TDA - Fall 2025

Clustering & TDA

Clustering

Unsupervised learning/clustering is essentially 0-dimensional topological inference:

understanding connectivity or closeness of points

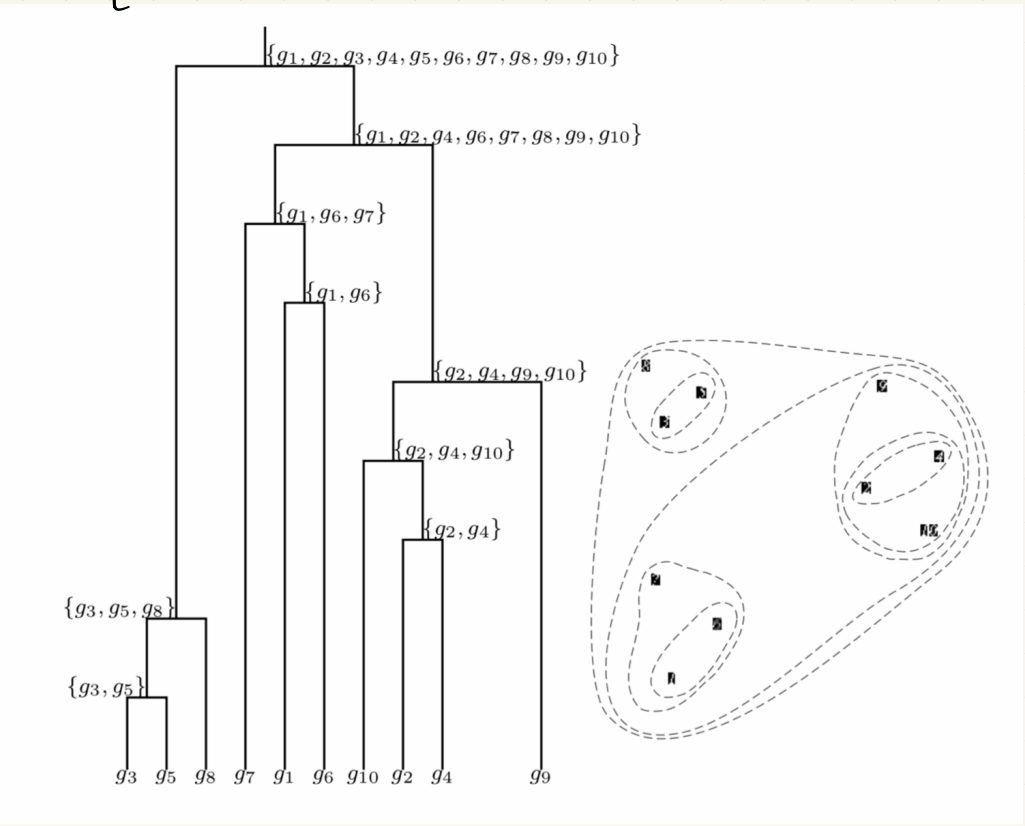
High level:

Input: Points $P \subset \mathbb{R}^n$
(possibly also $k \in \mathbb{N}$)

Output: A set of k clusters which best reflect connectivity (or "nearness")

Hierarchical Clustering

Goal is to build this in layered fashion - common in computation biology in particular:



Approach: Be greedy

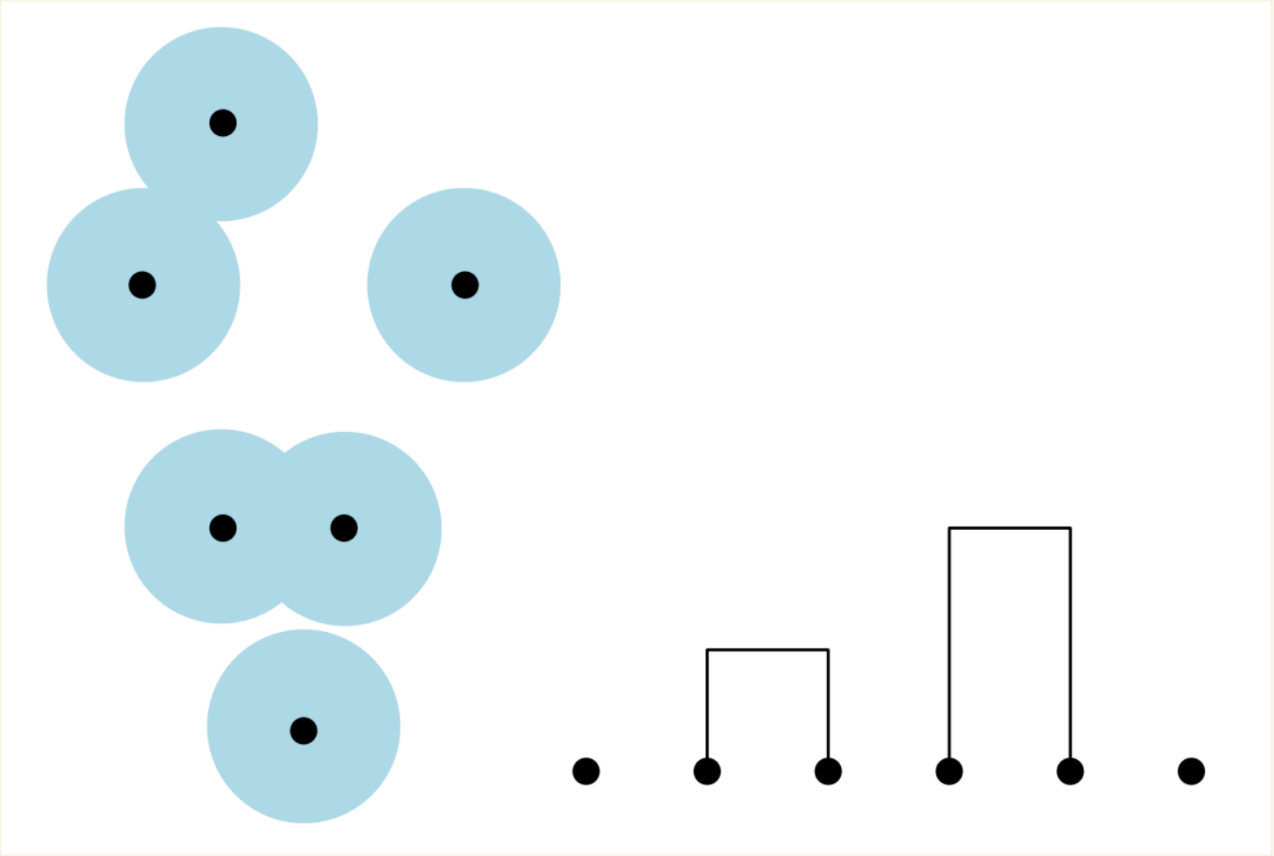
HIERARCHICALCLUSTERING$(\mathbf{d}, n)$

```
1  Form n clusters, each with 1 element
2  Construct a graph T by assigning an isolated vertex to each cluster
3  while there is more than 1 cluster
4      Find the two closest clusters C1 and C2
5      Merge C1 and C2 into new cluster C with |C1| + |C2| elements
6      Compute distance from C to all other clusters
7      Add a new vertex C to T and connect to vertices C1 and C2
8      Remove rows and columns of d corresponding to C1 and C2
9      Add a row and column to d for the new cluster C
10 return T
```

