Graph Clustering: Affinity Clustering and Higher-Order Clustering

Laxman Dhulipala
MIT (Postdoc)
https://ldhulipala.github.io/

Based on papers by Bateni et al. (Neurips 2017) and Yin et al. (KDD 2017)
Outline

❖ Clustering and Graph Clustering Overview
❖ Affinity Clustering
❖ Higher-Order Clustering
❖ Future Directions
❖ Conclusion
Clustering

Problem (informal):
Group objects in such a way that objects in the same group (cluster) are more similar than those in other groups (clusters).

Points in ambient space

Vertices and edges in a (potentially weighted) graph
Flat and Hierarchical Clustering

Flat Clustering:
Assign objects to clusters (no structure relating clusters to other clusters)

Hierarchical Clustering:
Build a hierarchy of clusters called a *dendrogram*
Often want clusters to be formed by *binary merges* of sub-clusters
Dendrograms usually equipped with a weight (*similarity*) indicating how similar the two merged clusters are
Hierarchical Graph Clustering

Problem:
Given a graph with positive edge weights representing distances (smaller is more similar), compute a hierarchical clustering of the graph.

Input

Clustering

Validate

labels

threshold

\[ A = \text{pig} \]

\[ B = \text{bear} \]

\[ C = ? \]

\[ D = \text{dog} \]
Hierarchical Agglomerative Clustering (on graphs)

- defined using different **linkage function**
- can either work in **similarity** or **dissimilarity** setting. Let’s stick with (D) for now.

<table>
<thead>
<tr>
<th>(s)</th>
<th>(D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>larger weights are more similar</td>
<td>smaller weights are more similar</td>
</tr>
</tbody>
</table>

**Generic HAC algorithm: II dissimilarity**

while $J$ more than one cluster:

- let $(u,v)$ be the most similar (smallest-weight) edge
- merge $(u,v)$ into a new cluster
- update weights in the graph using the specified **linkage function**
Suppose the HAC algorithm merges two vertices $A, B$ to form a cluster $A \cup B$. How do we weight edges out of this new cluster?
Linkage Functions:

Single Linkage: \( W(A \cup B, C) = \min \{ w(A, C), w(B, C) \} \)

Complete Linkage: \( W(A \cup B, C) = \max \{ w(A, C), w(B, C) \} \)

Weighted Average-Linkage: \( W(A \cup B, C) = \frac{(w(A, C) + w(B, C))}{2} \)
Linkage Functions Cont'd.

Unweighted Average-Linkage:

\[ W(A \cup B, C) = \frac{\sum_{(a,b) \in E} w(a, b)}{|A| \cdot |B|} \]

\[ w(a, b) = \begin{cases} 2 & \text{if } a \neq b, \text{ and } a, b \in A \cup B \cup C \\ 0 & \text{otherwise} \end{cases} \]

Weighted Average-Linkage:

\[ \frac{1}{|A| \cdot W(A, C) + 1B| \cdot W(B, C)} \]

\[ \frac{|A| \cdot 4 + 1B| \cdot 5}{|A| \cdot 1B|} = \frac{4 + 5}{4} \]

\[ \text{Weighted avg } = \frac{13}{2} \]
HAC: Output (unweighted avg-link)

merge weight: 1

merge weight: 2

merge weight: 4
Parallelizing Hierarchical Agglomerative Graph Clustering

Is it possible to solve this problem in \( \text{NC} \)?

What about using different linkage functions?

- Single-Linkage
  - \( \text{MST}(n) \) postprocesses \( \text{ENC} \) dendrogram
  - \( O(n \log n) \) work, \( O(\text{polylog} n) \) depth

\[ \text{Class of problems solvable in today: want close to best seq.} \]

- \( \text{poly}(n) \) work?
- \( \text{polylog}(n) \) depth alg

\[ \text{RNC} \]

A generic \( \text{HAC} \) alg:

- Complete Linkage \( \rightarrow \) \( \text{CC-hard} \)
- \( \approx \) \( \text{Weighted/Unweighted} \) \( \text{P-complete} \)

straightforward parallelization

\( O(m+n) \) work

\( O(n) \) depth
Background: Boruvka’s Algorithm

Cut property:
Let $S$ be any subset of vertices. The minimum cost edge on the boundary of $S$ is in the MST.
def Boruvka(G(V, E, w)):
    # Compute the minimum edge out of each vertex.
    Let the set of min-weight edges be MinE.
    # Compute connected components on the graph induced
    # by only edges in MinE.
    C = Components(G[V, MinE])
    # Contract the graph to the components of C. An edge
    # (u,v) in E is discarded if C(u) = C(v). For
    # duplicate edges (u,v) with C(u) != C(v), keep the
    # minimum-weight edge.
    GC = ContractMin(G, C)
    return MinE U Boruvka(GC)

How many components can there be in C?

