Multi-Level Edge Separator Using a Hybrid Combinatorial-Quadratic Programming Approach

Nuri Yeralan, Timothy A. Davis,
and William Hager

University of Florida

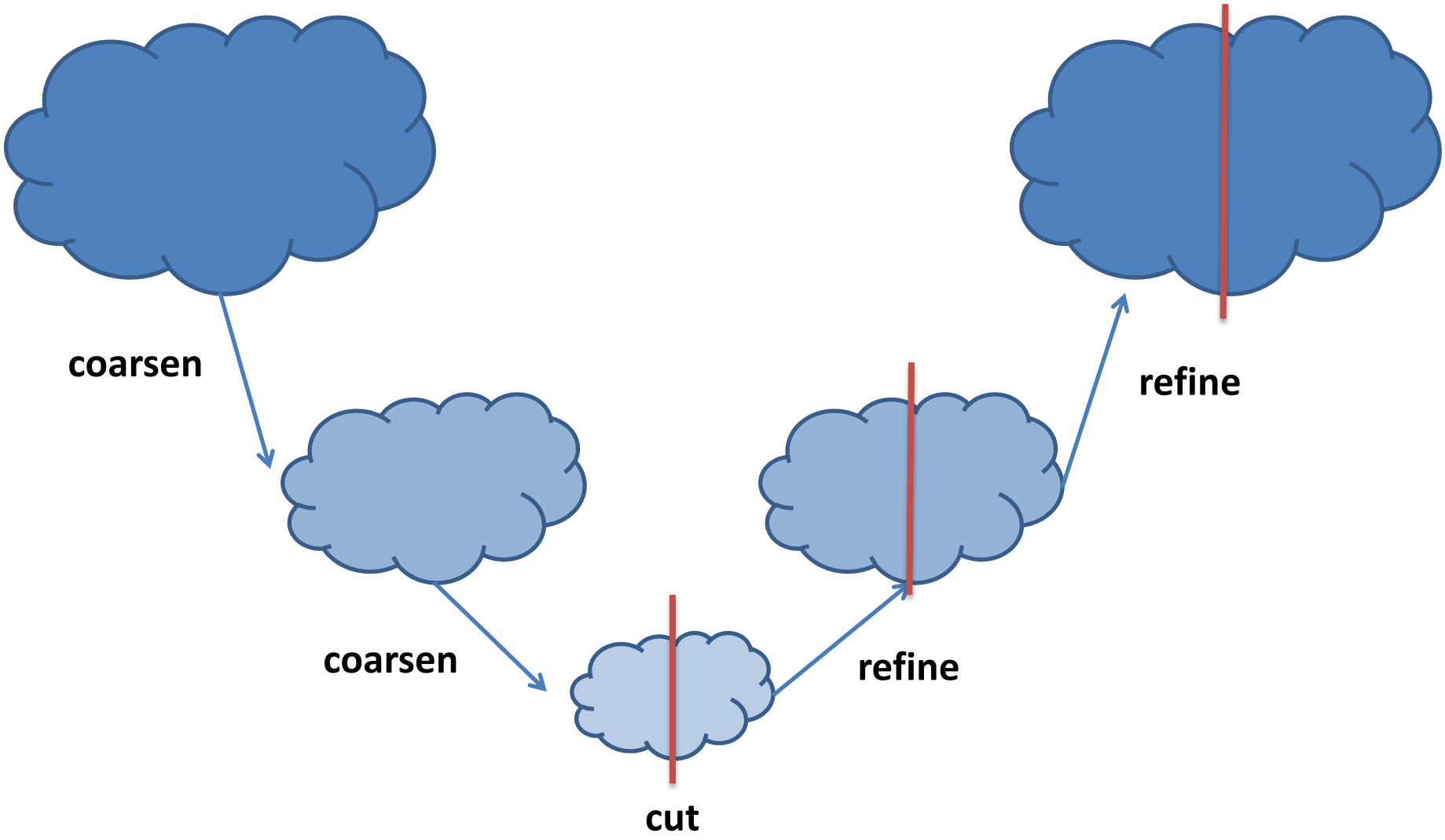
Graph Partitioning Problem

- Partition an undirected graph into 2 subgraphs
- Minimize the sum of edge weights in cut set
- Maintain target partition balance
- NP Hard

Multi-Level Partitioning

- **Coarsen** the input to reduce problem size
- **Cut** the coarsest graph
- **Refine** back to the original input

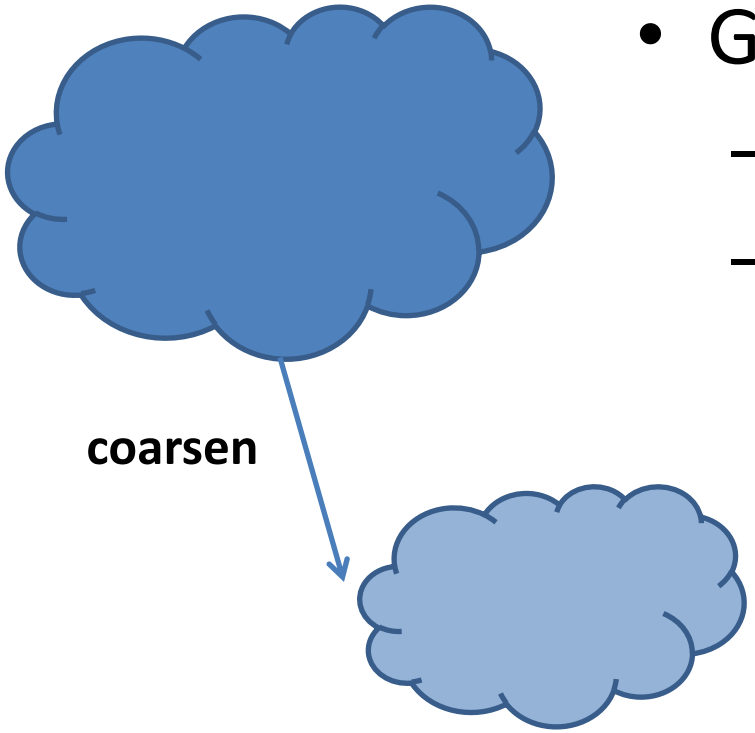
Multi-Level Partitioning



Multi-Level Partitioning

- Variations
  - "V" cycle
    - Single sweep, down and up
  - "W" cycle
    - Refine & alternate between deepest levels
  - Randomization
    - Bifurcate on way down
    - Select best of alternatives on way up
    - Parallelism!

Coarsening



- Goals
  - Reduce problem size
  - Remove heavy edges

Matching

- Glue two or more vertices together
  - Update connectivity as set union of adjacency lists summing edge weights
  - Sum the node weights

Matching Strategies

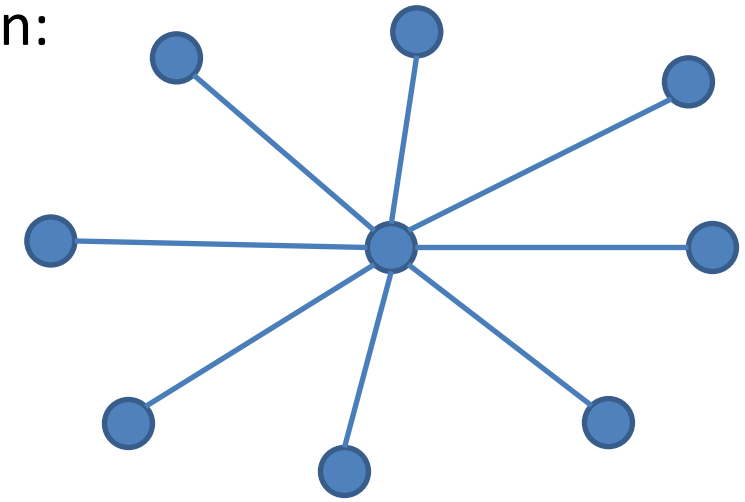
- Random Matching (RM)
  - Match vertices across the graph at random
- Heavy Edge Matching (HEM)
  - Traverse adjacency lists of unmatched vertices and match to the heaviest unmatched neighbor.
- Heaviest Edge Matching (SHEM)
  - Sort edges in decreasing order by edge weight
  - Perform valid matches in order

Matching Strategies

- Heavy Triangle Matching (HTM)
  - Do a local 2-step BFS on unmatched vertices
  - Match the heaviest valid triangle
  - (Treat absence of edge as 0-weight edge)
- Heaviest Clique Matching (HCM)
  - Find and sort 3-cliques in descending order of sum of edge weights
  - Perform valid matches in order
- Many More!

Matching Strategies

- Pitfalls
  - Some matching strategies are expensive
    - SHEM, HCM, HTM
  - No compression guarantees
    - High degree vertices, “stars” are notorious
    - Villain:

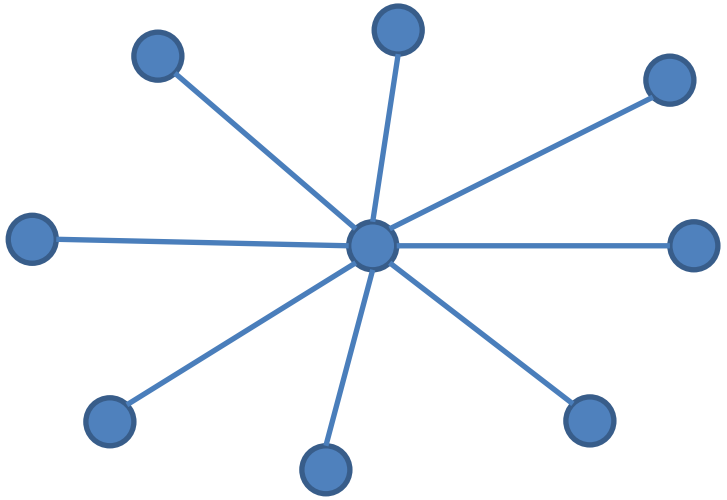


Matching Strategies

- “Brotherly” Matching
  - Match topologically close vertices through a common neighbor
  - Ex:

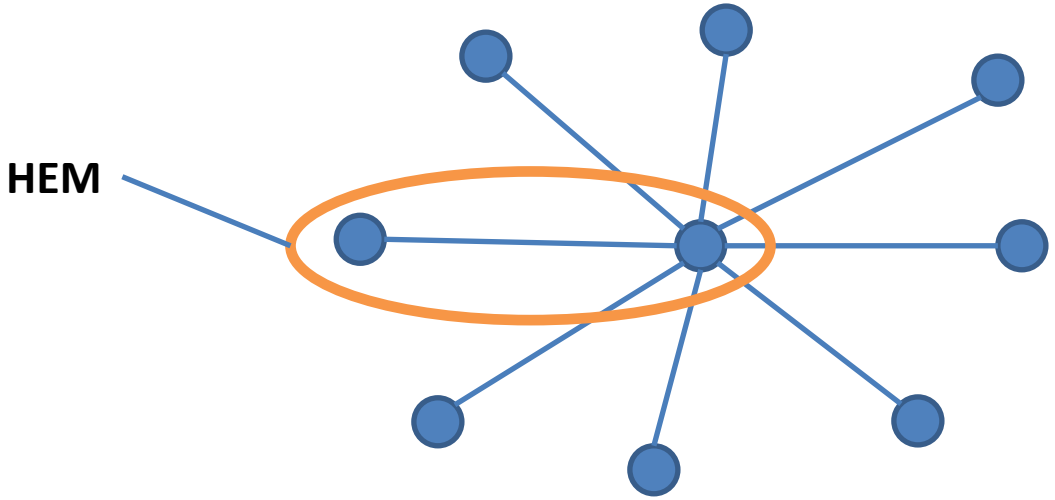
Matching Strategies

- “Brotherly” Matching
  - Match topologically close vertices through a common neighbor
  - Ex:



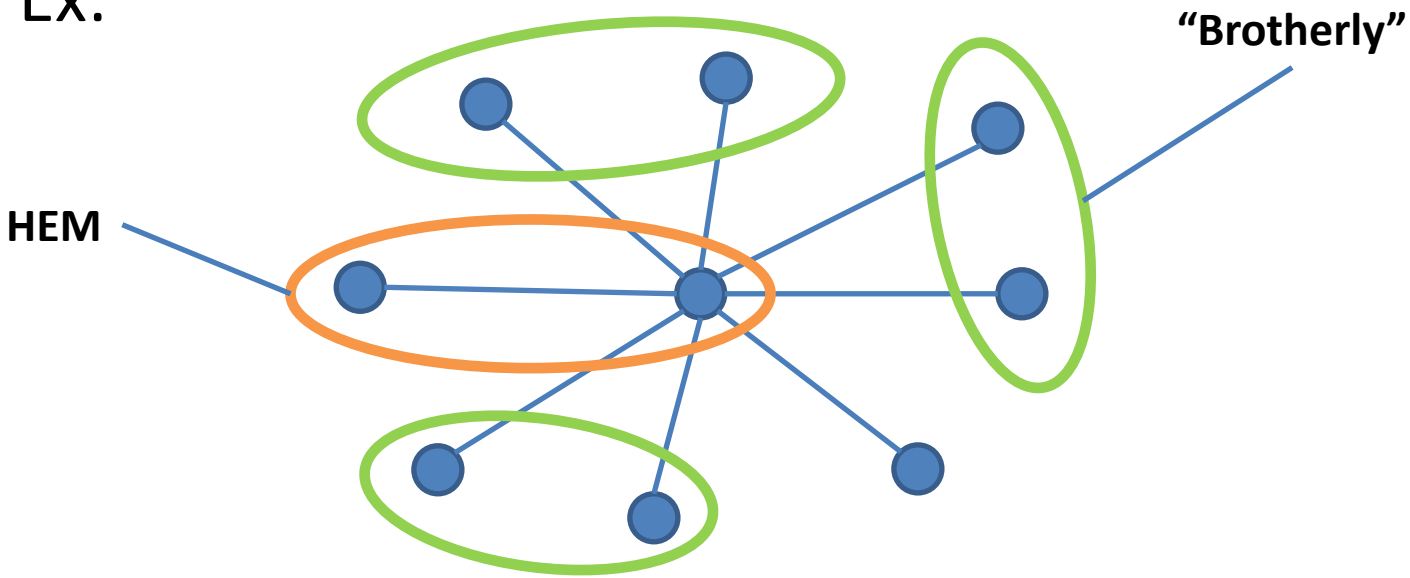
Matching Strategies

- "Brotherly" Matching
  - Match topologically close vertices through a common neighbor
  - Ex:



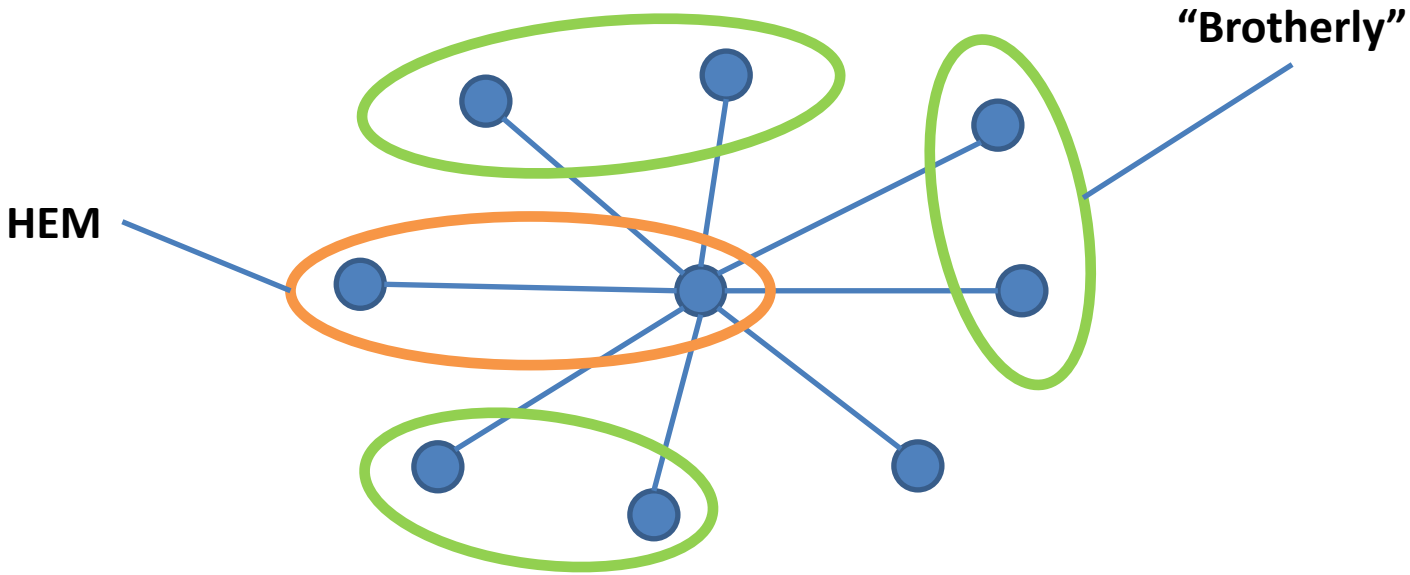
Matching Strategies

- "Brotherly" Matching
  - Match topologically close vertices through a common neighbor
  - Ex:



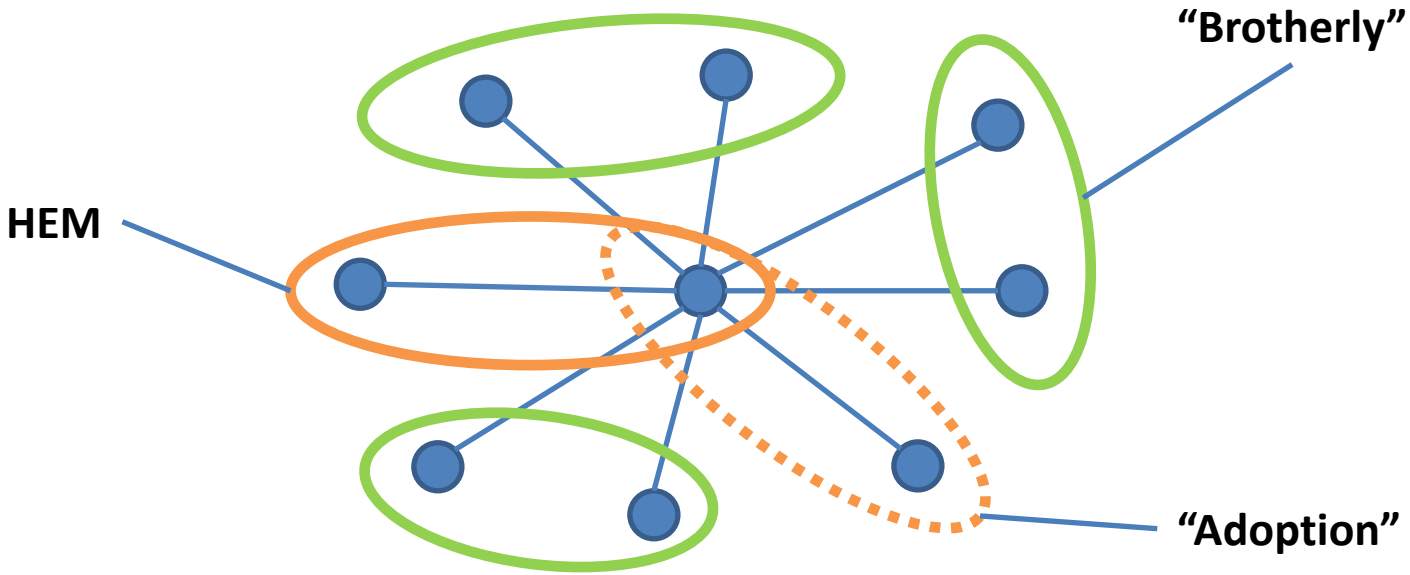
Matching Strategies

- "Adoption" Matching
  - Toss another vertex into an existing 2-way match
  - Ex:



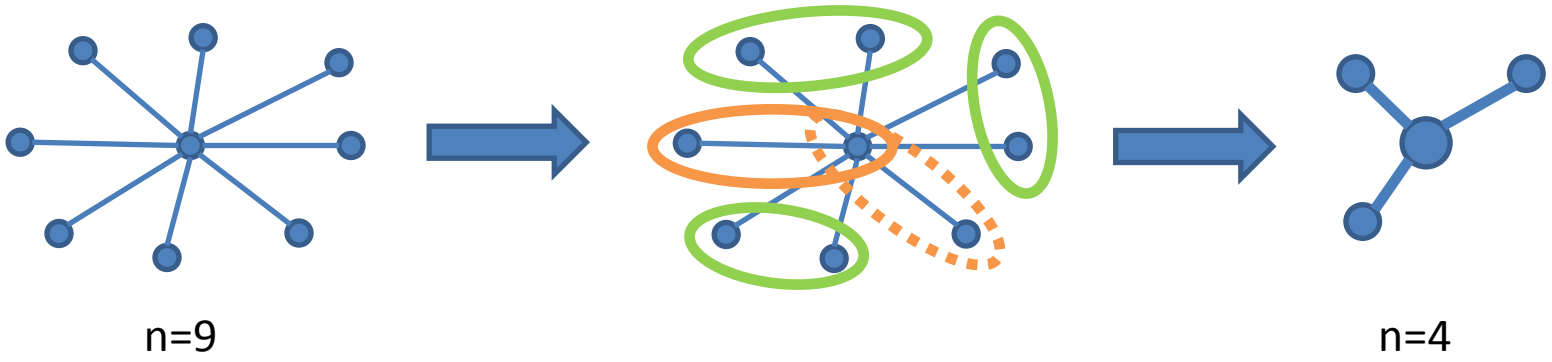
Matching Strategies

- "Adoption" Matching
  - Toss another vertex into an existing 2-way match
  - Ex:



Matching Strategies

- "Adoption" Matching
  - Toss another vertex into an existing 2-way match
  - Ex:



Matching Strategies

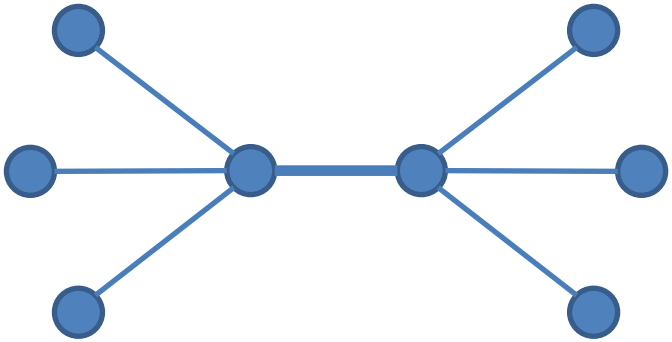
- To ensure productive coarsening, we want a bound on the number of coarsening phases.
- Can we guarantee O(log n) coarsening phases?

