Revisiting (multi)core-periphery network structures: a machine learning approach

Amit Gal

Background & Motivation

- Core-Periphery (CP) structures involve partitioning the network into subgroups - but unlike standard clustering/community detection, these subgroups has inherent relationships. They are not equal.
- These structures seem to appear a lot in social networks, and are consequential to their functioning.
- Detecting/Measuring CP is difficult.
  - Some measures exist, but E&B(2000) dominates the field
    - It has computational limitations
    - Its conceptual underpinnings are somewhat misspecified
    - It does not support detecting multiple cores
    - Other proposed measures do not always address these concerns
- The purpose of this research is to improve this situation :-)

Detecting/measuring CP structures

- E&B's measure is based on an idealized block model
- The measure is based on a correlation between the idealized model and the actual network
- BUT - to find an optimal solution is computationally hard
- So, we relaxed a bit the assumptions and instead of partitioning the graph we give a “coreness” score for each node.
  - Note that this transfers our discussion from a whole network perspective, to a property of individuals.
  - Abandoning the block model formulation makes multiple cores impossible with this approach

CP and block modeling

- Like most community detection, we vision the core as a dense subgroup with sparse connections to the outside
- What kind of a block is the periphery?
  - We allow the periphery to be connected to the core
  - We assume the periphery is relatively sparse (internally)
  - There is some confusion regarding which of the above is more important and what are the consequences for detecting CP
    - compare with the various centrality formulations and measures
- The above presents the CP as a generalized block modeling problem.

Multiple cores

- With the formulation given above, we can easily envision multiple cores, and even multiple peripheries
  - Does each core has its own periphery?
  - or is it one periphery for all?
  - Does a node must be in a core or a periphery?
  - Is a node that is “in between” cores, a periphery?

What machine learning can do?

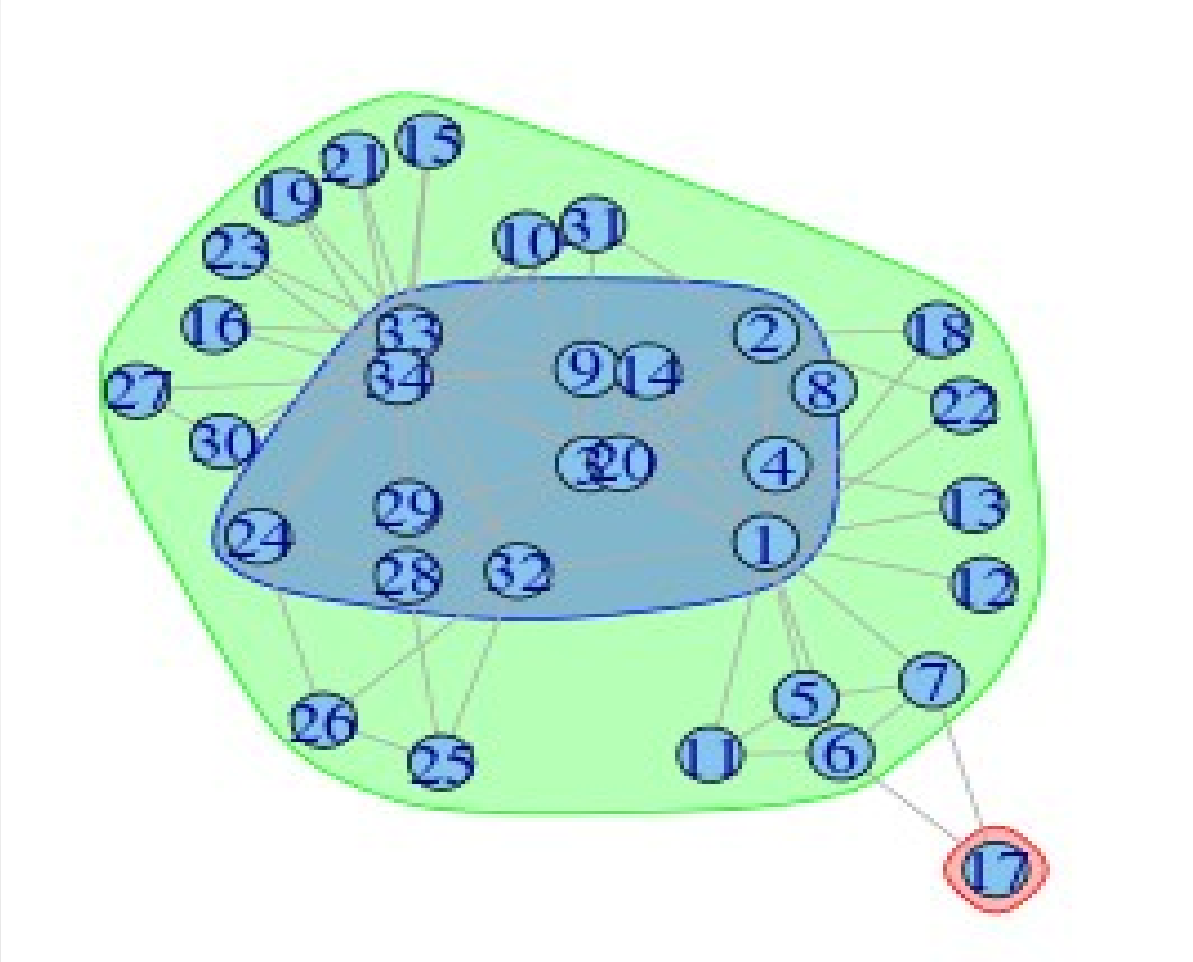
- Machine learning algorithms deal with detecting structure in data.
- There are lots of algorithms that deal with clustering data. In this presentation I will present 3 such approaches:
  - dbscan – a distance based algorithm that inherently detects a boundary for each cluster.
  - Clustering with noise – treating the peripheral nodes as noise to clustering.
  - Learning from random partitions to infer the optimal partition (a heuristic approach)

