Hybridized Threshold Clustering for Massive Datasets

Michael J. Higgins

Kansas State University

Sept. 30, 2017

Collaborators

- Work with Jianmei Luo

Collaborators

- Work with Jianmei Luo
- Other collaborators:
  - Jasjeet Sekhon
  - Fredrik Sävje
  - Zhihan Zhang
  - K-State Comp Sci
    - Bill Hsu
    - ChandraVyas Annakula
    - Aruna Sai Kannamareddy

Statistical clustering

- A frequent goal in statistical analysis is to divide partition the data into groups—*clusters*—so that units with each groups have similar attributes—values of their *covariates*.

Statistical clustering

- A frequent goal in statistical analysis is to divide partition the data into groups—*clusters*—so that units with each groups have similar attributes—values of their *covariates*.
- Idea: Units within the same group often behave similarly, and units in different groups have different behavior.
- Groups often have some meaning associated with them.

Statistical clustering examples

- Million Song Dataset:
  - Data point: A song.
  - Covariates: Tempo, Max Volume, Min Volume, Length, etc.
  - Goal: Identify the genre of the song using only covariates.
  - Clusters: Sets of songs of the same genre.

Statistical clustering examples

- Million Song Dataset:
  - Data point: A song.
  - Covariates: Tempo, Max Volume, Min Volume, Length, etc.
  - Goal: Identify the genre of the song using only covariates.
  - Clusters: Sets of songs of the same genre.
- Stock Market Trades:

  Automatic trading: Algorithms make stock trades automatically based on instantaneous market changes—no human input.
  - Data point: Trade.
  - Covariates: Time of trade, volume of trade, which stock traded, etc.
  - Goal: Determine which trades were performed automatically.
  - Clusters: Groups of trades that were all either made automatically or "by hand."

Why massive data is hard

- Many statistical clustering algorithms with good properties currently exist.
- Most are too computationally expensive to be used in *massive data* settings—when the number of units N is very large.

Why massive data is hard

- Many statistical clustering algorithms with good properties currently exist.
- Most are too computationally expensive to be used in *massive data* settings—when the number of units N is very large.
- Examples:
  - NASDAQ: 10 million trades a day
  - Twitter: 500 million tweets a day
  - Google: 3.5 billion searches a day

Why massive data is hard

- At their core, most clustering algorithms are graph partitioning problems:
- Given a graph $G(V, E)$, find an optimal partition of V (w.r.t. some objective) into blocks that are connected in G.

Why massive data is hard

- At their core, most clustering algorithms are graph partitioning problems:
- Given a graph $G(V, E)$, find an optimal partition of V (w.r.t. some objective) into blocks that are connected in G.
- Optimal graph partitioning problems tend to be NP–hard.

  Intuitive definition: $NP =$ **N**o **P**olynomial-time algorithm.
  Time to solve problem increases exponentially with N. Hence, no chance of solving for massive data.
- Even good approximate or heuristic algorithms may be prohibitively computationally expensive for massive data.

Hybridized threshold clustering

Threshold clustering with a bottleneck objective [Higgins et. al.].

- *Threshold clustering*—each block of the partition contains at least k vertices for some prespecified threshold k.
- *Bottleneck objective*—Minimize the maximum within-block dissimilarity (MWBD). Forces all units in a block to be "similar" to each other.

Hybridized threshold clustering

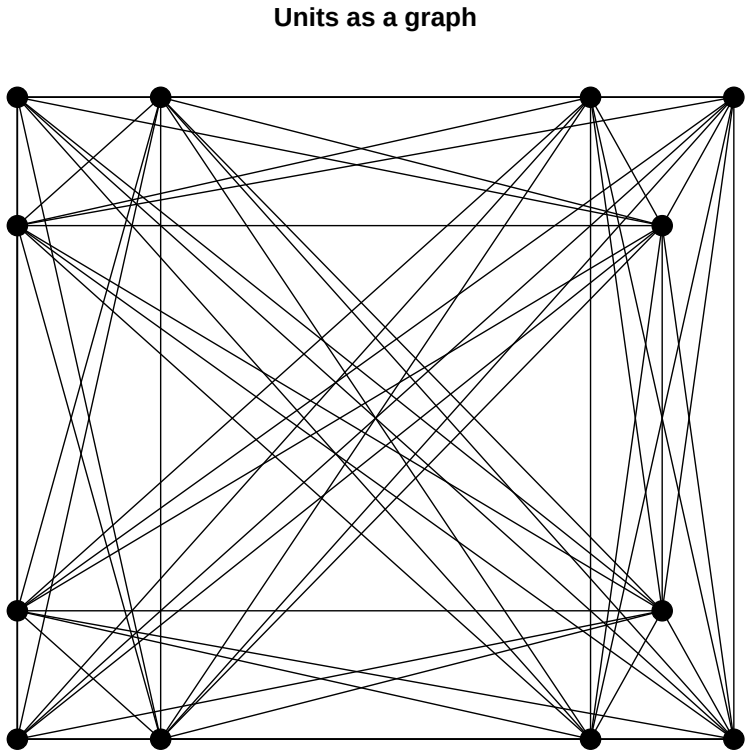
Threshold clustering with a bottleneck objective [Higgins et. al.].

- *Threshold clustering*—each block of the partition contains at least k vertices for some prespecified threshold k.
- *Bottleneck objective*—Minimize the maximum within-block dissimilarity (MWBD). Forces all units in a block to be "similar" to each other.
- Algorithm finds solution within a factor of 4 of optimal in $O(kN)$ time, outside of the construction of a $(k-1)$–nearest-neighbors graph.
- Efficient enough for massive datasets, especially when k is small.

Algorithm step-by-step: Construct a graph

- Begin with a complete graph.
- N units $\rightarrow$ N vertices.
- Dissimilarity measure: Small when values of covariates are similar.
- Can be computed between every pair of units (though may not be reqired)

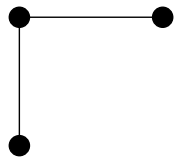
$k = 2$

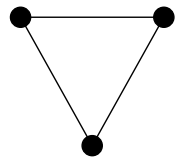


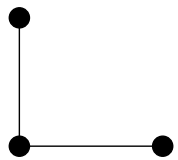
Algorithm step-by-step: Find nearest neighbor graph

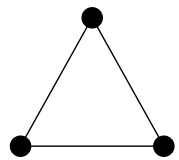
- Construct a $(k-1)$–nearest-neighbors graph NNG_k.

$k = 2$



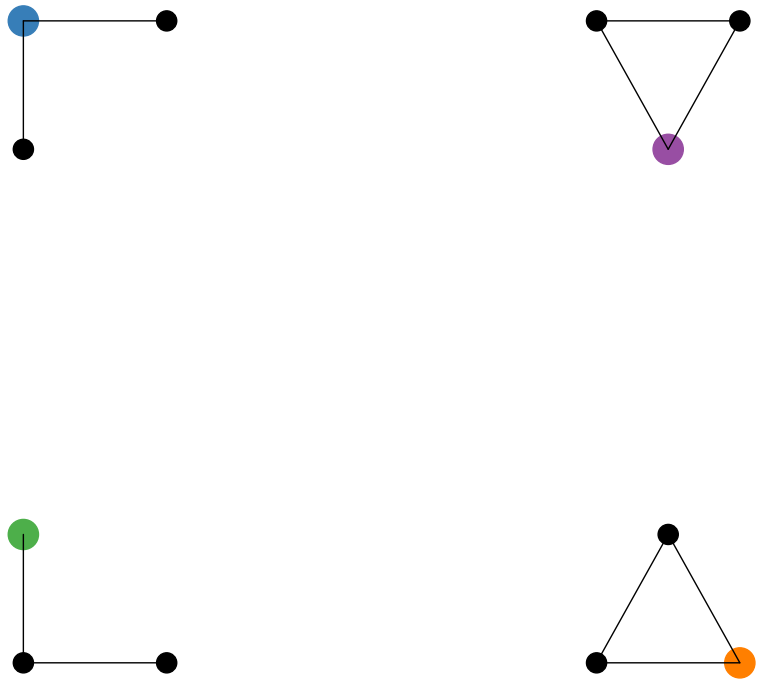






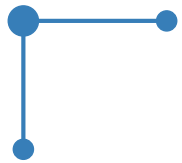
Algorithm step-by-step: Find block seeds

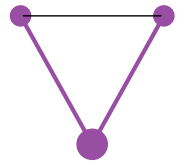
- Find a set of independent vertices in NNG_k^2.

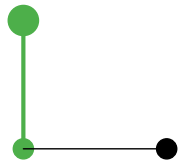


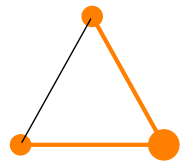
Algorithm step-by-step: Grow from block seeds

- Form blocks comprised of a block seed and any vertices adjacent to the seed.



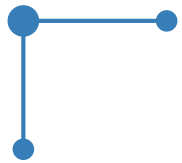


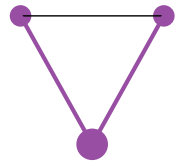


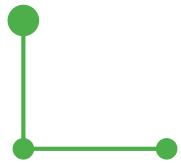


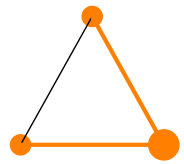
Algorithm step-by-step: Assign all unassigned vertices

- For each unassigned vertex, find its closest seed in the nearest neighbor graph. Add that vertex to the seed's corresponding block.







