✉ lorenzo.dallamico@unito.it

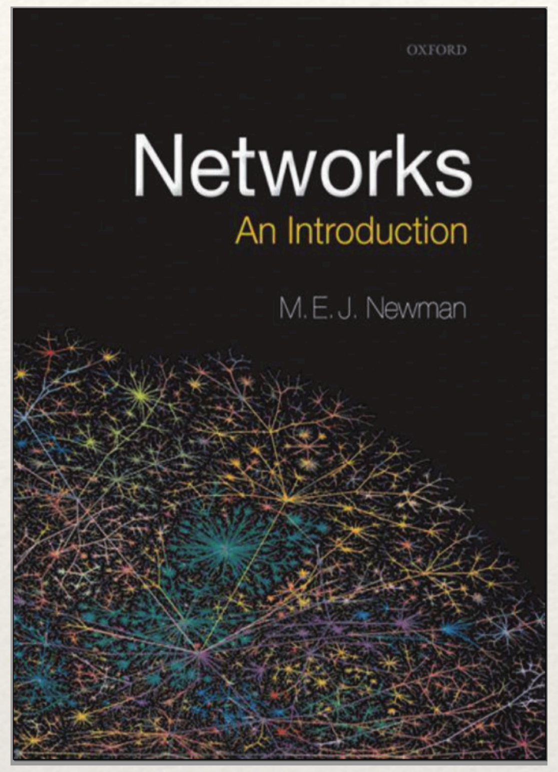
di.unito.it

Lecture 17ns12

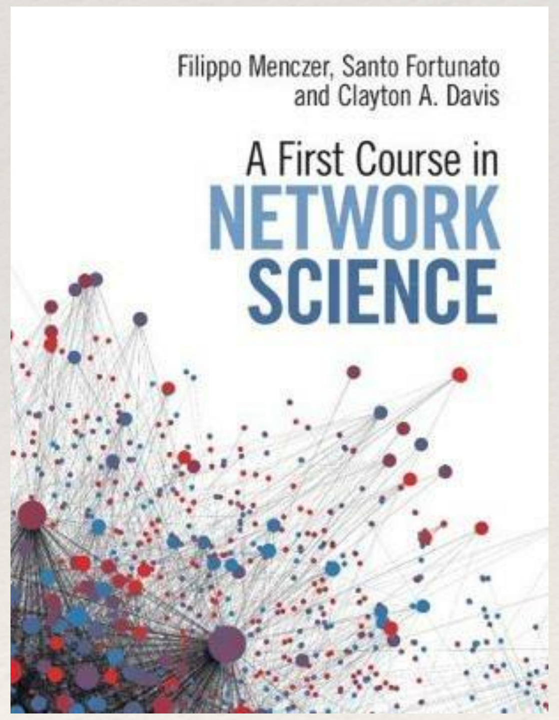
Communities 1

Course: **Complex Networks Analysis and Visualization**
Sub-Module: **NetSci**

References



Chapter 14



Chapter 6

Other useful references are

https://arxiv.org/pdf/0711.0189.pdf

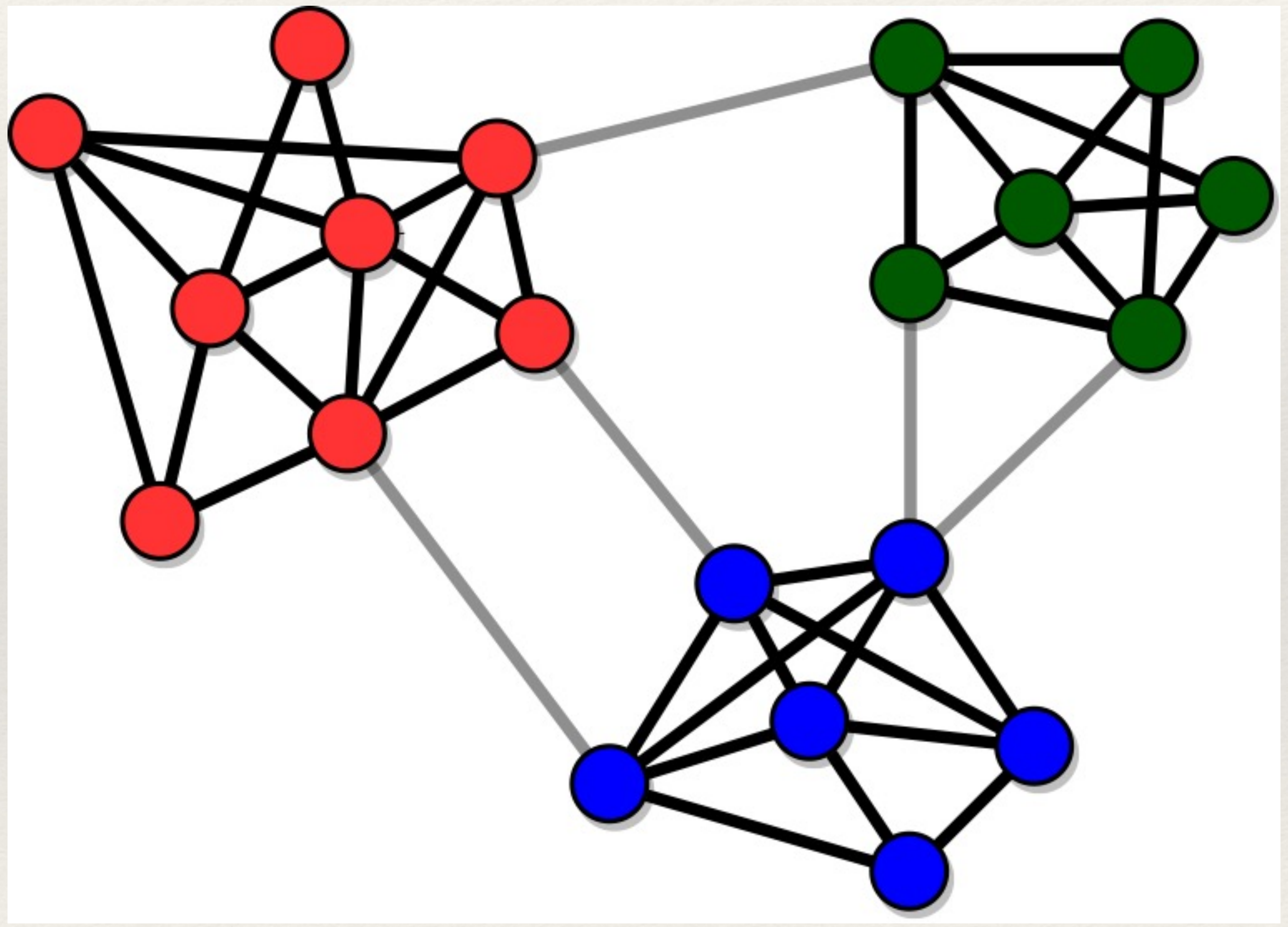
https://arxiv.org/pdf/0906.0612.pdf
(very broad reference)

https://arxiv.org/pdf/1702.00467.pdf
(for stochastic block modeling)

What are communities?

Community structure

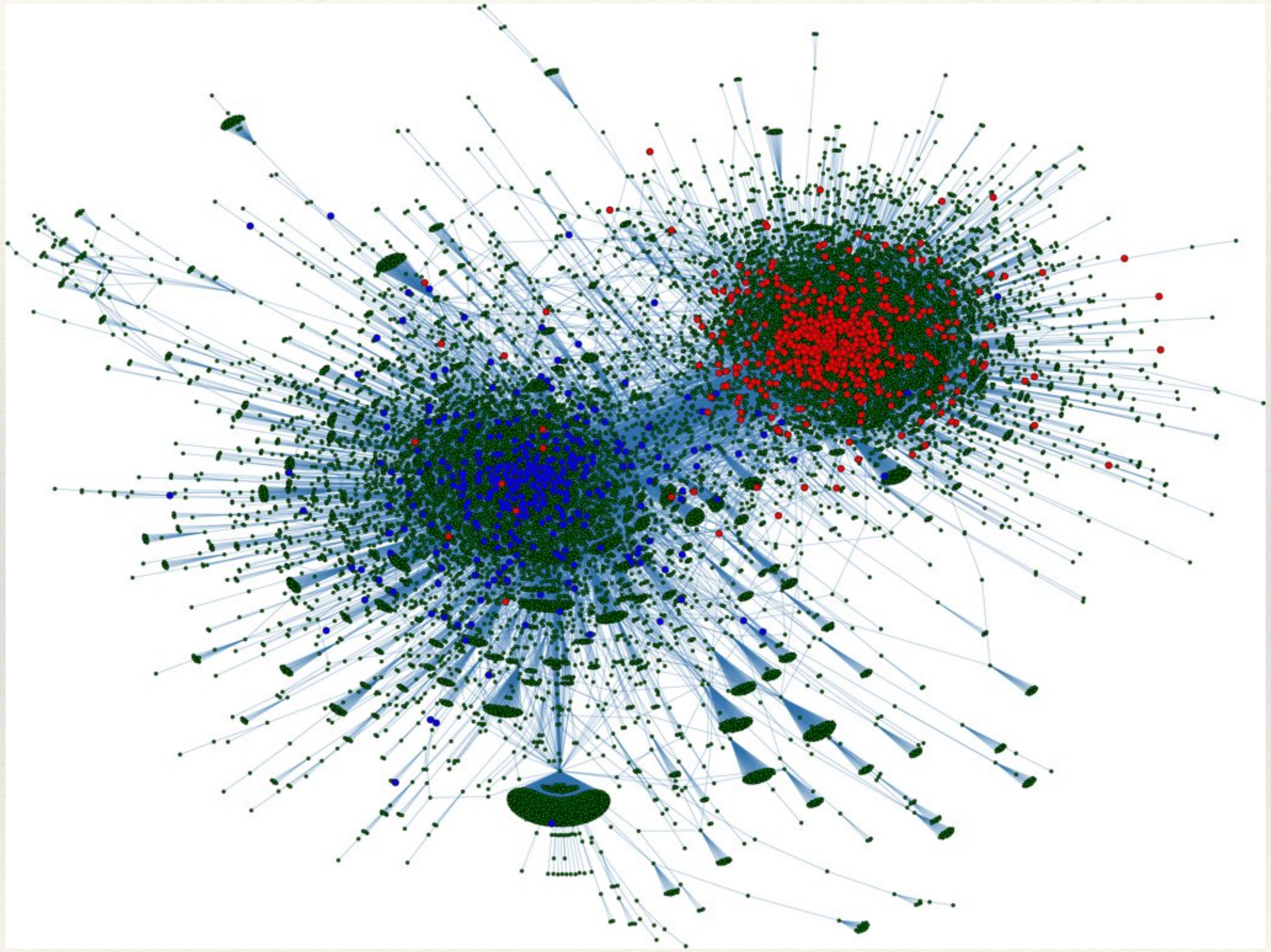
Communities (or clusters): sets of tightly connected nodes