Matching Strategies

- “Community” Matching
  - After an adoption, if a second adoption is required at the same match site, match the old adoptee with the new adoptee.
  - Ex:

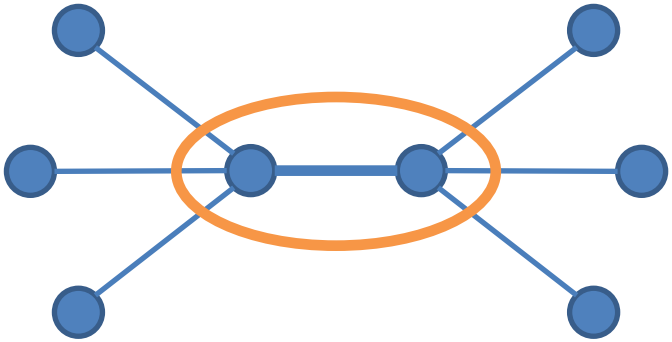
Matching Strategies

- "Community" Matching
  - After an adoption, if a second adoption is required at the same match site, match the old adoptee with the new adoptee.
  - Ex:



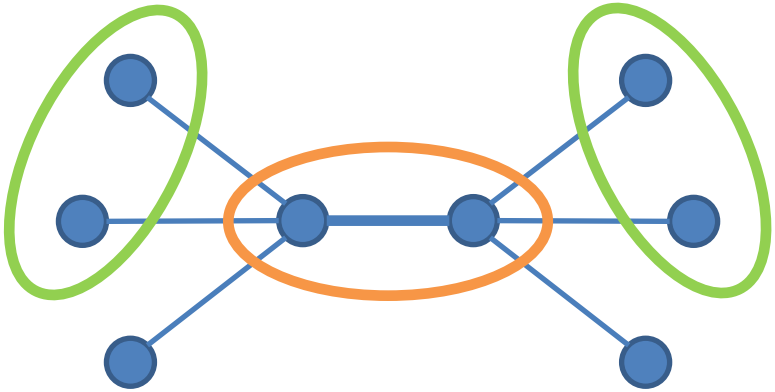
Matching Strategies

- "Community" Matching
  - After an adoption, if a second adoption is required at the same match site, match the old adoptee with the new adoptee.
  - Ex:



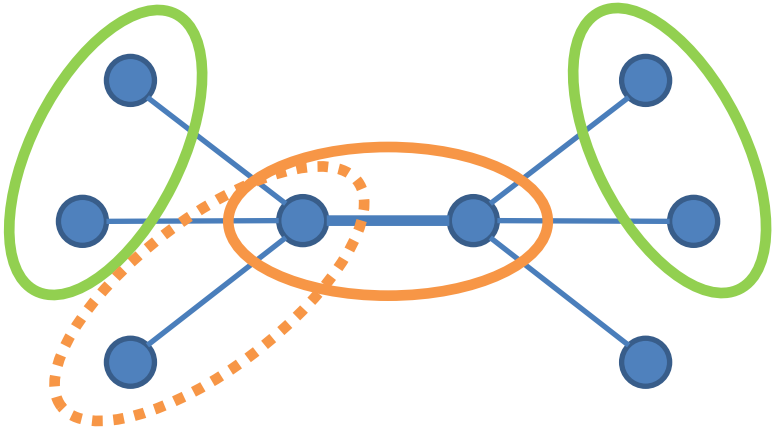
Matching Strategies

- "Community" Matching
  - After an adoption, if a second adoption is required at the same match site, match the old adoptee with the new adoptee.
  - Ex:



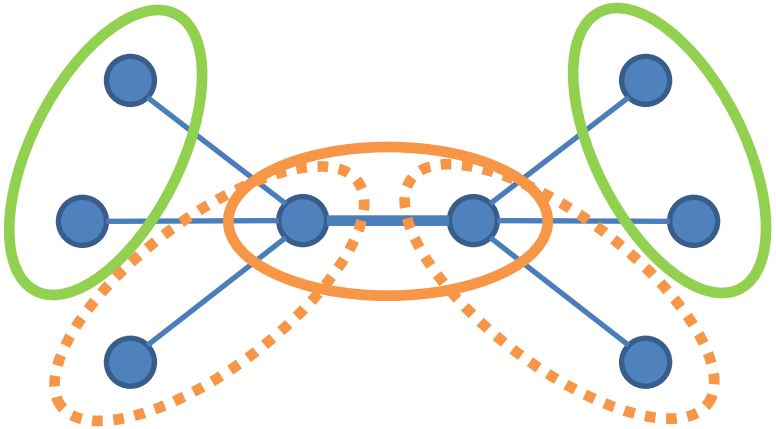
Matching Strategies

- "Community" Matching
  - After an adoption, if a second adoption is required at the same match site, match the old adoptee with the new adoptee.
  - Ex:



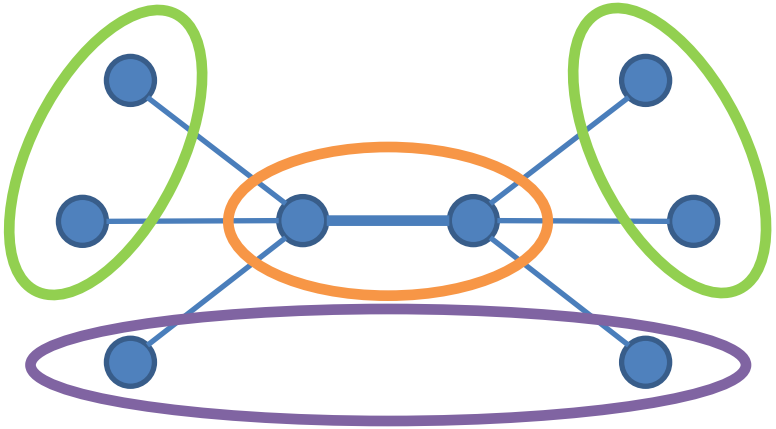
Matching Strategies

- "Community" Matching
  - After an adoption, if a second adoption is required at the same match site, match the old adoptee with the new adoptee.
  - Ex:



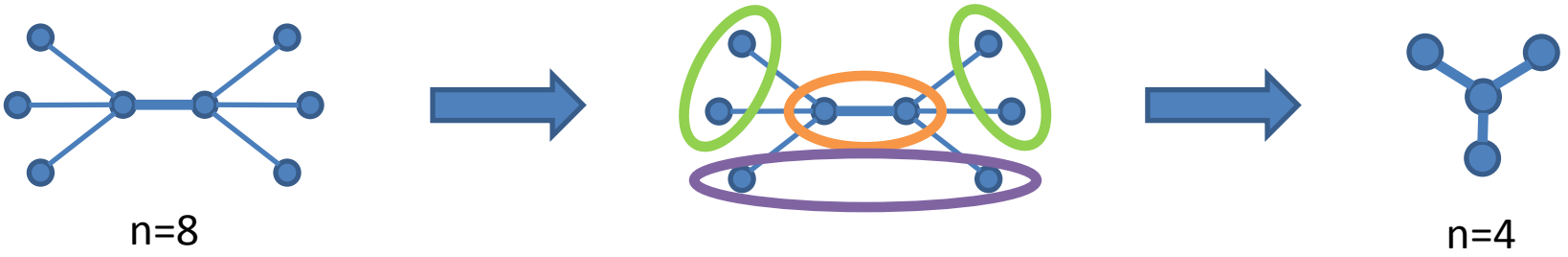
Matching Strategies

- "Community" Matching
  - After an adoption, if a second adoption is required at the same match site, match the old adoptee with the new adoptee.
  - Ex:



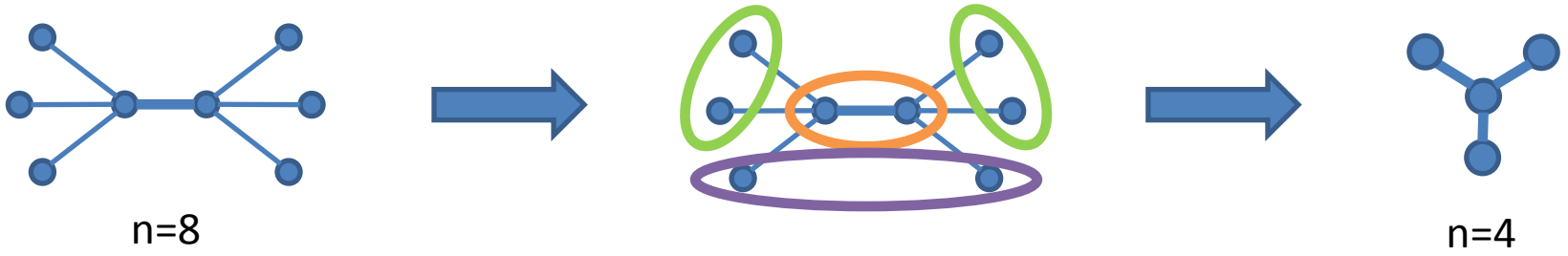
Matching Strategies

- "Community" Matching
  - After an adoption, if a second adoption is required at the same match site, match the old adoptee with the new adoptee.
  - Ex:



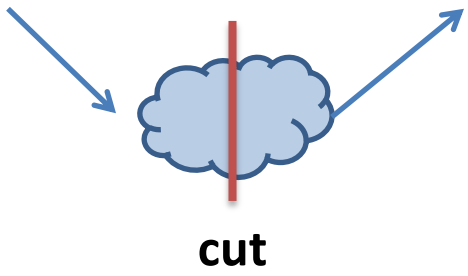
Matching Strategies

- "Community" Matching
  - After an adoption, if a second adoption is required at the same match site, match the old adoptee with the new adoptee.
  - Ex:



  - Equivalent to a brotherly match at the next level.

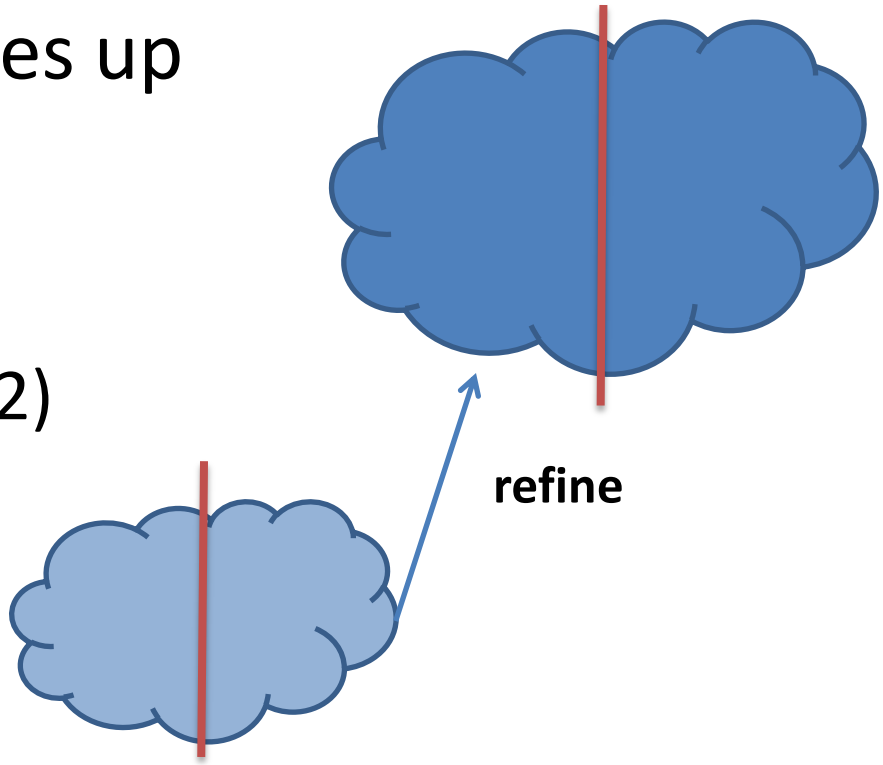
Initial Cut



- Random
- Clustering
- BFS-based
- Brute Force

Refinement

- Pass the partition choices up
- Swap vertex partitions
  - Kernighan-Lin (‘70)
  - Fiduccia-Mattheyses (‘82)
- Maintain balance



Mongoose - Overview

- Nuri Yeralan, Tim Davis, William Hager
- Hybrid graph partitioner
  - combinatorial & quadratic programming
- Single-threaded
- No randomization
- Handles floating point weights
- Interoperability with MATLAB, GraphViz, SuiteSparse, QPDelta

Mongoose - Coarsening

- Matching
  - "Jumpstart" HEM while building coarse graphs
  - Pass over unmatched vertices
    - Note: All neighbors of unmatched vertices are matched.
  - Select heaviest matched neighbor
  - Perform brotherly matching
  - Adopt/Community match orphaned vertices
  - Guarantees O(log n) coarsening levels

Mongoose - Coarsening

- Matching
  - In practice, we initiate brotherly matching if the vertex's degree exceeds twice the average.
  - Still productive (empirically) but no guarantee

Mongoose – Initial Cut

- Single Pseudoperipheral
  - Find a Pseudoperipheral Node (George ‘79)
  - Collect vertices into one partition using a BFS from a pseudoperipheral node until the partition exceeds the desired size.
  - Vertices sitting on the cut are considered “boundary” vertices and placed into a heap corresponding to their partition.

Mongoose - Refinement

- Boundary Fiduccia-Mattheyes
  - One heap per partition stores boundary vertices.
  - An entry's value is the result of an objective function combining FM gains with a linear balance penalty
    - Incorporates user-defined balance ratio and tolerance
    - Allows otherwise infeasible moves
  - Inspect the top 3 elements of each heap and swap the vertex with the highest heuristic value.
  - Karypis and Kumar introduced boundary KL in '97.

Mongoose - Refinement

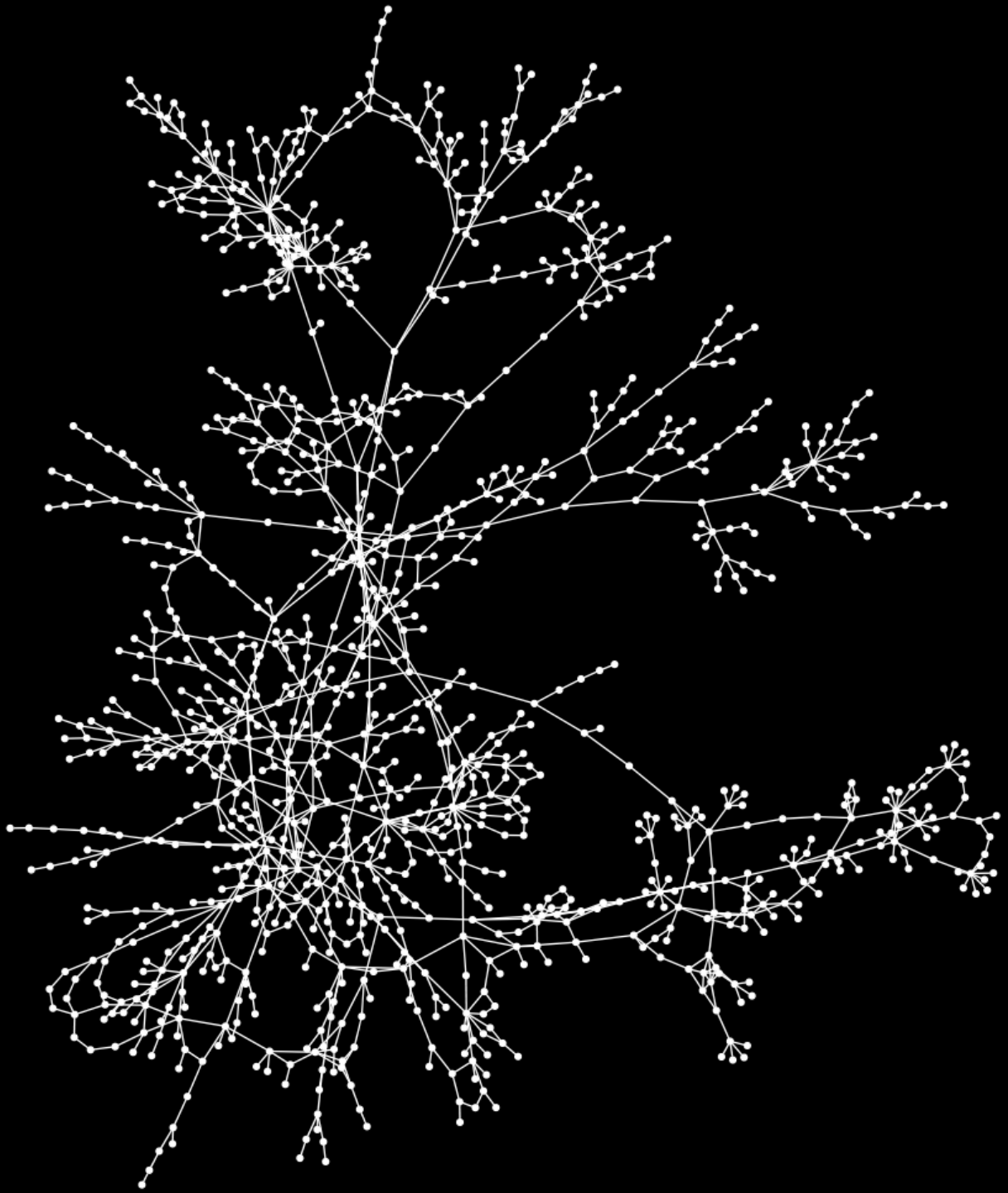
- Use optimization to solidify balance constraints and make moves that combinatorial methods miss.
- Quadratic Programming Formulation of the Graph Partitioning Problem (Hager '99)
  - minimize: $(1-x)^T(A+D)x$
    - D is a diagonal matrix such that $d_{ii} + d_{jj} \geq a_{ij}$
    - Mongoose sets d_{ii} to the sum of i's edge weights.
  - subject to: $0 \leq x \leq 1, 1^T x = m$
    - m: desired node weight for one partition

