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)

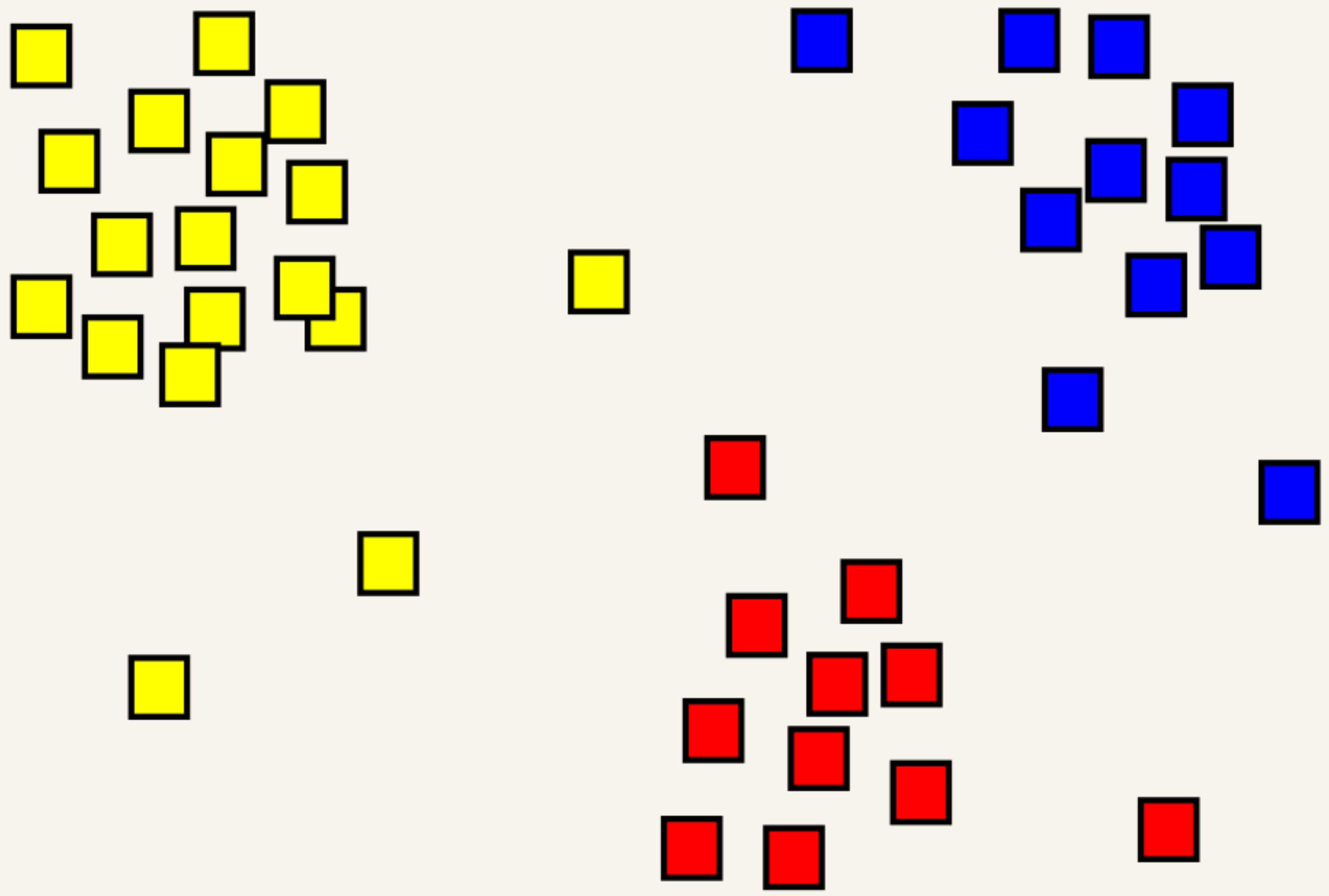
Outline

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

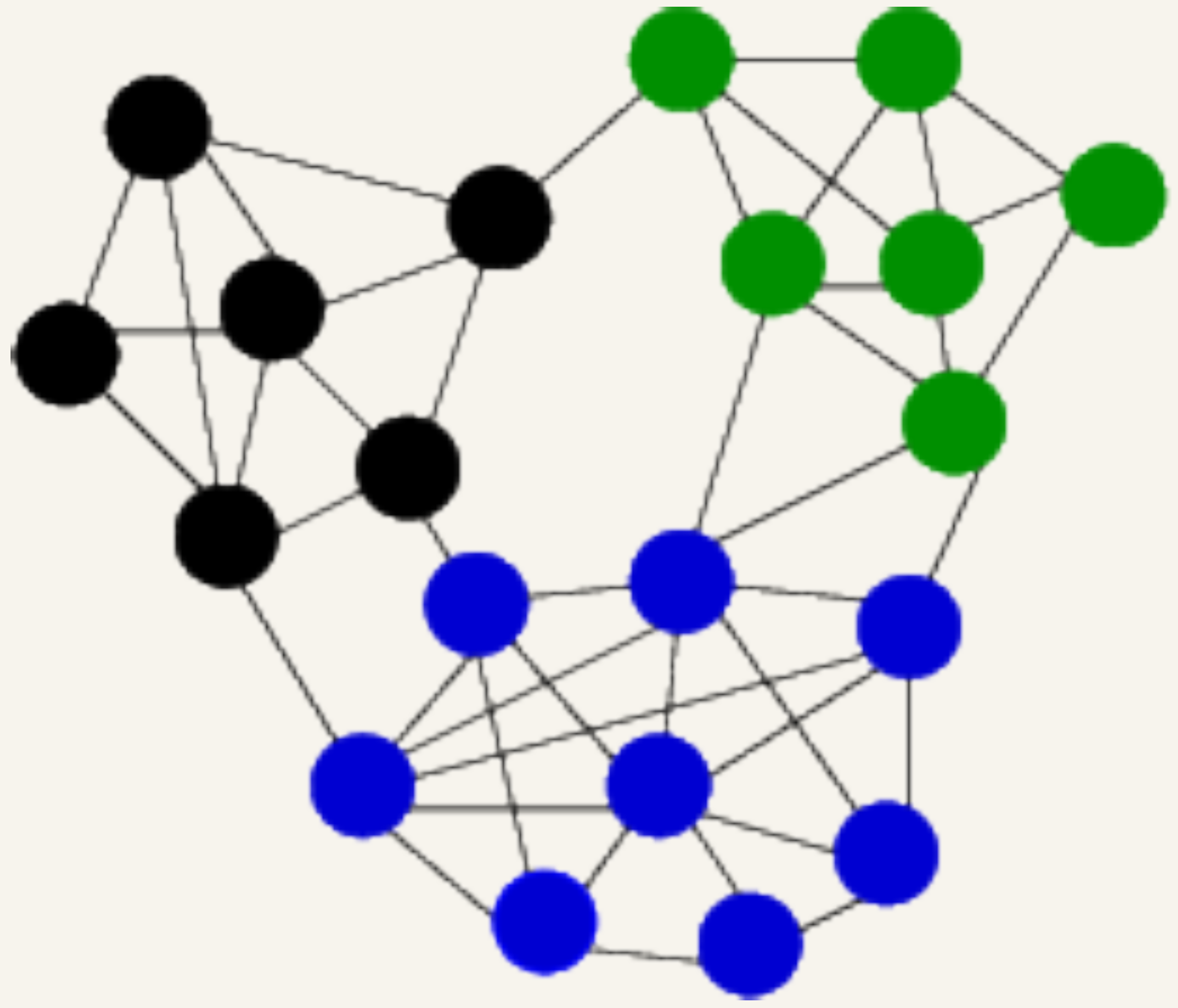
Clustering

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

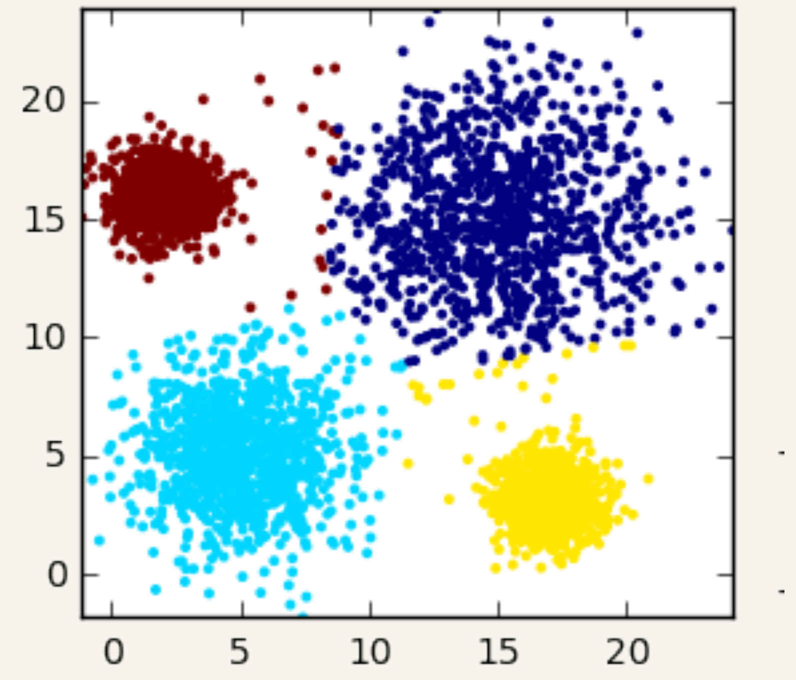


Vertices and edges in a (potentially weighted) graph

Flat and Hierarchical Clustering

Flat Clustering:

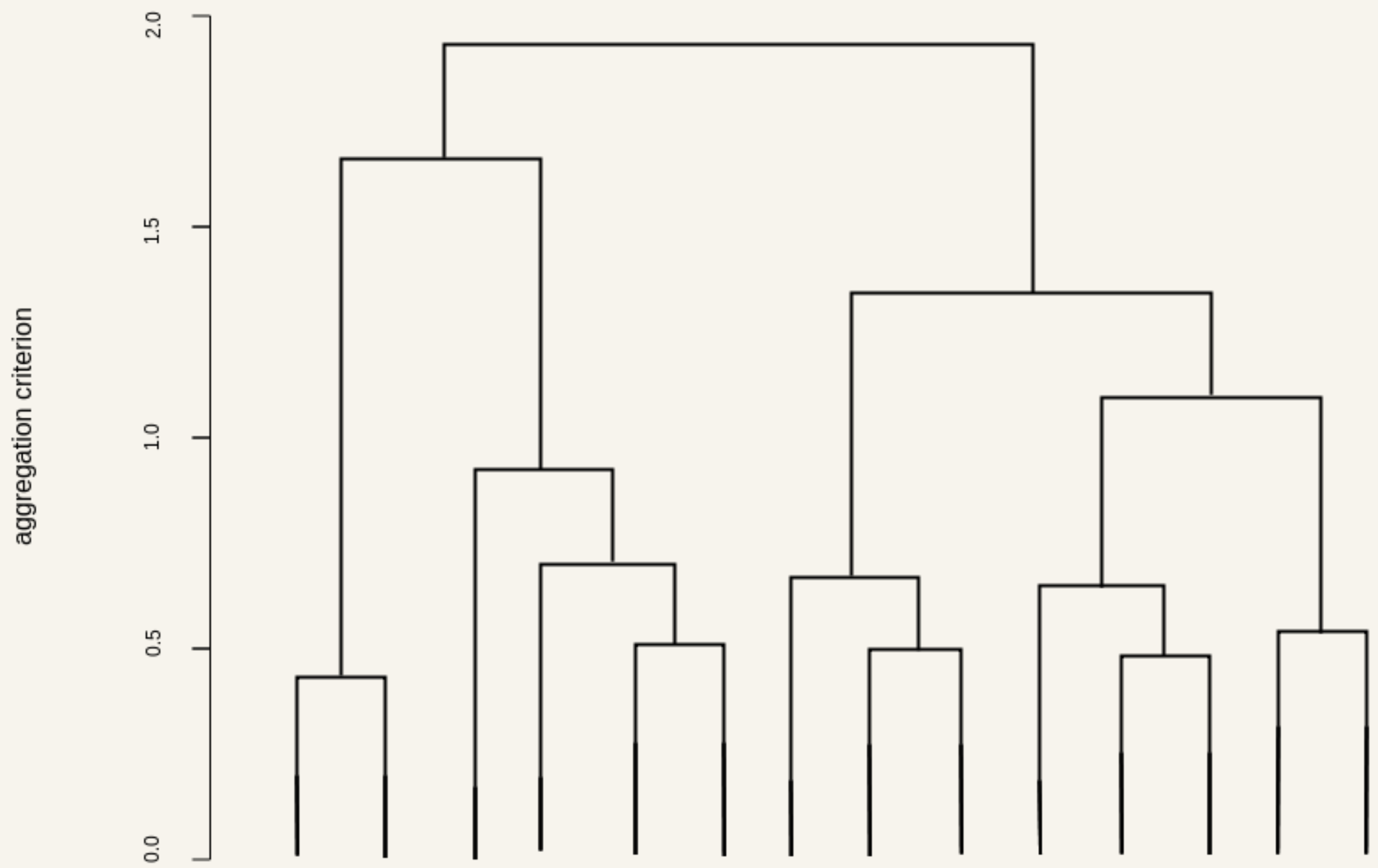
Assign objects to clusters (no structure relating clusters to other clusters)



