Lecture 21: Advanced Topics

Tim LaRock

larock.t@northeastern.edu

bit.ly/cs3000syllabus

Business

Midterm 2 Grades Posted

- Submit regrade requests via GradeScope ASAP

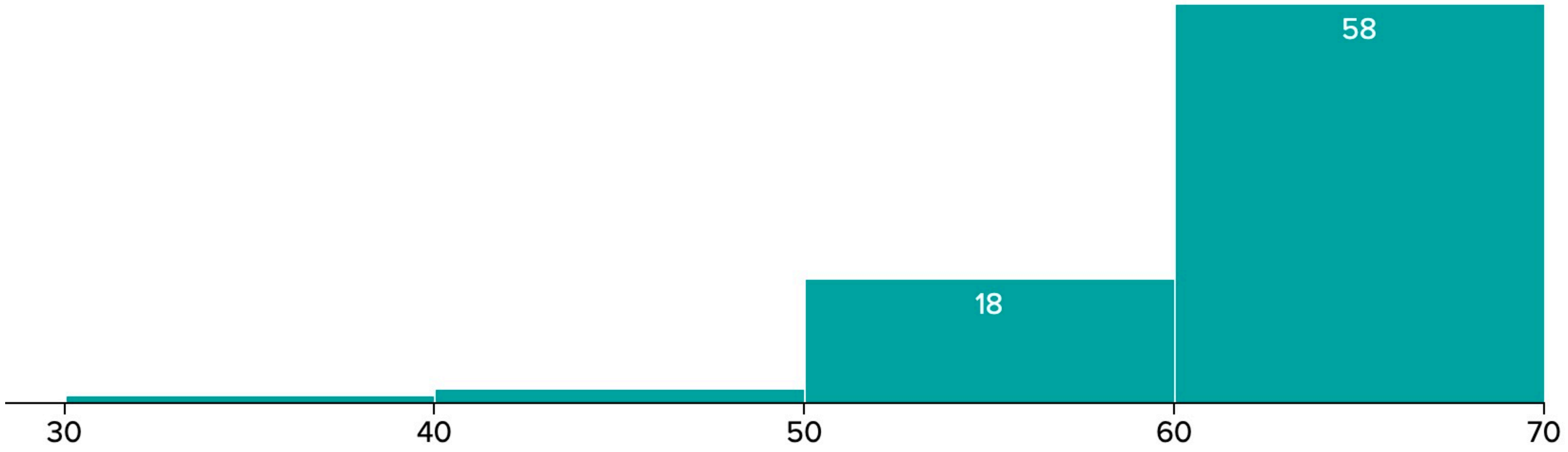
Extra Credit 1 & 2 both still available

- EC2 due tomorrow at 5PM
- EC1 due Sunday

Midterm 2

Grades were overall very good!

Median 63/70, mean 61/70



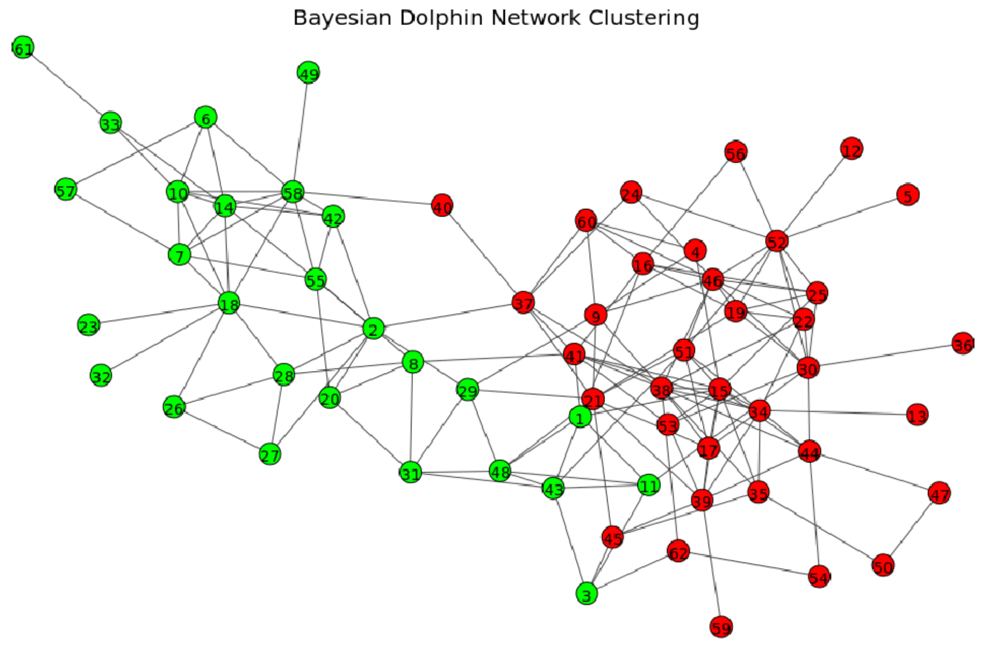
Final Exam

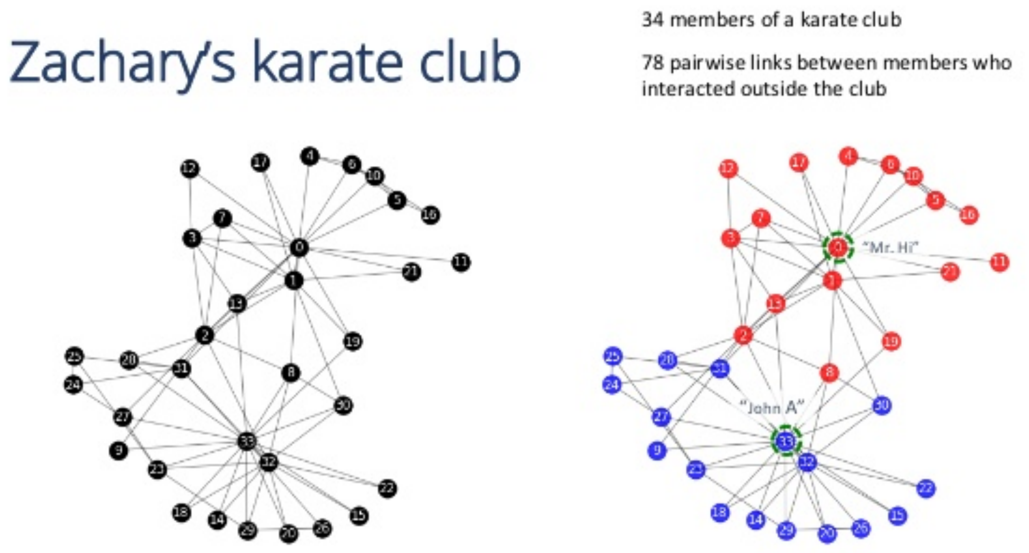
Review session/Q&A tomorrow during class

Exam released tomorrow night at 6PM and due Monday at midnight

Community Detection in Networks

Given a network structure, we want to partition the nodes into *communities*




Zachary's karate club
34 members of a karate club
78 pairwise links between members who interacted outside the club
"Mr. Hi"
"John A"

Girvan-Newman Approach

Community detection in early days relied on hierarchical clustering

- The same process we discussed yesterday, but keeping more data

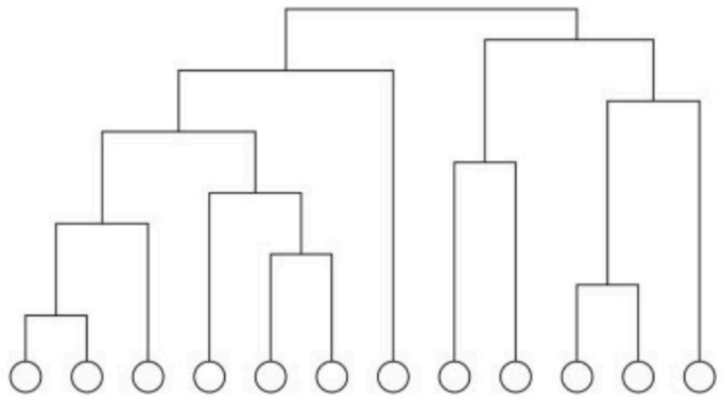


Fig. 2. An example of a small hierarchical clustering tree. The circles at the bottom represent the vertices in the network, and the tree shows the order in which they join together to form communities for a given definition of the weight W_{ij} of connections between vertex pairs.