Some imprecision here...

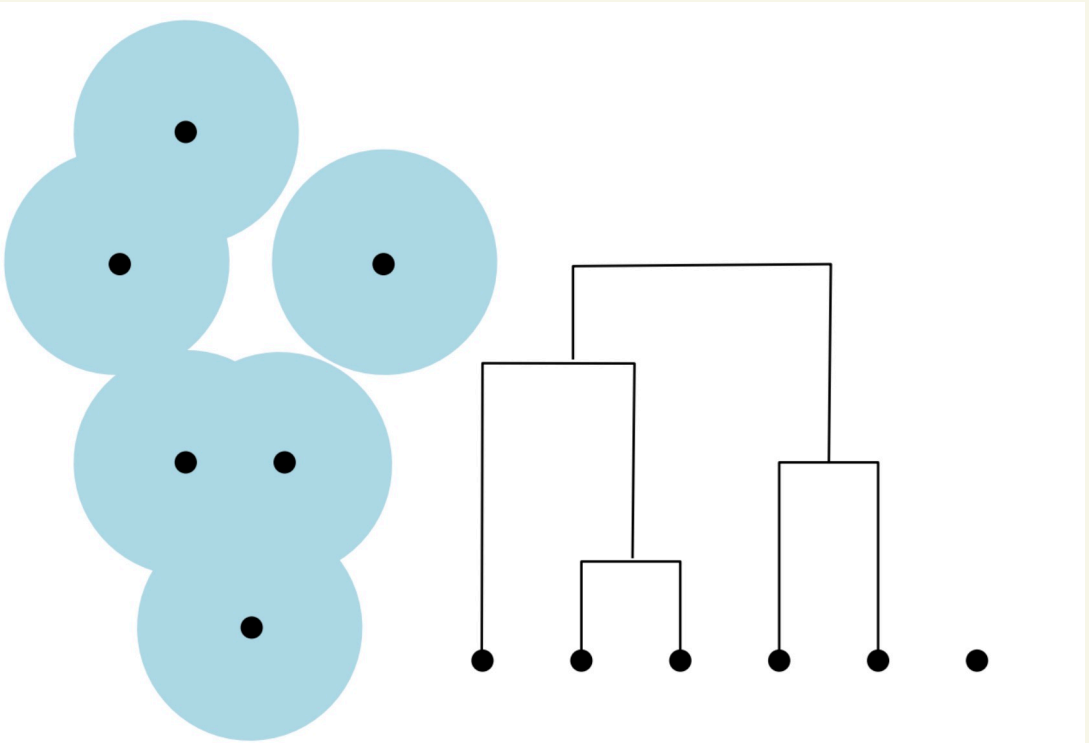
Connection to persistence: Via Rips complex