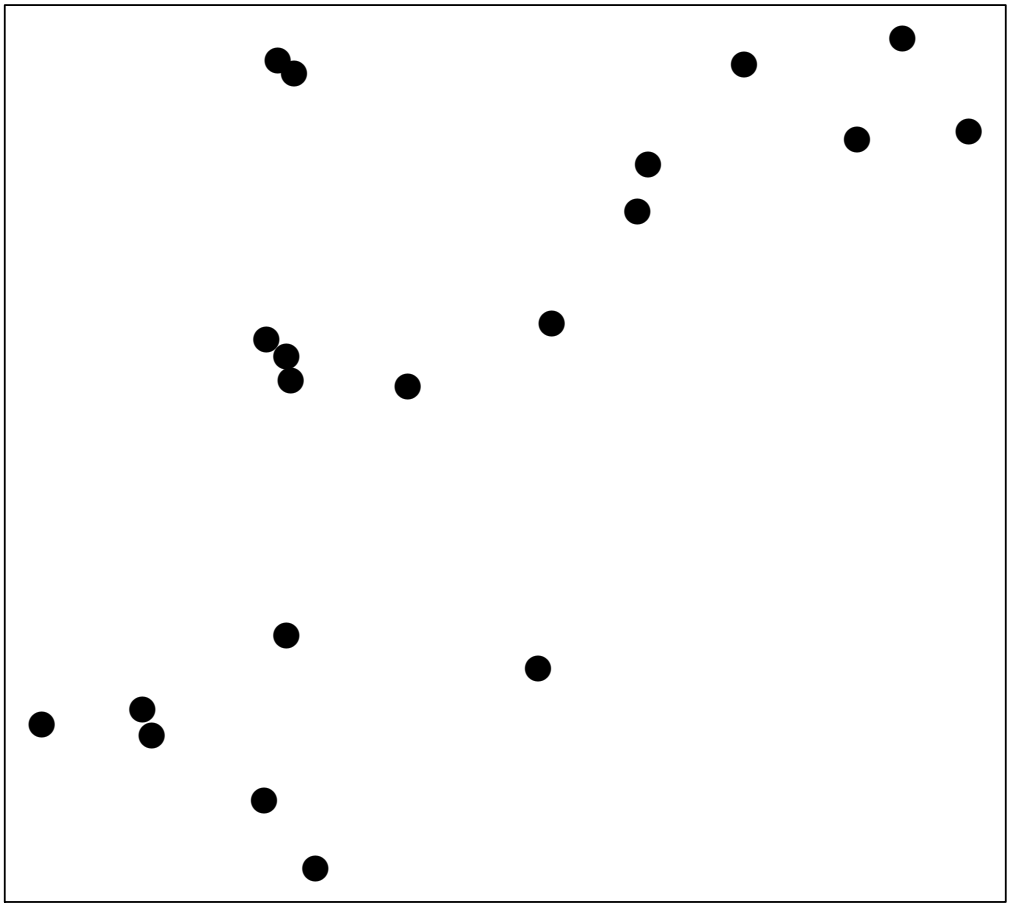


Hybridized threshold clustering

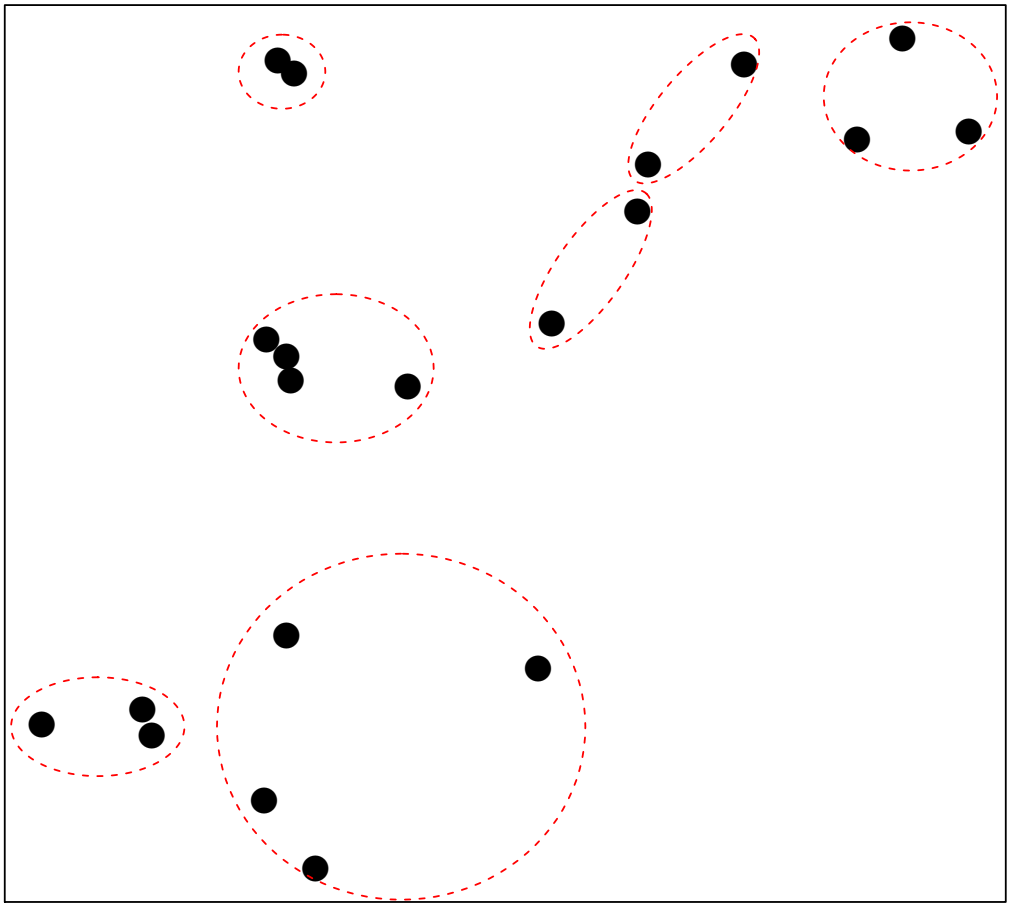
Our Solution: For massive data problems, use threshold clustering as a preprocessing step to facilitate more sophisticated clustering algorithms.

- Apply threshold clustering and obtain c groups, $c << N$.
- Condense each of the c clusters into a single point p_j, $j \in \{1, \ldots, c\}$. E.g.: Average covariate values within each group.
- Obtain a clustering on these c points using desired clustering algorithm.
- Obtain clustering on all units N by assigning each unit i to the cluster associated with its condensed point p_j.