Hierarchical Clustering:

Build a hierarchy of clusters called a *dendrogram*

Often want clusters to be formed by *binary merges* of sub-clusters

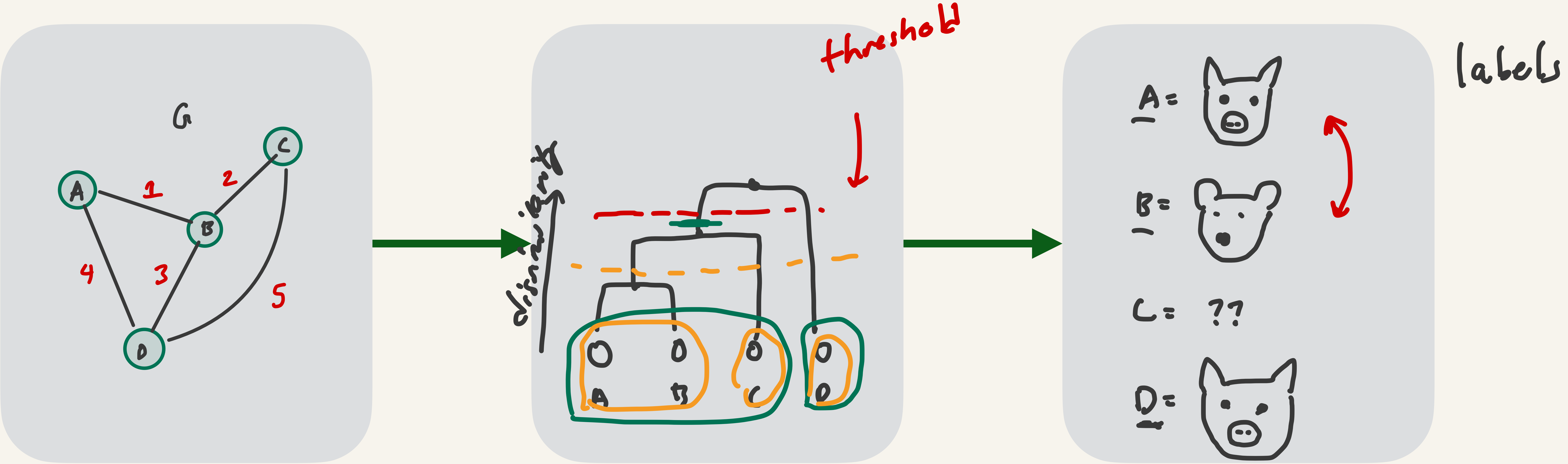


Dendrograms usually equipped with a weight (*similarity*) indicating how similar the two merged clusters are

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

Hierarchical Agglomerative Clustering (on Graphs)

- defined using different <u>linkage</u> function
- can either work in <u>similarity</u> (S) or <u>dissimilarity</u> (D) setting. Let's stick with (D) for now.

  similarity → larger weights are more similar

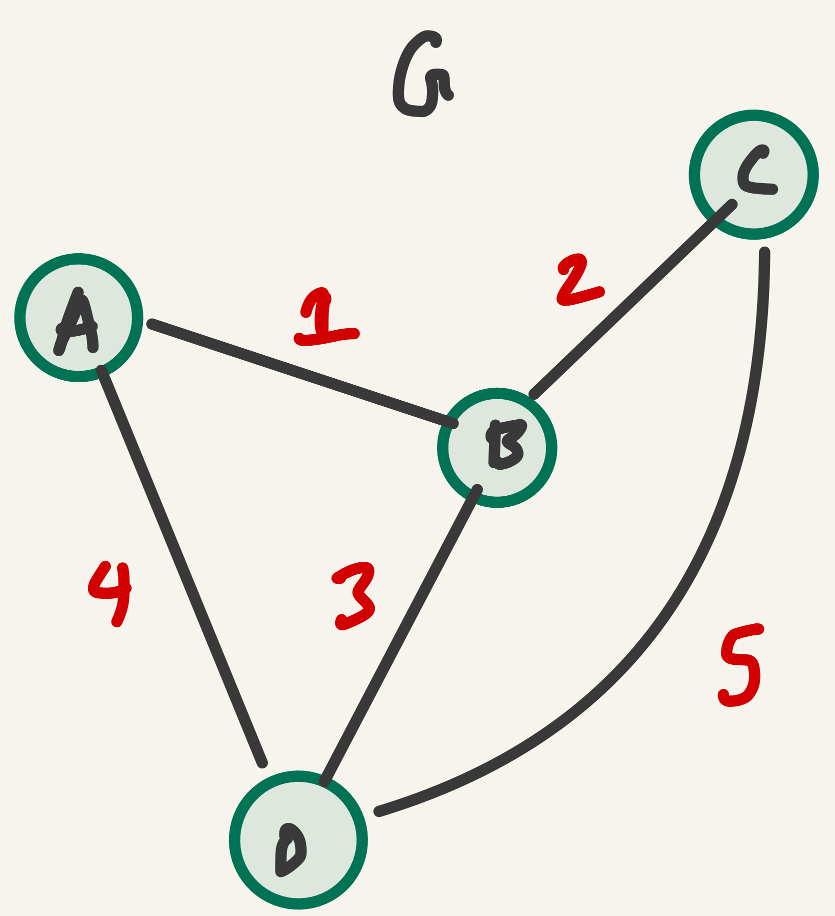
  dissimilarity → smaller weights are more similar

Generic HAC algorithm: // dissimilarity

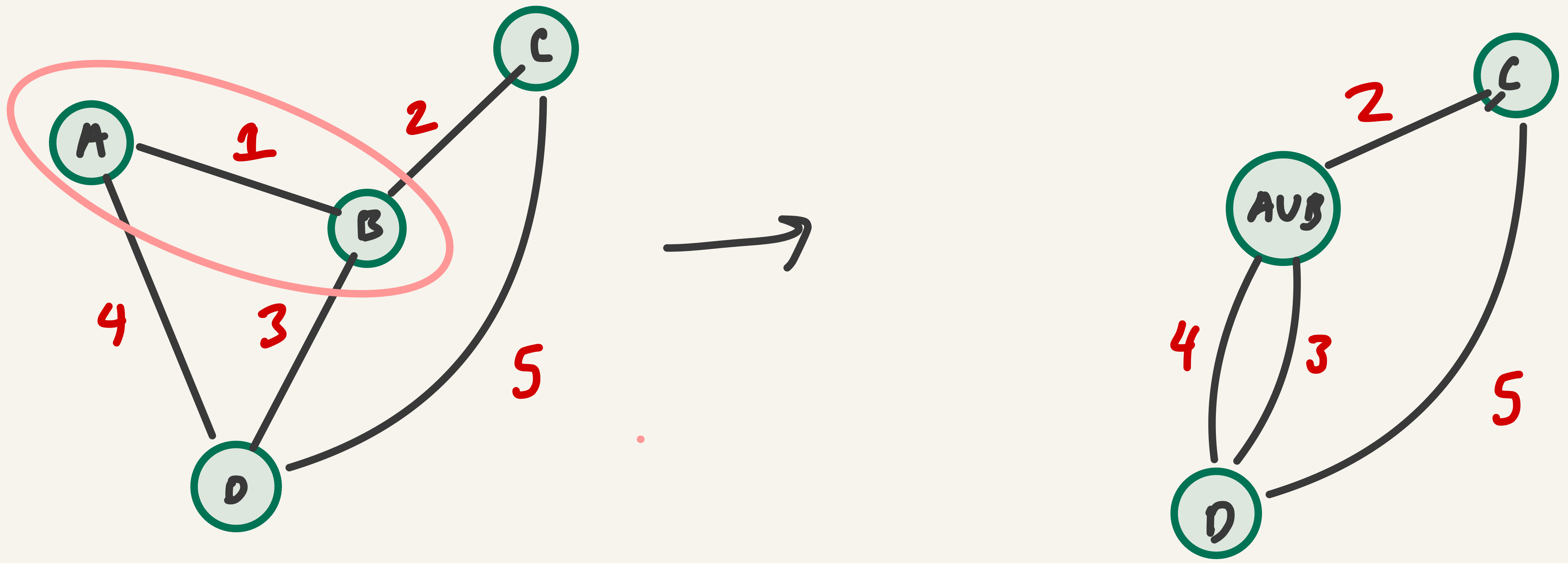
While ∃ more than one cluster:

- 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 <u>linkage</u> function

Graph-HAC Example:

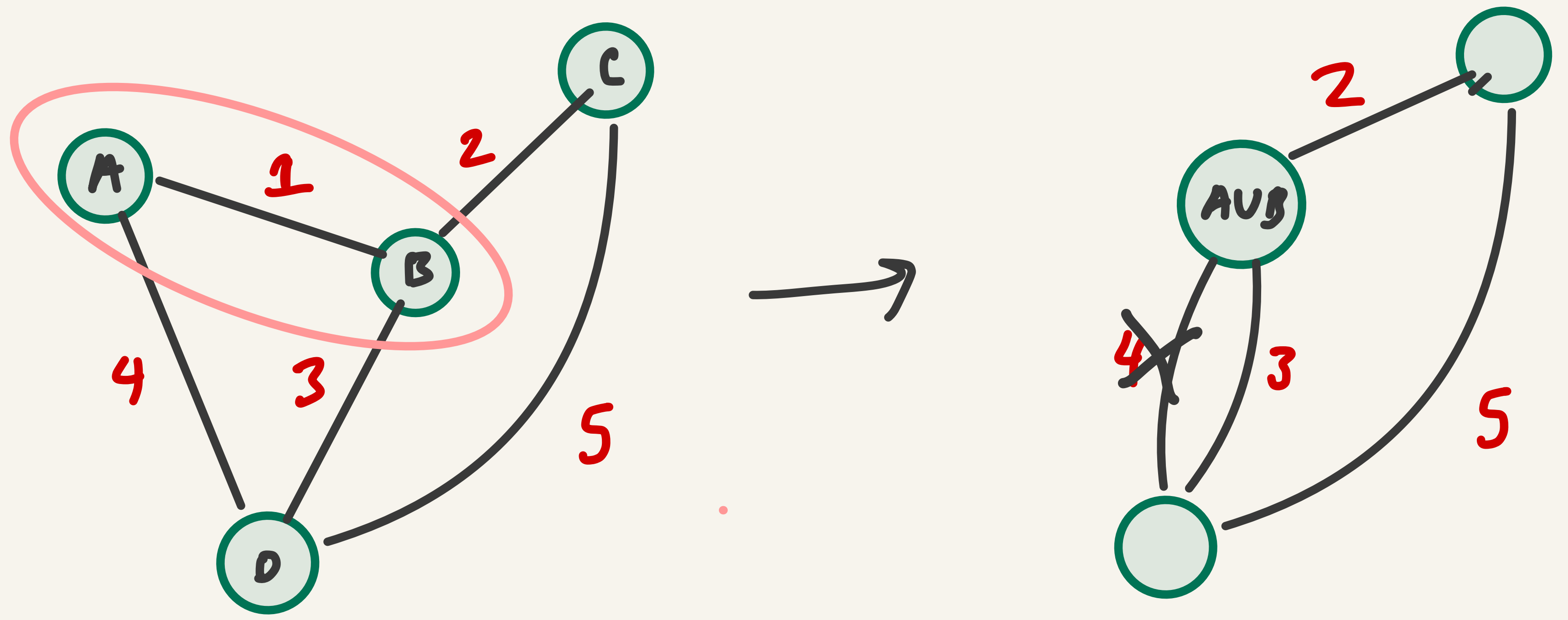


Suppose the HAC algorithm merges two vertices A, B to form a cluseter $A \cup B$. How do we weight edges out of this new cluster?

