Introduction to Community Detection in Networks via Modularity Maximization

Cason Konzer

casonk@umich.edu

December 4, 2022

Introduction

What exactly is a network?

Introduction

What exactly is a network?

- A network is a group of objects in which we call nodes.

Introduction

What exactly is a network?

- A network is a group of objects in which we call nodes.
  - for example web pages, authors, and transit hubs.

Introduction

What exactly is a network?

- A network is a group of objects in which we call nodes.
  - for example web pages, authors, and transit hubs.
  - the objects may have attributes such as age, expertise, and city.

Introduction

What exactly is a network?

- A network is a group of objects in which we call nodes.
  - for example web pages, authors, and transit hubs.
  - the objects may have attributes such as age, expertise, and city.
- Objects are then connected by edges which represent node relations.

Introduction

What exactly is a network?

- A network is a group of objects in which we call nodes.
  - for example web pages, authors, and transit hubs.
  - the objects may have attributes such as age, expertise, and city.
- Objects are then connected by edges which represent node relations.
  - for example references, and transit paths.

Introduction

What exactly is a network?

- A network is a group of objects in which we call nodes.
  - for example web pages, authors, and transit hubs.
  - the objects may have attributes such as age, expertise, and city.
- Objects are then connected by edges which represent node relations.
  - for example references, and transit paths.
  - the connections may also have attributes such as hyperlink, road, and shipping lane.

Example: Web Pages

Contact

Mailing list
Issue tracker
Source

Releases

Stable (notes)

2.8.8 — November 2022
download | doc | pdf

Latest (notes)

3.0 development
github | doc | pdf

Archive

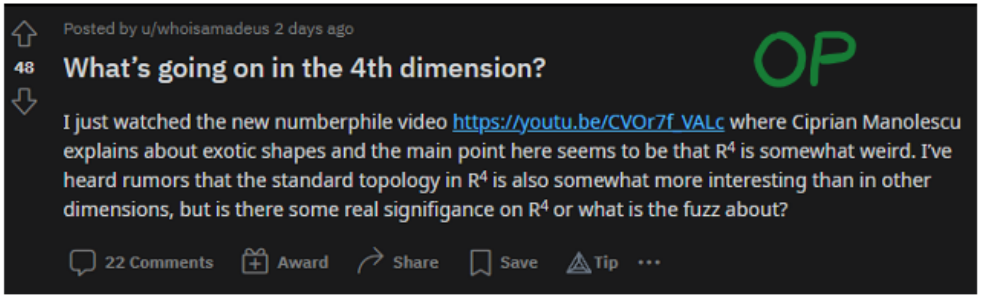
NetworkX
Network Analysis in Python

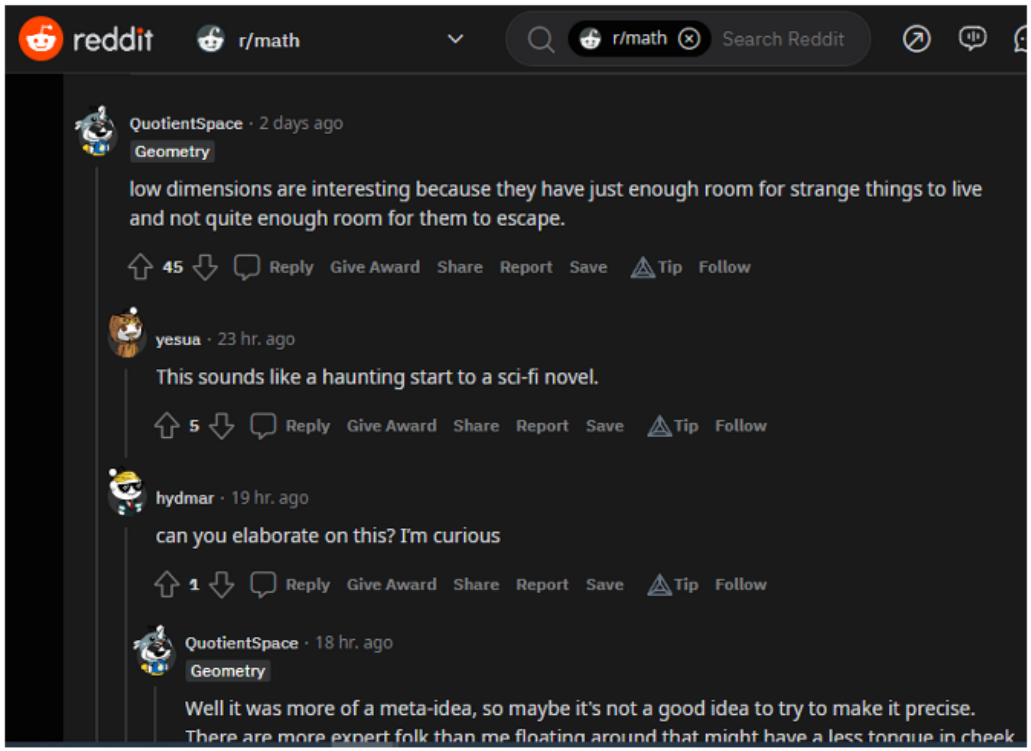
NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

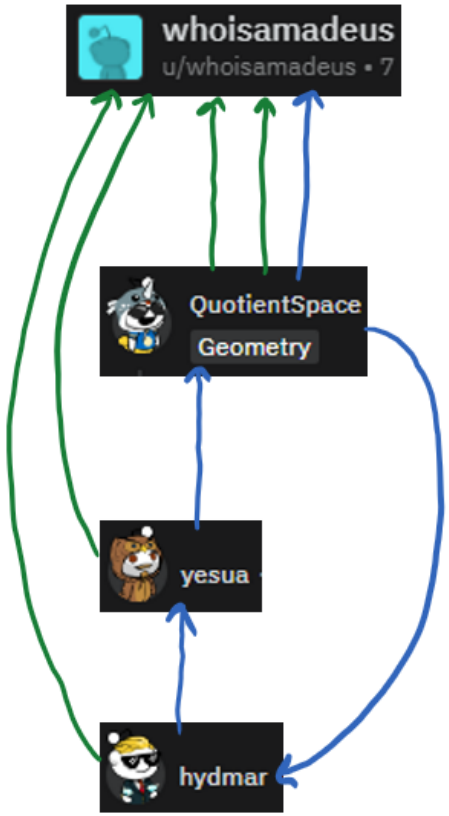
Software for complex networks

- Data structures for graphs, digraphs, and multigraphs
- Many standard graph algorithms
- Network structure and analysis measures

Example: Social Media







Example: Transit

Communities

What is a community within a network?

Communities

What is a community within a network?

- Communities are partitions of the network.

Communities

What is a community within a network?

- Communities are partitions of the network.
  - every node should fall into at least one community.

Communities

What is a community within a network?

- Communities are partitions of the network.
  - every node should fall into at least one community.
  - communities may be disjoint, hierarchical, or overlapping.

Communities

What is a community within a network?

- Communities are partitions of the network.
  - every node should fall into at least one community.
  - communities may be disjoint, hierarchical, or overlapping.
    - think of sets.

Communities

What is a community within a network?

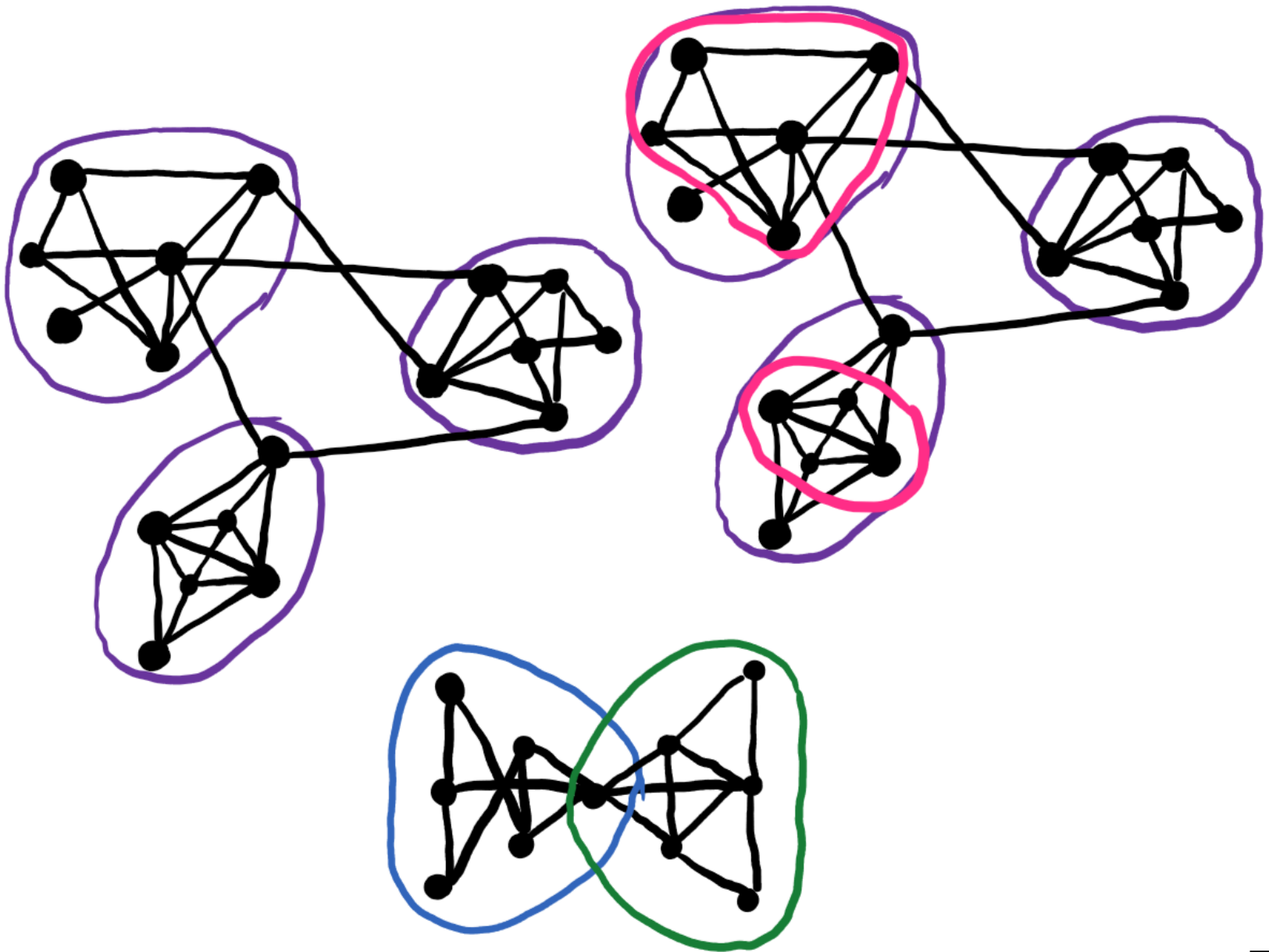
- Communities are partitions of the network.
  - every node should fall into at least one community.
  - communities may be disjoint, hierarchical, or overlapping.
    - think of sets.
- We expect nodes within a community to share similar attributes.

Communities

What is a community within a network?