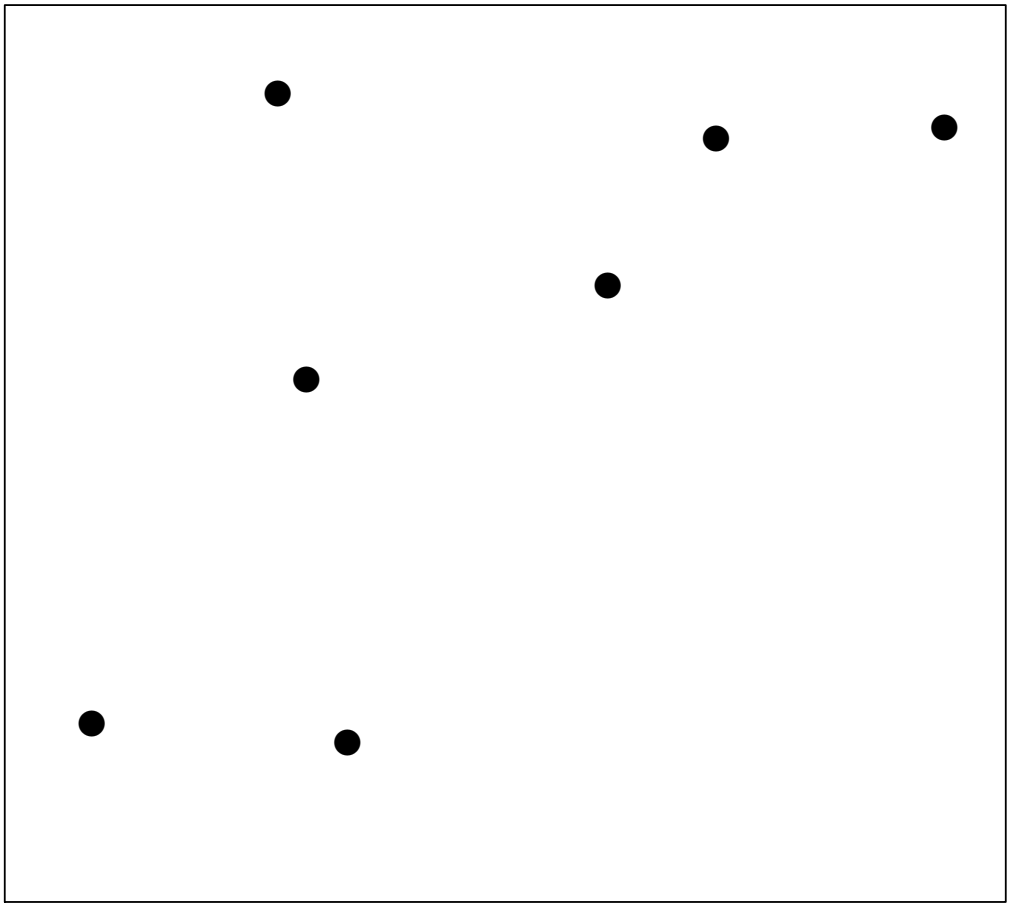
Hybridized clustering: Initial points



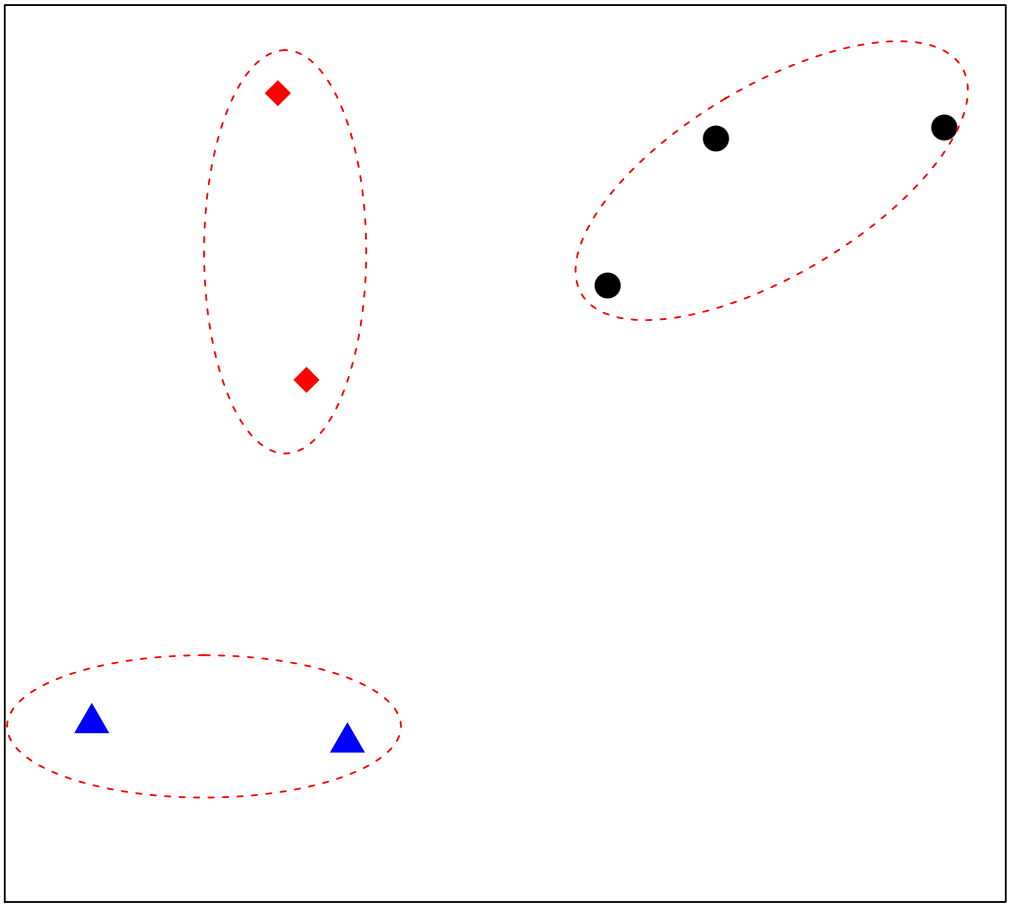
Hybridized clustering: Threshold clustering



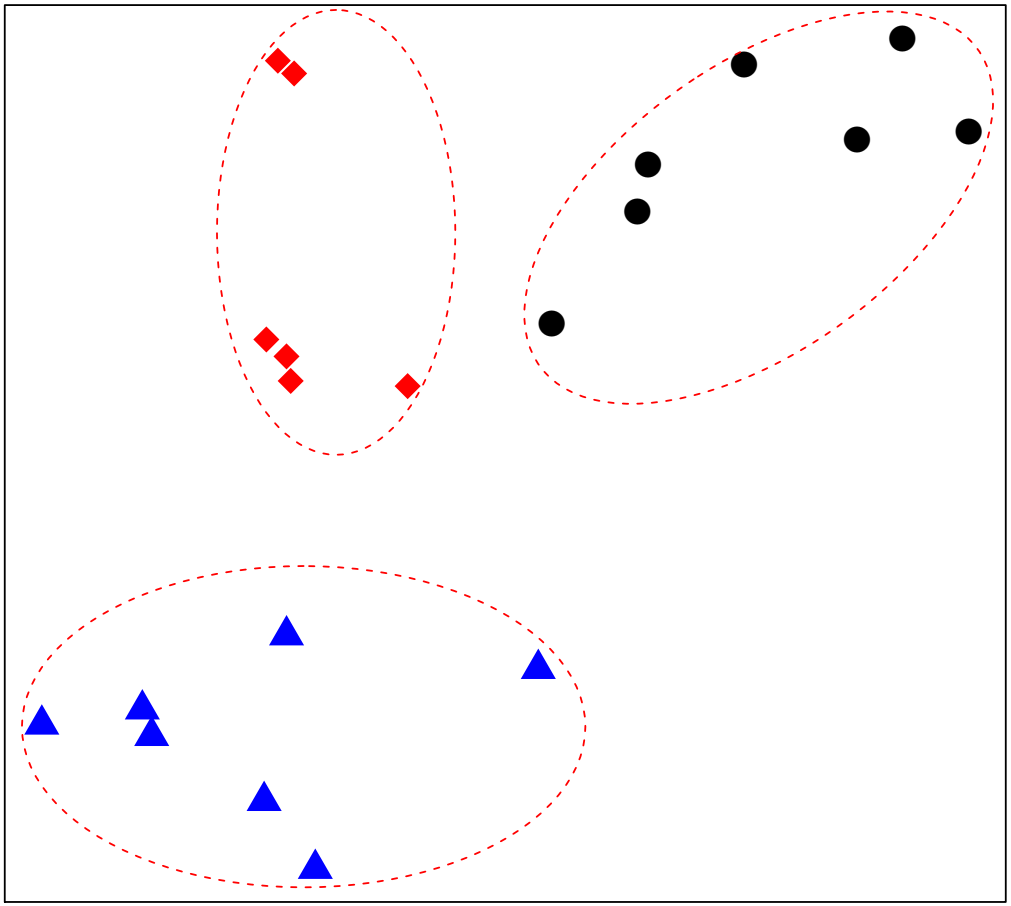
Hybridized clustering: Condense points



Hybridized clustering: 2nd Clustering algorithm



Hybridized clustering: Back out clusters for all points



Hybridized threshold clustering

Hybridized threshold clustering: Two versions.

- Version I: Run threshold clustering once with large k before applying second clustering algorithm
  - Pro: Tends to produce a better clustering of units.
  - Con: May be prohibitively expensive to run when k is large.

Hybridized threshold clustering

Hybridized threshold clustering: Two versions.

- Version I: Run threshold clustering once with large k before applying second clustering algorithm
  - Pro: Tends to produce a better clustering of units.
  - Con: May be prohibitively expensive to run when k is large.
- Version II: Iteratively run threshold clustering with small k until data is sufficiently reduced.
  - Pro: Incredibly efficient at finding clusters.
  - Con: Performance slightly worse than large k method. Becomes worse as iterations increase.

Simulation study

To assess the performance of hybridized threshold clustering:

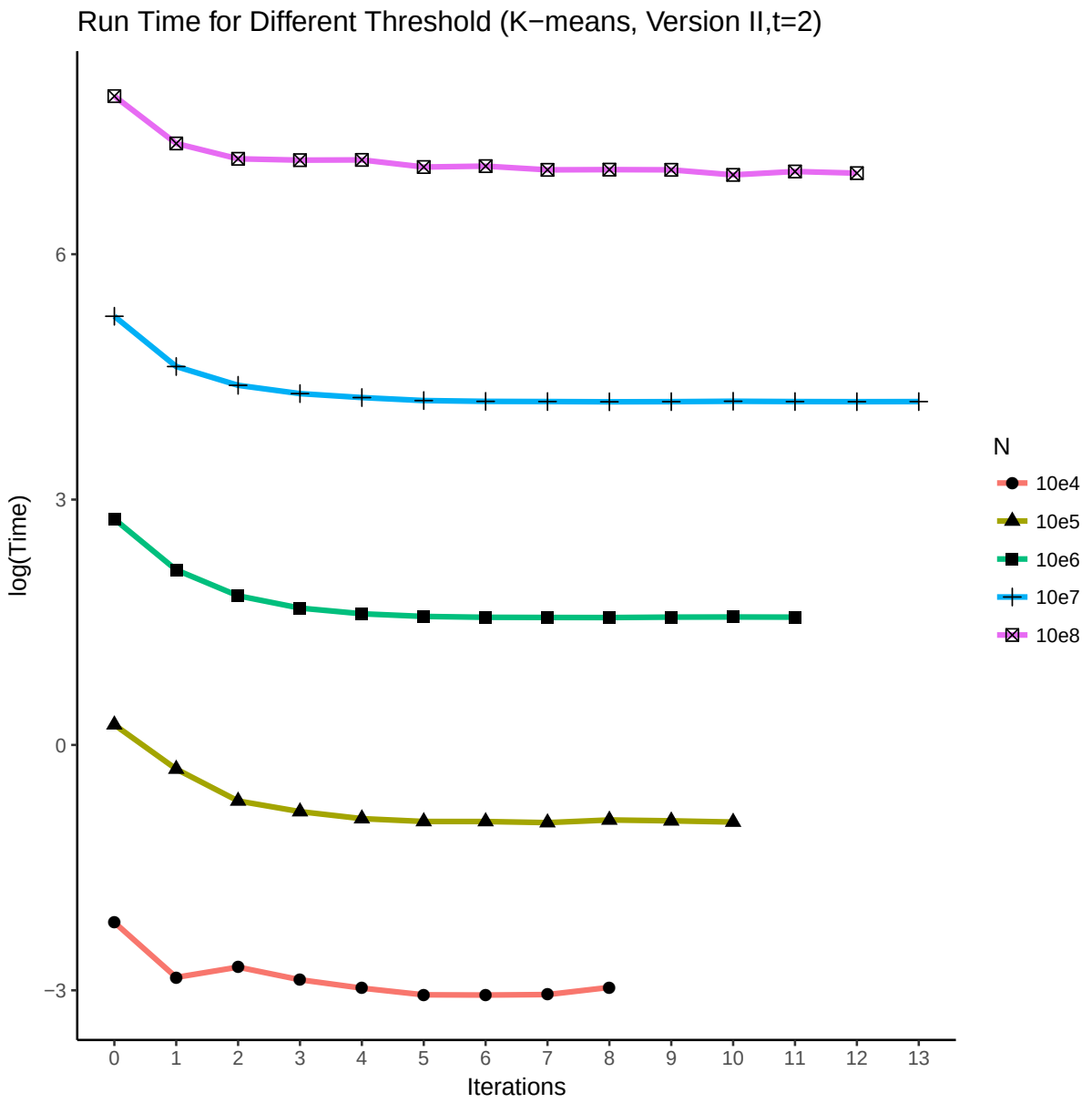
- Generate many data points
- Each data point is generated at random from one of three bivariate normal distributions.
- Each bivariate normal has a different mean and variance.