Community structure in social and biological networks. PNAS, 2001.

Girvan-Newman Approach

Community detection in early days relied on hierarchical clustering

- The same process we discussed yesterday, but keeping more data

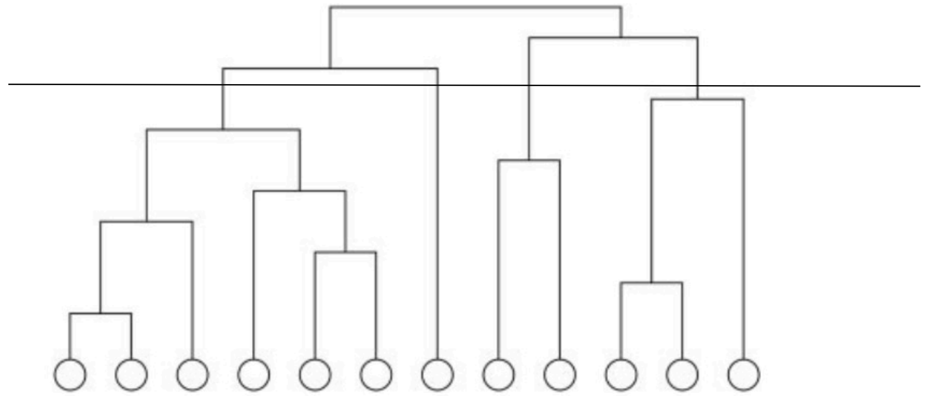


Fig. 2. An example of a small hierarchical clustering tree. The circles at the bottom represent the vertices in the network, and the tree shows the order in which they join together to form communities for a given definition of the weight W_{ij} of connections between vertex pairs.

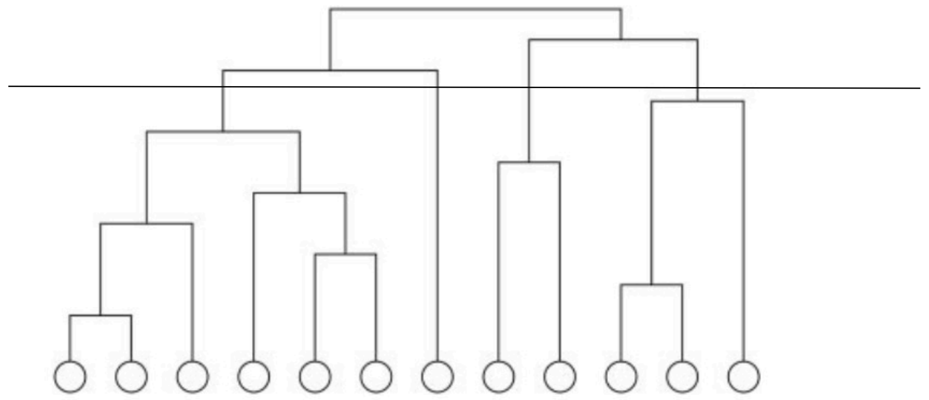
Our k-cluster algorithm just cuts this clustering tree such that we have k communities.

Community structure in social and biological networks. PNAS, 2001.

Girvan-Newman Approach

Community detection in early days relied on hierarchical clustering

- The same process we discussed yesterday, but keeping more data



Our k-cluster algorithm just cuts this clustering tree such that we have k communities.

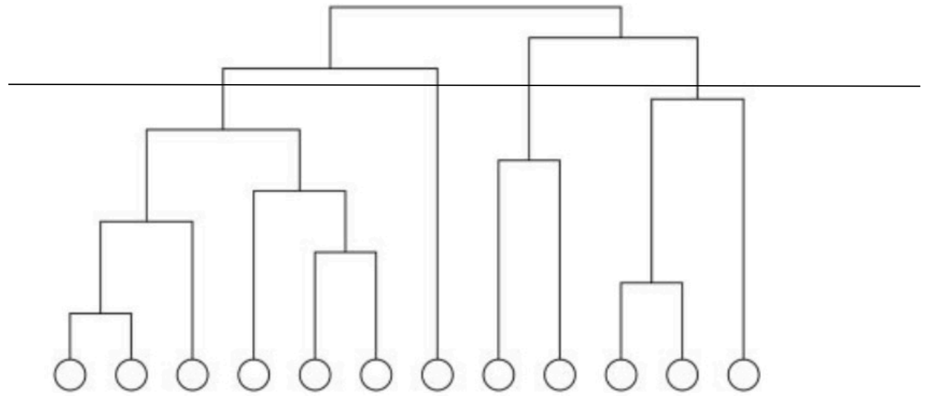
Fig. 2. An example of a small hierarchical clustering tree. The circles at the bottom represent the vertices in the network, and the tree shows the order in which they join together to form communities for a given definition of the weight W_{ij} of connections between vertex pairs.

The problem with this for networks is that it relies on the edge weights always being meaningful distances, which may not be the case!

Girvan-Newman Approach

Community detection in early days relied on hierarchical clustering

- The same process we discussed yesterday, but keeping more data



Our k-cluster algorithm just cuts this clustering tree such that we have k communities.

Fig. 2. An example of a small hierarchical clustering tree. The circles at the bottom represent the vertices in the network, and the tree shows the order in which they join together to form communities for a given definition of the weight W_{ij} of connections between vertex pairs.

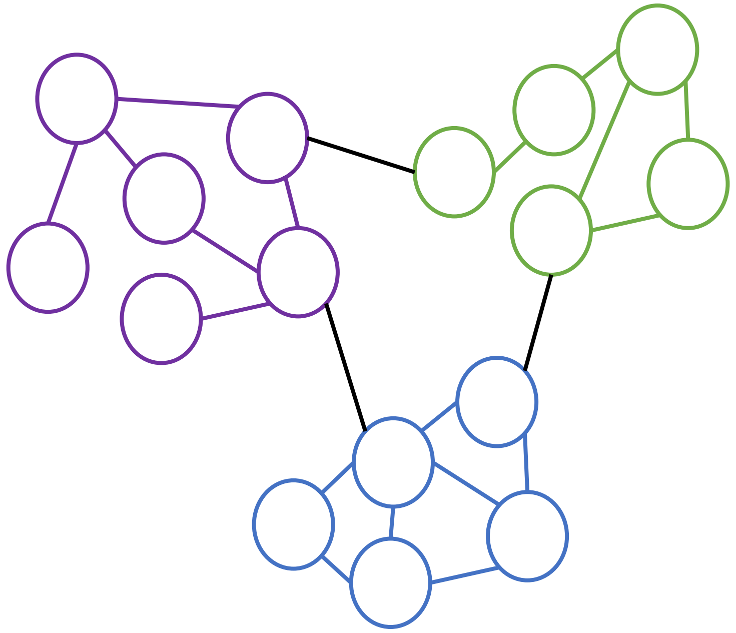
The problem with this for networks is that it relies on the edge weights always being meaningful distances, which may not be the case!