- Communities are partitions of the network.
  - every node should fall into at least one community.
  - communities may be disjoint, hierarchical, or overlapping.
    - think of sets.
- We expect nodes within a community to share similar attributes.
  - often communities are considered to have a high density of internal edges compared to that of external edges.

Community Examples



Motivation

Why study communities in networks?

Motivation

Why study communities in networks?

- Community detection has many applications.

Motivation

Why study communities in networks?

- Community detection has many applications.
  - often it can be used for recommendation purposes.

Motivation

Why study communities in networks?

- Community detection has many applications.
  - often it can be used for recommendation purposes.
    - finding web pages with similar topics.

Motivation

Why study communities in networks?

- Community detection has many applications.
  - often it can be used for recommendation purposes.
    - finding web pages with similar topics.
    - finding friends within your social circle.

Motivation

Why study communities in networks?

- Community detection has many applications.
  - often it can be used for recommendation purposes.
    - finding web pages with similar topics.
    - finding friends within your social circle.
    - finding groups of bad actors.

Motivation

Why study communities in networks?

- Community detection has many applications.
  - often it can be used for recommendation purposes.
    - finding web pages with similar topics.
    - finding friends within your social circle.
    - finding groups of bad actors.
    - finding products the community as a whole may like.

Motivation

Why study communities in networks?

- Community detection has many applications.
  - often it can be used for recommendation purposes.
    - finding web pages with similar topics.
    - finding friends within your social circle.
    - finding groups of bad actors.
    - finding products the community as a whole may like.
  - community connecting edges may be of concern.

Motivation

Why study communities in networks?

- Community detection has many applications.
  - often it can be used for recommendation purposes.
    - finding web pages with similar topics.
    - finding friends within your social circle.
    - finding groups of bad actors.
    - finding products the community as a whole may like.
  - community connecting edges may be of concern.
    - finding roads in need of additional lanes.

Motivation

Why study communities in networks?

- Community detection has many applications.
  - often it can be used for recommendation purposes.
    - finding web pages with similar topics.
    - finding friends within your social circle.
    - finding groups of bad actors.
    - finding products the community as a whole may like.
  - community connecting edges may be of concern.
    - finding roads in need of additional lanes.
    - finding authors with cross domain knowledge.

Motivation

Why study communities in networks?

- Community detection has many applications.
  - often it can be used for recommendation purposes.
    - finding web pages with similar topics.
    - finding friends within your social circle.
    - finding groups of bad actors.
    - finding products the community as a whole may like.
  - community connecting edges may be of concern.
    - finding roads in need of additional lanes.
    - finding authors with cross domain knowledge.
  - etc.

Methods

There are many methods in which one can find communities. Some examples include the following.

Methods

There are many methods in which one can find communities. Some examples include the following.

- Shortest Path Betweeness.

Methods

There are many methods in which one can find communities. Some examples include the following.

- Shortest Path Betweeness.
- Random Walk Betweeness.

Methods

There are many methods in which one can find communities. Some examples include the following.

- Shortest Path Betweeness.
- Random Walk Betweeness.
- Modularity Maximization.

Methods

There are many methods in which one can find communities. Some examples include the following.

- Shortest Path Betweeness.
- Random Walk Betweeness.
- Modularity Maximization.
- Maximum Liklihood Estimation.

Method Evaluation

How might we choose a method in application? In general we must consider two criteria.

Method Evaluation

How might we choose a method in application? In general we must consider two criteria.

- Accuracy.

Method Evaluation

How might we choose a method in application? In general we must consider two criteria.

- Accuracy.
  - How close are detected communities to their ground truth?

Method Evaluation

How might we choose a method in application? In general we must consider two criteria.

- Accuracy.
  - How close are detected communities to their ground truth?
- Complexity.

Method Evaluation

How might we choose a method in application? In general we must consider two criteria.

- Accuracy.
  - How close are detected communities to their ground truth?
- Complexity.
  - How long does it take to detect communities?

Method Evaluation

How might we choose a method in application? In general we must consider two criteria.

- Accuracy.
  - How close are detected communities to their ground truth?
- Complexity.
  - How long does it take to detect communities?
    - How big of a network can the method run on?

Method Evaluation

How might we choose a method in application? In general we must consider two criteria.

- Accuracy.
  - How close are detected communities to their ground truth?
- Complexity.
  - How long does it take to detect communities?
    - How big of a network can the method run on?

Today we will focus on Modularity Maximization methods.

Nomenclature

Let us define basic terminology.

Nomenclature

Let us define basic terminology.

- Let n be the number of nodes in the network.

Nomenclature

Let us define basic terminology.

- Let n be the number of nodes in the network.
- Let m be the number of edges in the network.

Nomenclature

Let us define basic terminology.

- Let n be the number of nodes in the network.
- Let m be the number of edges in the network.
- Let $s = 2m$ be the number of spokes (edge ends) in the network.

Nomenclature

Let us define basic terminology.

- Let n be the number of nodes in the network.
- Let m be the number of edges in the network.
- Let $s = 2m$ be the number of spokes (edge ends) in the network.
- Let k_i be the number of spokes incident upon a node i.

Nomenclature

Let us define basic terminology.

- Let n be the number of nodes in the network.
- Let m be the number of edges in the network.
- Let $s = 2m$ be the number of spokes (edge ends) in the network.
- Let k_i be the number of spokes incident upon a node i.
- Let k_i^{in} be the number of inbound edges for a node i.

Nomenclature

Let us define basic terminology.

- Let n be the number of nodes in the network.
- Let m be the number of edges in the network.
- Let $s = 2m$ be the number of spokes (edge ends) in the network.
- Let k_i be the number of spokes incident upon a node i.
- Let k_i^{in} be the number of inbound edges for a node i.
- Let k_i^{out} be the number of outbound edges for a node i.

Nomenclature

Let us define basic terminology.

- Let n be the number of nodes in the network.
- Let m be the number of edges in the network.
- Let $s = 2m$ be the number of spokes (edge ends) in the network.
- Let k_i be the number of spokes incident upon a node i.
- Let k_i^{in} be the number of inbound edges for a node i.
- Let k_i^{out} be the number of outbound edges for a node i.
- Let A be an adjacency matrix with entries $A_{i,j}$ representing the number of spokes connecting nodes i and j.

Nomenclature

Let us define basic terminology.

- Let n be the number of nodes in the network.
- Let m be the number of edges in the network.
- Let $s = 2m$ be the number of spokes (edge ends) in the network.
- Let k_i be the number of spokes incident upon a node i.
- Let k_i^{in} be the number of inbound edges for a node i.
- Let k_i^{out} be the number of outbound edges for a node i.
- Let A be an adjacency matrix with entries $A_{i,j}$ representing the number of spokes connecting nodes i and j.
- Let $\vec{A}$ be a directed adjacency matrix with entries $\vec{A}_{i,j}$ representing an edge from node i to j.

Network Types

Networks can take on the following types.

Network Types

Networks can take on the following types.

- Undirected Graph.

Network Types

Networks can take on the following types.

- Undirected Graph.
  - An edge from node i to j implies an edge from node j to i.

Network Types

Networks can take on the following types.

- Undirected Graph.
  - An edge from node i to j implies an edge from node j to i.
- Weighted or Multi-Graph.

Network Types

Networks can take on the following types.

- Undirected Graph.
  - An edge from node i to j implies an edge from node j to i.
- Weighted or Multi-Graph.
  - Multiple edges can run between two nodes, or otherwise edges can be given weights as an attribute.

Network Types

Networks can take on the following types.

- Undirected Graph.
  - An edge from node i to j implies an edge from node j to i.
- Weighted or Multi-Graph.
  - Multiple edges can run between two nodes, or otherwise edges can be given weights as an attribute.
- Directed Graph.

Network Types

Networks can take on the following types.

- Undirected Graph.
  - An edge from node i to j implies an edge from node j to i.
- Weighted or Multi-Graph.
  - Multiple edges can run between two nodes, or otherwise edges can be given weights as an attribute.
- Directed Graph.
  - An edge from node i to j does not imply an edge from node j to i.

Network Modeling

In graph theory there are many ways one can model networks.

Network Modeling

In graph theory there are many ways one can model networks.

- Erdős–Rényi Model.

Network Modeling

In graph theory there are many ways one can model networks.

- Erdős–Rényi Model.
- Exponential Random Graph Model.

Network Modeling

In graph theory there are many ways one can model networks.

- Erdős–Rényi Model.
- Exponential Random Graph Model.
- Stochastic Block Model.

Network Modeling

In graph theory there are many ways one can model networks.

- Erdős–Rényi Model.
- Exponential Random Graph Model.
- Stochastic Block Model.
- Configuration Model.

Network Modeling

In graph theory there are many ways one can model networks.

- Erdős–Rényi Model.
- Exponential Random Graph Model.
- Stochastic Block Model.
- Configuration Model.

In general different models generate graphs based on an underlying theory of how edges are formed between nodes. Modularity Maximization leverages the configuration model.

Configuration Model

The configuration model leverages a set degree distribution.

Configuration Model

The configuration model leverages a set degree distribution.

- In real-world graphs, the degree distribution is given.

Configuration Model

The configuration model leverages a set degree distribution.

- In real-world graphs, the degree distribution is given.
- Graphs generated from the configuration model must obey the set degree distribution.

Configuration Model

The configuration model leverages a set degree distribution.

- In real-world graphs, the degree distribution is given.
- Graphs generated from the configuration model must obey the set degree distribution.
  - many graphs may be possible with the same degree distribution.

Configuration Model

The configuration model leverages a set degree distribution.

- In real-world graphs, the degree distribution is given.
- Graphs generated from the configuration model must obey the set degree distribution.
  - many graphs may be possible with the same degree distribution.
  - the model creates edges between nodes based on the probability of them existing.

Configuration Model

The configuration model leverages a set degree distribution.

- In real-world graphs, the degree distribution is given.
- Graphs generated from the configuration model must obey the set degree distribution.
  - ◇ many graphs may be possible with the same degree distribution.
  - ◇ the model creates edges between nodes based on the probability of them existing.
  - ∗ this probability is derived from the degree distribution.

Configuration Model Features

The configuration model provides us with some nice features.

Configuration Model Features

The configuration model provides us with some nice features.

- The total number of spokes will be, $\sum_{i}^{n} k_i = s$.

Configuration Model Features

The configuration model provides us with some nice features.

- The total number of spokes will be, $\sum_{i}^{n} k_i = s$.
- The total number of edges will be, $\sum_{i}^{n} k_i^{out} = \sum_{i}^{n} k_i^{in} = m$.

Configuration Model Features

The configuration model provides us with some nice features.

- The total number of spokes will be, $\sum_{i}^{n} k_i = s$.
- The total number of edges will be, $\sum_{i}^{n} k_i^{out} = \sum_{i}^{n} k_i^{in} = m$.
- The number of ways to connect spokes (form edges) between two distinct nodes i, j in an undirected network is $k_i k_j$.

Configuration Model Features

The configuration model provides us with some nice features.

- The total number of spokes will be, $\sum_{i}^{n} k_i = s$.
- The total number of edges will be, $\sum_{i}^{n} k_i^{out} = \sum_{i}^{n} k_i^{in} = m$.
- The number of ways to connect spokes (form edges) between two distinct nodes i, j in an undirected network is $k_i k_j$.
- The number of ways to from self loops for a node i is $k_i(k_i - 1)$.