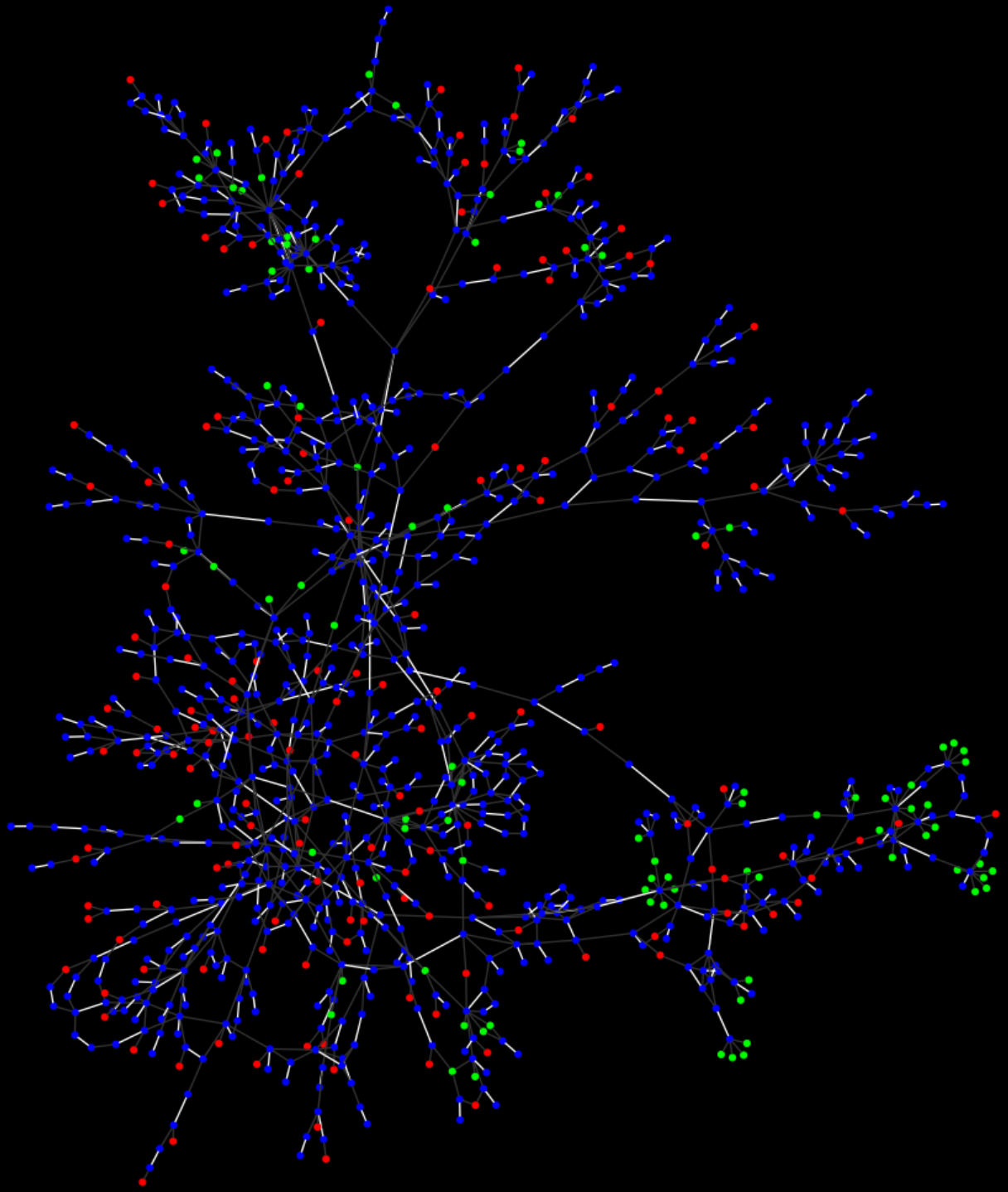
Mongoose - Refinement

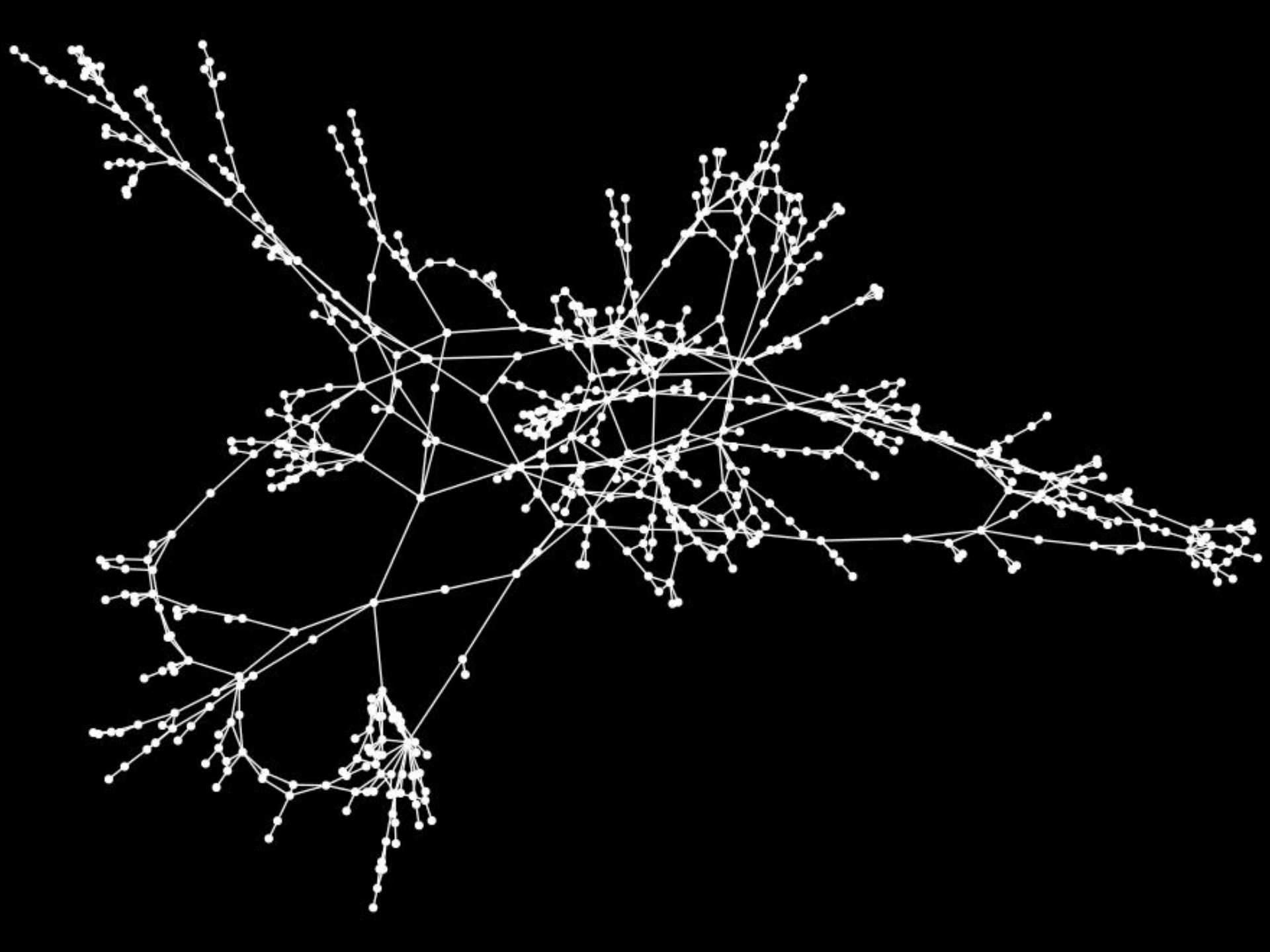
- QP Formulation: Gradient Projection
  - Compute a gradient vector at each vertex
    - $Grad = \left(0.5 - x^T\right)D + (0.5 - x^T)A$
      - $x_i \in [0,1]$ represents vertex i's partition choice
  - Move along the gradient
  - Project back onto feasible set

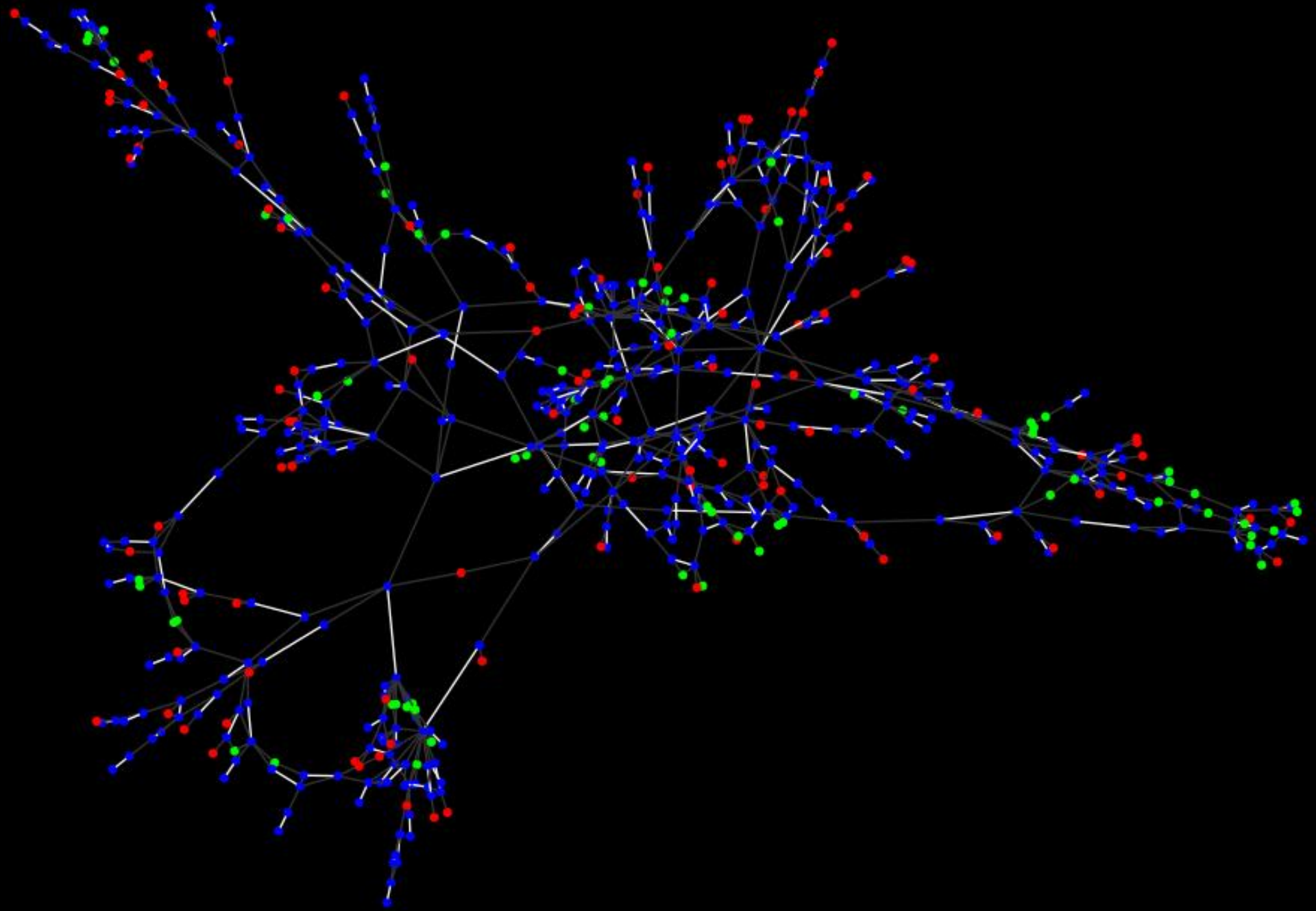
Mongoose In Action

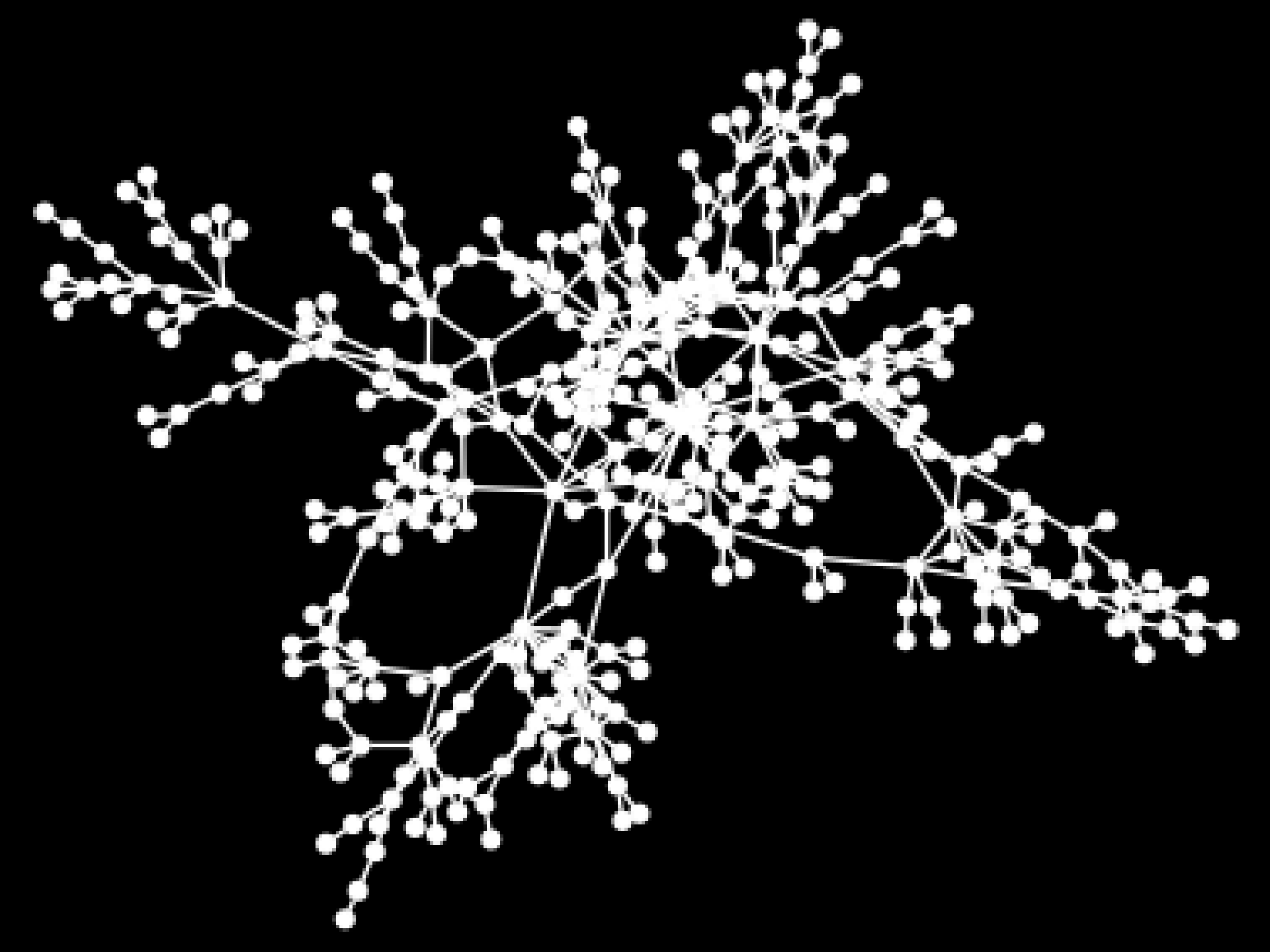
- Mongoose cut of **1138_bus** from the UF Sparse Matrix Library (Davis ‘10)
  - n = 1138
  - nz = 4054
  - Power network problem (bus power system) submitted by D. Tylavsky (‘87)
- GraphViz is used to render using a force-directed layout (AT&T Research, ‘88)