- Their solution: Define *edge betweenness centrality* and use that as the weight

Girvan-Newman Approach

Starting with the complete graph, repeat until no edges remain:

1. Compute *edge betweenness centrality* for every edge
2. Remove the edge with the highest betweenness



Community structure in social and biological networks. PNAS, 2001.

Girvan-Newman Approach

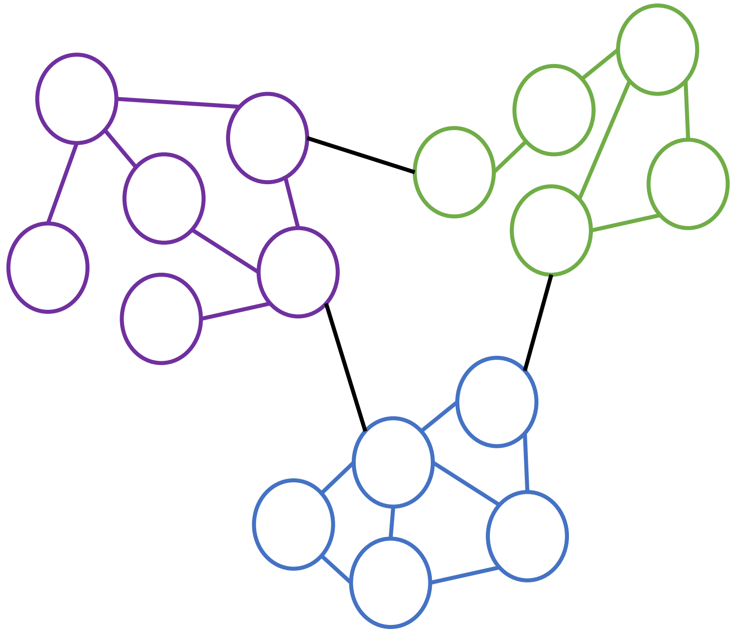
Starting with the complete graph, repeat until no edges remain:

1. Compute *edge betweenness centrality* for every edge
2. Remove the edge with the highest betweenness

Computing edge betweenness over and over again is expensive!

Assuming we run it once per m edges, the running time is at least $O(m^2 n)$.

However, the authors claim that in practice the algorithm can be sped up by making selective updates (some values will not change between iterations) and stopping before all edges are removed.



Community structure in social and biological networks. PNAS, 2001.

Modularity

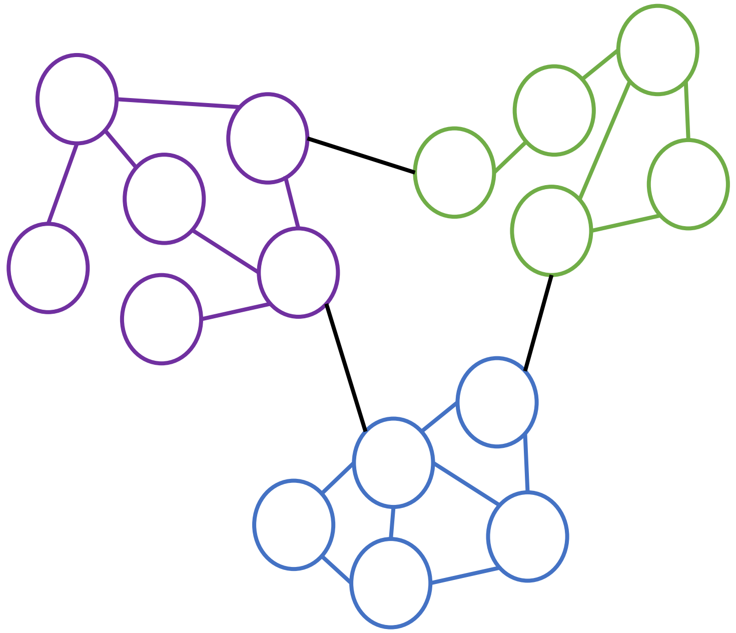
The Girvan-Newman algorithm is implicitly optimizing towards a notion of “internal connectedness”

Modularity is a measure of in-connections versus out-connections given a partitioning of the nodes in a graph into communities

Intuitively: How much more likely is it that a node in community i connects to others in its community rather than in some other community?

It is computed as:

$$Q = \frac{1}{2m}\sum_{i,j}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j)$$



Fast unfolding of communities in large networks. J. Stat. Mech., 2008

Louvain Algorithm

Find a partition that maximizes modularity! How?

1. Assign every node to its own community
2. Repeat until no increase in modularity is possible:
   a) For every node, merge communities with the neighbor that would maximally increase modularity (do nothing if there is no way to increase for that node)
3. Construct a new weighted graph G_c where the vertices are the communities and weighted links between them represent the sum of links between communities
4. Repeat 2-3 until no increase in modularity is possible

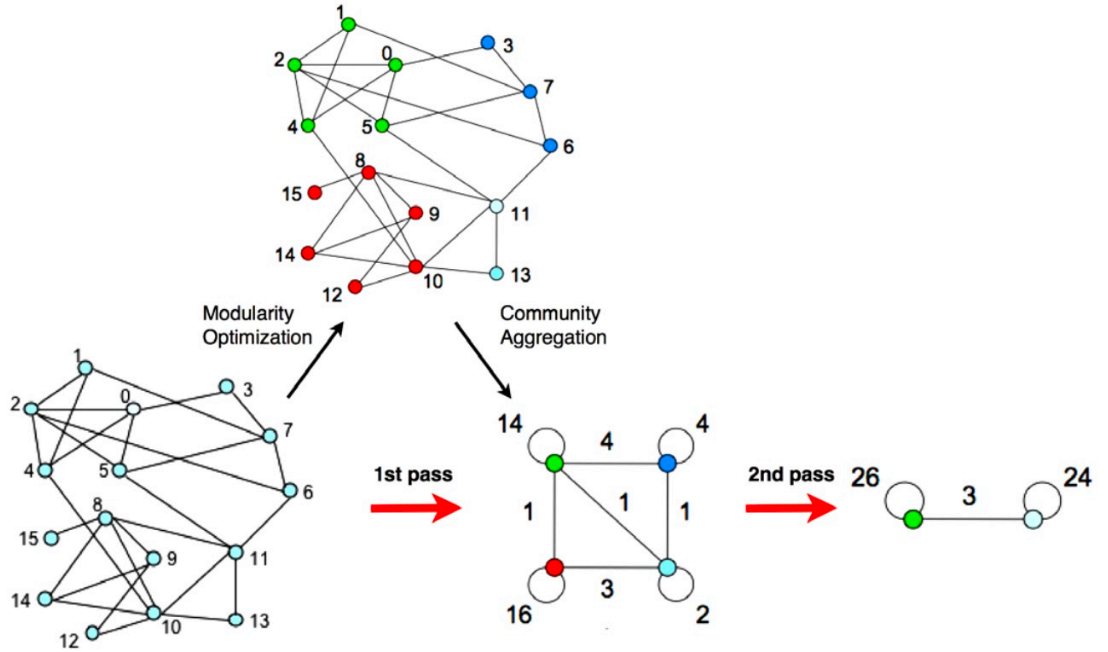
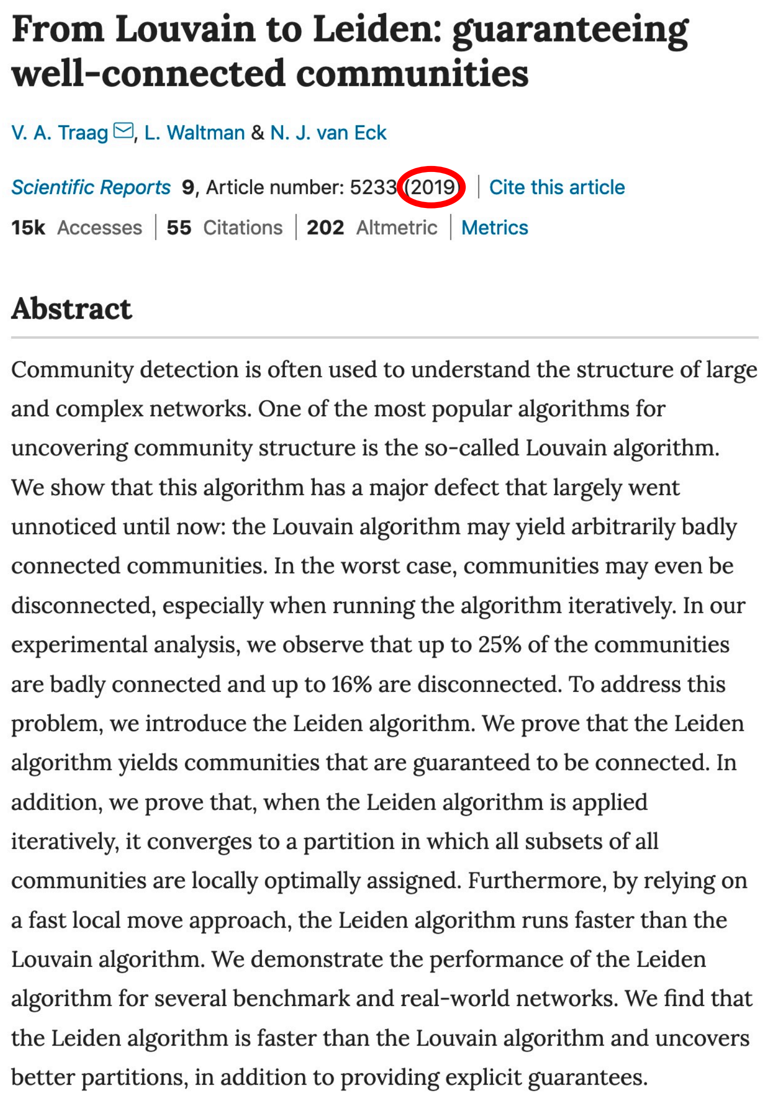