#vertices (deterministically) decreases by a constant factor per-round

Overall parallel cost is:

$O(m \log n)$ work

$O(\log^2 n)$ depth
Affinity Clustering

Idea: Stop Borůvka's Alg. after \( r > 0 \) rounds, at the first time when there are \( \leq k \) clusters for some desired \( k > 0 \).

If \( < k \) clusters, delete the edges added in the last round in decreasing order to get exactly \( k \) clusters.

if \( k = 2 \), cut the weight 4 edge to get 2 clusters.
Hierarchical Affinity Clustering

**Round 1**

**Round 2**

The fanout/arity of a cluster can be arbitrarily large:

Figure 1: An example of how affinity may produce a large component in one round.
Contributions of This Paper:

- Theoretical characterization of Affinity clustering under randomly distributed points.
  Note that worst case guarantees on cluster sizes not possible.

- Characterization of the "cost" of affinity clustering wrt any non-singleton clusterings (min cluster size $\geq 2$)

- Characterization of single-linkage clustering.
  Each vertex in single-linkage clustering w. k clusters (non-singleton) has a neighbor inside its cluster which is closer than any vertex outside the cluster.
Algorithms Contributions:

- $O(1)$ round MPC algorithm for MST for dense graph
  - $m = \Theta(n^{1+c})$ for any constant $c > 0$
  - space-per-machine = $S = \tilde{O}(n^{1+c})$ w.h.p. for $0 < c < C$
  - total machines = $T = O(n^{c-\eta})$
  - polylog factors
  - runs in $\lceil \log (C/\varepsilon) \rceil + 1$ rounds of MPC

- $O(\log n)$ round MPC algorithm using Distributed Hash Tables (DHT)
  - $O(\log^2 n)$ rounds without DHT
Massively Parallel Computation (MPC) Model

- \( N \) input size
- total of \( M \) machines each with space \( S \)
- Both \( M \) and \( S \) are sublinear in \( N \), e.g., \( M = O(N^{1-\epsilon}) \) \( S = O(N^\epsilon) \)

Within one round machines can perform arb. polytime computation on local data.

\[ \text{Round 1} \]

\( \sum_s \) # messages sent (received = \( O(s) \))

\[ \text{Round 2} \]
MST Algorithm (Dense Graphs)\[ S = N = m \]

Recall that \[ S = O(n^{1+\varepsilon}) \], \[ m = O(n^{1+c}) \], and \( 0 < \varepsilon < c \)

\[ \rightarrow \quad \text{if } S = O(n^{1+c}) \text{ can solve MST in one round} \]

- Can't fit all edges in one machine; have to compute edges some other way

Q: What about running Borůvka?

A: \( O(\log^2 n) \) round complexity (follows from work-depth discussion)

Hint: Connectivity can be solved in \( O(\log n) \) MPC rounds through PRAM simulations.
**MST Algorithm (Dense Graphs)**

**Observation:** If $G' = (V', E')$ is an arbitrary subgraph of $G$ and an edge $e' e E' \notin \text{MST}(G')$ then $e' \notin \text{MST}(G)$

**Idea:** Divide $G$ into subgraphs s.t. each edge of $G$ is in at least 1 subgraph (and subgraph sizes $\leq S$). Then since $|\text{MST}| = O(n)$ and $S = O(n^{1+\epsilon})$, we will get rid of a lot of edges.

\[\Downarrow\]

Repeat until only $S = O(n^{1+\epsilon})$ edges left and solve on a single machine.
**Algorithm 1** MST of Dense Graphs