Community structure

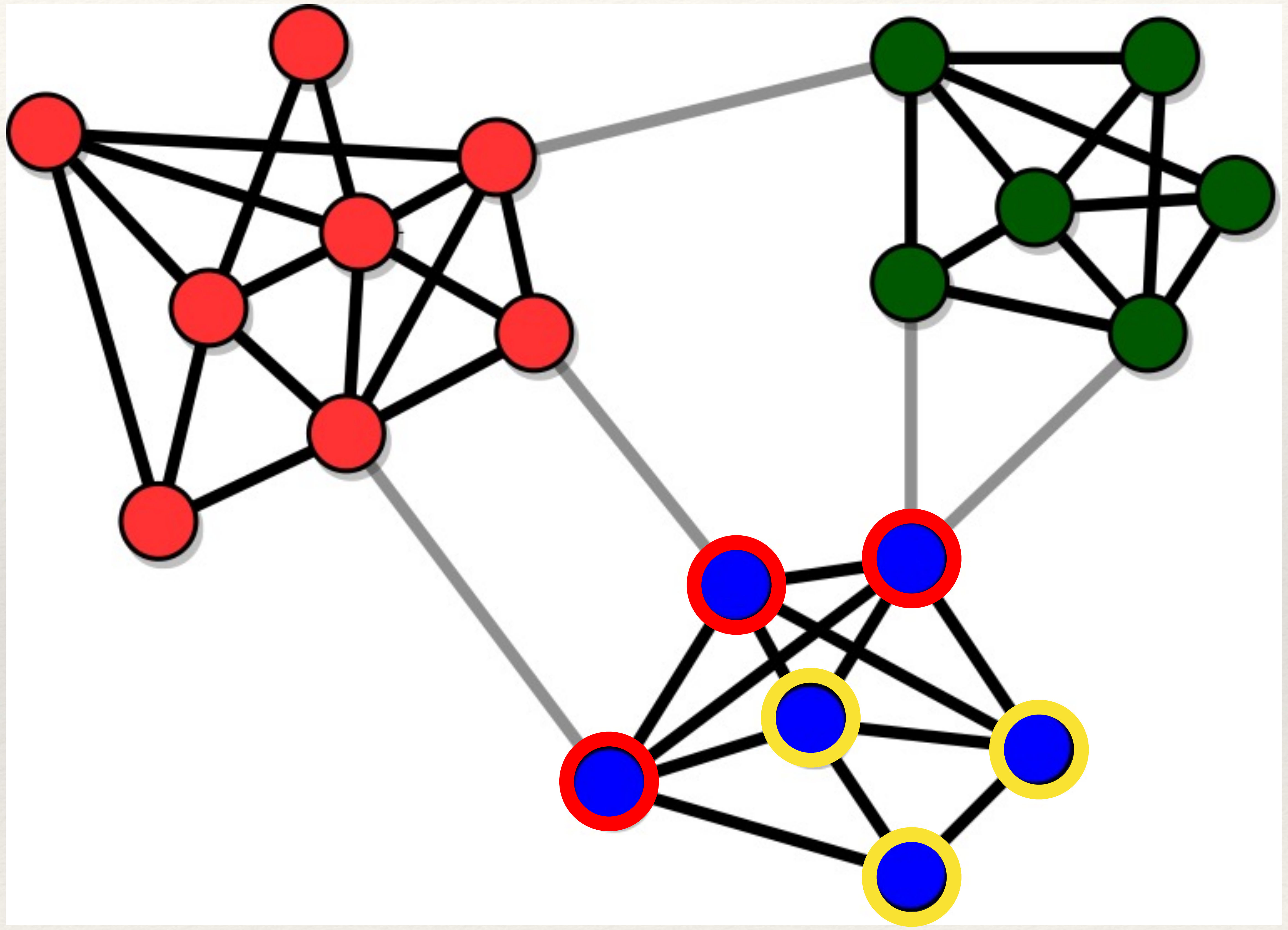
Example: Twitter users with strong political preferences tend to follow those aligned with them and not to follow users with different political orientation

Other examples: social circles in social networks, functional modules in protein interaction networks, groups of pages about the same topic on the Web, etc.



Why study communities?

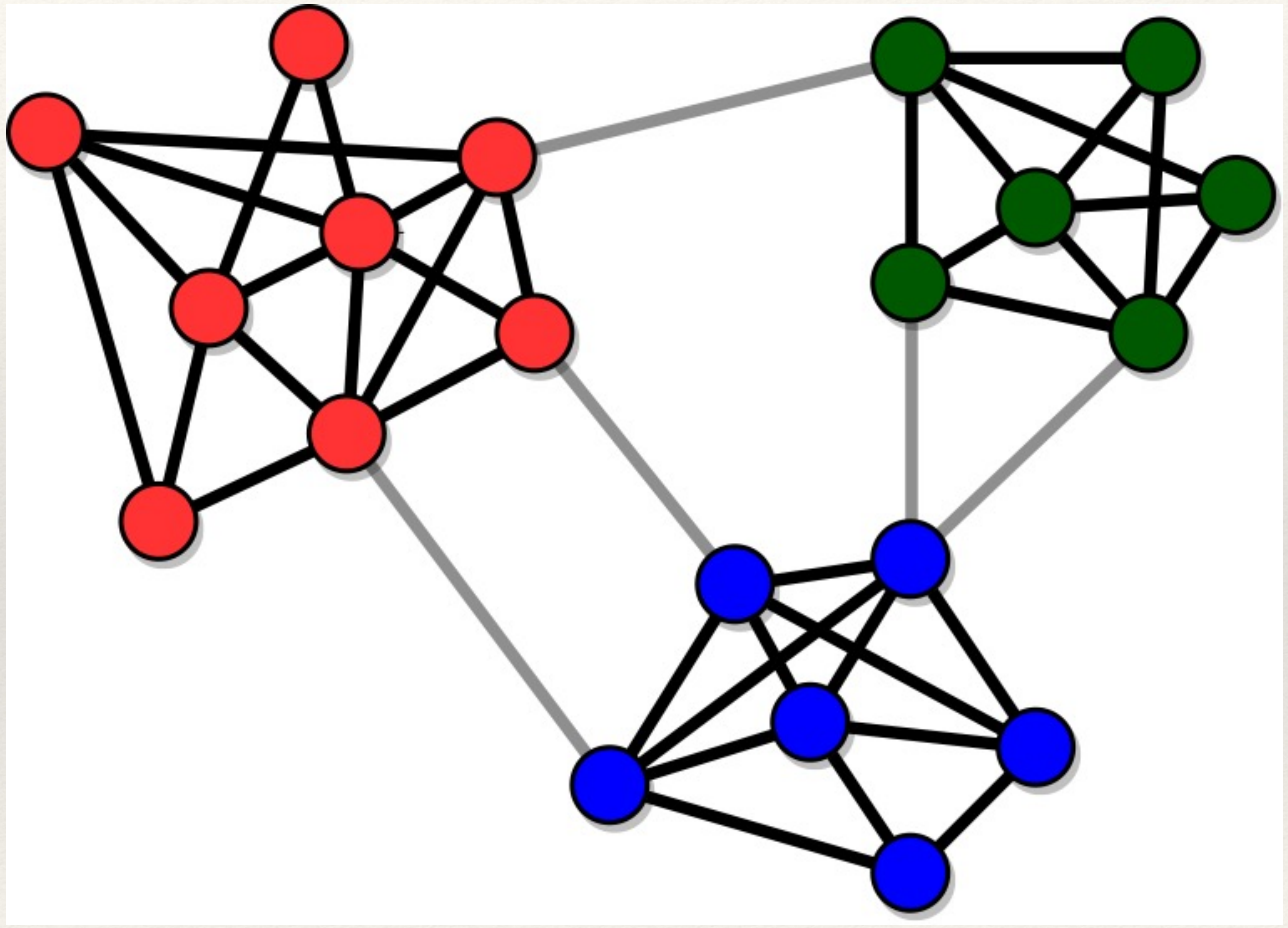
- Uncover the organization of the network
- Identify features of the nodes
- Classify the nodes based on their position in the clusters
- Find missing links



Basic definitions: community

Two main features:

- **High cohesion:** communities have many internal links, so their nodes stick together
- **High separation:** communities are connected to each other by few links



How to define communities?

Defining communities and community detection

- **COMMUNITIES**
- To retrieve communities we must first **formally** define them
- There are many ways to do so, none of which can be considered the best
- In the remainder we consider only non-overlapping communities on undirected and unweighted graphs and try to understand strengths and limitations of the various approaches

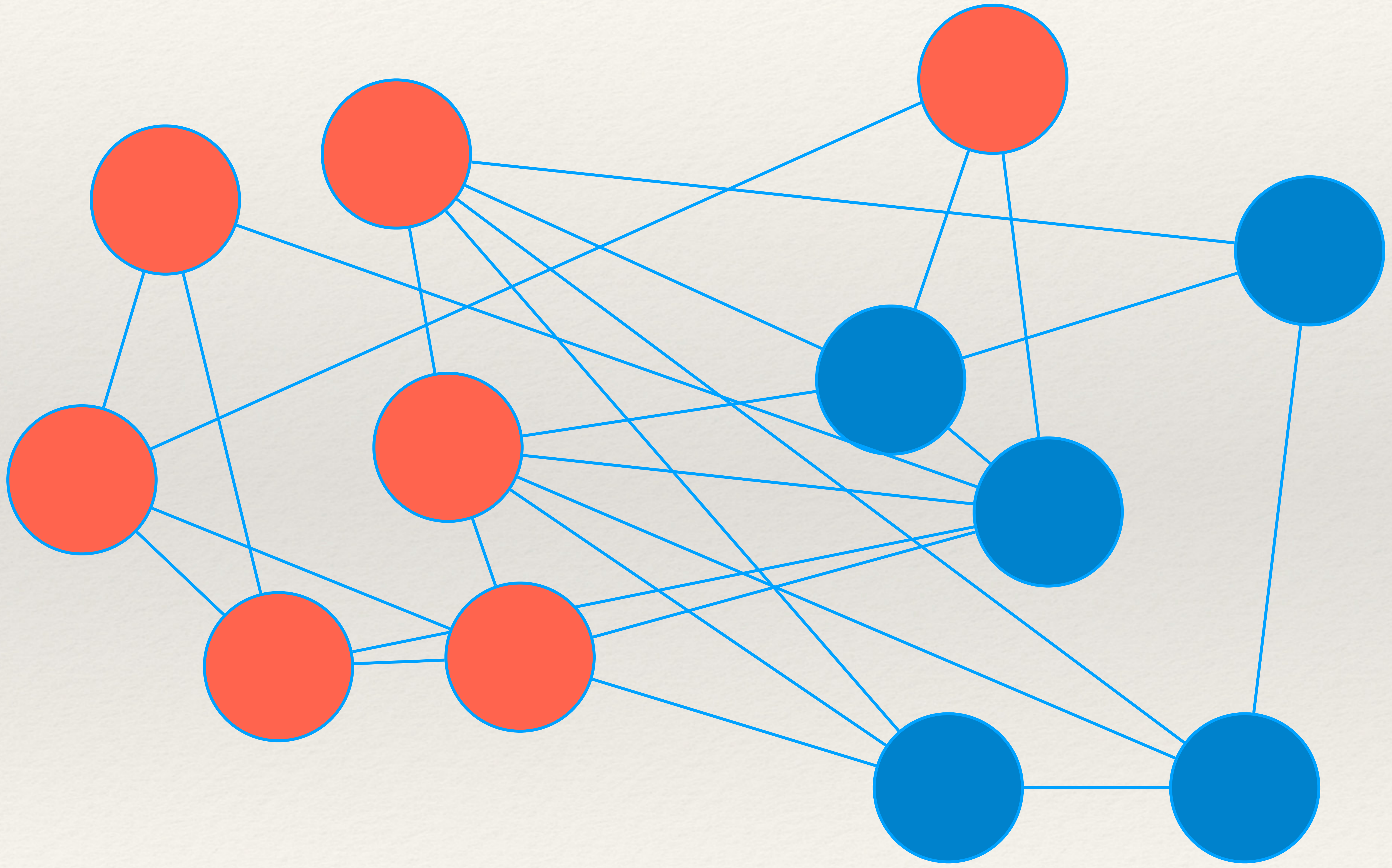
- **COMMUNITY DETECTION**
- Given a definition of community and a graph, a community detection algorithm is an unsupervised algorithm that outputs a node partition, assigning a community label to each node