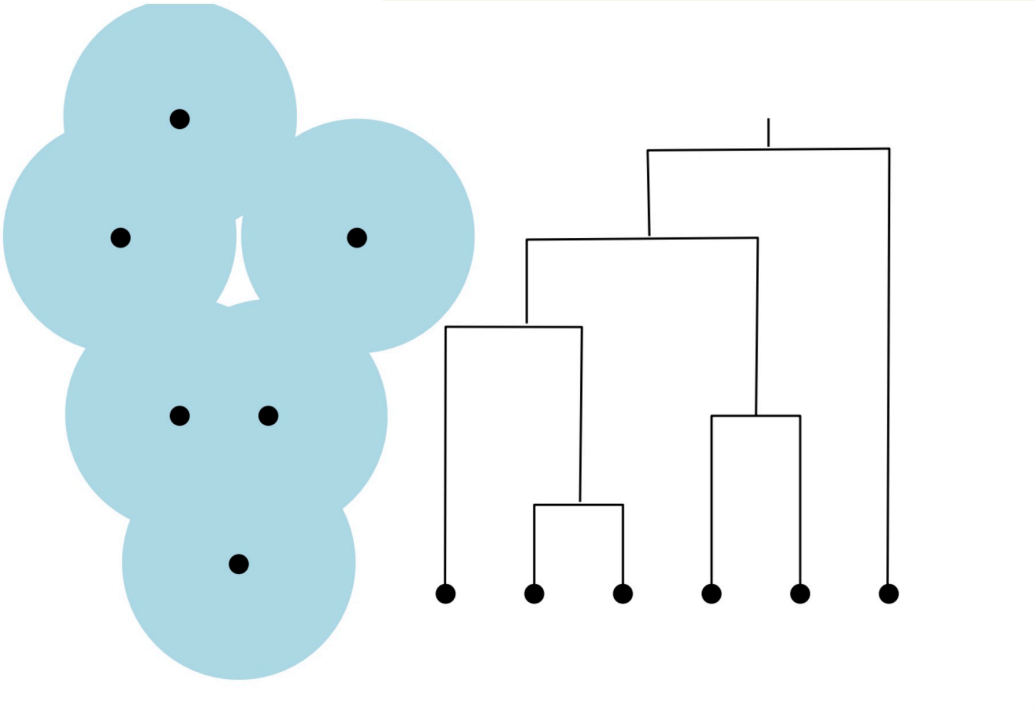
Barcodes / Merge trees



n points, with r-balls around them

grow r:

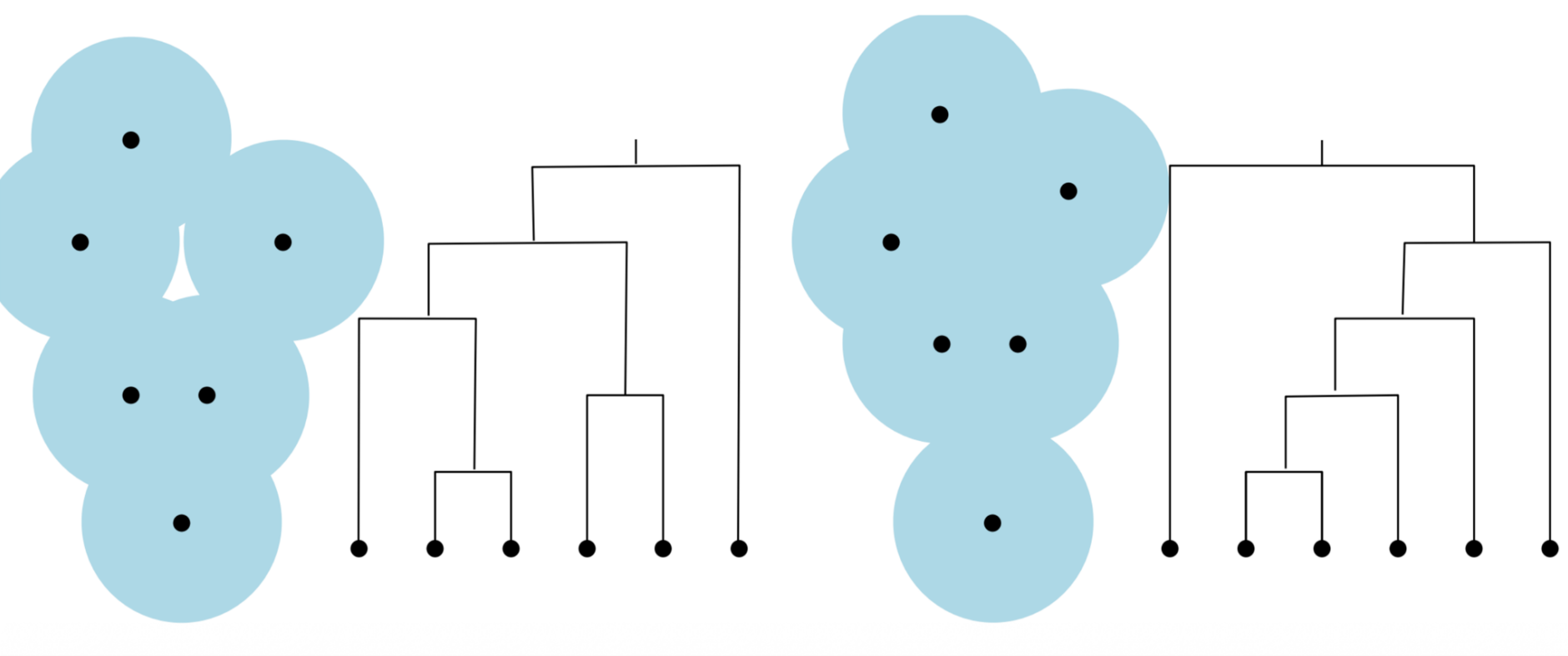




Observation:

Clustering is unstable

↳ but times are not!



Why?

Assignment-based clustering: K-means

Input: $X \subseteq \mathbb{R}^d$, $X = \{x_1, \ldots, x_n\}$, plus a distance $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

Output: A set of clusters $X_1, \ldots, X_k$, each associated with a point $s_i \in \mathbb{R}^d$

plus $\Phi_c : \mathbb{R}^d \to C$, $C = \{s_1, \ldots, s_k\}$

$\Phi_c(x) =$

Cost (k-means): $\min \sum_{x \in X}$

Lloyd's algorithm

Find set of k clusters to minimize:

$$\text{cost}(X, S) = \sum_{x \in X} \|\Phi_S(x) - x\|^2$$

A simple idea:

Choose k points $S \subset X$

Repeat:

for all $x \in S$, assign x to closest center s_i

Then, update to new s_i:

"average" s_i

until S does not change

In action:

Algorithm 8.3.1 Lloyd's Algorithm(X, k)