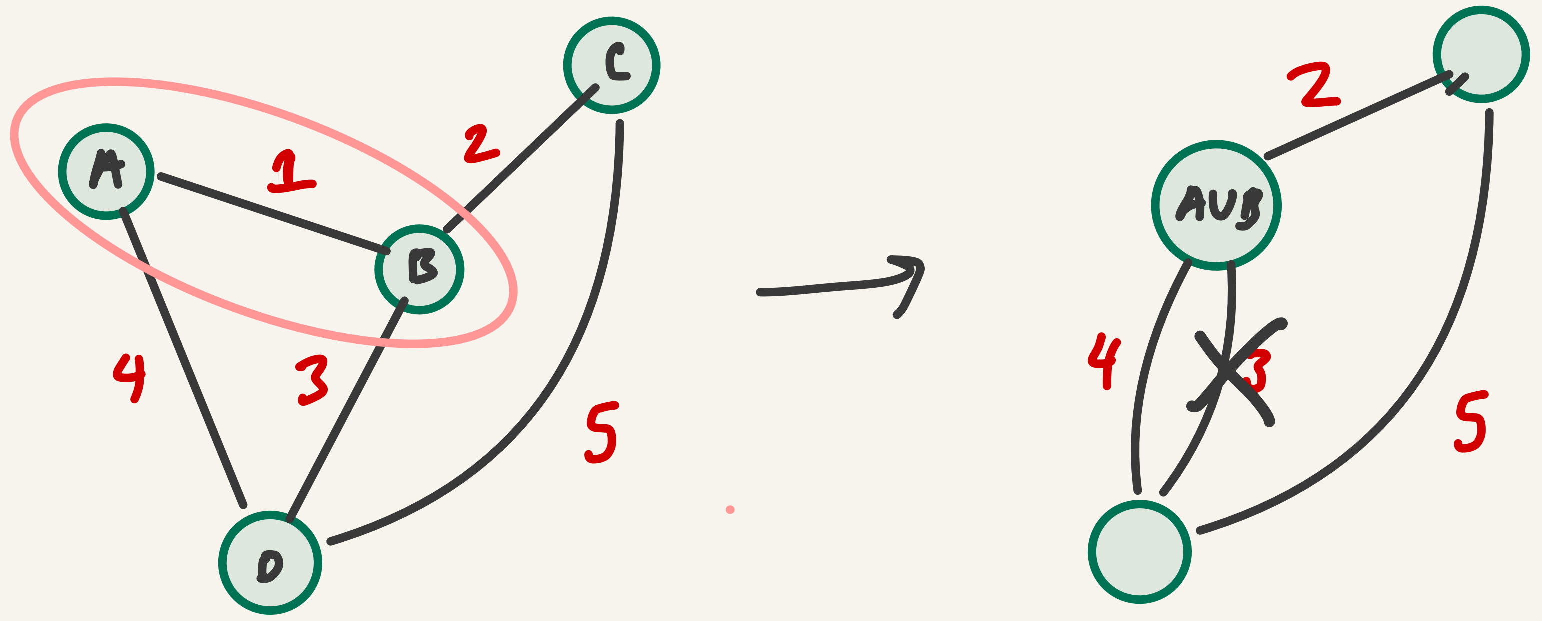


Linkage Functions:

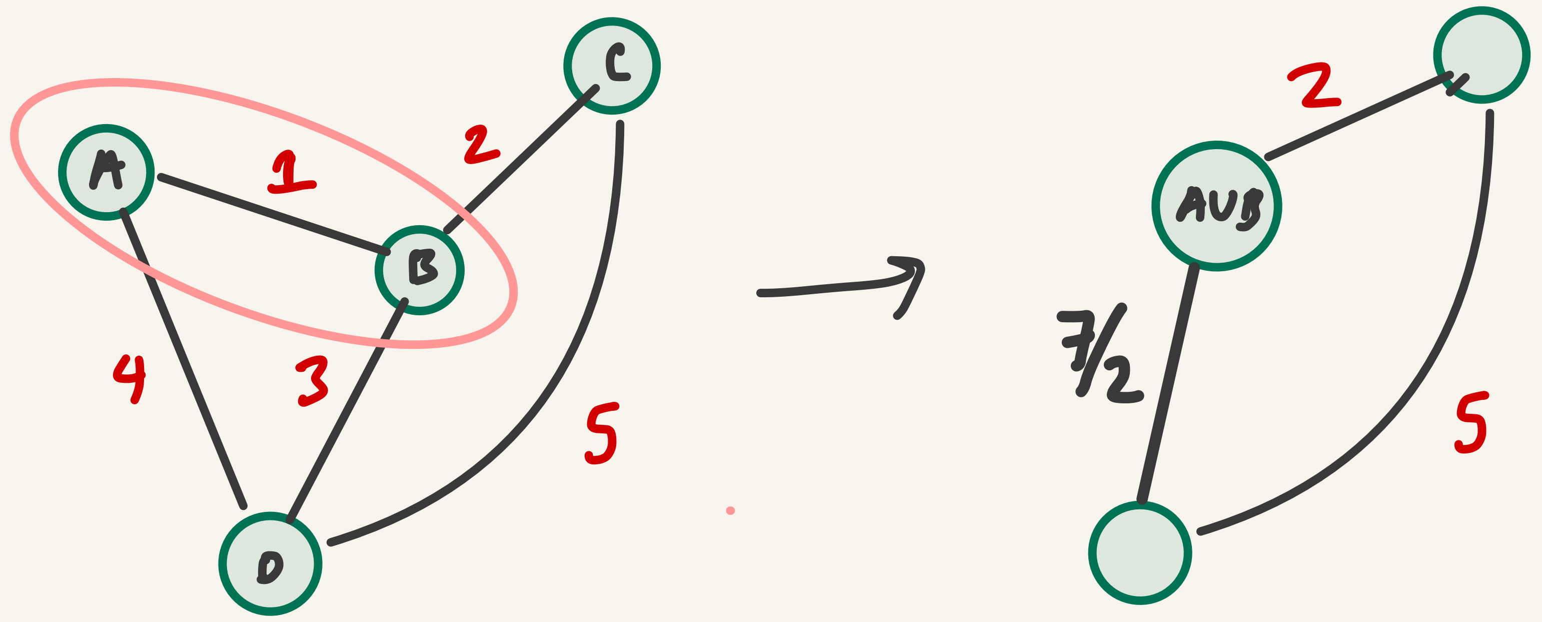
Single Linkage: $w(A \cup B, C) = \min\{w(A,C), w(B,C)\}$



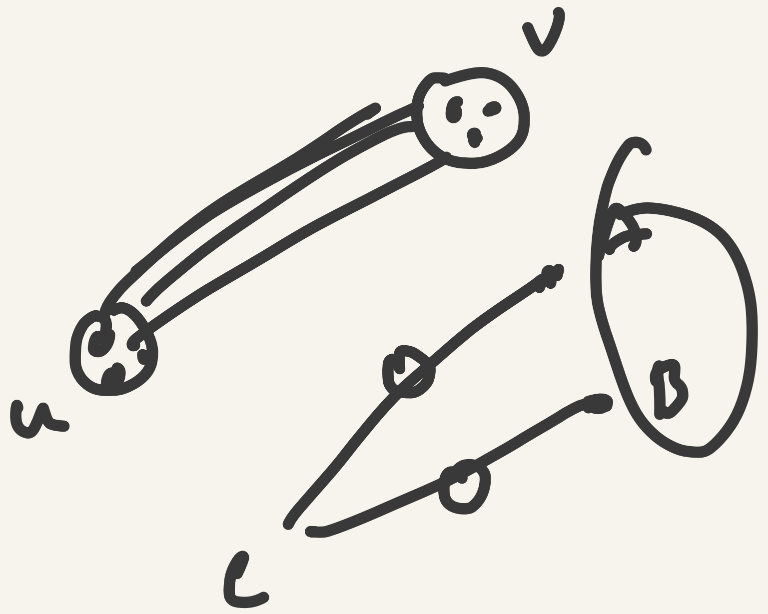
Complete Linkage: $w(A \cup B, C) = \max\{w(A,C), w(B,C)\}$



Weighted Average-Linkage: $w(A \cup B, C) = \left(w(A,C) + w(B,C)\right)/2$

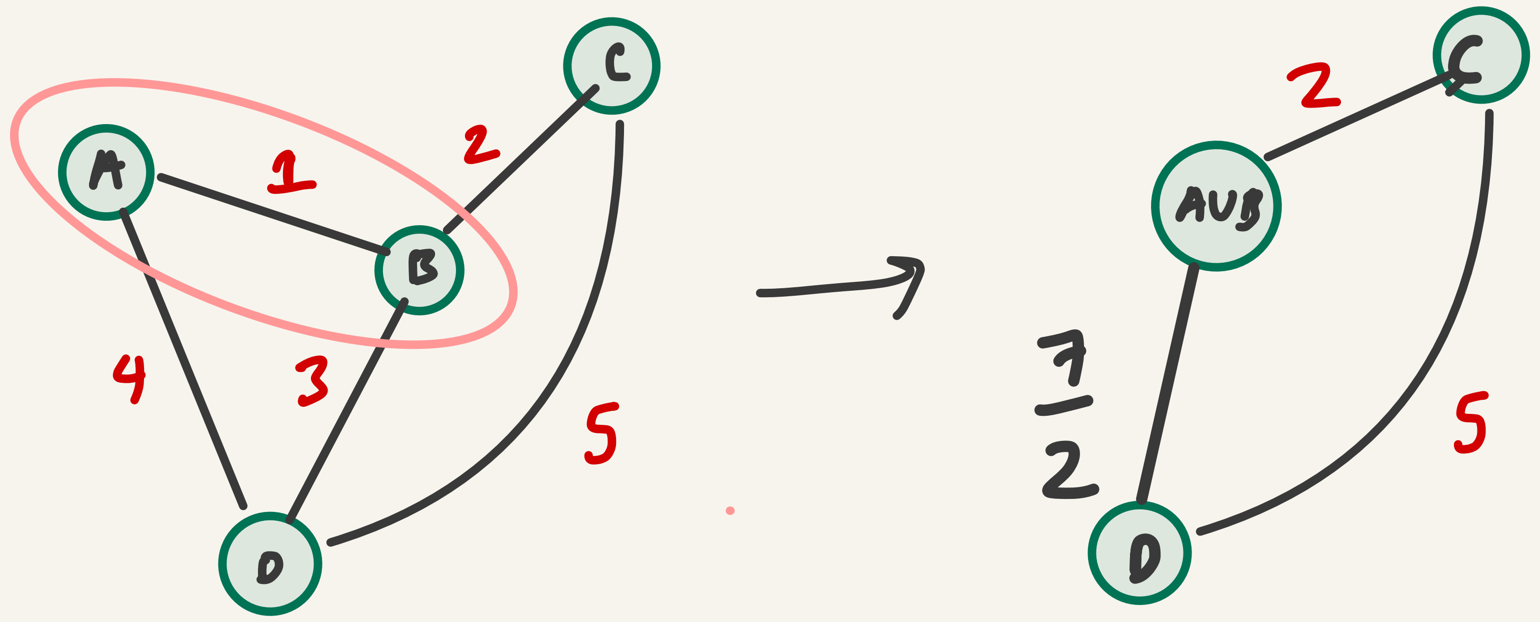


Linkage Functions Cont'd.

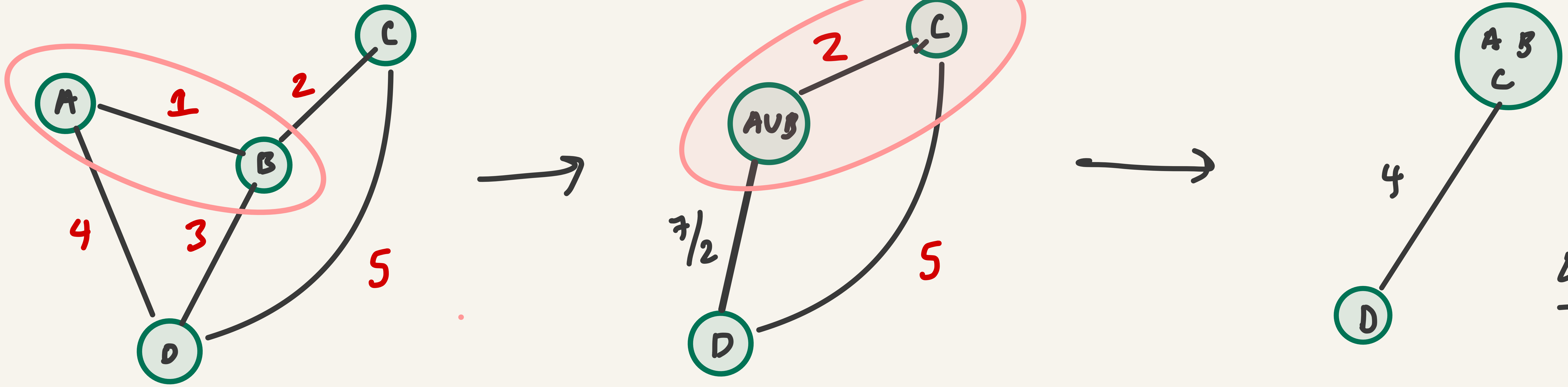


Unweighted Average-Linkage:

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



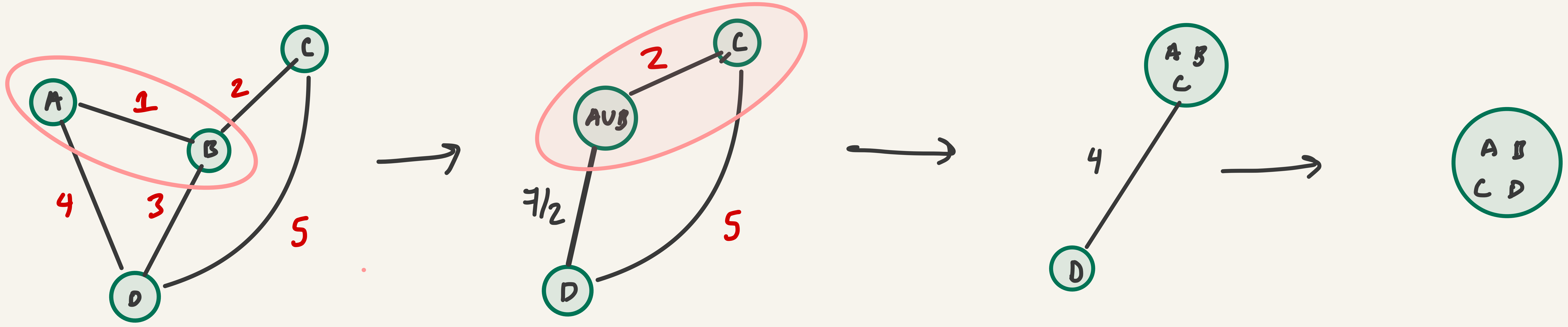
$$\frac{|A| \cdot W(A,C) + |B| \cdot W(B,C)}{|A| \cdot |B|}$$



$$\frac{4+3+5}{3 \cdot 1} = \underline{4}$$

weighted-avg $= 17/2$

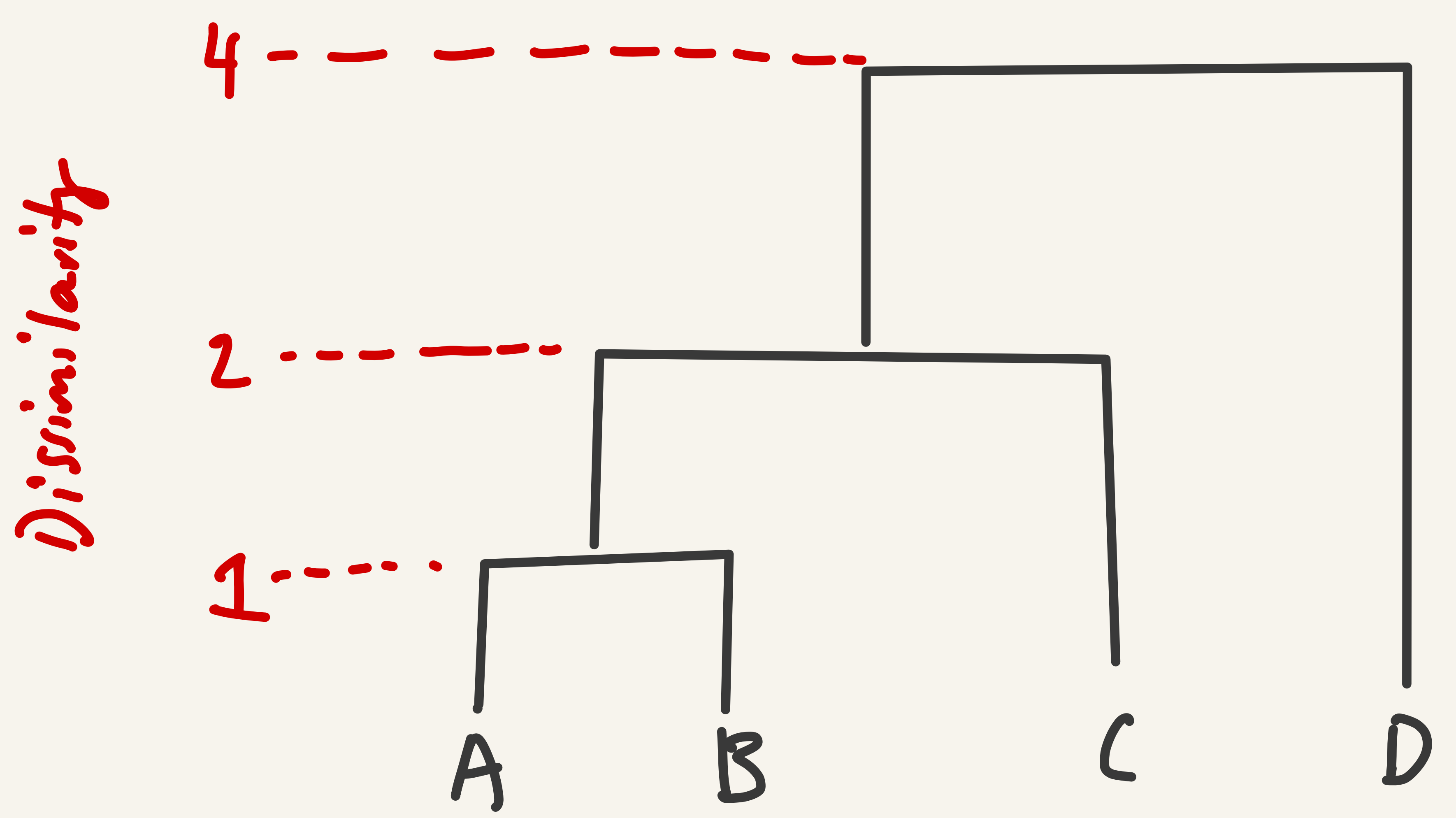
HAC: Output (unweighted avg-link)



merge weight: 1

merge weight: 2

merge weight: 4



Parallelizing Hierarchical Agglomerative Graph Clustering

Is it possible to solve this problem in NC?

Class of problems solvable in
- poly(n) work } today: want close to best seq. alg
- polylog(n) depth

RNC

What about using different linkage functions?

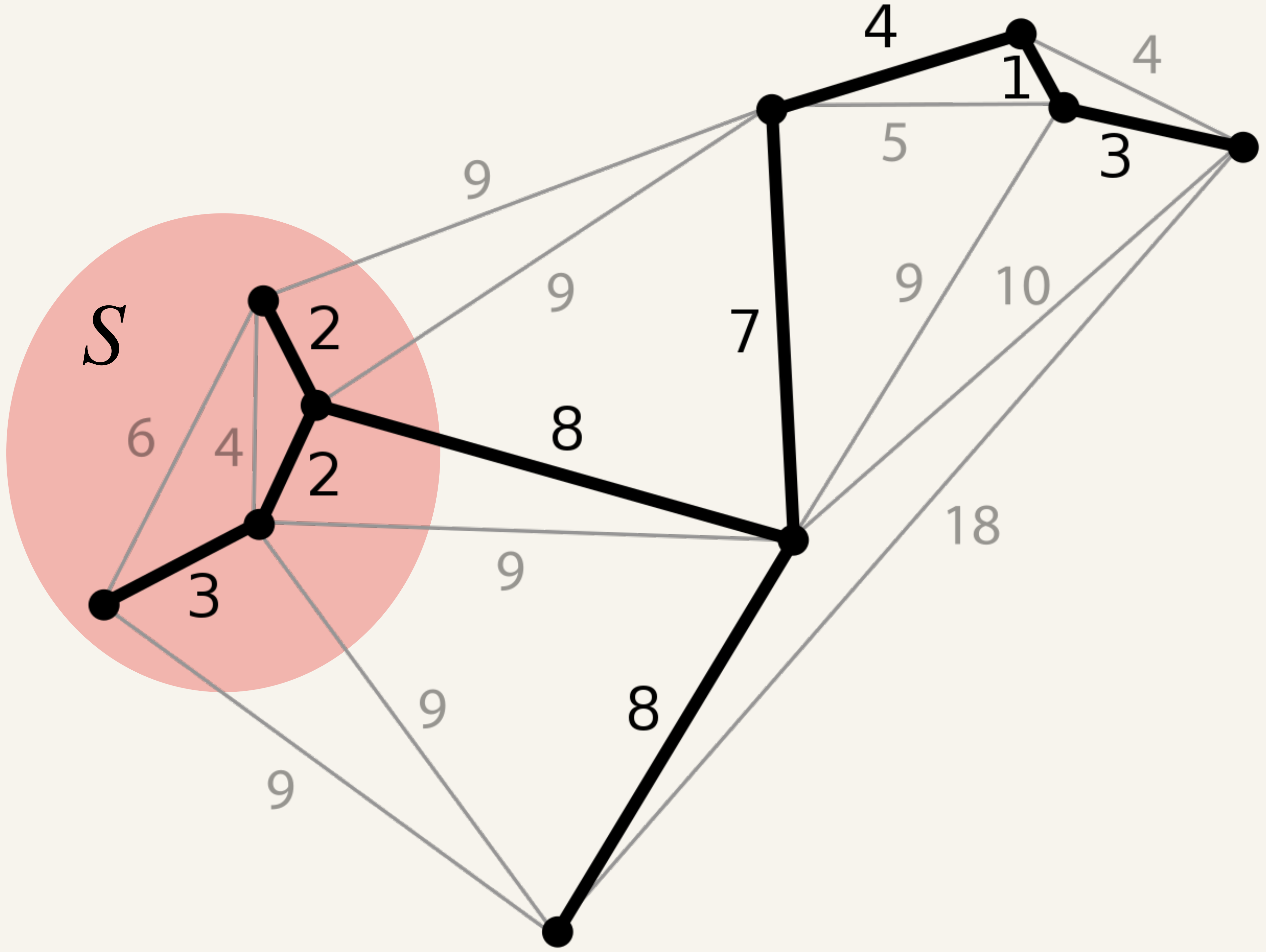
$\hat{O}(m+n)$ work
$\tilde{O}(n)$ depth
straightforward parallelization of generic HAC alg.

Single-Linkage
- MST(G) ENC → postprocesses MST + dendrogram ENC
  $O(n \log n)$ work, $O(\text{polylog}\, n)$ depth
- Complete Linkage → CC-hard — comparator circuit
  EP ↑ probably $\not\subset$ NC
- approx — Weighted/unweighted — P-complete

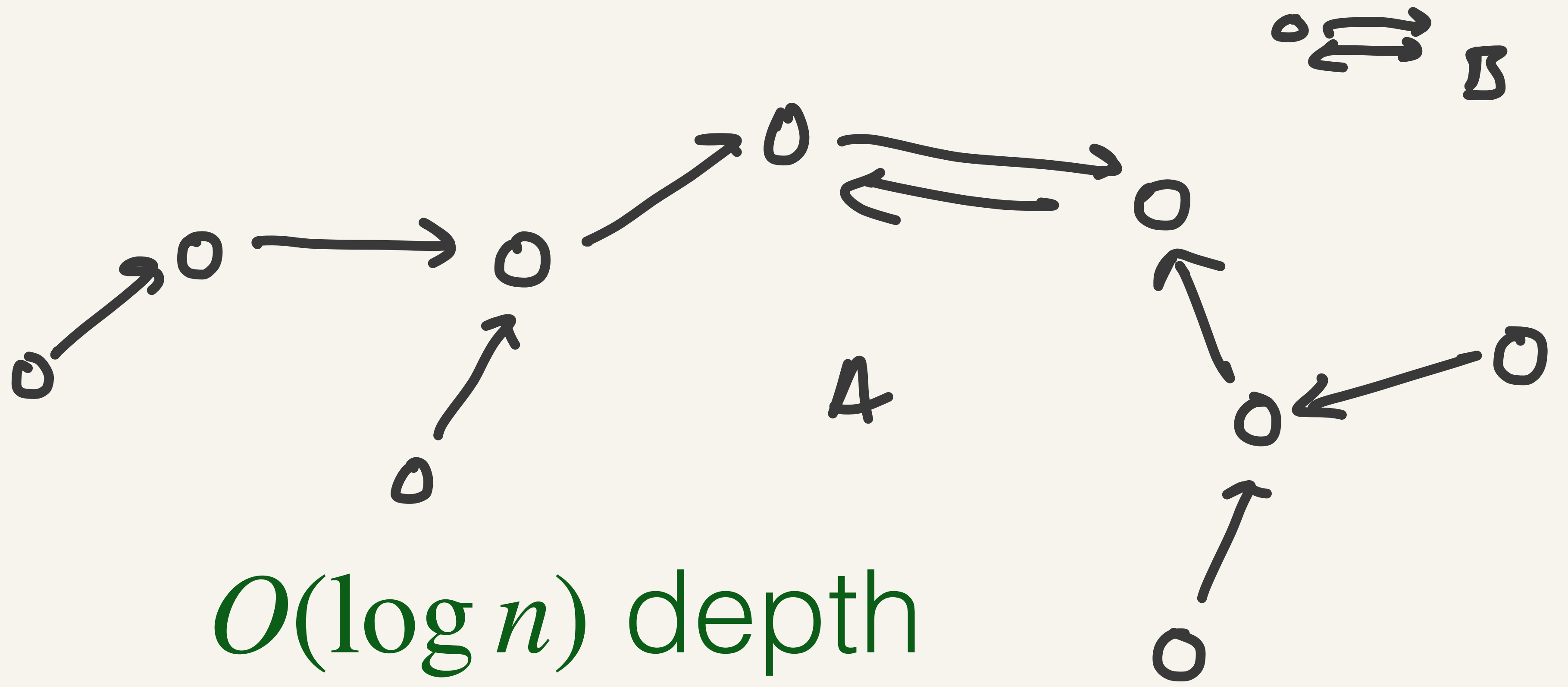
Background: Boruvka's Algorithm

Cut property:

Let S be any subset of vertices. The minimum cost edge on the boundary of S is in the MST.



