Social and Behavioural Networks

Uniroma1 2018-19 and 2019-20

Andrea Proietto, Giovanni Ficarra, Donato Crisostomi

December 7, 2019

Contents

Chapter 1

Introductory pills

1.1 Objectives of the course

The main objective of the course are:

1. Find efficient solutions to analyze social networks;
2. Define models that describe users' behavior;
3. Find ways to infer missing data in a social network.

Our models have to adapt well to real cases but also to be understandable.

Our algorithms have to be efficient: they need few memory and execution time, and they can easily be executed in parallel on more computers, using frameworks such as Pregel or Map-Reduce.

1.2 Basic ideas

Here we enumerate some preliminary notes and formulas that should be already known and will be useful during the course:

- *Note*: "pick" and "sample" are used interchangeably;
- We call *node* an entity on the web that can be a blog, a website or a social network profile
- A useful property of logarithms:
$$e^{\ln x} = x \tag{1.1}$$
- A recurrent inequality:
$$1 - x \leq e^{-x} \tag{1.2}$$
- A helpful limit:
$$\lim_{x \to +\infty} \left(1 - \frac{1}{x}\right)^x = \frac{1}{e} \tag{1.3}$$
- Let E_i be mutually independent events, then
$$\Pr\left\{\bigcap_{i=0}^{n} E_i\right\} \leq \prod_{i=0}^{n} \Pr\{E_i\} \tag{1.4}$$

Then we have some useful definitions.

Definition 1.2.1 (Expected value)**.** *Let X be a random variable with a finite number n of finite outcomes $x_1, x_2, \ldots, x_n$ occurring with probabilities $p_1, p_2, \ldots, p_n$ respectively. Then The expected value of X is*

$$E[X] = \sum_{i=1}^{n} x_i \cdot p_i. \tag{1.5}$$

Note that $E[X]$ is called the expected value because it means the midpoint where all outcomes, compounded with their weight, contribute to the equilibrium.

Question 1.2.1. Can we still talk about expected value when the outcomes are not numeric? What could be a good bijection between a generic sample space Ω and $\mathbb{N}$?

Definition 1.2.2 (Sampling)**.** *It is a simple algorithmic approach used for abstracting and estimating the percentage of elements of a certain type in a large set.*

Example 1.2.1. If there is a huge number of black and white marbles in a (very big) bowl and you'd like to know the ratio of black marbles, you can't count all of them, so you extract a sample of marbles many times, until you have sufficient confidence in the obtained result.

Definition 1.2.3 (Bayes' Theorem)**.** *Bayes' theorem (or law or rule) describes the probability of an event, based on prior knowledge of conditions that might be related to the event.*

$$Pr\{(A \cap B)\} = Pr\{(A)\} \cdot Pr\{(B|A)\} \tag{1.6}$$

1.2.1 Pregel framework

Brief description:

- Each node can communicate only with its neighbors;
- The computation si asynchronous;
- Each node can use few local memory (since there are many nodes);
- The nodes of a network (graph) are distributed on many computers, so it's important to group them based on community to reduce the number of communications among computers, and to accurately choose a threshold between the memory to assign to each node and the number of nodes we want to put into each single machine.

Example 1.2.2 (Find connected components of a graph using Pregel). Each node contains a unique number at the beginning (an ID), then every one send its ID to its neighbors, and when a node receives a new number replace its ID with the received one, if it's geater than its own.
When the values stop changing every connected component has the same value. The maximum number of iterations is given by the diameter of the graph.

1.2.2 Union bound

Definition 1.2.4 (Union Bound)**.** *Let A and B be two events, then*

$$Pr\{A \vee B\} \leq Pr\{A\} + Pr\{B\} \tag{1.7}$$

Proof. The theorem directly follows from the fact that probabilities are non-negative and from the following property:

$$\Pr\{A \vee B\} = \Pr\{A\} + \Pr\{B\} - \Pr\{A \wedge B\} \tag{1.8}$$

□

Definition 1.2.5 (Generalized Union Bound)**.** *Let E_i be a countable set of events, then*

$$Pr\left\{\bigcup_{i=0}^{n} E_i\right\} \leq \sum_{i=0}^{n}(Pr\{E_i\}) \tag{1.9}$$

If the events are mutually disjoint, then the equality is strict.

1.2.3 Markov's inequality

Let X be a non-negative random variable (RV) in a sample space Ω, and a an outcome in Ω, then:

$$\Pr\{X \geq a\} \leq \frac{\mathbb{E}[X]}{a} \tag{1.10}$$

Proof.

$$\begin{aligned}\mathbb{E}[X] &= \int_0^\infty x\Pr\{X = x\}\,dx \\ &\geq \int_a^\infty x\Pr\{X = x\}\,dx \\ &\geq \int_a^\infty a\Pr\{X = x\}\,dx \\ &= a\Pr\{X \geq a\}\end{aligned}$$

□

Informally, the Markov inequality let us tell something about an extreme event in a probability distribution of which we know only the expectation: if $\mathbb{E}[X]$ is small, then X is unlikely to be very large.

Another way of claiming this inequality is the following:

$$\Pr\{X \geq a \cdot \mathbb{E}[X]\} \leq \frac{1}{a} \tag{1.11}$$

More about this here or here.

1.2.4 Chebyshev's inequality

Let X (integrable) be a random variable with finite expected value μ and finite non-zero variance σ^2. Then, for any real number $k > 0$,

$$\Pr\{|X - \mu| \geq k\sigma\} \leq \frac{1}{k^2} \tag{1.12}$$

1.2.5 Moment generating functions

Moments, in a mathematical-analytical sense, are "quantitative measures" that describe characteristic of a shape. E.g.: a generic function's first moment is its slope (first derivative).
Given a random variable X, then its moment generating function is

$$M_X(t) = e^{tX}, t \in \mathbb{R} \tag{1.13}$$

Reminder! Power series of the e^x function: $e^x = \sum_{i=0}^{\infty} \frac{x^n}{n!}$.

Chapter 2

Chernoff bound

The Chernoff bound allows you to control how much a RV is close to its expected value.

Let $X_1, \ldots, X_n \in \mathcal{B}er(p)$ IID, i.e. $X_1, \ldots, X_n$ are n RV such that $\Pr\{X_i = 1\} = p$ and $\Pr\{X_i = 0\} = 1 - p$. In other words, they are n independent throws of the same coin.

Let $X = \sum_{i=1}^{n}(X_i)$. X is essentially a binomial RV: $X \in \mathcal{B}inom(2^n, p)$, thus $E[X] = p \cdot n$. It represents the number of times you get head with the aforesaid n throws.

So, given an error limit ε, we have the following definition:

Definition 2.0.1 (Chernoff bound)**.**

$$Pr\{|X - E[X]| \geq t\} \leq e^{-2t^2/n} \tag{2.1}$$

where $t := \varepsilon n$.

In the binomial case, [2.1] translates to:

$$\Pr\left\{\left|\sum_{i=1}^{n}(X_i) - p \cdot n\right| > t\right\} = \Pr\left\{\left|\frac{1}{n}\sum_{i=1}^{n}(X_i) - p\right| > \frac{t}{n}\right\}$$

Then, replacing t with εn, we get

$$\Pr\left\{\left|\frac{1}{n}\sum_{i=1}^{n}(X_i) - p\right| > \varepsilon\right\} \leq 2e^{-2n^3\varepsilon^2} \tag{2.2}$$

Example 2.0.1. If you want to know how many Facebook users have more than 100 friends, you can't simply count, but you can obtain a good approximation sampling a reasonable number of profiles.

Observation 2.0.1. It's important to observe that the size of the sample we need in order to obtain a good approximation depends on the error we tolerate, not on the total population.

There are many possible formulations of the Chernoff bound, another one that can be useful is the following:

$$\Pr\{|X - E[X]| \geq \varepsilon \cdot n\} \leq 2e^{-\frac{\varepsilon^2}{3}n} \tag{2.3}$$

Example 2.0.2. Let be $E[X] = \frac{50}{100}$ and $\varepsilon = \frac{1}{100}$, the probability that $X \geq \frac{51}{100}$ or $X \leq \frac{49}{100}$ is less then or equal to $2e^{-\frac{1}{30000}n}$.

Another useful formulation:

$$\Pr\left\{\left|\left(\frac{1}{n}\sum_{i=1}^{n}X_i\right) - p\right| > \varepsilon\right\} \leq 2e^{-2n\varepsilon^2} \tag{2.4}$$

Example 2.0.3. Let the true expected value be $p = \frac{1}{2}$, and the additive error be $\varepsilon = \sqrt{\frac{\ln(1/\delta)}{n}}$:

$$\Pr\{|\text{empirical average} - \text{true expected value}| > \varepsilon\} \leq 2e^{-2n\varepsilon^2} = 2e^{-2n\frac{\ln(1/\delta)}{n}} = 2\delta^2$$

Observation 2.0.2. The resulting approximation doesn't depend on the probability p.

Chapter 3

Modeling users' behaviors
(Graph models)

If we want to infer some information about a huge graph, such as the ones behind the web or Facebook, we usually can't perform our computation directly on the real graph, so we use models of the graph.

Some properties we usually are interested in when dealing with a network (or a graph) are the following:

- the overall number of nodes,
- the average degree,
- the number of nodes with a certain degree,
- the size of the communities,
- the degree distribution.

3.1 Erdős–Rényi model

This is a model for generating random graph, that has many applications due to its simplicity, but doesn't fit real world networks: each node has more or less the same degree, based on a fixed probability, while real networks have a gaussian distribution of the degrees.

We denote by $G(n, p)$ a random process that produces a graph with n nodes, in which each edge appears with probability p. This process can be described by the following algorithm:

```
Sample G(n,p):
    let E := [n]
    E ← ∅
    for each {i,j} ∈ (V choose 2):
        flip a coin with head probability p
        if the coin comes up head:
            E ← E ∪ {{i,j}}
    return G(V,E)
```

Listing 3.1: The G(n,p) algorithm

We are interested in studying the properties oh the graphs generated by this process when the parameter p changes.

3.1.1 Degree distribution

We begin our study of $G(n,p)$ graphs looking at their degree distribution.

Observation 3.1.1. The average degree of each node in G is:

$$\mathbb{E}[\deg(i)] = \sum_{j \neq i} \Pr\{\{i,j\} \in E\} = \sum_{j \neq i} p = (n-1)p \tag{3.1}$$

At this point we aren't sure if this information is valuable, since it could be that the actual degrees are far from the average (i.e. there is high variance), so we want to proof that, instead, those values are *concentrated* around the average.

Let $X = \deg(i) = \sum_{j \neq i} x_j$ where $x_j = \begin{cases} 1 & \text{if } \{i,j\} \in E(G) \quad \text{(with probability } p) \\ 0 & \text{otherwise} \quad \text{(with probability } 1-p) \end{cases}$
and the x_j are mutually independent.

Thanks to the mutual independence of the x_j, we can apply the Chernoff Bound [2.1] to prove our claim:

$$X = \sum x_j = \frac{n}{2} \pm \sqrt{n \ln \frac{1}{\delta}} \text{ with probability } 1 - O(\delta)$$

Thus, since in average each node has the same degree, $G(n,p)$ produces *almost regular graphs*, that are not suitable for representing real world social graphs, as stated at the beginning of this section.

3.1.2 Connectivity

Now we want to analize with which probability $G(n,p)$ will produce a connected graph $G = (V,E)$.

Theorem 3.1.1 (Connectivity of G(n,p)). *If* $p \geq \frac{8 \ln n}{n}$, *then* $G(n,p)$ *is connected with probability* $\geq 1 - \frac{1}{n}$.

Before going into the proof of the theorem, we introduce a combinatorial property that will be useful later.

Lemma 3.1.2. *If* $\forall\ \emptyset \subset S \subset V,\ \exists \{u,v\} \in E$ *such that* $u \in S,\ v \in V - S$, *then the graph is connected.*

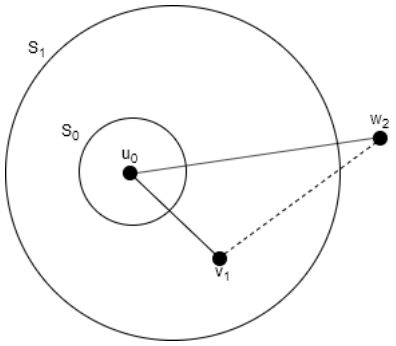


Figure 3.1: inductive proof of the lemma

Informal proof by induction.

- Base step: In S_0 there is only the node u_0, so there must be an edge from u_0 to a node $v_1 \in V - S$;

- Inductive step: Go on expanding the frontier (the set of nodes in S that have an edge connecting them to a node in $V - S$) until the BFS visit reaches all the nodes.

See the figure [3.1]. □

Proof of the theorem 3.1.1. Let ξ_S be the bad event "there are no edges in the cut $(S, V - S)$" for each proper subset S of V.

Thanks to the lemma [3.1.2], we know that $\Pr\{G(n,p) \text{ is not connected}\} = \Pr\left\{\bigcup_{S \subset V} \xi_S\right\}$.

Let's note that there are $2^n - 2$ of such subsets, but we are interested only in half of the corresponding events, since $\xi_s = S_{V-S}\ \forall\ S \subset V$ (the events ξ_i are **not** independent), so we will consider only the $\frac{1}{2}(2^n - 2)$ events ξ_S for which $|S| \leq \frac{|V|}{2}$.

Let $n := |V|$ and $s := |S|$, now we can compute the probability of a single event ξ_S:

$$\begin{aligned}
\Pr\{\xi_S\} &= (1-p)^{s\cdot(n-s)} && \text{(by } [*^1]\text{)}\\
&\leq e^{-p\cdot s\cdot(n-s)} && \text{(by [1.2])}\\
&\leq e^{-p\cdot s\cdot n/2} && \text{(since } s \leq n/2\text{)}\\
&\leq e^{-\frac{8\ln n}{n}\cdot s\cdot\frac{n}{2}}\\
&= \left(e^{-\ln n}\right)^{4\cdot s} = n^{-4\cdot s}
\end{aligned}$$

$[*^1]$ there are $s \cdot |V - S|$ edges from S to $V - S$, each of which does not appear w.p. $1 - p$, and $s \cdot |V - S| = s \cdot (n - s)$.

Now, let's consider the probability that ξ_S happens for any $S \subset V$:

$$\begin{aligned}
\Pr\left\{\exists S \in \binom{V}{s} \;\middle|\; \xi_s \text{ happens}\right\} &\leq \sum_{S\in\binom{V}{s}} \Pr\{\xi_S\} && \text{(by Union Bound [1.7])}\\
&\leq \sum_{S\in\binom{V}{s}} n^{-4s} = \binom{n}{s}\cdot n^{-4s} && \text{(since } \left|\binom{V}{s}\right| = \binom{n}{s}\text{)}\\
&\leq n^s \cdot n^{-4s} = n^{-3s} && \text{(since } \binom{n}{s} \leq n^s\text{)}
\end{aligned}$$

Now we are ready to compute the probability that G is connected. To do that we will distribute sets in "buckets" according to their cardinality:

$$\begin{aligned}
\Pr\{G(n,p) \text{ is not connected}\} &= \Pr\left\{\bigcup_{S\subset V}\xi_S\right\}\\
&= \Pr\left\{\exists s \in \{1,\dots,n\} \;\middle|\; \exists S \in \binom{V}{s} \;\middle|\; \xi_S \text{ happens}\right\}\\
&\leq \sum_{s=1}^{n/2} n^{-3s} && \text{(by Union Bound [1.7])}\\
&\leq \sum_{s=1}^{n/2} n^{-3} = \frac{1}{2n^2}
\end{aligned}$$

So we proved that $\Pr\{G(n,p) \text{ is connected}\} \geq 1 - \frac{1}{2n^2} \geq 1 - \frac{1}{n}$, so we showed a stronger claim than the one we wanted to prove. □

Observation 3.1.2. The theorem we just proved is a so called *zero-one law*: It holds for the given bound of p, but it ceases to hold almost immediately when p goes under that bound, so $p = \frac{8 \ln n}{n}$ is sort of a threshold between $G(n,p)$s that produce connected graphs with high probability and $G(n,p)$s that produce **un**connected graphs with high probability.

We won't proof exactly what we just claimed, since it would be hard, but something weaker:

Theorem 3.1.3. *If* $p < \frac{\varepsilon}{n} \ll \frac{8 \ln n}{n}$*, then* $G(n,p)$ *is disconnected with probability* $\geq 1 - \varepsilon$.

Proof. Let $X := \deg(i)$ and x_j defined as before (see [3.1.1]).

1. $X = \sum_{j \neq i} x_j = \sum_{j \neq i} \Pr\{\{i,j\} \in E(G)\}$;
2. We know by hypothesis that $\mathbb{E}[X_j] = p < \frac{\varepsilon}{n}$;
3. $\mathbb{E}[X] < (n-1)\frac{\varepsilon}{n}$;
4. $\Pr\left\{X \geq \frac{1}{\varepsilon} \cdot \mathbb{E}[X]\right\} \leq \varepsilon$ (by Markov inequality [1.11]);
5. Since $\frac{1}{\varepsilon} \cdot \mathbb{E}[X] \approx 1$ and X has integer values, we can write the previous expression as $\Pr\{X > 0\} \leq \varepsilon$, and this is the probability that exists a node i that has at least one neighbor;
6. Thus, by complement, $\Pr\{\exists\, i \mid i \in V \text{ and } i \text{ has 0 neighbors}\} = \Pr\{G(n,p) \text{ is not connected}\} \geq 1 - \varepsilon$.