**Input:** A weighted graph \( G \)

**Output:** The minimum spanning tree of \( G \)

1. function \( \text{MST}(G = (V, E), c) \)
2. \( c \leftarrow \log_n (m/n) \)
3. \( \text{while } |E| > O(n^{1+c}) \text{ do} \)
4. \( \text{REDUCEEDGES}(G, c) \)
5. \( c \leftarrow (c - \varepsilon)/2 \)
6. Move all the edges to one machine and find MST of \( G \) in there.
7. function \( \text{REDUCEEDGES}(G = (V, E), c) \)
8. \( k \leftarrow n^{(c-\varepsilon)/2} \)
9. Independently and u.a.r. partition \( V \) into \( k \) subsets \( \{V_1, \ldots, V_k\} \).
10. Independently and u.a.r. partition \( V \) into \( k \) subsets \( \{U_1, \ldots, U_k\} \).
11. Let \( G_{i,j} \) be a subgraph of \( G \) with vertex set \( V_i \cup U_j \) containing any edge \((v, u) \in E(G)\) where \( v \in V_i \) and \( u \in U_j \).
12. for any \( i, j \in \{1, \ldots, k\} \) do
13. Send all the edges of \( G_{i,j} \) to the same machine and find its MST in there.
14. Remove an edge \( e \) from \( E(G) \), if \( e \in G_{i,j} \) and it is not in MST of \( G_{i,j} \).
Lemma: Alg. 1 correctly finds the MST in \(\lceil \log (\varepsilon/n) \rceil + 1\) rounds.

**Correctness:** each call to Reduce Edge randomly partitions vertices into

\[ V = \{ V_1, \ldots, V_k \} \]

\[ U = \{ U_1, \ldots, U_k \} \]

and for each \((i, j)\) pair \(i \in \{ 1, \ldots, k \}\)
finds \(MST(C_i, j)\), discarding any edge in \(G_{i, j} \cap MST(C_i, j)\).

\[ \rightarrow \text{none of the discarded edge are part of } MST(C) \]

---

**Algorithm 1 MST of Dense Graphs**

**Input:** A weighted graph \(G\)

**Output:** The minimum spanning tree of \(G\)

1. function \(MST(G = (V, E), \varepsilon)\)
2. \[ c \leftarrow \log_{\varepsilon} (m/n) \] \(\triangleright\) Since \(G\) is assumed to be dense we know \(c > 0\).
3. while \(|E| > O(n^{1+\varepsilon})\) do
4. \(\text{REDUCE EDGES}(G, c)\)
5. \[ c \leftarrow (c - \varepsilon)/2 \]
6. Move all the edges to one machine and find MST of \(G\) in there.
7. function \(\text{REDUCE EDGES}(G = (V, E), \varepsilon)\)
8. \[ k \leftarrow \frac{n(c - \varepsilon)/2}{\varepsilon} \]
9. Independently and u.a.r. partition \(V\) into \(k\) subsets \(\{V_1, \ldots, V_k\}\).
10. Independently and u.a.r. partition \(V\) into \(k\) subsets \(\{U_1, \ldots, U_k\}\).
11. Let \(G_{i,j}\) be a subgraph of \(G\) with vertex set \(V_i \cup U_j\) containing any edge \((v, u) \in E(G)\) where \(v \in V_i\) and \(u \in U_j\).
12. for any \(i, j \in \{1, \ldots, k\}\) do
13. \(\text{Send all the edges of } G_{i,j} \text{ to the same machine and find its MST in there.}\)
14. \(\text{Remove an edge } e \text{ from } E(G), \text{ if } e \in G_{i,j} \text{ and it is not in MST of } G_{i,j}\).
Round complexity:

- Let \( c_r = \text{value of } c \text{ in } r\text{-th iter.} \)
- Let \( k_r = n \frac{c_r - \epsilon}{2} \)
- For each \( G_{i,j} \) let \( T_{i,j} = \text{MST}(k_{i,j}) \). Notice that only \( \bigcup_{i,j} T_{i,j} \) are kept in next round.

\[ \text{MST on } n' \text{ vertices has } \leq n' - 1 \text{ edges.} \]

Next, conceptually charge each edge \( e \in T_{i,j} \) to a vertex in \( T_{i,j} \)

Claim: each vertex in \( G_{i,j} \) charged at most once

\[ \text{e.g. } \]

\[ \text{Idea?} \]