Configuration Model Features

The configuration model provides us with some nice features.

- The total number of spokes will be, $\sum_{i}^{n} k_i = s$.
- The total number of edges will be, $\sum_{i}^{n} k_i^{out} = \sum_{i}^{n} k_i^{in} = m$.
- The number of ways to connect spokes (form edges) between two distinct nodes i, j in an undirected network is $k_i k_j$.
- The number of ways to from self loops for a node i is $k_i(k_i - 1)$.
- The number of ways to form an edge from node i to node j in a directed network is $k_i^{out} k_j^{in}$.

Spoke/Edge Expectation

Building upon the features of the configuration model, let us define the expected number of spokes/edges between two nodes.

Spoke/Edge Expectation

Building upon the features of the configuration model, let us define the expected number of spokes/edges between two nodes.

First let us note the following relating to the number of ways to connect nodes in networks.

Spoke/Edge Expectation

Building upon the features of the configuration model, let us define the expected number of spokes/edges between two nodes.

First let us note the following relating to the number of ways to connect nodes in networks.

- $\sum_{i}^{n} k_i \sum_{j}^{n} k_j = s^2.$

Spoke/Edge Expectation

Building upon the features of the configuration model, let us define the expected number of spokes/edges between two nodes.

First let us note the following relating to the number of ways to connect nodes in networks.

- $\sum_{i}^{n} k_i \sum_{j}^{n} k_j = s^2$.
  - undirected case (spokes).

Spoke/Edge Expectation

Building upon the features of the configuration model, let us define the expected number of spokes/edges between two nodes.

First let us note the following relating to the number of ways to connect nodes in networks.

- $\sum_{i}^{n} k_i \sum_{j}^{n} k_j = s^2.$
  - undirected case (spokes).
- $\sum_{i}^{n} k_i^{out} \sum_{j}^{n} k_j^{in} = m^2.$

Spoke/Edge Expectation

Building upon the features of the configuration model, let us define the expected number of spokes/edges between two nodes.

First let us note the following relating to the number of ways to connect nodes in networks.

- $\sum_{i}^{n} k_i \sum_{j}^{n} k_j = s^2$.
  - undirected case (spokes).
- $\sum_{i}^{n} k_i^{out} \sum_{j}^{n} k_j^{in} = m^2$.
  - directed case (edges).

Spoke/Edge Expectation

When summing over the expected number of spokes for each undirected node pair we should get the total number of spokes (s).

Spoke/Edge Expectation

When summing over the expected number of spokes for each undirected node pair we should get the total number of spokes (s).

When summing over the expected number of edges for each directed node pair we should get the total number of edges (m).

Spoke/Edge Expectation

When summing over the expected number of spokes for each undirected node pair we should get the total number of spokes (s).

When summing over the expected number of edges for each directed node pair we should get the total number of edges (m).

Building upon the previous slide we can retrieve the expected number of spokes/edges $\mathbb{E}_{i,j}$ and $\vec{\mathbb{E}}_{i,j}$ in an undirected and directed network.

Spoke/Edge Expectation

When summing over the expected number of spokes for each undirected node pair we should get the total number of spokes (s).

When summing over the expected number of edges for each directed node pair we should get the total number of edges (m).

Building upon the previous slide we can retrieve the expected number of spokes/edges $\mathbb{E}_{i,j}$ and $\vec{\mathbb{E}}_{i,j}$ in an undirected and directed network.

- $\mathbb{E}_{i,j\neq i} = \frac{k_i k_j}{s-1}$.

Spoke/Edge Expectation

When summing over the expected number of spokes for each undirected node pair we should get the total number of spokes (s).

When summing over the expected number of edges for each directed node pair we should get the total number of edges (m).

Building upon the previous slide we can retrieve the expected number of spokes/edges $\mathbb{E}_{i,j}$ and $\vec{\mathbb{E}}_{i,j}$ in an undirected and directed network.

- $\mathbb{E}_{i,j\neq i} = \dfrac{k_i k_j}{s-1}$.
- $\mathbb{E}_{i,i} = \dfrac{k_i(k_i-1)}{s-1}$.

Spoke/Edge Expectation

When summing over the expected number of spokes for each undirected node pair we should get the total number of spokes (s).

When summing over the expected number of edges for each directed node pair we should get the total number of edges (m).

Building upon the previous slide we can retrieve the expected number of spokes/edges $\mathbb{E}_{i,j}$ and $\vec{\mathbb{E}}_{i,j}$ in an undirected and directed network.

- $\mathbb{E}_{i,j\neq i} = \dfrac{k_i k_j}{s-1}$.
- $\mathbb{E}_{i,i} = \dfrac{k_i(k_i-1)}{s-1}$.
- $\vec{\mathbb{E}}_{i,j} = \dfrac{k_i^{out} k_j^{in}}{m}$.

Proof of Spoke Expectation (Undirected)

$$\sum_{i,j}^{n} \mathbb{E}_{i,j} = \sum_{i,j\neq i}^{n} \mathbb{E}_{i,j} + \sum_{i}^{n} \mathbb{E}_{i,i}$$

Proof of Spoke Expectation (Undirected)

$$\sum_{i,j}^{n} \mathbb{E}_{i,j} = \sum_{i,j\neq i}^{n} \mathbb{E}_{i,j} + \sum_{i}^{n} \mathbb{E}_{i,i}$$

$$= \sum_{i,j\neq i}^{n} \frac{k_i k_j}{s-1} + \sum_{i}^{n} \frac{k_i(k_i-1)}{s-1}$$

Proof of Spoke Expectation (Undirected)

$$\sum_{i,j}^{n} \mathbb{E}_{i,j} = \sum_{i,j\neq i}^{n} \mathbb{E}_{i,j} + \sum_{i}^{n} \mathbb{E}_{i,i}$$

$$= \sum_{i,j\neq i}^{n} \frac{k_i k_j}{s-1} + \sum_{i}^{n} \frac{k_i(k_i-1)}{s-1}$$

$$= \frac{1}{s-1}\left[\sum_{i}^{n} k_i \sum_{j\neq i}^{n} k_j + \sum_{i}^{n} k_i \sum_{i}^{n} k_i - \sum_{i}^{n} k_i\right]$$

Proof of Spoke Expectation (Undirected)

$$\sum_{i,j}^{n} \mathbb{E}_{i,j} = \sum_{i,j\neq i}^{n} \mathbb{E}_{i,j} + \sum_{i}^{n} \mathbb{E}_{i,i}$$

$$= \sum_{i,j\neq i}^{n} \frac{k_i k_j}{s-1} + \sum_{i}^{n} \frac{k_i(k_i-1)}{s-1}$$

$$= \frac{1}{s-1}\left[\sum_{i}^{n} k_i \sum_{j\neq i}^{n} k_j + \sum_{i}^{n} k_i \sum_{i}^{n} k_i - \sum_{i}^{n} k_i\right]$$

$$= \frac{1}{s-1}\left[\sum_{i}^{n} k_i \sum_{j}^{n} k_j - s\right]$$

Proof of Spoke Expectation (Undirected)

$$\sum_{i,j}^{n} \mathbb{E}_{i,j} = \sum_{i,j\neq i}^{n} \mathbb{E}_{i,j} + \sum_{i}^{n} \mathbb{E}_{i,i}$$

$$= \sum_{i,j\neq i}^{n} \frac{k_i k_j}{s-1} + \sum_{i}^{n} \frac{k_i(k_i-1)}{s-1}$$

$$= \frac{1}{s-1}\left[\sum_{i}^{n} k_i \sum_{j\neq i}^{n} k_j + \sum_{i}^{n} k_i \sum_{i}^{n} k_i - \sum_{i}^{n} k_i\right]$$

$$= \frac{1}{s-1}\left[\sum_{i}^{n} k_i \sum_{j}^{n} k_j - s\right]$$

$$= \frac{s^2-s}{s-1}$$

Proof of Spoke Expectation (Undirected)

$$\sum_{i,j}^{n} \mathbb{E}_{i,j} = \sum_{i,j\neq i}^{n} \mathbb{E}_{i,j} + \sum_{i}^{n} \mathbb{E}_{i,i}$$

$$= \sum_{i,j\neq i}^{n} \frac{k_i k_j}{s-1} + \sum_{i}^{n} \frac{k_i(k_i-1)}{s-1}$$

$$= \frac{1}{s-1}\left[\sum_{i}^{n} k_i \sum_{j\neq i}^{n} k_j + \sum_{i}^{n} k_i \sum_{i}^{n} k_i - \sum_{i}^{n} k_i\right]$$

$$= \frac{1}{s-1}\left[\sum_{i}^{n} k_i \sum_{j}^{n} k_j - s\right]$$

$$= \frac{s^2-s}{s-1} = \frac{s(s-1)}{s-1}$$

Proof of Spoke Expectation (Undirected)

$$\sum_{i,j}^{n} \mathbb{E}_{i,j} = \sum_{i,j\neq i}^{n} \mathbb{E}_{i,j} + \sum_{i}^{n} \mathbb{E}_{i,i}$$

$$= \sum_{i,j\neq i}^{n} \frac{k_i k_j}{s-1} + \sum_{i}^{n} \frac{k_i(k_i-1)}{s-1}$$

$$= \frac{1}{s-1}\left[\sum_{i}^{n} k_i \sum_{j\neq i}^{n} k_j + \sum_{i}^{n} k_i \sum_{i}^{n} k_i - \sum_{i}^{n} k_i\right]$$

$$= \frac{1}{s-1}\left[\sum_{i}^{n} k_i \sum_{j}^{n} k_j - s\right]$$

$$= \frac{s^2-s}{s-1} = \frac{s(s-1)}{s-1} = s$$

Proof of Edge Expectation (Directed)

$$\sum_{i,j}^{n} \vec{\mathbb{E}}_{i,j} = \sum_{i,j}^{n} \frac{k_i^{out} k_j^{in}}{m}$$

Proof of Edge Expectation (Directed)

$$\sum_{i,j}^{n} \vec{\mathbb{E}}_{i,j} = \sum_{i,j}^{n} \frac{k_i^{out} k_j^{in}}{m}$$

$$= \frac{1}{m} \left[\sum_{i}^{n} k_i^{out} \sum_{j}^{n} k_j^{in} \right]$$

Proof of Edge Expectation (Directed)

$$\sum_{i,j}^{n} \vec{\mathbb{E}}_{i,j} = \sum_{i,j}^{n} \frac{k_i^{out} k_j^{in}}{m}$$

$$= \frac{1}{m} \left[\sum_{i}^{n} k_i^{out} \sum_{j}^{n} k_j^{in} \right]$$

$$= \frac{m^2}{m}$$

Proof of Edge Expectation (Directed)

$$\sum_{i,j}^{n} \vec{\mathbb{E}}_{i,j} = \sum_{i,j}^{n} \frac{k_i^{out} k_j^{in}}{m}$$

$$= \frac{1}{m} \left[\sum_{i}^{n} k_i^{out} \sum_{j}^{n} k_j^{in} \right]$$

$$= \frac{m^2}{m} = m$$

Modularity (Undirected)

Modularity is meant to measure the goodness of community partitions, under the assumption that communities have a high density of internal edges.