□

3.1.3 Diameter

It is known that with high probability $G(n,p)$ graphs have small diameter. The proof is left as an exercise.

3.2 Preferential attachment[1]

Preferential attachment is another model for generating random graphs, but it follows the rule "the rich get richer", meaning that nodes with higher degree will see their degree becoming higher and higher.

The preferential attachment is a generative (or sequential) model, and the rule used to build a graph is the following: you begin with a single node with a self loop, when you have built a graph with $N-1$ nodes, you add the N-th node with an edge that goes from N to a node i chosen accordingly with a probability proportional to the degree of i.

There exist many variants of this model, for example with each node creating k edges instead of one, with no self-loops, etc.

The graphs generated by the Preferential attachment model follow a power law degree distribution, like the one observed in many real world networks, indeed, the fraction of nodes of degree x approaches x^{-3}.

[1] Part of this section is taken from this repo by Cristian Di Pietrantonio.

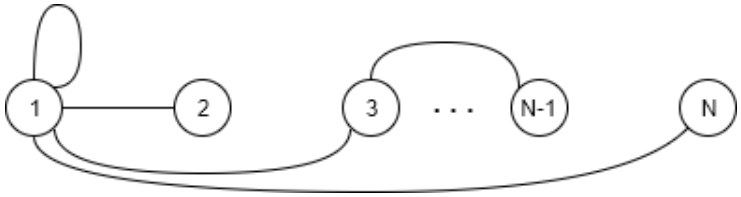


Figure 3.2: Example of a graph generated by the preferential attachment model

This characteristic causes that this model fits well with social and biological networks, so it can be used to develop efficient algorithms that actually work in practice.

Now we give a general definition of power law:

Definition 3.2.1 (Power law). *A power law is a functional relationship* $y = ax^{-c}$ *between two quantities, where one quantity varies as a power of the other.*

As a consequence of the definition, we get that a power law appears as a line in a log log scale plot, as can be seen in fig. 3.3.

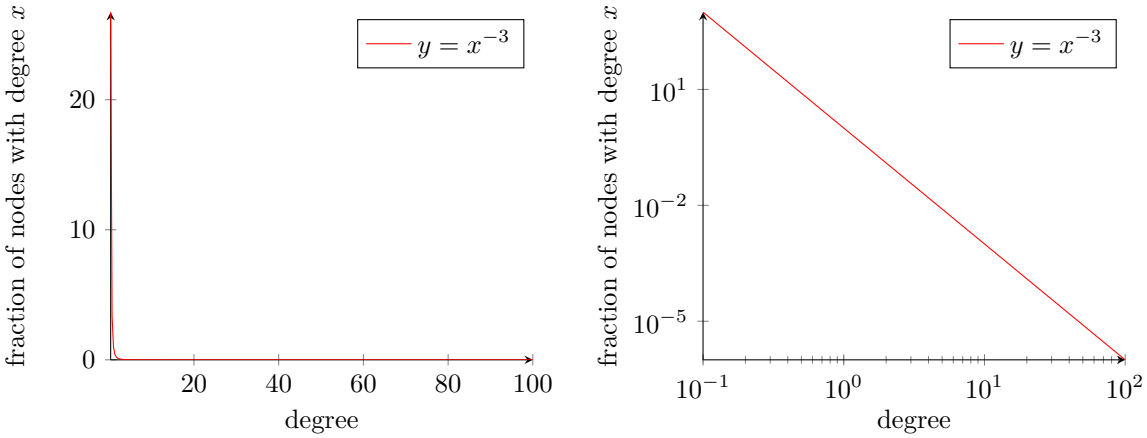


Figure 3.3: Power law distribution with linear and log log scale

In the social network context it means that it is exponentially more likely to pick "normal people" with few friends or followers rather than popular profiles, called "celebrities" or "authorities".
Nodes with high degree in a social network are few, but they exists. So, as an example, an advertisement agency could pay those celebrities to publicize a product, enabling a spread of information due to the high number of connections those nodes have.

3.2.1 Formalization

Inductive definition of the model:

- Base step: G_1 is a single node with a self loop;
- Inductive step (for $i = 2, 3, \ldots$):
  1. add node i to G_i;
  2. add a "*half edge*" coming out from node i;
  3. choose a node j randomly, with probability proportional to its degree, i.e., $\Pr\{\text{neighbor of } N \text{ is } i\} = \frac{deg(i)}{\sum_{k=1}^{N} deg(k)}$, where the denominator is a normalization factor;

4. close the *half edge* from i, by connecting it to j.

Note that there are many possible configurations from the second step onwards, and the probability of each of them depends on the degrees of the nodes (where a loop counts twice and a self loop once). We can see the possible configurations with their respective probabilities in picture [3.4].

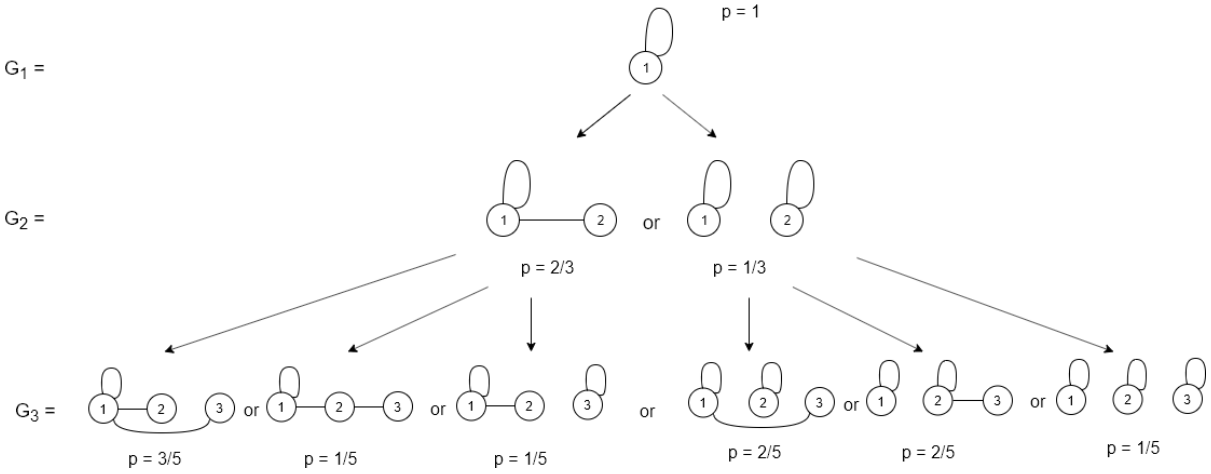


Figure 3.4: Example of a graph generated by the preferential attachment model

3.2.2 Degree distribution

We are interest in studying the degree distribution of the graphs generated by the preferential attachment model, so that we can show that it follows a power low, as previously claimed.

We begin with what we already know from the definition of the model.

Observation 3.2.1. $|V(G_t)| = t$.

Observation 3.2.2. $|E(G_t)| = t$.

Observation 3.2.3. There are no node of degree zero.

Observation 3.2.4. $\Pr\{v_{t+1} \sim v_i\} = \Pr\{\{v_{t+1}, v_i\} \in E(G_{t+1})\} = \dfrac{\deg_{G_t}(v_i)}{2t+1}$.

Observation 3.2.5. $\Pr\{v_{t+1} \sim v_{t+1}\} = \dfrac{1}{2t+1}$.

Now, our claim about the degree distribution.

Theorem 3.2.1. *Let $X_t^{(d)}$ be the number of nodes that have degree d in G_t, then* $E\left[X_t^{(d)}\right] = t \cdot \dfrac{4}{(d+2)\cdot(d+1)\cdot d} \pm O(1)$.

Note that is meaningful to study the expected value because the random variable X is concentrated.

Observation 3.2.6. $\lim\limits_{d\to\infty} \dfrac{\frac{4}{(d+2)\cdot(d+1)\cdot d}}{d^{-3}} = \Theta(1)$, in other words $E\left[X_t^{(d)}\right]^{d\to\infty} \to t \cdot d^{-3}$.

We are going to demonstrate the theorem by double induction, on the degree d and on the time t, so we fix the degree and we study the number of nodes when t changes.

For this proof, we have two base steps and an inductive step, for each of which we have different possible possibilities for the new node v_{t+1}:

1. Base step 1 ($d = 1$):
   1.1. v_{t+1} creates a self loop, so the number of nodes with deg $= 1$ doesn't change,
   1.2. v_{t+1} connects itself with a node of degree 1, in this case v_{t+1} is a new node with deg $= 1$, but the other node now have a greater degree, so the overall number of nodes with deg $= 1$ does not change,
   1.3. v_{t+1} connects itself with a node of degree 2 or more, in this case v_{t+1} is a new node with deg $= 1$, so the number of nodes with deg $= 1$ increases by one;
2. Base step 2 ($d = 2$):
   2.1. v_{t+1} creates a self loop, so the number of nodes with deg $= 2$ increases by one,
   2.2. v_{t+1} connects itself with a node of degree 1, so the number of nodes with deg $= 2$ increases by one,
   2.3. v_{t+1} connects itself with a node of degree 2, so the number of nodes with deg $= 2$ decreases by 1 (since now that node has a greater degree),
   2.4. v_{t+1} connects itself with a node of degree 3 or more, so the number of nodes with deg $= 2$ does not change;
3. Inductive step ($d \geq 3$):
   3.1. v_{t+1} connects itself with a node of degree $d-1$, so the number of nodes with deg $= d$ increases by one,
   3.2. v_{t+1} connects itself with a node of degree d, so the number of nodes with deg $= d$ decreases by one (since now that node has a greater degree),
   3.3. v_{t+1} connects itself to any other node, so the number of nodes with deg $= d$ does not change.

Base step 1: $d = 1$.

Since G_{t+1} depends on G_t, $\forall\, t = 1, 2, \ldots$, we will need to condition on all the previous random choices.

Lemma 3.2.2.

$$E\left[X_{t+1}^{(1)} \;\middle|\; X_t^{(1)} = x\right] = (x+1) \cdot \left(1 - \frac{1}{2t+1}\right) \tag{3.2}$$

Proof.

$$\begin{aligned} E\left[X_{t+1}^{(1)} \;\middle|\; X_t^{(1)} = x\right] &= \Pr\left\{X_{t+1}^{(1)} = x\right\} \cdot x + \Pr\left\{X_{t+1}^{(1)} = x+1\right\} \cdot (x+1) \\ &= \frac{x+1}{2t+1} \cdot x + \frac{(2t+1)-(x+1)}{2t+1} \cdot (x+1) \\ &= \frac{x+1}{2t+1} \cdot (x-(x+1)) + \frac{2t+1}{2t+1}(x+1) \\ &= -\frac{x+1}{2t+1} + (x+1) \\ &= (x+1) \cdot \left(1 - \frac{1}{2t+1}\right) \end{aligned}$$

□

Lemma 3.2.3.

$$E\left[X_{t+1}^{(1)}\right] = \left(E\left[X_t^{(1)}\right] + 1\right) \cdot \left(1 - \frac{1}{2t+1}\right) \tag{3.3}$$

Proof.

$$E\left[X_{t+1}^{(1)}\right] = \sum_{x\geq 0}\left(\Pr\left\{X_t^{(1)}=x\right\}\cdot E\left[X_{t+1}^{(1)}\;\middle|\;X_t^{(1)}=x\right]\right)$$

$$= \sum_{x\geq 0}\left(\Pr\left\{X_t^{(1)}=x\right\}\cdot(x+1)\cdot\left(1-\frac{1}{2t+1}\right)\right)$$

(by *law of total expectation* and lemma [3.2.2])

$$= \left(1-\frac{1}{2t+1}\right)\cdot\sum_{x\geq 0}\left(\Pr\left\{X_t^{(1)}=x\right\}\cdot(x+1)\right)$$

$$= \left(1-\frac{1}{2t+1}\right)\cdot\left(\sum_{x\geq 0}\left(\Pr\left\{X_t^{(1)}=x\right\}\cdot x\right)+\sum_{x\geq 0}\left(\Pr\left\{X_t^{(1)}=x\right\}\right)\right)$$

(by distributive law)

$$= \left(1-\frac{1}{2t+1}\right)\cdot\left(E\left[X_t^{(1)}\right]+1\right)$$

(since the green part corresponds to the probability that $X_t^{(1)}$ assumes any non negative value, that is 1)

□

Lemma 3.2.4.

$$\exists\; c>0 \;\; s.\;\; t.\;\; \forall\; t\geq 0,\;\; \frac{2}{3}(t+1)-c\leq E\left[X_{t+1}^{(1)}\right]\leq\frac{2}{3}(t+1)+c \tag{3.4}$$

Proof by induction.

Base step, with $t=0$:
At time $t+1$ the graph is G_1 and is composed by a single node with degree 2, as previously described (see [3.4]), so the claim holds for $c\geq\frac{2}{3}$:

$$\frac{2}{3}-c\leq E\left[X_1^{(1)}\right]=0\leq\frac{2}{3}+c$$

Inductive step for the upper bound: suppose that the claim holds for $E\left[X_t^{(1)}\right]$,

$$\begin{aligned}
E\left[X_{t+1}^{(1)}\right] &= \left(E\left[X_t^{(1)}\right]+1\right)\cdot\left(1-\frac{1}{2t+1}\right) && \text{(by lemma [3.2.3])}\\
&\le \left(\frac{2}{3}t+c+1\right)\cdot\left(1-\frac{1}{2t+1}\right) && \text{(by inductive hypothesis)}\\
&= \left(1-\frac{1}{2t+1}\right)\cdot\left(\frac{2}{3}t+c\right)+\left(1-\frac{1}{2t+1}\right)\\
&\le \left(1-\frac{1}{2t+1}\right)\cdot\left(\frac{2}{3}t+c\right)+1 && \text{(since } 1-\frac{1}{2t+1}\le 1\text{)}\\
&= \frac{2}{3}t+c-\frac{2t}{3(2t+1)}-\frac{c}{2t+1}+1\\
&= \frac{2}{3}t+c-\frac{2t}{6t+3}-\frac{c}{2t+1}+1\\
&= \frac{2}{3}t+\left(1-\frac{2t}{6t+3}\right)+c\cdot\left(1-\frac{1}{2t+1}\right)\\
&= \frac{2}{3}t+\frac{4t+3}{6t+3}+c\cdot\left(1-\frac{1}{2t+1}\right)\\
&= \frac{2}{3}t+\left(\frac{4t+2}{6t+3}+\frac{1}{6t+3}\right)+c\cdot\left(1-\frac{1}{2t+1}\right)\\
&= \frac{2}{3}(t+1)+c-\frac{c}{2t+1}+\frac{1}{6t+3} && \text{(since } \frac{4t+2}{6t+3}=\frac{2}{3}\text{)}\\
&\le \frac{2}{3}(t+1)+c && \text{(since, for } c\ge\frac{1}{3},\ -\frac{c}{2t+1}+\frac{1}{6t+3}\le 0\text{)}
\end{aligned}$$

Inductive step for the lower bound: suppose that the claim holds for $E\left[X_t^{(1)}\right]$,

$$\begin{aligned}
E\left[X_{t+1}^{(1)}\right] &= \left(E\left[X_t^{(1)}\right]+1\right)\cdot\left(1-\frac{1}{2t+1}\right) && \text{(by lemma [3.2.3])}\\
&\ge \left(\frac{2}{3}t-c+1\right)\cdot\left(1-\frac{1}{2t+1}\right) && \text{(by inductive hypothesis)}\\
&= \frac{2}{3}t+1-\frac{2t}{6t+3}-c\left(1-\frac{1}{2t+1}\right)-\frac{1}{2t+1}\\
&= \frac{2}{3}t+\left(1-\frac{2t}{6t+3}-\frac{1}{2t+1}\right)-c\left(1-\frac{1}{2t+1}\right)\\
&= \frac{2}{3}t+\left(\frac{6t+3-2t-3}{6t+3}\right)-c\left(1-\frac{1}{2t+1}\right)\\
&= \frac{2}{3}t+\frac{4t}{6t+3}-c\left(1-\frac{1}{2t+1}\right)\\
&= \frac{2}{3}t+\underbrace{\frac{4t}{6t+3}+\frac{2}{6t+3}}_{2/3}-\frac{2}{6t+3}-c-\frac{c}{2t+1}\\
&= \frac{2}{3}(t+1)-c+\frac{3c-2}{6t+3}\\
&\ge \frac{2}{3}(t+1)-c && \text{(since, for } c\ge\frac{2}{3},\ \frac{3c-2}{6t+3}\ge 0\text{)}
\end{aligned}$$

□

Base step 2: $d = 2$.