Simulation study

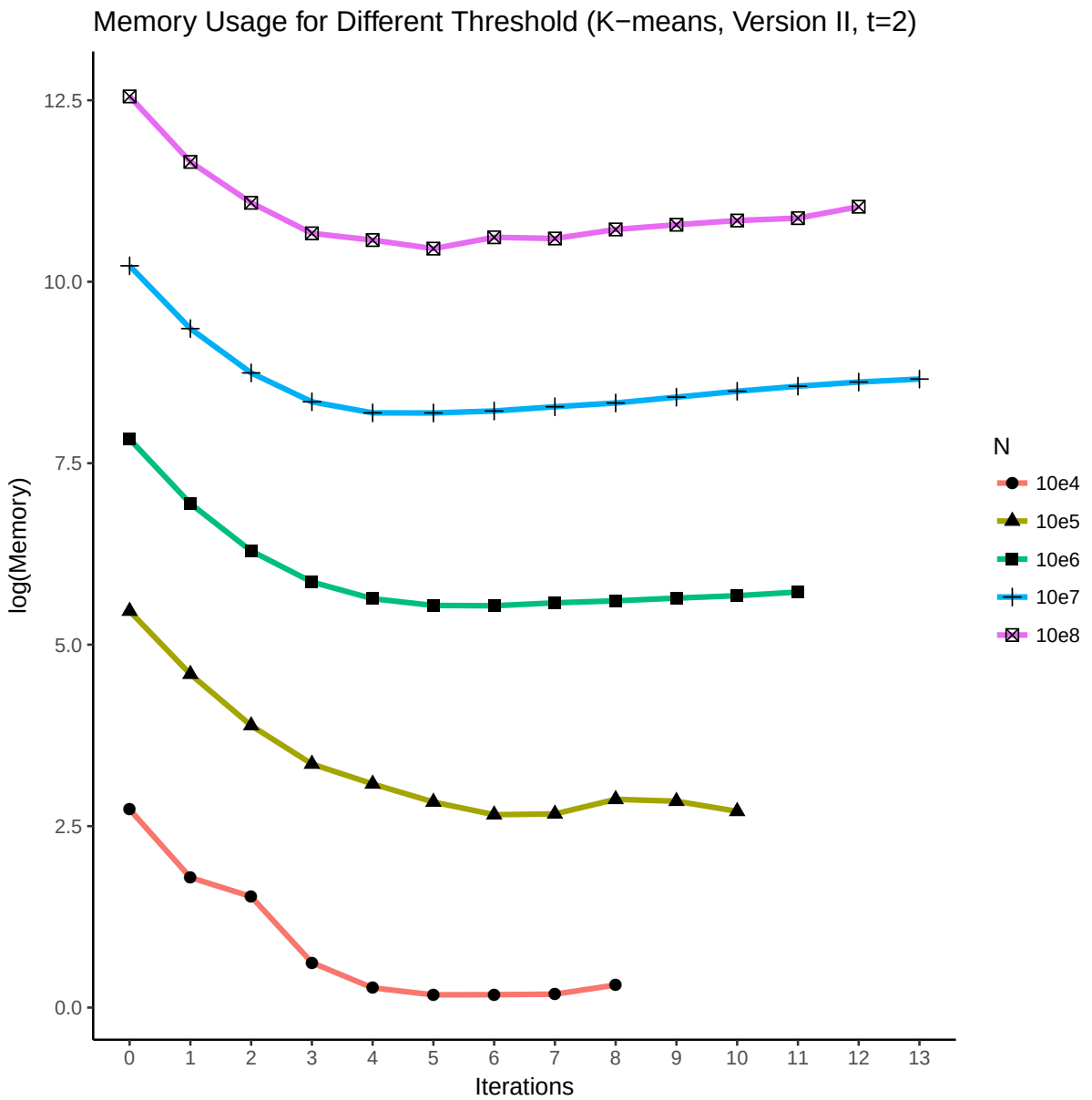
To assess the performance of hybridized threshold clustering:

- Generate many data points
- Each data point is generated at random from one of three bivariate normal distributions.
- Each bivariate normal has a different mean and variance.
- Assess performance through:
  - Run time.
  - Memory use.
  - Prediction accuracy.