Background: Boruvka's Algorithm

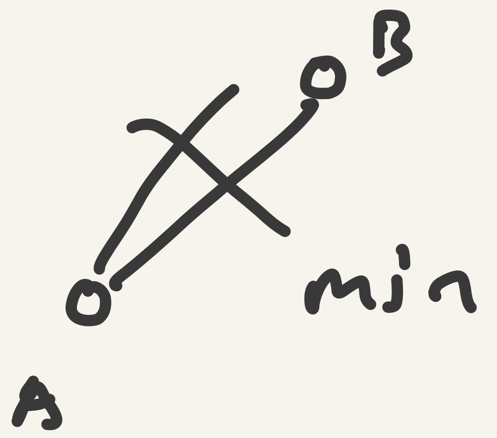



```
def Boruvka(G(V, E, w)):
  # Compute the minimum edge out of each vertex.
  Let the set of min-weight edges be MinE.

  # 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)
```

$O(\log n)$

$O(\log n)$ depth

$O(\log n)$ depth



$O(\log n)$ depth

$n/2$ components

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 $\quad O(\log^2 n)$ depth

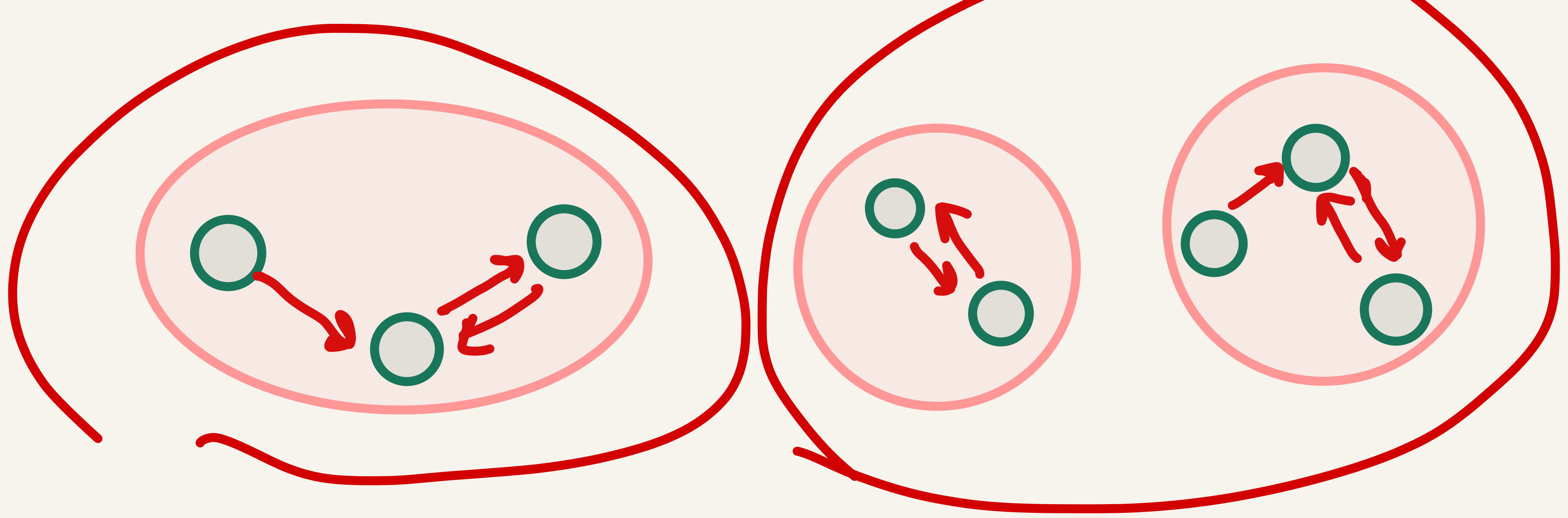
$n \to n/2 \cdot - \to O(1)$

Affinity Clustering

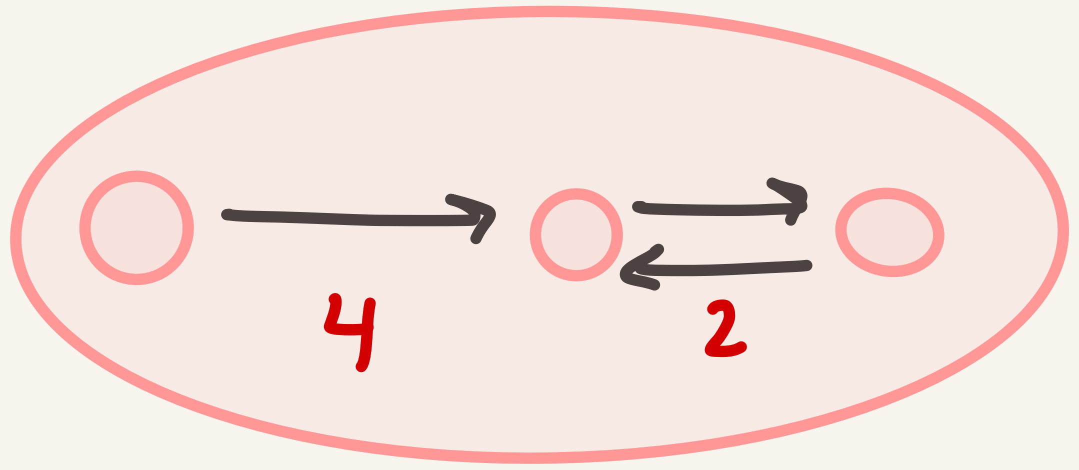
Idea: stop Borůvka's Alg. after $r > 0$ rounds, at the first time when there are $\leq k$ clusters for some desired # $k > 0$.

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

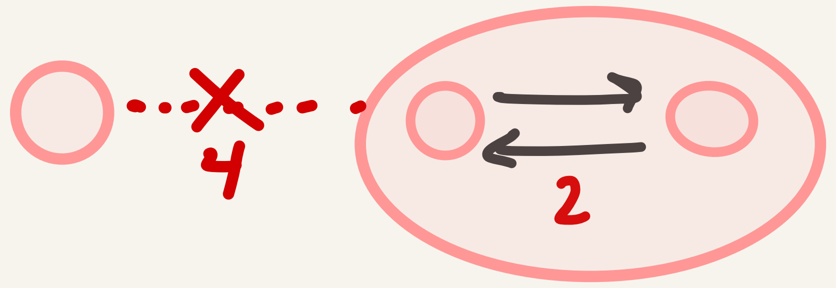
Round 1



Round 2



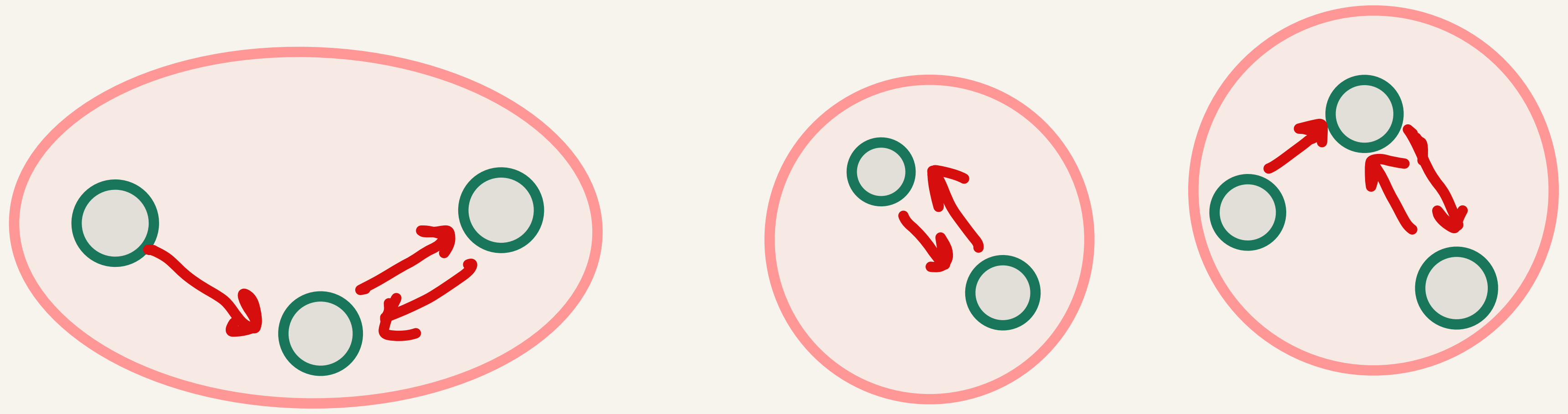
if $k = 2$, cut the weight 4 edge to get 2 clusters



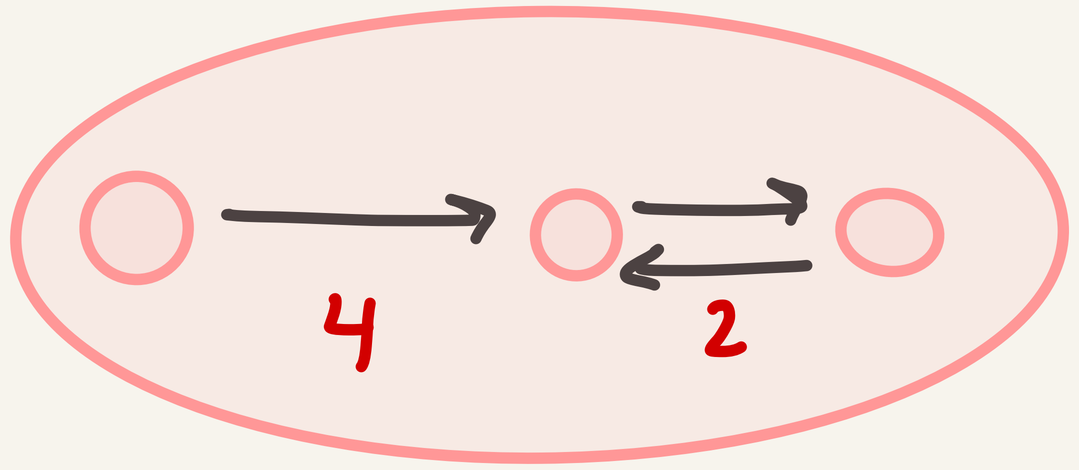
Hierarchical Affinity Clustering

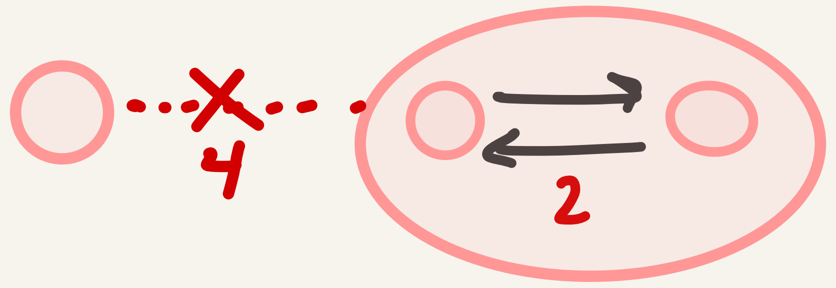
K = 2

Round 1

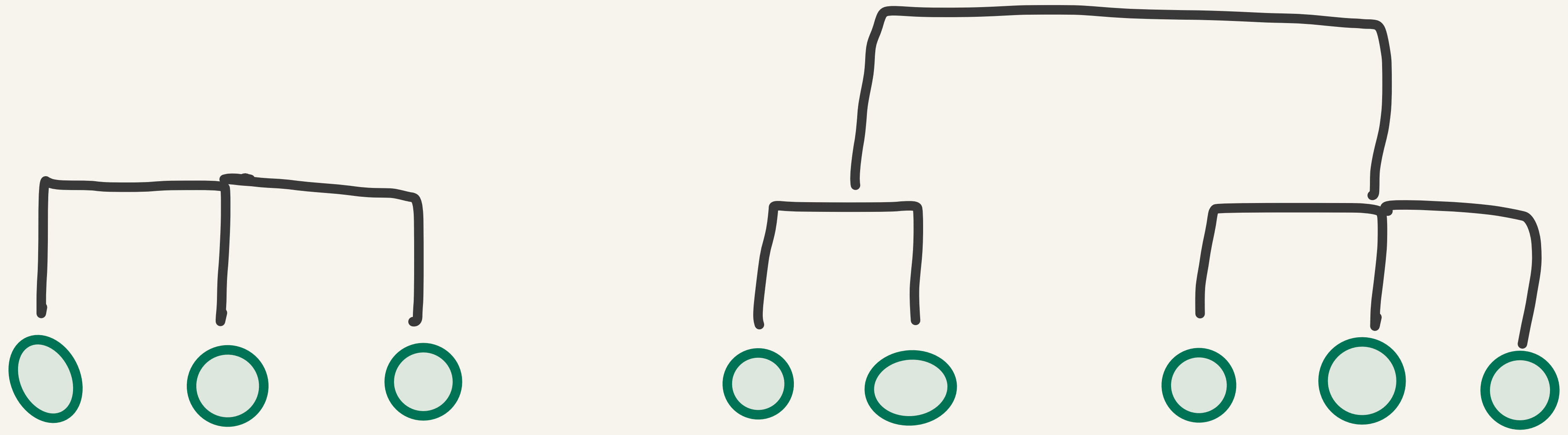


Round 2





Hierarchical Affinity Clustering:



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

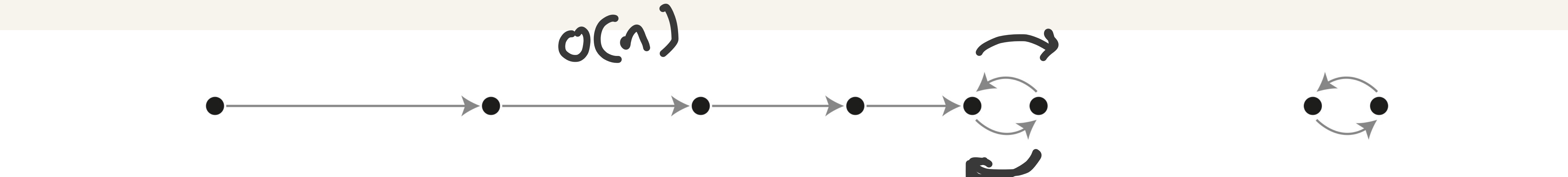
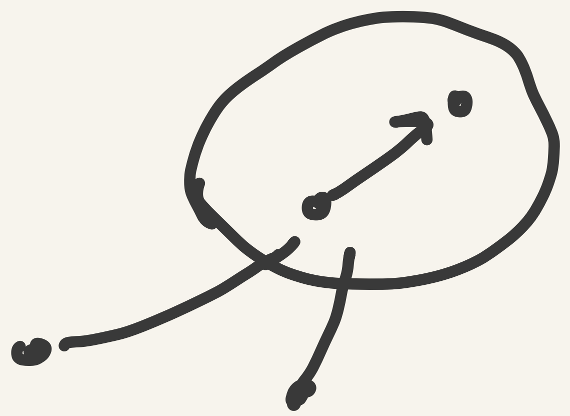


Figure 1: An example of how affinity may produce a large component in one round.

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 ≥ 2)
- Characterizations 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 vertices outside cluster



Algorithmic Contributions:

- $O(1)$ round MPC algorithm for MST for dense graphs
  - $m = \Theta(n^{1+c})$ for any constant $c > 0$
  - space-per-machine $= S = \tilde{O}(n^{1+\varepsilon})$ w.h.p. for $0 < \varepsilon < c$
  - total machines $= T = O(n^{c-\varepsilon})$

  $\Big\}$ space-efficient wrt input up to polylog factors

  $\tilde{O}(m+n)$

  $\rightarrow$ runs in $\lceil \log(c/\varepsilon) \rceil + 1$ rounds of MPC

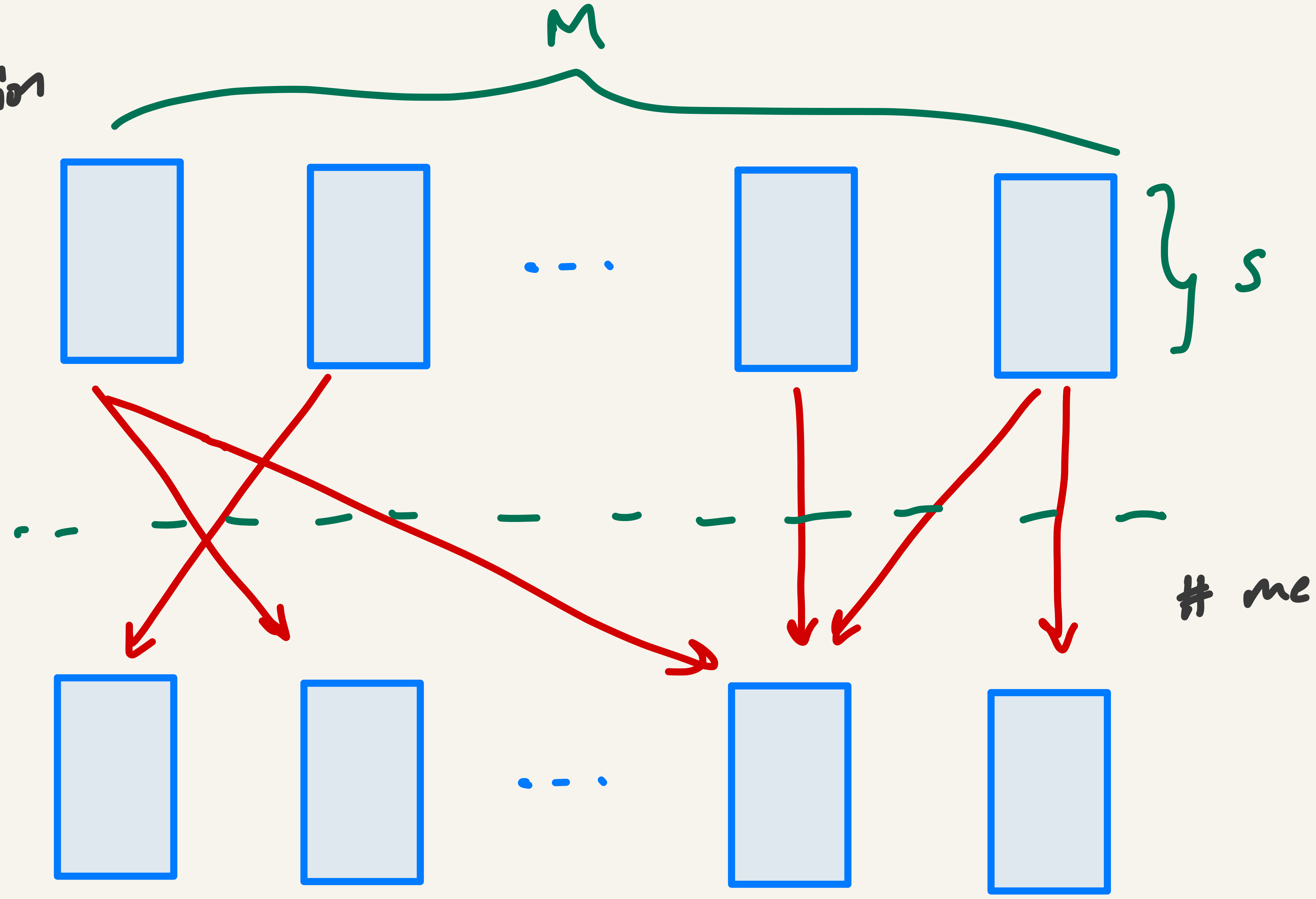
- $O(\log n)$ round MPC algorithm using Distributed Hash Tables (DHT)

  $\rightarrow O(\log^2 n)$ 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. $M = O(N^{1-\varepsilon})$ $S = O(N^{\varepsilon})$ for constant $\varepsilon > 0$

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



Round 1

messages sent (recieved = $O(S)$

Round 2

MST Algorithm (Dense Graphs)

$S = N = m$

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

→ if $S = O(n^{1+c})$ 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 $e' \in E' \notin MST(G')$ Then $e' \notin MST(G)$

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

↓

Repeat until only $S = O(n^{1+\varepsilon})$ edges left and solve on a single machine.

MST Algorithm (Dense Graphs)

$x = \log_n(m/n) = \log_n(n^{1+c}/n)$

$= \log_n(n^c)$

$= c$

Algorithm 1 MST of Dense Graphs

Input: A weighted graph G

Output: The minimum spanning tree of G

function MST($G = (V, E)$, ϵ) S

  $c \leftarrow \log_n(m/n)$ ▷ Since G is assumed to be dense we know $c > 0$.

  while $|E| > O(n^{1+\epsilon})$ **do** // while not yet solvable on single machine

    ReduceEdges(G, c)

    $c \leftarrow (c - \epsilon)/2$

  Move all the edges to one machine and find MST of G in there.

function ReduceEdges($G = (V, E)$, c)

  $k \leftarrow n^{(c-\epsilon)/2}$

  Independently and u.a.r. partition V into k subsets $\{V_1, \ldots, V_k\}$.

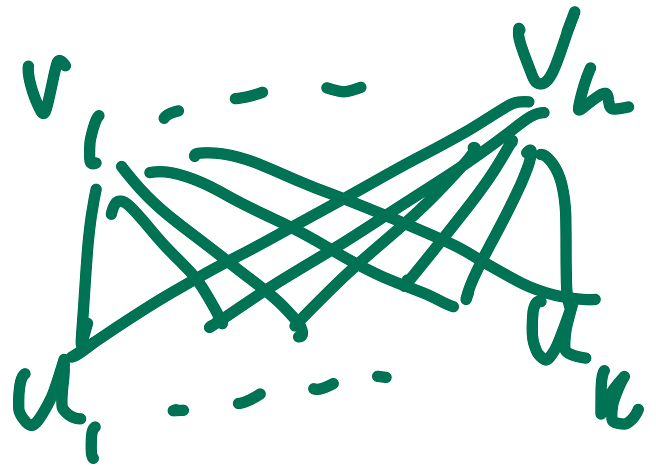
  Independently and u.a.r. partition V into k subsets $\{U_1, \ldots, U_k\}$.

  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 $u \in U_j$.

  for any $i, j \in \{1, \ldots, k\}$ **do**

    Send all the edges of $G_{i,j}$ to the same machine and find its MST in there.

    Remove an edge e from $E(G)$, if $e \in G_{i,j}$ and it is not in MST of $G_{i,j}$.



$G_{i,j} = (V_i \cup U_j, \{(v,u) \in E \mid v \in V_i \text{ and } u \in U_j\})$

$|\text{MST on } n \text{ vertices}| \leq n-1$

$|\text{MST}(G_{i,j})| \leq |V_i| \cup |U_j| - 1$

[Lemma: Alg. 1 correctly finds the MST in $\lceil \log(c/\varepsilon) \rceil + 1$ rounds]

Correctness: each call to Reduce Edges

randomly partitions vertices into

$V = \{V_1, \dots V_k\}$

$U = \{U_1, \dots U_k\}$

And for each (i,j) pair $\in \{1 \dots k\}$

finds $MST(G_{i,j})$, discarding any

edges in $G_{i,j} \notin MST(G_{i,j})$. $\rightarrow$ none of the discarded edges are part of $MST(G)$

Algorithm 1 MST of Dense Graphs

Input: A weighted graph G
Output: The minimum spanning tree of G

function MST($G = (V, E), \epsilon$)
  $c \leftarrow \log_n (m/n)$ ▷ Since G is assumed to be dense we know $c > 0$.
  while $|E| > O(n^{1+\epsilon})$ **do**
    REDUCEEDGES(G, c)
    $c \leftarrow (c - \epsilon)/2$
  Move all the edges to one machine and find MST of G in there.
function REDUCEEDGES($G = (V, E), c$)
  $k \leftarrow n^{(c-\epsilon)/2}$
  Independently and u.a.r. partition V into k subsets $\{V_1, \dots, V_k\}$.
  Independently and u.a.r. partition V into k subsets $\{U_1, \dots, U_k\}$.
  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 $u \in U_j$.
  for any $i, j \in \{1, \dots, k\}$ **do**
    Send all the edges of $G_{i,j}$ to the same machine and find its MST in there.
    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 = value of c in r-th iter.
- Let $k_r = n^{(c_r - \epsilon)/2}$

Algorithm 1 MST of Dense Graphs

Input: A weighted graph G
Output: The minimum spanning tree of G

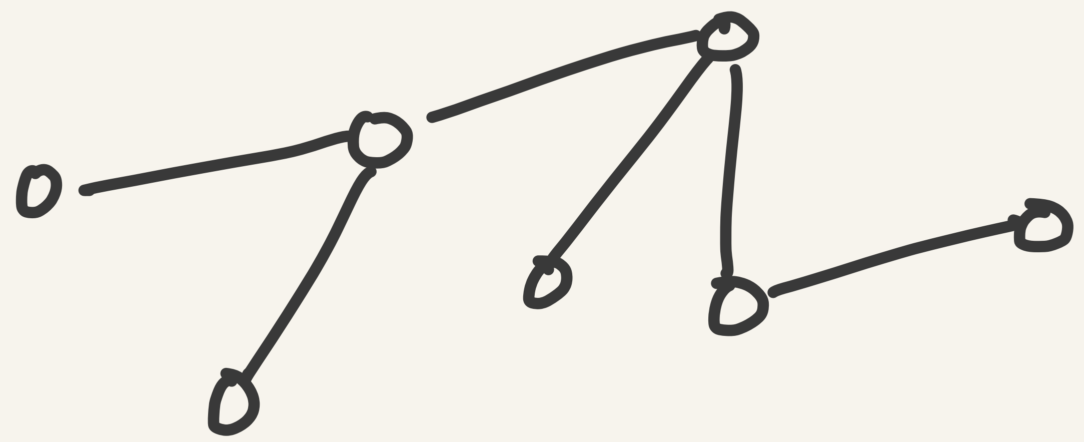
function MST($G = (V, E)$, ϵ)
  $c \leftarrow \log_n (m/n)$ ▷ Since G is assumed to be dense we know $c > 0$.
  while $|E| > O(n^{1+\epsilon})$ **do**
    REDUCEEDGES(G, c)
    $c \leftarrow (c - \epsilon)/2$
  Move all the edges to one machine and find MST of G in there.
function REDUCEEDGES($G = (V, E)$, c)
  $k \leftarrow n^{(c-\epsilon)/2}$
  Independently and u.a.r. partition V into k subsets $\{V_1, \ldots, V_k\}$.
  Independently and u.a.r. partition V into k subsets $\{U_1, \ldots, U_k\}$.
  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 $u \in U_j$.
  for any $i, j \in \{1, \ldots, k\}$ **do**
    Send all the edges of $G_{i,j}$ to the same machine and find its MST in there.
    Remove an edge e from $E(G)$, if $e \in G_{i,j}$ and it is not in MST of $G_{i,j}$.