Lemma 3.2.5.

$$E\left[X_{t+1}^{(2)} \;\middle|\; X_t^{(2)} = x,\ X_t^{(1)} = y\right] = x \cdot \left(1 - \frac{2}{2t+1}\right) + y \cdot \frac{1}{2t+1} + \frac{1}{2t+1} \tag{3.5}$$

Proof.

$$\begin{aligned}
&E\left[X_{t+1}^{(2)} \;\middle|\; X_t^{(2)} = x,\ X_t^{(1)} = y\right] = \\
&= \Pr\left\{\begin{array}{l}\text{node } t+1 \text{ connects itself to some node that had}\\ \text{degree 1 in } G_t \text{ or to } t+1 \text{ itself}\end{array} \;\middle|\; X_t^{(2)} = x,\ X_t^{(1)} = y\right\} \cdot (x+1) \\
&\qquad\qquad \text{(we are adding a new node of degree 2)} \\
&\quad + \Pr\left\{\begin{array}{l}\text{node } t+1 \text{ connects itself to some node that had}\\ \text{degree 2 in } G_t\end{array} \;\middle|\; X_t^{(2)} = x,\ X_t^{(1)} = y\right\} \cdot (x-1) \\
&\qquad\qquad \text{(we are loosing a node of degree 2)} \\
&\quad + \Pr\left\{\begin{array}{l}\text{node } t+1 \text{ connects itself to some node that had}\\ \text{degree} \geq 3 \text{ in } G_t\end{array} \;\middle|\; X_t^{(2)} = x,\ X_t^{(1)} = y\right\} \cdot x \\
&\qquad\qquad \text{(the number of nodes of degree 2 does not change)} \\
&= \left(y \cdot \frac{1}{2t+1} + \frac{1}{2t+1}\right) \cdot (x+1) + \left(x \cdot \frac{2}{2t+1}\right) \cdot (x-1) + \left(1 - y \cdot \frac{1}{2t+1} - \frac{1}{2t+1} - \frac{2x}{2t+1}\right) \cdot x \\
&\qquad\qquad \text{(since we have three disjoint events, where the third one is the complement of the first two)} \\
&= \frac{y+1}{2t+1}x + \frac{y+1}{2t+1} + \frac{2x}{2t+1}x - \frac{2x}{2t+1} + \left(1 - \frac{y+1}{2t+1} - \frac{2x}{2t+1}\right) \\
&= x - \frac{2x}{2t+1} + \frac{y+1}{2t+1} \\
&= x \cdot \left(1 - \frac{2}{2t+1}\right) + \frac{y+1}{2t+1}
\end{aligned}$$

□

Lemma 3.2.6.

$$E\left[X_{t+1}^{(2)}\right] = E\left[X_t^{(2)}\right] \cdot \left(1 - \frac{2}{2t+1}\right) + \frac{E\left[X_t^{(1)}\right] + 1}{2t+1} \tag{3.6}$$

Proof. It is possible to prove this lemma explicitly with a procedure similar to the one used for lemma [3.2.3], but here we will present a more compact (implicit) proof:

$$E_{X_t^{(2)}=x}\left[E_{X_t^{(1)}=y \mid X_t^{(2)}=x}\left[x \cdot \left(1 - \frac{2}{2t+1}\right) + \frac{y+1}{2t+1}\right]\right]$$

Note that this is possible because the expected value is linear and (in this case) also its value is linear with respect to x and y. □

Now we are going to use the bounds found for $E\left[X_t^{(1)}\right]$ in lemma [3.2.4] to proof similar bounds for $E\left[X_{t+1}^{(2)}\right]$.

Lemma 3.2.7.

$$\exists\ c > 0\ \ s.\ t.\ \ \forall\ t \geq 0,\ \ \frac{1}{6}(t+1) - c \leq E\left[X_{t+1}^{(2)}\right] \leq \frac{1}{6}(t+1) + c \tag{3.7}$$

Proof by induction.

Base step, with $t = 0$:
At time $t+1$ the graph is G_1 and is composed by a single node with degree 2, as previously described (see [3.4]), so the claim holds for $c \geq \frac{5}{6}$:

$$\frac{1}{6} - c \leq E\left[X_1^{(2)}\right] = 1 \leq \frac{1}{6} + c$$

Inductive step for the upper bound: suppose that the claim holds for $E\left[X_t^{(2)}\right]$,

$$\begin{aligned}
E\left[X_{t+1}^{(2)}\right] &= E\left[X_t^{(2)}\right] \cdot \left(1 - \frac{2}{2t+1}\right) + \frac{E\left[X_t^{(1)}\right] + 1}{2t+1} && \text{(by lemma [3.2.6])} \\
&\leq \left(\frac{t}{6} + c\right) \cdot \left(1 - \frac{2}{2t+1}\right) + \frac{\frac{2t}{3} + c + 1}{2t+1} && \text{(by inductive hypothesis and upper bound given by [3.2.4])} \\
&= \frac{t}{6} - \frac{1}{2t+1} \cdot \frac{t}{3} + c \cdot \left(1 - \frac{2}{2t+1}\right) + \frac{2t}{3} \cdot \frac{1}{2t+1} + \frac{c}{2t+1} + \frac{1}{2t+1} \\
&= \frac{t}{6} + \frac{1}{2t+1} \cdot \left(\frac{2}{3}t - \frac{1}{3}t\right) + c \cdot \left(1 - \frac{2-1}{2t+1}\right) + \frac{1}{2t+1} \\
&= \frac{t}{6} + \frac{1}{2t+1} \cdot \frac{t}{3} + c \cdot \left(1 - \frac{1}{2t+1}\right) + \frac{1}{2t+1} \\
&= \frac{t}{6} + \frac{t}{6t+3} + \frac{3}{6t+3} + c \cdot \left(1 - \frac{1}{2t+1}\right) \\
&= \frac{t}{6} + \frac{t+1/2}{6t+3} + \frac{5/2}{6t+3} + c \cdot \left(1 - \frac{1}{2t+1}\right) \\
&= \frac{t+1}{6} + \frac{5/2}{6t+3} + c - \frac{3c}{6t+3} && \text{(since } \frac{t+1/2}{6t+3} = \frac{1}{6}\text{)} \\
&\leq \frac{t+1}{6} + c && \text{(since } \frac{5/2}{6t+3} - \frac{3c}{6t+3} \leq 0 \text{ if } c \geq \frac{5}{6}\text{)}
\end{aligned}$$

Inductive step for the lower bound: suppose that the claim holds for $E\left[X_t^{(2)}\right]$,

$$\begin{aligned}
E\left[X_{t+1}^{(2)}\right] &= E\left[X_t^{(2)}\right] \cdot \left(1 - \frac{2}{2t+1}\right) + \frac{E\left[X_t^{(1)}\right] + 1}{2t+1} && \text{(by lemma [3.2.6])} \\
&\geq \left(\frac{t}{6} - c\right) \cdot \left(1 - \frac{2}{2t+1}\right) + \frac{\frac{2t}{3} - c + 1}{2t+1} && \text{(by inductive hypothesis and lower bound given by [3.2.4])} \\
&= \frac{t}{6} + \frac{1}{2t+1} \cdot \left(\frac{2}{3}t - \frac{1}{3}t\right) - c \cdot \left(1 - \frac{2-1}{2t+1}\right) + \frac{1}{2t+1} \\
&= \frac{t}{6} + \frac{t+3}{6t+3} - c \cdot \left(1 - \frac{1}{2t+1}\right) \\
&\geq \frac{t}{6} + \frac{t+1/2}{6t+3} - c && (*) \\
&= \frac{t+1}{6} - c && \text{(since } \frac{t+1/2}{6t+3} = \frac{1}{6}\text{)}
\end{aligned}$$

Note that most steps are analogous to those in the previous part of the proof (the inductive step for the upper bound), so we skipped some intermediate steps, while the step marked with $*$ is due to the fact that we subtracted the positive value $\frac{5/2}{6t+3}$ to $\frac{t+3}{6t+3}$ and divided $c \cdot \left(1 - \frac{1}{2t+1}\right)$ by its second factor, that is < 1. □

Inductive step: $d \geq 3$.

Lemma 3.2.8.

$$E\left[X_{t+1}^{(d)} \;\middle|\; X_t^{(d)} = x,\ X_t^{(d-1)} = y\right] = x \cdot \left(1 - \frac{d}{2t+1}\right) + y \cdot \frac{d-1}{2t+1} \tag{3.8}$$

Note that the third term in [3.2.5] is absent here, since an eventual self loop wouldn't affect the number of nodes of degree d, with $d \geq 3$.

Proof.

$$E\left[X_{t+1}^{(d)} \;\middle|\; X_t^{(d)} = x,\ X_t^{(d-1)} = y\right] =$$

$$= \Pr\left\{\begin{matrix}\text{node } t+1 \text{ connects itself to some node that had} \\ \text{degree } d-1 \text{ in } G_t\end{matrix} \;\middle|\; X_t^{(2)} = x,\ X_t^{(1)} = y\right\} \cdot (x+1)$$

(we are adding a new node of degree d)

$$+ \Pr\left\{\begin{matrix}\text{node } t+1 \text{ connects itself to some node that had} \\ \text{degree } d \text{ in } G_t\end{matrix} \;\middle|\; X_t^{(2)} = x,\ X_t^{(1)} = y\right\} \cdot (x-1)$$

(we are loosing a node of degree d)

$$+ \Pr\left\{\begin{matrix}\text{node } t+1 \text{ connects itself to some node (possibly} \\ \text{itself) that had degree} \neq d \text{ and} \neq d-1 \text{ in } G_t\end{matrix} \;\middle|\; X_t^{(2)} = x,\ X_t^{(1)} = y\right\} \cdot x$$

(the number of nodes of degree d does not change)

$$= \frac{d-1}{2t+1} \cdot y \cdot (x+1) + \frac{d}{2t+1} \cdot x \cdot (x-1) + \left(1 - \frac{(d-1)y}{2t+1} - \frac{dx}{2t+1} - \frac{2x}{2t+1}\right) \cdot x$$

(since we have three disjoint events, where the third one is the complement of the first two)

$$= x \cdot \left(1 - \frac{d}{2t+1}\right) + y \cdot \frac{d-1}{2t+1}$$

(since we simplify similar terms)

□

Lemma 3.2.9.

$$E\left[X_{t+1}^{(d)}\right] = E\left[X_t^{(d)}\right] \cdot \left(1 - \frac{d}{2t+1}\right) + E\left[X_t^{(d-1)}\right] \cdot \frac{d-1}{2t+1} \tag{3.9}$$

Proof. It is possible to prove this lemma explicitly as we did for lemma [3.2.3], or implicitly as we did for lemma [3.2.6]. □

Lemma 3.2.10.

$$\frac{4(t+1)}{(d+2)(d+1)d} - c \leq E\left[X_{t+1}^{(d)}\right] \leq \frac{4(t+1)}{(d+2)(d+1)d} + c \tag{3.10}$$

Proof by induction.

Base step, with $t = 0$:
At time $t+1$ the graph is G_1 and is composed by a single node with degree 2, as previously described (see [3.4]), so the claim holds for $c \geq 1$:

$$0 \leq E\left[X_{t+1}^{(d)}\right] = 0 \leq 1$$

Inductive step for the upper bound: suppose that the claim holds for $E\left[X_t^{(d)}\right]$,

$$
\begin{aligned}
E\left[X_{t+1}^{(d)}\right] &= E\left[X_t^{(d)}\right]\cdot\left(1-\frac{d}{2t+1}\right)+E\left[X_t^{(d-1)}\right]\cdot\frac{d-1}{2t+1} && \text{(by lemma [3.2.9])}\\
&\le \left(\frac{4t}{(d+2)(d+1)d}+c\right)\cdot\left(1-\frac{d}{2t+1}\right)+\left(\frac{4t}{(d+1)(d-1)d}+c\right)\cdot\frac{d-1}{2t+1} && \text{(by inductive hypothesis)}\\
&= \frac{4t}{(d+2)(d+1)d}-\frac{4t}{(d+2)(d+1)(2t+1)}+c\cdot\left(1-\frac{d}{2t+1}\right)+\frac{4t}{(d+1)(2t+1)d}+c\cdot\frac{d-1}{2t+1}\\
&= \frac{4t}{(d+2)(d+1)d}+\frac{4t}{2t+1}\cdot\left(\frac{1}{(d+1)d}-\frac{1}{(d+1)(d+2)}\right)+c\cdot\left(1-\frac{d}{2t+1}+\frac{d-1}{2t+1}\right)\\
&= \frac{4t}{(d+2)(d+1)d}+\frac{4t}{2t+1}\cdot\frac{d+2-d}{(d+2)(d+1)d}+c\cdot\left(1-\frac{1}{2t+1}\right)\\
&\le \frac{4t}{(d+2)(d+1)d}+\frac{8t+4}{(2t+1)(d+2)(d+1)d}+c && \text{(since } 1-\frac{1}{2t+1}<1\text{)}\\
&= \frac{4t}{(d+2)(d+1)d}+\frac{4}{(d+2)(d+1)d}+c && \begin{array}{l}\text{we added } +4 \text{ to the numerator of the second}\\ \text{(fraction, since this allows us the simplification)}\\ \text{and doesn't affect the upper bound}\end{array}\\
&= \frac{4(t+1)}{(d+2)(d+1)d}+c
\end{aligned}
$$

Inductive step for the lower bound: suppose that the claim holds for $E\left[X_t^{(d)}\right]$,

$$
\begin{aligned}
E\left[X_{t+1}^{(d)}\right] &= E\left[X_t^{(d)}\right]\cdot\left(1-\frac{d}{2t+1}\right)+E\left[X_t^{(d-1)}\right]\cdot\frac{d-1}{2t+1} && \text{(by lemma [3.2.9])}\\
&\le \left(\frac{4t}{(d+2)(d+1)d}-c\right)\cdot\left(1-\frac{d}{2t+1}\right)+\left(\frac{4t}{(d+1)(d-1)d}-c\right)\cdot\frac{d-1}{2t+1} && \text{(by inductive hypothesis)}\\
&= \frac{4t}{(d+2)(d+1)d}-\frac{4t}{(d+2)(d+1)(2t+1)}-c\cdot\left(1-\frac{d}{2t+1}\right)+\frac{4t}{(d+1)(2t+1)d}-c\cdot\frac{d-1}{2t+1}\\
&= \frac{4t}{(d+2)(d+1)d}+\frac{4t}{2t+1}\cdot\left(\frac{1}{(d+1)d}-\frac{1}{(d+1)(d+2)}\right)-c\cdot\left(1-\frac{1}{2t+1}\right)\\
&= \frac{4t}{(d+2)(d+1)d}+\frac{4t}{2t+1}\cdot\frac{2}{(d+2)(d+1)d}-c+\frac{c}{2t+1}\\
&= \frac{4t}{(d+2)(d+1)d}+\frac{8t+4}{(2t+1)(d+2)(d+1)d}-\frac{4}{(2t+1)(d+2)(d+1)d}+\frac{c}{2t+1}-c && (*^1)\\
&= \frac{4t}{(d+2)(d+1)d}+\frac{4}{(d+2)(d+1)d}-\frac{4}{(2t+1)(d+2)(d+1)d}+\frac{c}{2t+1}-c\\
&= \frac{4(t+1)}{(d+2)(d+1)d}+\frac{1}{2t+1}\left(c-\frac{4}{(d+2)(d+1)d}\right)-c\\
&\ge \frac{4(t+1)}{(d+2)(d+1)d}-c && (*^2)
\end{aligned}
$$

Note that, in the step marked by $*^1$, we added and subtracted $\dfrac{4}{(2t+1)(d+2)(d+1)d}$ so that we could simplify the second fraction.
In the last step, marked with $*^2$, we subtracted a positive term, since $\dfrac{4}{(d+2)(d+1)d}\le\dfrac{4}{5\cdot4\cdot3}=\dfrac{1}{15}\le c$, for $d\ge3$, so the inequality holds. □

This finally concludes the proof of the theorem [3.2.1].

Chapter 4

Information spreading [1]

On the web, information is published by one or more *sources* and often spreads quickly to other parties. The most popular kind of information that spreads quickly is a meme. From Wikipedia:

Definition 4.0.1 (Meme). *A meme is an idea, behavior, or style that spreads from person to person within a culture - often with the aim of conveying a particular phenomenon, theme or meaning represented by the meme.*

In this chapter we are going to define some mathematical models that describes the spread of information.
Let's note that in this case, unlike in the models seen in section [3], the models work on the graph without changing it.

4.1 Push

First, a source node picks a neighbor UAR and sends to it the meme.

Then, each node that contains the meme choose a random neighbor and sends to it the meme, iteratively.

4.2 Pull

As in the previous model, the information starts in a single node.

Each node x in the network, that doesn't contain the meme, chooses a neighbor y UAR; if y has the information, x obtains the information too.

The procedure repeats many times, but the upper bound of the number of iterations is given by the number of rounds necessary to spread the ifnromation in the whole graph, that depends on the structure of the graph. For example, a star will require much less time than a chain.

This model can be useful to choose the best node to advertise a product.

[1] Most of this chapter is taken from this repo by Cristian Di Pietrantonio.

4.3 The Trace Spreading Model

In this model for meme flow, nodes can receive or take information from other nodes, enabling the spread of the meme. They can also publish the same meme independently (without "copying" each other).

Whenever a node publishes information, a timestamp of the event is also available.

At the end, considering a particular meme, all we can see is a bunch of nodes having published with an associated timestamp. What we would like to know is the *social graph* that links these nodes, i.e., who retrieves information from who. Clearly, we can't obtain a perfect reconstruction of the graph, but we can approximate it based on *cascades* or *traces* of the memes.

Definition 4.3.1 (Trace). *A trace is a sequence of nodes ordered by their timestamp.*
Usually we write it as "v_1 `->` v_1 `->` v_2 `->` t_2 `->` $\cdots$ `->` t_{n-2} `->` v_n*", where v_i are nodes (or vertex of the social graph) and t_i are times elapsed from v_i to v_{i+1}.*

Define an undirected graph $G = (V, E)$, according to the Trace spreading model:

1. Let V be the set of nodes that holds the information of interest;
2. Pick a single source of information, chosen UAR;
3. $e = \{v, v'\} \in E$ iif information propagated from v to v', $v, v' \in V$ (the actual direction is not important, as you will notice later);
4. Each edge $e = \{v, v'\}$ has a *weight* $w(e)$ sampled IIDin $Exp(\lambda)$, which represent the time needed for the information to propagate from v to v' (in fact, for our purpose, any **continuous** distribution will do);
5. Build a trace following the shortest path tree from the source.