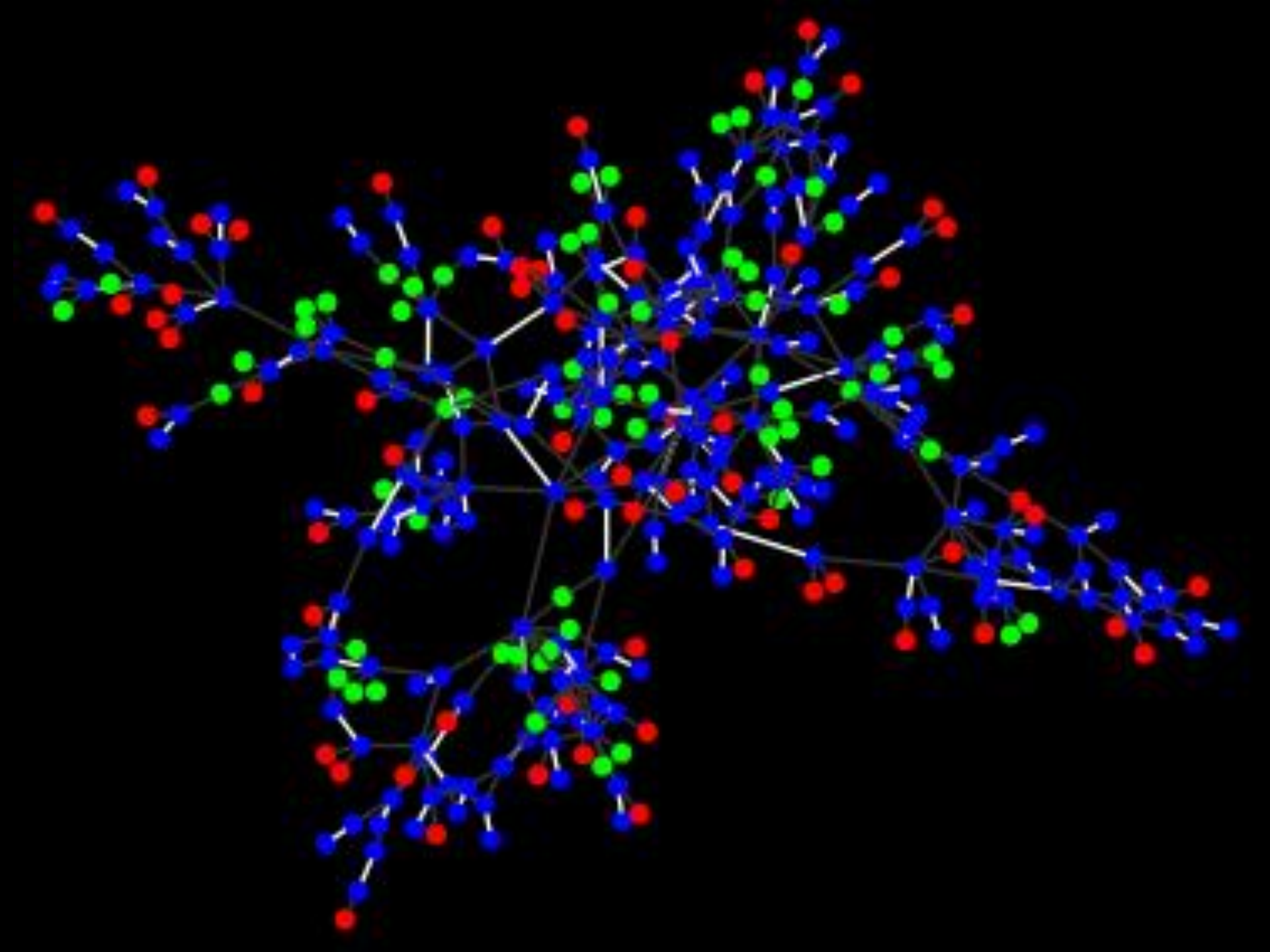


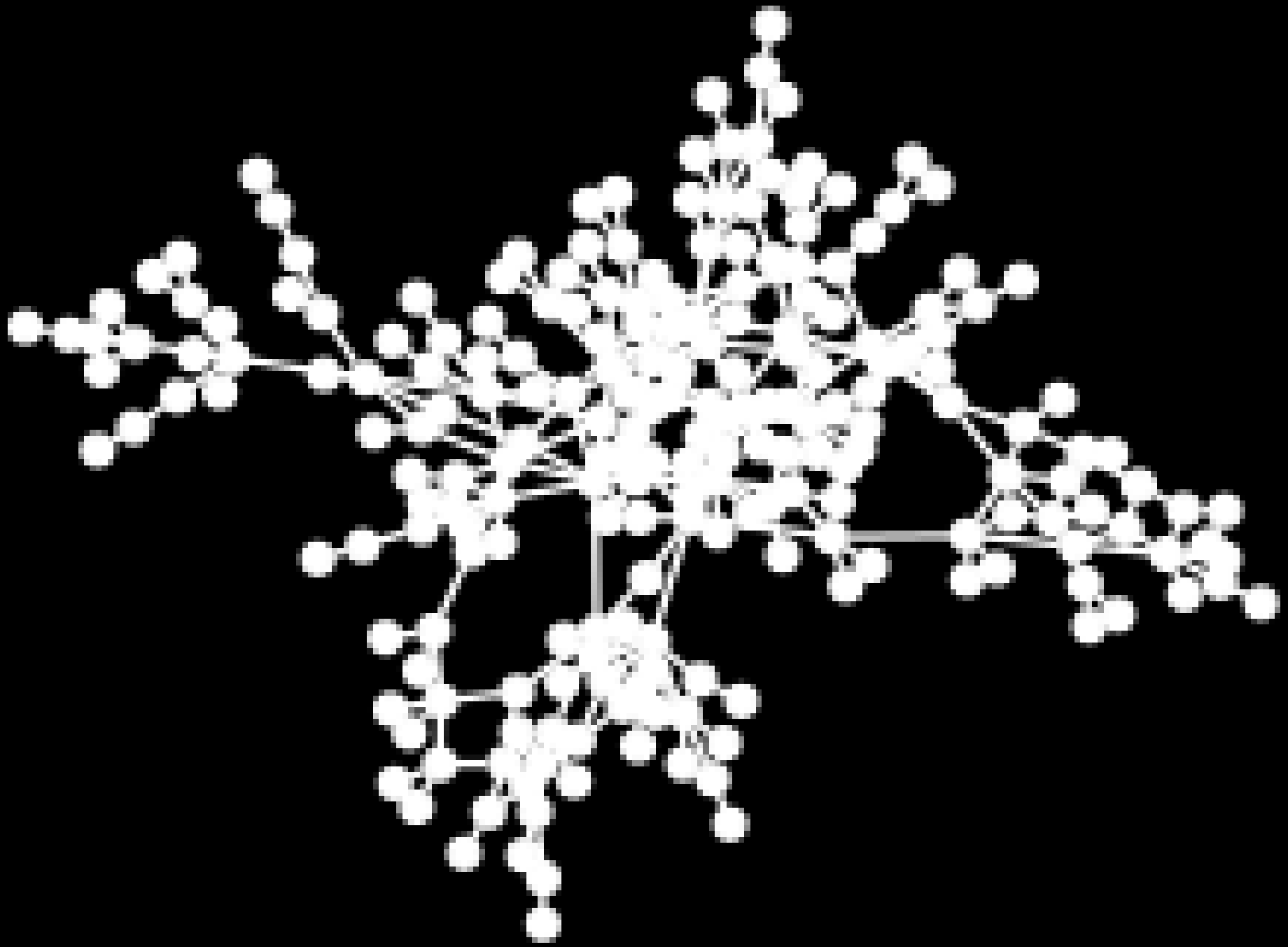


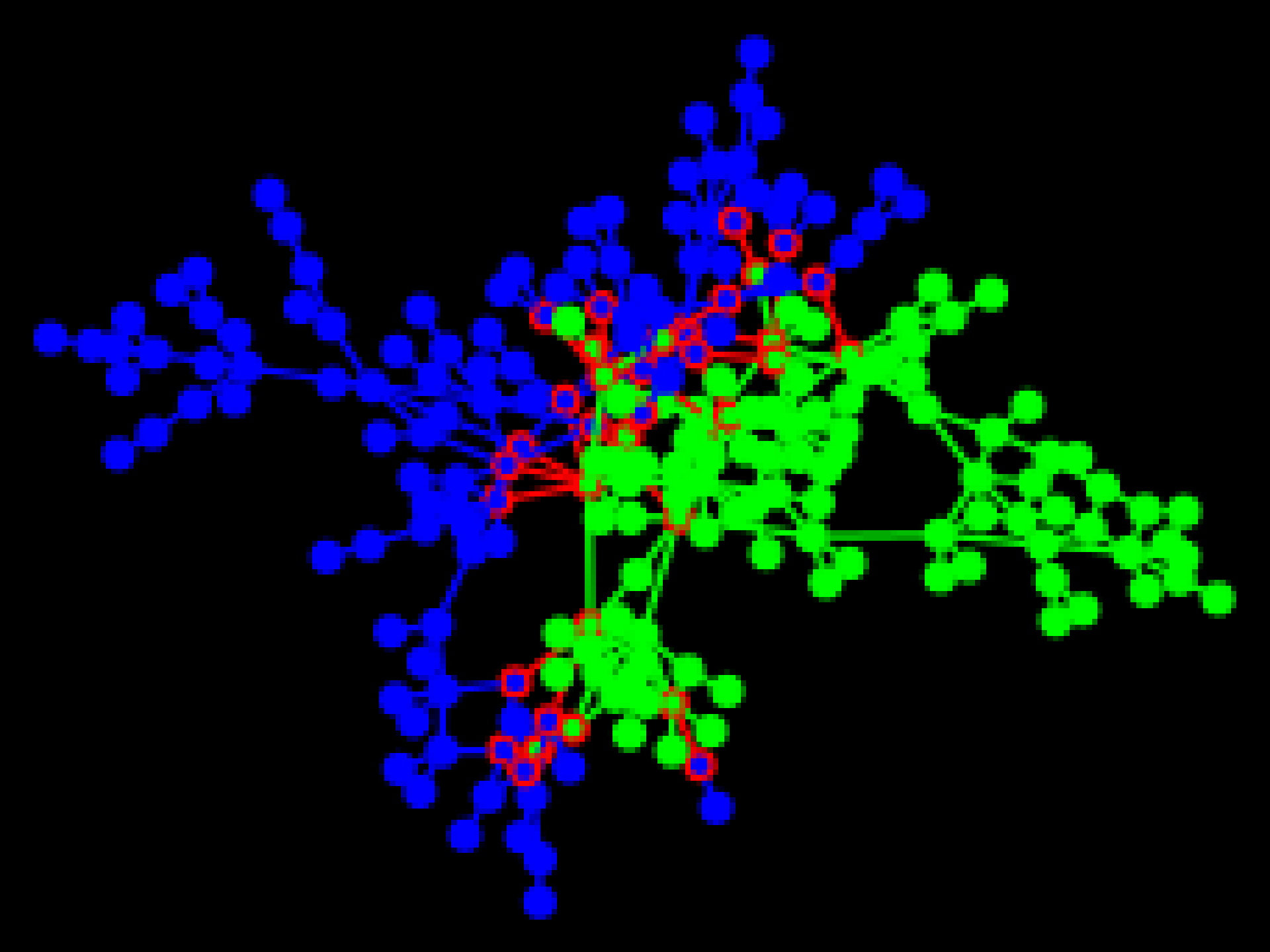


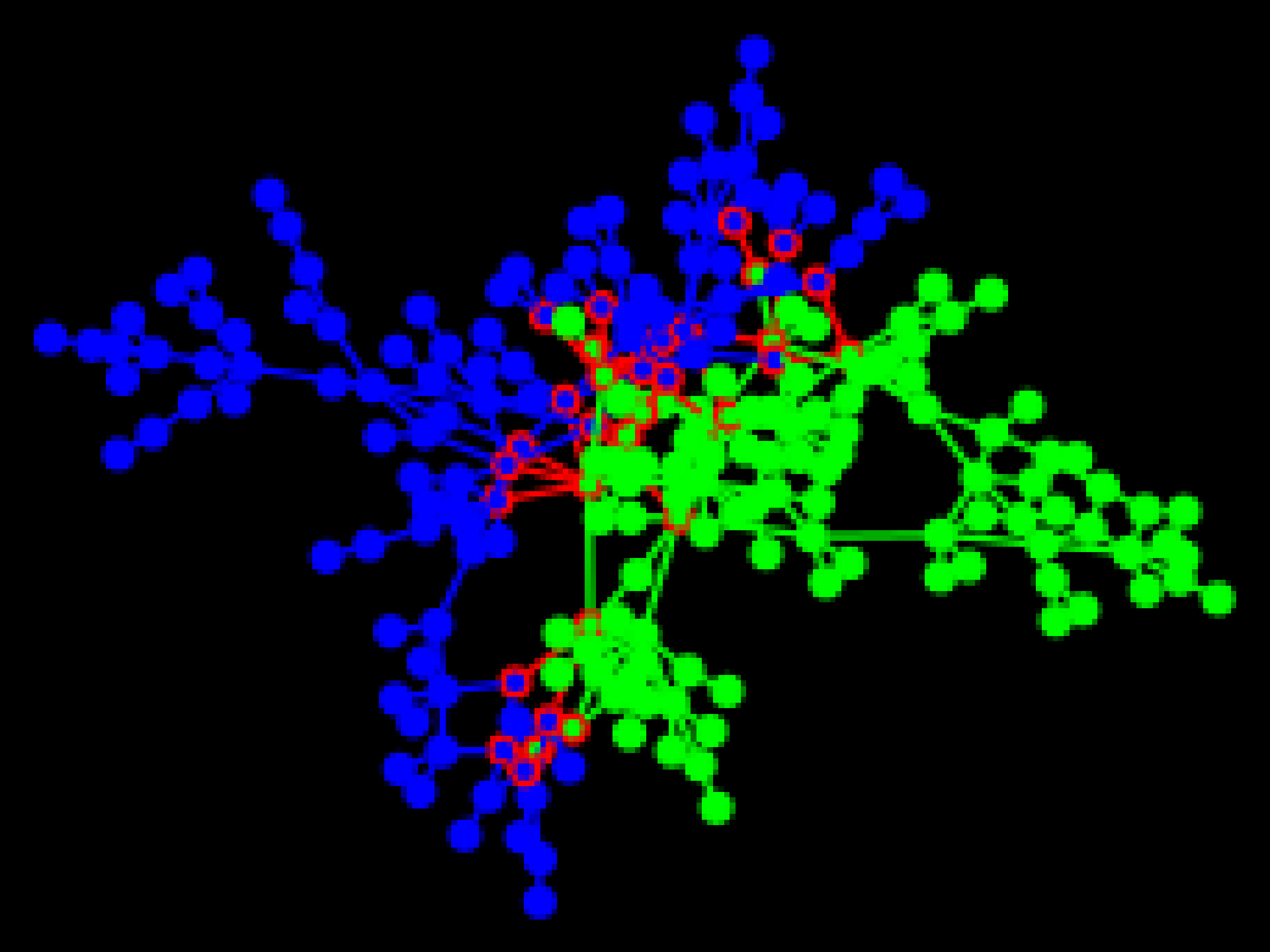


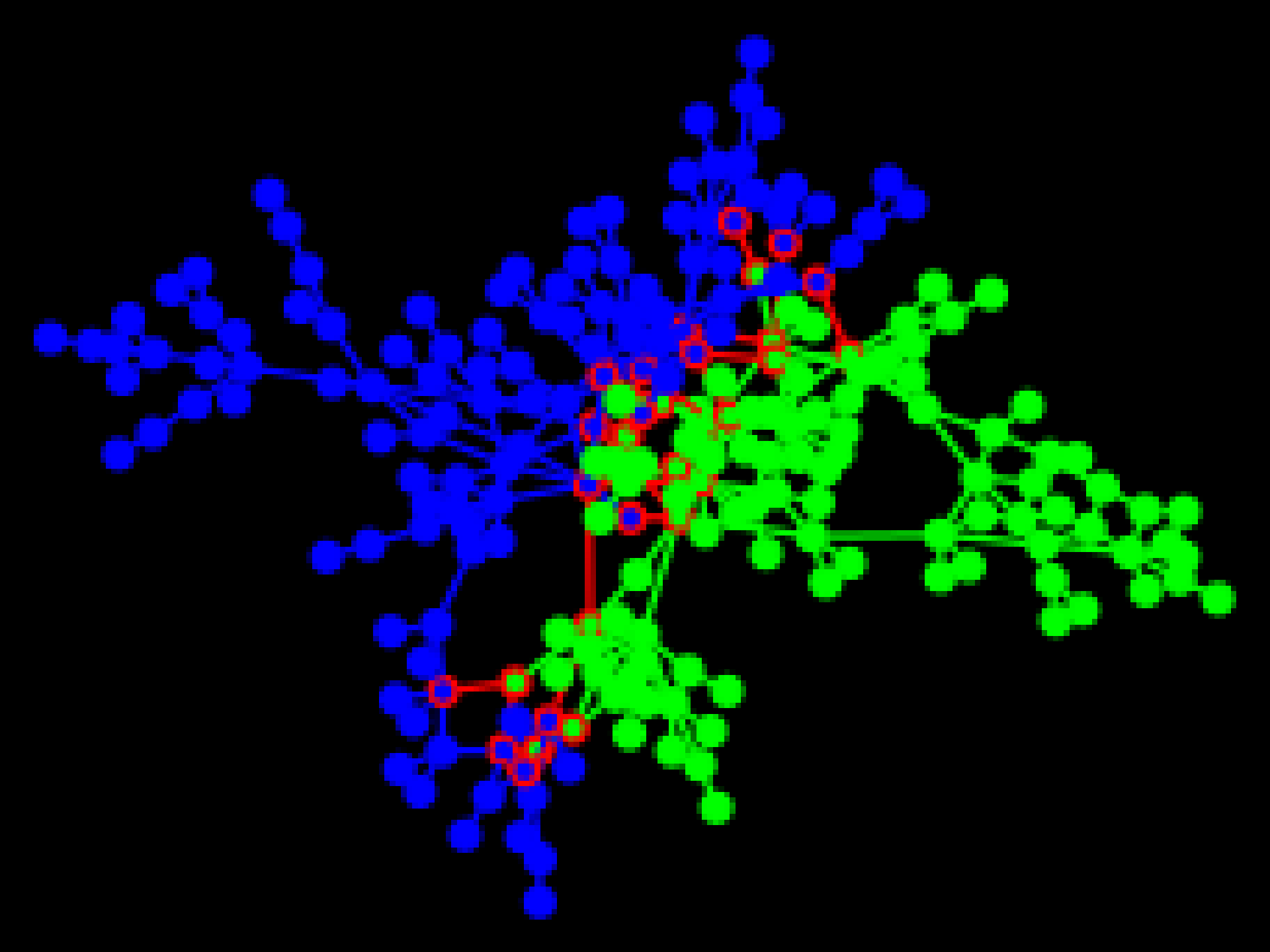


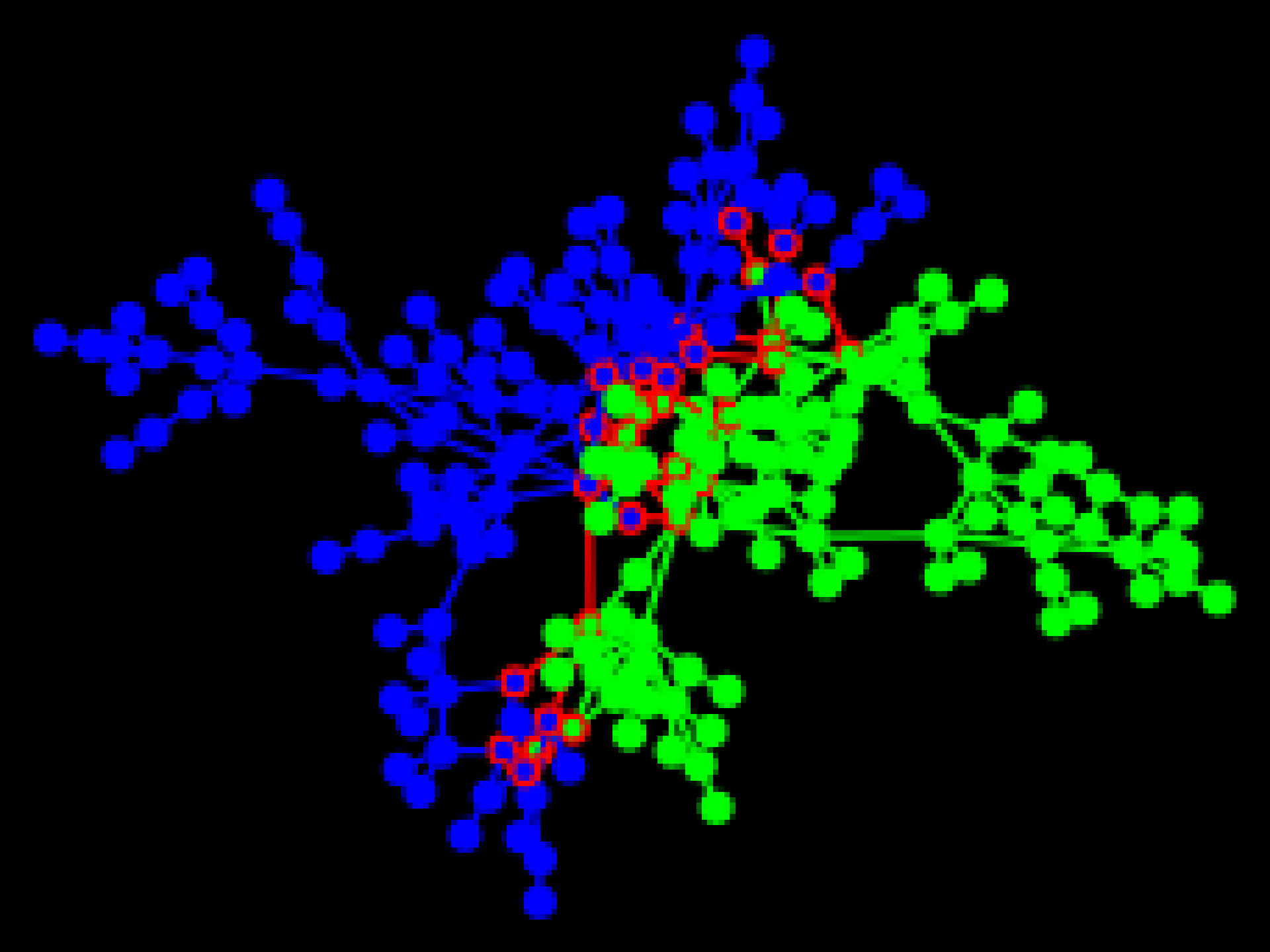


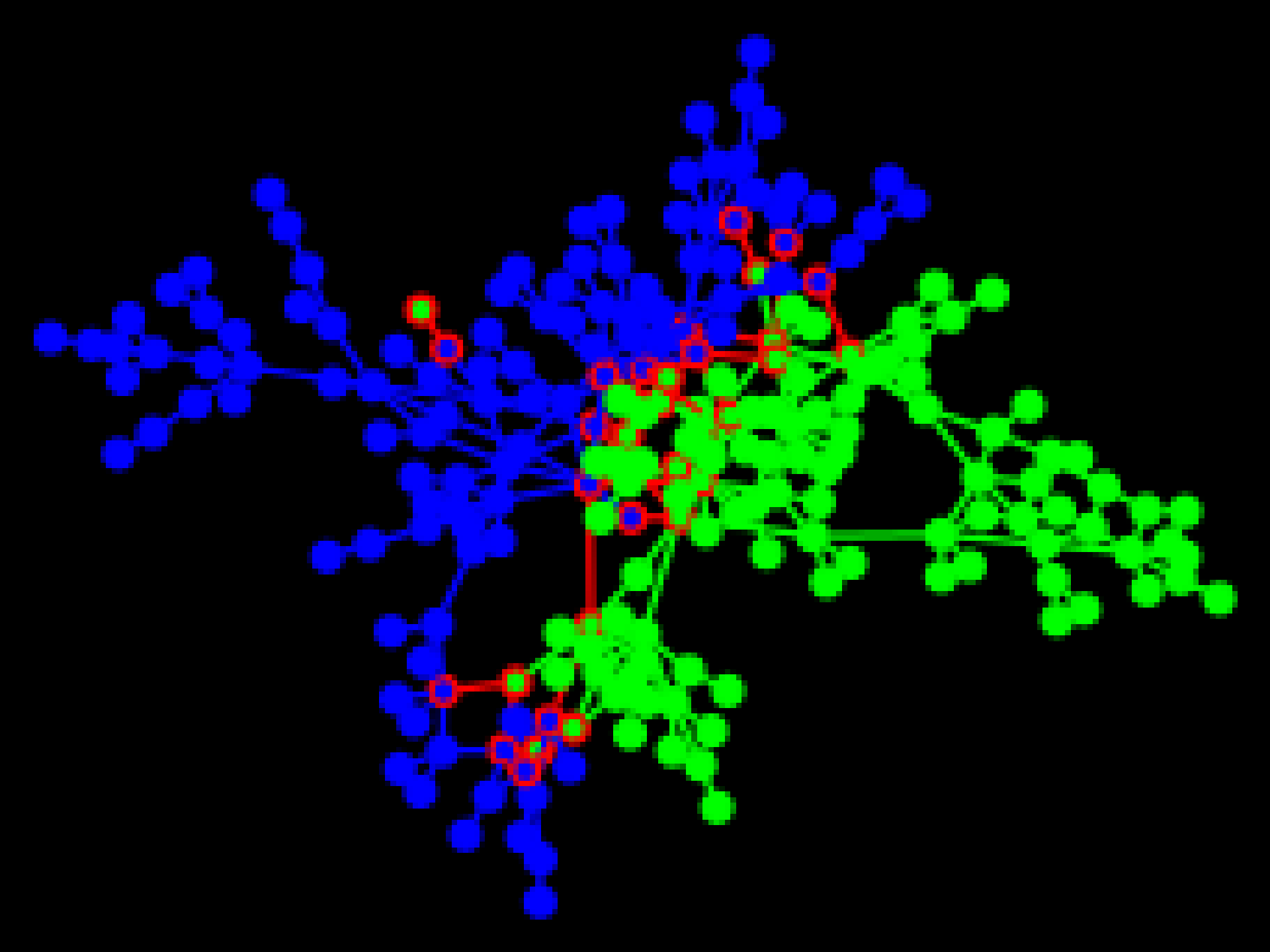


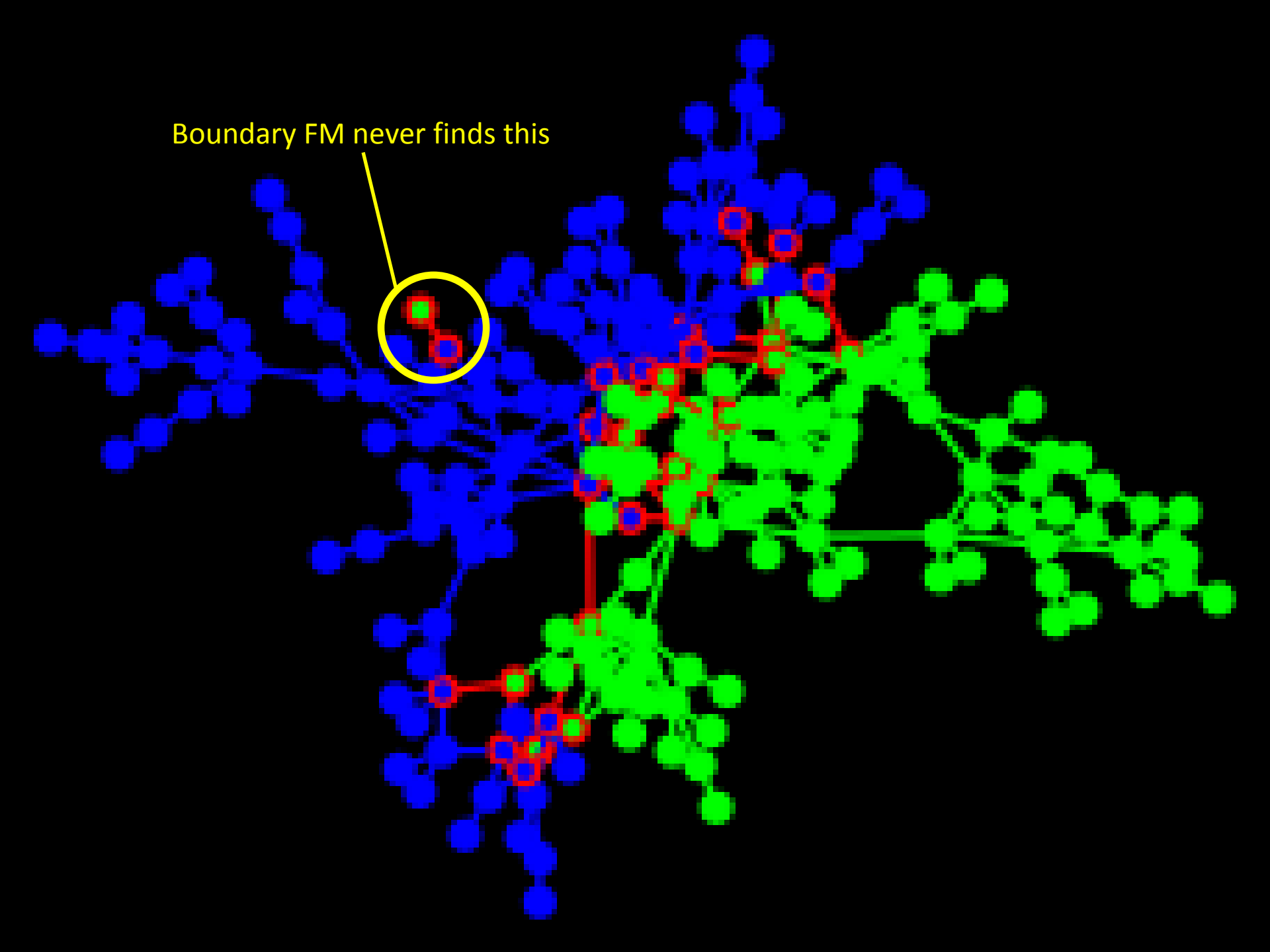


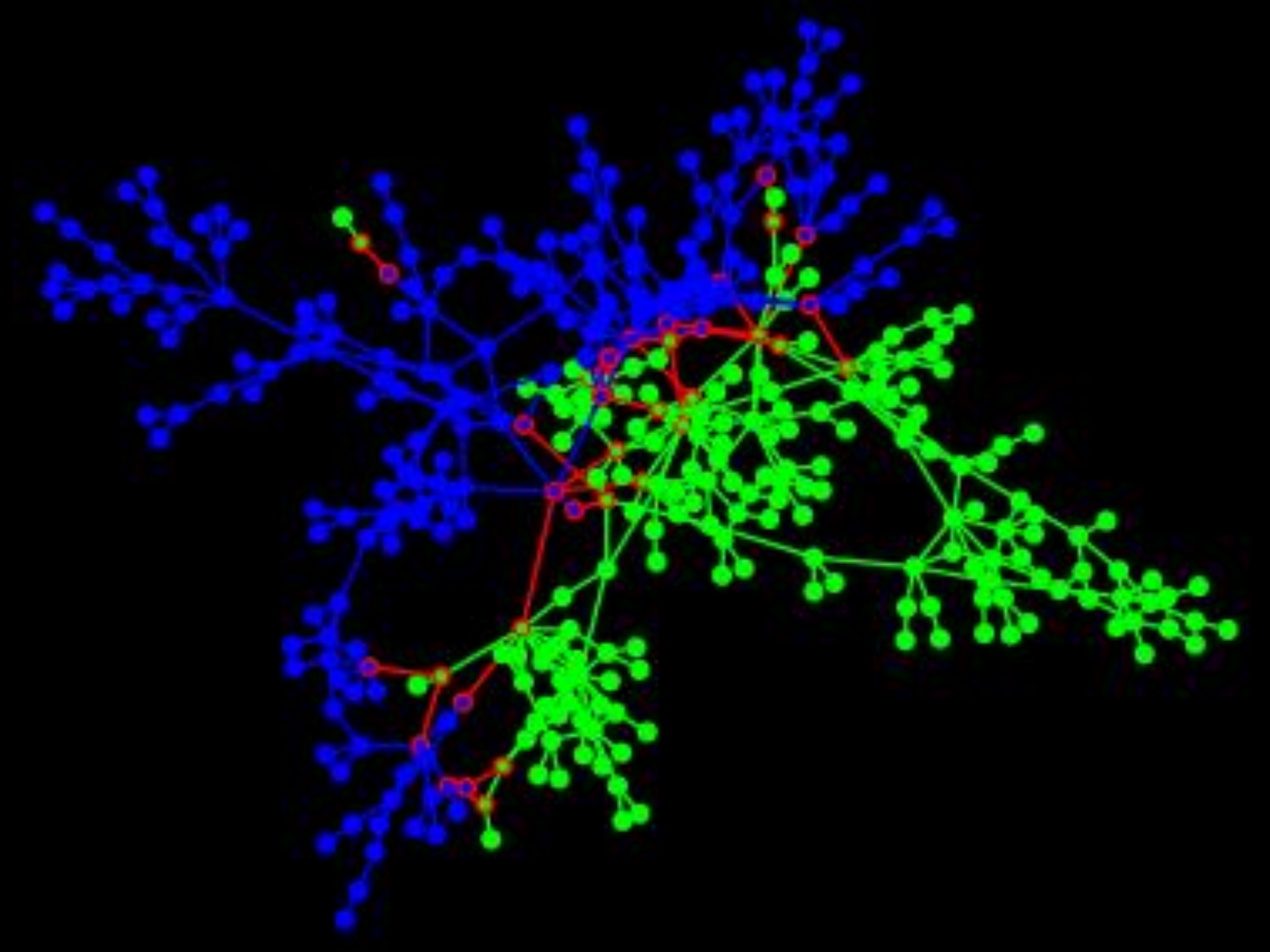


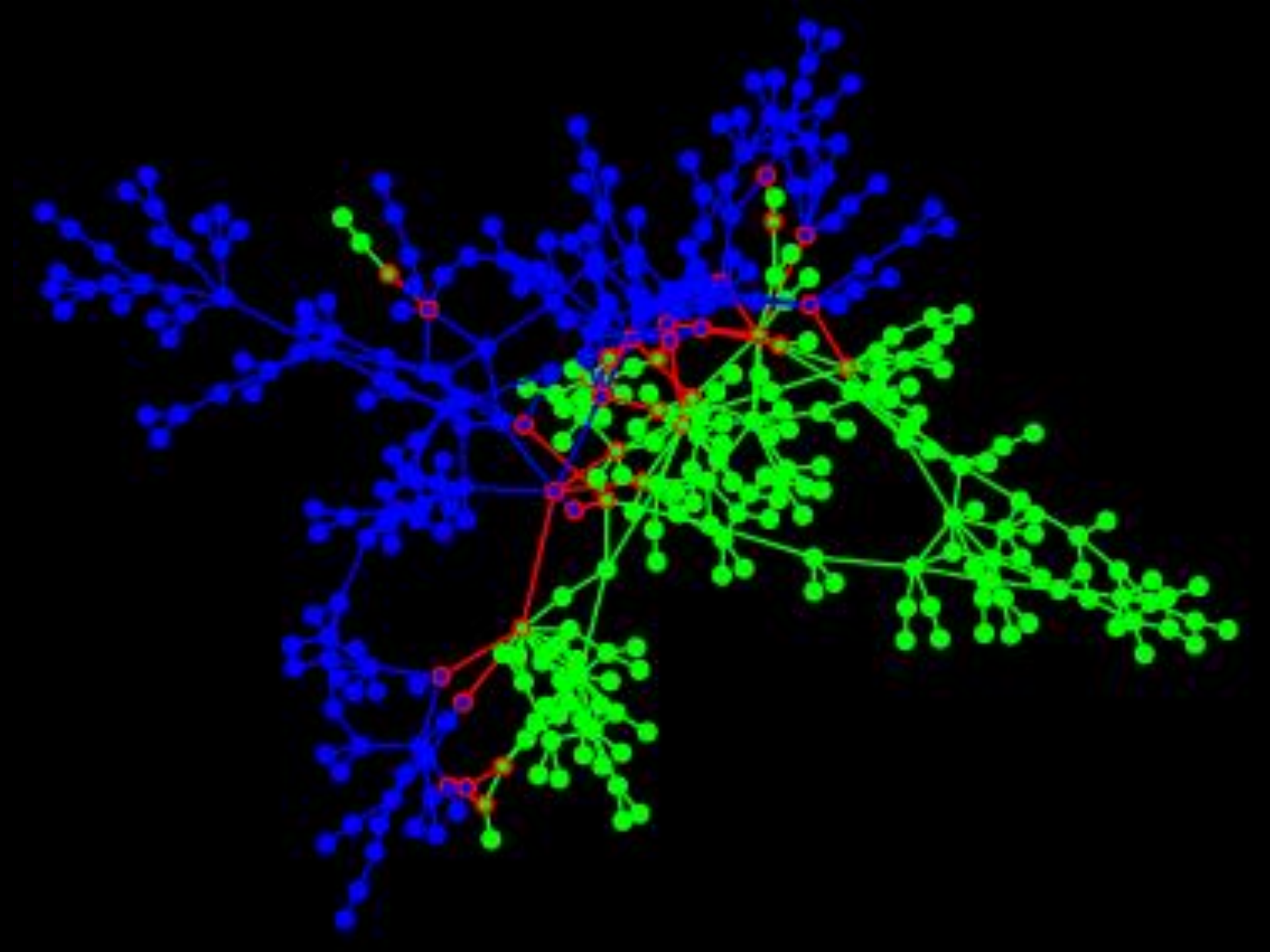