Figure 1. Visualization of the steps of our algorithm. Each pass is made of two phases: one where modularity is optimized by allowing only local changes of communities; one where the communities found are aggregated in order to build a new network of communities. The passes are repeated iteratively until no increase of modularity is possible.

Fast unfolding of communities in large networks. J. Stat. Mech., 2008

Louvain Algorithm

This is still an active area of research!



From Louvain to Leiden: guaranteeing well-connected communities

V. A. Traag, L. Waltman & N. J. van Eck

Scientific Reports **9**, Article number: 5233 (2019) | Cite this article

15k Accesses | **55** Citations | **202** Altmetric | Metrics

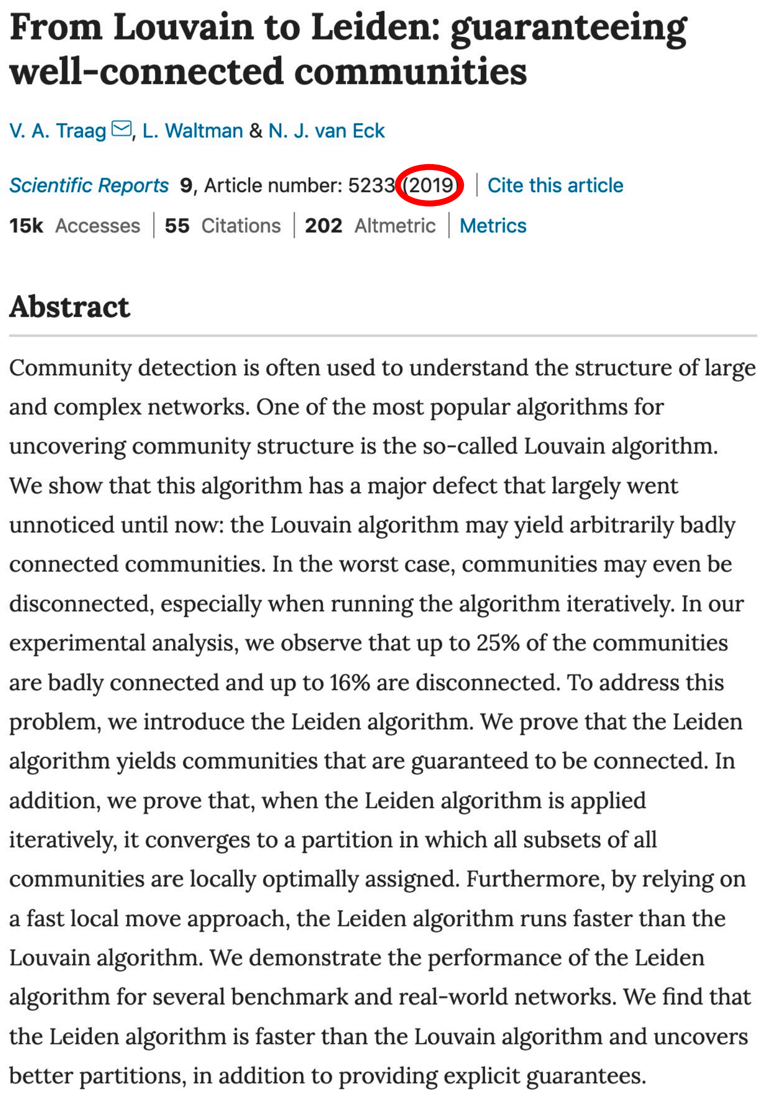
Abstract

Community detection is often used to understand the structure of large and complex networks. One of the most popular algorithms for uncovering community structure is the so-called Louvain algorithm. We show that this algorithm has a major defect that largely went unnoticed until now: the Louvain algorithm may yield arbitrarily badly connected communities. In the worst case, communities may even be disconnected, especially when running the algorithm iteratively. In our experimental analysis, we observe that up to 25% of the communities are badly connected and up to 16% are disconnected. To address this problem, we introduce the Leiden algorithm. We prove that the Leiden algorithm yields communities that are guaranteed to be connected. In addition, we prove that, when the Leiden algorithm is applied iteratively, it converges to a partition in which all subsets of all communities are locally optimally assigned. Furthermore, by relying on a fast local move approach, the Leiden algorithm runs faster than the Louvain algorithm. We demonstrate the performance of the Leiden algorithm for several benchmark and real-world networks. We find that the Leiden algorithm is faster than the Louvain algorithm and uncovers better partitions, in addition to providing explicit guarantees.

Researchers found last year that the Louvain algorithm can return partitionings with strange structure in some cases and have proposed an algorithm they claim is better (the Leiden algorithm).

Louvain Algorithm

This is still an active area of research!