Modularity (Undirected)

Modularity is meant to measure the goodness of community partitions, under the assumption that communities have a high density of internal edges.

- $Q = \%\ internal\ edges - Expected\ \%\ internal\ edges.$

Modularity (Undirected)

Modularity is meant to measure the goodness of community partitions, under the assumption that communities have a high density of internal edges.

- $Q = \%\ internal\ edges - Expected\ \%\ internal\ edges.$
- $\delta(c_i, c_j) = \begin{cases} 1 & i\ \&\ j\ are\ in\ the\ same\ community \\ \end{cases}$

Modularity (Undirected)

Modularity is meant to measure the goodness of community partitions, under the assumption that communities have a high density of internal edges.

- $Q = \%\ internal\ edges - Expected\ \%\ internal\ edges.$
- $\delta(c_i, c_j) = \begin{cases} 1 & i\ \&\ j\ are\ in\ the\ same\ community \\ 0 & else \end{cases}.$

Modularity (Undirected)

Modularity is meant to measure the goodness of community partitions, under the assumption that communities have a high density of internal edges.

- $Q = \%\ internal\ edges - Expected\ \%\ internal\ edges.$
- $\delta(c_i, c_j) = \begin{cases} 1 & i\ \&\ j\ are\ in\ the\ same\ community \\ 0 & else \end{cases}.$
- Note: $\delta(c_i, c_i) = 1.$

Modularity (Undirected)

Modularity is meant to measure the goodness of community partitions, under the assumption that communities have a high density of internal edges.

- $Q = \%\ internal\ edges - Expected\ \%\ internal\ edges.$
- $\delta(c_i, c_j) = \begin{cases} 1 & i\ \&\ j\ are\ in\ the\ same\ community \\ 0 & else \end{cases}.$
- Note: $\delta(c_i, c_i) = 1.$

$$Q = \frac{1}{s}\left[\sum_{i,j\neq i}^{n}\left(A_{i,j} - \frac{k_i k_j}{s-1}\right)\delta(c_i, c_j) + \sum_{i}^{n}\left(A_{i,i} - \frac{k_i(k_i-1)}{s-1}\right)\right].$$

Modularity Simplified (Undirected)

Heuristics:

Modularity Simplified (Undirected)

Heuristics:

1. As networks grow large the subtraction of 1 from the number of possible spoke to spoke connections becomes negligible.

Modularity Simplified (Undirected)

Heuristics:

1. As networks grow large the subtraction of 1 from the number of possible spoke to spoke connections becomes negligible.
2. The number of self edges is proportionally small compared to the total number of edges, and can be treated as if the repeated nodes are distinct.

Modularity Simplified (Undirected)

Heuristics:

1. As networks grow large the subtraction of 1 from the number of possible spoke to spoke connections becomes negligible.
2. The number of self edges is proportionally small compared to the total number of edges, and can be treated as if the repeated nodes are distinct.

Simplifications:

Modularity Simplified (Undirected)

Heuristics:

1. As networks grow large the subtraction of 1 from the number of possible spoke to spoke connections becomes negligible.
2. The number of self edges is proportionally small compared to the total number of edges, and can be treated as if the repeated nodes are distinct.

Simplifications:

1. $\frac{k_i k_j}{s-1} \approx \frac{k_i k_j}{s}$.

Modularity Simplified (Undirected)

Heuristics:

1. As networks grow large the subtraction of 1 from the number of possible spoke to spoke connections becomes negligible.
2. The number of self edges is proportionally small compared to the total number of edges, and can be treated as if the repeated nodes are distinct.

Simplifications:

1. $\frac{k_i k_j}{s-1} \approx \frac{k_i k_j}{s}$.
2. $k_i(k_i - 1) \approx k_i^2$.

Modularity Simplified (Undirected)

Heuristics:

1. As networks grow large the subtraction of 1 from the number of possible spoke to spoke connections becomes negligible.
2. The number of self edges is proportionally small compared to the total number of edges, and can be treated as if the repeated nodes are distinct.

Simplifications:

1. $\frac{k_i k_j}{s-1} \approx \frac{k_i k_j}{s}$.
2. $k_i(k_i - 1) \approx k_i^2$.

$$Q = \frac{1}{s}\left[\sum_{i,j}^{n}\left(A_{i,j} - \frac{k_i k_j}{s}\right)\delta(c_i, c_j)\right].$$

Modularity (Directed)

Only a slight modification is needed to extend modularity to directed networks.

Modularity (Directed)

Only a slight modification is needed to extend modularity to directed networks.

This modification follows from previous work and is to use $\vec{\mathbb{E}}_{i,j}$ for edge expectation rather than $\mathbb{E}_{i,j}$

Modularity (Directed)

Only a slight modification is needed to extend modularity to directed networks.

This modification follows from previous work and is to use $\vec{\mathbb{E}}_{i,j}$ for edge expectation rather than $\mathbb{E}_{i,j}$

$$\vec{Q} = \frac{1}{m}\left[\sum_{i,j}^{n}\left(\vec{A}_{i,j} - \frac{k_i^{out}k_j^{in}}{m}\right)\delta(c_i, c_j)\right].$$

Modularity (Directed)

Only a slight modification is needed to extend modularity to directed networks.

This modification follows from previous work and is to use $\vec{\mathbb{E}}_{i,j}$ for edge expectation rather than $\mathbb{E}_{i,j}$

$$\vec{Q} = \frac{1}{m}\left[\sum_{i,j}^{n}\left(\vec{A}_{i,j} - \frac{k_i^{out}k_j^{in}}{m}\right)\delta(c_i, c_j)\right].$$

If one likes modularity can be written in the following form:

Modularity (Directed)

Only a slight modification is needed to extend modularity to directed networks.

This modification follows from previous work and is to use $\vec{\mathbb{E}}_{i,j}$ for edge expectation rather than $\mathbb{E}_{i,j}$

$$\vec{Q} = \frac{1}{m}\left[\sum_{i,j}^{n}\left(\vec{A}_{i,j} - \frac{k_i^{out}k_j^{in}}{m}\right)\delta(c_i, c_j)\right].$$

If one likes modularity can be written in the following form:

$$\vec{Q} = \frac{1}{m}\left[\sum_{i,j}^{n}\left(\vec{A}_{i,j} - \vec{\mathbb{E}}_{i,j}\right)\delta(c_i, c_j)\right].$$

Introduction to Modularity Maximization

Initially, let us assume all nodes belong to a distinct community.

Introduction to Modularity Maximization

Initially, let us assume all nodes belong to a distinct community.

From here we can consider changes in modularity based on reassigning node communities.

Introduction to Modularity Maximization

Initially, let us assume all nodes belong to a distinct community.

From here we can consider changes in modularity based on reassigning node communities.

In an iterative fashion, we can performed the reassignment with the greatest increase in modularity.

Introduction to Modularity Maximization

Initially, let us assume all nodes belong to a distinct community.

From here we can consider changes in modularity based on reassigning node communities.

In an iterative fashion, we can performed the reassignment with the greatest increase in modularity.

One note on modularity maximization methods is that they detect disjoint communities, so it has a decreased accuracy for networks with underlying hierarchical or overlapping mechanics.

Complexity Optimization in Modularity Maximization [1/2]

In general, modularity maximization techniques are often used due to their ability to analyze massive networks (millions of nodes).

Complexity Optimization in Modularity Maximization [1/2]

In general, modularity maximization techniques are often used due to their ability to analyze massive networks (millions of nodes).

In order to have a reasonable runtime, there are a few steps in which we can optimize.

Complexity Optimization in Modularity Maximization [1/2]

In general, modularity maximization techniques are often used due to their ability to analyze massive networks (millions of nodes).

In order to have a reasonable runtime, there are a few steps in which we can optimize.

- How we reassign communities.

Complexity Optimization in Modularity Maximization [1/2]

In general, modularity maximization techniques are often used due to their ability to analyze massive networks (millions of nodes).

In order to have a reasonable runtime, there are a few steps in which we can optimize.

- How we reassign communities.
- How we calculate modularity changes.

Complexity Optimization in Modularity Maximization [1/2]

In general, modularity maximization techniques are often used due to their ability to analyze massive networks (millions of nodes).

In order to have a reasonable runtime, there are a few steps in which we can optimize.

- How we reassign communities.
- How we calculate modularity changes.
- When we stop iterating.

Complexity Optimization in Modularity Maximization [2/2]

How might we go about such optimizations?

- Reassign multiple node communities in each iteration.

Complexity Optimization in Modularity Maximization [2/2]

How might we go about such optimizations?

- Reassign multiple node communities in each iteration.
- Consider reassignments only within node neighborhoods (the neighbors of a node are all nodes in which it shares an edge).

Complexity Optimization in Modularity Maximization [2/2]

How might we go about such optimizations?

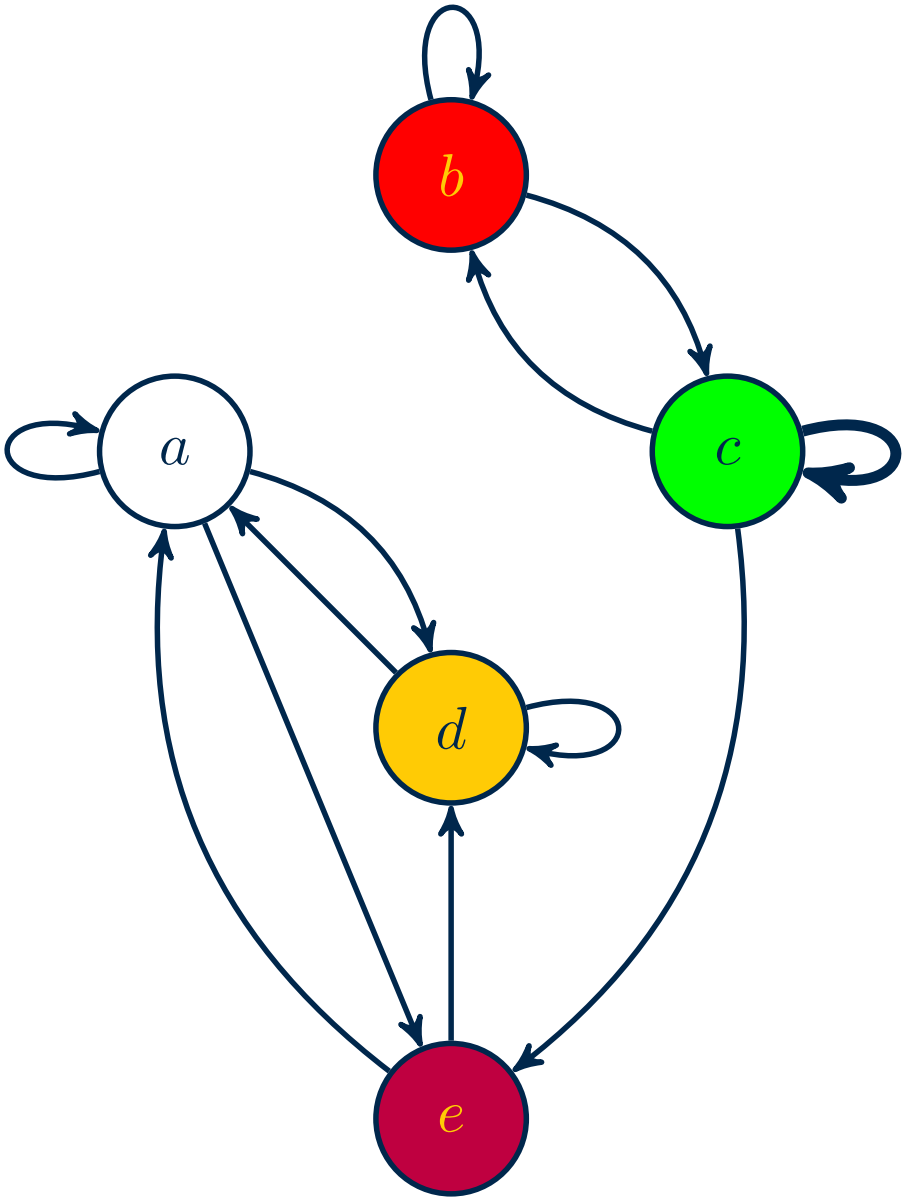
- Reassign multiple node communities in each iteration.
- Consider reassignments only within node neighborhoods (the neighbors of a node are all nodes in which it shares an edge).
- Cache relevant values (node degrees, expectations) rather than computing during runtime.