- For each $G_{i,j}$ let $T_{i,j} = \text{MST}(G_{i,j})$. Notice that only $\boxed{\bigcup_{i,j} T_{i,j}}$ are kept in next round.

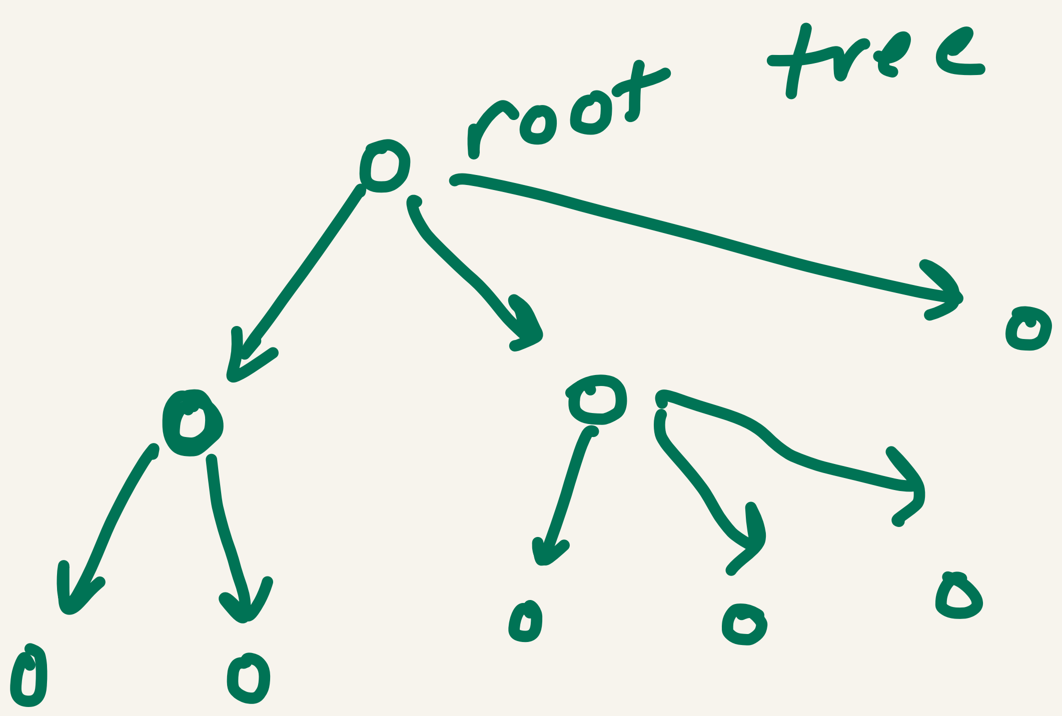
$\rightarrow$ MST on n' vertices has $\leq n' - 1$ edges.

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

Claim: each vertex in $G_{i,j}$ charged at most once

e.g. 

Idea?



$c_0 = c\,, \quad c_r = (c_{r-1} - \varepsilon)/2$

Round Complexity:

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

Algorithm 1 MST of Dense Graphs

Input: A weighted graph G

Output: The minimum spanning tree of G

function MST($G = (V, E)$, ϵ)
    $c \leftarrow \log_n (m/n)$ ▷ Since G is assumed to be dense we know $c > 0$.
    while $|E| > O(n^{1+\epsilon})$ **do**
        REDUCEEDGES(G, c)
        $c \leftarrow (c - \epsilon)/2$
    Move all the edges to one machine and find MST of G in there.
function REDUCEEDGES($G = (V, E)$, c)
    $k \leftarrow n^{(c-\epsilon)/2}$
    Independently and u.a.r. partition V into k subsets $\{V_1, \ldots, V_k\}$.
    Independently and u.a.r. partition V into k subsets $\{U_1, \ldots, U_k\}$.
    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 $u \in U_j$.
    for any $i, j \in \{1, \ldots, k\}$ **do**
        Send all the edges of $G_{i,j}$ to the same machine and find its MST in there.
        Remove an edge e from $E(G)$, if $e \in G_{i,j}$ and it is not in MST of $G_{i,j}$.

Consider $v \in V_i$ (WLOG). v appears in

$\underbrace{G_{i,1}, \ldots, G_{i,k_r}}_{k_r \text{ times}}$. Therefore v can be charged for at most k_r edges.

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

$\Rightarrow$ $k_r \cdot n = n^{1 + (c_r - \varepsilon)/2}$ and $c_r < \dfrac{c}{2^r}$.

$$c_{\lceil \log(c/\varepsilon) \rceil} < \frac{c}{2^{\lceil \log(c/\varepsilon) \rceil}} \leq \varepsilon$$

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

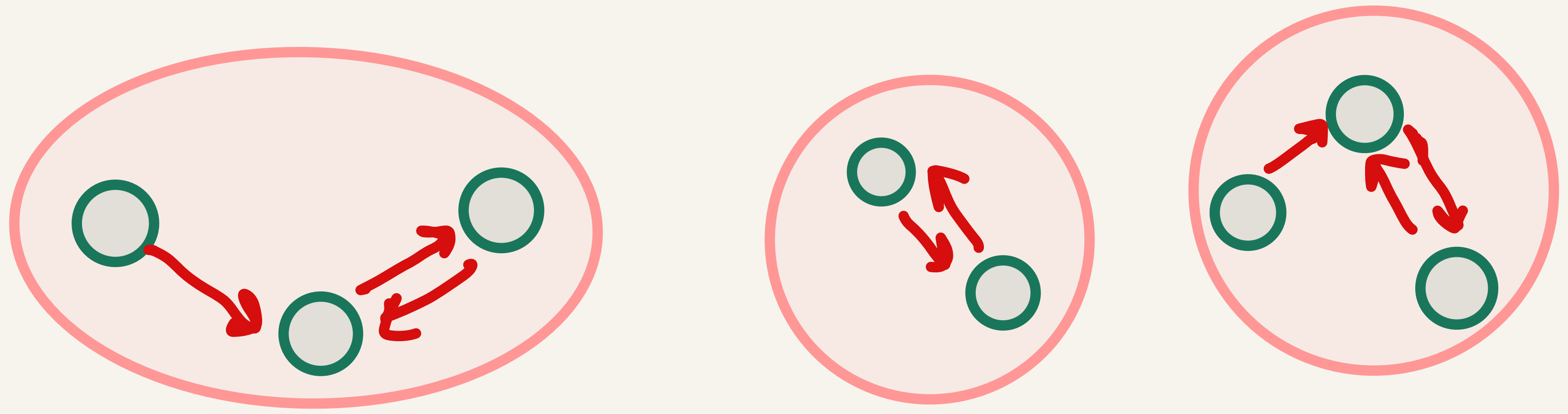
MST Algorithm (Sparse Graphs)

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

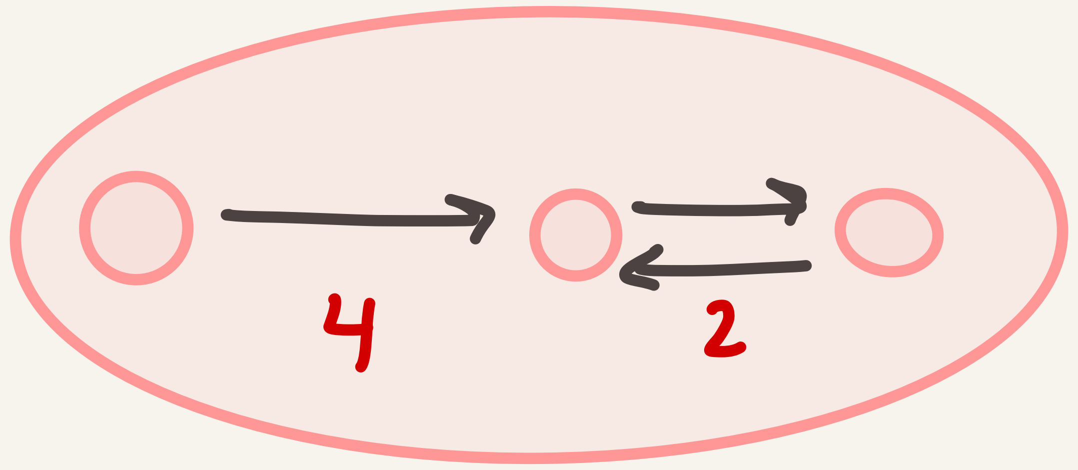
Algorithm: (proceed in rounds)

- each vertex finds its best edge (most similar edge)
- graph is contracted along the selected edges

Round 1



Round 2



(EREW / CRCW)
PRAM < multi-prefix PRAM

MPC

$O(\log \log n)$

$O(1)$ 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) // $O(1)$ rounds

(2) graph is contracted along the selected edges // ?? rounds

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

Turns 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)

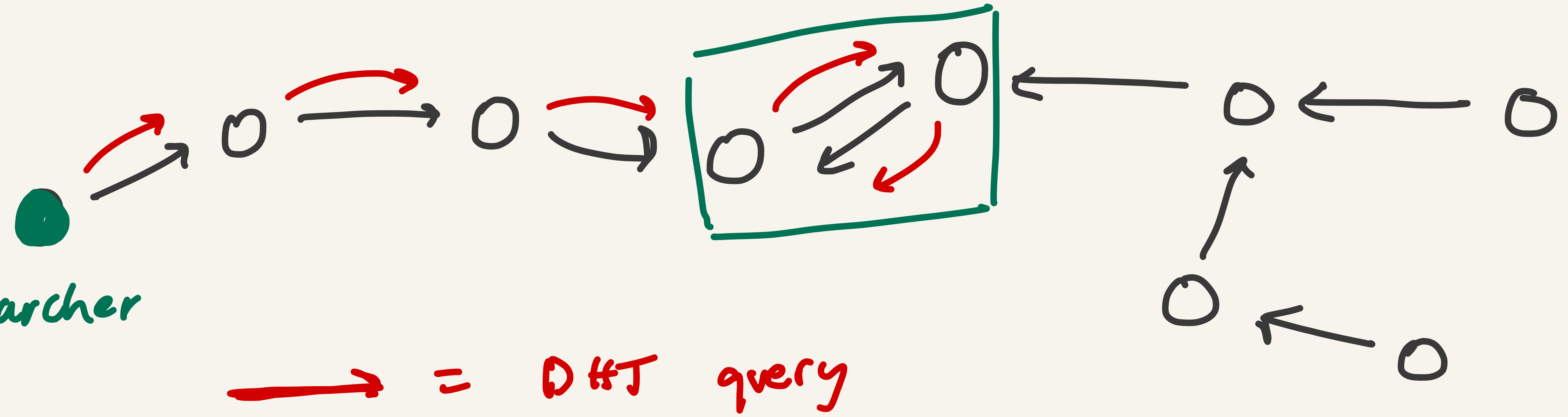
(1) each vertex finds its best edge (most similar edge)

(2) graph is contracted along the selected edges using DHT

$S = n^{\varepsilon}$ $\varepsilon < 1$

$O(S)$ queries

$O(n)$ queries to DHT



$O(1)$ MST

- 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 $r(X, T)$.