From this model and the relative traces, we want to find an efficient algorithm that reconstructs the underlying graph with a certain confidence.

Question 4.3.1. How may traces do we need to reconstruct the graph?

4.4 The First-Edge Algorithm

We can reconstruct (in most cases) the graph $G = (V, E)$ with high probability with a simple algorithm:

```
FirstEdge(π_1, π_2, ..., π_t) //the set of traces
        E ← ∅ //the set of edges in the graph
        for i = 1, ..., t //t = number of traces
                E ← E ∪ {(π_i(1), π_1(2))}
```

Listing 4.1: The first-edge algorithm

Since the algorithm adds one edge for each trace, intuitively we would need as many traces as edges, to find all the edges.

What the algorithm does is simply adding an edge between the first and second node in a trace, for every trace. That is because we can only be sure about the existence of that edge: the first node in a trace is the source (for that trace), since it has the lowest timestamp, the second node, which has the second lowest timestamp, could only get the information from the first node.

Example 4.4.1. Let pick the execution of the trace spreading model [4.3] in [4.1] as an example: the node A, marked with a double line, is chosen as the source, then each edge is assigned a random weight, then the shortest path is followed and each node is labeled with the timestamp at which the meme reaches it. The resulting trace is π_1 `= A -> C -> B -> D`.

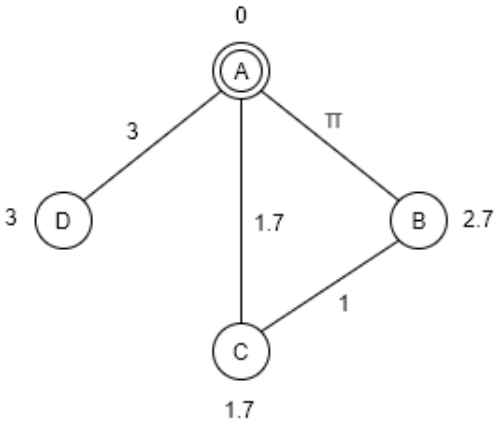


Figure 4.1: First edge example

The following traces could be:
π_2 `= A -> B -> D -> C`,
π_3 `= C -> B -> A -> D`,
π_1 `= D -> A -> B -> C`,
π_1 `= A -> D -> B -> C`.

Theorem 4.4.1 (First Edge correctness). *Let $G = (V, E)$, $n = |V|$, Δ = max degree of G =, t = number of traces. The output of the algorithm [4.1] will be equal to $E(G)$, i.e., the algorithm will be correct, with probability $\geq 1 - \frac{1}{n^2}$ if $t \geq 2n\Delta \ln(n)$.*

The high level meaning of the theorem is: the algorithm will give an exact answer if it is fed with enough traces.

Note that each trace is generated IIDwith the trace spreading model in [4.3], by assigning new random weights and choosing a new random source.

Lemma 4.4.2. *Let $u, v \in E(G)$, $n = |V|$; then, Pr{a random trace π begins with u, v}* $= \frac{1}{n \deg(u)}$.

Proof.

$$
\begin{aligned}
&\Pr\{\pi \text{ begins with } u, v\} = \\
&= \Pr\{\text{the random soursce is } u, \text{ and } v \text{ is the first neighbor of } u \text{ to be informed}\} \\
&= \Pr\{\text{the soursce is } u\} \cdot \Pr\{\{u, v\} \text{ is the shortest edge incident on } u \mid \text{source is } u\} \qquad \text{(by Bayes [1.6])} \\
&= \frac{1}{n} \cdot \frac{1}{\deg u}
\end{aligned}
$$

Note that, in the last step, $1/n$ is due to the fact that the source is chosen UAR, and $1/\deg(u)$ to the fact that each edge outgoing from u has the same probability of being the shortest, since all the weights of these edges are taken from the same probability distribution (*symmetry argument*), and the probability that two edges have the same weight is 0, since that distribution is continuous. □

Lemma 4.4.3. *For any $u, v \in E(G)$, Pr{a random trace π begins with u, v or v, u}* $\geq \frac{2}{n\Delta}$.

Proof.

$$\begin{aligned}
&\Pr\{\pi \text{ begins with } u,v \text{ or } v,u\} = \\
&= \Pr\{\pi \text{ begins with } u,v\} + \Pr\{\pi \text{ begins with } v,u\} - \Pr\{\pi \text{ begins with } u,v \textbf{ and } \text{it begins with } v,u\} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by [1.8])} \\
&= \Pr\{\pi \text{ begins with } u,v\} + \Pr\{\pi \text{ begins with } v,u\} \qquad\qquad \text{(since the third probability is 0)} \\
&\frac{1}{\deg(u)} + \frac{1}{\deg(u)} \geq \frac{2}{n\Delta} \qquad\qquad \text{(because } \Delta \text{ is greater or equal than any degree)}
\end{aligned}$$

□

Now our aim is to demonstrate that every $u, v \in E(G)$ appears at the beginning of a trace with positive probability, so that the expected value will be sufficiently high with enough traces.

Lemma 4.4.4. *Let $u, v \in E(G)$, suppose we are given $t \geq 2n\Delta\ln(n)$ traces,*
Pr{at least one trace begins with u, v or v, u} $\geq 1 - \frac{1}{n^4}$.

Proof. Let A_i be the event "*trace i does not begin with u, v, nor it does begin with v, u*".

$$\begin{aligned}
&\Pr\{A_1 \wedge A_2 \wedge \ldots \wedge A_t\} = \\
&= \Pr\{u, v \text{ or } v, u \text{ never appear at the beginning of a trace}\} \\
&= \prod_{i=0}^{n} \Pr\{E_i\} \qquad\qquad \text{(by [1.4])} \\
&\leq \left(1 - \frac{2}{n\Delta}\right)^t \\
&\leq \left(e^{-\frac{2}{n\Delta}}\right)^t \qquad\qquad \text{(by [1.2])} \\
&\leq \left(e^{-\frac{2}{n\Delta}}\right)^{2n\Delta\ln(n)} \qquad\qquad \text{(because } 2n\Delta\ln(n) \text{ is the smallest value that } t \text{ may assume)} \\
&= e^{-4\ln(n)} = \frac{1}{n^4} \qquad\qquad \text{(by [1.1])}
\end{aligned}$$

Therefore the lemma is demonstrated, since we proved that the probability of the complement of our claim is $1/n^4 = 1 - (1 - 1/n^4)$. □

Proof of Theorem 4.4.1. Let $B_{u,v}$ be the event "*no trace begins with u, v or v, u*" = "*the algorithm [4.1] doesn't learn about the edge u, v*".

$$\Pr\left\{\bigvee_{\{u,v\}\in E(G)} B_{\{u,v\}}\right\} \leq \sum_{\{u,v\}\in E(G)} \Pr\left\{B_{\{u,v\}}\right\} \leq \sum_{\{u,v\}\in E(G)} n^{-4} \leq n^2 \cdot n^{-4} = n^{-2}$$

where n^2 is an upper bound of $|E(G)|$.
The theorem is shown, since the complement of the bad event is what we were looking for. □

Chapter 5

Locality Sensitive Hashing

The goal of hashing techniques is to reduce a big "object" to a small "signature" or "fingerprint".
In general, what happens in locality sensitive hashing (or LSH) is to have some notion of similarity, and then define a "scheme" which computes it. The process of creating a scheme usually involves some sort of preprocessing step, and a function family which, by choosing one or another function according a probability distribution, statistically classifies the objects in the same way as the similarity function does.
The bottom line of LSH schemes is: similar objects hash to similar values (this is "*locality*").

Here are some common similarities:

Definition 5.0.1 (Jaccard similarity). *Given two sets of objects A and B, their Jaccard similarity is defined as follows:*

$$\mathcal{J}acc(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{5.1}$$

Definition 5.0.2 (Hamming similarity). *Given two sets of objects A and B taken from a universal set U, their Hamming similarity is defined as follows:*

$$\mathcal{H}amm(A,B) = \frac{|A \cap B| + \left|\overline{A \cup B}\right|}{|U|} \tag{5.2}$$

Observation 5.0.1. Generally, the Jaccard similarity is more used then the Hamming similarity, because usually we have to compare sets whose size is much smaller than the size of the universe set U, this, using $\mathcal{H}amm$, we would obtain a high similarity because of the big size of $\overline{A \cup B}$.

5.1 A case study: Web-page indexing

A search engine crawls periodically the whole Internet and stores valuable information in its own index for search optimization purposes.

Observation 5.1.1. Some kinds of documents, that are very similar to each other, are stored sparsely through the net; to save storage space, only one of a kind of document's info is stored in the index, whereas all others are linked to the first one, because of their similarity.

To find a useful hashing scheme, A. Broder came up with an idea, and he succeeded in reducing the storage space needed by Altavista by a factor of 10.

First off, let us fix some definitions:

- U: the set of all words, i.e. the English vocabulary

- U^*: the set of all strings composed of English words

The starting point is to treat web pages as strings:

T_1 : *"The black board is black"*

T_2 : *"The white board is white"*

Then, let $distinct(T)$ return the set of distinct words appearing in a string (ref. Bag of Words model), and let $A := distinct(T_1),\ B := distinct(T_2)$.

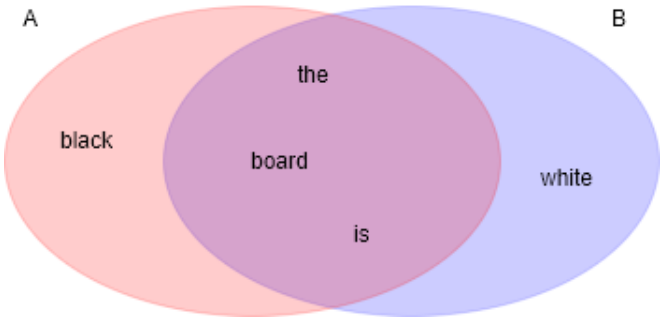


Figure 5.1: Venn diagram with the BoW representation of T_1 and T_2.

So for example, by using the Jaccard similarity:

$$\mathcal{J}acc(A, B) = \frac{3}{5} \tag{5.3}$$

Over a half: they look close. If we used the Hamming distance instead, we would (almost always) get a number very close to 1, because we're using a minuscule part of the universe set (in our case, the English dictionary), thus (almost) all words are absent from the sets.

Now, our objective is to construct a scheme over web-pages that implement the Jaccard similarity. Our pre-processing step: Choose a permutation (or total ordering) $\pi \in \mathcal{P}erm(A \cup B)$ UAR. To construct said order is a simple task, as we can see with the following algorithm.

Algorithm for sampling a UAR permutation π of $[n]$, where $[n]$ is a numeric representation of U, and using it to compute the hash of A:

```
MinHash(A):
    // preprocessing
    S := [n]
    π := empty sequence
    while S ≠ ∅:
        pick i ∈ S UAR
        append i to π
        remove i from S
    end
    // computing hash
    h_π(A) := minimum element of A, accordig to π
    return h_π(A)
```

Listing 5.1: min hash or shingles algorithm

Proof of uniform choice of π.

$$\begin{aligned}\Pr\{X = \pi\} &= \frac{1}{s} \cdot \frac{1}{s-1} \cdot \cdots \cdot 1 \\ &= \frac{1}{s!} \Rightarrow X \in \mathcal{U}nif(\mathcal{P}erm(A \cup B))\end{aligned}$$

□

So, from π, we obtained the following definition:

Definition 5.1.1 (Hashing function).

$$h_\pi \in \mathcal{P}(U) \to U : h_\pi(A) = min_\pi(A) \tag{5.4}$$

In other words, we take the "minimum" in A according to the ordering specified by π.

Observation 5.1.2. A simple but useful observation would be:

$$\forall A \subseteq U \Rightarrow h_\pi(A) \in A \tag{5.5}$$

Example 5.1.1.

$$\pi = (black, the, is, white, board) \wedge A = \{the, black, board, is\} \Rightarrow h_\pi(A) = black$$

Thus, we say that A is similar to B iff $h_\pi(A) = h_\pi(B)$.
Recall that A and B are fixed, π is the focus of this definition. What can be said about $\Pr\{h_\pi(A) = h_\pi(B)\}$? Looking at a corresponding Venn diagram [5.1]:

- $A \cap B = \emptyset \Rightarrow \Pr\{h_p(A) = h_p(B)\} = 0$; they have no words in common, so their hashes must be different, independently of the chosen order;
- $A = B \Rightarrow \Pr\{h_p(A) = h_p(B)\} = 1$; this time, all words are in common, so their hashes must coincide, again, independently of the chosen order;
- Otherwise, since π is chosen UAR, the probability that the hashes are equal has the same meaning of the probability of finding the lowest element of A and B in the intersection with respect of the union (and not in U as a whole, as our previous observation suggests), which is the Jaccard similarity of A and B by its very definition:

$$\begin{aligned}\Pr\{h_\pi(A) = h_\pi(B)\} &= \frac{\Pr\{\min(A) \in A \cap B \wedge \min(B) \in A \cap B\}}{\Pr\{\min(A) \in A \cup B \wedge \min(B) \in A \cup B\}} \\ &= \frac{|A \cap B|}{|A \cup B|} = \mathcal{J}acc(A, B).\end{aligned}$$

Possible question about third point: Why not respect to the universal set? because A and B will have hashes which, as we observed earlier, do not live outside the union: the union between A and B is our true set of outcomes when hashing either A or B.

Now, if h_π is evaluated only once over a given permutation, only a binary response can be obtained. In order to obtain the probability value without resorting to compute unions and intersections, we can repeat evaluation over different permutations; this can be regulated by the **Chernoff-Hoeffding bound**:

Let $A, B \subseteq U$, and $X_{1\ldots n} \in \mathcal{B}er(p)$ IID, with $\Pr\{X_i = 1\} = p$ and $\Pr\{X_i = 0\} = 1 - p$, with each X_i defined over a distinct element of $\Pi \subseteq \mathcal{P}erm(U)$, such that $X_i \mapsto 1 \Leftrightarrow A \sim_{\pi_i} B, 0$ *otherwise*, then:

$$\Pr\{|avg_{i=1}^{n}(X_i - p)| \geq \varepsilon\} \leq 2e^{-n\varepsilon^2} \tag{5.6}$$

In other words, the difference between the average of the X_i (i.e., the average of the empirically observed results) and their exact probability is greater then ε only with a very small probability.

So, how many trials (evaluations, observations) are needed to have a good estimate of the similarity? That is, what is a good value for n?
Let $X_i = \begin{cases} 1 & \text{if } h_{\pi_i}(A) = h_{\pi_i}(B) \\ 0 & \text{otherwise} \end{cases}$ and $\Pr\{X_i = 1\} = \mathcal{J}acc(A, B) = p$; we can apply the Chernoff bound

on X_i to compute our n.
If our database has m pages (sets) to store, we can chose $n = \frac{\lg \frac{2m}{\delta}}{\varepsilon^2}$ to get a high probability of making zero errors; δ and ε are parameters we can set to adjust the size of n: even if the bound gives us a high probability for a quite small n, we can choose an even smaller n if we can accept big errors for very few pages.

We can now observe that the min hash algorithm [5.1] is efficient: instead of comparing two entire pages, it only compares n integers.

5.2 A case study: Comparing DNAs

In the previous case study, we considered small subsets of the universe set: each web page has only few words with respect to the whole English language.
However, there are cases in which the overlays between two subsets are often relevant, for example, if we want to compare the DNA of two people.
In such cases, the Hamming similarity is preferable (more significant) to the Jaccart similarity.

We can describe two DNA sequences A and B as two arrays of size n, in which each position corresponds to a certain component of the DNA, and contains a 1 if that component is present in that sequence and a 0 if it's absent.

$$\begin{aligned}\mathcal{H}amm(A,B) &= \frac{|A \cap B| + \overline{|A \cup B|}}{n} \\ &= \frac{\text{number of common } 1s + \text{number of common } 0s}{n}\end{aligned}$$

To get our hash function we pick an index $i \in [n]$ UAR, and we define
$h_i(A) = \begin{cases} 1 & \text{if } i \in A \\ 0 & \text{otherwise} \end{cases}$.

Example 5.2.1. We have two DNA sequences A and B such that $A :=$ | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |,
$B :=$ | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | and $n := 8$, so we can write A and B as $A = \{4, 5, 8\}$,
$B = \{2, 3, 6, 7, 8\}$ using the positions with a 1 inside (starting from 1).
If we randomly choose $i = 8$, we obtain $h_i(A) = h_i(B) = 1$, so we can conclude that A and B are similar.

Now we can see that the probability that two hashes are equal is the Hamming similarity:

$$\begin{aligned}\Pr\{h_i(A) = h_i(B)\} &= \frac{\Pr\{A[i] = B[i]\}}{n} \\ &= \frac{\Pr\{(A[i] \text{ and } B[i] \text{ are both } 1) \vee (A[i] \text{ and } B[i] \text{ are both } 0)\}}{n} \\ &= \frac{|A \cap B| + \overline{|A \cup B|}}{n} = \mathcal{H}amm(A,B)\end{aligned}$$

It's possible to obtain a good estimate of the similarity by repeating the test an appropriate number of time, given by the Chernoff Bound.

5.3 LSH formalization

First let us focus around the hash function as an object: its true purpose in a scheme is to classify objects based on how much they "look like", whatever this means in the chosen similarity's terms. Therefore, in our theoretical analysis, the codomain of a hash function is not that important; what is important is

how the function partitions its own domain, U. In a sense, we're interested only in the partitions of U themselves, not in the functions that generate them.