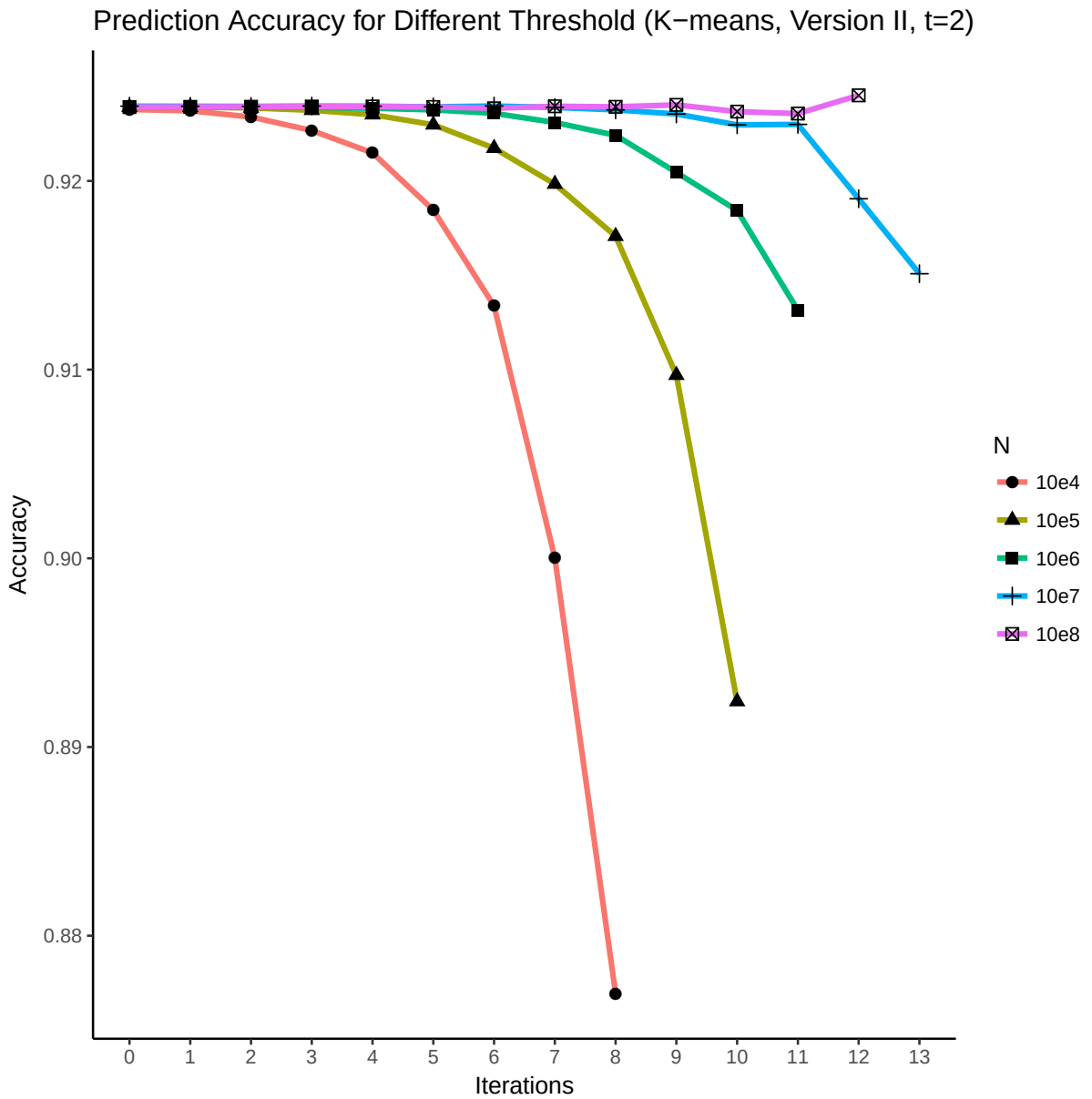
Hybridized version II with K-means



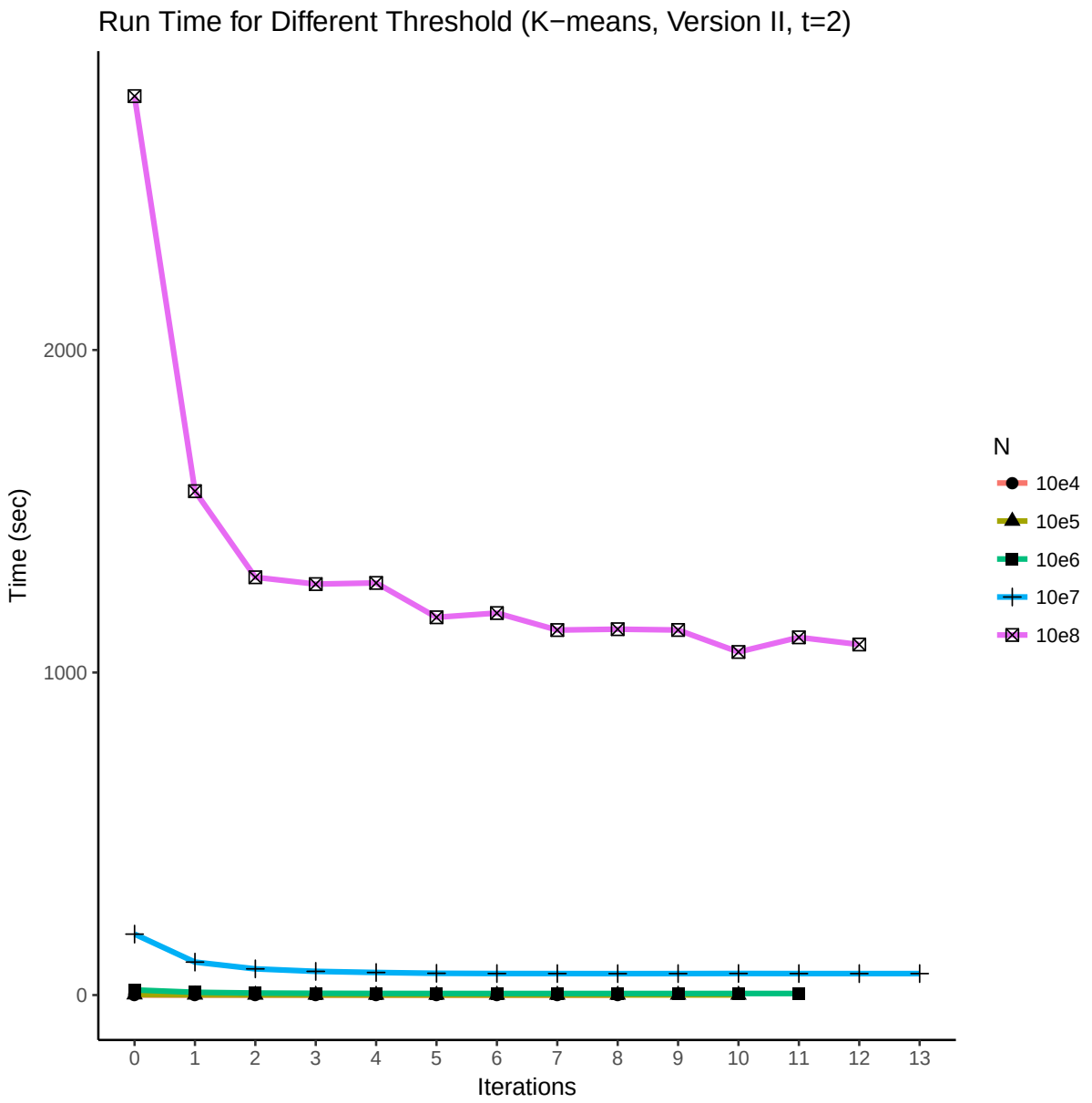
Hybridized version II with K-means



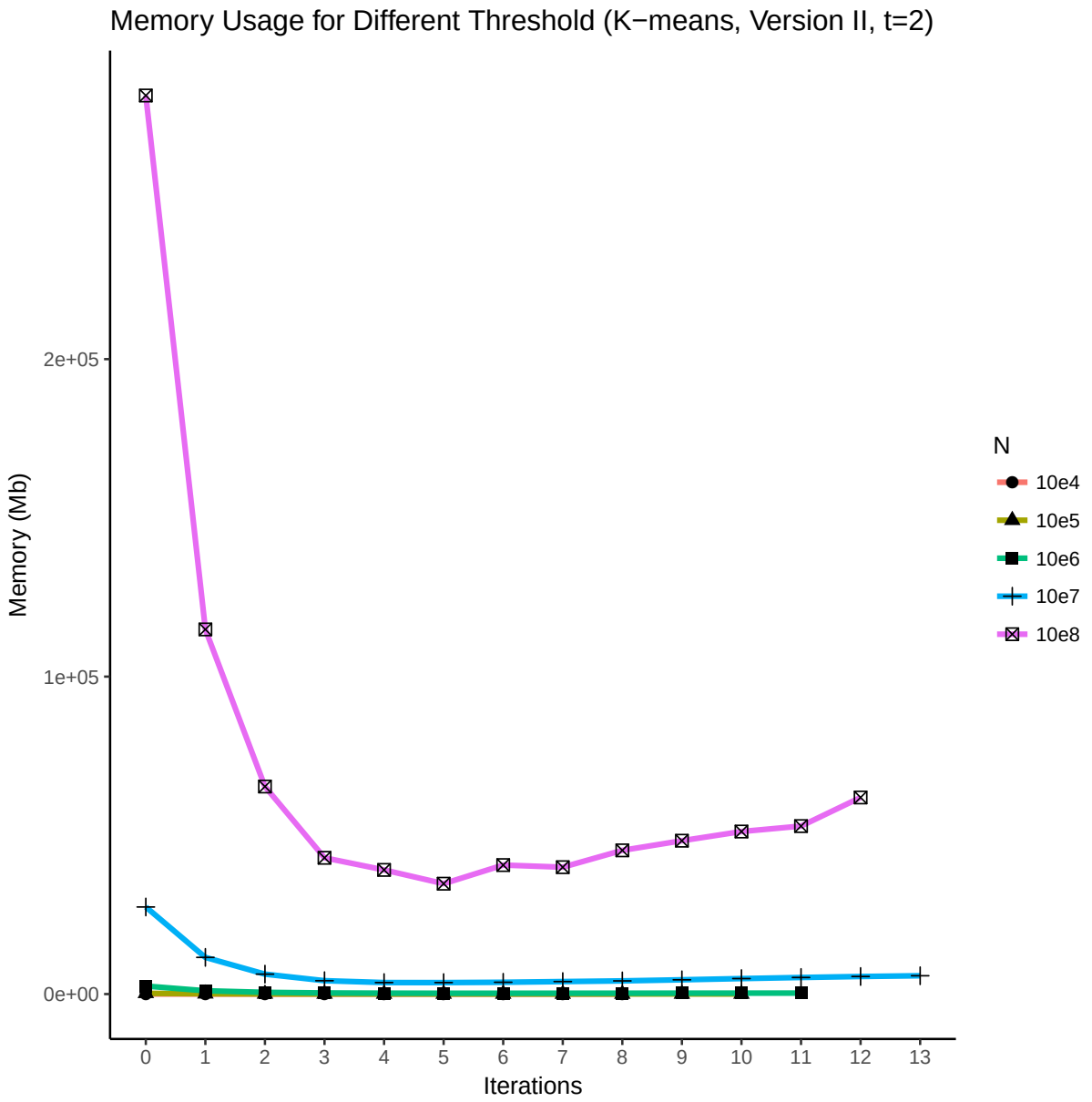