Cost function optimization

Cost function optimization

More connections inside the community than outside

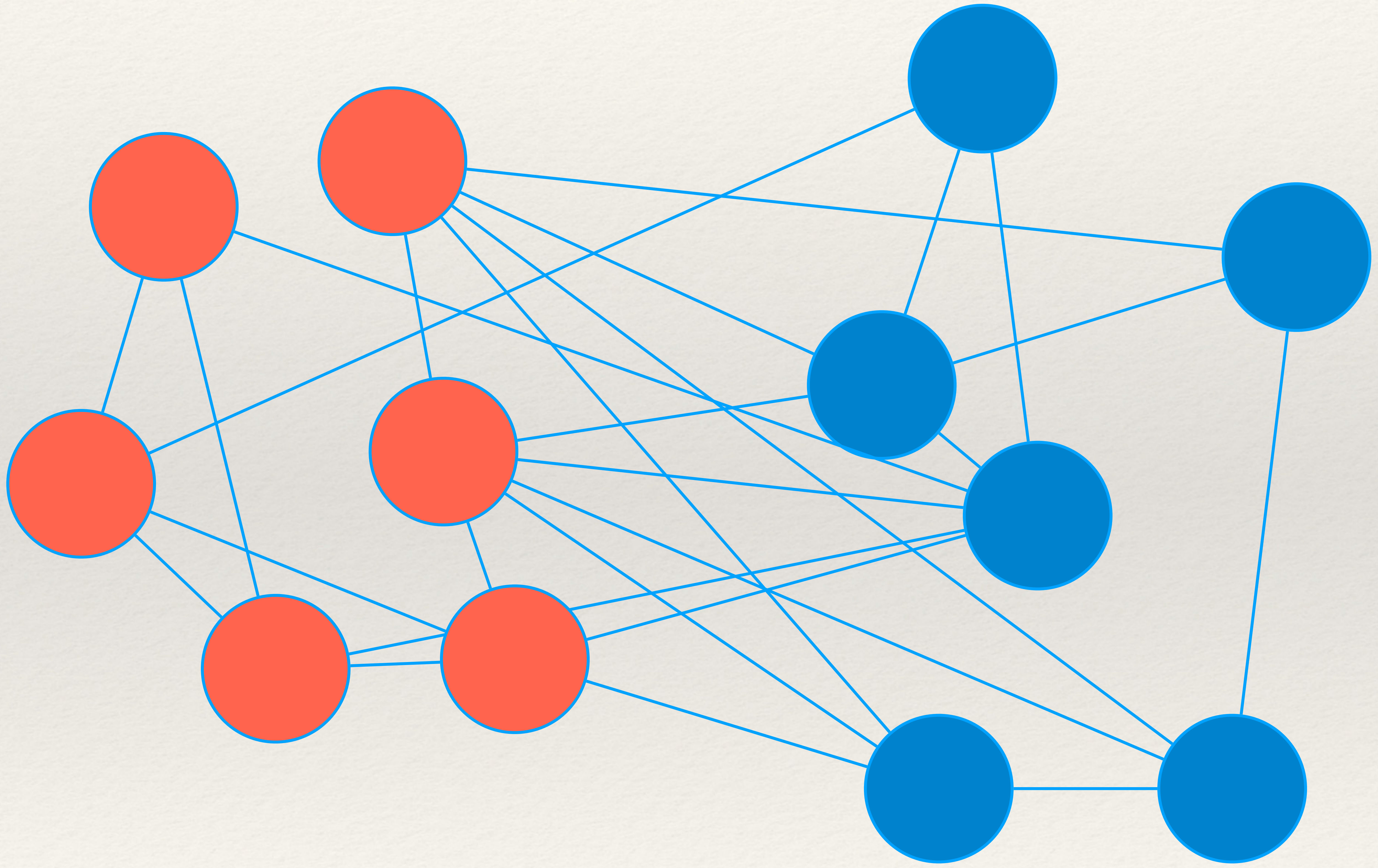
Is this good?



Cost function optimization

More connections inside the community than outside

Is this better?



Cost function optimization

Why was the second configuration "better" than the first?

We can define a cost function that tells us how good a node partition is and perform community detection by simply optimizing that cost function.

In this case, the cost function defines the concept of communities, so it must be done carefully.

Let us now see some examples.

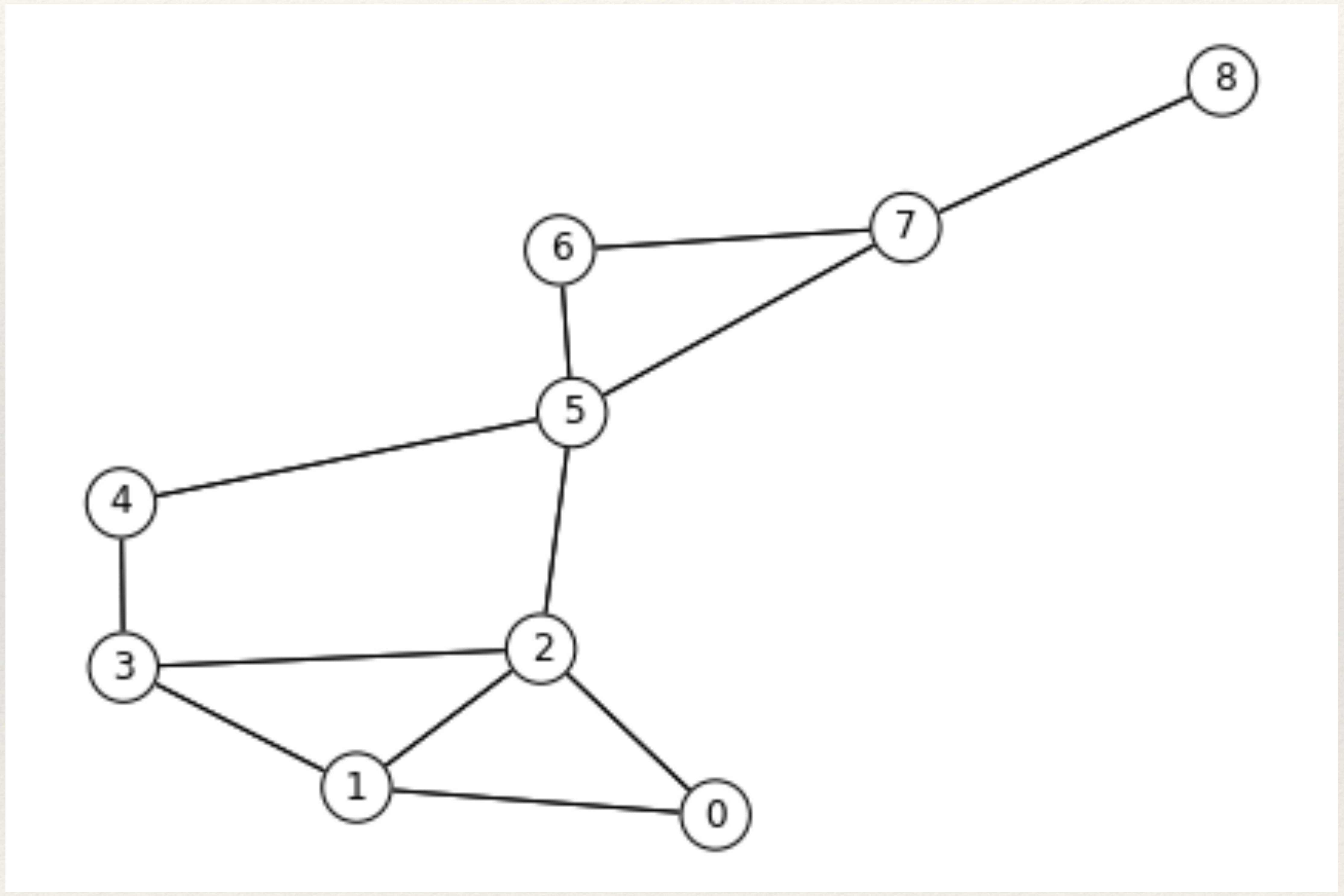
Optimizing the cut

- Let $N_1, N_2, \ldots, N_q$ be a note partition in q communities. Each node of the graph belongs to one and one only of these sets.
- We define the graph cut as the number of edges falling between nodes in different communities