Why have we dealt with functions back then? Moving from a purely mathematical perspective to a more computational one, what is usually done for measuring similarities is sampling some object's characteristics, and observe how "distant", or else "similar" they are. This is done by means of some program; and programs are (oh so) easily associated with functions. The computational approach gives a more intuitive vision of the problem we're confronting ourselves with.

Still, what could happen, is to have a couple of functions that map values into wildly different codomains, but partition U in exactly the same way! And in our journey, we're just interested in classifying objects; so these kind of "duplicate" functions are, well, useless (unless we delve in complexity studies, but that's out of our scope).

So, let us reform the foundations by taking as our core object a universe partition, instead of a universe-domained hash function. First, though, we need to formalize what a similarity is, and to get to a good definition, we have to carefully select them from their space $U^2 \to [0,1]$. It should be noted that the codomain might very well be $\mathbb{R}$ itself, but to get some bearings we'll treat an image of 1 as a complete equivalence between two objects, and 0 for complete difference, with the interval expressing the degree of similarity.

Let U be a set, and $S \in U^2 \to [0,1]$ a symmetric function; then S is called a **similarity** over U.

Tidbit: Let $f \in A^n \to B$, then f is **symmetric** iff *argument order does not change the image.*

Example 5.3.1.

$$U = 2^{[n]} = \{A | A \subseteq [n]\}$$
$$S = \mathcal{J}acc$$
$$S(\{1,2\},\{2,3\}) = \frac{1}{3}$$

A **LSH scheme** over U is a probability distribution over the partitions of U.

Example 5.3.2. We can apply the min-hash scheme [5.1] to the Jaccard Similarity. With $U = [3]$ and $\pi = 1 < 2 < 3$, the function maps each subset of U to a hash as follows:

$$\emptyset \mapsto \perp \text{ (the hash of the empy set is a special symbol)}$$
$$\{1\},\{2,1\},\{1,3\},\{1,2,3\} \mapsto 1 \text{ (all sets containing 1 have hash 1)}$$
$$\{2\},\{2,3\} \mapsto 2 \text{ (all remaining sets containing 2 have hash 2)}$$
$$\{3\} \mapsto 3 \text{ (all remaining sets containing 3 have hash 3)}$$

thus defining 4 partitions over U.

Example 5.3.3. Similarly, we can apply the function based on the Hamming Similarity we saw in [5.2] to the same set $U = [3]$ and see we get new partitions. We choose $i = 2$ UAR, the function maps each subset of U to a hash as follows:

$$\{2\},\{2.1\},\{2,3\},\{1,2,3\} \mapsto 1 \text{ (all sets containing } i \text{ have hash 1)}$$
$$\emptyset,\{1\},\{3\},\{1,3\} \mapsto 0 \text{ (all remaining sets have hash 0)}$$

thus defining 2 partitions over U.

Let's make some other considerations about what we have just seen.

Observation 5.3.1. Given a similarity ϕ, a LSH scheme is a family of hash functions H, coupled with a probability distribution D over H such that, chosen a function h from the family H according to D, h satisfies the property $\Pr\{h(a) = h(b)\} = \phi(a,b) \forall a, b \in U$.
In other words, let $S \in U^2 \to [0,1]$ be a similarity, and H be a RV over a family of hash functions over U, then H is a LSH scheme iff $\Pr\{H(a) = H(b)\} = S(a,b) \forall a, bin U$

Observation 5.3.2. Preprocess and hash function (aka a scheme) determine the similarity function (most people attempt to do the reverse)

Observation 5.3.3. In the previous webpage example [5.1], we're not dealing with a single hashing function, but with a family of functions each built with its own word permutation: the scheme distributes over the permutations of the union!

Before going on, let's recap what we have discussed so far:
A LSH scheme for a similarity S is a prob. dist. over U's partitions such that

$$\forall A, B \in U \Rightarrow \Pr\{A \sim_p B\}_p = S(A,B) = \Pr\{h(A) = h(B)\}_h \tag{5.7}$$

where p is a partitioning of U and $\sim_p$ means A and B are in the same partition.

Another possible definition: Given $s \in U^2 \to [0,1]$ a similarity, then we define X to be a LSH scheme for s as the following:

$$X \in \mathcal{R}and(\mathcal{P}(U)) : \forall A, B \in U \implies \mathcal{E}(A \sim_X B) = S(A,B) \tag{5.8}$$

Challenge: Can we find a LSH scheme for an arbitrary S function? NO

Example 5.3.4. Given a universe set $U = \{a,b,c\}$ and a similarity function S s. t. $S \in U^2 \to [0,1] : S(a,b) \mapsto 1, S(b,c) \mapsto 1, S(a,c) \mapsto 0$ we don't have an LSH, since we're violating transitivity: Translating into probabilities and using equality's transitivity, we obtain: $\Pr\{h(a), h(c)\} = 1$, which contradicts the third mapping.

5.4 More distances

Let $A \vartriangle B = (A - B) \cup (B - A)$.

Definition 5.4.1 (Dice similarity).

$$D(A,B) = \frac{|A \cap B|}{|A \cap B| + \frac{1}{2}|A \vartriangle B|} \tag{5.9}$$

Definition 5.4.2 (Anderberg similarity).

$$An(A,B) = \frac{|A \cap B|}{|A \cap B| + 2|A \vartriangle B|} \tag{5.10}$$

Definition 5.4.3 (Generalizing Jaccard, Dice and Anderberg).

$$S_\gamma(A,B) = \frac{|A \cap B|}{|A \cap B| + \gamma|A \vartriangle B|} \tag{5.11}$$

By this third definition we can obtain $\mathcal{J}acc$ with $\gamma = 1$, D with $\gamma = \dfrac{1}{2}$, and An with $\gamma = 2$.

Question 5.4.1. for which values of γ does an LSH exist for S? The next lemma helps us finding an answer.

Lemma 5.4.1 (by Charikar). *If a similarity S admits an LSH, then the function $1 - S$ must satisfy the triangular inequality, i.e. $\forall A, B, C \in U$*
$d(A,B) \leq d(A,C) + d(B,C)$, where the distance $d(A,B) = 1 - S(A,B)$ is our function $1 - S$.

Proof. Let E_{XY} be the event "$h(X) \neq h(Y)$"; since S admits a LSH, we have:

$$\begin{aligned} d(A,B) = 1 - S(A,B) &= \Pr_h\{E_{AB}\} = p_1 + p_2 + p_3 + p_4 \\ d(A,C) = 1 - S(A,C) &= \Pr_h\{E_{AC}\} = p_1 + p_3 + p_5 + p_7 \\ d(B,C) = 1 - S(B,C) &= \Pr_h\{E_{BC}\} = p_1 + p_2 + p_5 + p_6 \end{aligned}$$

with the probabilities p_i defined as in the following table, where an X under the "may exist" column means that the corresponding probability is 0 (it happens because transitivity generates contradictions, see example [5.3.4]):

	E_{AB}	E_{AC}	E_{BC}	may exist
p_1	T	T	T	✓
p_2	T	T	F	✓
p_3	T	F	T	✓
p_4	T	F	F	X
p_5	F	T	T	✓
p_6	F	T	F	X
p_7	F	F	T	X
p_8	F	F	F	✓

Hence we have

$$d(A,C) + d(B,C) = 2p_1 + p_2 + p_3 + 2p_5 \geq p_1 + p_2 + p_3 = d(A,B)$$

and so $1 - S$ doesn't comply with the triangular inequality, QED. □

Observation 5.4.1. Similarities are actually defined in most cases as the inverses of measures, which in turn give (oh so surprisingly!) a notion of distance.

Corollary 5.4.1.1. *By Charikar's lemma, we can prove that Dice's similarity cannot admit a LSH scheme.*

Proof by counterexample. Assume $A = \{1\}, B = \{2\}, C = \{1,2\}$, then use the triangular inequality over the distances:

$D(A,C) = \frac{2}{3},\ D(B,C) = \frac{2}{3},\ D(A,B) = 0$

$d(A,B) = 1 - D(A,B) = 1 > \frac{2}{3} = (1 - D(A,C)) + (1 - D(B,C)) = d(A,C) + d(B,C)$

hence it doesn't comply with the triangular inequality.

(Note that D stands for Dice similarity and d for distance) □

Observation 5.4.2. Parameterizing this counterexample with S_γ, we obtain a bounds for γ: let $A = \{1\},\ B = \{2\},\ C = \{1,2\}$ and $S_\gamma(A,C) = \frac{1}{1+\gamma},\ S_\gamma(B,C) = \frac{1}{1+\gamma},\ S_\gamma(A,B) = 0$, thus

$1 = 1 - S_\gamma(A,B) = d(A,B) > d(A,C) + d(B,C) =$

$= (1 - S_\gamma(A,C)) + (1 - S_\gamma(B,C)) =$

$= 2\left(1 - \frac{1}{1+\gamma}\right) = \frac{2\gamma}{1+\gamma}$

from which we obtain $1 > \frac{2\gamma}{1+\gamma} \Rightarrow \gamma < 1$.

Hence, if $\gamma < 1$, the triangular inequality doesn't hold, so no LSH can exist, as in the case of the Dice similarity.

5.5 Probability generating functions

Intuition: A probability generating function (PGF) is a power series representation of a given probability distribution

Definition 5.5.1. *Given a (discrete) RV X, its **PGF** is the function:*

$$\mathcal{G}en_X(\alpha) = \sum_{x=0}^{\infty} Pr\{X = x\}\, \alpha^x \tag{5.12}$$

(note that all outcomes appear by their probability).

How to get back to pmf: $\Pr\{X = x\} = \dfrac{\mathcal{D}^x(\mathcal{G}en_X(0))}{x!}$

In other terms, a PGF f is a function:

$$f(x) = \sum_{x=0}^{\infty} \left(p_i x^i\right) \tag{5.13}$$

s.t. $p_i \geq 0 \forall i$ and $\sum_{x=0}^{\infty} p_i = 1$.

Theorem 5.5.1. *If a similarity S admits a LSH and a given function f is a PGF, then $f(S)$ admits a LSH.*

$$f(S(A,B)) = (f(S))(A,B) =: T(A,B)$$

Before going into the proof of the theorem we show a consequence of it.

Observation 5.5.1. Applying the theorem to the Jaccard similarity:
Our function is f_γ, with $\gamma > 1$, defined as

$$f_\gamma(x) = \frac{x}{x + \gamma(1-x)} \tag{5.14}$$

In order to proof f_γ is a PGF, we have to demonstrate that the coefficients represent a probability distribution: i.e., they are all positive and they sum to zero.
By applying the Taylor series expansion to [5.14], we get
$f_\gamma(x) = \sum_{x=1}^{\infty} \left(\frac{\left(1 - \frac{1}{\gamma}\right)^i}{\gamma - 1} x^i \right)$; now f_γ is a power series, so all the coefficients are positive.
In order for the sum of the coefficients to be equal to 1, the numerator must be equal to the denominator:
$\sum_{i=1}^{\infty} \left(\frac{\left(1 - \frac{1}{\gamma}\right)^i}{\gamma - 1} \right) = 1 \Rightarrow \sum_{i=1}^{\infty} \left(\left(1 - \frac{1}{\gamma}\right)^i \right) = \gamma - 1.$
Indeed we have:

$$\begin{aligned}
\sum_{i=1}^{\infty} \left(1 - \frac{1}{\gamma}\right)^i &= \left(1 - \frac{1}{\gamma}\right) \sum_{i=0}^{\infty} \left(1 - \frac{1}{\gamma}\right)^i \\
&=^{(*)} \left(1 - \frac{1}{\gamma}\right) \frac{1}{1 - \left(1 - \frac{1}{\gamma}\right)} \\
&= \frac{\gamma - 1}{\gamma} \frac{1}{1/\gamma} = \gamma - 1
\end{aligned}$$

where the step marked with $(*)$ is due to the equality $\sum_{i=0}^{\infty} \alpha^i = \dfrac{1}{1-\alpha}$.
And with this we proved that f_γ is a PGF.

Now we can apply PGF to a similarity S_γ:

$$\begin{aligned}
f_\gamma(S_\gamma(A,B)) &= f_\gamma\left(\frac{|A \cap B|}{|A \cup B|}\right) \\
&= \frac{\frac{|A \cap B|}{|A \cup B|}}{\frac{|A \cap B|}{|A \cup B|} + \gamma \frac{|A \cup B| - |A \cap B|}{|A \cup B|}} \\
&= \frac{|A \cap B|}{|A \cap B| + \gamma |A \triangle B|} = S_\gamma(A,B)
\end{aligned}$$

which in turn is LSH-able

5.6 More about PGF

Recap: Given a universe U, a function $S \in U^2 \to [0,1]$ is said to be a **LSHable similarity** iff exists a prob. distr. over (a family/subset of) the hash functions in U, such that:

$$\forall \{X, Y\} \in \binom{U}{2} \ \Pr_h \{h(X) = h(Y)\} = S(X, Y) \tag{5.15}$$

Theorem 5.6.1. *If a similarity S is LSH-able and f is a PGF, then $f(S)$ is LSHable.*

Equivalent statement:

$$f(S) := T \in U^2 \to [0,1] \ \textit{s.t.} \ \forall \{A, B\} \in \binom{U}{2} \ T(A, B) = f(S(A, B)) \tag{5.16}$$

Lemma 5.6.2 (L1). *The similarity $O \in U^2 \to [0,1]$ s.t. $O(A,B) \mapsto 1$ $\forall \{A, B\} \in \binom{U}{2}$ admits a LSH.*

Proof. Give probability 1 to the constant hash function h: $h(A) = 1 \ \forall A \in U$
$\Pr \{h(A) = h(B)\} = 1 = O(A, B) \ \forall \{A, B\} \in \binom{U}{2}$.

□

Purpose: This will be the base case for theorem proof...

Lemma 5.6.3 (L2). *If S and T similarities over U have a scheme, then $S \cdot T : (S \cdot T)(A, B) = S(A, B) \cdot T(A, B)$ has a scheme.*
I.e. LSHability is preserved upon composition/multiplication.

Proof by construction (Algorithm).

- Sample hash functions for $S \cdot T$ as follows:
  1. first, sample h_S from the LSH for S;
  2. then, sample h_T independently from the LSH for T;
- Return the hash function h s.t. $H(A) = (h_S(A), h_T(A)) \ \forall A \in U$;
- Thus, $\forall \{A, B\} \in \binom{U}{2}$, we have:

  $$\begin{aligned} \Pr_h \{h(A) = h(B)\} &= \Pr_{h_S} \{h_S(A) = h_S(B)\} \cdot \Pr_{h_T} \{h_T(A) = h_T(B)\} \\ &= S(A, B) \cdot T(A, B) \end{aligned}$$

  by independency.

□

Lemma 5.6.4 (L3). *If S is LSHable, then S^i is LSHable, $\forall i \in \mathbb{N}$.*

Proof by induction.

- Base: $i = 0$ and $S^0 = O$
  True for L1 [5.6.2];
- Inductive hypothesis: S^i is LSHable;

- Inductive step: S^{i+1} is LSHable
  True because $S^{i+1} = S \cdot S^i$ is LSHable by L2 [5.6.3], since S has a scheme by def and S^i has a scheme by inductive hypothesis.

□

Lemma 5.6.5 (L4). *If* $p_0, p_1, ..., p_i, ...$ *is s.t.* $\sum_{i=0}^{\infty} p_i = 1$, $p_i \geq 0 \ \forall i$, *and* $S_0, S_1, ..., S_i, ...$ *are lshable similarities, then* $\sum_{i=0}^{\infty} p_i S_i$ *is lshable.*

Proof by scheme.

1. First, sample $i*$ at random from $\mathbb{N}$ with probability $p_0, ..., p_i, ...$;
2. Then, sample h from the hash functions (LSH) of S_{i*} (that is, we are choosing which LSH to use);
3. Thus we have:

$$\begin{aligned}\Pr_h\{h(A) = h(B)\} &= \sum_{i=0}^{\infty}(p_i S_i(A,B)) \\ &= \sum_{i=0}^{\infty}(\underbrace{\Pr\{(\} i = i*)}_{p_i} \cdot \underbrace{\Pr_h\{h(A) = h(B) | i = i*)\}}_{S_i(A,B)}\end{aligned}$$

□

Observation 5.6.1. This lemma is useful if we need a weighted average
$W(A,B) = \sum_{i=0}^{\infty}(p_i S_i(A,B))$.

Proof of the theorem 5.6.1.

- We want to prove that $\sum_{i=0}^{\infty} p_i S^i$ has a scheme (is LSHable);
- By L3 [5.6.4] we know S^i has a scheme;
- By L4 [5.6.5] we know the sum is lshable;

□

Observation 5.6.2. This theorem tells us how to build a LSH: by concatenating the results of different hash functions, keeping the output small.
I.e. PGF is an approach for making schemes for similarities from other schemes.

Example 5.6.1. $\sum_{i=1}^{a}(2^{-i} \cdot x^i)$.

5.7 A mention of the sketches

Like LSH, they're a means for simplify the storage of data; they are slower than LSH but allow you to keep interesting information in a small space.

Definition 5.7.1 (Sketch). *A sketch is a representation of big objects with small images. They can be used to retrieve interesting information about original objects without regain the whole original objects (i.e. they are not compression algorithms).*

Example 5.7.1. It's possible to go back to $|A \cap B|$ from only $|A|, |B|, h(A), h(B)$, $\mathcal{J}acc(A,B) \pm \varepsilon$. Note that $\mathcal{J}acc(A,B) \pm \varepsilon$ can be obtained from $h(A), h(B)$ by applying the Chernoff Bounf [2.1] and the other data need only few bits to be stored.