---

Algorithm 1 MST of Dense Graphs

**Input:** A weighted graph \( G \)

**Output:** The minimum spanning tree of \( G \)

1: function \( \text{MST}(G = (V, E), c) \)  
2: \( c \leftarrow \log_2(n/m) \)  
3: \( \text{while } |E| > O(n^{1+\epsilon}) \text{ do} \)
4: \( \text{REDUCE EDGES}(G, c) \)
5: \( c \leftarrow (c - \epsilon)/2 \)
6: \( \text{Move all the edges to one machine and find MST of } G \text{ in there.} \)
7: function \( \text{REDUCE EDGES}(G = (V, E), c) \)
8: \( k \leftarrow n^{(1-\epsilon)/2} \)
9: Independently and u.a.r. partition \( V \) into \( k \) subsets \( \{V_1, \ldots, V_k\} \).
10: Independently and u.a.r. partition \( V \) into \( k \) subsets \( \{U_1, \ldots, U_k\} \).
11: Let \( G_{i,j} \) be a subgraph of \( G \) with vertex set \( V_i \cup U_j \) containing any edge \((v, u) \in E(G)\)
\[ \text{where } v \in V_i \text{ and } u \in U_j. \]
12: for any \( i, j \in \{1, \ldots, k\} \) do
13: Send all the edges of \( G_{i,j} \) to the same machine and find its MST in there.
14: Remove an edge \( e \) from \( E(G) \), if \( e \in G_{i,j} \) and it is not in MST of \( G_{i,j} \).
Round Complexity:

- Let $c_r = \text{value of } c \text{ in } r\text{-th iter.}$
- Let $k_r = \frac{(c_r - \varepsilon)}{2}$
- Let each $G_{i,j}$ let $T_{i,j} = \text{MST}(G_{i,j})$.
- Each vertex in $G_{i,j}$ charged at most once.

Consider $v \in V_i$ (w.l.o.g.). $v$ appears in $G_{i,1}, \ldots, G_{i, k_r}$. Therefore $v$ can be charged for at most $k_r$ edges.

$k_r \text{ times } \Rightarrow k_r \cdot n$ is an upper bound for $\# \text{ edges at end of } r\text{-th round.}$

$\Rightarrow k_r \cdot n = n + (c_r - \varepsilon)/2$ and $c_r < \frac{c}{2\varepsilon} \cdot \frac{\log(\log(\varepsilon/c))}{2} \leq \varepsilon$

$\Rightarrow O(n^{1+\varepsilon}) \text{ edges after } \lceil \log(\varepsilon/c) \rceil \text{ rounds.}$

Algorithm 1 MST of Dense Graphs

**Input:** A weighted graph $G$

**Output:** The minimum spanning tree of $G$

1. \textbf{function} MST$(G = (V, E), c)$
2. \hspace{1em} $c \leftarrow \log_{\text{base } c}(m/n)$ \hspace{1em} $\triangleright$ Since $G$ is assumed to be dense we know $c > 0$.
3. \hspace{1em} \textbf{while} $|E| > O(n^{1+\varepsilon})$ \textbf{do}
4. \hspace{2em} \textbf{REDUCE EDGES}(G, c)
5. \hspace{1em} \hspace{1em} $c \leftarrow (c - \varepsilon)/2$
6. \hspace{1em} Move all the edges to one machine and find MST of $G$ in there.
7. \textbf{function} REDUCE EDGES$(G = (V, E), c)$
8. \hspace{1em} $k \leftarrow n(1-\varepsilon)/2$
9. \hspace{1em} Independently and u.a.r. partition $V$ into $k$ subsets $\{V_1, \ldots, V_k\}$.
10. Independently and u.a.r. partition $V$ into $k$ subsets $\{U_1, \ldots, U_k\}$.
11. \hspace{1em} Let $G_{i,j}$ be a subgraph of $G$ with vertex set $V_i \cup U_j$ containing any edge $(v, u) \in E(G)$ where $v \in V_i$ and $u \in U_j$.
12. \hspace{1em} \textbf{for any } i, j \in \{1, \ldots, k\} \textbf{ do}
13. \hspace{2em} Send all the edges of $G_{i,j}$ to the same machine and find its MST in there.
14. \hspace{1em} Remove an edge $e$ from $E(G)$, if $e \in G_{i,j}$ and it is not in MST of $G_{i,j}$.