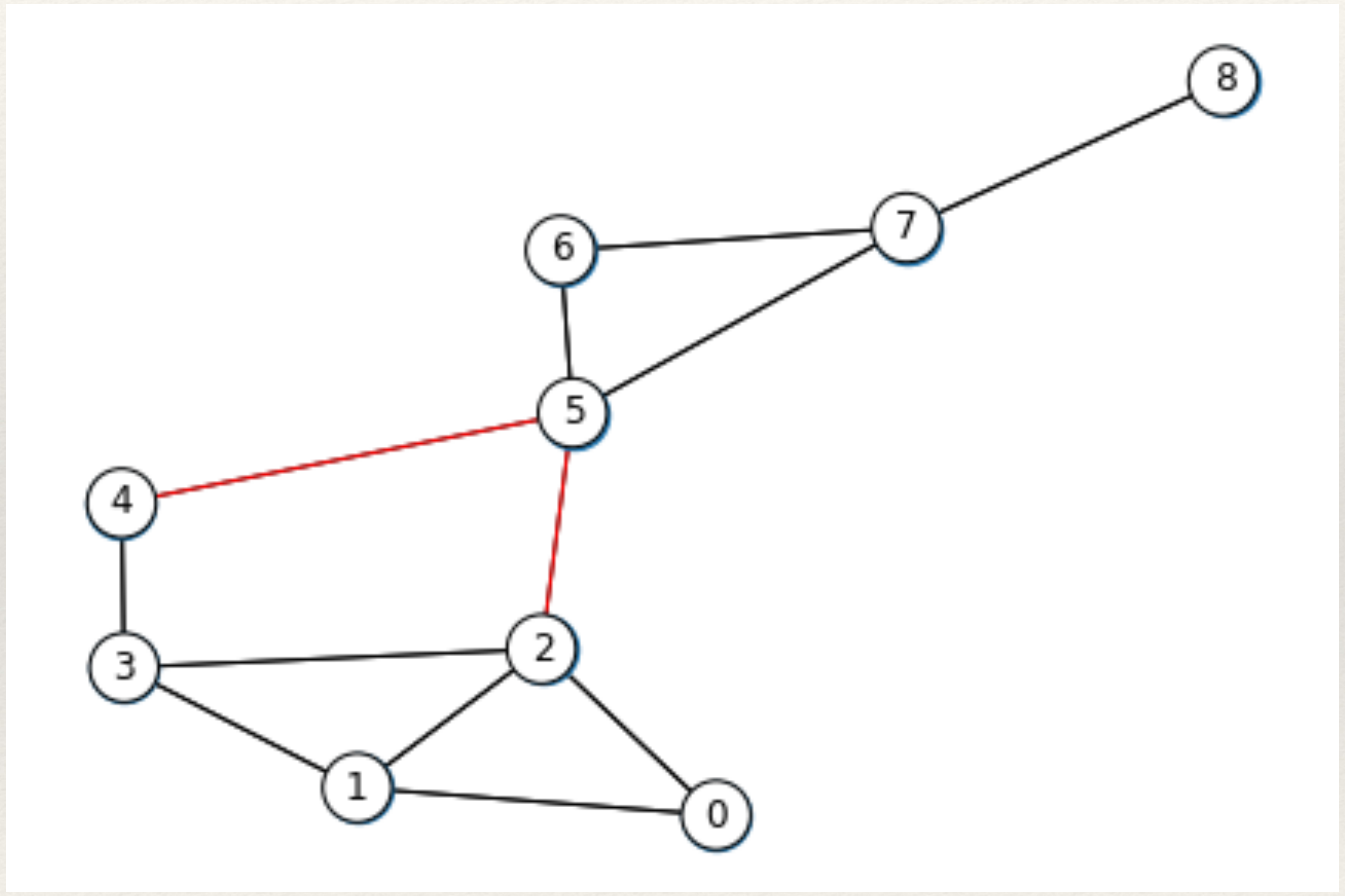
$$cut(N_1 \ldots, N_q) = \frac{1}{2} \sum_{a=1}^{q} \sum_{i \in N_a} \sum_{b \neq a} \sum_{j \in N_b} A_{ij}$$

What is the partition that minimizes the cut?
Does this seem a reasonable approach?

Optimizing the cut

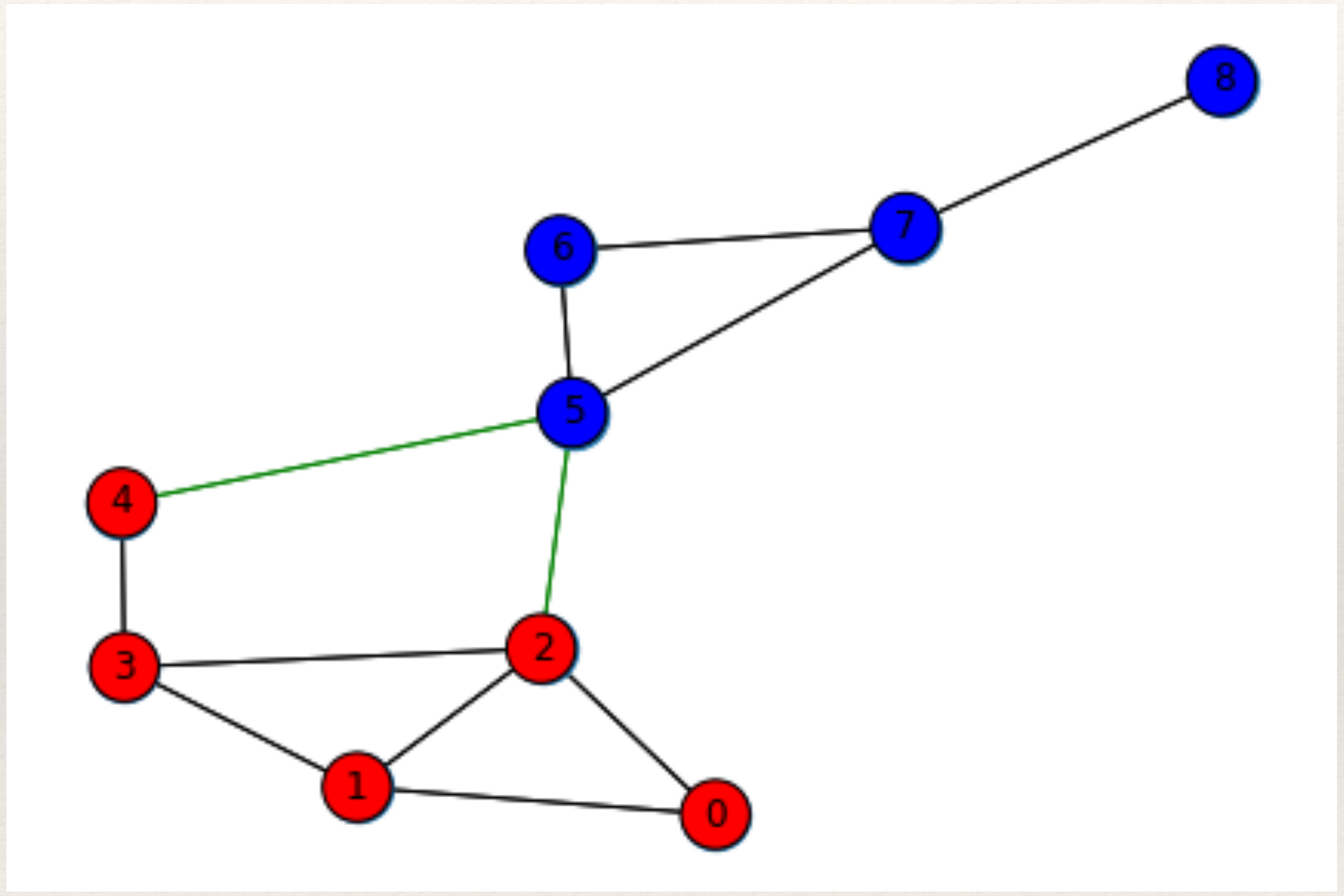


Optimizing the cut



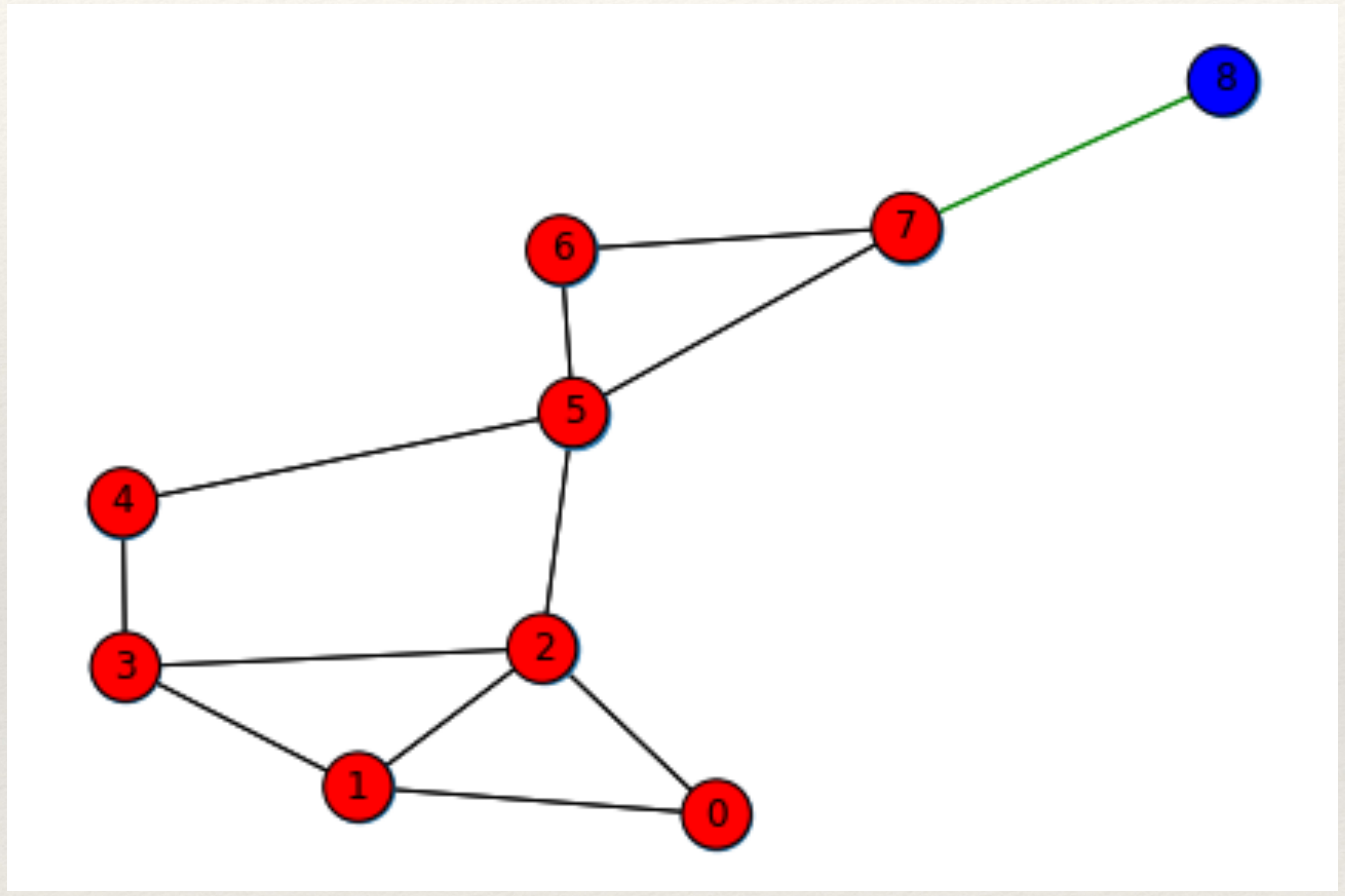
Let's try to cut it here

Optimizing the cut



The cut equals 2, so this is good

Optimizing the cut



But unfortunately this is better

Optimizing the cut

The most intuitive definition of the cost function does not seem to be efficient in creating meaningful node partitions because it tends to isolate single nodes from all the rest of the network.

But after all, that is what we asked for: fewer connections between nodes in the same community that between nodes in different communities.

We need to revise this cost function keeping the community size into account.