Example 5.7.2. It's possible to approximate $\dfrac{|A \cap B|}{|A \cap B| + \frac{1}{2}|A \triangle B|}$ with $\dfrac{1}{2} \cdot |A \cup B| + \dfrac{1}{2} \cdot |A \cap B|$.

5.8 Approximation of non-LSHable functions

What can we do if there is no LSH for the similarity we want/need to use? We can approximate a function without a LSH with an LSHable one.

Example 5.8.1. We can approximate S_γ with $\mathcal{J}acc$ with an error of $\dfrac{1}{\gamma}$.

Theorem 5.8.1. *Let f be a PGF, $\alpha \in [0,1]$.*

$\alpha f = \alpha f(S) \mid (1-\alpha)T$

$T \in U^2 \to [0,1] : \forall t, t' \in \binom{U}{2}\ T(t,t') = 0$ *but* $T(t,t) = 1$

$h(t) = -t$

For the 1 case we wanted a banal partition, now with 0 we want a punctual partition, so we need a hash function that assigns a distinct value to each argument. (You can actually use the identity)
Not a good scheme, because we're not shrinking data.

Definition 5.8.1 (Distortion). *Let $S : U^2 \to [0,1]$ be a similarity, then its* ***distortion*** *is the minimum* $\delta \geq 1$ s.t. $\exists$ an LSHable similarity S' s.t. $\forall A, B \in \binom{U}{2}\ \dfrac{1}{\delta} \cdot S(A,B) \leq S'(A,B) \leq S(A,B)$, where minimum* is the lower bound (if U isn't finite, there is no actual minimum).*

Observation 5.8.1. S' is the LSHable similarity closest to S; the more δ is near to 1, the closer S' is to S (i.e. δ is the *approximation factor*); if δ tends to 1, then S is LSHable.

Example 5.8.2. In the example 5.8.1, where we used $\mathcal{J}acc$ to approximate S_γ, we had $\delta \to \dfrac{1}{\gamma}$.

Question 5.8.1. : How can I know if there is a better approximation than the one I found?

Lemma 5.8.2 (Center lemma). *Let S be a LSHable similarity s.t. $\exists \mathcal{X} \subseteq U : \forall \{x, x'\} \in \binom{\mathcal{X}}{2}\ S(x,x') = 0$, then $\forall y \in U\ avg_{x \in \mathcal{X}}(S(X,Y)) \leq \dfrac{1}{|\mathcal{X}|}$.*

Observation 5.8.2. $\mathcal{X}$ is a set of the most different objects in U and y is a center in U; so the meaning of the lemma is that the average of the distances between each point and the center is $\leq \dfrac{1}{|\mathcal{X}|}$.

Observation 5.8.3. $\mathcal{X}$ is actually a (possibly incomplete) section of the partition of U induced by h.

Observation 5.8.4. (important) From the lemma, it trivially follows that $\exists\ x^* \in \mathcal{X} : S\left(x^*, Y \leq \dfrac{1}{|\mathcal{X}|}\right)$.

Proof of the Center lemma. [ref. 5.8.2]

- Fix $y \in U$;
- If the hash function h has positive probability in the (chosen) LSH for S, then $\forall x, x' \in \binom{\mathcal{X}}{2}$ $h(x) \neq h(x')$ (otherwise we would have $S(x, x') > 0$);
- Thus, $\forall$ hash functions with positive probability, there can exist at most one $x \in \mathcal{X}$ s.t. $h(x) = h(y)$ (by transitivity of equality);
- Than we have:

$$\begin{aligned}\sum_{x\in\mathcal{X}} S(x,y) &= \sum_{x\in\mathcal{X}} \Pr_h\{h(x) = h(y)\} \\ &= \sum_{x\in\mathcal{X}}\sum_{h} \Pr\{h \text{ is chosen}\} \cdot \underbrace{[h(x) = h(y)]}_{\text{pr. that } x=y \text{ with the choosen } h} \\ &= \sum_{h} \Pr\{h \text{ is chosen}\} \cdot \sum_{x\in\mathcal{X}} [h(x) = h(y)] \\ &\leq \sum_{h} \Pr\{h \text{ is chosen}\} = 1\end{aligned}$$

  (Note that he square brackets here are a boolean evaluation operator);
- Thus, $\sum_{x\in\mathcal{X}} S(x, y) \leq 1$, that implies that $avg(S(X, Y)) \leq \frac{1}{|\mathcal{X}|}$, and the lemma is proven.

□

Example 5.8.3. We will now prove what we said about S_γ's approximation in examples [5.8.1] and [5.8.2], using the lemma we have just demonstrated [5.8.2]:

- Let us give some definitions:
  - $S := S_\gamma$, with $0 < \gamma < 1$,
  - $U := 2^{[n]} = \{S | S \subseteq [n]\}$,
  - $\mathcal{X} := \mathcal{P}_1([n]) = \{\{1\}, \{2\}, ..., \{n\}\}$;
- So, by definition of S_γ, we have $S_\gamma(\{i\}, \{j\}) = 0$ $\forall i \neq j \in [n]$, since $\{i\} \cap \{j\} = \emptyset$;
- Let us assume that T (that will be our S') finitely distorts S_γ, and T is LSHable;
- Then $T(\{i\}, \{j\}) = 0$ $\forall \{i, j\} in \binom{[n]}{2}$, since $S' \leq S_\gamma = 0$;

- Since T is LSHable we can apply the center lemma:

  $\exists\{i\} \in \mathcal{X}$ s.t. $T(\underbrace{\{i\}}_{\text{our } x^*}, \underbrace{[n]}_{\text{our } y}) \leq \frac{1}{|\mathcal{X}|} = \frac{1}{n}$

  $S_\gamma(\{i\}, [n]) = \frac{1}{1 + \gamma \cdot (n-1)} = \frac{1}{\gamma \cdot n + (1-\gamma)}$

  so $\exists i \in [n]$ s.t. $S_\gamma(\{i\}, [n]) \geq \frac{1}{\gamma \cdot n + (1-\gamma)}$

  but $T(\{i\}, [n]) \leq \frac{1}{n}$

  thus $\frac{1}{\delta} \frac{1}{\gamma \cdot n + (1-\gamma)} = \frac{1}{\delta} S_\gamma(\{i\}, [n]) \leq T(\{i\}, [n]) \leq \frac{1}{n}$

  hence $\frac{1}{\gamma + \frac{1-\gamma}{n}} = \frac{n}{\gamma \cdot n + (1-\gamma)} \leq \delta$

  $\liminf_{n \to \infty} \delta \geq \frac{1}{\gamma}$ QED

  i.e. the more n grows, the more δ approaches $\frac{1}{\gamma}$;

- Now we know that the distortion of S_γ is $\frac{1}{\gamma}$.

Example 5.8.4. Now we would like to apply the same method to the **cosine similarity** (a.k.a. *inner product similarity*).

- Let's start again with some definitions:
  - $U := \{\vec{x} \mid \vec{x} \in \mathbb{R}^n_+,\ ||\vec{x}||_2 = 1\}$ (i.e., U is a positive hypersphere with the center in the origin and the radius equal to 1),
  - $C \in U^2 \to [0, 1]$ s.t. $C(\vec{x}, \vec{y}) = \langle \vec{x}, \vec{y} \rangle = \sum_{i=1}^{n} x_i y_i$;
- Now, we want to know if C has an LSH and, if not, what is its distortion;
- Another definition:
  $\mathcal{X} = \{\vec{x}_1, \vec{x}_2, ..., \vec{x}_n\}$ where $x_i(j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$
  i.e. $\vec{x}_i = (0, ..., 0, 1, 0, ..., 0)$ with the only 0 in position i;
- So we have that $||\vec{x}||_2 = \sqrt{1^2 + 0^2(n-1)} = 1$;
- Moreover $C(\vec{x_i}, \vec{x_j}) = 0\ \forall i \neq j$;
- Let $\vec{y} = \left(\underbrace{\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}, ..., \frac{1}{\sqrt{n}}}_{\text{n times}} \right)$, then $||\vec{y}||_2 = \sqrt{\sum_{i=1}^{n} y_i^2} = \sqrt{\sum_{i=1}^{n} \frac{1}{n}} = 1$;
- So, $C(\vec{x_i}, \vec{y}) = \sum_{j=1}^{n} (x_i(j) \cdot y_i(j)) = x_i(i) \cdot y_i(i) = \frac{1}{\sqrt{n}}$;
- Then $\delta \geq \sqrt{n}$, so, unlike before, the distortion is big and grows bigger with n.

Example 5.8.5. **Weighed Jaccard** is a generalization of $\mathcal{J}acc$ for vectors:
$\mathcal{WJ} = \frac{\sum_i \min(x_i, y_i)}{\sum_i \max(x_i, y_i)}$.

Example 5.8.6. **Sim Hash** is an LSH scheme similar to the cosine similarity:

- $\mathcal{CS}(\vec{x}, \vec{y}) = \cos(\theta_{\vec{x},\vec{y}})$;
- $\mathcal{SH}(\vec{x}, \vec{y}) = 1 - \frac{\theta_{\vec{x},\vec{y}}}{\pi}$;
- $\mathcal{SH}$ is high if the angle θ is small;
- $\frac{\theta_{\vec{x},\vec{y}}}{\pi}$ is the probability that an hyperplane divides $\vec{x}$ and $\vec{y}$, where the hyperplane is a threshold between similar and dissimilar elements of the universe, so it creates a partition of the universe in two sets.

Chapter 6

Linear programming for approximation algorithms [1]

In this chapter Linear Programming is introduced as a tool to build approximation algorithms. In particular, we use the *Densest Subgraph* problem (DSG for short) as a practical example, because some practitioners believe that it is a good primitive to find community of people; also, this problem is in P and we will solve it through linear programming.

6.1 Linear Programming [2]

Linear Programming (LP) is arguably the most important technique when it comes to approximation algorithms, and many approximations proved with other methods can be understood as linear programming proof.

Definition 6.1.1 (Primal linear program). *A linear program is a convex program written in order to solve an optimization problem, whose aim is to maximize an* ***objective function*** *with n variables, of the form*

$$\max \; c_1x_1 + c_2x_2 + \ldots + c_nx_n \tag{6.1}$$

with $x_i \in \mathbb{R}_{\geq 0}$, and m constraints of the form

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n & \leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n & \leq b_2 \\ \quad \vdots & \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n & \leq b_m \end{cases} \tag{6.2}$$

Since this is a maximization problem, this is conventionally called a ***primal****.*

Note that, if there were no constraints, the optimization problem would have been trivial.

Furthermore, the assumptions that the x_i are non negative is made for simplicity and without loss of generality. In fact, if we had a negative variable, we could obtain it from positive variables, such as $y = x_1 - x_2$.

A primal LP can be written in matrix form in the following way:

$$\begin{aligned} &\max \; \bar{c}^T\bar{x} \\ &\bar{A}\bar{x} \leq \bar{b} \\ &\bar{x} \geq \bar{0} \end{aligned} \tag{6.3}$$

[1] Part of this chapter is taken from this repo by Cristian Di Pietrantonio.
[2] You can read more about this here.

where $\bar{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \bar{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{mn} \end{pmatrix}, \bar{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}, \bar{c} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ b_n \end{pmatrix}$.

Definition 6.1.2. *(Dual linear program) A dual LP can be represented with the following matrix form:*

$$\begin{aligned} &\min \ \bar{y}^T \bar{b} \\ &\bar{y}^T \bar{A} \geq \bar{c} \\ &\bar{y} \geq \bar{0} \end{aligned} \tag{6.4}$$

Note that it uses the same (transposed) coefficient and constant matrices of the primal, but the variable vector has dimension m instead of n. So, to pass from the primal to the dual, we have to change max *into* min *and to let any constraint become a variable and vice versa.*

Theorem 6.1.1 (Weak Duality). *If $\bar{x}$ is a feasible solution to the primal and $\bar{y}$ is a feasible solution to the dual, then*

$$\bar{c}^T \bar{x} \leq \bar{y}^T \bar{b} \tag{6.5}$$

I.e., opt(primal) ≤ opt(dual).

If we have a primal, then we can guess a solution $\bar{x}$, check that it satisfies the constraints and evaluate how good it is by using the objective function; to know if the solution is optimal or close to optimal, we can write and solve the dual, then we would know the dual's optimum is greater or equal to the primal's. If one guesses a solution for the primal and guesses a solution for the dual, and their costs are close, then that solution is close to the optimal one.

Proof.

$\bar{c}^T \bar{x} \leq (\bar{y}^T \bar{A})\bar{x} = \bar{y}^T (\bar{A}\bar{x}) \leq \bar{y}^T \bar{b}$

where the first inequality holds by definition of dual, and the second by definition of primal. □

Theorem 6.1.2 (Strong Duality). *If $\bar{x}$ is an optimal solution to the primal and $\bar{y}$ is an optimal solution to the dual, and if $\bar{c}^T$ and $\bar{y}\bar{b}$ are finite, then*

$$\bar{c}^T \bar{x} = \bar{y}\bar{b} \tag{6.6}$$

I.e., opt(primal) = opt(dual).

The strong duality tells us that if both primal and dual admit solution they have the same value for the optimum solution.

Observation 6.1.1. It is a common approach to transform a combinatorial problem that one is faced with into a linear one, then solve the linear one (that requires polynomial time), and then transform the solution to the linear problem in a solution to the original problem.

6.2 Densest subgraph

6.2.1 A linear program to solve the densest subgraph problem

We are interested in this problem since it allows to find communities into social networks.

Definition 6.2.1 (Induced subgraph). *If $G(V, E)$ is a graph and $S \subset G$, then $G[S]$ (read "G induced on S") is a graph with node set S and edge set $\{\{u, v\} \mid \{u, v\} \in E \wedge u, v \in S\}$.*

Definition 6.2.2 (Densest subgraph). *The densest subgraph of $G(V, E)$ is one induced subgraph $G[S]$ with the largest average degree, i.e., with S such that the density of the graph $f(S) = \frac{|E(S)|}{|S|}$ is maximized.*

Observation 6.2.1. Note that, by maximizing the density of the graph, we maximize the average degree of the graph $\frac{2|E(S)|}{|S|}$.

Observation 6.2.2. This is a particular approximation problem, since it can actually be solved exactly in polynomial time.

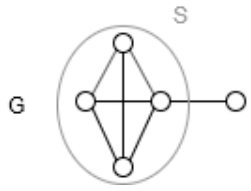


Figure 6.1: An example of densest subgraph $G[S]$.

Now we want to express the densest subgraph problem as a (primal) linear problem (we will explain it in detail immediately after the statement):

$$\begin{aligned} &\max \sum_{\{i,j\} \in E(G)} X_{\{i,j\}} \\ &\begin{cases} X_{\{i,j\}} \leq Y_i & \forall \{i,j\} \in E(G) \\ X_{\{i,j\}} \leq Y_j & \forall \{i,j\} \in E(G) \\ \sum_{i=1}^{n} Y_i \leq 1 \end{cases} \\ &X_{\{i,j\}}, Y_i \geq 0 \quad \forall \{i,j\} \in E(G) \end{aligned} \tag{6.7}$$

Starting from the objective function, we want something that says "get as many edges as possible in the final graph"; so, it seems natural to maximizes the number of edges we can put in the final subset of nodes. Observe, though, that we cannot maximize $f(S)$ since it is not linear. One can take another approach and look only at the numerator. We define a variable $X_{\{i,j\}}$ to be the fractional amount of edge $\{i, j\}$ that is put inside the solution. The objective function is

$$\max \sum_{\{i,j\} \in E} X_{\{i,j\}}. \tag{6.8}$$

For each edge, we cannot pick that edge fractionally more than the amount with which we pick one of its endpoints. For example, if we pick $\{i, j\}$ for $\frac{1}{2}$ of its entirely, then it is the case that we had to pick both i and j for at least $\frac{1}{2}$. This can be expressed introducing a variable Y_i for each node i, representing the fractional amount with which we pick node i; then, we can say that

$$X_{\{i,j\}} \leq Y_i, \tag{6.9}$$
$$X_{\{i,j\}} \leq Y_j. \tag{6.10}$$

Up until now the LP is unbounded. We now need to model the denominator by saying that the total amount with which nodes of the graph are picked must be limited; we can think of that amount being, say, 1 to normalize it. Making the denominator a constant allows us to linearize the objective function.

$$\sum_{i=1}^{n} Y_i \leq 1. \tag{6.11}$$

Finally, the variables must be non-negative:

$$X_{\{i,j\}}, Y_j \geq 0 \tag{6.12}$$

What we will be proving is that the optimal solution OPT_{LP} of this LP is equal to the optimal solution of the original densest subgraph problem $f(S^*)$, where S^* is the densest subgraph of G. To do so we need the following steps:

1. There exists a feasible solution to the LP having value $f(S)$;
2. $OPT_{LP} \geq f(S^*)$;
3. $f(S^*) \geq OPT_{LP}$.

Lemma 6.2.1. *For any graph $G(V, E)$ and for any $\emptyset \neq S \subset V$, there exists a feasible solution to the linear problem having value $f(S) = \dfrac{|E(S)|}{|S|}$ (the density of the graph).*

Proof. We will give a solution and show that it meets the requirements.

Let $Y_i := \begin{cases} \dfrac{1}{|S|} & i \in S, \\ 0 & i \notin S \end{cases}$.
Remember that the Y_is were the variables representing how much of each node in the graph we are taking, this definition states that we split the unit among the nodes in the set S.