Complexity Optimization in Modularity Maximization [2/2]

How might we go about such optimizations?

- Reassign multiple node communities in each iteration.
- Consider reassignments only within node neighborhoods (the neighbors of a node are all nodes in which it shares an edge).
- Cache relevant values (node degrees, expectations) rather than computing during runtime.
- Halt the algorithm once the highest possible change in modularity is below a set threshold.

Example: Network



Example: Adjacency Matrix

$\vec{A}$	a	b	c	d	e
a	1	0	0	1	1
b	0	1	1	0	0
c	0	1	2	0	1
d	1	0	0	1	0
e	1	0	0	1	0

Example: Adjacency Matrix

$\vec{A}$	a	b	c	d	e
a	1	0	0	1	1
b	0	1	1	0	0
c	0	1	2	0	1
d	1	0	0	1	0
e	1	0	0	1	0

The adjacency matrix is the expected form computers will store networks in.

Example: Out-Degree Calculations

$\vec{A}$	a	b	c	d	e
a	1	0	0	1	1
b	0	1	1	0	0
c	0	1	2	0	1
d	1	0	0	1	0
e	1	0	0	1	0

Example: Out-Degree Calculations

$\vec{A}$	a	b	c	d	e
a	1	0	0	1	1
b	0	1	1	0	0
c	0	1	2	0	1
d	1	0	0	1	0
e	1	0	0	1	0

$k_a^{out} = 1 + 0 + 0 + 1 + 1 = 3.$

Example: In-Degree Calculations

$\vec{A}$	a	b	c	d	e
a	1	0	0	1	1
b	0	1	1	0	0
c	0	1	2	0	1
d	1	0	0	1	0
e	1	0	0	1	0

Example: In-Degree Calculations

$\vec{A}$	a	b	c	d	e
a	1	0	0	1	1
b	0	1	1	0	0
c	0	1	2	0	1
d	1	0	0	1	0
e	1	0	0	1	0

$k_a^{in} = 1 + 0 + 0 + 1 + 1 = 3.$

Example: All Degree Calculations

i	a	b	c	d	e
k_i^{out}	3	2	4	2	2
k_i^{in}	3	2	3	3	2
k_i	6	4	7	5	4

Example: All Degree Calculations

i	a	b	c	d	e
k_i^{out}	3	2	4	2	2
k_i^{in}	3	2	3	3	2
k_i	6	4	7	5	4

$$m = 3 + 2 + 4 + 2 + 2$$

Example: All Degree Calculations

i	a	b	c	d	e
k_i^{out}	3	2	4	2	2
k_i^{in}	3	2	3	3	2
k_i	6	4	7	5	4

$$m = 3 + 2 + 4 + 2 + 2 = 3 + 2 + 3 + 3 + 2$$

Example: All Degree Calculations

i	a	b	c	d	e
k_i^{out}	3	2	4	2	2
k_i^{in}	3	2	3	3	2
k_i	6	4	7	5	4

$$m = 3 + 2 + 4 + 2 + 2 = 3 + 2 + 3 + 3 + 2 = 13.$$

Example: Modularity Calculation [1/2]

We need only consider the diagonal of the adjacency matrix as all node pairs but the self pair are not in the same community.

Example: Modularity Calculation [1/2]

We need only consider the diagonal of the adjacency matrix as all node pairs but the self pair are not in the same community.

$$\vec{Q} = \frac{1}{m}\Big[(\vec{A}_{a,a} - \vec{\mathbb{E}}_{a,a}) + (\vec{A}_{b,b} - \vec{\mathbb{E}}_{b,b}) + (\vec{A}_{c,c} - \vec{\mathbb{E}}_{c,c}) + (\vec{A}_{d,d} - \vec{\mathbb{E}}_{d,d}) + (\vec{A}_{e,e} - \vec{\mathbb{E}}_{e,e})\Big].$$

Example: Modularity Calculation [1/2]

We need only consider the diagonal of the adjacency matrix as all node pairs but the self pair are not in the same community.

$$\vec{Q} = \frac{1}{m}\Big[(\vec{A}_{a,a} - \vec{\mathbb{E}}_{a,a}) + (\vec{A}_{b,b} - \vec{\mathbb{E}}_{b,b}) + (\vec{A}_{c,c} - \vec{\mathbb{E}}_{c,c}) + (\vec{A}_{d,d} - \vec{\mathbb{E}}_{d,d}) + (\vec{A}_{e,e} - \vec{\mathbb{E}}_{e,e})\Big].$$

$$\vec{Q} = \frac{1}{13}\Big[(1 - \frac{3 \cdot 3}{13}) + (1 - \frac{2 \cdot 2}{13}) + (2 - \frac{4 \cdot 3}{13}) + (1 - \frac{2 \cdot 3}{13}) + (0 - \frac{2 \cdot 2}{13})\Big].$$

Example: Modularity Calculation [2/2]

$$\vec{Q} = \frac{1}{13}\left[\frac{4}{13} + \frac{9}{13} + \frac{14}{13} + \frac{7}{13} - \frac{4}{13}\right].$$

Example: Modularity Calculation [2/2]

$$\vec{Q} = \frac{1}{13}\left[\frac{4}{13} + \frac{9}{13} + \frac{14}{13} + \frac{7}{13} - \frac{4}{13}\right].$$

$$\vec{Q} = \frac{1}{13} \cdot \frac{30}{13}$$

Example: Modularity Calculation [2/2]

$$\vec{Q} = \frac{1}{13}\left[\frac{4}{13} + \frac{9}{13} + \frac{14}{13} + \frac{7}{13} - \frac{4}{13}\right].$$

$$\vec{Q} = \frac{1}{13} \cdot \frac{30}{13} = \frac{30}{169}.$$

Modularity Maximization Algorithm [1/2]

How does a modularity maximization algorithm operate in practice?

Modularity Maximization Algorithm [1/2]

How does a modularity maximization algorithm operate in practice?

Typically, community reassignments are realized as node merges.

Modularity Maximization Algorithm [1/2]

How does a modularity maximization algorithm operate in practice?

Typically, community reassignments are realized as node merges.

By doing so we can treat merged nodes as meta nodes, and form a new adjacency matrix with them.

Modularity Maximization Algorithm [1/2]

How does a modularity maximization algorithm operate in practice?

Typically, community reassignments are realized as node merges.

By doing so we can treat merged nodes as meta nodes, and form a new adjacency matrix with them.

As a result we need only update the cached values which change.

Modularity Maximization Algorithm [2/2]

To calculate $\Delta\vec{Q}$ we need only consider changes in modularity resulting from the node merge.

Modularity Maximization Algorithm [2/2]

To calculate $\Delta\vec{Q}$ we need only consider changes in modularity resulting from the node merge.

For our adjacency matrix this is just a row and column merge.

Modularity Maximization Algorithm [2/2]

To calculate $\Delta\vec{Q}$ we need only consider changes in modularity resulting from the node merge.

For our adjacency matrix this is just a row and column merge.

Similarly, for recalculating k_i^{out} and k_i^{in} we need only one addition.

Modularity Maximization Algorithm [2/2]

To calculate $\Delta\vec{Q}$ we need only consider changes in modularity resulting from the node merge.

For our adjacency matrix this is just a row and column merge.

Similarly, for recalculating k_i^{out} and k_i^{in} we need only one addition.

Last, we will only consider merges for nodes which are neighbors.

Example: Neighborhoods

Form our example graph we can create a neighborhood table as follows.

Example: Neighborhoods

Form our example graph we can create a neighborhood table as follows.

i	$neighborhood$
a	$\{d, e\}$
b	$\{c\}$
c	$\{b, e\}$
d	$\{a, e\}$
e	$\{a, c, e\}$

Example: Neighborhoods

Form our example graph we can create a neighborhood table as follows.

i	$neighborhood$
a	$\{d, e\}$
b	$\{c\}$
c	$\{b, e\}$
d	$\{a, e\}$
e	$\{a, c, e\}$

Further, we can derive all node merges which must be considered.

Example: Neighborhoods

Form our example graph we can create a neighborhood table as follows.

i	$neighborhood$
a	$\{d, e\}$
b	$\{c\}$
c	$\{b, e\}$
d	$\{a, e\}$
e	$\{a, c, e\}$

Further, we can derive all node merges which must be considered.

$$\{a, d\}, \{a, e\}, \{b, c\}, \{c, e\}, \{d, e\}.$$

Example: Neighborhoods

Form our example graph we can create a neighborhood table as follows.

i	$neighborhood$
a	$\{d, e\}$
b	$\{c\}$
c	$\{b, e\}$
d	$\{a, e\}$
e	$\{a, c, e\}$

Further, we can derive all node merges which must be considered.

$$\{a, d\}, \{a, e\}, \{b, c\}, \{c, e\}, \{d, e\}.$$

The merges we need consider should be much less that the $\binom{n}{2}$ node pairs.

Example: $\Delta\vec{Q}$ Calculation [1/2]

We will choose the node merge with the best change $\Delta\vec{Q}$.

Example: $\Delta\vec{Q}$ Calculation [1/2]

We will choose the node merge with the best change $\Delta\vec{Q}$.

$merge$	$A_{i,i}$	$A_{j,j}$	$A_{i,j}$	$A_{j,i}$	$A_{i',i'}$
$\{a, d\}$	1	1	1	1	4
$\{a, e\}$	1	0	1	1	3
$\{b, c\}$	1	2	1	1	5
$\{c, e\}$	2	0	1	0	3
$\{d, e\}$	1	0	0	1	2

Example: $\Delta\vec{Q}$ Calculation [1/2]

We will choose the node merge with the best change $\Delta\vec{Q}$.

