(w_P(u, v) = \|u - v\|_2 + \epsilon \Delta_P(u, v)\)

\[
\|u - v\|_2 \leq \mathbb{E}_P[w_P(u, v)] \leq (1 + O(\epsilon ld))\|u - v\|_2
\]

- Our algorithm implements Kruskal for weights \(w_P\)
“Solve-And-Sketch” Framework

(1 + \(\epsilon\))-MST:

– “Load balancing”: partition the tree into parts of the same size

– **Almost linear time locally**: Approximate Nearest Neighbor data structure [Indyk’99]

– Dependence on dimension \(d\) (size of \(\epsilon\)-net is \(O \left( \frac{d^d}{\epsilon^d} \right) \))

– Generalizes to bounded **doubling dimension**

– Implementation in MapReduce
“Solve-And-Sketch” Framework

(1 + \(\epsilon\))-Earth-Mover Distance, Transportation Cost

- No simple “divide-and-conquer” Arora-Mitchell-style algorithm (unlike for general matching)
- Only recently sequential (1 + \(\epsilon\))-approximation in \(O_\epsilon(n \log^{O(1)} n)\) time [Sharathkumar, Agarwal ‘12]

Our approach (convex sketching):

- Switch to the flow-based version
- In every cell, send the flow to the closest net-point until we can connect the net points
“Solve-And-Sketch” Framework

Convex sketching the cost function for $\tau$ net points

- $F: \mathbb{R}^{\tau-1} \rightarrow \mathbb{R}$ = the cost of routing fixed amounts of flow through the net points

- Function $F' = F +$ “normalization” is monotone, convex and Lipschitz, $(1 + \epsilon)$-approximates $F$

- We can $(1 + \epsilon)$-sketch it using a lower convex hull
Open problems:

• Exetension to high dimensions?
  – Probably no, reduce from connectivity => conditional lower bound: $\Omega(\log n)$ rounds for MST in $\mathbb{L}_\infty^n$
  – The difficult setting is $d = \Theta(\log n)$ (can do JL)

• Streaming alg for EMD and Transportation Cost?

• Our work:
  – First near-linear time algorithm for Transportation Cost
  – Is it possible to reconstruct the solution itself?
Class Project

• Survey of 3-5 research papers
  – Closely related to the topics of the class
    • Streaming
    • MapReduce
    • Convex Optimization
    • Sublinear Time Algorithms
  – Office hours if you need suggestions
  – Individual or groups of 2 people
  – **Deadline**: December 18, 2015 at 23:59 EST

• Submission by e-mail [grigory@grigory.us](mailto:grigory@grigory.us)
  – Submission Email Title: Project + Space + “Your Name”
  – One submission per group listing participants
  – Submission format
    • PDF from LaTeX (best)
    • PDF
Example: Gradient Descent in TensorFlow

- Gradient Descent (covered in class)
- Adagrad: [http://www.magicbroom.info/Papers/DuchiHaSi10.pdf](http://www.magicbroom.info/Papers/DuchiHaSi10.pdf)
K-means Clustering

• Given $X = \{x_1, \ldots, x_n\} \in \mathbb{R}^d$ find a set of centers $C = (c_1, \ldots, c_k)$ that minimizes

$$\sum_{x \in X} \min_{i \in [k]} \|x - c_i\|^2$$

• NP-hard problem

• Popular heuristic local search (Lloyd’s alg.)

• For a fixed partitioning $P_1, \ldots, P_k$:

$$c_j = \frac{1}{|P_j|} \cdot \sum_{i \in P_j} x_i$$
Dimension reduction for K-means

• Let \( \text{cost}_P(X) = \inf_c \text{cost}_{P,c}(X) \)

• For \( 0 < \epsilon < \frac{1}{2} \) let \( f : X \to \mathbb{R}^n \) be such that

\[
\forall i, j: (1 - \epsilon) \left\| x_i - x_j \right\|^2_2 \leq \left\| f(x_i) - f(x_j) \right\|^2_2 \leq (1 + \epsilon) \left\| x_i - x_j \right\|^2_2
\]

• \( \hat{P} \) is a \( \gamma \)-approx. clustering for \( f(X) \)

• \( P^* \) is an optimal clustering for \( X \)

• Lemma.

\[
\text{cost}_{\hat{P}} \leq \gamma \left( \frac{1 + \epsilon}{1 - \epsilon} \right) \text{cost}_{P^*}(X)
\]
Dimension reduction for K-means

- Let $\text{cost}_P(X) = \inf_c \text{cost}_{P,c}(X)$
- For $0 < \epsilon < \frac{1}{2}$ let $f: X \to \mathbb{R}^{d'}$ be such that
  \[ \forall i, j: (1 - \epsilon) \|x_i - x_j\|^2 \leq \|f(x_i) - f(x_j)\|^2 \leq (1 + \epsilon) \|x_i - x_j\|^2 \]
- $\hat{P}$ is a $\gamma$-approx. clustering for $f(X)$
- $P^*$ is an optimal clustering for $X$
- **Lemma.**
  \[ \text{cost}_{\hat{P}} \leq \gamma \left( \frac{1 + \epsilon}{1 - \epsilon} \right) \text{cost}_{P^*}(X) \]
- $d' = O \left( \log \frac{n}{\epsilon^2} \right)$ suffices by the JL-lemma
Dimension reduction for K-means