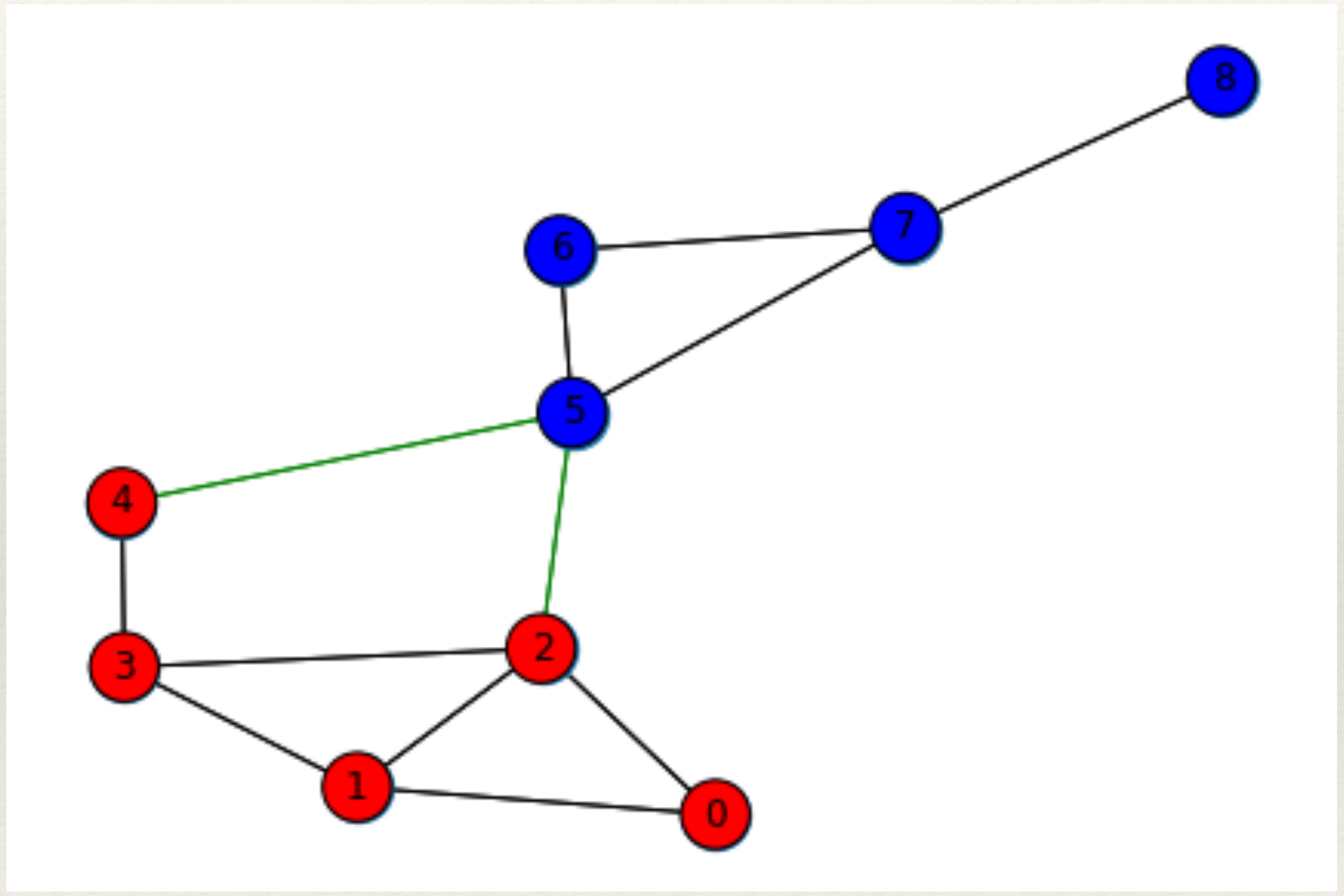
To do so we introduce the Ratio Cut

The ratio cut

We count the average number of connections each node has with nodes in another community. This defines the ratio cut.

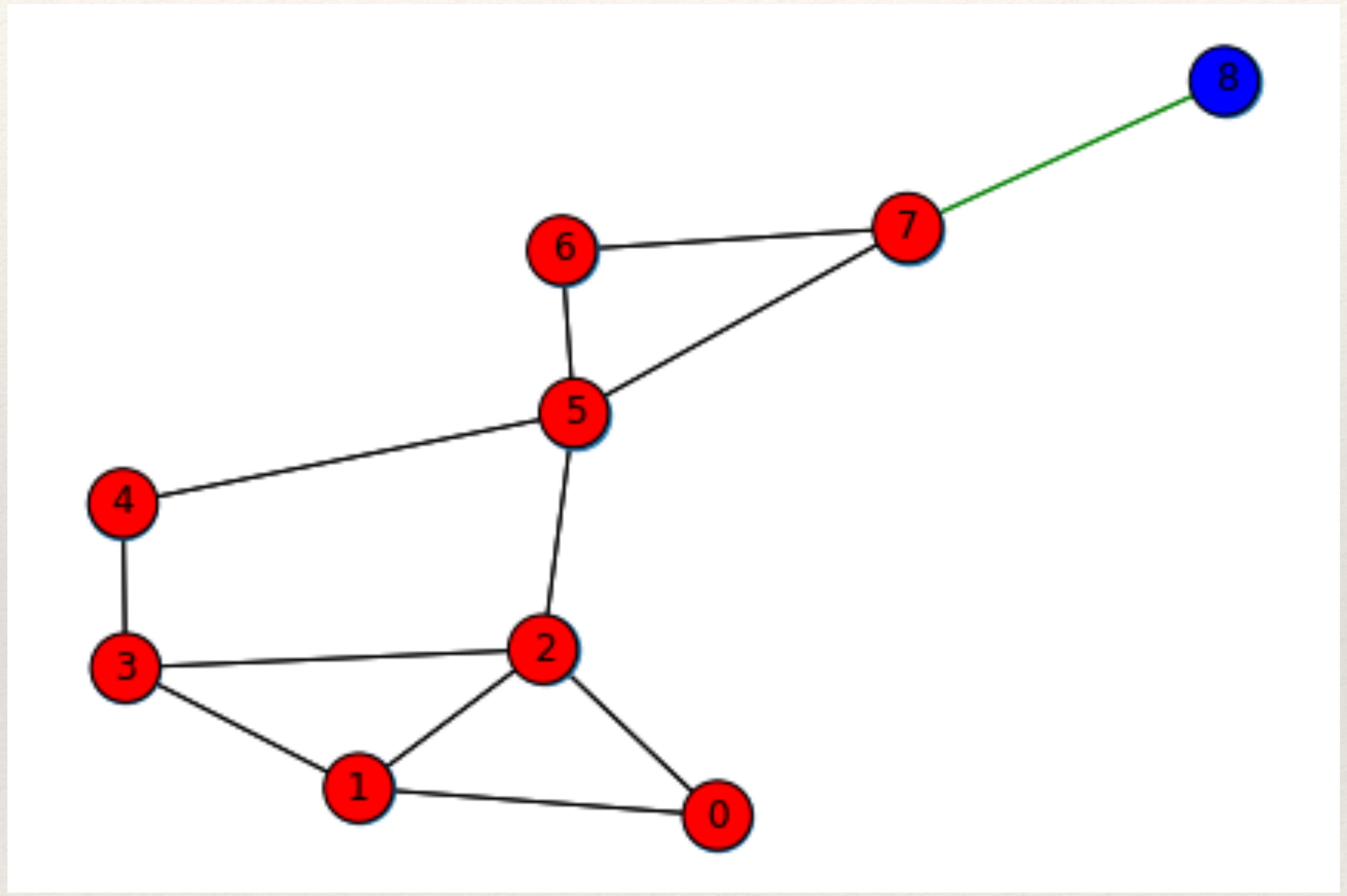
$$Rcut(N_1, \ldots, N_q) = \frac{1}{2} \sum_{a=1}^{q} \frac{1}{|N_a|} \sum_{i \in N_a} \sum_{b \neq a} \sum_{j \in N_b} A_{ij}$$

Optimizing the Rcut



The Rcut is $\frac{1}{2}\left[\frac{2}{5}+\frac{2}{4}\right]=\frac{9}{20}$

Optimizing the Rcut



The Rcut is $\frac{1}{2}\left[\frac{1}{8}+\frac{1}{1}\right]=\frac{9}{16}$

Optimizing the Rcut

This definition seems to be more convenient but we introduced a massive problem!

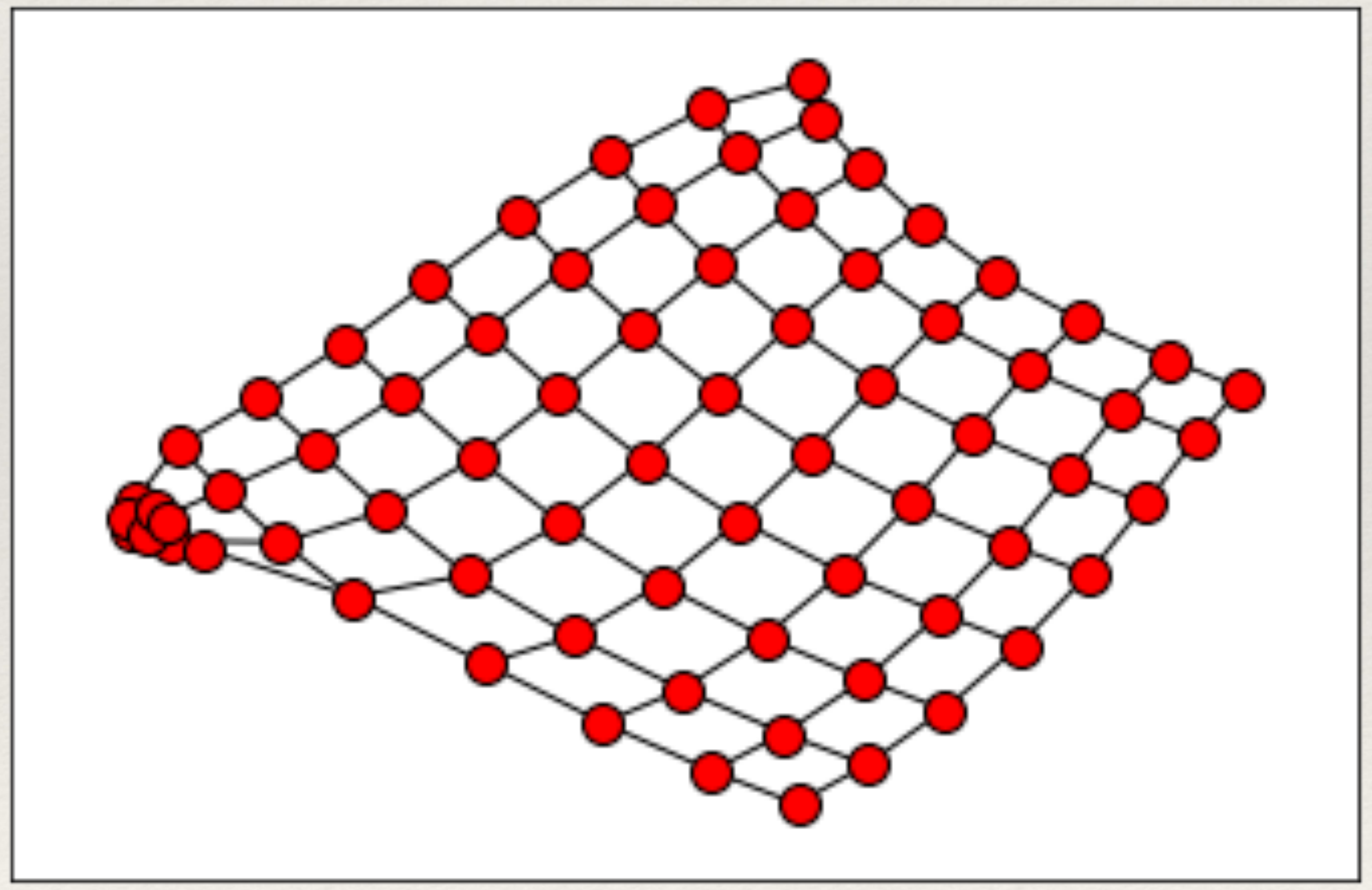
The min-cut problem can be solved in polynomial time, while the Rcut problem can be proved to be NP complete.