Hybridized version II with K-means



Hybridized version II with K-means

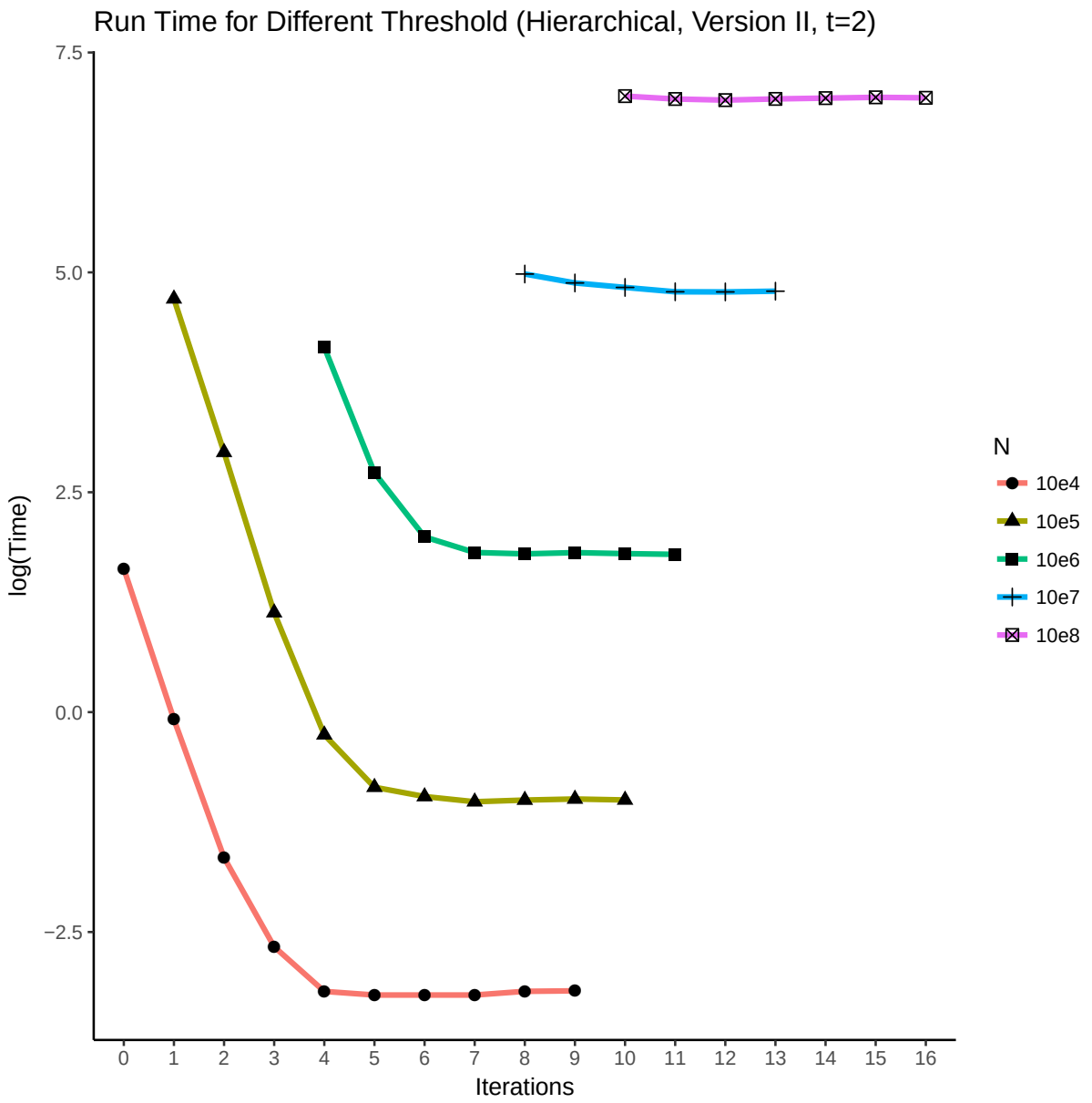


Hybridized version II with K-means

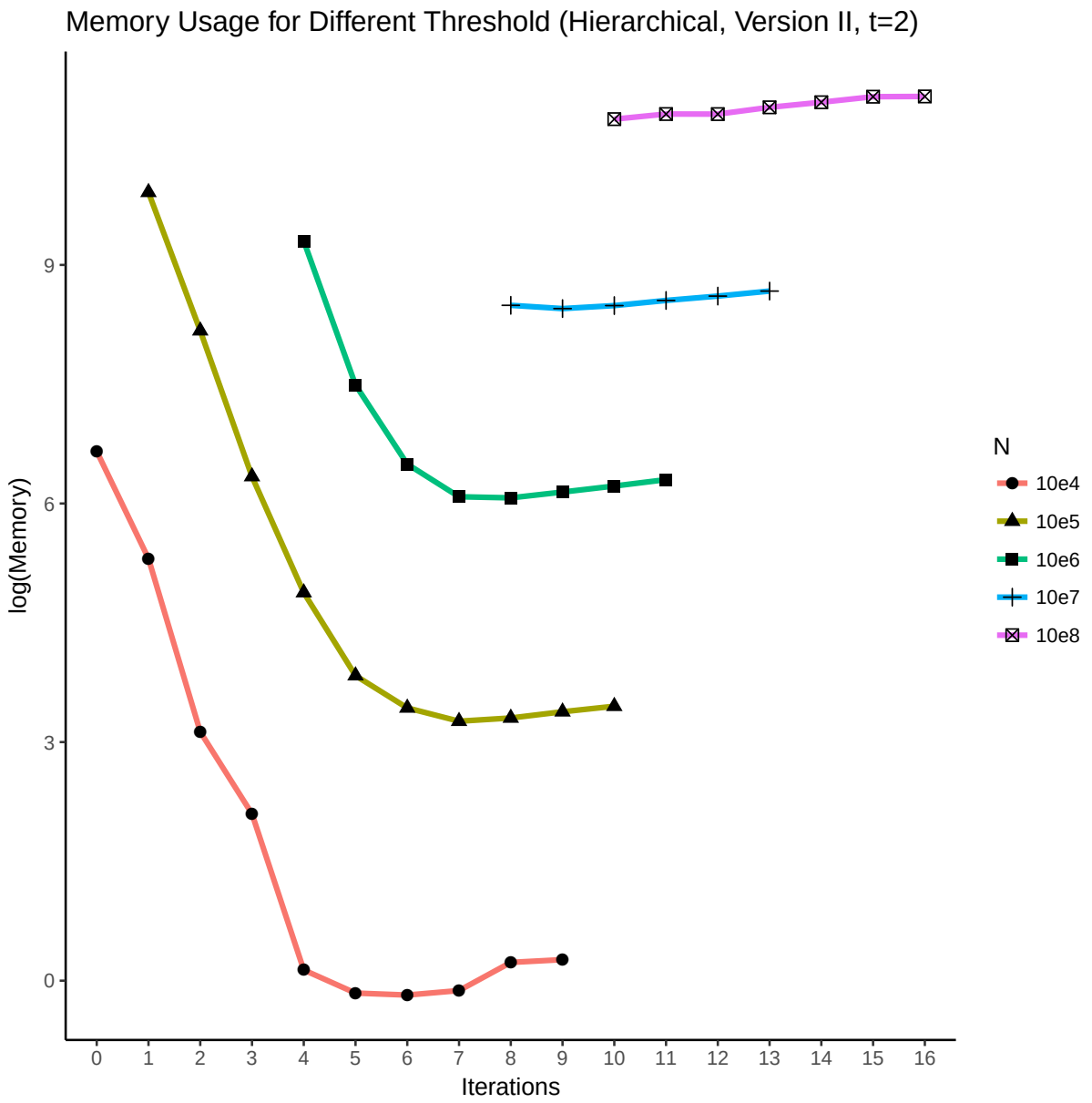


Thank you.