$A_{\varepsilon} = c_r$, $c_r = (c_{r-1} - \varepsilon)/2$
MST Algorithm (Sparse Graphs)

Let \( G(V, E, w) \) be given. \( n = |V|, m = |E| \) (no requirements on \( m \))

Algorithm: (proceed in rounds)
- each vertex finds its best edge (most similar edge)
- graph is contracted along the selected edges

Round 1

Round 2

\[ \text{PRAM} \quad \begin{array}{c} \text{multi-prefix} \\ \text{PRAM} \end{array} \quad \text{MPM} \quad 0(1) \text{ rounds} \]
MST Algorithm (Sparse Graphs)

Let \( G(V, E, w) \) be given. \( n = |V|, m = |E| \) (no requirements on \( m \))

Algorithm: (proceed in rounds)

1. Each vertex finds its best edge (most similar edge) \( \Omega(1) \) rounds
2. Graph is contracted along the selected edges \( \Omega(n^2) \) rounds

(2) solvable using connectivity as we discussed before but requires \( \Omega(n \log n) \) rounds.

Turn out that we can solve (2) in \( O(1) \) rounds using a distributed hash table (DHT)
MST Algorithm (Sparse Graphs)

Let \( G(V, E, w) \) be given. \( n = |V| \), \( m = |E| \) (no requirements on \( m \))

Algorithm: (proceed in rounds)
1. each vertex finds its best edge (most similar edge)
2. graph is contracted along the selected edges using DHT

- Each loop of two gives a unique label for a connected component.
- Performing all queries takes \( O(1) \) rounds

Adaptive MPC (AMPC) model
Experiments

First, how do we compare two different clusterings?

**Definition 4** (Rand index [40]). Given a set \( V = \{v_1, \ldots, v_n\} \) of \( n \) points and two clusterings \( X = \{X_1, \ldots, X_r\} \) and \( Y = \{Y_1, \ldots, Y_s\} \) of \( V \). Define the following.

- \( a \): the number of pairs in \( V \) that are in the same cluster in \( X \) and in the same cluster in \( Y \).
- \( b \): the number of pairs in \( V \) that are in different clusters in \( X \) and in different clusters in \( Y \).

The Rand index \( r(X, Y) \) is defined to be \( (a + b)/\binom{n}{2} \). By having the ground truth clustering \( T \) of a data set, we define the Rand index score of a clustering \( X \), to be \( r(X, T) \).

**Example.**

\[ n = 4 \quad V = \{v_1, v_2, v_3, v_4\} \]

\[ X = \{\{v_1, v_4\}, \{v_2, v_3\}\} \quad a = \]

\[ T = \{\{v_1, v_4, v_2\}, \{v_3\}\} \quad b = \]

\[ r(X, T) = \]
Evaluation

- Generally single affinity performs very well! Surprisingly better than HAC algorithms.
- K-means is also close.
- For hierarchical clustering, level of tree w. highest score is used.

Idea is to reweight edge after each round. 
Graph-based HAC on complete graph.

run in original metric
different linkage versions of affinity
Scalability

Table 1: Statistics about datasets used. (Numbers for ImageGraph are approximate.) The fifth column shows the relative running time of affinity clustering, and the last column is the speedup obtained by a ten-fold increase in parallelism.

<table>
<thead>
<tr>
<th>Dataset</th>
<th># nodes</th>
<th># edges</th>
<th>max degree</th>
<th>running time</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>LiveJournal</td>
<td>4,846,609</td>
<td>7,861,383,690</td>
<td>444,522</td>
<td>1.0</td>
<td>4.3</td>
</tr>
<tr>
<td>Orkut</td>
<td>3,072,441</td>
<td>42,687,055,644</td>
<td>893,056</td>
<td>2.4</td>
<td>9.2</td>
</tr>
<tr>
<td>Friendster</td>
<td>65,608,366</td>
<td>1,092,793,541,014</td>
<td>2,151,462</td>
<td>54</td>
<td>5.9</td>
</tr>
<tr>
<td>ImageGraph</td>
<td>$2 \times 10^{10}$</td>
<td>$10^{12}$</td>
<td>14000</td>
<td>142</td>
<td>4.1</td>
</tr>
</tbody>
</table>

- Construct weighted graphs by setting $w(u,v) = |N(u) \cap N(v)|$ (number of common neighbors) and discarding 0-weight edges.
- Procedure basically connects a vertex's 2-hop neighborhood.
- Use maximum spanning tree (similarity) version of affinity.

increase $W$ by $10^x$. See $4-10x$ speedup.
Hierarchical Clustering using MST

Compute MST

Postprocess MST to compute a Dendrogram

recent result due to Yiqiu Wang, Shangdi Yu, Yan Gu, and Julian Shun (2021)

Turn out that one can get the single-linkage dendrogram in NC