This means that in the worst case, we must try all possible configurations (q^n) and see which one is the best. This is absolutely not feasible in a reasonable time.

Moreover, the ratio cut tends to favor configurations in which communities have approximately the same size. But why should that be the case?

Optimizing the Rcut

Let us consider a lattice with 64 nodes connected to a clique of 8 nodes through 4 edges. The natural community partition has a rcut $= 9/32 \approx 0.28$, while dividing the network in two groups of equal size we get rcut $= 2/9 \approx 0.22$.



Optimizing the modularity

Communities are much more than just "counting edges". We want to consider communities as a *non trivial* local structure that deviates from randomness.

Let's go back to the min-cut problem

$$
\begin{aligned}
cut(N_1, \ldots, N_q) &= \tfrac{1}{2} \textstyle\sum_{a=1}^{q} \sum_{b \neq a} \sum_{i \in N_a} \sum_{j \in N_b} A_{ij} \\
&= \tfrac{1}{2} \textstyle\sum_{i \in N} \sum_{j \in N} A_{ij} \left(1 - \delta_{\ell(i),\ell(j)}\right) \\
&= L - \textstyle\sum_{i \in N} \sum_{j \in N} A_{ij} \delta_{\ell(i),\ell(j)}
\end{aligned}
$$

Optimizing the modularity

Let us now consider a matrix A' obtained from the preferential attachment model with the same degrees of A. What is the expected cut in that case?

$$\begin{aligned}\mathbb{E}[cut'(N_1, \ldots, N_q)] &= L - \textstyle\sum_{i \in N} \sum_{j \in N} \mathbb{E}[A'_{ij}]\delta_{\ell(i),\ell(j)} \\ &= L - \textstyle\sum_{i \in N} \sum_{j \in N} \frac{k_i k_j}{2L}\delta_{\ell(i),\ell(j)}\end{aligned}$$

Optimizing the modularity

The modularity is obtained comparing cut with E[cut'] as follows

$$
\begin{aligned}
mod(N_1, \ldots, N_q) &= \frac{cut'(N_1,\ldots,N_q) - cut(N_1,\ldots,N_q)}{2L} \\
&= \frac{1}{4L} \sum_{i,j \in N} \left(A_{ij} - \frac{k_i k_j}{2L} \right) \delta_{\ell(i),\ell(j)}
\end{aligned}
$$

High modularity values (empirically approx 0.3 - 0.4) correspond to "good" partitions

Optimizing the modularity

The optimization of the modularity is one of the most popular approaches to perform community detection on graphs. This method has, however, several known problems that we here summarize

1) The modularity optimization is capable of finding communities in a structureless graph. Simply looking at the modularity value is dangerous.

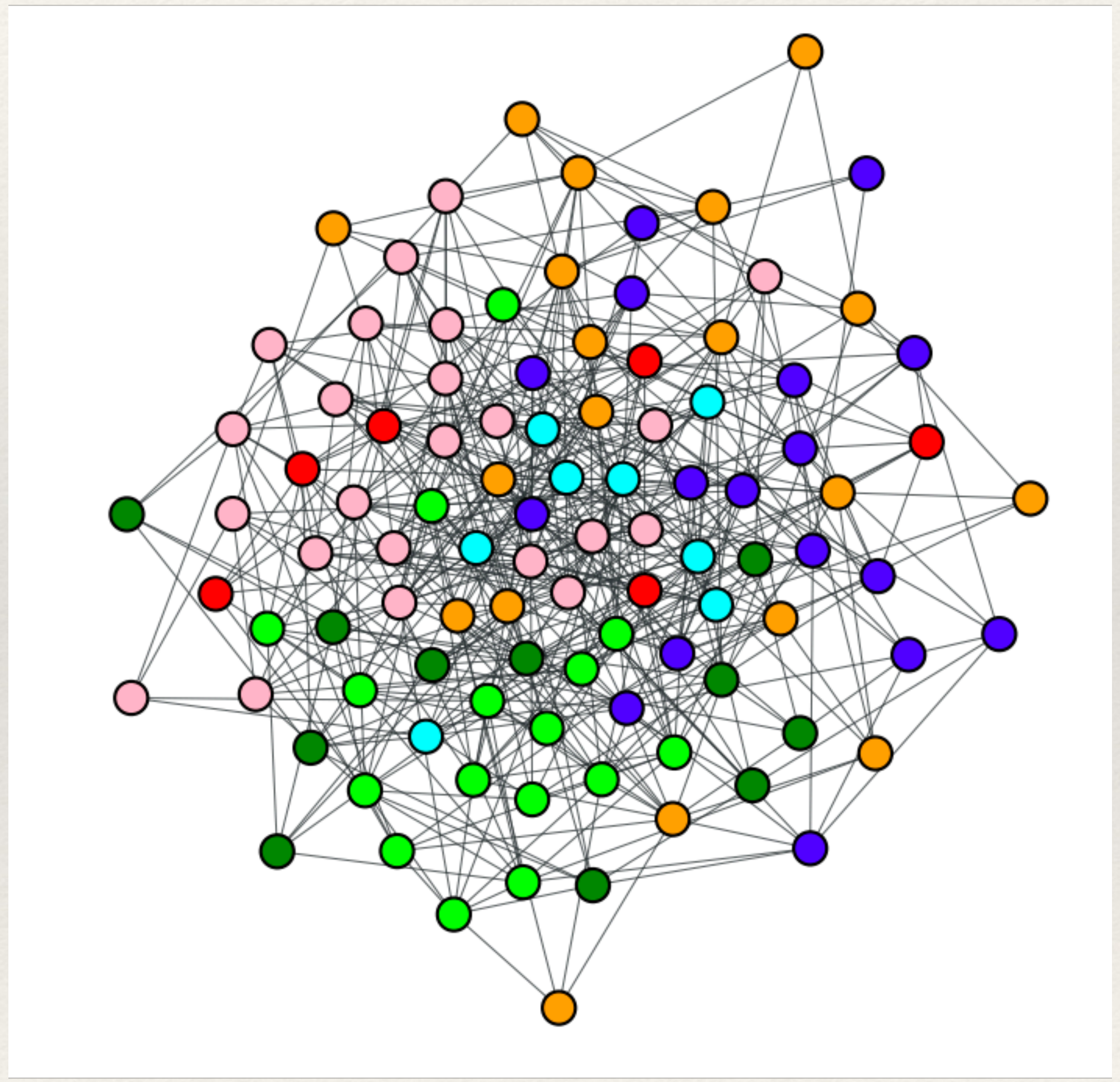
2) The most common way of optimizing the modularity considers the number of communities q as a variable of the optimization. <u>This is an improvement over the cut approach</u>. It can be shown however that the modularity is not strictly comparable for different values of q and this approach may lead to obtaining too many or too few communities.

3) This optimization problem is also NP hard. There exist approximate but efficient methods to optimize it, but, in typical settings, the landscape of the cost function is very rough and it is nearly impossible to find the actual minimum.

Optimizing the modularity

Communities in structureless graphs

The approach on which the modularity maximization is based on relies in comparing a signal (the community partition on the graph) on a random realization thereof. This is a standard and solid statistical method but if we optimize over it, we risk to overfit and obtain a node partition that is "over optimistic".



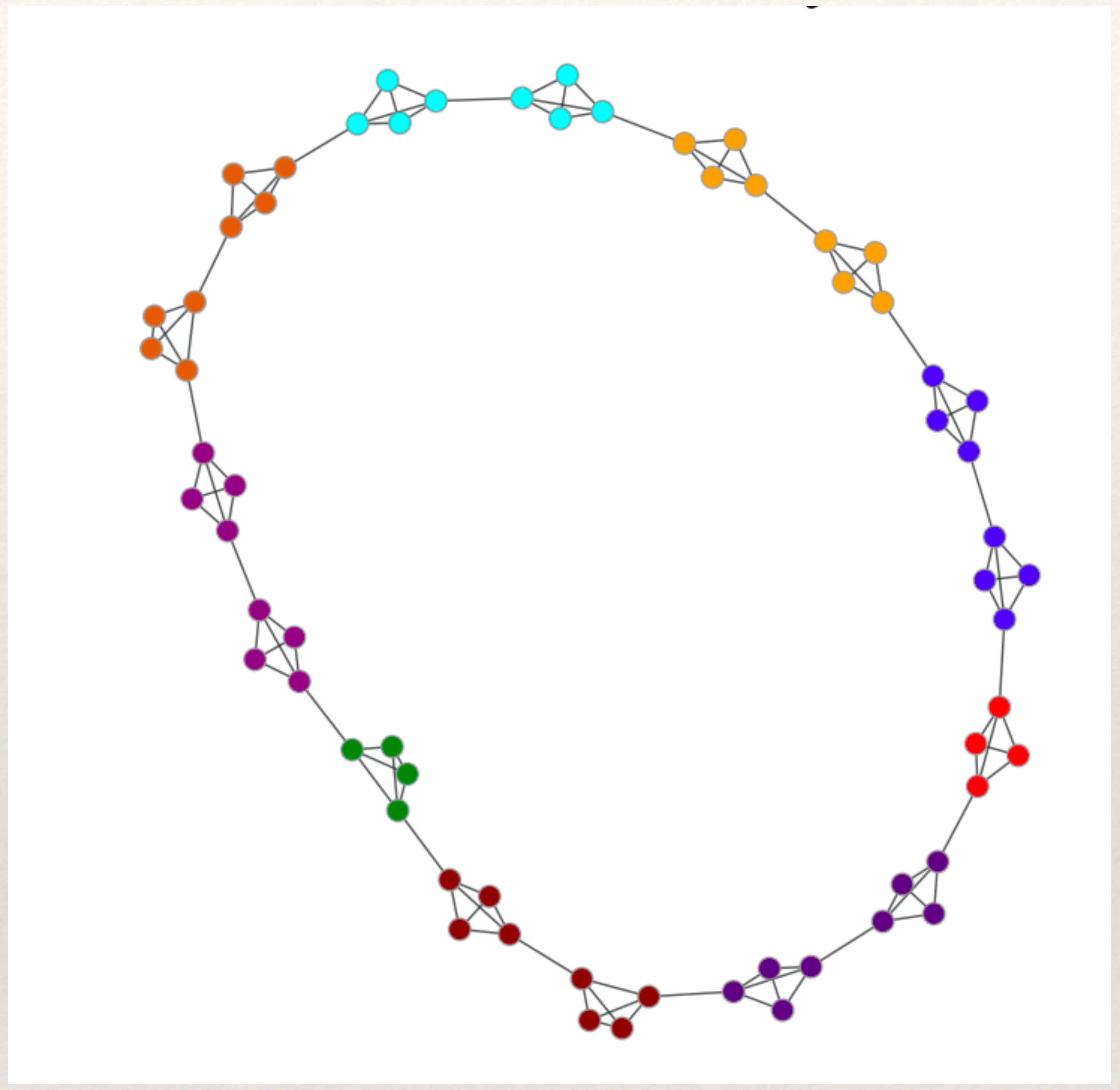
For instance, I built an Erdos-Renyi graph with n = 1000, p = 4/n and obtained a community structure with q = 34 and mod = 0.52!