Choose k points $S \subset X$ *arbitrarily?*

repeat

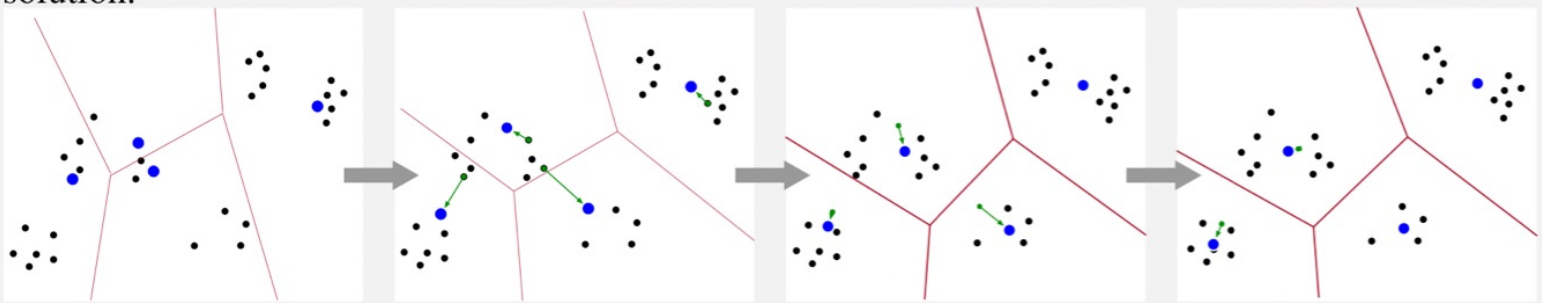
for all $x \in X$: assign x to X_i so $\phi_S(x) = s_i$ *the closest site $s_i \in S$ to x*

for all $s_i \in S$: update $s_i = \frac{1}{|X_i|} \sum_{x \in X_i} x$ *the average of $X_i = \{x \in X \mid \phi_S(x) = s_i\}$*

until (the set S is unchanged, *or other termination condition*)

Example: Lloyd's Algorithm for k-Means Clustering

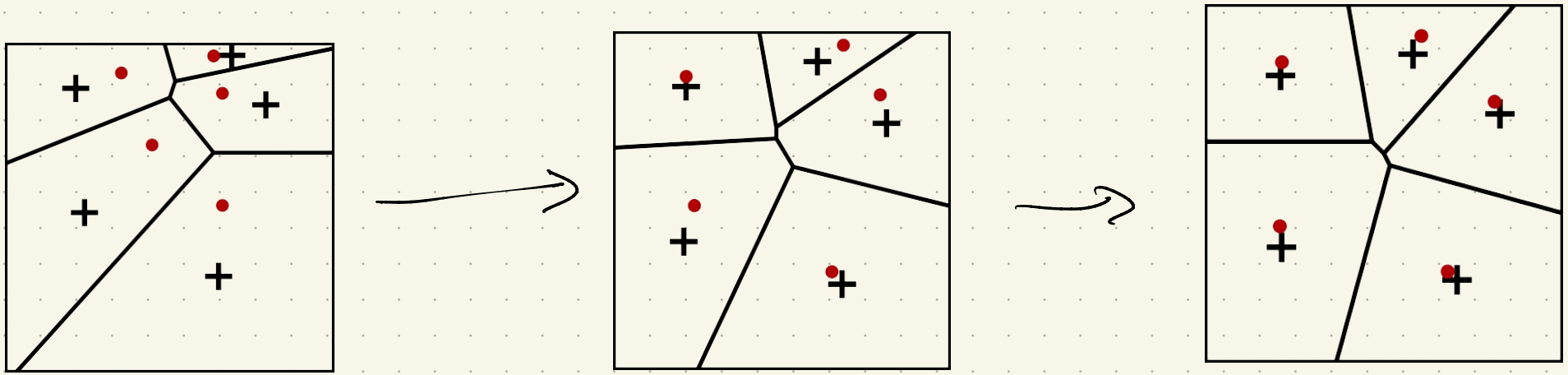
After initializing the sites S as $k = 4$ points from the data set, the algorithm assigns data points in X to each site. Then in each round, it updates the site as the average of the assigned points, and re-assigned points (represented by the Voronoi cells). After 4 rounds it has converged to the optimal solution.

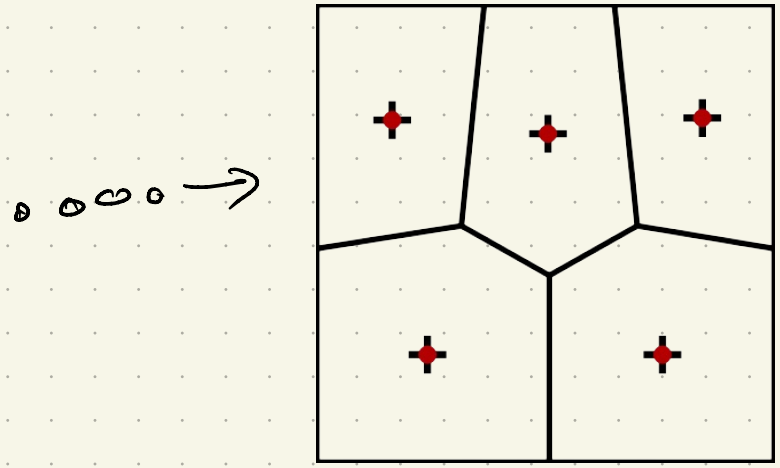


Taken from Mathematical Foundations for Data Analysis" by Jeff Phillips

Note: Can actually do this for all of $\mathbb{R}^d$!

"Centroidal Voronoi tesselations"





(Iteration 15)

But: does this ever converge?

Well, cost is always decreasing:

$$\text{cost}(X, S) = \sum_{x \in S} \|\phi_S(x) - x\|^2$$

$$= \sum_{S_i \in S} \sum_{x \in X_i} \|s_i - x\|^2$$

If things change in a round, choose S_i minimizing cost

↳ no 2 steps have same set of centers!