The DBSCAN algorithm

- Description:
  - The algorithm “grows” clusters, starting initial nodes that have very close neighborhood, then expand them to near nodes, until there are none.
  - Those nodes from which the algorithm cannot go further are the “boundary”
  - Main proposition: the boundary is the periphery.
- Main properties
  - Optimizes more on core-periphery connectivity, and less on peripheral sparsity
  - Has a periphery for each core
  - Allows non-periphery, non-core nodes (true noise!)
  - Supports multiple cores
- Disadvantage
  - No differentiation between peripheral and bridge nodes
  - Very sensitive to parameters.

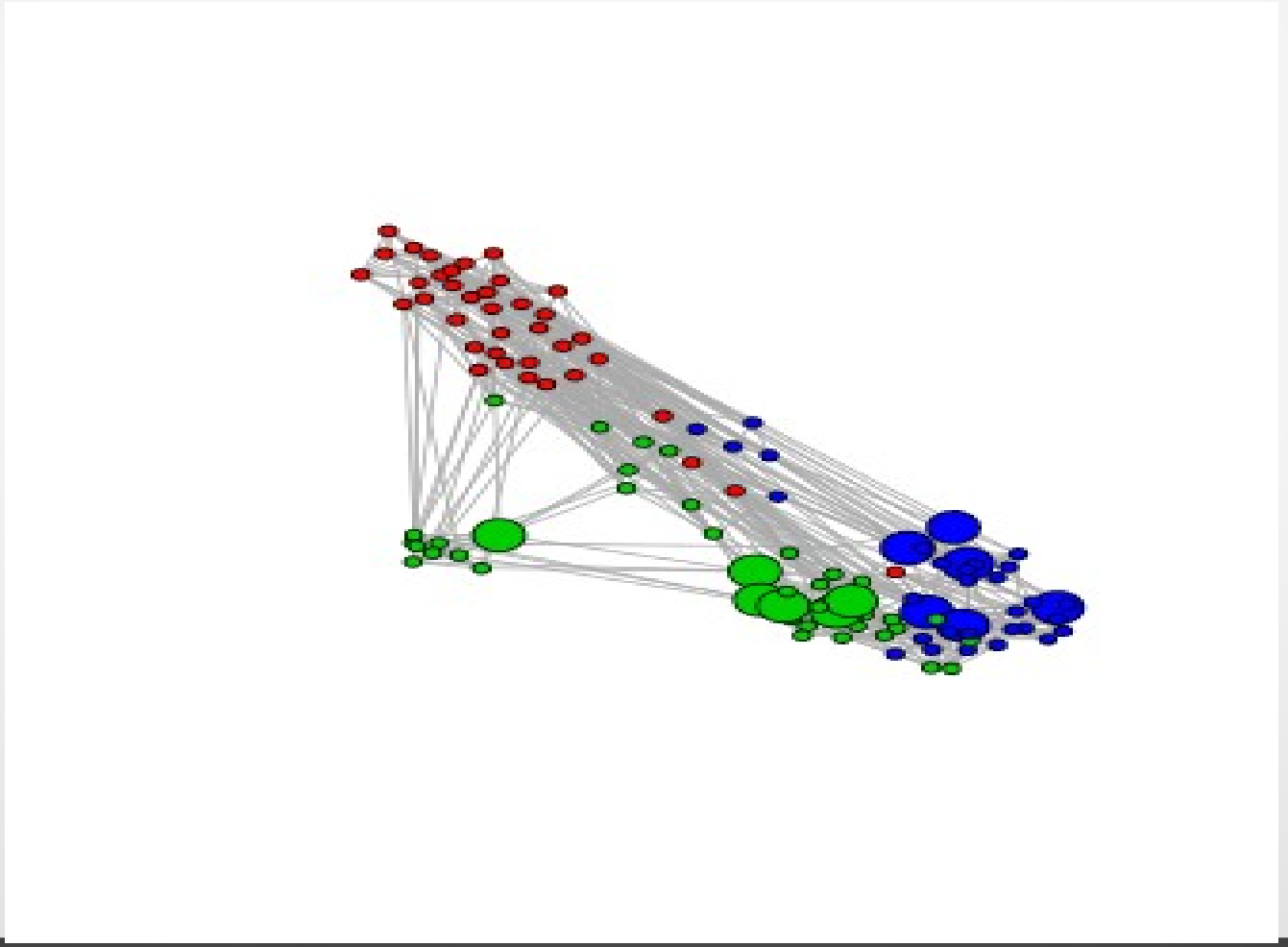
An example:

- The karate network (Zachary, 1977)



Another example:

- Jazz musicians interactions (P.Gleiser and L. Danon , Adv. Complex Syst.6, 565 (2003))