From Louvain to Leiden: guaranteeing well-connected communities

V. A. Traag, L. Waltman & N. J. van Eck

Scientific Reports **9**, Article number: 5233 (2019) | Cite this article

15k Accesses | **55** Citations | **202** Altmetric | Metrics

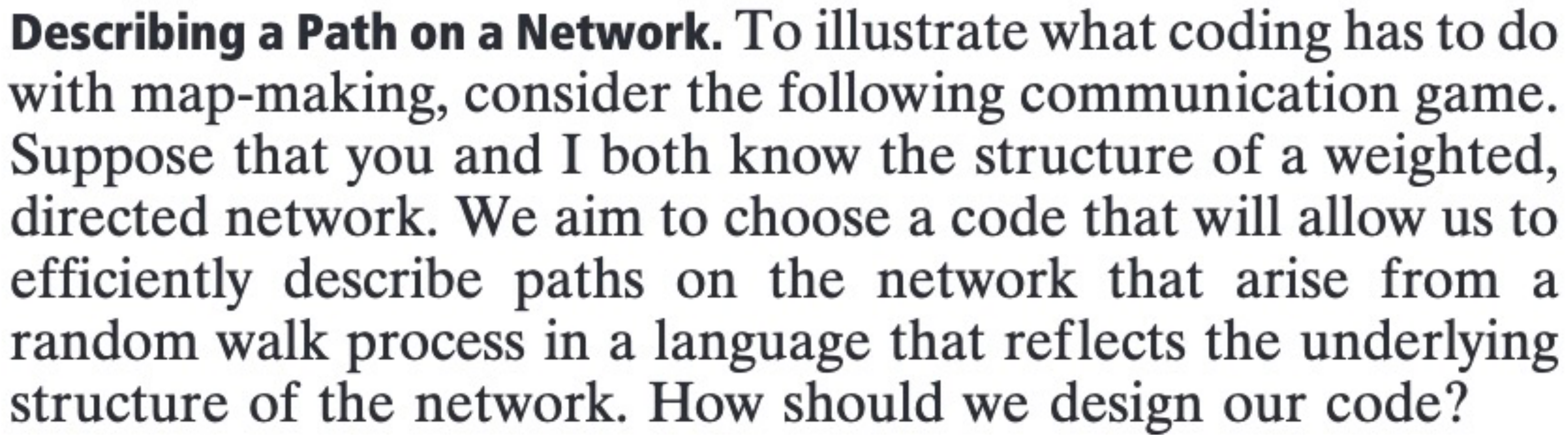
Abstract

Community detection is often used to understand the structure of large and complex networks. One of the most popular algorithms for uncovering community structure is the so-called Louvain algorithm. We show that this algorithm has a major defect that largely went unnoticed until now: the Louvain algorithm may yield arbitrarily badly connected communities. In the worst case, communities may even be disconnected, especially when running the algorithm iteratively. In our experimental analysis, we observe that up to 25% of the communities are badly connected and up to 16% are disconnected. To address this problem, we introduce the Leiden algorithm. We prove that the Leiden algorithm yields communities that are guaranteed to be connected. In addition, we prove that, when the Leiden algorithm is applied iteratively, it converges to a partition in which all subsets of all communities are locally optimally assigned. Furthermore, by relying on a fast local move approach, the Leiden algorithm runs faster than the Louvain algorithm. We demonstrate the performance of the Leiden algorithm for several benchmark and real-world networks. We find that the Leiden algorithm is faster than the Louvain algorithm and uncovers better partitions, in addition to providing explicit guarantees.

Researchers found last year that the Louvain algorithm can return partitionings with strange structure in some cases and have proposed an algorithm they claim is better (the Leiden algorithm).

InfoMap: Information Theoretic Communities

Yet another way to define communities is based on *information flow* between partitions of a network.



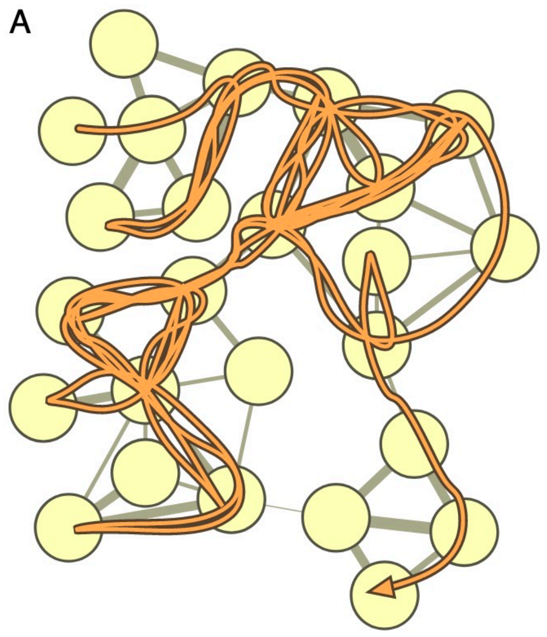
Describing a Path on a Network. To illustrate what coding has to do with map-making, consider the following communication game. Suppose that you and I both know the structure of a weighted, directed network. We aim to choose a code that will allow us to efficiently describe paths on the network that arise from a random walk process in a language that reflects the underlying structure of the network. How should we design our code?

Maps of random walks on complex networks reveal community structure. PNAS, 2008.

InfoMap: Information Theoretic Communities

Yet another way to define communities is based on *information flow* between partitions of a network.

A "random walk process" is what it sounds like: Start a "walker" on a random node, and let it choose which neighbor to visit randomly based on some probability distribution.



The intuition behind InfoMap is to give every node a symbol, then encode random walks using our old friend the Huffman Code!

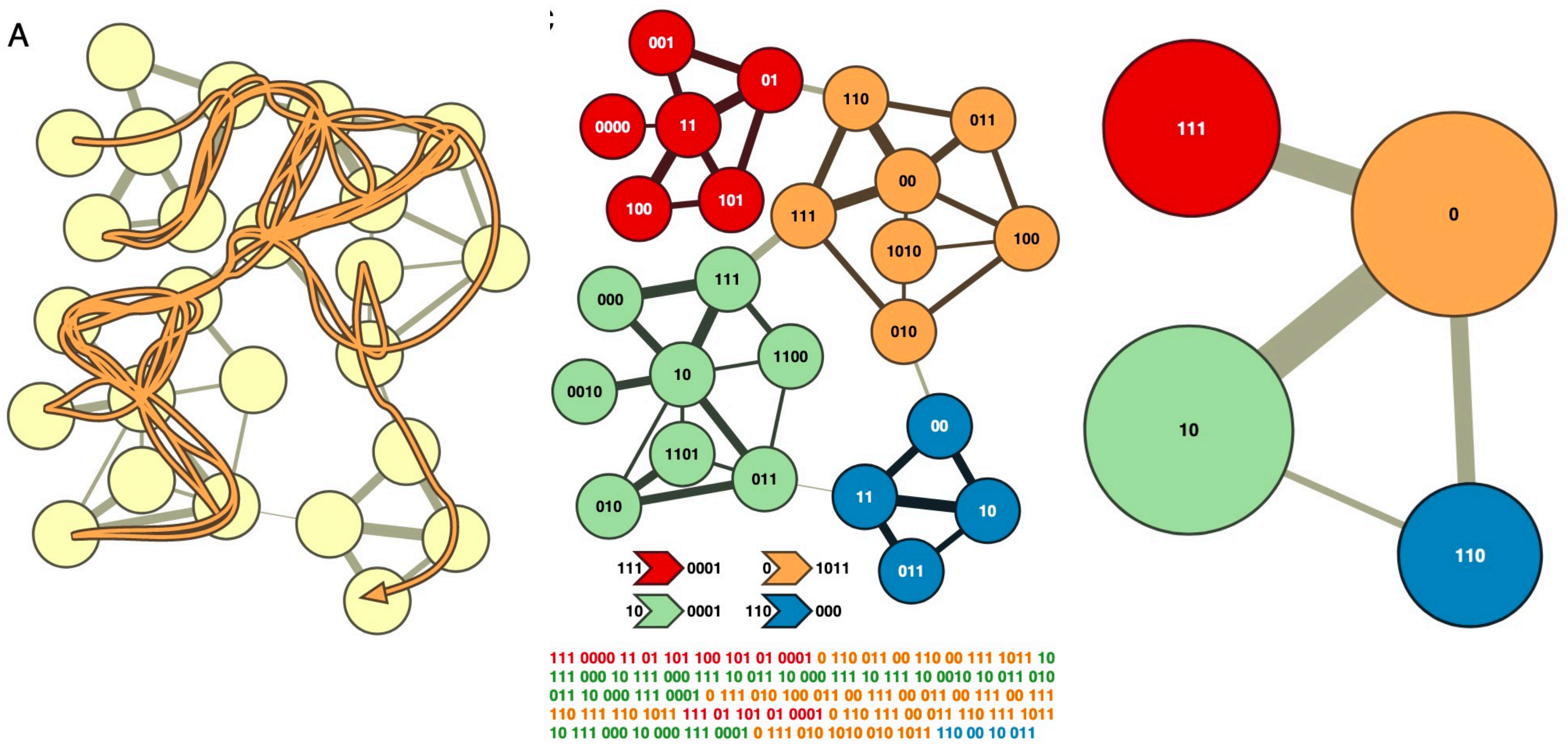
Maps of random walks on complex networks reveal community structure. PNAS, 2008.