Optimizing the modularity

The number of communities

The modularity is not directly comparable for different values of q and this typically leads to be overconfident and split the graph in many small pieces. However, the opposite can happen as well and it is known as the *resolution limit* of modularity maximization.

In the example on the right, the natural partition in q = 17 communities has mod = 0.8, while the one shown in the picture with q = 10 has mod = 0.81.



This problem is solved with a regularized modularity

$$\frac{1}{4L}\sum_{i,j}\left(A_{ij}-\gamma\frac{k_ik_j}{2L}\right)\delta_{\ell_i,\ell_j}$$

Optimizing the modularity

Solving an NP hard problem

One of the most popular (and fast) solutions to optimize the modularity is the Louvain algorithm (more recently improved by the Leiden algorithm).

We give each node a different community label. We then iterate two steps until convergence

1) For all nodes, consider the modularity gain obtained by moving that node to the community of one of its neighbors. After that, change its community label to the class that produces the highest gain of the modularity.

2) After this operation is done for all nodes, merge all nodes in the same community reweighing the edges and start over with the smaller graph.

Optimizing the modularity

Solving an NP hard problem

This method is very fast and gives a solution in O(n log(n)) steps. This strategy is, however largely suboptimal and it is very likely to end up in a local extreme of the optimization function. To be more quantitative, depending on the graph structure, this may become exponentially likely.

As a consequence, different runs of the same algorithm on the same graph may produce node partitions that have almost nothing in common but that have very similar (and potentially high) modularity values.

The Louvain algorithm is hence powerful and fast, but is must be used with great care.

Optimization approaches

Defining communities as the solution of an optimization problem is probably the most straightforward strategy. Among the major pitfalls associated to this class of algorithms we recall that

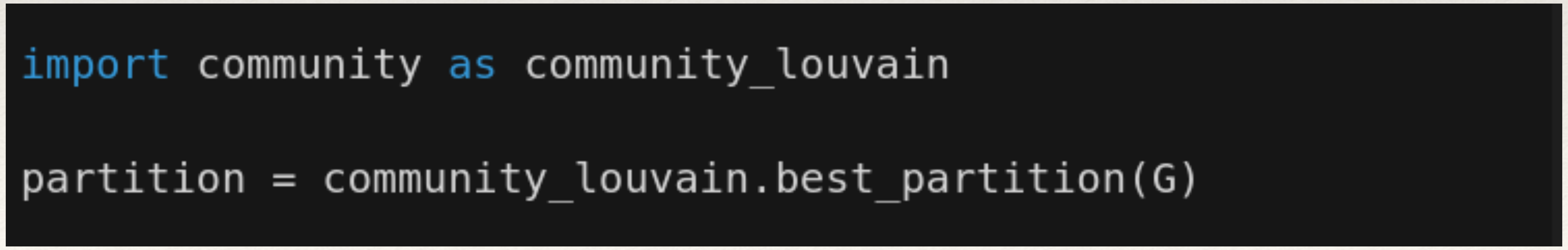
1) The role played by the number of communities q is not clear and may lead to over- or dow-partitioning
2) We only introduced few cost functions but none of them is problem-free. A typical approach consists in changing the cost function to prevent bad behaviors. But in the end, what is the good function to optimize?
3) In most of the cases, the cost function to optimize is NP hard and, even though efficient approximations exist, they are only approximations.

Optimization approaches

When is it good to use these methods?

If your goal is truly to optimize a cost function, then this is undeniably a good, fast and well studied approach. But be aware of all the problems that may come with it.

To use Louvain in Python go to https://python-louvain.readthedocs.io/en/latest/api.html



```
import community as community_louvain

partition = community_louvain.best_partition(G)
```

Bayesian approach

Let′s play a game

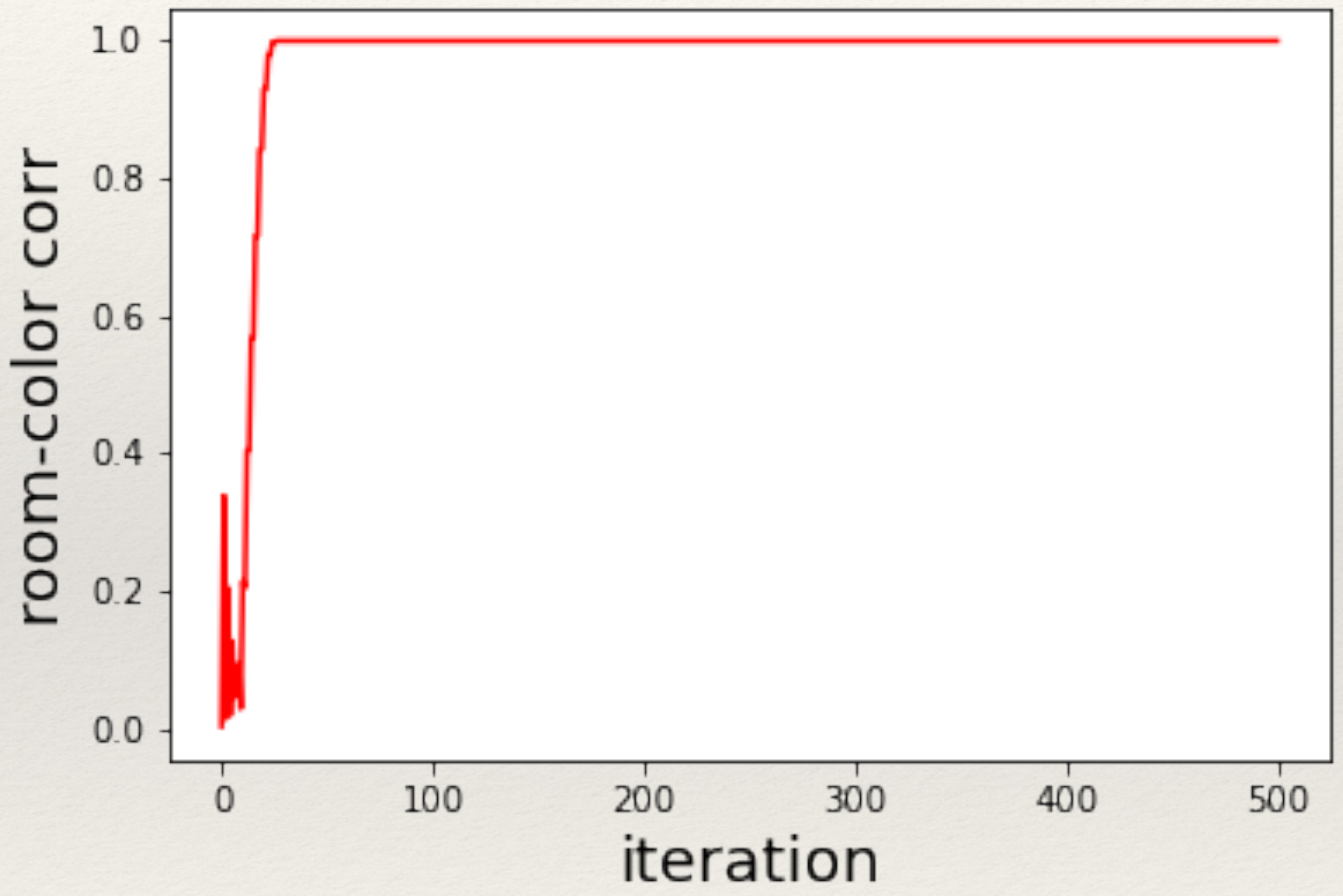
Suppose we have two groups of people: pink and green. They are randomly divided in two rooms and each of them has a paper with a list of friends (F) and a list of enemies (E).

What we know is that green people are more likely to have friends among the greens and enemies among the pinks and vice-versa. The best configuration hence seems to be the one in which the two groups split according to their color. Suppose however that no one knows their color and they have to find a way to organize themselves. They adopt this strategy

* each friend in your room counts as +1 and each enemy in your room counts as -1. The signed are reversed for the opposite room.
* each individual computes the score and moves if the score is negative or stays if its positive.
* we do that until we reach convergence.

Will it work?

Let's play a game



correlation between room and color assignment as a function of time when all friends of yours have the same color and all enemies have the opposite color.