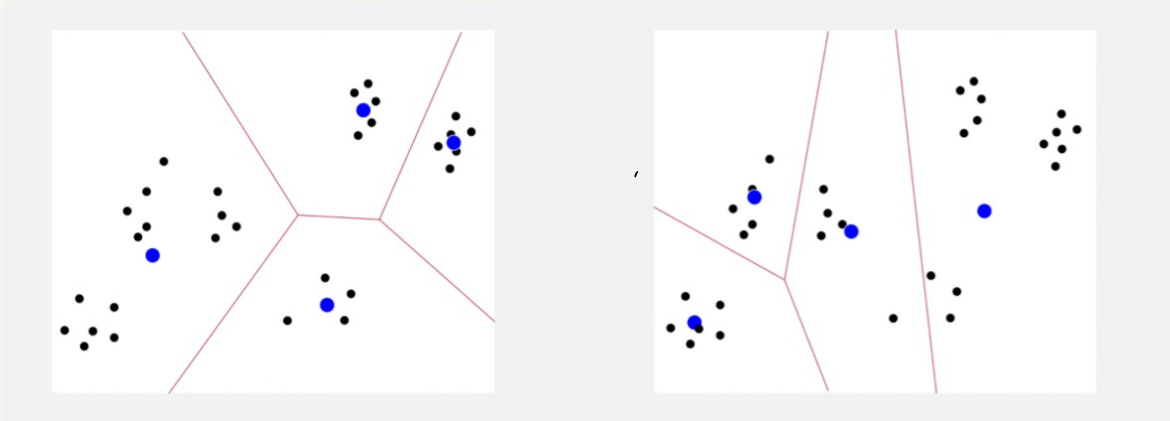
How many ways to make k groups?

So, downside:

- Runtime:
- Also, we know it ends with centers, but are they always the best ones?

Unfortunately, no!

Algorithm can get "stuck" in a local minimum:



One key strategy: initialization

Lloyd's algorithm originally proposed dividing points into k sets aritrarily.

However, not a good idea!

> Central limit theorem
> ⇒ mean of each set is close to mean of overall.

Better:

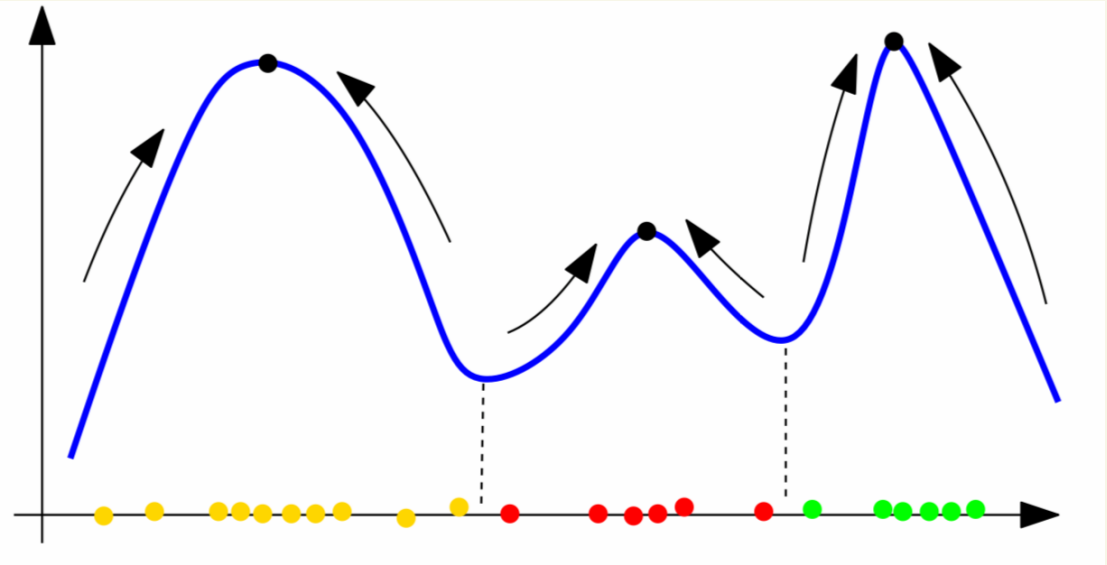
Mode-seeking

Assume data is drawn from some unknown density function.

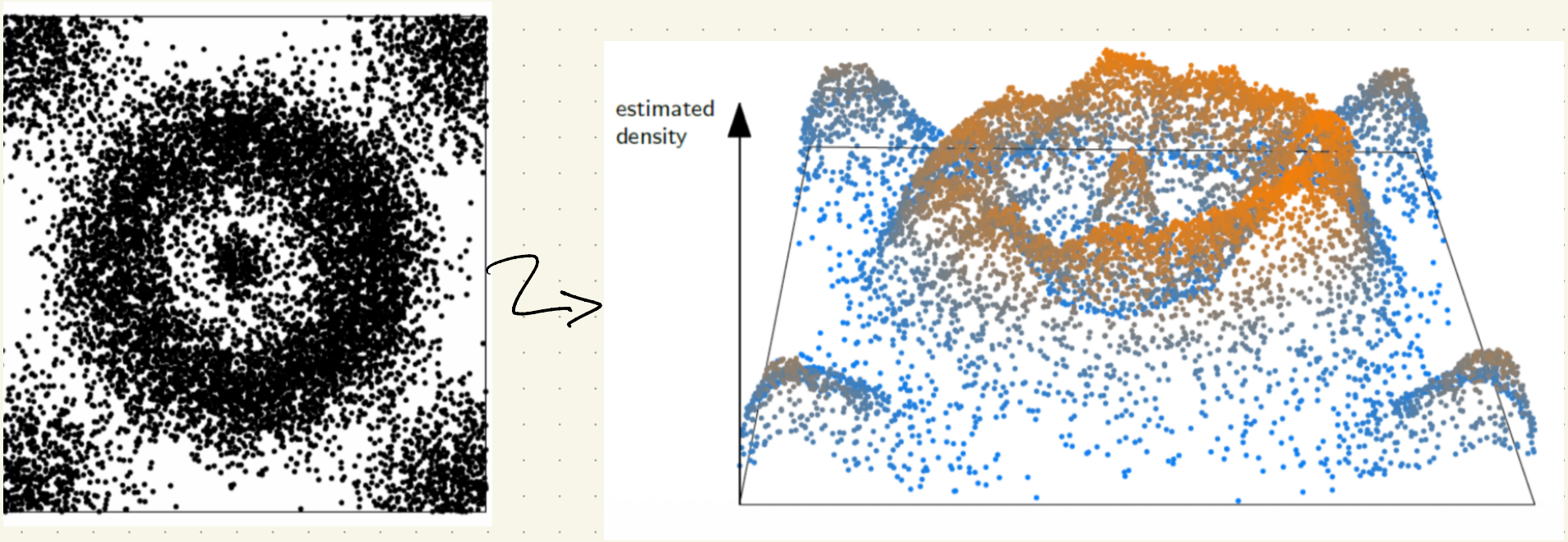
Define clusters according to density!