InfoMap: Information Theoretic Communities

Idea: Treating each (partition, node) pair as a symbol, find the partitioning that minimizes the theoretical description length of a (very long or infinite) random walk through the graph structure.

The description length is computed at two levels: within a partition, and across partitions.

The algorithm starts with all nodes in their own partition, then greedily merges partitions that will lead to maximum lowering of the description length.

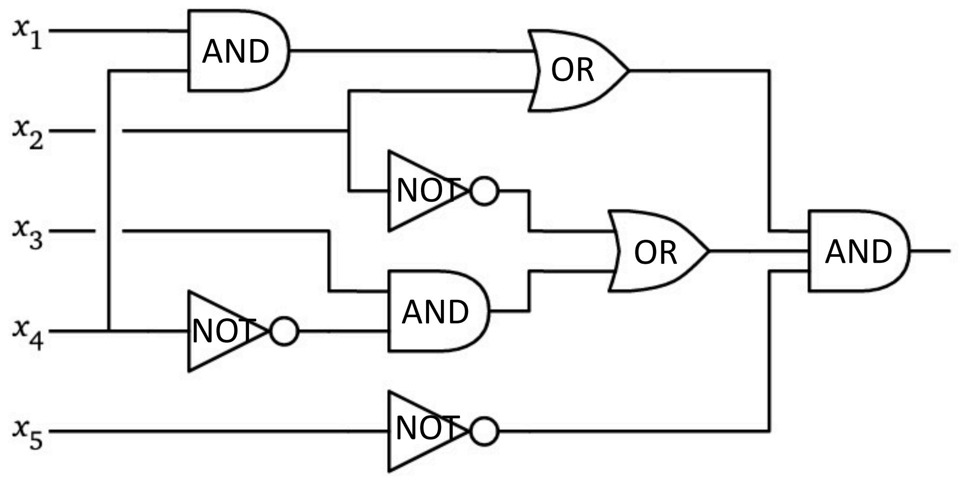


Maps of random walks on complex networks reveal community structure. PNAS, 2008.

Complexity

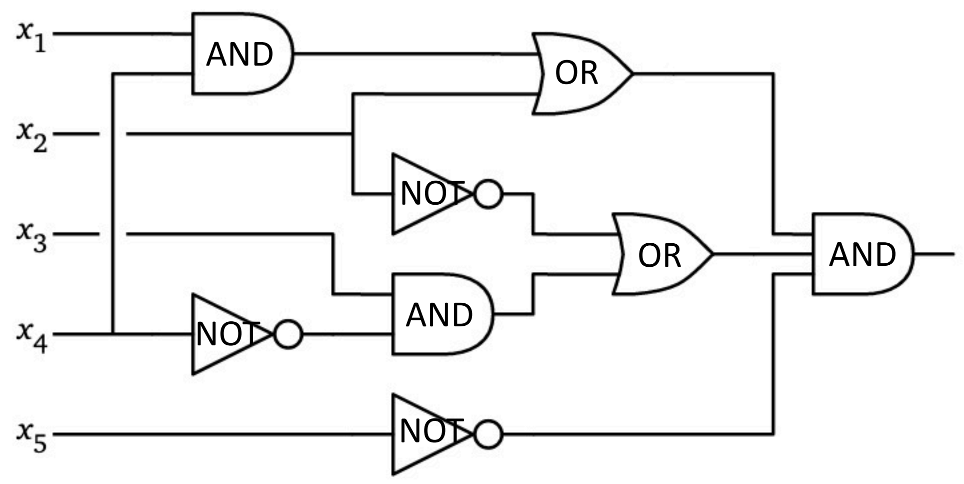
Complexity

Circuit Satisfiability: Is it possible to set the inputs of the n input switches such that the output is True?



Complexity

Circuit Satisfiability: Is it possible to set the inputs of the n input switches such that the output is True?

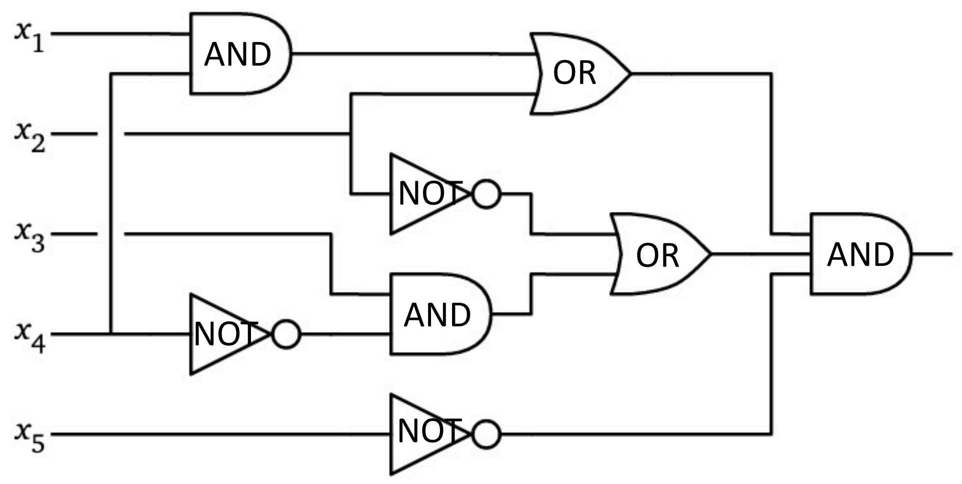


You are presented with a grand bargain: Solve the circuit satisfiability problem and earn $1 trillion dollars, or lose everything.

Complexity

Circuit Satisfiability: Is it possible to set the inputs of the n input switches such that the output is True?

Should you take the deal?



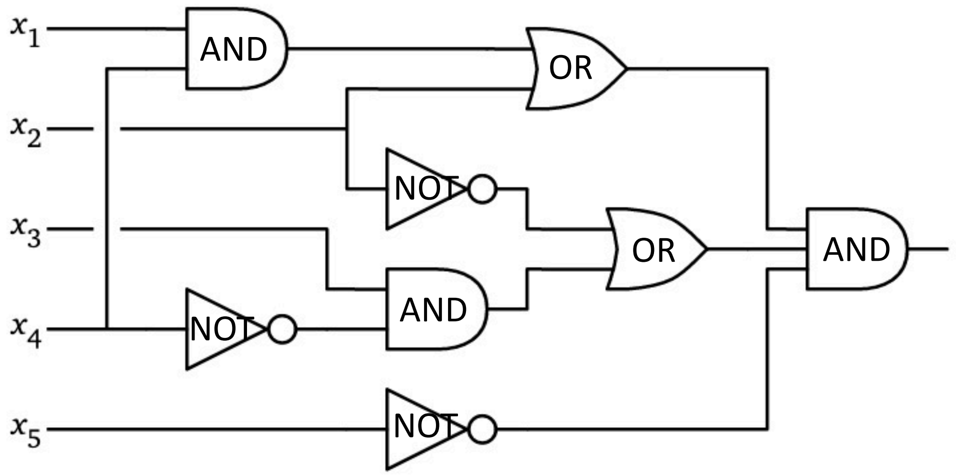
You are presented with a grand bargain: Solve the circuit satisfiability problem and earn $1 trillion dollars, or lose everything.