Then let $X_{\{i,j\}} = \begin{cases} \dfrac{1}{|S|} & \{i,j\} \subseteq S, \\ 0 & \{i,j\} \not\subseteq S \end{cases}$.

To show this solution is feasible we check that every constraint is satisfied.
Let's start with Inequality 6.11: $\sum_i Y_i = |S|\dfrac{1}{|S|} = 1$, so far, so good.

Let's proceed with Inequality 6.9, starting with a generic $\{i,j\} \in E$. We have two cases:

- $\{i,j\} \not\subseteq S$: In this case we want to guarantee that

$$\begin{cases} X_{\{i,j\}} \leq Y_i \\ X_{\{i,j\}} \leq Y_j \end{cases} \implies X_{\{i,j\}} \leq \min\{Y_i, Y_j\}, \tag{6.13}$$

  but both sides of the two equations are zero, since $\{i,j\} \not\subseteq S$.

- $\{i,j\} \subseteq S$: Then

$$\frac{1}{|S|} = X_{\{i,j\}} \leq \min\{Y_i, Y_j\} = \frac{1}{|S|}. \tag{6.14}$$

By that, we demonstrated that the solution is feasible. At this point it's easy to compute its value:

$$\sum_{\{i,j\} \in E(G)} X_{\{i,j\}} = |E(S)|\frac{1}{|S|} = f(S). \tag{6.15}$$

□

Corollary 6.2.1.1. *Let OPT_{LP} be the value of the optimal feasible solution to the LP, and let S^* be the densest subgraph of G, then*

$$OPT_{LP} \geq OPT = f(S^*) = \frac{|E(S^*)|}{|S^*|}. \tag{6.16}$$

What we have shown is that the LP's optimal value is never worst than the original densest subgraph problem's optimum.

Lemma 6.2.2. *Let* $\{X_{\{i,j\}}, Y_i\}$ *be an optimal solution to the Linear Program 6.7, having value* v*, then* $\exists S \subseteq V \ f(S) \geq v$.

Proof by Charikar. To prove this lemma, we will proceed step by step claiming some properties that will help us to reach our aim.

Claim 6.2.2.1. *The following property holds:*

$$\forall \{i,j\} \in E \ X_{\{i,j\}} = \min\{Y_i, Y_j\}. \tag{6.17}$$

Proof of claim 6.2.2.1. Suppose by contradiction that this was not the case, then $\exists \ X_{\{i,j\}} < \min\{Y_i, Y_j\}$. But now we can increase the value of $X_{\{i,j\}}$ up to the minimum and so the objective function's value increases and the solution remains feasible. It follows that $X_{\{i,j\}}$ wasn't optimal value (contradiction). □

Now we proceed with the Lemma's proof. Let

$$S(r) := \{i \mid i \in V(G) \wedge Y_i \geq r\}, \tag{6.18}$$
$$E(r) := \{\{i,j\} \mid \{i,j\} \in E(G) \wedge X_{\{i,j\}} \geq r\}. \tag{6.19}$$

We are defining a parametric set, that contains all the nodes i such that $Y_i \geq r$ in the optimal solution. $S(0)$ contains all the nodes but, as r increases, the number of nodes included becomes smaller and smaller.

Claim 6.2.2.2. *No matter which* r *we pick, it will select a set of nodes and also all the edges induced by that set of nodes:*

$$\forall r, \ 0 \leq r \leq \max\{Y_i\}, \ \{i,j\} \in E(r) \iff \{i,j\} \subseteq S(r). \tag{6.20}$$

Proof of claim 6.2.2.2.

$$\begin{aligned}
\{i,j\} \in E(r) &\implies X_{\{i,j\}} \geq r && \text{(by [6.19])} \\
&\implies \min\{Y_i, Y_j\} \geq r && \text{(by claim [6.2.2.1])} \\
&\implies Y_i \geq r \wedge Y_j \geq r \\
&\implies \{i,j\} \subseteq S(r) && \text{(by [6.18])}
\end{aligned}$$

Now the other direction.

$$\begin{aligned}
\{i,j\} \subseteq S(r) &\implies r \leq X_{\{i,j\}} = \min\{Y_i, Y_j\} && \text{(by claim [6.2.2.1])} \\
&\implies \{i,j\} \in E(r). && \text{(by [6.19])}
\end{aligned}$$

□

Now we introduce an ordering of the Ys and two integrals that will be useful later: let π be a permutation such that $0 =: Y_{\pi_{(0)}} \leq Y_{\pi_{(1)}} \leq Y_{\pi_{(2)}} \leq Y_{\pi_{(2)}} \leq \ldots \leq\leq Y_{\pi_{(n)}}$, with $n = |V|$.

Let's look at the following integral which gives a property of the nodes.

Claim 6.2.2.3.

$$\int_0^{\max\{Y_i\}} |S(r)| dr = \sum_{i=1}^{n} Y_i. \tag{6.21}$$

Proof of claim 6.2.2.3. First of all, let's note that the integral in [6.2.2.3] can be graphically represented as in the picture [6.2], where each gray rectangle is the integral for a certain value of r, i.e., an addend of the sum that makes up the integral (see ($*^1$)).

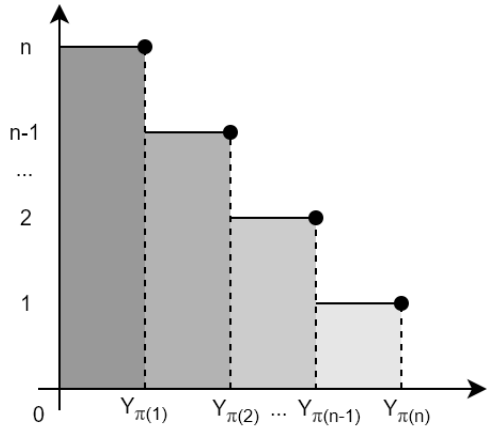


Figure 6.2: A representation of the integral $\int_0^{\max\{Y_i\}} |S(r)|dr$.

$$\begin{aligned}
\int_0^{\max\{Y_i\}} |S(r)|dr &= \sum_{i=1}^{n} \underbrace{\left((n-1+1)\cdot\left(Y_{\pi_{(i)}} - Y_{\pi_{(i-1)}}\right)\right)}_{(*^1)} \\
&= \sum_{i=1}^{n} \left((n-1+1)\cdot Y_{\pi_{(i)}}\right) + \sum_{i=1}^{n} \left((n-1+1)\cdot Y_{\pi_{(i-1)}}\right) \\
&= \sum_{i=1}^{n} \left((n-1+1)\cdot Y_{\pi_{(i)}}\right) + \sum_{i=0}^{n-1} \left((n-1)\cdot Y_{\pi_{(i)}}\right) \qquad (*^2) \\
&= \sum_{i=1}^{n} \left((n-1+1)\cdot Y_{\pi_{(i)}}\right) + \sum_{i=1}^{n} \left((n-1)\cdot Y_{\pi_{(i)}}\right) \qquad (*^3) \\
&= \sum_{i=1}^{n} Y_{\pi_{(i)}} = \sum_{i=1}^{n} Y_i
\end{aligned}$$

The step marked by $(*^2)$ is due to the fact that we can remove the last term of the sum by just working on the indices.
The step marked by $(*^3)$ is due to the fact that for $i = 0$ and for $i = n$ the value of the term is 0, so we can sum from 1 to n as in the first sum, to obtain compatible addends. □

Let's do the same for the edges.

Claim 6.2.2.4.

$$\int_0^{\max\{Y_i\}} |E(r)|dr = \sum_{i=1}^{n} X_{\{i,j\}}. \tag{6.22}$$

Proof of claim 6.2.2.4. The proof is analogous to the one for the claim [6.2.2.3]. □

Claim 6.2.2.5.

$$\exists r \ \in [0, \max\{Y_i\}] \ s.t. \ |E(r)| \geq v|S(r)|. \tag{6.23}$$

Making this claim actually means saying that

$$f(S(r)) = \frac{|E(r)|}{|S(r)|} \geq v = \sum_{\{i,j\}\in E(G)} X_{\{i,j\}}, \tag{6.24}$$

thus, for at least one value r, we have a DSG value that is at least v, what we wanted to prove.

Proof by contradiction of claim 6.2.2.5.

- Assume that, $\forall\ r \in [0, \max\{Y_i\}]$, it holds that $|E(r)| < v|S(r)|$;
- Then, $\int_0^{\max\{Y_i\}} |E(r)|dr < v \int_0^{\max\{Y_i\}} |S(r)|dr$;
- By claims [6.2.2.3] and [6.2.2.4], it follows that $\sum_{\{i,j\}\in E} X_{\{i,j\}} < v \sum_{i=1}^{n} Y_i$;
- But, by optimality of the LP solution, it is true that $\sum_{i=1}^{n} Y_i = 1 \ \wedge \ \sum_{\{i,j\}\in E} X_{\{i,j\}} = v$;
- It follows a contradiction, $v < v$.

Thus, our assumption was false, and our claim was true. □

At this point we have proved Lemma 6.2.2 and, more importantly, that the LP we gave finds the exact optimal solution for the Densest Subgraph problem. □

As a side note, in this proof we used integrals to avoid to sum over all the possible values of the Y_is. How can we actually find the set S given the LP's optimal value v? How many candidate sets there can be? In principle infinitely many, but the only ones that matter are given by $r \in \{Y_i : 1 \leq i \leq n\}$ (because of how $S(r)$ is defined). So we try all the n possible values of r and pick the best one.

Let's note that solving this Linear Problem requires polynomial time, that is pretty good in general, but it is effectively impossible to use for graphs with more than 10000 nodes, such as the social graphs, so we will look for linear and sublinear approximations.

6.2.2 A greedy algorithm to solve the densest subgraph problem

Here, we explore a greedy algorithm to approximate the optimal solution in linear time. Recall that $f(S)$ is the function defined in [6.2.2].

```
Greedy G(V,E):
    S_0 ← V
    for i = 1,...,n-1:
        let v_{i-1} ∈ S be a node of minimum degree in G[S_{i-1}]
        S_i ← S_{i-1} - {v_{i-1}}
    return the ``best'' S_i (in terms of its value f(S_i))
```

Listing 6.1: The Greedy algorithm to solve the densest subgraph problem

Observation 6.2.3. This algorithm runs in $O(|V| + |E|)$ time, i.e., in linear time.

Theorem 6.2.3. *The solution returned by Algorithm [6.1] is a 2-approximation of the optimal solution.*

Observation 6.2.4. We will see that Greedy can be analyzed as an implicit LP. We introduce here a picture that will be more clear later, but is useful since now to show that the optimal and the greedy solution are sandwiched between the solutions of the dual and the primal problems underlying this algorithm: see figure [6.3].

Proof. As it often happens in primal-dual proofs, even in this case it will be useful to refer to an underlying algebraic structure, for this proof it will be the *orientation.*

Definition 6.2.3 (Orientation). *An orientation φ of the simple undirected graph $G(V, E)$ is an assignmento f each edge in E to one of its endpoints.*

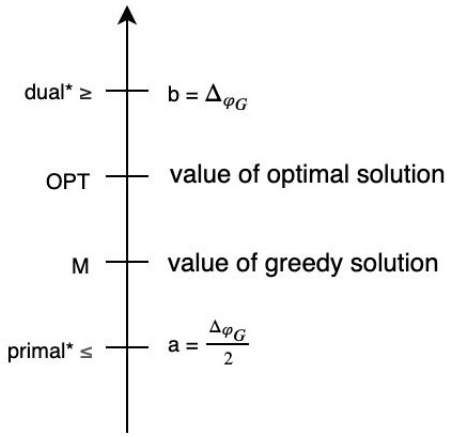


Figure 6.3: A plot of the optimal and the greedy solution sandwiched between the dual's and the primal's.

Observation 6.2.5. There are $2^{|E|}$ possible orientations φ.

Let's introduce some other definitions that will be useful in the following proof.

Definition 6.2.4 (φ-degree). *Given φ and $v \in V$, let $d_\varphi(v) = |\{e \mid v \in e,\ e$ oriented towards v by $\varphi\}|$.*

Definition 6.2.5 (Maximum φ-degree). *Given φ and $v \in V$, let $\Delta_\varphi = \max_{v \in V}\{d_\varphi(v)\}$.*

What we want to do now is to bound the quality of the optimal solution in terms of some property of all orientations of the graph G. The reason is that the algorithm will be analyzed by making it implicitly select an orientation during its execution. Since we are about to show we can bound the value of the optimal solution to DSG by some function of any orientation, this will allow us to give a bound on the quality of the algorithm.

Lemma 6.2.4.

$$\forall\ \varphi \max_{\emptyset \subset S \subseteq V}\{f(S)\} \leq \Delta_\varphi. \tag{6.25}$$

Proof.

$$\begin{aligned}
|E(S)| &\leq \sum_{v \in S} d_\varphi(v) && (*)\\
&\leq \sum_{v \in S} \Delta_\varphi(v) && \text{(we upperbound each } d_\varphi(v) \text{ with } \Delta_\varphi(v))\\
&= |S| \cdot \Delta_\varphi(v)\\
&\Downarrow && \text{(by dividing by } |S|)\\
\frac{|E(S)|}{|S|} &\leq \Delta_\varphi
\end{aligned}$$

The reason of the step marked by $*$ is the following: Pick $e = \{v, w\} \in E(S)$, the orientation φ will orient the edge e towards v or w, that is, towards some node of S, and therefore it will be counted in the sum; Moreover, if we sum up the φ-degrees, we will be also counting edges that come from nodes outside S to nodes in S.

The fact that $f(S) = \frac{|E(S)|}{|S|}$ concludes the proof. □

This prove the upper bound part of Theorem [6.2.3], that is valid $\forall\ \varphi$; now we want to find a lower bound that is close to the optimal solution of the dual LP underlying Greedy [6.1] (b in figure [6.3]), and that holds for Greedy as well as for the optimal solution given by the LP [6.7].

This bound will be valid only for the φ implicitly built by Greedy, that we will call φ_G, defined as follows:

- The orientation is created as the algorithm progresses;
- At the beginning no edge is directed towards anything;
- When Greedy remove a node w_i, all the edges incident in w_i and that are still in the graph will be oriented towards w_i.

An example of φ_G is given in figure [6.4].

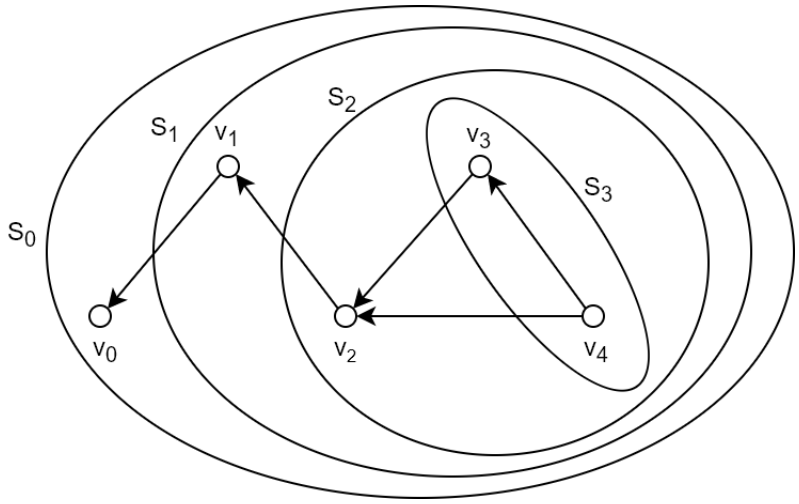


Figure 6.4: Example of computation of φ_G

Note that the algorithm doesn't care about orientations, but we will use this concept in the proof.

Lemma 6.2.5. *Let M be the solution returned by Greedy, i.e., $M := \max_{i=0,\dots,n-1}\{f(S_i)\}$, then the following inequality holds:*

$$\Delta_{\varphi_G} \leq 2M \tag{6.26}$$

Proof. Pick any S_i, let v_i be a node of minimum degree in $G[S_i]$:

$$\begin{aligned} d_\varphi(v_i) &= \deg_{S_i}(v_i) && \text{(if I remove } v_i\text{, then } d\varphi(v_i) = d_{S_i}(v_i)\text{)} \\ &= \min_{v\in S_i} \deg_{S_i}(v) \\ &\leq \operatorname*{avg}_{v\in S_i} \deg_{S_i}(v) \\ &= \frac{1}{|S_i|}\sum_{v\in S_i} \deg_{S_i}(v) \\ &= 2\frac{|E(S_i)|}{|S_i|} = 2f(S_i) \leq 2M. \end{aligned}$$

□

Lemma 6.2.6.

$$\max_{\emptyset\subset S\subseteq V} f(S) \leq 2M \tag{6.27}$$

Proof. Apply lemma [6.2.4] with $\varphi = \varphi_G$ together with lemma [6.2.5]. □

This finally concludes the proof of Theorem [6.2.3]. □

Observation 6.2.6. To obtain the actual approximated densest subgraph from Greedy, it isn't necessary to store each one of the S_is, it is sufficient to keep in memory the degree of the node v_i we deleted. Furthermore, by returning the obtained community and $\deg(v_i)$, we have a proof of the goodness of that community, since the degree gives a bound for the optimal community.