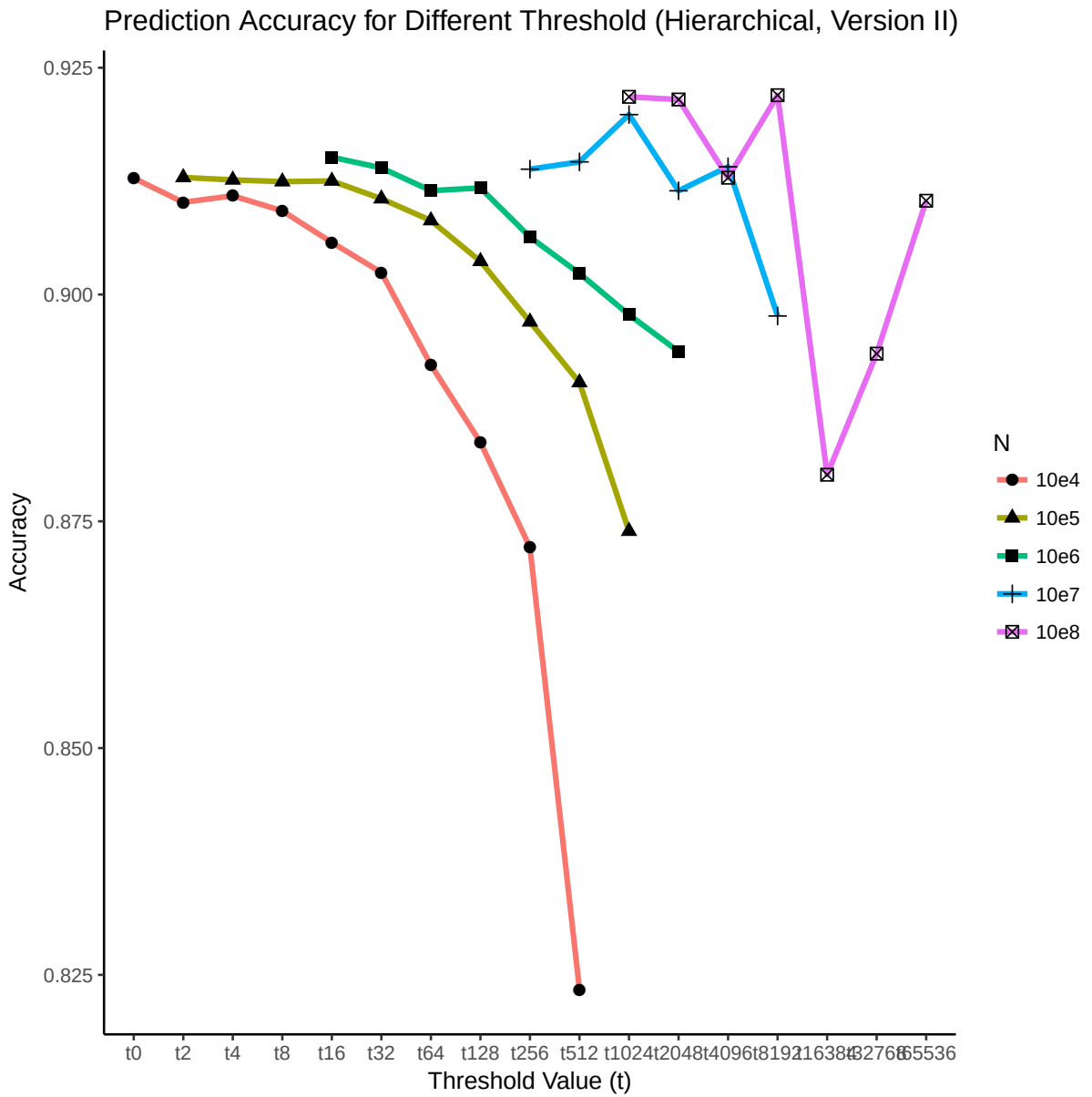
Hybridized version II with K-means



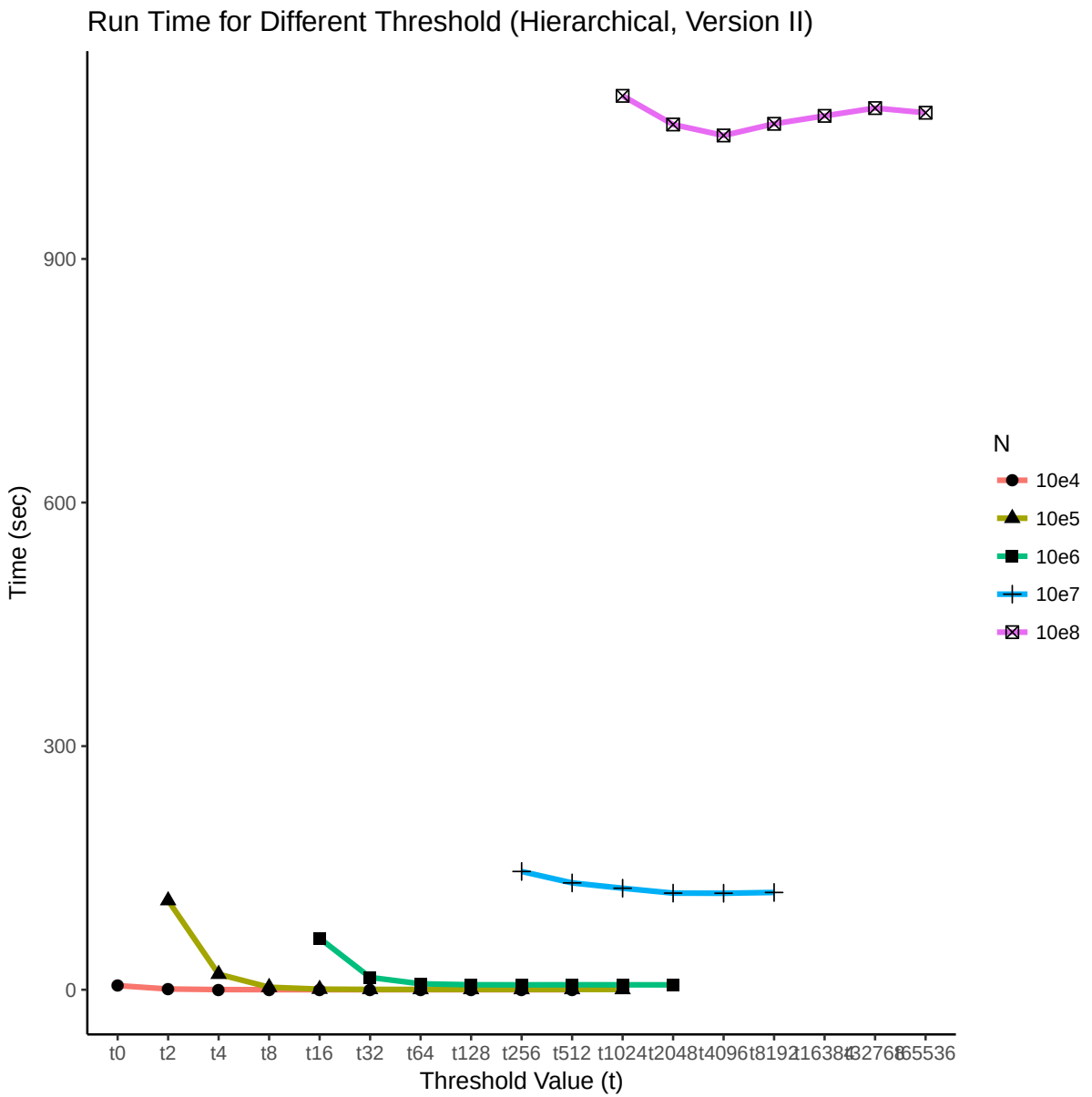
Hybridized version II with K-means



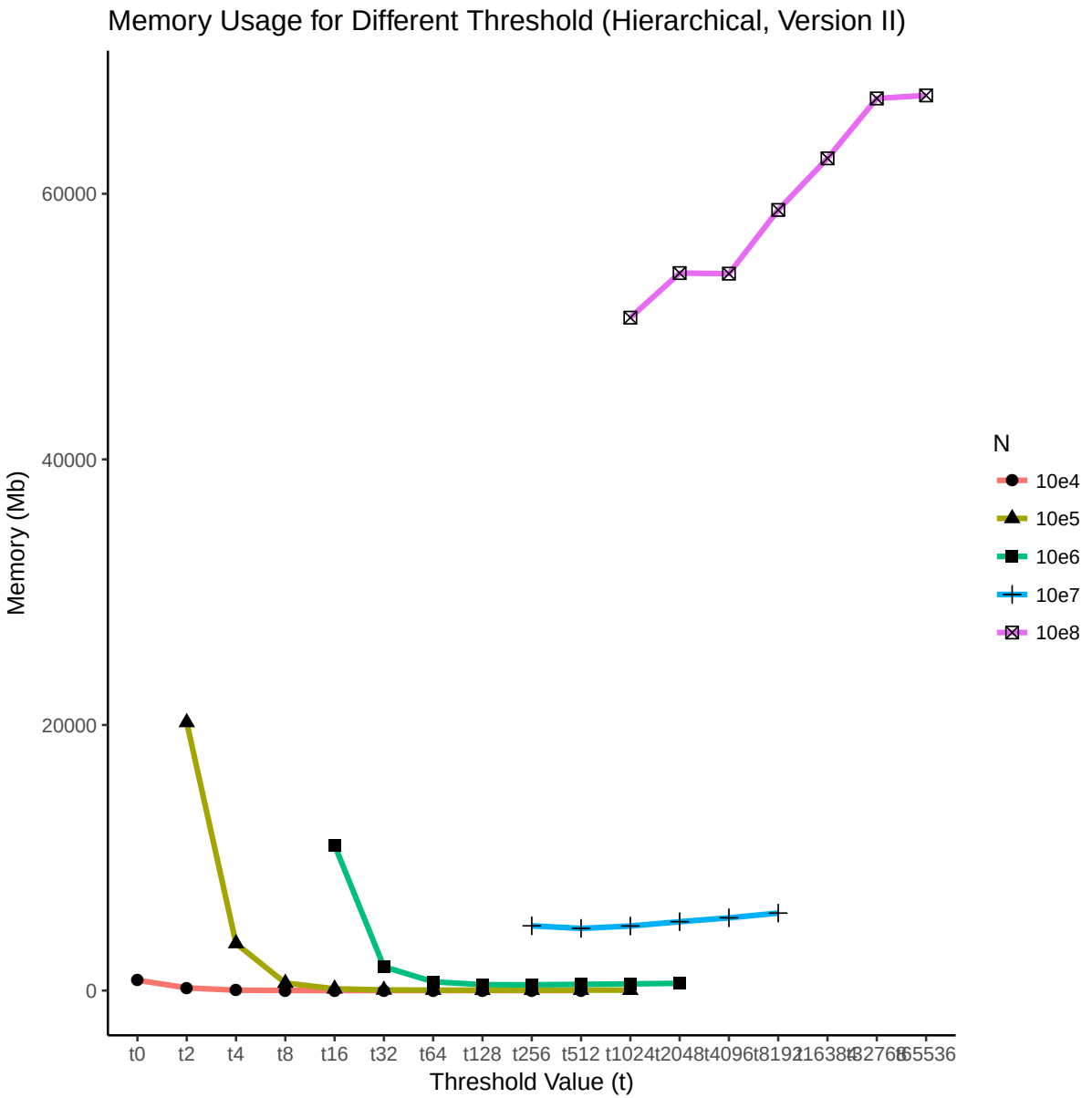
Hybridized version II with K-means



Hybridized version II with K-means



Hybridized version II with K-means



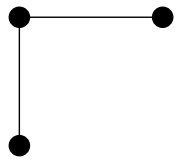
Approximate algorithm outline:

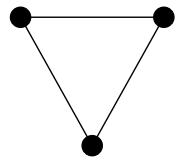
- Construct a $(k-1)$–nearest neighbor subgraph.
- Select block seeds that are "just far enough apart."
- Grow from these block centers to obtain an approximately optimal blocking.
- Approach extends from Hochbaum and Shmoys [1986].

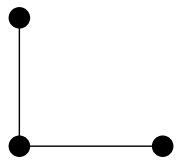
Algorithm step-by-step: Find nearest neighbor graph

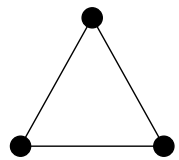
- Construct a $(k-1)$–nearest-neighbors graph.
- For observational study: Use directed nearest-neighbors digraph.
- Can show that edge costs are, at most, λ.