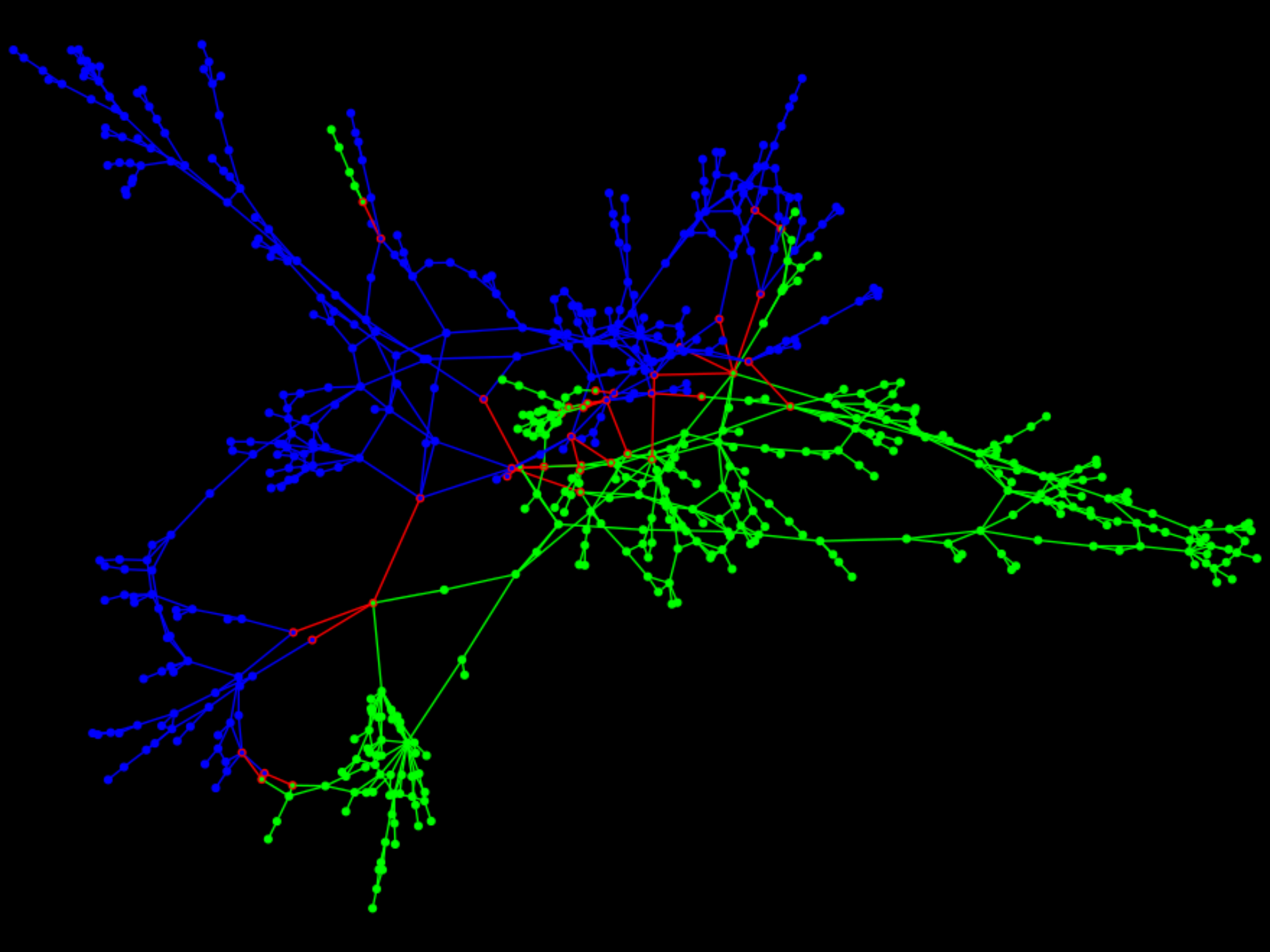


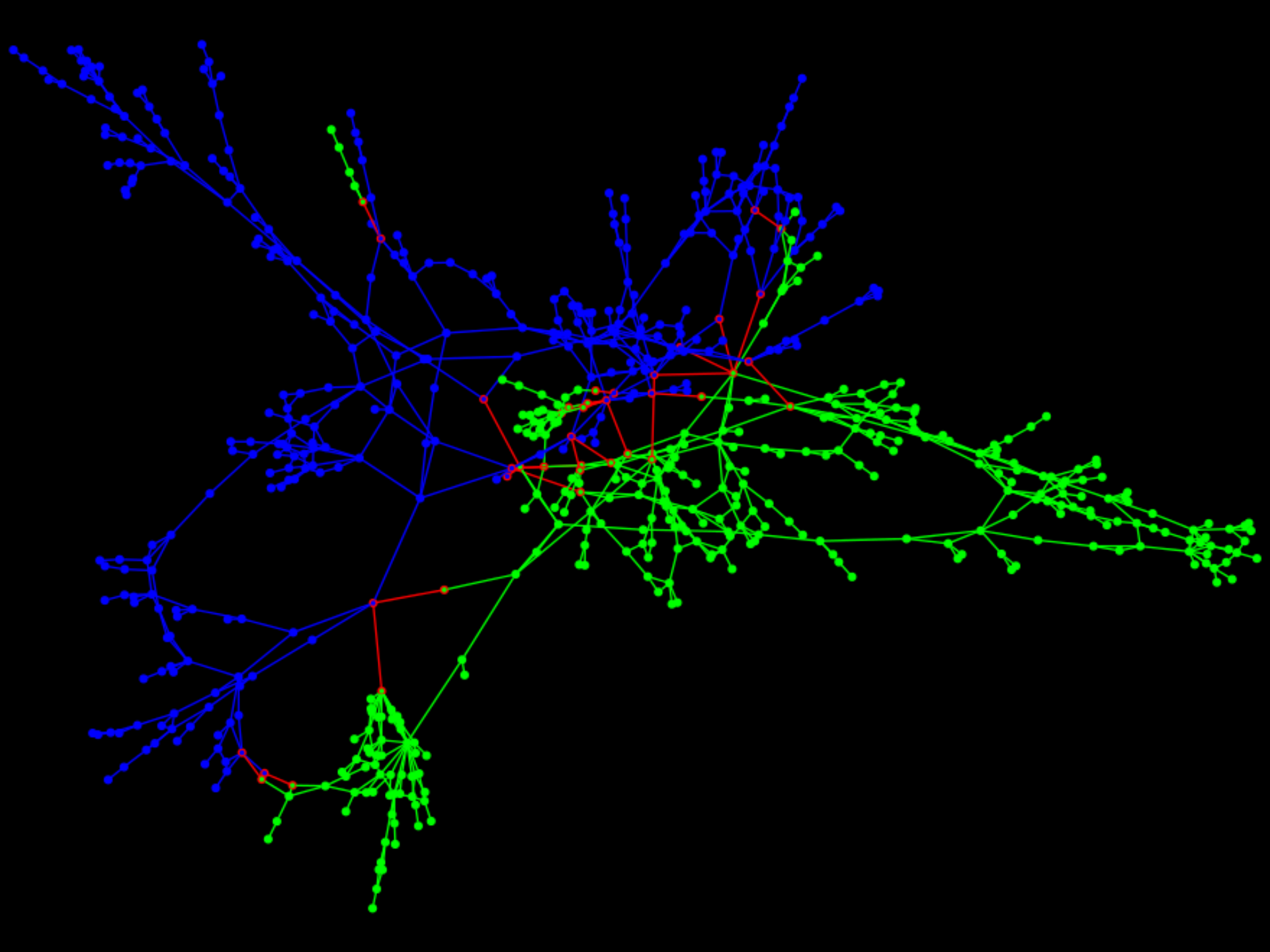


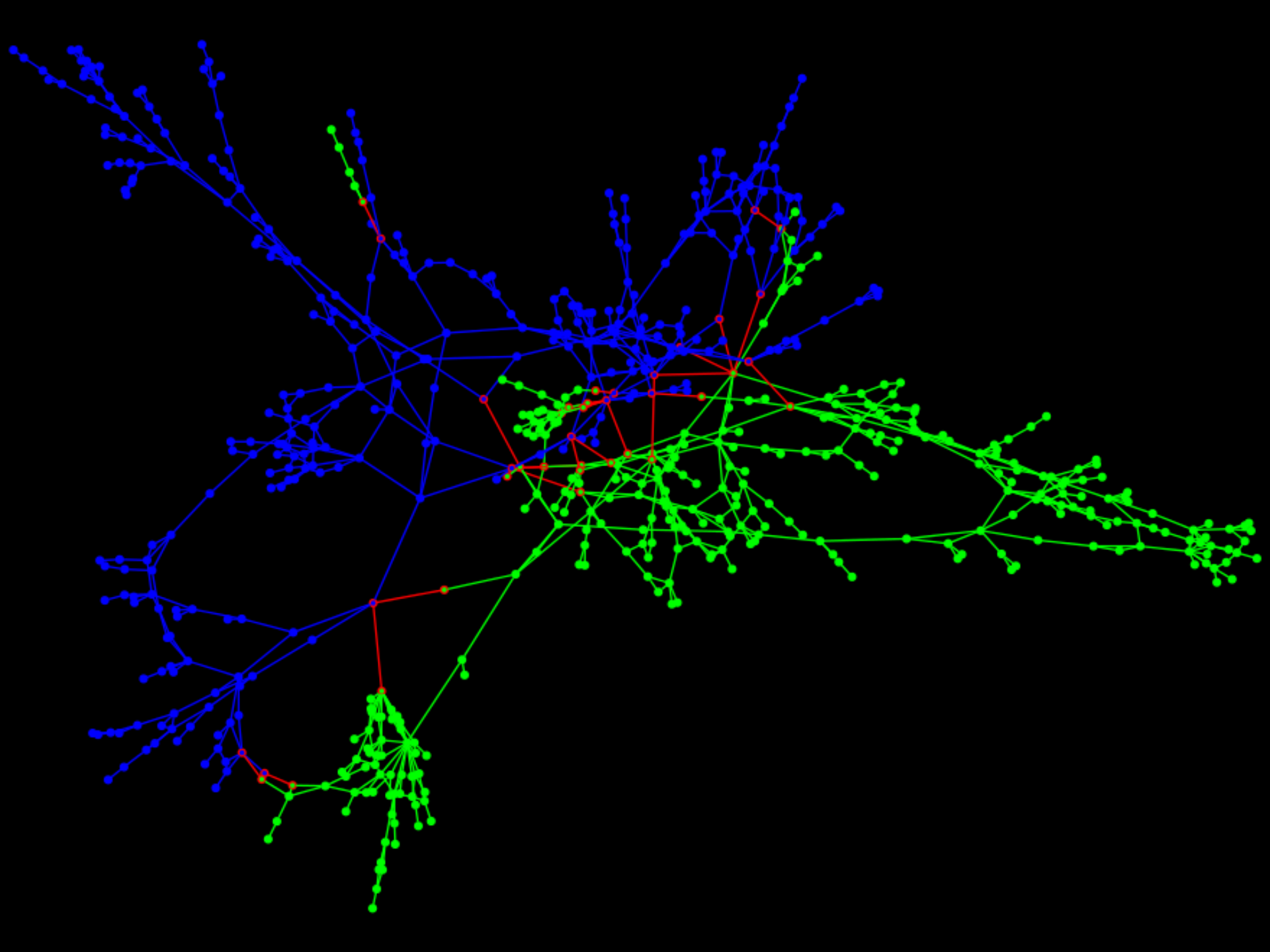

Boundary FM never finds this

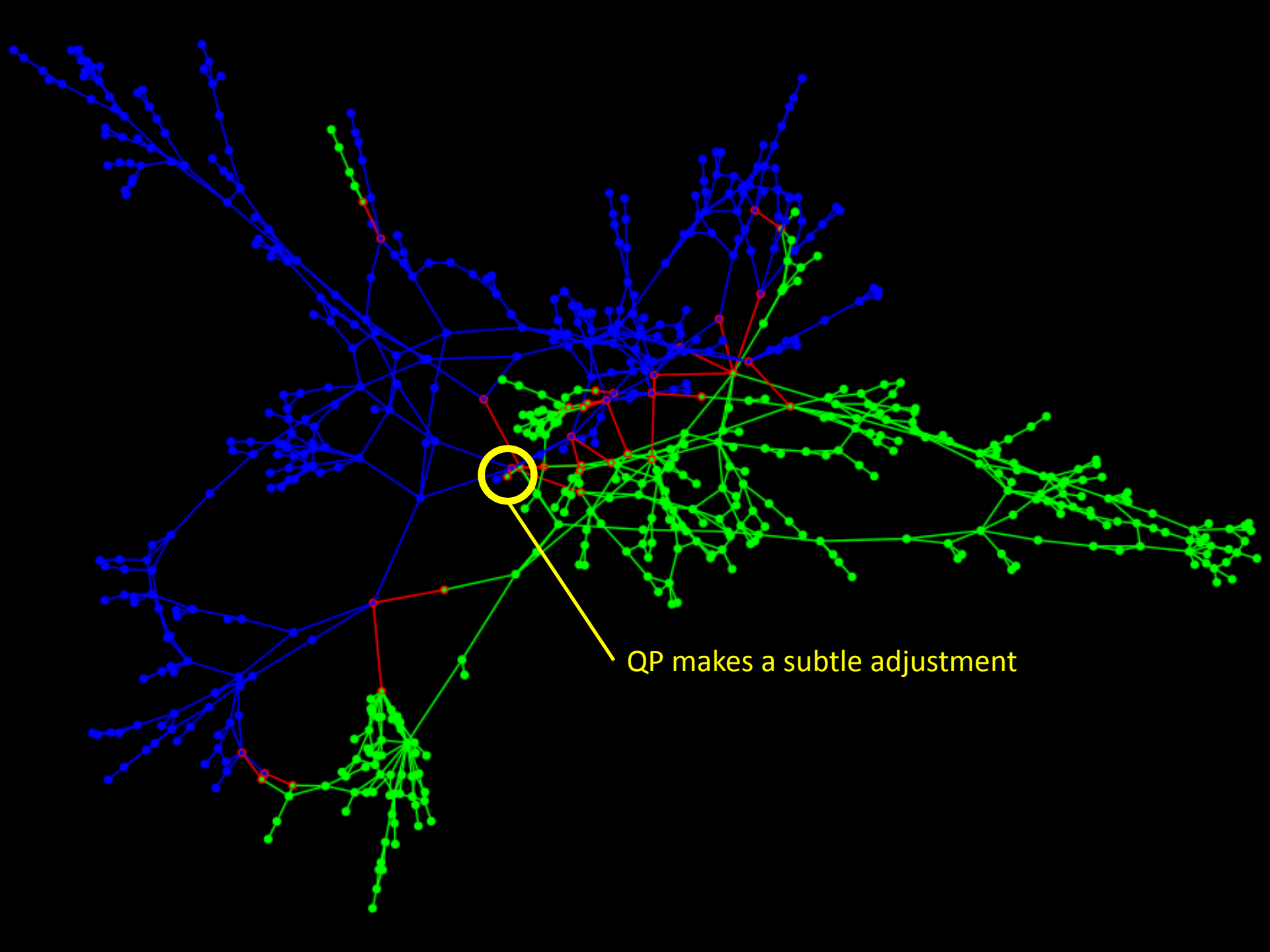


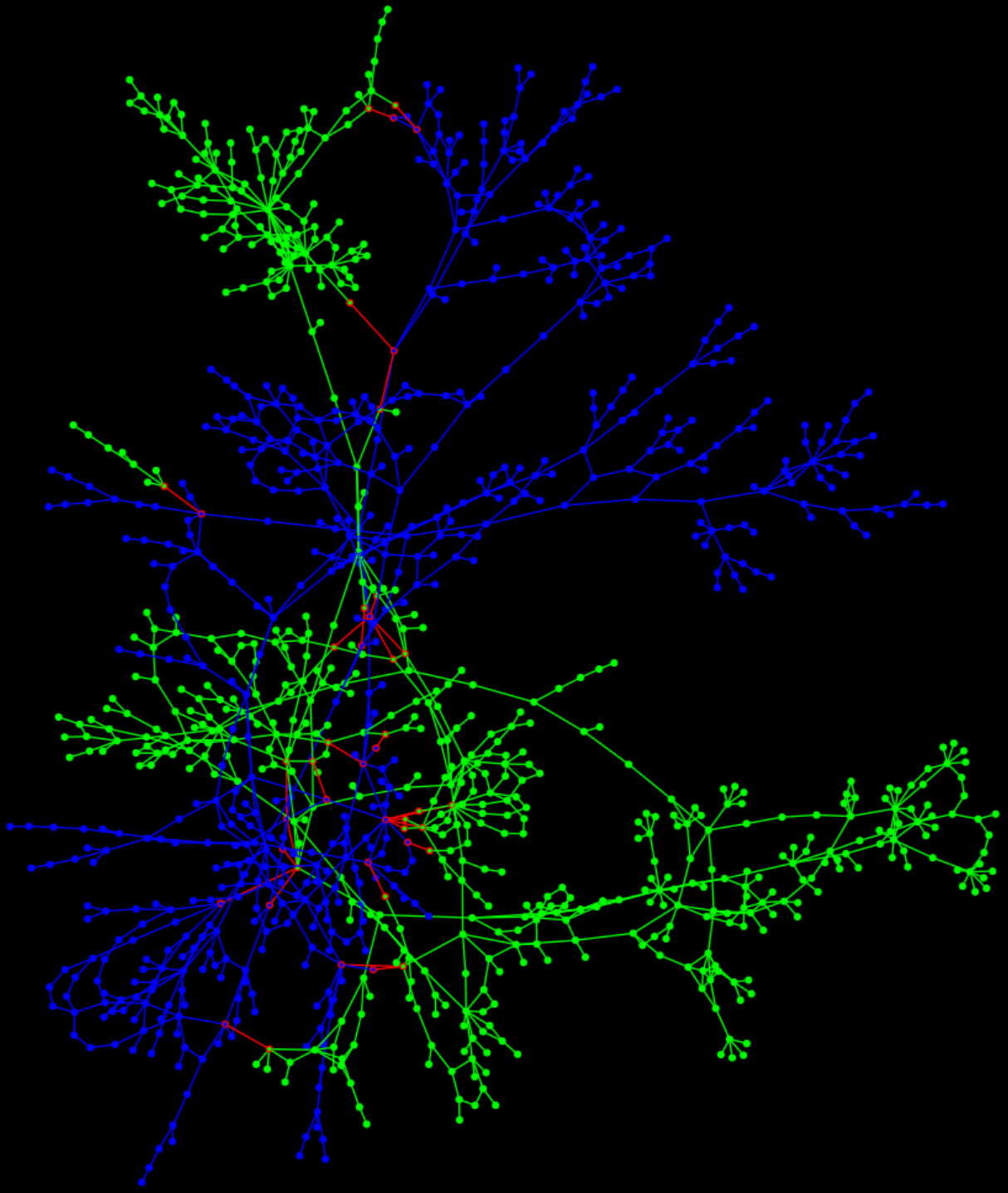


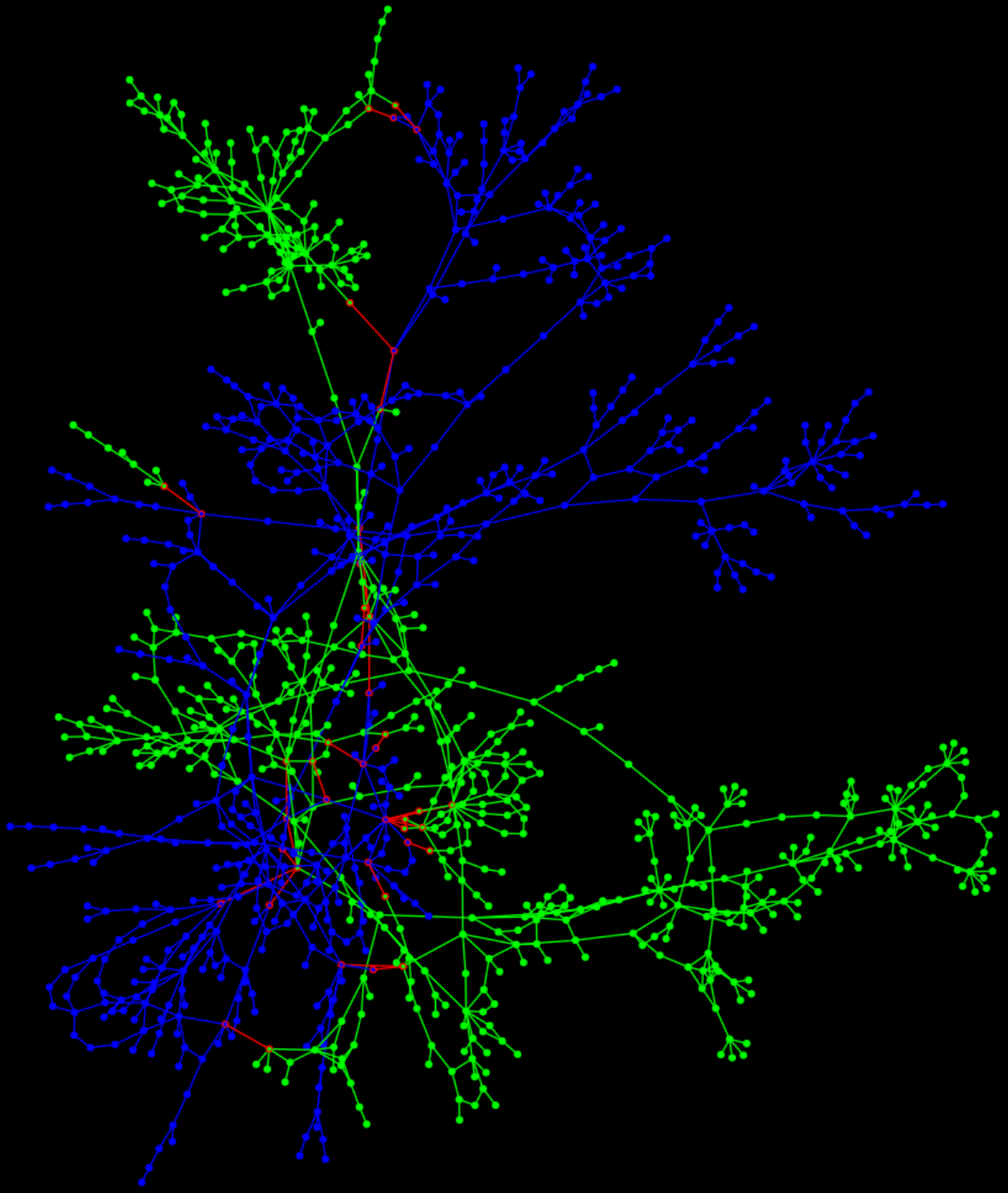


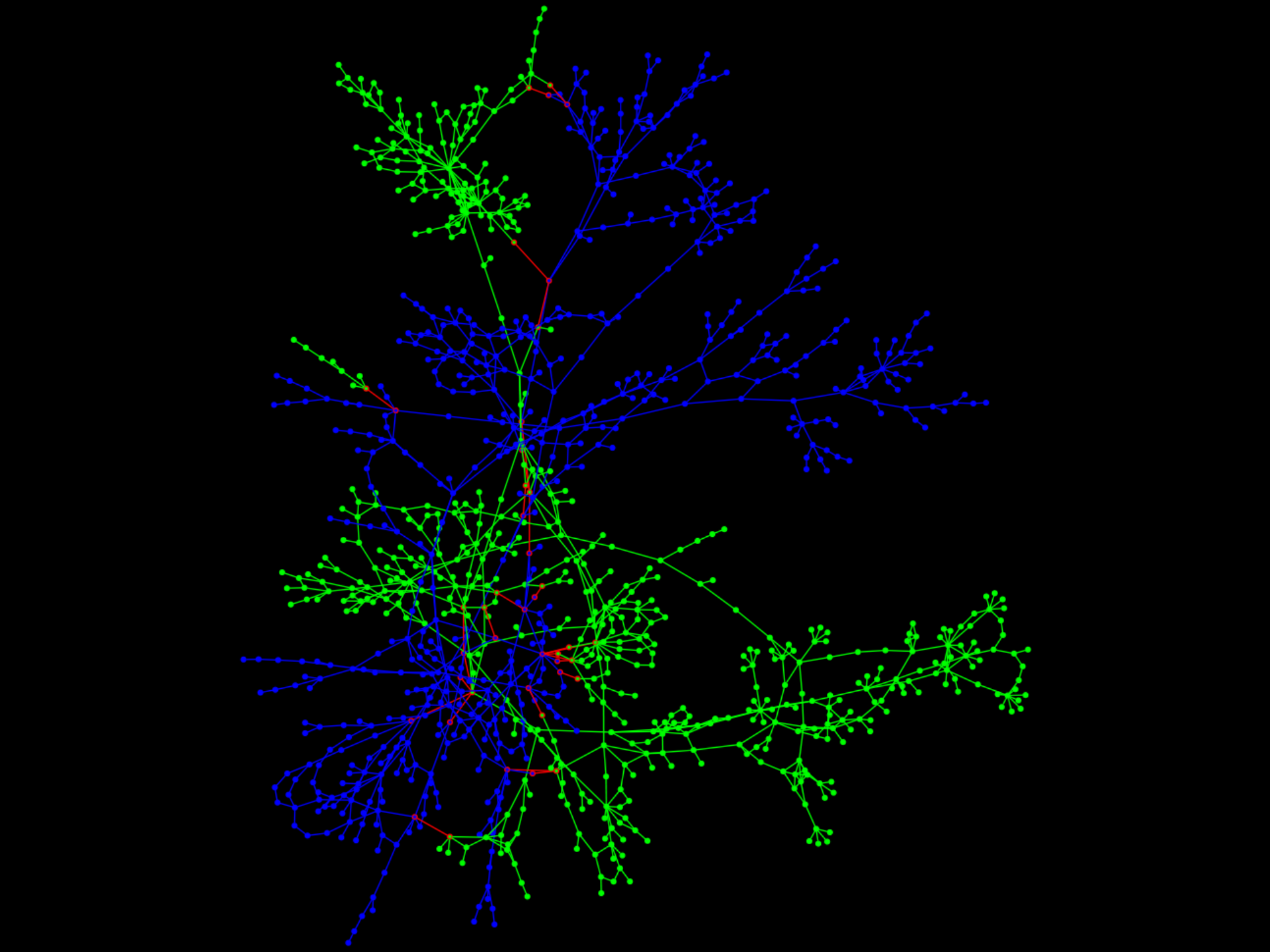


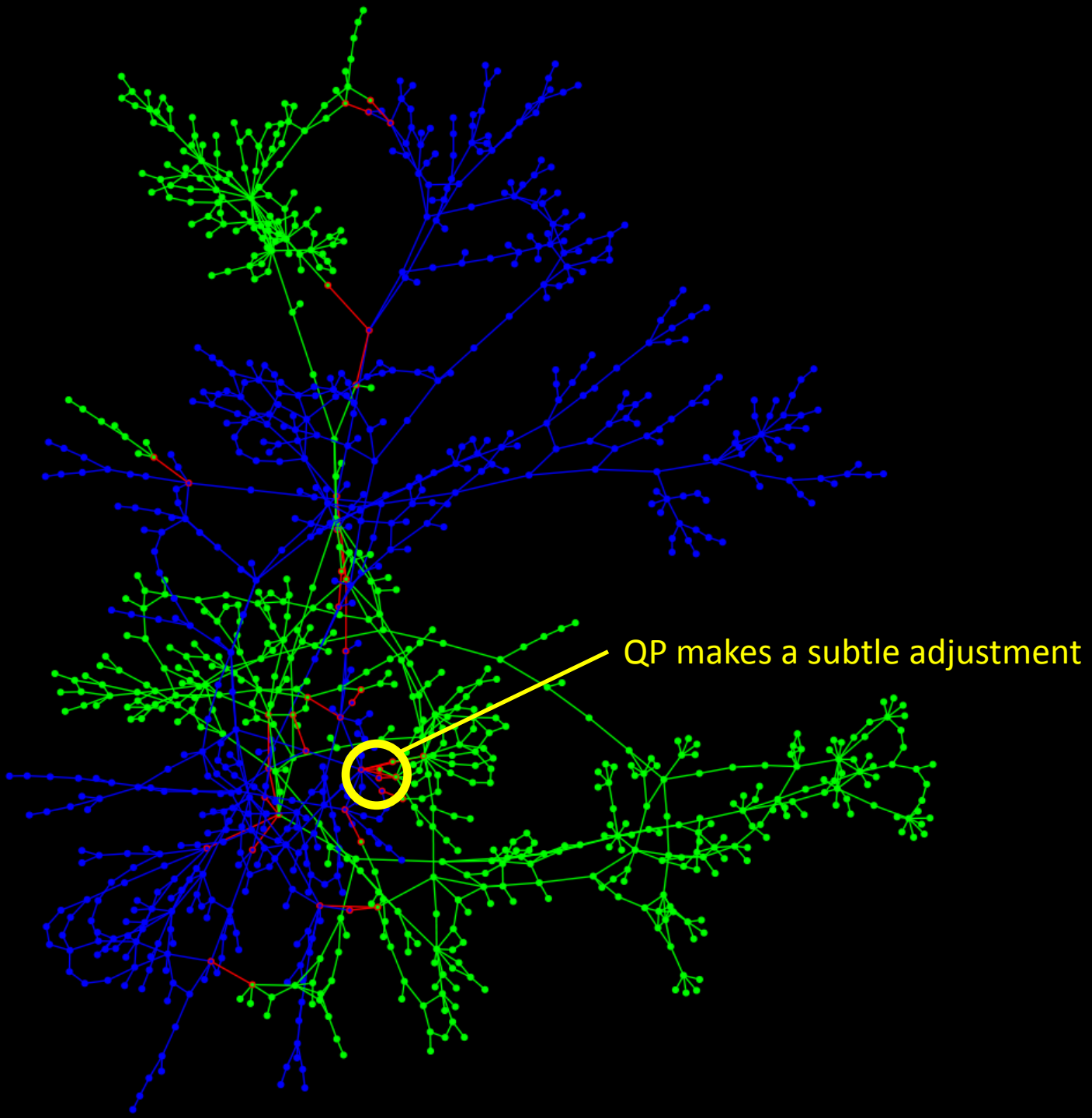