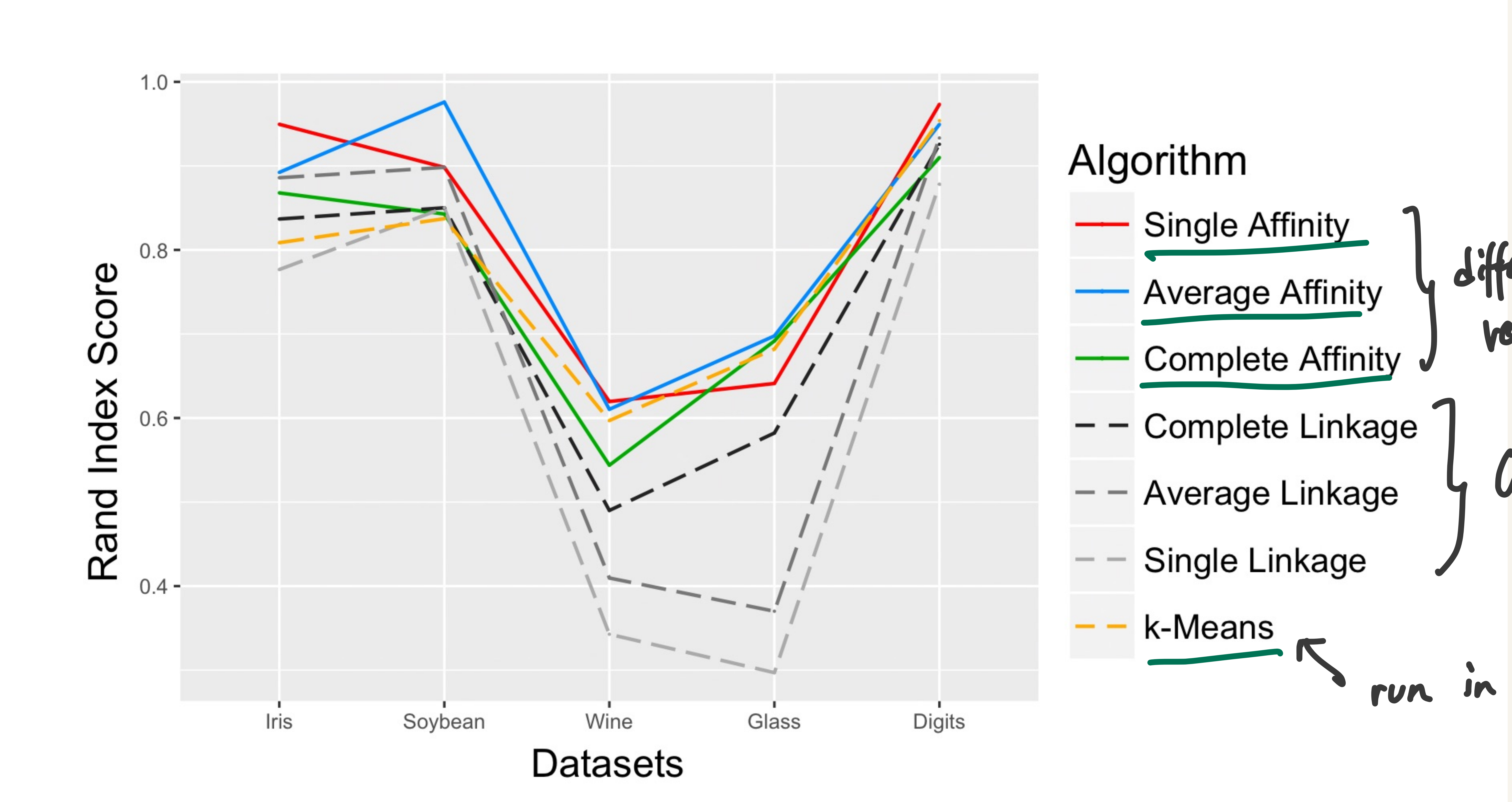
E.g. if $n = 4$, $V = \{v_1, v_2, v_3, v_4\}$

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

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

$r(X, T) =$

Evaluation



different linkage versions of affinity.

Idea is to reweight edge after each round.

Graph-based HAC on complete graph.

run in original metric

- 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.

Scalability

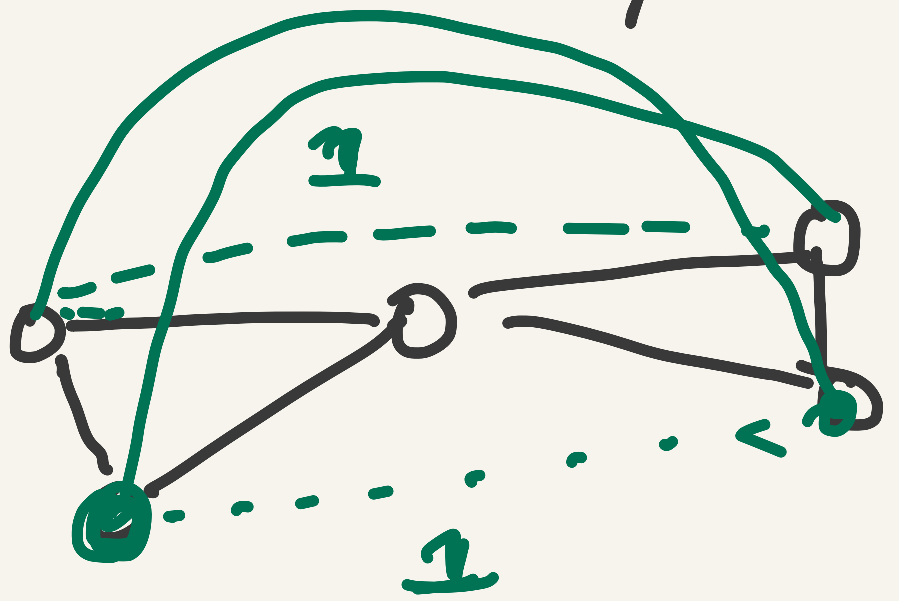
Table 1: Statistics about datasets used. (Numbers for ImageGraph are approximate.) The fifth column shows the relative running time of affinity clustering, and the last column is the speedup obtained by a ten-fold increase in parallelism.

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

increase W by 10x. See 4—10x speedup.

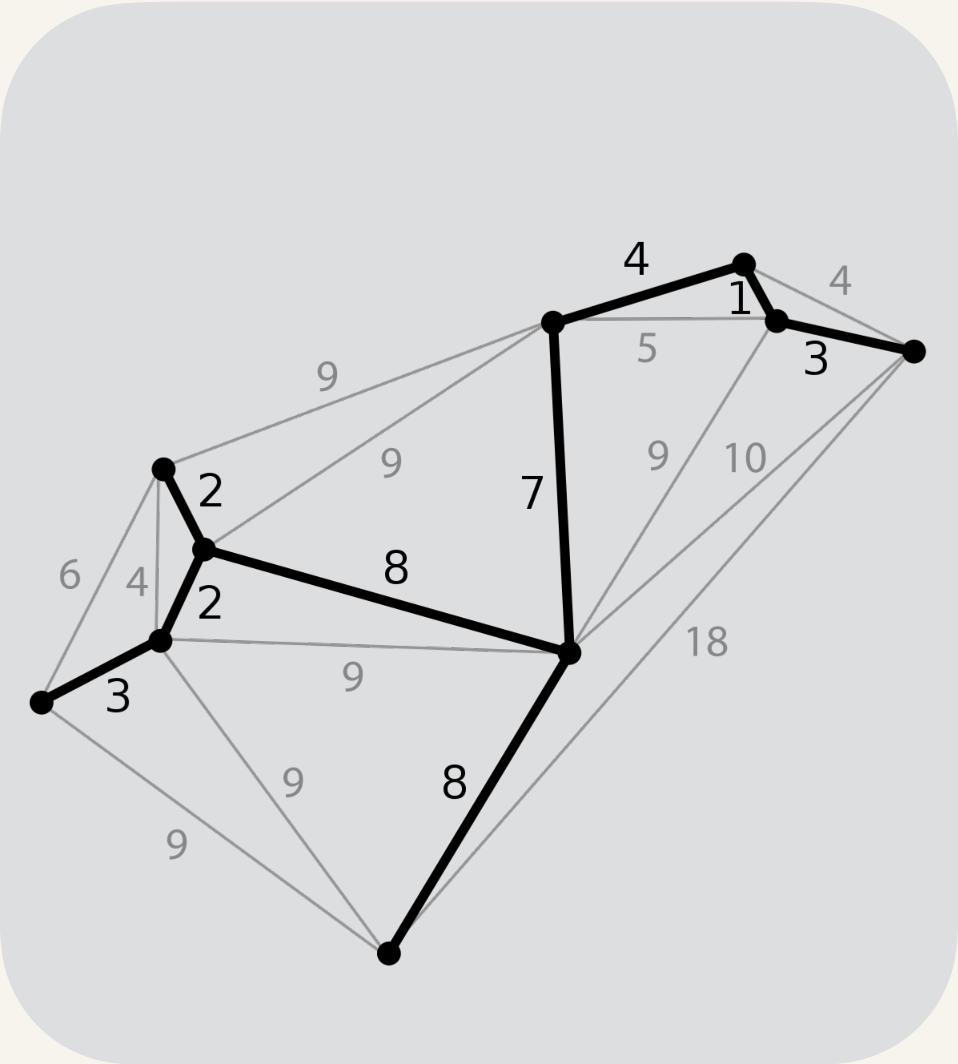
- Construct weighted graphs by setting $w(u,v) = N(u) \cap N(v)$ (number of common neighbors) and discarding 0-weight edges.

→ procedure basically connects a vertex's 2-hop neighborhood

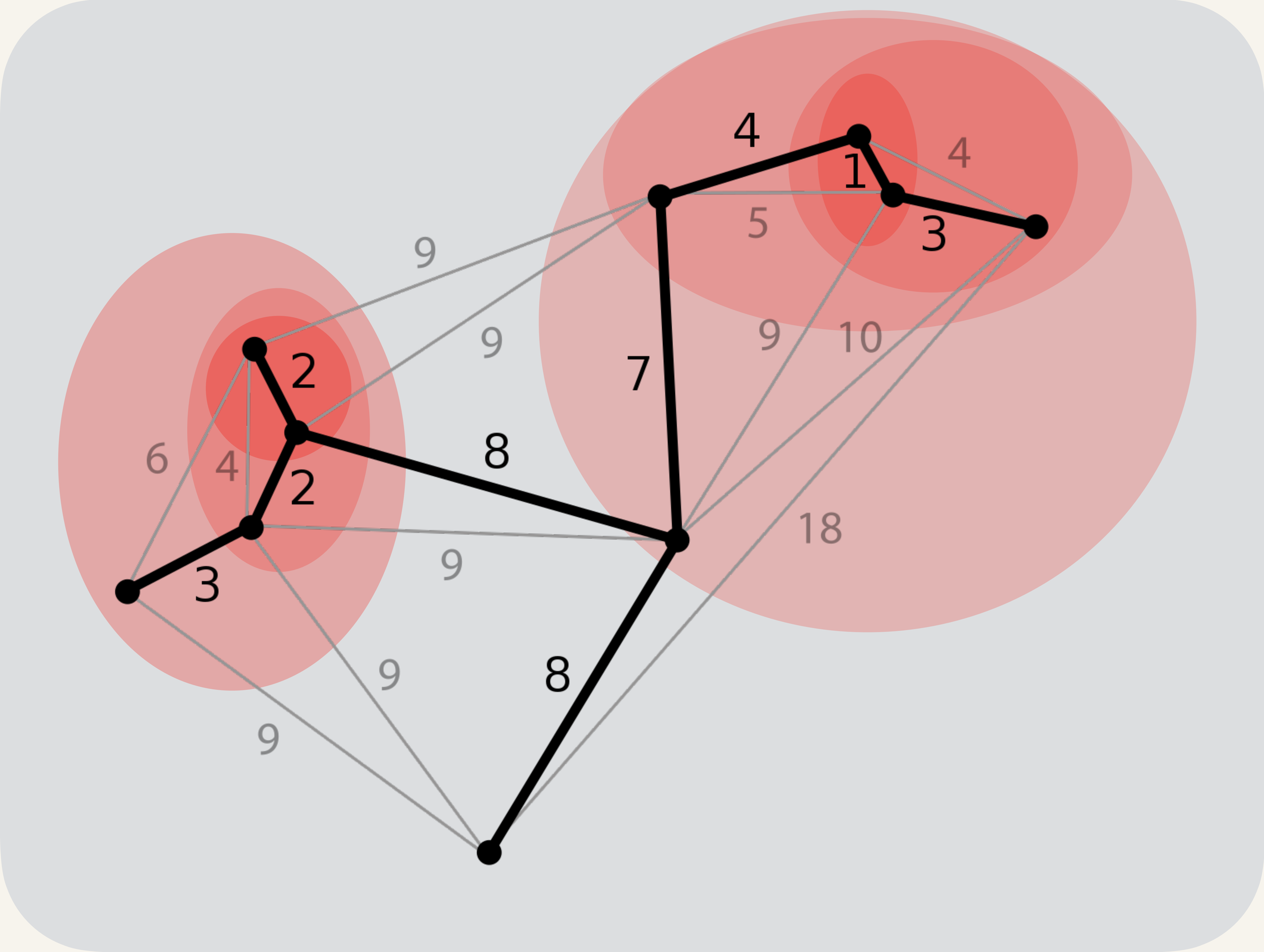


→ use maximum spanning tree (similarity) version of affinity.

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)

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