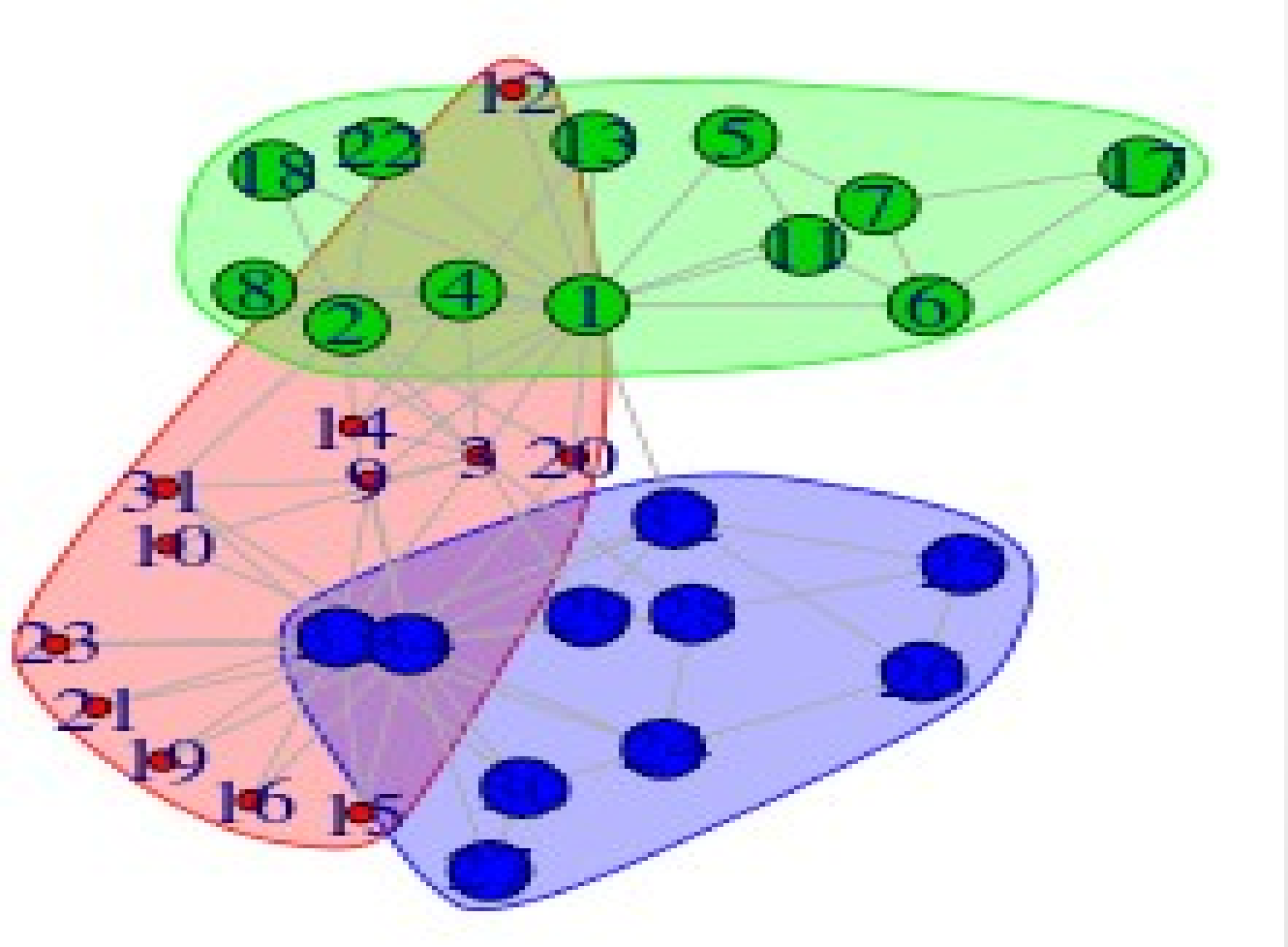


Clustering with dropouts

- Description:
  - Performs standard clustering, but allow marking nodes as “do not know”, that will eventaully become the periphery
  - At each step, it selects the nodes, that if dropped will best improve the clustering “score”
  - Repeats until no improvements can be done.
- Main properties:
  - emphasizes peripheral sparsity (rather than cp-relations)
  - one periphery for all
  - Allows multiple cores
  - Can work with every underlying clustering and various scoring schemas
- Disadvantages
  - It is an approximation, due to the stepwise dropping scheme.