QP makes a subtle adjustment







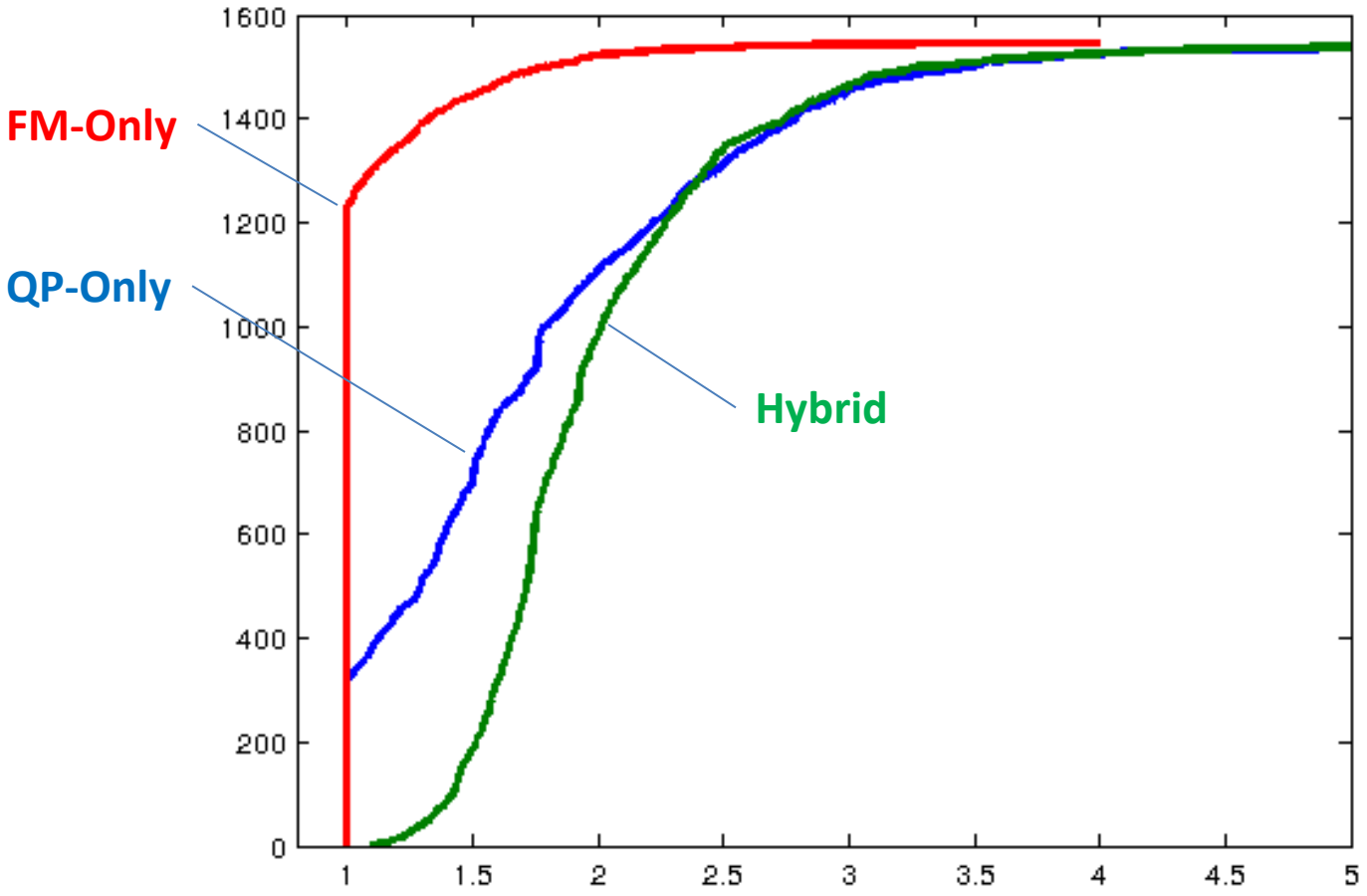

QP makes a subtle adjustment

Results: Set 1, Hodgepodge

- 1550 square matrices
- Between 15 and 2mil edges
- Come from a variety of contexts
  - Optimization, fluid dynamics, quantum mechanics, PDE, circuit simulation, power line networks, 2d & 3d mesh, etc

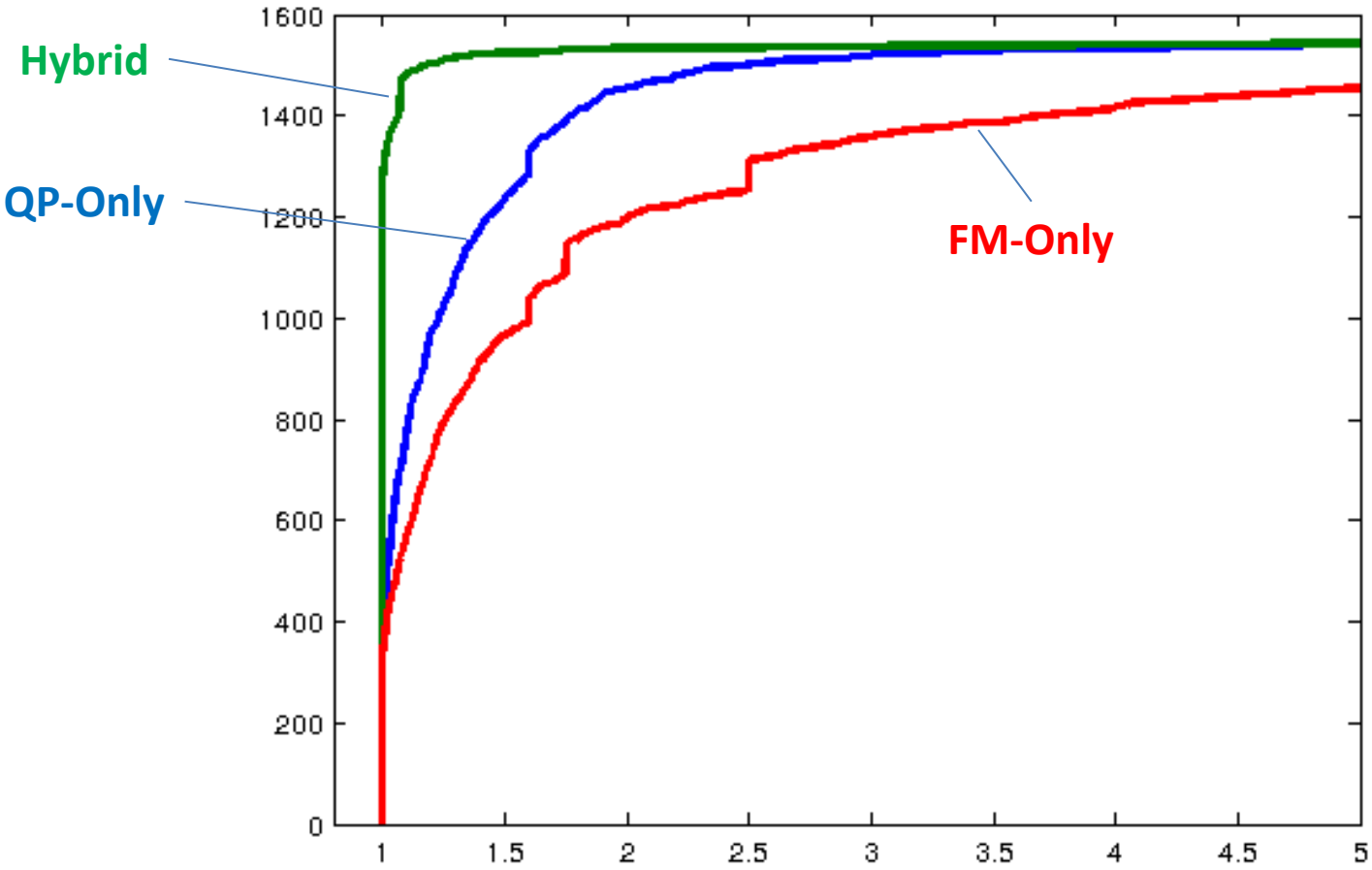
Profile: Timing

1550 General Problems: FM, QP, Hybrid