$merge$	$A_{i,i}$	$A_{j,j}$	$A_{i,j}$	$A_{j,i}$	$A_{i',i'}$
$\{a,d\}$	1	1	1	1	4
$\{a,e\}$	1	0	1	1	3
$\{b,c\}$	1	2	1	1	5
$\{c,e\}$	2	0	1	0	3
$\{d,e\}$	1	0	0	1	2

$merge$	k_i^{out}	k_j^{out}	$k_{i'}^{out}$	k_i^{in}	k_j^{in}	$k_{i'}^{in}$
$\{a,d\}$	3	2	5	3	3	6
$\{a,e\}$	3	2	5	3	2	5
$\{b,c\}$	2	4	6	2	3	5
$\{c,e\}$	4	2	6	3	2	5
$\{d,e\}$	2	2	4	3	2	5

Example: $\Delta\vec{Q}$ Calculation [1/2]

We will choose the node merge with the best change $\Delta\vec{Q}$.

$merge$	$A_{i,i}$	$A_{j,j}$	$A_{i,j}$	$A_{j,i}$	$A_{i',i'}$
$\{a,d\}$	1	1	1	1	4
$\{a,e\}$	1	0	1	1	3
$\{b,c\}$	1	2	1	1	5
$\{c,e\}$	2	0	1	0	3
$\{d,e\}$	1	0	0	1	2

$merge$	k_i^{out}	k_j^{out}	$k_{i'}^{out}$	k_i^{in}	k_j^{in}	$k_{i'}^{in}$
$\{a,d\}$	3	2	5	3	3	6
$\{a,e\}$	3	2	5	3	2	5
$\{b,c\}$	2	4	6	2	3	5
$\{c,e\}$	4	2	6	3	2	5
$\{d,e\}$	2	2	4	3	2	5

Where i' is the new meta node formed from merging nodes i and j.

Example: $\Delta\vec{Q}$ Calculation [2/2]

Let $\partial\vec{Q}_i, \partial\vec{Q}_j, \partial\vec{Q}_{i'}$ denote the contribution to modularity by the communities $i, j,$ and i' respectively.

Example: $\Delta\vec{Q}$ Calculation [2/2]

Let $\partial\vec{Q}_i, \partial\vec{Q}_j, \partial\vec{Q}_{i'}$ denote the contribution to modularity by the communities $i, j,$ and i' respectively.

Additionally, Let $\Delta\vec{Q} = \frac{1}{m}\left[\partial\vec{Q}_{i'} - (\partial\vec{Q}_i + \partial\vec{Q}_j)\right]$

Example: $\Delta\vec{Q}$ Calculation [2/2]

Let $\partial\vec{Q}_i, \partial\vec{Q}_j, \partial\vec{Q}_{i'}$ denote the contribution to modularity by the communities i, j, and i' respectively.

Additionally, Let $\Delta\vec{Q} = \frac{1}{m}\left[\partial\vec{Q}_{i'} - (\partial\vec{Q}_i + \partial\vec{Q}_j)\right]$

$merge$	$\partial\vec{Q}_i$	$\partial\vec{Q}_j$	$\partial\vec{Q}_{i'}$	$\Delta\vec{Q}$
$\{a, d\}$	$4/13$	$7/13$	$22/13$	$11/169$
$\{a, e\}$	$4/13$	$-4/13$	$14/13$	$14/169$
$\{b, c\}$	$9/13$	$14/13$	$35/13$	$12/169$
$\{c, e\}$	$14/13$	$-4/13$	$9/13$	$-1/169$
$\{d, e\}$	$7/13$	$-4/13$	$6/13$	$3/169$

Example: $\Delta\vec{Q}$ Calculation [2/2]

Let $\partial\vec{Q}_i, \partial\vec{Q}_j, \partial\vec{Q}_{i'}$ denote the contribution to modularity by the communities i, j, and i' respectively.

Additionally, Let $\Delta\vec{Q} = \frac{1}{m}\left[\partial\vec{Q}_{i'} - (\partial\vec{Q}_i + \partial\vec{Q}_j)\right]$

$merge$	$\partial\vec{Q}_i$	$\partial\vec{Q}_j$	$\partial\vec{Q}_{i'}$	$\Delta\vec{Q}$
$\{a, d\}$	$4/13$	$7/13$	$22/13$	$11/169$
$\{a, e\}$	$4/13$	$-4/13$	$14/13$	$14/169$
$\{b, c\}$	$9/13$	$14/13$	$35/13$	$12/169$
$\{c, e\}$	$14/13$	$-4/13$	$9/13$	$-1/169$
$\{d, e\}$	$7/13$	$-4/13$	$6/13$	$3/169$

We see now the best node merge is $\{a, e\}$.

Example: $\Delta\vec{Q}$ Calculation [2/2]

Let $\partial\vec{Q}_i, \partial\vec{Q}_j, \partial\vec{Q}_{i'}$ denote the contribution to modularity by the communities i, j, and i' respectively.

Additionally, Let $\Delta\vec{Q} = \frac{1}{m}\left[\partial\vec{Q}_{i'} - (\partial\vec{Q}_i + \partial\vec{Q}_j)\right]$

$merge$	$\partial\vec{Q}_i$	$\partial\vec{Q}_j$	$\partial\vec{Q}_{i'}$	$\Delta\vec{Q}$
$\{a, d\}$	$4/13$	$7/13$	$22/13$	$11/169$
$\{a, e\}$	$4/13$	$-4/13$	$14/13$	$14/169$
$\{b, c\}$	$9/13$	$14/13$	$35/13$	$12/169$
$\{c, e\}$	$14/13$	$-4/13$	$9/13$	$-1/169$
$\{d, e\}$	$7/13$	$-4/13$	$6/13$	$3/169$

We see now the best node merge is $\{a, e\}$.

The process should continue until the best change in modularity is lower than a set threshold.

Example: Multiple Merges [1/2]

To minimize complexity, one may perform many merges in a single iteration.

Example: Multiple Merges [1/2]

To minimize complexity, one may perform many merges in a single iteration.

As is, we already compute modularity gain for each considered merge.

Example: Multiple Merges [1/2]

To minimize complexity, one may perform many merges in a single iteration.

As is, we already compute modularity gain for each considered merge.

Let us perform the best merge for each node.

Example: Multiple Merges [1/2]

To minimize complexity, one may perform many merges in a single iteration.

As is, we already compute modularity gain for each considered merge.

Let us perform the best merge for each node.

$merge$	$\partial\vec{Q}_i$	$\partial\vec{Q}_j$	$\partial\vec{Q}_{i'}$	$\Delta\vec{Q}$
$\{a, d\}$	$4/13$	$7/13$	$22/13$	$11/169$
$\{a, e\}$	$4/13$	$-4/13$	$14/13$	$14/169$
$\{b, c\}$	$9/13$	$14/13$	$35/13$	$12/169$
$\{c, e\}$	$14/13$	$-4/13$	$9/13$	$-1/169$
$\{d, e\}$	$7/13$	$-4/13$	$6/13$	$3/169$

Example: Multiple Merges [1/2]

To minimize complexity, one may perform many merges in a single iteration.

As is, we already compute modularity gain for each considered merge.

Let us perform the best merge for each node.

$merge$	$\partial\vec{Q}_i$	$\partial\vec{Q}_j$	$\partial\vec{Q}_{i'}$	$\Delta\vec{Q}$
$\{a, d\}$	$4/13$	$7/13$	$22/13$	$11/169$
$\{a, e\}$	$4/13$	$-4/13$	$14/13$	$14/169$
$\{b, c\}$	$9/13$	$14/13$	$35/13$	$12/169$
$\{c, e\}$	$14/13$	$-4/13$	$9/13$	$-1/169$
$\{d, e\}$	$7/13$	$-4/13$	$6/13$	$3/169$

The best merge for nodes a, and e, is to merge them together, similarly, the best for b, and c, is to merge them together, last, the best merge for d is to merge it with a.

Example: Multiple Merges [2/2]

For a small network as in the example, we detect the underlying community structure in one iteration.

Example: Multiple Merges [2/2]

For a small network as in the example, we detect the underlying community structure in one iteration.

Our communities evolve in the following manner.

Example: Multiple Merges [2/2]

For a small network as in the example, we detect the underlying community structure in one iteration.

Our communities evolve in the following manner.

- $\{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$

Example: Multiple Merges [2/2]

For a small network as in the example, we detect the underlying community structure in one iteration.

Our communities evolve in the following manner.

- $\{\{a\},\{b\},\{c\},\{d\},\{e\}\}$
- $\{\{a,e\},\{b\},\{c\},\{d\}\}$

Example: Multiple Merges [2/2]

For a small network as in the example, we detect the underlying community structure in one iteration.

Our communities evolve in the following manner.

- $\{\{a\},\{b\},\{c\},\{d\},\{e\}\}$
- $\{\{a,e\},\{b\},\{c\},\{d\}\}$
- $\{\{a,e\},\{b,c\},\{d\}\}$

Example: Multiple Merges [2/2]

For a small network as in the example, we detect the underlying community structure in one iteration.

Our communities evolve in the following manner.

- $\{\{a\},\{b\},\{c\},\{d\},\{e\}\}$
- $\{\{a,e\},\{b\},\{c\},\{d\}\}$
- $\{\{a,e\},\{b,c\},\{d\}\}$
- $\{\{a,e,d\},\{b,c\}\}$

Example: Multiple Merges [2/2]

For a small network as in the example, we detect the underlying community structure in one iteration.

Our communities evolve in the following manner.

- $\{\{a\},\{b\},\{c\},\{d\},\{e\}\}$
- $\{\{a,e\},\{b\},\{c\},\{d\}\}$
- $\{\{a,e\},\{b,c\},\{d\}\}$
- $\{\{a,e,d\},\{b,c\}\}$

Thus nodes a, d, e form one community, and nodes b, c another.

Naive Complexity [1/4]

Let us consider the algorithmic complexity of a naive modularity maximization algorithm.

Naive Complexity [1/4]

Let us consider the algorithmic complexity of a naive modularity maximization algorithm.

- Compute initial modularity.

Naive Complexity [1/4]

Let us consider the algorithmic complexity of a naive modularity maximization algorithm.

- Compute initial modularity.
  - Calculation of m: $\mathcal{O}(n^2)$.

Naive Complexity [1/4]

Let us consider the algorithmic complexity of a naive modularity maximization algorithm.

- Compute initial modularity.
  - Calculation of m: $\mathcal{O}(n^2)$.
    - n^2 adjacency matrix retrievals.

Naive Complexity [1/4]

Let us consider the algorithmic complexity of a naive modularity maximization algorithm.

- Compute initial modularity.
  - Calculation of m: $\mathcal{O}(n^2)$.
    - n^2 adjacency matrix retrievals.
    - $n^2 - 1$ additions.

Naive Complexity [1/4]

Let us consider the algorithmic complexity of a naive modularity maximization algorithm.

- Compute initial modularity.
  - Calculation of m: $\mathcal{O}(n^2)$.
    - n^2 adjacency matrix retrievals.
    - $n^2 - 1$ additions.
    - $\mathcal{O}(2n^2 - 1) = \mathcal{O}(n^2)$.

Naive Complexity [1/4]

Let us consider the algorithmic complexity of a naive modularity maximization algorithm.