- Fix a partition $P = (P_1, ..., P_k)$

\[
\text{cost}_P(X) = \sum_{j \in [k]} \sum_{i \in P_j} \left( \sum_{i' \in P_j} \|x_i - \frac{1}{|P_j|} \sum_{i' \in P_j} x_{i'} \|^2_2 \right)
\]

\[
= \sum_{j \in [k]} \frac{1}{|P_j|} \sum_{i \in P_j} \left( \sum_{i' \in P_j} \|x_i\|^2_2 - 2 \langle x_i, \sum_{i' \in P_j} x_{i'} \rangle + \left\| \sum_{i' \in P_j} x_{i'} \right\|^2_2 \right)
\]

\[
= \sum_{j \in [k]} \frac{1}{|P_j|} \sum_{i \in P_j} \sum_{i' \in P_j} \left( \frac{\|x_i\|^2_2 + \|x_{i'}\|^2_2}{2} - \langle x_i, x_{i'} \rangle \right)
\]

\[
\sum_{j \in [k]} \frac{1}{2|P_j|} \sum_{i \in P_j} \sum_{i' \in P_j} \left( \|x_i - x_{i'}\|^2_2 \right)
\]

- $(1 - \epsilon)\text{cost}_P(X) \leq \text{cost}_P(f(X)) \leq (1 + \epsilon)\text{cost}_P(X)$
- $(1 - \epsilon)\text{cost}_P(X) \leq \text{cost}_P(f(X)) \leq \gamma \text{cost}_P(\hat{f}(X)) \leq \gamma \text{cost}_P(X)$
K-means++ Algorithm

- First center uniformly at random from $X$
- For a set of centers $C$ let:
  $$d^2(x, C) = \min_{c \in C} \|x - c\|_2^2$$
- Fix current set of centers $C$
- Subsequent centers: each $x_i$ with prob. 
  $$\frac{d^2(x_i, C)}{\sum_{x_j \in X} d^2(x_j, C)}$$
- Gives $O(\log k)$-approx. to OPT in expectation
K-means\parallel Algorithm

• First center $C$: sample a point uniformly
• Initial cost $\psi = \sum_x d^2(x, C)$
• For $O(\log \psi)$ times do:
  – Repeat $\ell$ times (in parallel)
    • $C' = \text{sample each } x_i \in X \text{ indep. with prob.}$
      \[
p_x = \frac{d^2(x_i, C)}{\sum_{x_j \in X} d^2(x_j, C)}
      \]
    • $C \leftarrow C \cup C'$
• For $x \in C$:
  $\nu_x = \text{the #points belonging to this center}$
• Cluster the weighted points in $C$ into $k$ clusters
K-means∥ Algorithm

- Oversampling factor $\ell = \Theta(k)$
- #points in $C$: $\ell \log \psi$
- **Thm.** If $\alpha$-approx. used in the last step then $k$-means∥ obtains an $O(\alpha)$-approx. to k-means
- If $\Psi$ and $\Psi'$ are the costs of clustering before and after one outer loop iteration then:

$$E[\Psi'] = O(OPT) + \frac{k}{e^{e\ell}} \Psi$$
K-means∥ Analysis

- For a set of points \( A = \{a_1, ..., a_t\} \) centroid \( c_A \):
  \[
c_A = \frac{1}{|T|} \sum a_t
\]
- Order \( a_1, ..., a_T \) in the increasing order by distance from \( c_A \)
- Fix a cluster \( A \) in OPT
- Fix \( C \) prior to the iteration and let:
  \[
  \phi(C) = \sum_x d^2(x, C)
  \]
  \[
  \phi_A(C) = \sum_a d^2(a, C)
  \]
- Let \( p_t = \frac{d^2(a_t, C)}{\phi(C)} \) be the probability of selecting \( a_t \)
- Probability that \( a_t \) is the smallest one chosen:
  \[
  q_t = p_t \prod_{j=1}^{t-1} (1 - p_j)
  \]
K-means∥ Analysis

• Can either assign all points to some selected $a_t$ or keep the original clustering:

$$s_t = \min \left( \phi_A, \sum_{a \in A} ||a - a_t||^2 \right)$$

• $E[\phi_A(C \cup C') \leq \sum_t q_t s_t + q_{T+1} \phi_A(C)$

  where $q_{T+1} = \text{prob. that no point in } A \text{ is selected}$

• Simplifying assumption: consider the case when all $p_t = p$ (mean field analysis)

• $q_t = p(1 - p)^t$ (decreasing sequence)
K-means|| Analysis

• \( s_t' = \sum_{a \in A} ||a - a_t||^2 \)
• \( \{s_t'\} \) is an increasing sequence

\[
\sum_t q_t s_t \leq \sum_t q_t s_t' \\
\leq \frac{1}{T} \left( \sum_t q_t \sum_t s'_t \right) \\
= \left( \sum_t q_t \cdot \frac{1}{T} \sum_t s'_t \right) \\
= \left( \sum_t q_t \cdot 2 \phi_A^* \right)
\]

• \( E[\phi_A(C \cup C')] \leq (1 - q_{T+1}) 2 \phi_A^* + q_{T+1} \phi_A(C) \)