$k = 2$



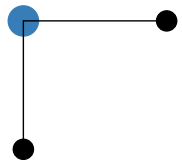


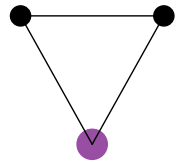


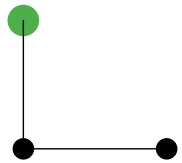


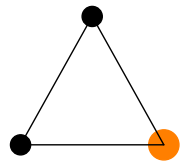
Algorithm step-by-step: Find block centers

- Find a set of vertices—*block seeds*—such that:
  - There is no path of two edges or less connecting any of the vertices in the set.
  - For any vertex not in the set, there is a path of two edges or less that connects that vertex to one in the set.
- Any set works, but some choices of seeds are better.
- Takes $O(kn)$ time.



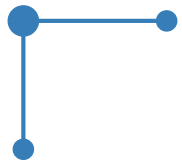


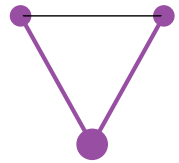


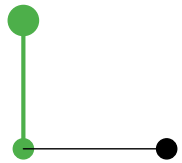


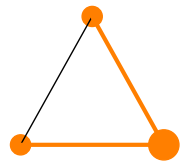
Algorithm step-by-step: Grow from block centers

- Form blocks comprised of a block seed and any vertices adjacent to the seed.
- The way we choose seeds (no path of two edges connects two seeds), these blocks will not overlap.
- By nearest neighbors, these blocks contain at least k units.
- Takes $O(n)$ time.



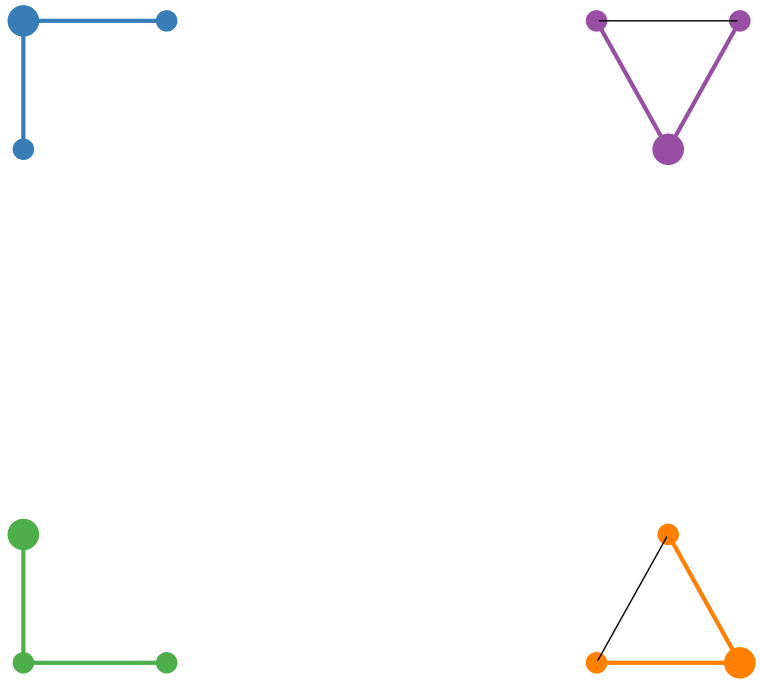






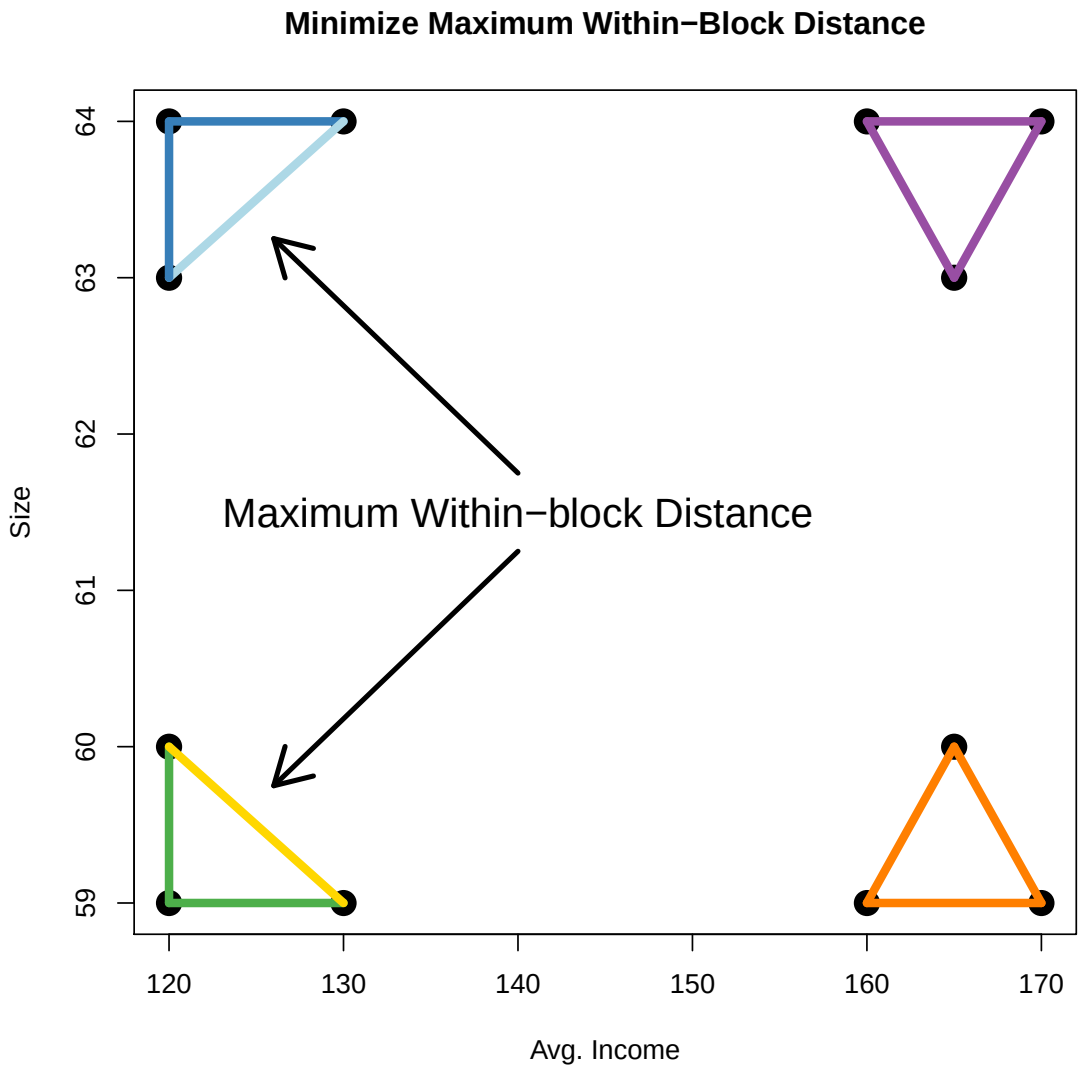
Algorithm step-by-step: Assign all unassigned vertices

- For each unassigned vertex, find its closest seed in the nearest neighbor graph. Add that vertex to the seed's corresponding block.
- We choose seeds so that unassigned vertices are at most a path of two edges away from a block seed.
- Takes $O(n)$ time.
- Since steps are sequential, total runtime is $O(kn)$ outside of nearest neighbor graph construction.



A simple example:

Threshold $k = 2$. Dissimilarity = Mahalanobis distance.



Sketch of proof of approximate optimality

- Algorithm is guaranteed to obtain a partition with MWBC $\leq 4\lambda$, though does much better than that in practice.

Sketch of proof of approximate optimality

- Algorithm is guaranteed to obtain a partition with MWBC $\leq 4\lambda$, though does much better than that in practice.
- Sketch of proof:
- Each vertex is at most a path of two edges away from a block seed

  $\Longrightarrow$

  Worst case: two vertices i, j in the same block can be connected by a path of four edges in the nearest neighbors graph:
  Two from i to block seed, two from seed to j.

Sketch of proof of approximate optimality

- Algorithm is guaranteed to obtain a partition with MWBC $\leq 4\lambda$, though does much better than that in practice.
- Sketch of proof:
- Each vertex is at most a path of two edges away from a block seed

  $\Longrightarrow$

  Worst case: two vertices i, j in the same block can be connected by a path of four edges in the nearest neighbors graph:
  Two from i to block seed, two from seed to j.
- Worst case: there are vertices ℓ_1, ℓ_2, ℓ_3 that form a path of 4 edges connecting i to j:

$$i\ell_1,\ \ell_1\ell_2,\ \ell_2\ell_3,\ \ell_3 j \tag{1}$$

Sketch of proof

- Each edge has cost at most $\lambda \implies$
  The corresponding edge costs satisfy:

$$c_{i\ell_1} + c_{\ell_1\ell_2} + c_{\ell_2\ell_3} + c_{\ell_3 j} \leq 4\lambda.$$

Sketch of proof

- Each edge has cost at most $\lambda \implies$
  The corresponding edge costs satisfy:

$$c_{i\ell_1} + c_{\ell_1\ell_2} + c_{\ell_2\ell_3} + c_{\ell_3 j} \leq 4\lambda.$$

- Since edge costs satisfy the triangle inequality:

$$c_{ij} \leq c_{i\ell_1} + c_{\ell_1\ell_2} + c_{\ell_2\ell_3} + c_{\ell_3 j} \leq 4\lambda.$$

- That is, every edge joining two vertices within the same block has cost $\leq 4\lambda$.
- Hence, MWBC of the approximately optimal partition is $\leq 4\lambda$.
- QED