- Compute initial modularity.
  - Calculation of m: $\mathcal{O}(n^2)$.
    - n^2 adjacency matrix retrievals.
    - $n^2 - 1$ additions.
    - $\mathcal{O}(2n^2 - 1) = \mathcal{O}(n^2)$.
  - Calculation of k_i^{in} or k_i^{out}: $\mathcal{O}(n)$.

Naive Complexity [1/4]

Let us consider the algorithmic complexity of a naive modularity maximization algorithm.

- Compute initial modularity.
  - Calculation of m: $\mathcal{O}(n^2)$.
    - n^2 adjacency matrix retrievals.
    - $n^2 - 1$ additions.
    - $\mathcal{O}(2n^2 - 1) = \mathcal{O}(n^2)$.
  - Calculation of k_i^{in} or k_i^{out}: $\mathcal{O}(n)$.
    - n adjacency matrix retrievals.

Naive Complexity [1/4]

Let us consider the algorithmic complexity of a naive modularity maximization algorithm.

- Compute initial modularity.
  - Calculation of m: $\mathcal{O}(n^2)$.
    - n^2 adjacency matrix retrievals.
    - $n^2 - 1$ additions.
    - $\mathcal{O}(2n^2 - 1) = \mathcal{O}(n^2)$.
  - Calculation of k_i^{in} or k_i^{out}: $\mathcal{O}(n)$.
    - n adjacency matrix retrievals.
    - $n - 1$ additions.

Naive Complexity [1/4]

Let us consider the algorithmic complexity of a naive modularity maximization algorithm.

- Compute initial modularity.
  - Calculation of m: $\mathcal{O}(n^2)$.
    - n^2 adjacency matrix retrievals.
    - $n^2 - 1$ additions.
    - $\mathcal{O}(2n^2 - 1) = \mathcal{O}(n^2)$.
  - Calculation of k_i^{in} or k_i^{out}: $\mathcal{O}(n)$.
    - n adjacency matrix retrievals.
    - $n - 1$ additions.
    - $\mathcal{O}(2n - 1) = \mathcal{O}(n)$.

Naive Complexity [2/4]

Single subtraction, multiplication, and division operations happen in constant time: $\mathcal{O}(1)$.

Naive Complexity [2/4]

Single subtraction, multiplication, and division operations happen in constant time: $\mathcal{O}(1)$.

- Compute modularity contributions for each node.

Naive Complexity [2/4]

Single subtraction, multiplication, and division operations happen in constant time: $\mathcal{O}(1)$.

- Compute modularity contributions for each node.
  - Calculation of $\partial\vec{Q}_i$: $\mathcal{O}(n)$.

Naive Complexity [2/4]

Single subtraction, multiplication, and division operations happen in constant time: $\mathcal{O}(1)$.

- Compute modularity contributions for each node.
  - Calculation of $\partial\vec{Q}_i$: $\mathcal{O}(n)$.
    - 1 adjacency matrix retrieval for $\vec{A}_{i,i}$.

Naive Complexity [2/4]

Single subtraction, multiplication, and division operations happen in constant time: $\mathcal{O}(1)$.

- Compute modularity contributions for each node.
  - Calculation of $\partial\vec{Q}_i$: $\mathcal{O}(n)$.
    - 1 adjacency matrix retrieval for $\vec{A}_{i,i}$.
    - $4n - 2$ computations for finding k_i^{in} and k_i^{out}.

Naive Complexity [2/4]

Single subtraction, multiplication, and division operations happen in constant time: $\mathcal{O}(1)$.

- Compute modularity contributions for each node.
  - Calculation of $\partial\vec{Q}_i$: $\mathcal{O}(n)$.
    - 1 adjacency matrix retrieval for $\vec{A}_{i,i}$.
    - $4n-2$ computations for finding k_i^{in} and k_i^{out}.
    - 3 operations to compute the value within the sum.

Naive Complexity [2/4]

Single subtraction, multiplication, and division operations happen in constant time: $\mathcal{O}(1)$.

- Compute modularity contributions for each node.
  - Calculation of $\partial\vec{Q}_i$: $\mathcal{O}(n)$.
    - 1 adjacency matrix retrieval for $\vec{A}_{i,i}$.
    - $4n-2$ computations for finding k_i^{in} and k_i^{out}.
    - 3 operations to compute the value within the sum.
    - $\mathcal{O}(4n+1) = \mathcal{O}(n)$.

Naive Complexity [2/4]

Single subtraction, multiplication, and division operations happen in constant time: $\mathcal{O}(1)$.

- Compute modularity contributions for each node.
  - Calculation of $\partial\vec{Q}_i$: $\mathcal{O}(n)$.
    - 1 adjacency matrix retrieval for $\vec{A}_{i,i}$.
    - $4n-2$ computations for finding k_i^{in} and k_i^{out}.
    - 3 operations to compute the value within the sum.
    - $\mathcal{O}(4n+1) = \mathcal{O}(n)$.
  - Do such for each node: $\mathcal{O}(n^2)$.

Naive Complexity [2/4]

Single subtraction, multiplication, and division operations happen in constant time: $\mathcal{O}(1)$.

- Compute modularity contributions for each node.
  - Calculation of $\partial\vec{Q}_i$: $\mathcal{O}(n)$.
    - 1 adjacency matrix retrieval for $\vec{A}_{i,i}$.
    - $4n-2$ computations for finding k_i^{in} and k_i^{out}.
    - 3 operations to compute the value within the sum.
    - $\mathcal{O}(4n+1) = \mathcal{O}(n)$.
  - Do such for each node: $\mathcal{O}(n^2)$.
    - n nodes.

Naive Complexity [2/4]

Single subtraction, multiplication, and division operations happen in constant time: $\mathcal{O}(1)$.

- Compute modularity contributions for each node.
  - Calculation of $\partial\vec{Q}_i$: $\mathcal{O}(n)$.
    - 1 adjacency matrix retrieval for $\vec{A}_{i,i}$.
    - $4n-2$ computations for finding k_i^{in} and k_i^{out}.
    - 3 operations to compute the value within the sum.
    - $\mathcal{O}(4n+1) = \mathcal{O}(n)$.
  - Do such for each node: $\mathcal{O}(n^2)$.
    - n nodes.
    - $\mathcal{O}(n(n)) = \mathcal{O}(n^2)$.

Naive Complexity [2/4]

Single subtraction, multiplication, and division operations happen in constant time: $\mathcal{O}(1)$.

- Compute modularity contributions for each node.
  - Calculation of $\partial\vec{Q}_i$: $\mathcal{O}(n)$.
    - 1 adjacency matrix retrieval for $\vec{A}_{i,i}$.
    - $4n-2$ computations for finding k_i^{in} and k_i^{out}.
    - 3 operations to compute the value within the sum.
    - $\mathcal{O}(4n+1) = \mathcal{O}(n)$.
  - Do such for each node: $\mathcal{O}(n^2)$.
    - n nodes.
    - $\mathcal{O}(n(n)) = \mathcal{O}(n^2)$.
- Total computationof modularity: $\mathcal{O}(n^2)$.

Naive Complexity [2/4]

Single subtraction, multiplication, and division operations happen in constant time: $\mathcal{O}(1)$.

- Compute modularity contributions for each node.
  - Calculation of $\partial\vec{Q}_i$: $\mathcal{O}(n)$.
    - 1 adjacency matrix retrieval for $\vec{A}_{i,i}$.
    - $4n-2$ computations for finding k_i^{in} and k_i^{out}.
    - 3 operations to compute the value within the sum.
    - $\mathcal{O}(4n+1) = \mathcal{O}(n)$.
  - Do such for each node: $\mathcal{O}(n^2)$.
    - n nodes.
    - $\mathcal{O}(n(n)) = \mathcal{O}(n^2)$.
- Total computationof modularity: $\mathcal{O}(n^2)$.
  - initialization and computation of m and each $\partial\vec{Q}_i$.

Naive Complexity [2/4]

Single subtraction, multiplication, and division operations happen in constant time: $\mathcal{O}(1)$.

- Compute modularity contributions for each node.
  - Calculation of $\partial\vec{Q}_i$: $\mathcal{O}(n)$.
    - 1 adjacency matrix retrieval for $\vec{A}_{i,i}$.
    - $4n-2$ computations for finding k_i^{in} and k_i^{out}.
    - 3 operations to compute the value within the sum.
    - $\mathcal{O}(4n+1) = \mathcal{O}(n)$.
  - Do such for each node: $\mathcal{O}(n^2)$.
    - n nodes.
    - $\mathcal{O}(n(n)) = \mathcal{O}(n^2)$.
- Total computationof modularity: $\mathcal{O}(n^2)$.
  - initialization and computation of m and each $\partial\vec{Q}_i$.
  - $n-1$ additions: $\sum_i^n \partial\vec{Q}_i$.

Naive Complexity [2/4]

Single subtraction, multiplication, and division operations happen in constant time: $\mathcal{O}(1)$.

- Compute modularity contributions for each node.
  - Calculation of $\partial\vec{Q}_i$: $\mathcal{O}(n)$.
    - 1 adjacency matrix retrieval for $\vec{A}_{i,i}$.
    - $4n-2$ computations for finding k_i^{in} and k_i^{out}.
    - 3 operations to compute the value within the sum.
    - $\mathcal{O}(4n+1) = \mathcal{O}(n)$.
  - Do such for each node: $\mathcal{O}(n^2)$.
    - n nodes.
    - $\mathcal{O}(n(n)) = \mathcal{O}(n^2)$.
- Total computationof modularity: $\mathcal{O}(n^2)$.
  - initialization and computation of m and each $\partial\vec{Q}_i$.
  - $n-1$ additions: $\sum_i^n \partial\vec{Q}_i$.
  - 1 final multiplication.

Naive Complexity [2/4]

Single subtraction, multiplication, and division operations happen in constant time: $\mathcal{O}(1)$.

- Compute modularity contributions for each node.
  - Calculation of $\partial\vec{Q}_i$: $\mathcal{O}(n)$.
    - 1 adjacency matrix retrieval for $\vec{A}_{i,i}$.
    - $4n-2$ computations for finding k_i^{in} and k_i^{out}.
    - 3 operations to compute the value within the sum.
    - $\mathcal{O}(4n+1) = \mathcal{O}(n)$.
  - Do such for each node: $\mathcal{O}(n^2)$.
    - n nodes.
    - $\mathcal{O}(n(n)) = \mathcal{O}(n^2)$.
- Total computationof modularity: $\mathcal{O}(n^2)$.
  - initialization and computation of m and each $\partial\vec{Q}_i$.
  - $n-1$ additions: $\sum_i^n \partial\vec{Q}_i$.
  - 1 final multiplication.
  - $\mathcal{O}(n^2+n^2+n) = \mathcal{O}(n^2)$.

Naive Complexity [3/4]

Computing modularity alone is quite intensive, and how about computing the change in modularity?

Naive Complexity [3/4]

Computing modularity alone is quite intensive, and how about computing the change in modularity?

- Calculating $\Delta\vec{Q}$: $\mathcal{O}(n^2)$.

Naive Complexity [3/4]

Computing modularity alone is quite intensive, and how about computing the change in modularity?