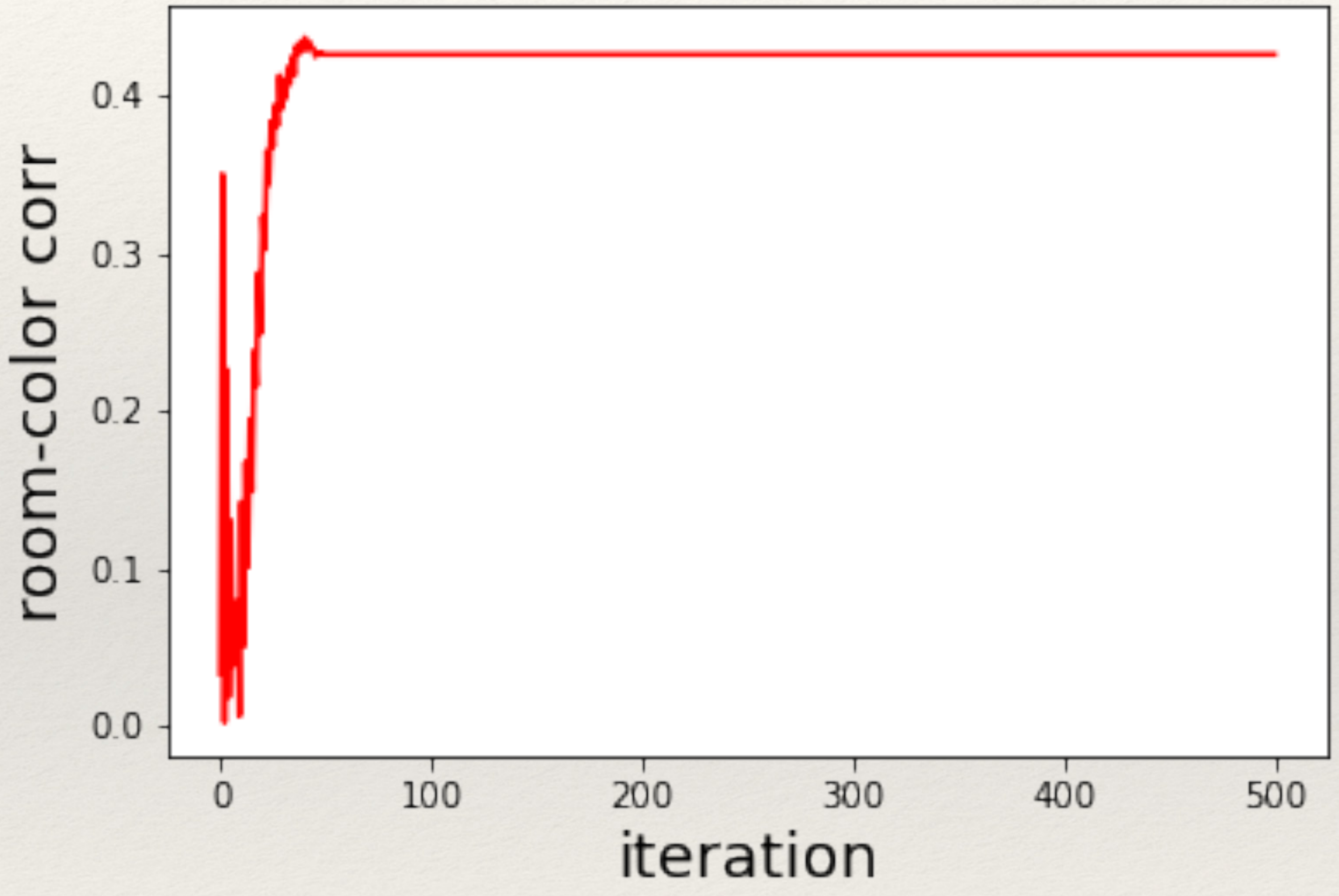
Let′s play a game

So, the result seems to be good. Note the similarity with the modularity maximization.

We have a community structure to be recovered (the color) and a guess (the room). To recover one from the other we define a cost function and at each step we move only if we have a gain.

This works well because the two groups are well separated. But what happens if this is not the case and I may have some enemies of the same color and some friends of the opposite?

Let's play a game



correlation between room and color assignment as a function of time when the majority of friends of yours have the same color and the majority of enemies have the opposite color.

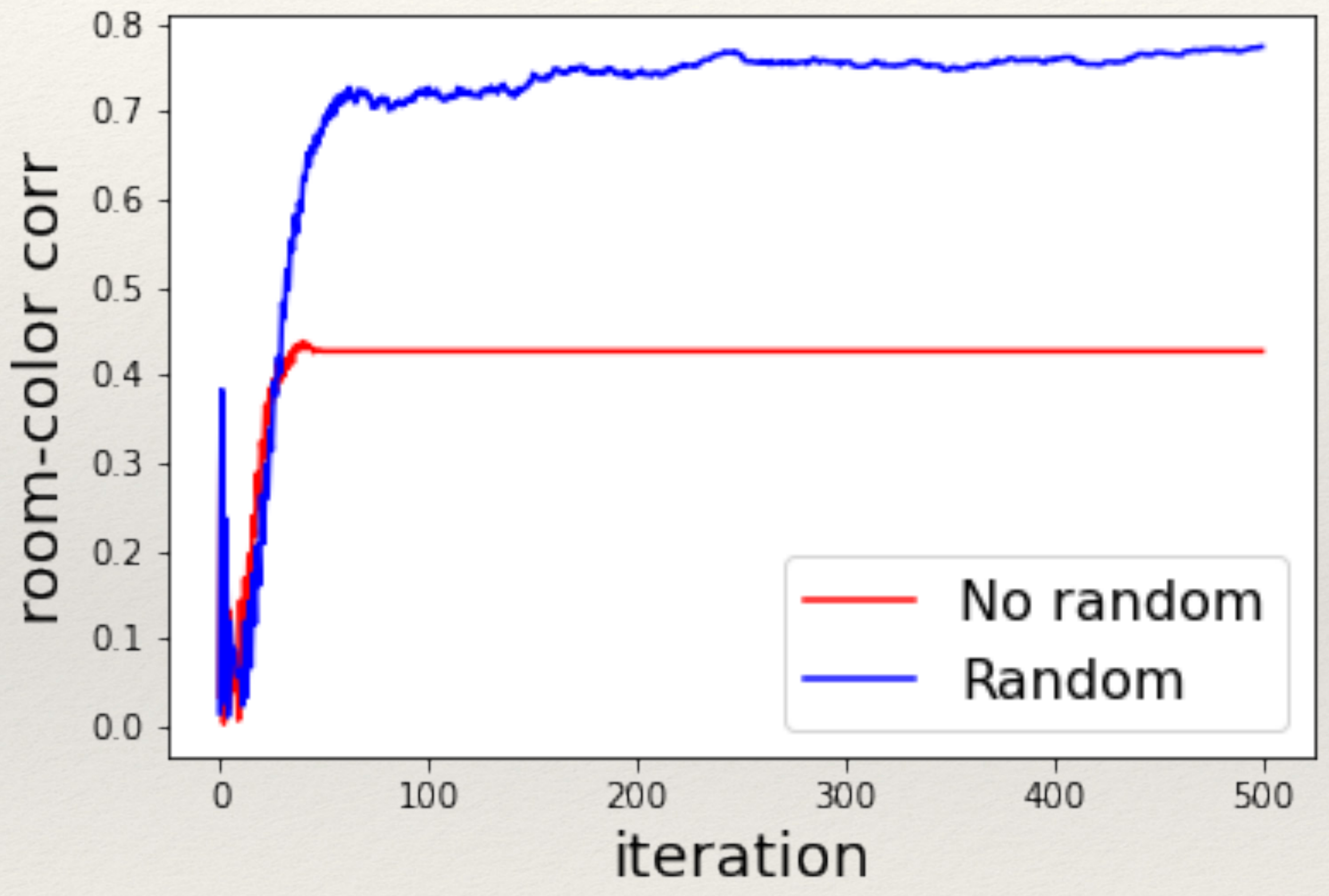
Let's play a game

It looks like we reach a local extreme of the cost function very soon and we are not able to escape it. Maybe that is actually the best we can hope for.

Let us try a different strategy:

* each friend in your room counts as +1 and each enemy in your room counts as -1. The signed are reversed for the opposite room.
* each individual computes the score and moves **with a probability p** if the score is negative or stays if its positive.

Let's play a game



correlation between room and color assignment as a function of time when the majority of friends of yours have the same color and the majority of enemies have the opposite color.

Let's play a game

As you can see, the randomized method performs much better than the optimization one.

This little game should explain you the main difference between the Bayesian approach (the randomized one) and the optimization approach.

This example has a deep relation with statistical physics. The cost function plays the role of the energy of the system that we want to minimize, while the randomness plays the role of the temperature that make disordered configurations occasionally more convenient.
Real-world datasets are not purely divided into communities and the optimization approach may be, in general too optimistic.

Community detection in the DCSBM

We generate a random graph with an arbitrary degree distribution and a community structure

$$P(A_{ij}|l) = \frac{k_i k_j}{c^2} \begin{cases} p_{\text{in}} & \text{if } l_i = l_j \\ p_{\text{out}} & \text{if } l_i \neq l_j \end{cases}$$

We then want to infer l from A

$$P(l|A) \propto P(A|l)$$

Community detection in the DCSBM

We now show the relation between this approach and the modularity maximization

$$
\begin{aligned}
P(A|l) &= \prod_{i<j} p_{ij}^{A_{ij}} (1 - p_{ij})^{(1-A_{ij})} \\
&= \exp\left\{ \tfrac{1}{2} \sum_{i,j} A_{ij} \log\, p_{ij} + (1 - A_{ij}) \log\, (1 - p_{ij}) \right\} \\
&= \exp\left\{ \tfrac{1}{2} \sum_{i,j} A_{ij} \log \left(\tfrac{p_{ij}}{1-p_{ij}} \right) - \log(1 - p_{ij}) \right\}
\end{aligned}
$$

Community detection in the DCSBM

Let's consider the first term for small connection probabilities

$$A_{ij}\log\left(\frac{p_{ij}}{1-p_{ij}}\right) \approx A_{ij}\log\, p_{ij}$$

$$= A_{ij}\left[\log\frac{k_i k_j}{c^2} p_{in}\delta_{l_i,l_j} + \log\frac{k_i k_j}{c^2} p_{out}(1-\delta_{l_i,l_j})\right]$$

$$= A_{ij}\ \log\frac{p_{in}}{p_{out}}\delta_{l_i,l_j} + const.$$

Community detection in the DCSBM

Now we do the same for the second term

$$\begin{aligned}\log(1 - p_{ij}) \approx -p_{ij} \quad &= \frac{k_i k_j}{nc^2}\left[p_{in}\delta_{l_i,l_j} + p_{out}(1 - \delta_{l_i,l_j})\right] \\ &= k_i k_j \frac{p_{in} - p_{out}}{nc^2}\delta_{l_i,l_j} + const.\end{aligned}$$

Community detection in the DCSBM

We put things together

$$\begin{aligned}
nc &= 2L \\
\gamma &= \frac{p_{in} - p_{out}}{c\ \log\left(\frac{p_{in}}{p_{out}}\right)} \\
P(l|A) &\propto \exp\left\{\tfrac{1}{2}\log\frac{p_{\text{in}}}{p_{\text{out}}}\textstyle\sum_{i,j}\left(A_{ij} - \gamma\frac{k_i k_j}{2L}\right)\delta_{l_i,l_j}\right\}
\end{aligned}$$

Community detection in the DCSBM

With Bayesian stochastic block modeling we can associate a probability that each node belongs to a given community. Consequently, if we look for q communities in an Erdos-Renyi random graph we will obtain for each node probabilities that are very close to 1/q to belong to any cluster.

Bayes inference is optimal when the parameters of the model (p_in, p_out, q...) are known. This is not a reasonable assumption in general, but there are ways to learn them or to choose them well to perform "mismatched inference".

The probabilities can be estimated in polynomial time with respect to n (for instance with Montecarlo algorithm) but, in general, they are slower than fast greedy approaches.

Community detection in the DCSBM

What happens when the graph is not DCSBM?

The inference algorithm can still be applied, but the most common issue is associated to convergence problems. This is due to the fact that the DCSBM has a low clustering coefficient while real world networks have a lot of triangles.

The consequence is that, even if we have provable guarantees of convergence of the algorithm on a DCSBM graph, for an arbitrary initial condition there may be convergence problems on a real graph.

Community detection in the DCSBM

When is it convenient to use the Bayesian approach?

In general, the optimization approach is a "corner case" of the Bayesian approach, hence, it is likely to give worse results. However, in practical settings, one must keep speed and stability into account and be able to interpret the results of the algorithm.

The author of graph-tool is the most well-known researcher in Bayesian stochastic block modeling and this Python package is certainly the way to go.