Profile: Cut Quality

1550 General Problems: FM, QP, Hybrid

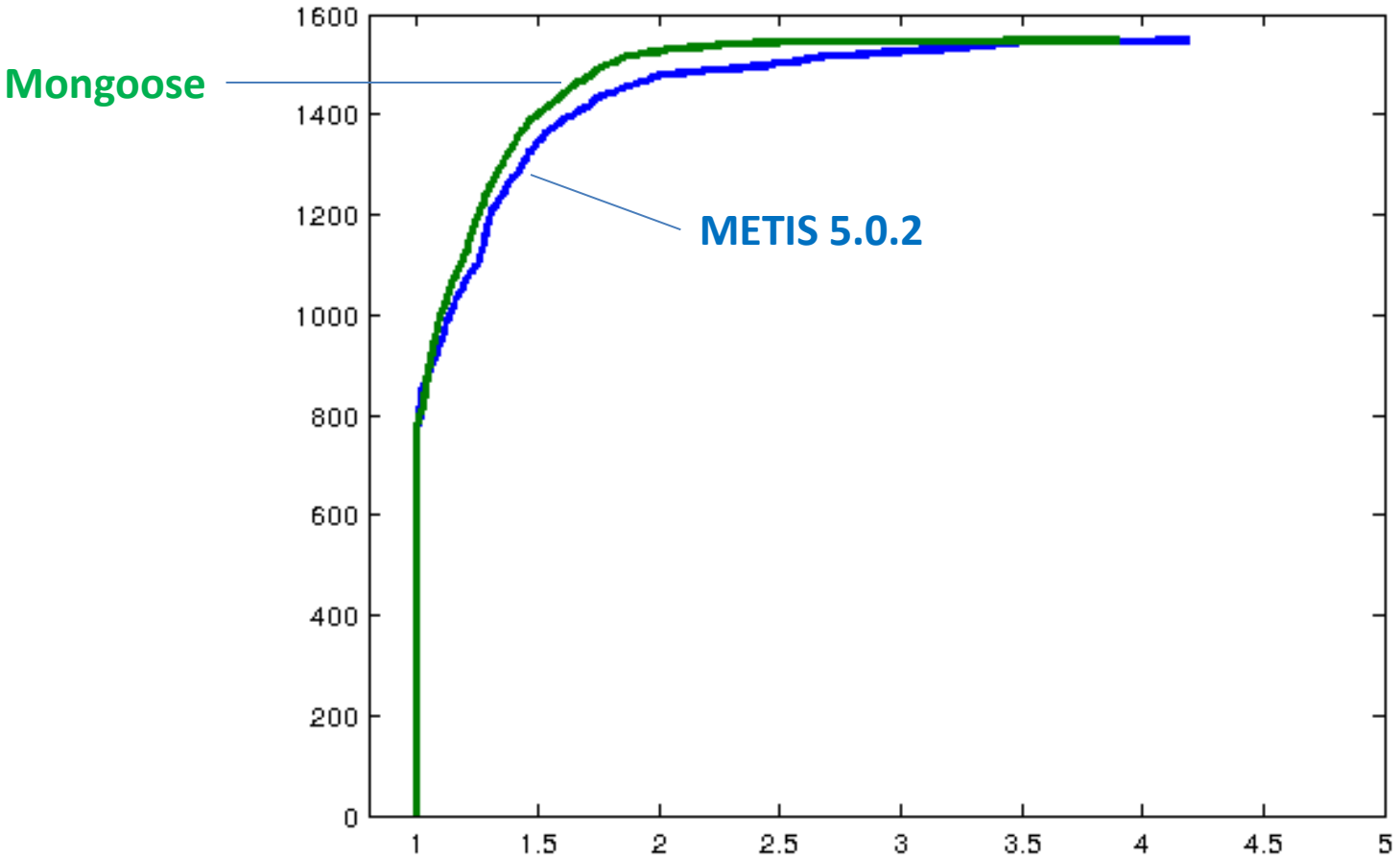


Remarks

- The quadratic programming formulation seems to work well in a multi-level setting.
- Combining the combinatorial method with the quadratic programming formulation is superior to either by itself.
- How well does Mongoose perform against contemporary partitioners?
  - METIS 5.0.2