Complexity

Circuit Satisfiability: Is it possible to set the inputs of the n input switches such that the output is True?

Should you take the deal?

No!



There is no polynomial time algorithm for solving CircuitSat and there are 2^n possible inputs.

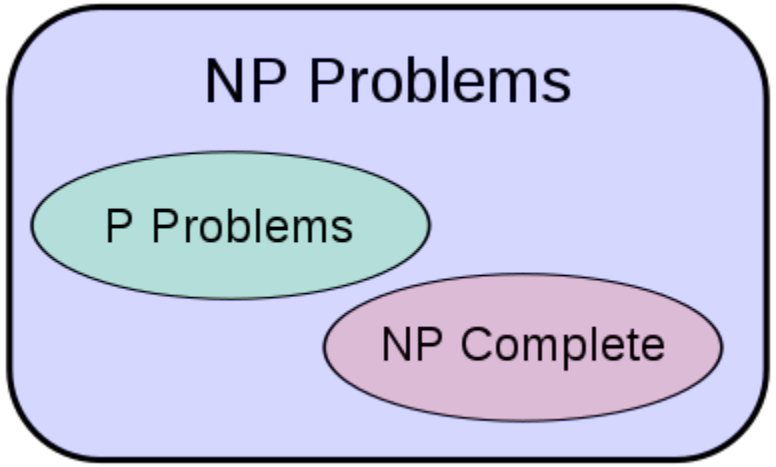
Since you are mortal, as n gets arbitrarily large you will run out of time on Earth before you can try every input!

You are presented with a grand bargain: Solve the circuit satisfiability problem and earn \$1 trillion dollars, or lose everything.

Some definitions

A *decision problem* is a problem whose output is a simple *Yes* or *No*

- $\boldsymbol{P}$ is the set of decision problems that can be solved in polynomial time
  - e.g. there is an algorithm that finds a solution in time $O(n^c)$ for some constant c
- $\boldsymbol{NP}$ is the set of decision problems whose solutions can be *verified* in polynomial time.
  - Every problem in P is in NP because we can verify a solution by running the polynomial time algorithm and checking the output



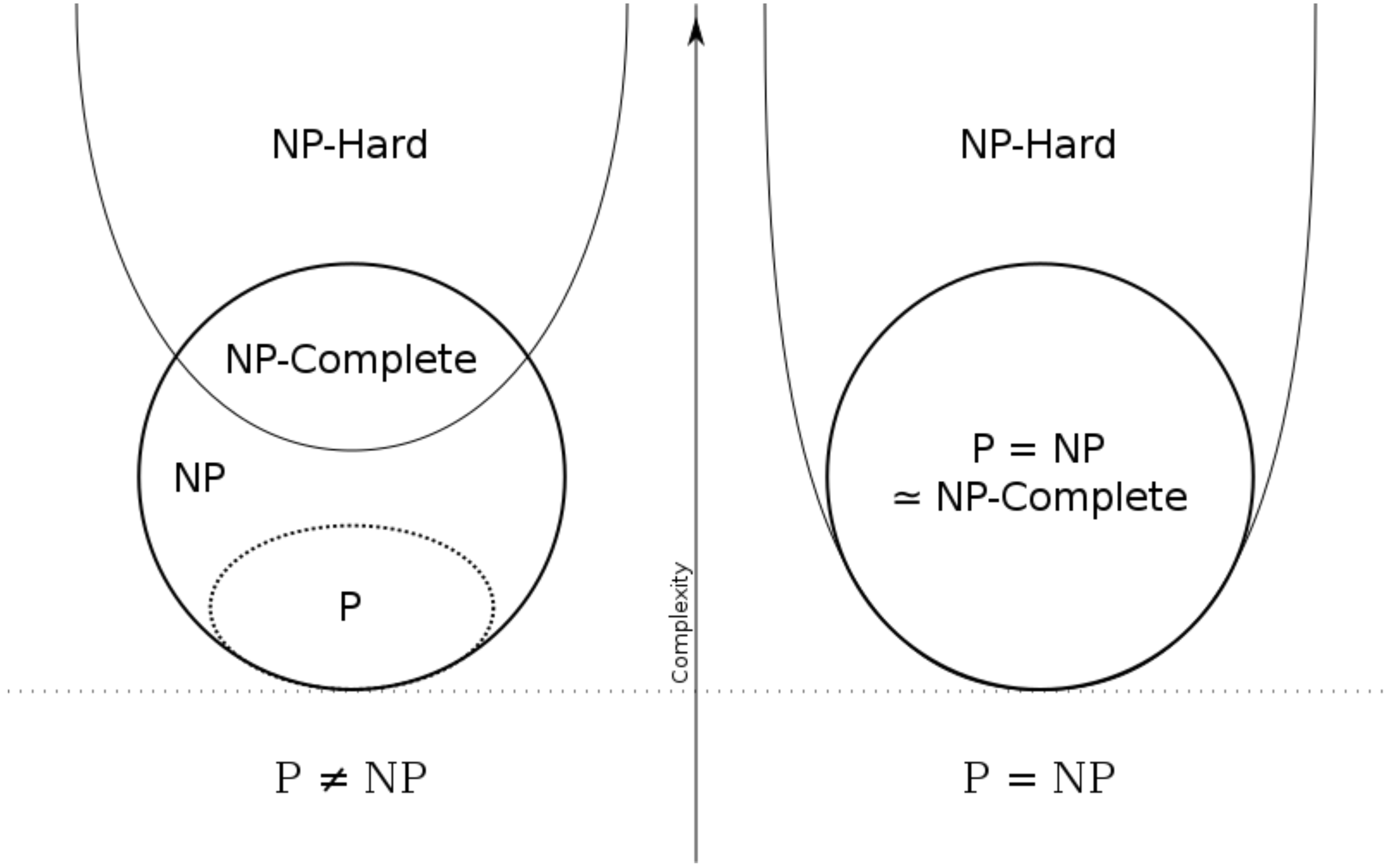
NP-hard and NP-complete

- A problem Π is **NP-hard** if it can be shown that if we had a polynomial time solution to Π we would have a solution to any problem in NP
  - Intuitively, if a problem is NP-hard it means it is *at least as hard as any other problem in NP*
  - Practically: To prove a problem Π is NP-hard, reduce a known NP-hard problem to Π
  - Your decision on the grand bargain would be made easier if you knew that **CircuitSat is NP-hard!**

- A problem Π is **NP-complete** if it is both in NP (solutions can be verified in polynomial time) and is proven to be NP-hard