ie distance to next closest point(s)

In 1-D:



In 2d:



Koontz, Narendra & Fukunaga 1976

Build a neighborhood graph:

Then, they approximate the gradient of this graph:

for each vertex, pick edge to highest value neighbor

More formally: pseudo code

Input: neighborhood graph G with n vertices, n-dimensional vector $\hat{f}$ (density estimator)

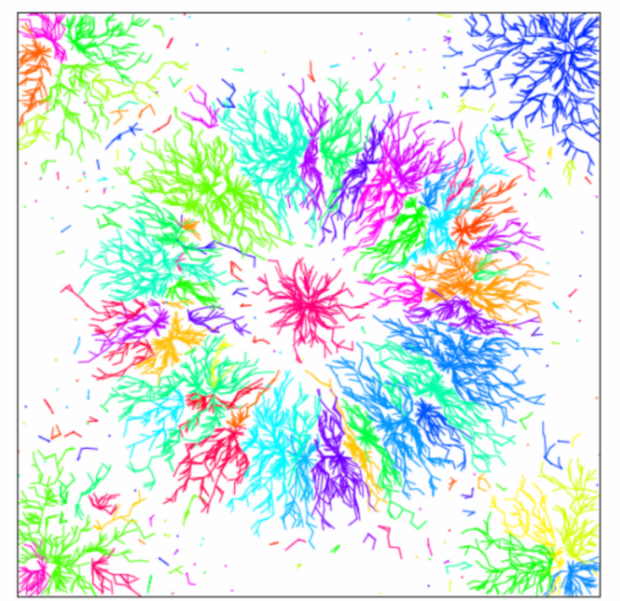
Sort the vertex indices $\{1, 2, \cdots, n\}$ so that $\hat{f}(1) \geq \hat{f}(2) \geq \cdots \geq \hat{f}(n)$;
Initialize a union-find data structure (disjoint-set forest) $\mathcal{U}$ and two vectors g, r of size n;

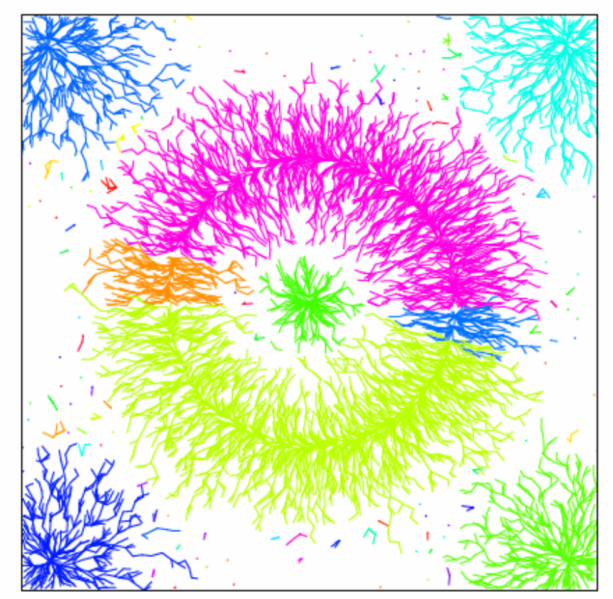
for $i = 1$ to n **do**
    Let $\mathcal{N}$ be the set of neighbors of i in G that have indices lower than i;
    if $\mathcal{N} = \emptyset$ *// vertex i is a peak of $\hat{f}$ within G*
        Create a new entry e in $\mathcal{U}$ and attach vertex i to it;
        $r(e) \leftarrow i$ *// $r(e)$ stores the root vertex associated with the entry e*
    else *// vertex i is not a peak of $\hat{f}$ within G*
        $g(i) \leftarrow \text{argmax}_{j \in \mathcal{N}} \hat{f}(j)$ *// $g(i)$ stores the approximate gradient at vertex i*
        $e_i \leftarrow \mathcal{U}.\texttt{find}(g(i))$;
        Attach vertex i to the entry e_i;

graph-based hill-climbing (1976)

(Uses union-find data structure)

Issue: Very sensitive to noise





There are as many clusters as there are local maxima of density.