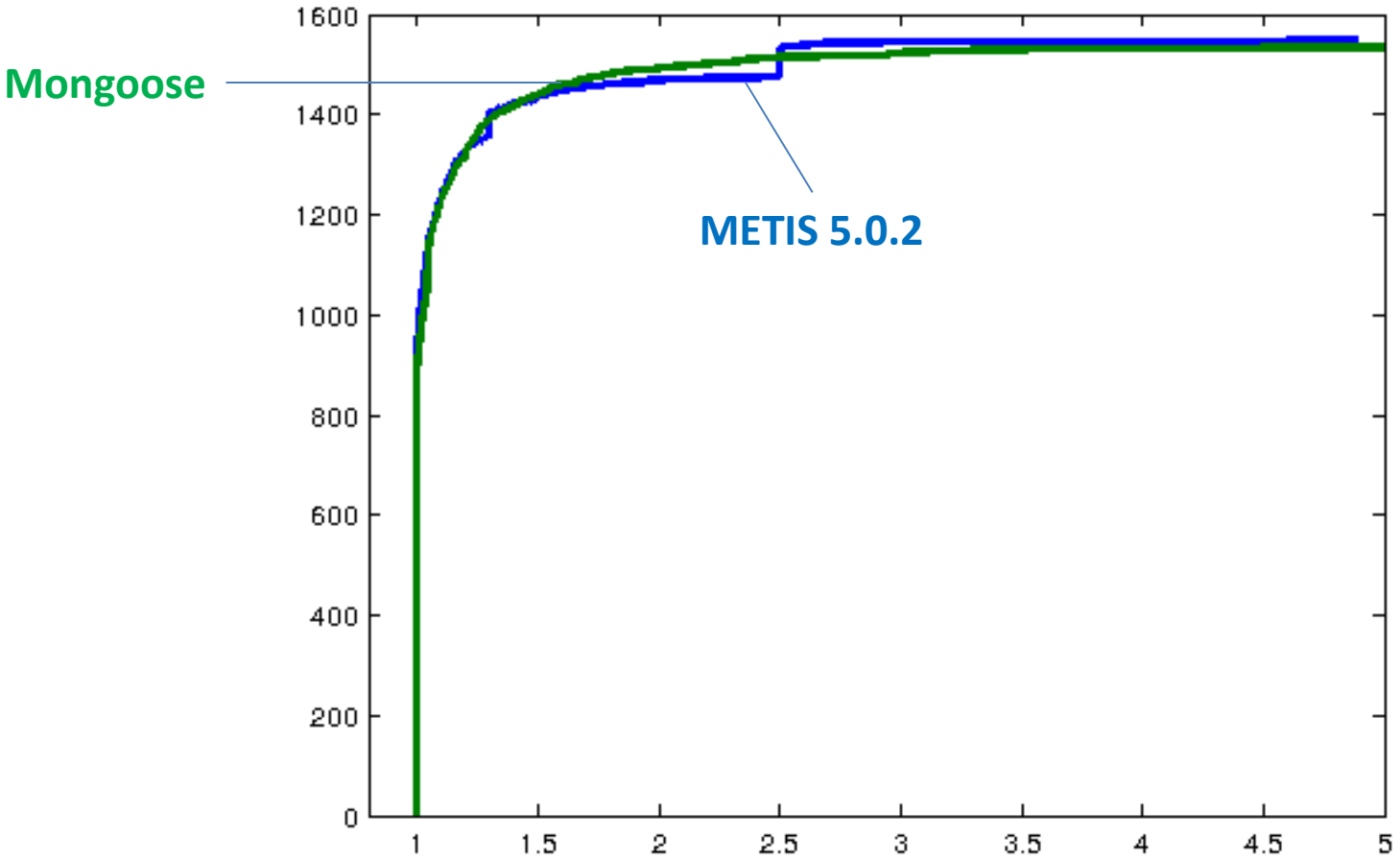
Profile: Timing

1550 General Problems: METIS 5, Mongoose



Profile: Cut Quality

1550 General Problems: METIS 5, Mongoose



Remarks

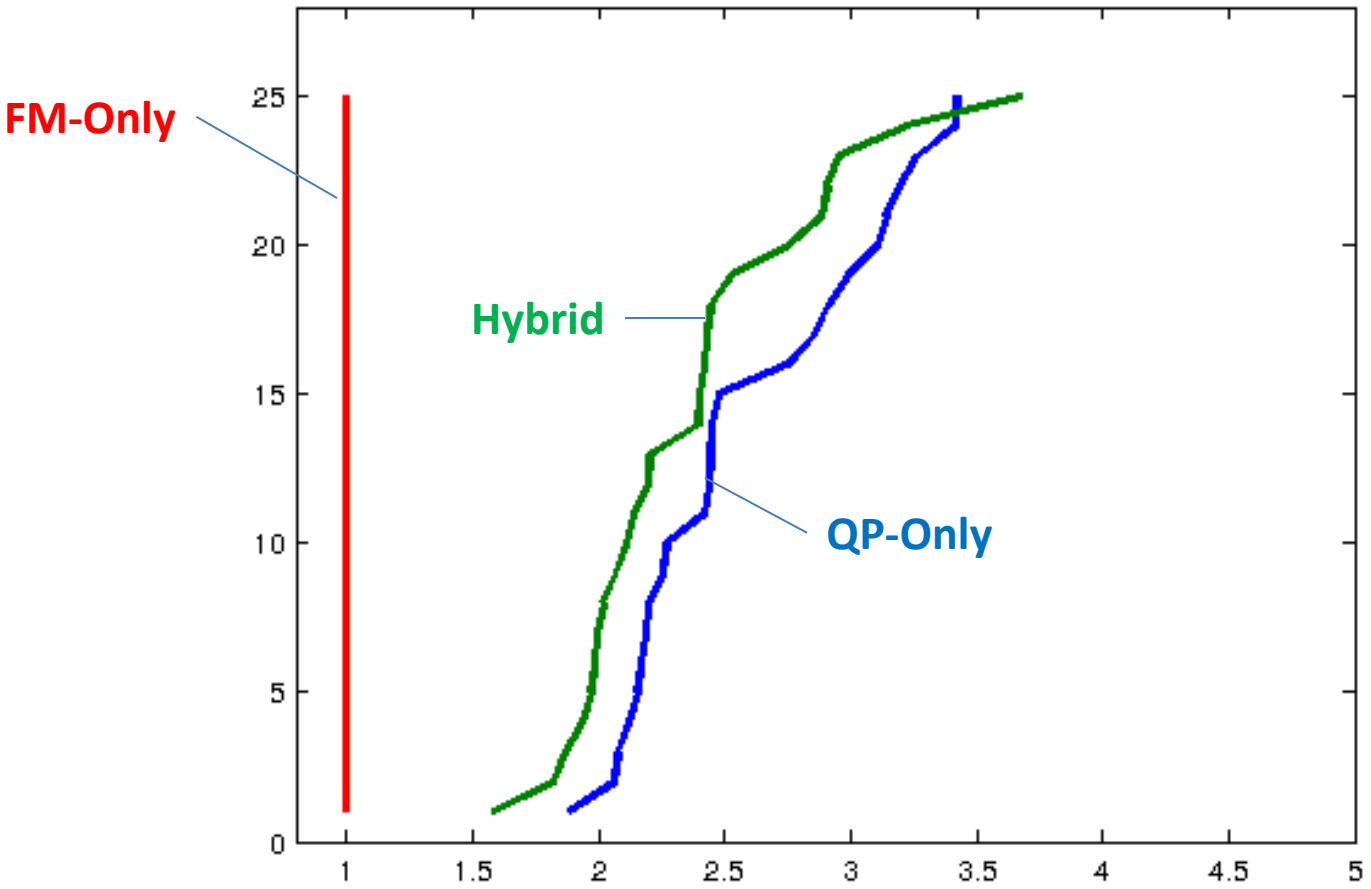
- In the general case, Mongoose seems to perform on par with METIS 5.0.2 in both performance and cut quality.

Results: Set 2, Power Law Graphs

- 25 square matrices
- Between 1mil and 10 mil edges
- Come from social networking, web networks

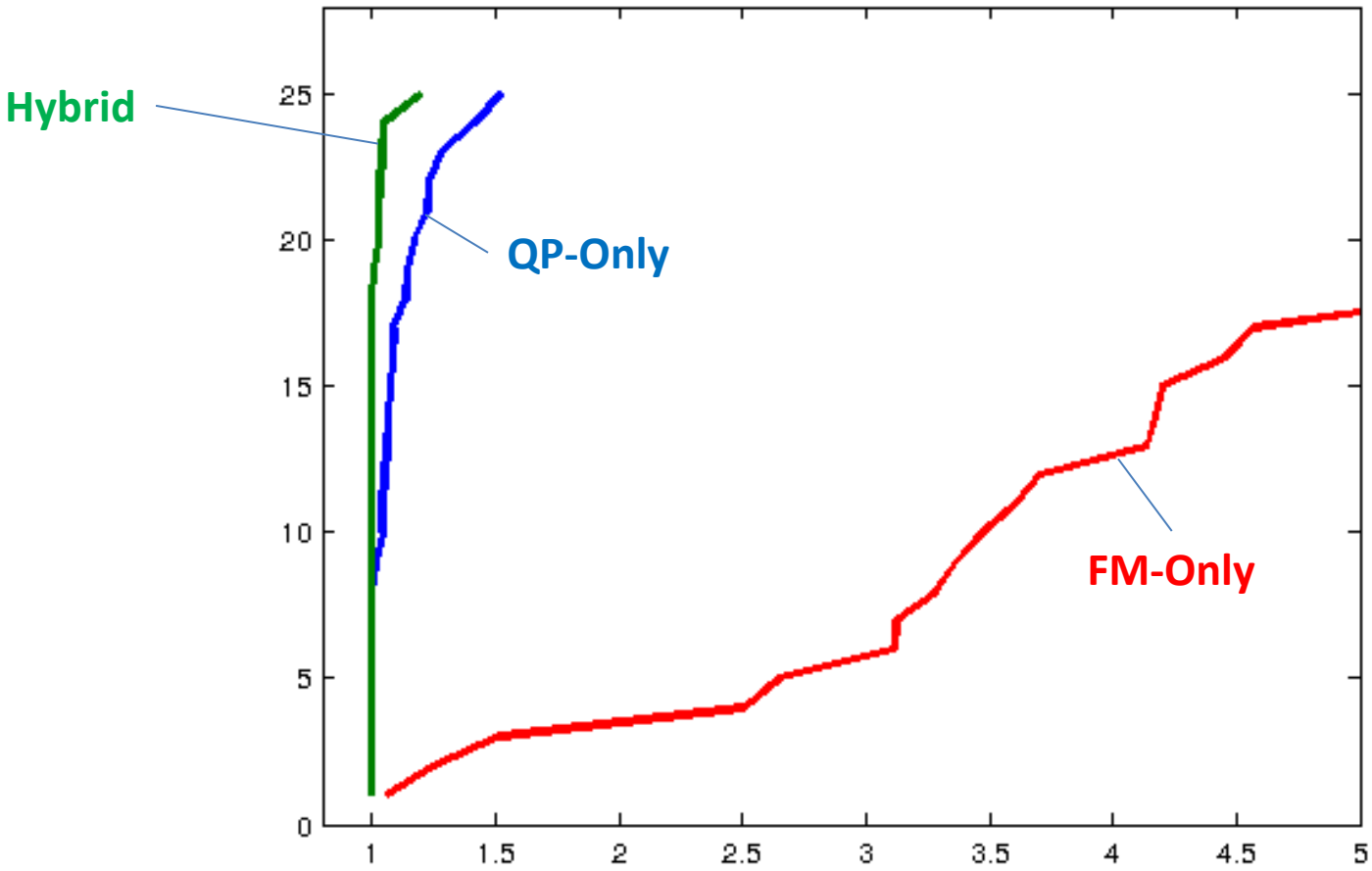
Profile: Timing

25 Power Law Problems: FM, QP, Hybrid



Profile: Cut Quality

25 Power Law Problems: FM, QP, Hybrid

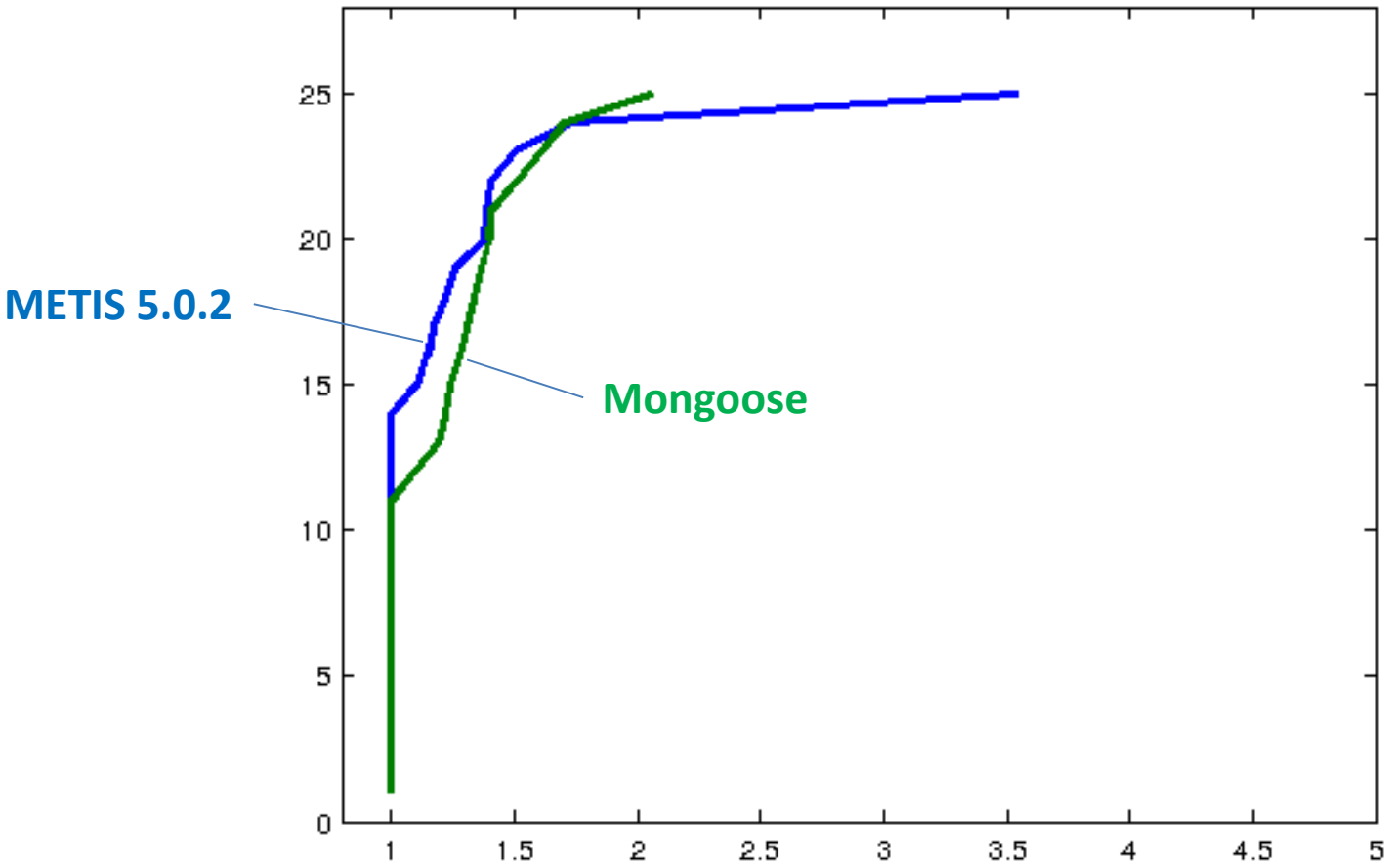


Remarks

- Providing gradient projection with an good guess via boundary-FM reduces the overall cost of the hybrid approach
  - Synergy – FM and QP mutually help each other
- Hybrid approach tends to find better cuts
- How well does Mongoose perform against contemporary partitioners?
  - METIS 5.0.2

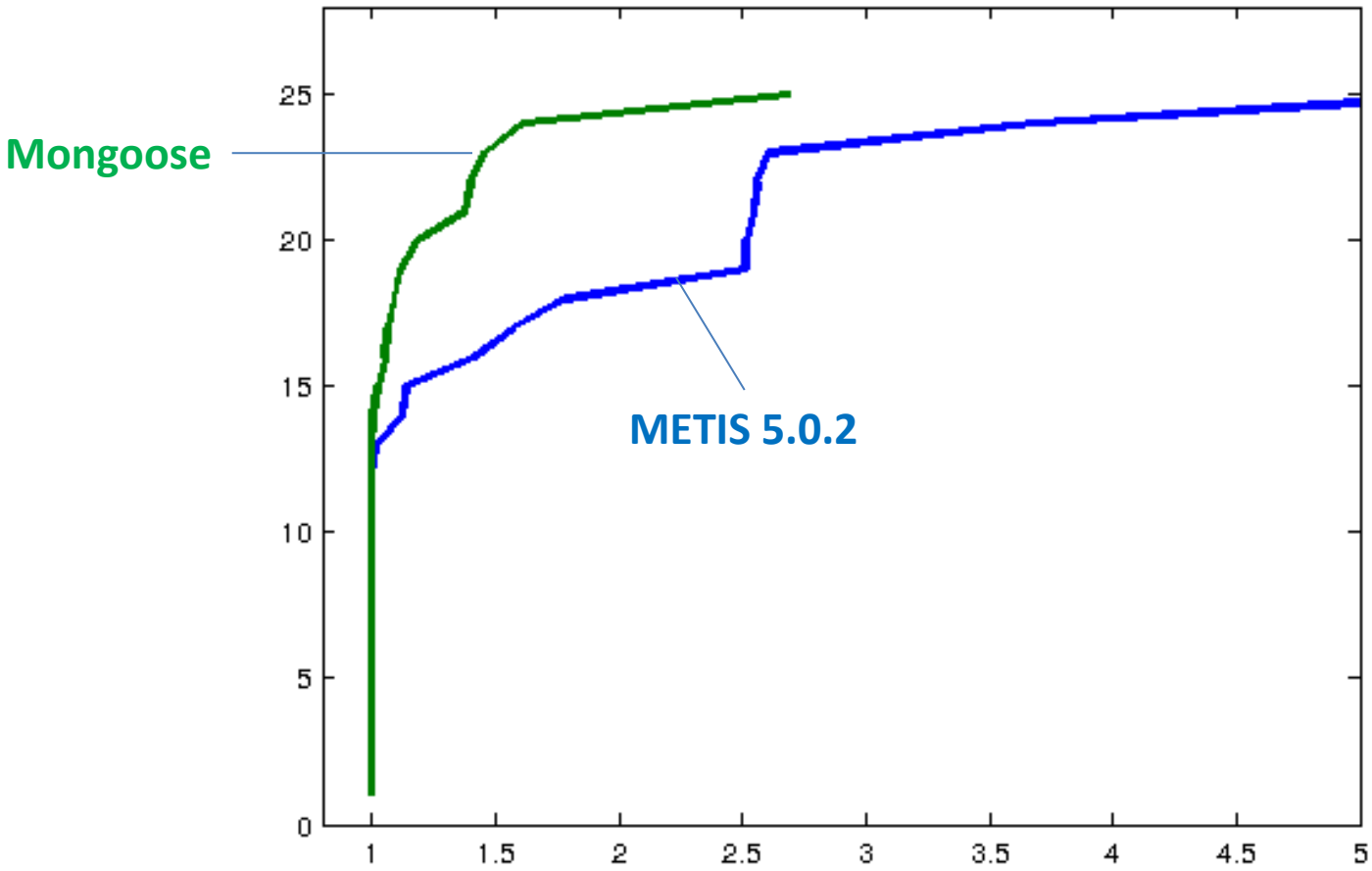
Profile: Timing

25 Power Law Problems: METIS 5, Mongoose



Profile: Cut Quality

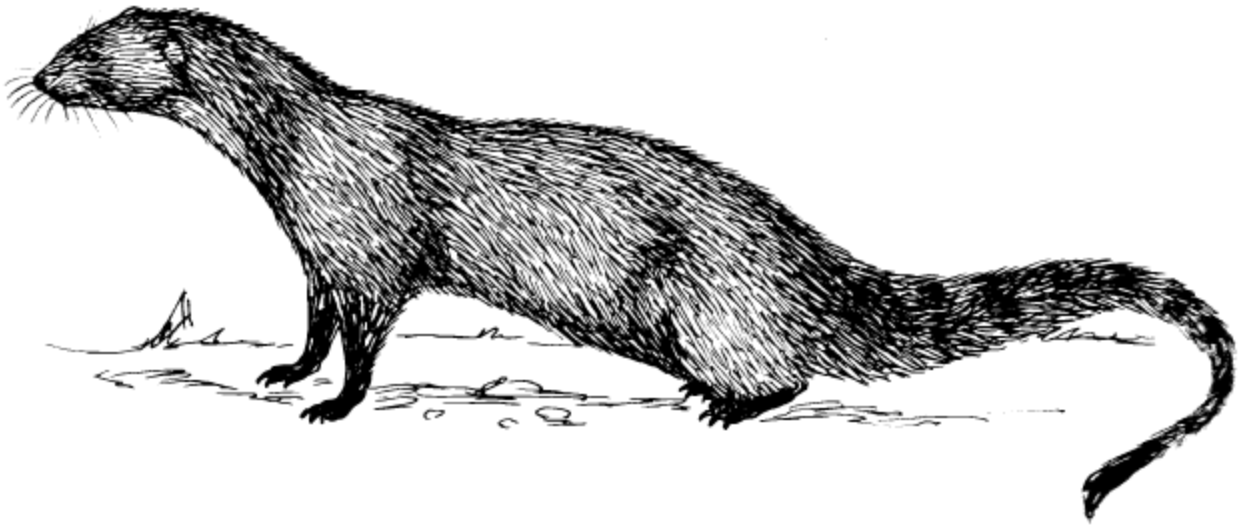
25 Power Law Problems: METIS 5, Mongoose



Remarks

- Mongoose tends to find better cuts than METIS 5.0.2 for large graphs.
- Mongoose ties METIS 5.0.2 for time.
  - Future Work
    - Optimize quadratic programming formulation
    - Use randomization and parallelization

Questions?



openclipart.org