An example:

- Karate network:

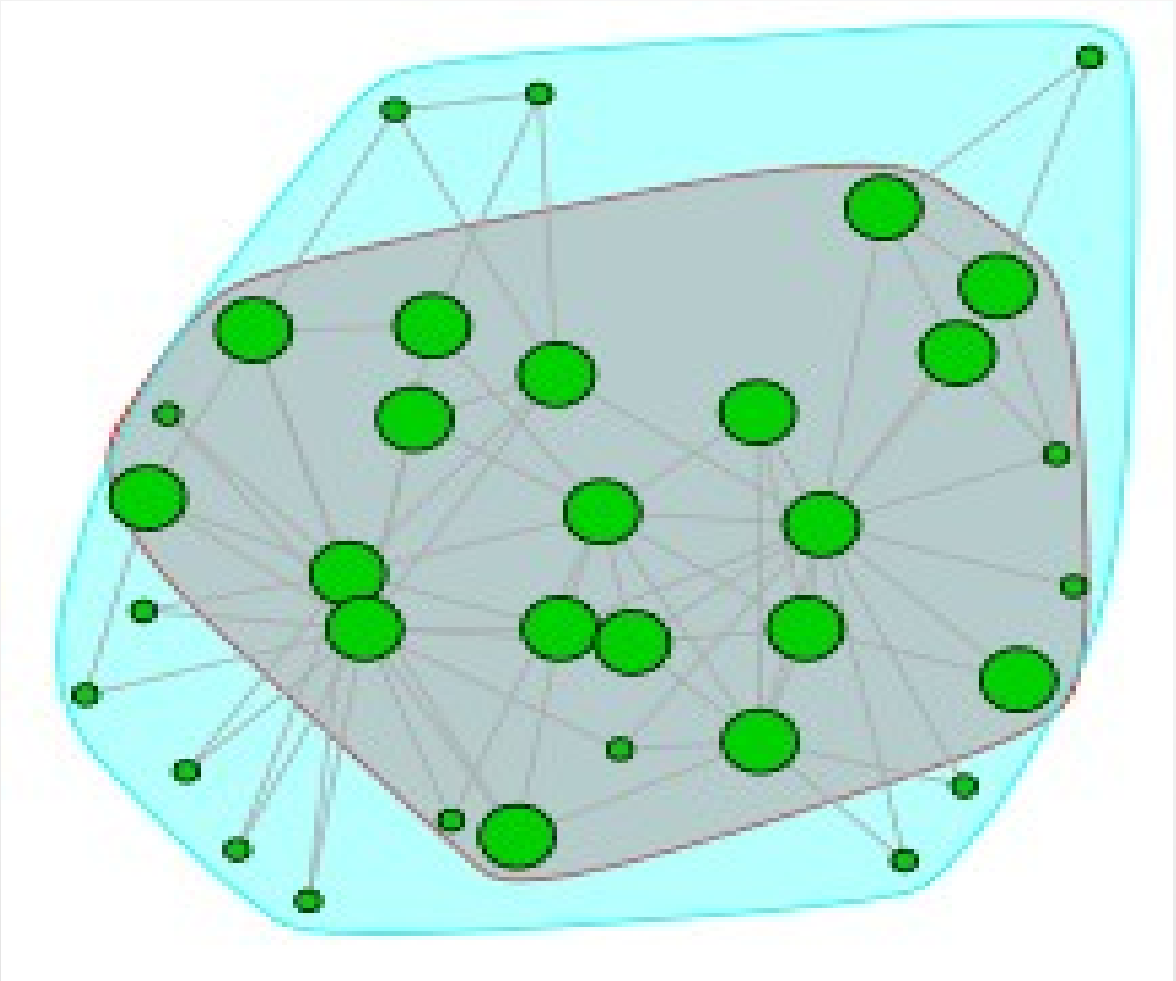


Inferrence from random partitions

- Description:
  - Creates a large set of random partitions of CP
  - Evaluates each partition using a “goodness of fit” score
  - Infers a CP partition from the above using an AUC-like criterion.
- Properties:
  - Can handle only single core and single periphery
  - Works extremely fast
  - Can work with any underlying scoring scheme, so can emphasize any property of the periphery or its relation to the core.
- Disadvantages
  - It is only a heuristics (but it has around 85% agreement with other methods)

Example:

- Karate network
-



Thank you!

Questions?

amitgal4@gmail.com