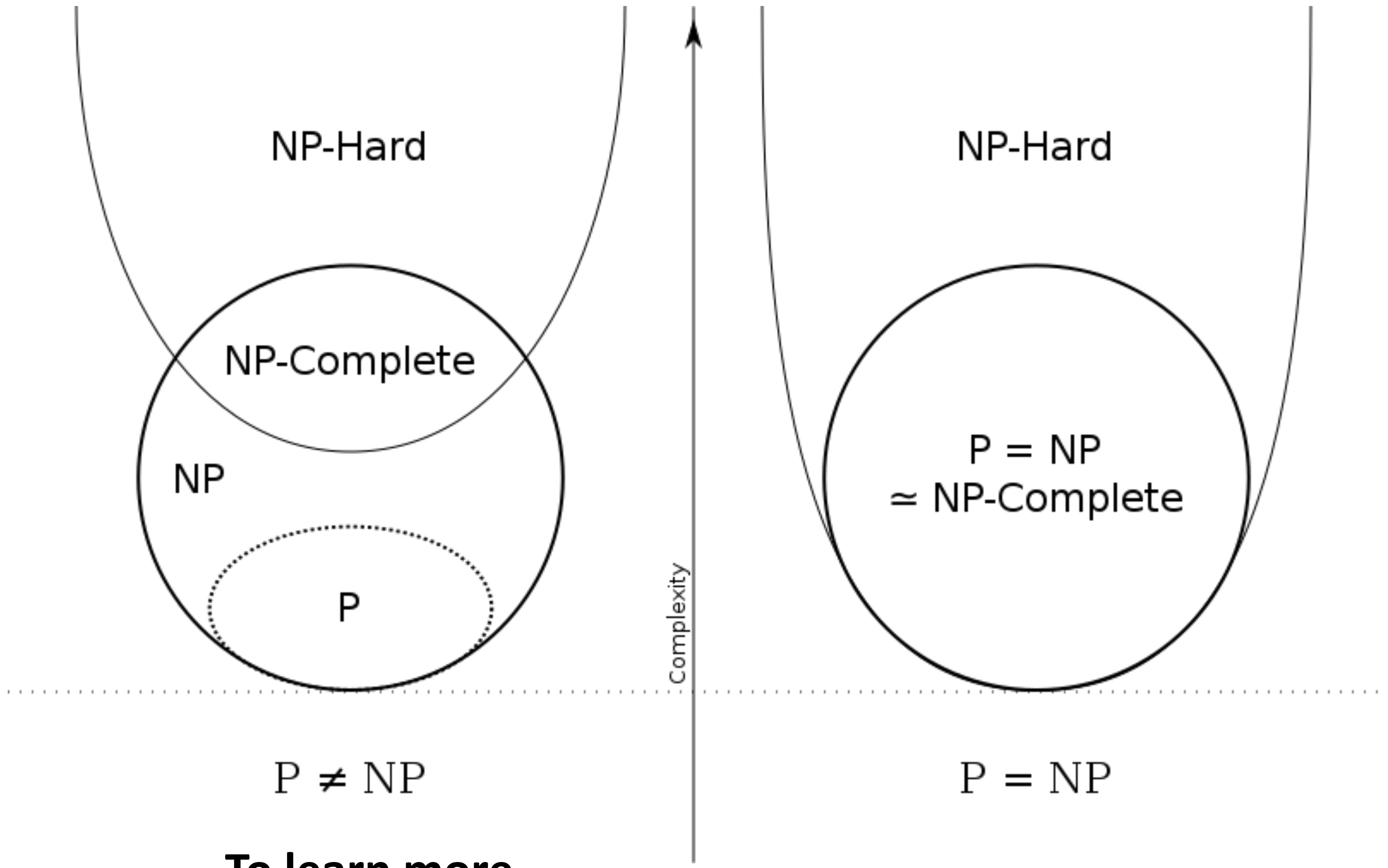
So what if $P = NP$?

1. We would be able to accept the grand bargain and emerge victorious with our $1 trillion.
2. Problems that are currently intractable to solve exactly efficiently – in particular, automated mathematical theorem proving –could potentially be solved efficiently. Many famous open problems in mathematics could likely be solved automatically if indeed $P = NP$.
3. Cryptography might just like, not work anymore
4. We would have an answer to one of the most interesting and consequential problems in modern science (and therefore the whole history of Computer Science).



So what if $P = NP$?

1. We would be able to accept the grand bargain and emerge victorious with our $1 trillion.
2. Problems that are currently intractable to solve exactly efficiently – in particular, automated mathematical theorem proving –could potentially be solved efficiently. Many famous open problems in mathematics could likely be solved automatically if indeed $P = NP$.
3. Cryptography might just like, not work anymore
4. We would have an answer to one of the most interesting and consequential problems in modern science (and therefore the whole history of Computer Science).



To learn more…
Read Erickson Chapter 12 and take a course on Theory of Computation (CS 3800)!

What we have covered in this course

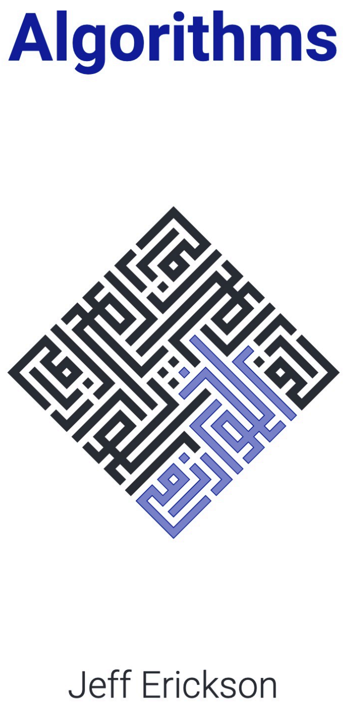
Asymptotic Analysis

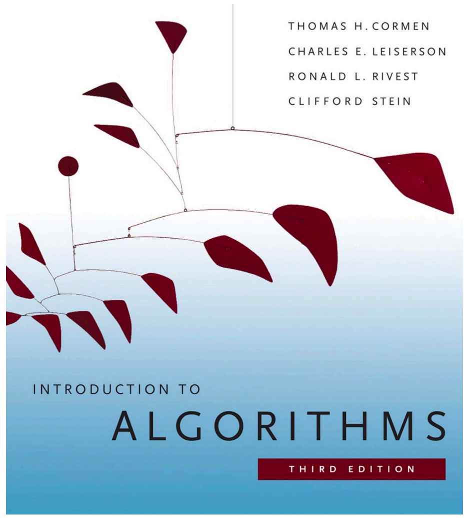
Divide and Conquer Algorithms
- Recursion/Backtracking
- Dynamic Programming

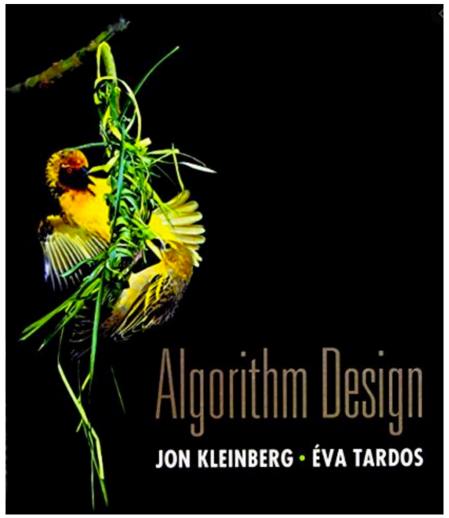
Graph Algorithms

Network Flow
- Reductions between algorithms

Greedy Algorithms
- Huffman Codes and entropy







What we did not cover

There are many topics related to algorithms that we did not cover in this class, including…

- Fairness, privacy, and ethics in algorithms
- Distributed/parallel algorithms
- Randomized algorithms
- Linear and convex optimization
- Numerical algorithms
- Algorithms for number theory
- Machine learning algorithms
- Algorithms for strategic agents
- Algorithms for quantum computers
- Naturally occurring algorithms

You can learn about these topics in more advanced courses in Khoury, online, or at other institutions in the future!

If any of them are particularly interesting to you and you want to learn more, you can send me an email and I will try to find you a starting point!

Wrap-up

This is our last lecture ☹ but not our last meeting!

Tomorrow: Final Exam Review Session/Q&A

Final Exam: Released tomorrow night 6PM, due Monday midnight