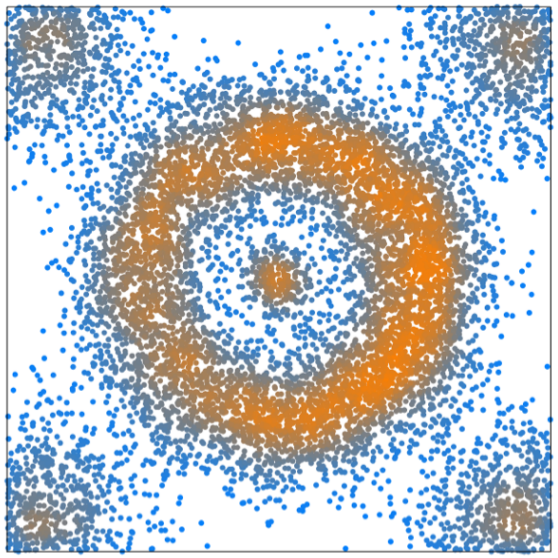
Use persistence: ToMATo

Chazal, Guibas, Oudot & Skraba

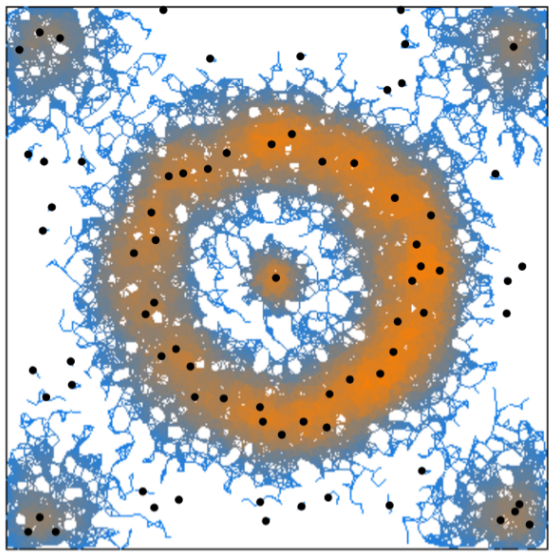
Build graph, & gives edges weights

$f((u,v)) = \min\{\text{density}(u), \text{density}(v)\}$

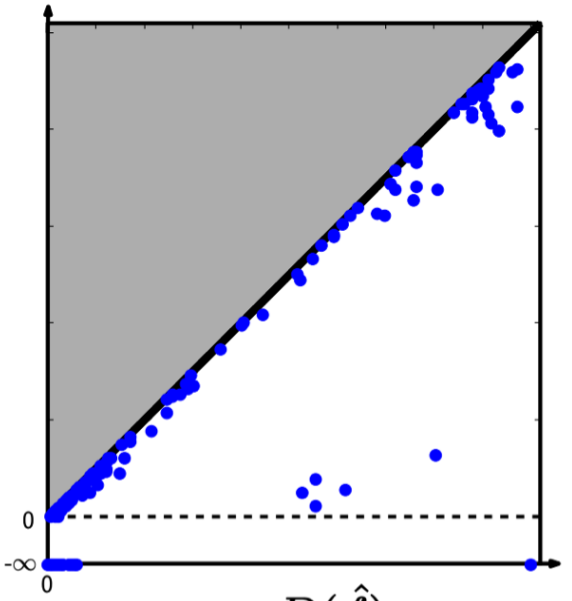
Superlevel set persistence:



$\hat{f}$



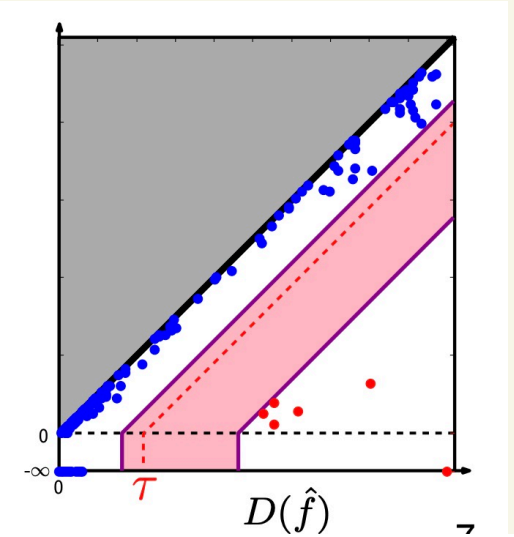
$\hat{f}$ extended to G

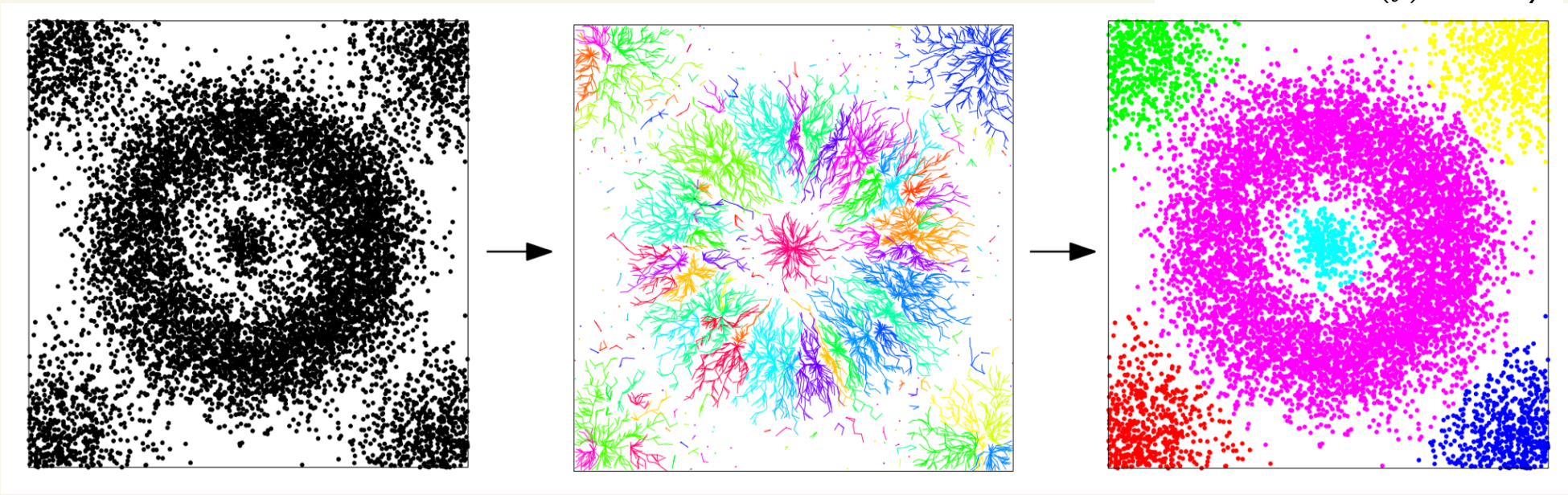


$D(\hat{f})$

Goal: infer a threshold τ ↗

Then, adapt hill-climbing to merge clusters with low persistence into parent's cluster:





Pseudocode:

Input: simple graph G with n vertices, n-dimensional vector $\hat{f}$, real parameter $\tau \geq 0$.

Sort the vertex indices $\{1, 2, \cdots, n\}$ so that $\hat{f}(1) \geq \hat{f}(2) \geq \cdots \geq \hat{f}(n)$;
Initialize a union-find data structure $\mathcal{U}$ and two vectors g, r of size n;

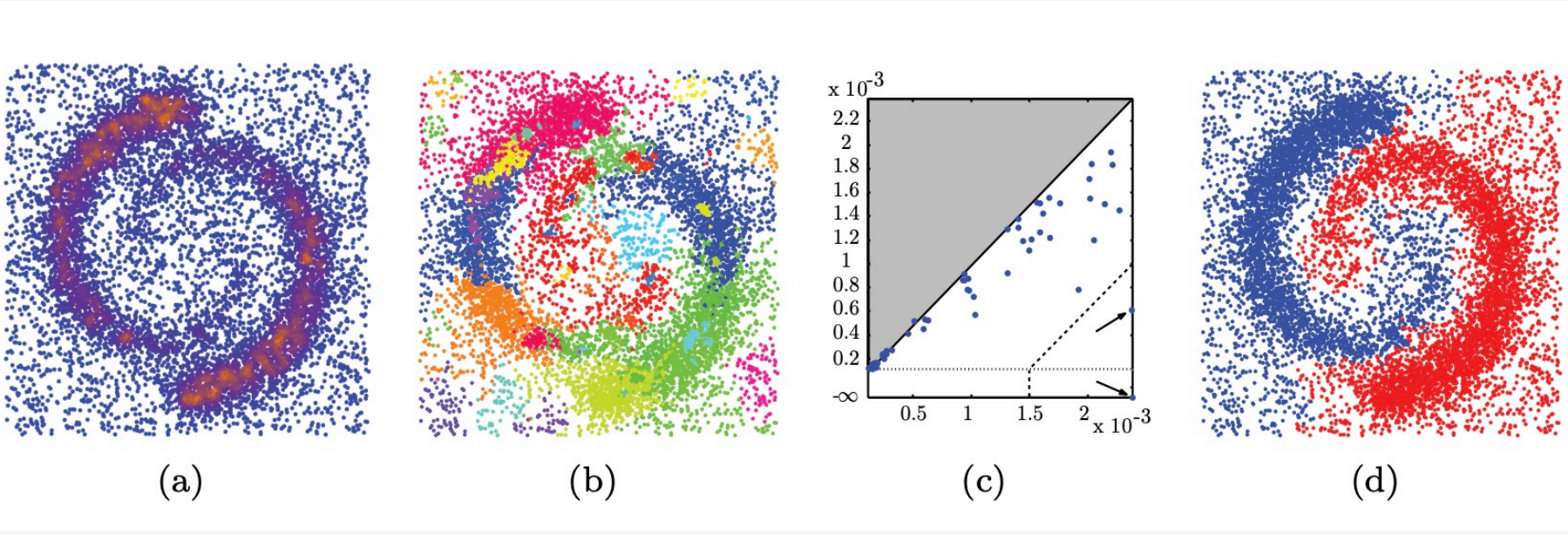
for $i = 1$ to n **do**
  Let $\mathcal{N}$ be the set of neighbors of i in G that have indices lower than i;
  if $\mathcal{N} = \emptyset$ *// vertex i is a peak of $\hat{f}$ within G*
    Create a new entry e in $\mathcal{U}$ and attach vertex i to it;
    $r(e) \leftarrow i$ *// $r(e)$ stores the root vertex associated with the entry e*
  else *// vertex i is not a peak of $\hat{f}$ within G*
    $g(i) \leftarrow \operatorname{argmax}_{j \in \mathcal{N}} \hat{f}(j)$ *// $g(i)$ stores the approximate gradient at vertex i*
    $e_i \leftarrow \mathcal{U}.\texttt{find}(g(i))$;
    Attach vertex i to the entry e_i;
    for $j \in \mathcal{N}$ **do**
      $e \leftarrow \mathcal{U}.\texttt{find}(j)$;
      if $e \neq e_i$ and $\min\{\hat{f}(r(e)),\ \hat{f}(r(e_i))\} < \hat{f}(i) + \tau$
        $\mathcal{U}.\texttt{union}(e,\ e_i)$;
        $r(e \cup e_i) \leftarrow \operatorname{argmax}_{\{r(e),\ r(e_i)\}} \hat{f}$;
        $e_i \leftarrow e \cup e_i$;

graph-based hill-climbing (1976)

cluster merges with persistence (2013)

Output: the collection of entries e of $\mathcal{U}$ such that $\hat{f}(r(e)) \geq \tau$.

Results: impressive!



Stability for persistence ⇒
Some guarantee of optimality
(but relies on good choice of τ
& "good enough" sampling)