- Calculating $\Delta\vec{Q}$: $\mathcal{O}(n^2)$.
  - Merging two nodes: $\mathcal{O}(n^2)$.

Naive Complexity [3/4]

Computing modularity alone is quite intensive, and how about computing the change in modularity?

- Calculating $\Delta\vec{Q}$: $\mathcal{O}(n^2)$.
  - Merging two nodes: $\mathcal{O}(n^2)$.
    - $4n - 2$ adjacency matrix retrievals node rows and columns.

Naive Complexity [3/4]

Computing modularity alone is quite intensive, and how about computing the change in modularity?

- Calculating $\Delta\vec{Q}$: $\mathcal{O}(n^2)$.
  - Merging two nodes: $\mathcal{O}(n^2)$.
    - $4n - 2$ adjacency matrix retrievals node rows and columns.
    - $2n - 1$ additions.

Naive Complexity [3/4]

Computing modularity alone is quite intensive, and how about computing the change in modularity?

- Calculating $\Delta\vec{Q}$: $\mathcal{O}(n^2)$.
  - Merging two nodes: $\mathcal{O}(n^2)$.
    - $4n - 2$ adjacency matrix retrievals node rows and columns.
    - $2n - 1$ additions.
    - $(n-1)^2$ insertions to form a merged matrix representation.

Naive Complexity [3/4]

Computing modularity alone is quite intensive, and how about computing the change in modularity?

- Calculating $\Delta\vec{Q}$: $\mathcal{O}(n^2)$.
  - Merging two nodes: $\mathcal{O}(n^2)$.
    - $4n-2$ adjacency matrix retrievals node rows and columns.
    - $2n-1$ additions.
    - $(n-1)^2$ insertions to form a merged matrix representation.
    - recalculating $\vec{Q}$ for matrix representation of merged.

Naive Complexity [3/4]

Computing modularity alone is quite intensive, and how about computing the change in modularity?

- Calculating $\Delta\vec{Q}$: $\mathcal{O}(n^2)$.
  - Merging two nodes: $\mathcal{O}(n^2)$.
    - $4n-2$ adjacency matrix retrievals node rows and columns.
    - $2n-1$ additions.
    - $(n-1)^2$ insertions to form a merged matrix representation.
    - recalculating $\vec{Q}$ for matrix representation of merged.
    - $\mathcal{O}(6n-3+(n-1)^2+n^2)=\mathcal{O}(n^2)$.

Naive Complexity [3/4]

Computing modularity alone is quite intensive, and how about computing the change in modularity?

- Calculating $\Delta\vec{Q}$: $\mathcal{O}(n^2)$.
  - Merging two nodes: $\mathcal{O}(n^2)$.
    - $4n-2$ adjacency matrix retrievals node rows and columns.
    - $2n-1$ additions.
    - $(n-1)^2$ insertions to form a merged matrix representation.
    - recalculating $\vec{Q}$ for matrix representation of merged.
    - $\mathcal{O}(6n-3+(n-1)^2+n^2)=\mathcal{O}(n^2)$.
  - Do such for each possible merge: $\mathcal{O}(mn^2)$.

Naive Complexity [3/4]

Computing modularity alone is quite intensive, and how about computing the change in modularity?

- Calculating $\Delta\vec{Q}$: $\mathcal{O}(n^2)$.
  - Merging two nodes: $\mathcal{O}(n^2)$.
    - $4n-2$ adjacency matrix retrievals node rows and columns.
    - $2n-1$ additions.
    - $(n-1)^2$ insertions to form a merged matrix representation.
    - recalculating $\vec{Q}$ for matrix representation of merged.
    - $\mathcal{O}(6n-3+(n-1)^2+n^2)=\mathcal{O}(n^2)$.
  - Do such for each possible merge: $\mathcal{O}(mn^2)$.
    - m edges.

Naive Complexity [3/4]

Computing modularity alone is quite intensive, and how about computing the change in modularity?

- Calculating $\Delta\vec{Q}$: $\mathcal{O}(n^2)$.
  - Merging two nodes: $\mathcal{O}(n^2)$.
    - $4n-2$ adjacency matrix retrievals node rows and columns.
    - $2n-1$ additions.
    - $(n-1)^2$ insertions to form a merged matrix representation.
    - recalculating $\vec{Q}$ for matrix representation of merged.
    - $\mathcal{O}(6n-3+(n-1)^2+n^2)=\mathcal{O}(n^2)$.
  - Do such for each possible merge: $\mathcal{O}(mn^2)$.
    - m edges.
    - $\mathcal{O}(m(n^2))=\mathcal{O}(mn^2)$.

Naive Complexity [4/4]

Computing modularity alone is quite intensive, and how about computing the change in modularity?

Naive Complexity [4/4]

Computing modularity alone is quite intensive, and how about computing the change in modularity?

- Total community detection: $\mathcal{O}(mn^3)$.

Naive Complexity [4/4]

Computing modularity alone is quite intensive, and how about computing the change in modularity?

- Total community detection: $\mathcal{O}(mn^3)$.
  - Calculate best $\Delta\vec{Q}_i$: $\mathcal{O}(mn^2)$.

Naive Complexity [4/4]

Computing modularity alone is quite intensive, and how about computing the change in modularity?

- Total community detection: $\mathcal{O}(mn^3)$.
  - Calculate best $\Delta\vec{Q}_i$: $\mathcal{O}(mn^2)$.
    - Calculate all $\Delta\vec{Q}_i$.

Naive Complexity [4/4]

Computing modularity alone is quite intensive, and how about computing the change in modularity?

- Total community detection: $\mathcal{O}(mn^3)$.
  - Calculate best $\Delta\vec{Q}_i$: $\mathcal{O}(mn^2)$.
    - Calculate all $\Delta\vec{Q}_i$.
    - Select best $\Delta\vec{Q}_i$ of n total.

Naive Complexity [4/4]

Computing modularity alone is quite intensive, and how about computing the change in modularity?

- Total community detection: $\mathcal{O}(mn^3)$.
  - Calculate best $\Delta\vec{Q}_i$: $\mathcal{O}(mn^2)$.
    - Calculate all $\Delta\vec{Q}_i$.
    - Select best $\Delta\vec{Q}_i$ of n total.
    - $\mathcal{O}(n + mn^2) = \mathcal{O}(mn^2)$.

Naive Complexity [4/4]

Computing modularity alone is quite intensive, and how about computing the change in modularity?

- Total community detection: $\mathcal{O}(mn^3)$.
  - Calculate best $\Delta\vec{Q}_i$: $\mathcal{O}(mn^2)$.
    - Calculate all $\Delta\vec{Q}_i$.
    - Select best $\Delta\vec{Q}_i$ of n total.
    - $\mathcal{O}(n + mn^2) = \mathcal{O}(mn^2)$.
  - Run for each possible node merge.

Naive Complexity [4/4]

Computing modularity alone is quite intensive, and how about computing the change in modularity?

- Total community detection: $\mathcal{O}(mn^3)$.
  - Calculate best $\Delta\vec{Q}_i$: $\mathcal{O}(mn^2)$.
    - Calculate all $\Delta\vec{Q}_i$.
    - Select best $\Delta\vec{Q}_i$ of n total.
    - $\mathcal{O}(n + mn^2) = \mathcal{O}(mn^2)$.
  - Run for each possible node merge.
    - $n - 1$ possible merges.

Naive Complexity [4/4]

Computing modularity alone is quite intensive, and how about computing the change in modularity?

- Total community detection: $\mathcal{O}(mn^3)$.
  - Calculate best $\Delta\vec{Q}_i$: $\mathcal{O}(mn^2)$.
    - Calculate all $\Delta\vec{Q}_i$.
    - Select best $\Delta\vec{Q}_i$ of n total.
    - $\mathcal{O}(n + mn^2) = \mathcal{O}(mn^2)$.
  - Run for each possible node merge.
    - $n - 1$ possible merges.
    - $\mathcal{O}((n-1)(mn^2)) = \mathcal{O}(mn^3)$.

Conclusion

We truly covered a lot, and yet this is only a glimpse of community detection algorithms.

Conclusion

We truly covered a lot, and yet this is only a glimpse of community detection algorithms.

Papers are still coming out on the topic, and new methods are being invented.

Conclusion

We truly covered a lot, and yet this is only a glimpse of community detection algorithms.

Papers are still coming out on the topic, and new methods are being invented.

QUESTIONS?

casonk@umich.edu

References [1/4]

[1] Arenas, A., Fernández, A., and Gómez, S. Analysis of the structure of complex networks at different resolution levels. New Journal of Physics 10, 5 (May 2008).
[2] Barabási, A. L. Network Science. Cambridge University Press, 2016.
[3] Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment (Oct. 2008).
[4] Dugué, N., and Perez, A. Directed Louvain : maximizing modularity in directed networks. Research report, Université d'Orléans, Nov. 2015.
[5] Girvan, M., and Newman, M. E. J. Community structure in social and biological networks. Proceedings of the National Academy of Sciences 99, 12 (2002), 7821–7826.

References [2/4]

[6] Krackhardt, D., and Stern, R. N. Informal networks and organizational crises: An experimental simulation. Social Psychology Quarterly 51, 2 (1988), 123–140.
[7] Lambiotte, R., Delvenne, J.-C., and Barahona, M. Laplacian dynamics and multiscale modular structure in networks. arXiv 1 (12 2008).
[8] Lloyd, S. Least squares quantization in pcm. IEEE Transactions on Information Theory 28, 2 (1982), 129– 137.
[9] MacQueen, J. Some methods for classification and analysis of multivariate observations.
[10] Newman, M. E. J. The structure and function of complex networks. SIAM Review 45, 2 (2003), 167–256.

References [3/4]

[11] Newman, M. E. J. Analysis of weighted networks. Physical Review E 70, 5 (Nov. 2004).
[12] Newman, M. E. J. Fast algorithm for detecting community structure in networks. Physical Review E 69, 6 (Jun. 2004).
[13] Newman, M. E. J. Equivalence between modularity optimization and maximum likelihood methods for community detection. Physical Review E 94 (Nov. 2016).
[14] Newman, M. E. J. "Networks". Oxford University Press, Jul. 2018.
[15] Newman, M. E. J., Clauset, A., and Moore, C. Finding community structure in very large networks. Physical Review E 70, 6 (Dec. 2004).

References [4/4]

[16] Newman, M. E. J., and Girvan, M. Finding and evaluating community structure in networks. Physical review. E, Statistical, nonlinear, and soft matter physics 69 2 Pt 2 (2004).
[17] Newman, M. E. J., and Leicht, E. A. Community structure in directed networks. Physical Review Letters 100, 11 (Mar. 2008).
[18] Reichardt, J., and Bornholdt, S. Statistical mechanics of community detection. Physical Review E 74, 1 (Jul. 2006).
[19] Traag, V. A., Waltman, L., and van Eck, N. J. From louvain to leiden: guaranteeing well-connected communities. Scientific Reports 9, 1 (Mar. 2019), 5233.
[20] Yang, J., and Leskovec, J. Overlapping community detection at scale: A nonnegative matrix factorization approach. In Proceedings of the Sixth ACM International Conference on Web Search and Data Mining (New York, NY, USA, 2013), WSDM '13, Association for Computing Machinery, p. 587–596.