Now we are going to show the *implicit linear problem* underlying Greedy.
The **primal** LP is the same we saw in the previous section, i.e. [6.7], we just rename some variables and give a name to the constraints (in square brackets):

$$\begin{aligned}
&\max \sum_{\{i,j\}\in E(G)} X_{\{i,j\}} \\
&\begin{cases}
X_{\{i,j\}} - X_i \leq 0 \quad \forall\, \{i,j\} \in E(G) & [Y_{i,j}] \\
X_{\{i,j\}} - X_j \leq 0 \quad \forall\, \{i,j\} \in E(G) & [Y_{j,i}] \\
\sum_{i=1}^{n} X_i \leq 1 & [Y^*]
\end{cases} \\
&X_{\{i,j\}}, X_i \geq 0 \ \ \forall\, \{i,j\} \in E(G)
\end{aligned} \tag{6.28}$$

The **dual** LP is the following:

$$\begin{aligned}
&\min Y^* \\
&\begin{cases}
Y_{i,j} + Y_{j,i} \geq 1 & \forall\, \{i,j\} \in E(G) & [X_{\{i,j\}}] \\
Y^* - \sum_{j \forall \{i,j\} \in E(G)} Y_{i,j} \geq 0 & \forall\, i \in V(G) & [X_i]
\end{cases} \\
&Y_i, Y_j, Y^* \geq 0
\end{aligned} \tag{6.29}$$

Now we give a feasible solution for the primal and a feasible solution for the dual.
Suppose that Greedy outputs the set S, the **primal solution** is the following:

$$\begin{aligned}
X_i &= \begin{cases} \frac{1}{|S|} & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases} \\
X_{i,j} &= \begin{cases} \frac{1}{|S|} & \text{if } i,j \in S \\ 0 & \text{otherwise} \end{cases}
\end{aligned} \tag{6.30}$$

Proof of feasibility. Given in the previous section, see the proof of Lemma [6.2.1]. □

Suppose that Greedy produces φ_G, the **dual solution** is the following:

$$\begin{aligned}
&Y_{i,j} = \begin{cases} 1 & \text{if } \{i,j\} \in E(G) \text{ and is directed towards } i \text{ according to } \varphi_G \\ 0 & \text{otherwise} \end{cases} \\
&Y_{j,i} = 1 - Y_{i,j} \\
&Y^* = \Delta_{\varphi_G}
\end{aligned} \tag{6.31}$$

Proof of feasibility. The first constraint is satisfied since each edge $\{i,j\}$ is always oriented by φ_G either towards i or towards j, so $Y_{i,j} + Y_{j,i} = 1$.
The second constraint is satisfied because $Y^* - \sum_{j \forall \{i,j\} \in E(G)} Y_{i,j} = Y^* - d_{\varphi_G} \geq 0$, since the optimal solution of the dual is $\Delta_{\varphi_G} = Y^* \geq d_{\varphi_G}$. □

By this we shown that the analysis of Greedy can be seen as a primal-dual proof, i.e., the solution of Greedy is sandwiched between the primal solution and the dual solution (in this particular case, the solution of Greedy is exactly the same as the solution of primal). Furthermore, this concludes the explanation of the figure [6.3].

Example 6.2.1. Now we are going to apply the primal-dual approach to a simple yet useful example of LP underlying Greedy.

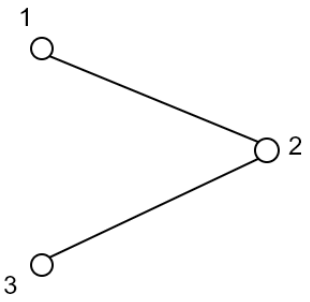


Figure 6.5: Example of primal-dual approach to Densest Subgraph problem.

First of all, we apply the primal LP [6.28] to the graph in the picture [6.5] and we obtain

$$\max X_{\{1,2\}} + X_{\{1,2\}} \; (+0X_1 + 0X_2 + 0X_3)$$
$$\begin{cases} X_{\{1,2\}} - X_1 & \leq 0 \\ X_{\{1,2\}} - X_2 & \leq 0 \\ X_{\{2,3\}} - X_2 & \leq 0 \\ X_{\{2,3\}} - X_3 & \leq 0 \\ X_1 + X_2 + X_3 & \leq 1 \end{cases}$$
$$X_{\{i,j\}}, X_i \geq 0 \quad \forall \, \{i,j\} \in E(G)$$

From this, we obtain the matrix form described in [6.3], that we represent here, together with the dual matrix presented in [6.4]:

Primal:		Dual:
$\max \; \bar{c}^T \bar{x}$		$\min \; \bar{y}^T \bar{b}$
$\bar{A}\bar{x} \leq \bar{b}$	$\Longrightarrow$	$\bar{y}^T \bar{A} \geq \bar{c}$
$\bar{x} \geq \bar{0}$		$\bar{y} \geq \bar{0}$

In our example we have: $\bar{x} = \begin{pmatrix} X_{\{1,2\}} \\ X_{\{2,3\}} \\ X_1 \\ X_2 \\ X_3 \end{pmatrix}, \bar{A} = \begin{pmatrix} 1 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}, \; \bar{b} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \; \bar{c} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$

Note that matrix $\bar{A}$ has one row for each constraint and one column for each variable.

Now we can compute the dual of our example: for the moment we take a vector $\bar{y}$ with one variable for each constraint in the primal $\bar{y}^T = \begin{pmatrix} Y_1 & Y_2 & Y_3 & Y_4 & Y_5 \end{pmatrix}$, later we will assign more significant names to the variables.

$$\min \; \bar{y}^T \bar{b} \Rightarrow min \begin{pmatrix} Y_1 & Y_2 & Y_3 & Y_4 & Y_5 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = Y_5$$

$$\bar{y}^T \bar{A} \geq \bar{c} \Rightarrow \begin{pmatrix} Y_1 + Y_2 \\ Y_3 + Y_4 \\ -Y_1 + Y_5 \\ -Y_2 - Y_3 + Y_5 \\ -Y_4 + Y_5 \end{pmatrix} \geq \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

From this formulation we can obtain our constraints:

$$\begin{cases} Y_1 + Y_2 & \geq 1 \\ Y_3 + Y_4 & \geq 1 \\ -Y_1 + Y_5 & \geq 0 \\ -Y_2 - Y_3 + Y_5 & \geq 0 \\ -Y_4 + Y_5 & \geq 0 \end{cases}$$

Finally, we can rename the Y_i variables to adapt our result to the LP in [6.29]: $Y_1 \to Y_{1,2},\ Y_2 \to Y_{2,1},\ Y_3 \to Y_{2,3},\ Y_4 \to Y_{3,2},\ Y_5 \to Y^*$; that gives us the following LP:

$$\min Y^*$$
$$\begin{cases} Y_{1,2} + Y_{2,1} & \geq 1 \\ Y_{2,3} + Y_{3,2} & \geq 1 \\ Y^* - Y_{1,2} & \geq 0 \\ Y^* - Y_{2,1} - Y_{2,3} & \geq 0 \\ Y^* - Y_{3,2} & \geq 0 \end{cases}$$

This allows us to see again the connection between those variables and constraint and the original graph.

6.2.3 A sublinear algorithm to solve the densest subgraph problem

We now have a linear algorithm to approximate the DSG solution. But being linear in a graph with billion of nodes is still unfeasible, though. People have tried to improve this algorithm under the assumption that there is a cluster of computers each of which is computing towards finding the best solution.

Observation 6.2.7. In Greedy algorithm [6.1] we don't really need to remove the node with minimum degree, since the only step in which we used the minimum degree value is in the proof of Lemma [6.2.5], but right after we upper bounded it with the average degree. In fact the algorithm could have picked a node with degree less or equal to the average degree and the proof still would have worked.

Before proceeding with the updated algorithm, let's give some counterexamples that show why it is not sufficient to pick and remove nodes with degree $\leq k \cdot \min_{v \in S_i} \deg_{S_i}(v)$.

Example 6.2.2. If we pick at each iteration all the nodes with at most twice the minimum degree, there is a chance that we remove only two nodes per round, for example if our graph is a path.

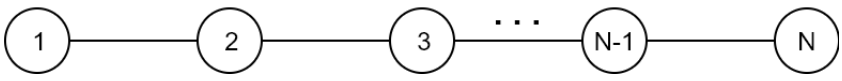


Figure 6.6: Example of path.

Example 6.2.3. If we pick at each iteration all the nodes with at most k times the minimum degree, with any integer $k \geq 2$, we could remove many nodes at each step, but always in constant number, if we have a graph $G(V, E)$ such that:

- The graph has n nodes and $\sqrt{n}$ layers L_i,
- Each layer L_i contains i nodes,
- $V = \bigcup_{i=1}^{k} L_i$,

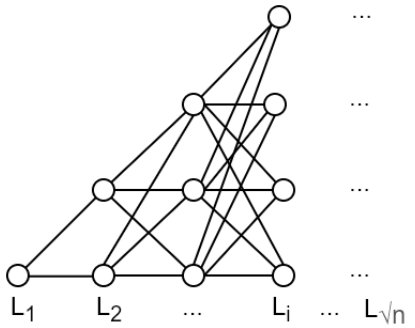


Figure 6.7: Example of graph with more layers.

- $E = \{\{v, w\} \mid \exists\ v \in L_i \wedge w \in L_{i+1}\}$.

Observation 6.2.8. The algorithm we are looking for must have two properties:

1. It requires logarithmic number of operations,
2. Its result is an approximation of the optimal solution by a constant factor.

Finally we present the algorithm DS_ε by Bahmani, Kumar, Vassilvitskii:

```
DS_ε(G(V, E)):
    i ← 0
    S_0 ← V
    while S_i ≠ ∅:
        A_i ← {v | v ∈ S_i ∧ deg_{S_i}(v) ≤ 2 · (1 + ε) · f(S_i) = (1 + ε) · avg_{w∈S_i} deg_{S_i}(w)}
        S_{i+1} ← S_i − A_i
    return arg max_{S_i} f(S_i)
```

Listing 6.2: The DS_ε algorithm to solve the densest subgraph problem

Observation 6.2.9. DS_ε is a parallelization of Greedy in the sense that it removes many nodes all together, i.e., an number of nodes that increases in each iteration.

Lemma 6.2.7. *DS_ε returns a $(2 + 2\varepsilon)$-approximation.*

Proof. Let S^* be an optimal solution to the densest subgraph problem on $G(V, E)$.

Claim 6.2.7.1.

$$\forall v \in S^* \deg_{S^*}(v) \geq f(S^*). \tag{6.32}$$

It is a pretty intuitive claim: If we have a node in the optimal solution that has a degree less than the average degree, we can remove that node to increase the quality of the solution.

Proof.

$$\frac{|E(S^*)|}{|S^*|} = f(S^*) \geq f(S^* - \{v\}) = \frac{|E(S^* - \{v\})|}{|S^*| - \{v\}} = \frac{|E(S^*)| - \deg_{S^*}(v)}{|S^*| - 1} \qquad \text{(by optimality)}$$

$$\Downarrow$$

$$\frac{|E(S^*)|}{|S^*|} \cdot (|S^*| - 1) \geq |E(S^*)| - \deg_{S^*}(v)$$

$$\Downarrow$$

$$|E(S^*)| - \frac{|E(S^*)|}{|S^*|} \geq |E(S^*)| - \deg_{S^*}(v)$$

$$\deg_{S^*}(v) \geq \frac{|E(S^*)|}{|S^*|} = f(S^*)$$

□

Now let's assure that the algorithm terminates. Observe that $A(S_i)$ is always not empty because there will be always some node with degree less than or equal to the average degree. It follows that at every iteration we remove at least one node. We formalize it in the following claim:

Claim 6.2.7.2.

$$|A_i| \geq 1, \ if |S_i| \geq 1. \tag{6.33}$$

Proof.

$$\sum_{v \in S_i} \deg_{S_i}(v) = f \cdot f(S_i) \cdot |S_i|$$

$$\min_{v \in S_i} \deg_{S_i}(v) \leq \operatorname*{avg}_{v \in S_i} \deg_{S_i}(v) = 2 \cdot f(S_i)$$

□

Now we can go back to the proof of Lemma [6.2.7]. So, at this point, we want to show that there exists one iteration where the quality of the solution in that iteration is a good approximation to the optimal quality.

Fix an optimal solution S^*. Let i be the first iteration such that DS_ε removes some element of S^* from the graph. Notice that there must be such i. Formally,

$$A_i \cap S^* \neq \emptyset \ \wedge \ \forall \ j < i \ Aj \cap S^* = \emptyset.$$

Let $v \in A_i \cap S^*$. Observe that $S^* \subseteq S_i$, since S_i is just S^* where we removed at least one node. Then the following holds:

$$\begin{aligned}
f(S^*) &\leq \deg_{S^*}(v) && \text{(by Claim [6.2.7.1])} \\
&\leq \deg_{S_i}(v) && \text{(since } S^* \subseteq S_i\text{)} \\
&\leq 2 \cdot (1 + \varepsilon) \cdot f(S_i) && \text{(by greedy choice made by [6.2])} \\
&\Downarrow \\
f(S_i) &\geq \frac{1}{2(1 + \varepsilon)}
\end{aligned}$$

And with this we have proven the desired approximation. □

Lemma 6.2.8. *Let $n = |V|$, DS_ε terminates after $O\left(\frac{\log n}{\varepsilon}\right)$.*

Proof. Fix an iteration i, the total degree is:

$$\begin{aligned}
2|E(S_i)| &= \sum_{v \in S_i} \deg_{S_i}(v) \\
&= \sum_{v \in A_i} \deg_{S_i}(v) + \sum_{v \in S_i - A_i} \deg_{S_i}(v) \\
&\geq \sum_{v \in A_i} 0 + \sum_{v \in S_i - A_i} (2 \cdot (1+\varepsilon) \cdot f(S_i)) && \text{(by greedy choice made by [6.2])} \\
&= |S_i - A_i| \cdot 2 \cdot (1+\varepsilon) \cdot f(S_i) \\
&= (|S_i| - |A_i|) \cdot 2 \cdot (1+\varepsilon) \cdot f(S_i) && \text{(because } A_i \subseteq S_i\text{)} \\
&= (|S_i| - |A_i|) \cdot 2 \cdot (1+\varepsilon) \cdot \frac{|E(S_i)|}{|S_i|}
\end{aligned}$$

It follows that:

$$\begin{aligned}
&2|E(S_i)| \geq (|S_i| - |A_i|) \cdot 2 \cdot (1+\varepsilon) \frac{|E(S_i)|}{|S_i|} \\
&\implies 1 \geq (|S_i| - |A_i|) \cdot (1+\varepsilon) \frac{1}{|S_i|} = \left(1 - \frac{|A_i|}{|S_i|}\right) \cdot (1+\varepsilon) = 1 + \varepsilon - (1+\varepsilon) \cdot \frac{|A_i|}{|S_i|} \\
&\implies (1+\varepsilon) \cdot \frac{|A_i|}{|S_i|} \geq \varepsilon \\
&\implies |A_i| = |S_i| - |S_i + 1| \geq \frac{\varepsilon}{1+\varepsilon} \cdot |S_i| \\
&\implies |S_i + 1| \leq \left(1 - \frac{\varepsilon}{1+\varepsilon}\right) \cdot |S_i| = \frac{1}{1+\varepsilon} \cdot |S_i|
\end{aligned}$$

We have proved that $|S_i|$ decreases exponentially. Let's see why in detail:

- $|S_0| = n$,
- $|S_1| \leq \left(1 - \frac{\varepsilon}{1+\varepsilon}\right) \cdot |S_0| = \left(1 - \frac{\varepsilon}{1+\varepsilon}\right) \cdot n$,
- $|S_2| \leq \left(1 - \frac{\varepsilon}{1+\varepsilon}\right) \cdot |S_1| = \left(1 - \frac{\varepsilon}{1+\varepsilon}\right)^2 \cdot n$,
- by induction, $|S_i| \leq \left(1 - \frac{\varepsilon}{1+\varepsilon}\right)^i \cdot n$,
- let $I = \lceil \log_{i+\varepsilon} n \rceil$, then $\left(1 - \frac{\varepsilon}{1+\varepsilon}\right)^I = \left(\frac{1}{1+\varepsilon}\right)^I = (1+\varepsilon)^{-I} \leq (1+\varepsilon)^{-log_{1+\varepsilon} n} = \frac{1}{n}$,
- thus, $|S_I| \leq \left(1 - \frac{\varepsilon}{1+\varepsilon}\right)^I \cdot n < 1$.

That means that the algorithm [6.2] will terminate after I iterations, and so it is sublinear, as we wanted to proof. □

To conclude, we can say that not only this algorithm is very efficient, but it can also be easily implemented in parallel with frameworks such as MapReduce or Pregel.

Chapter 7

Submodular functions[1]

In this section we will present a class of functions, called submodular functions, that generate a class of optimization problems that can be solved with the same algorithm (under some other assumptions). To start this topic we first introduce a problem of this class, the Max cover problem.

7.1 Max Cover Problem

Definition 7.1.1 (Max Cover). *Let $X = [n]$ be the ground set, $\mathscr{S} = 2^X$, $k \geq 1$ be a a parameter of the problem. The Max cover problem consists in finding $\max_{\mathscr{C} \subseteq \mathscr{S},\ |\mathscr{C}|=k} f(\mathscr{C})$, where*

$$f(\mathscr{C}) = \left| \bigcup_{S \subset \mathscr{C}} S \right| \tag{7.1}$$

is the coverage function.

In other words, the objective of the max cover problem is to maximize the number of nodes "covered" at least one time by a fixed size set of nodes.

Example 7.1.1. Let G be the social graph in picture [7.1], and let $k = 2$ the number of influencers we want to give our product as gifts, so that they will advertise it in such a way that as many people as possible will know about our product.

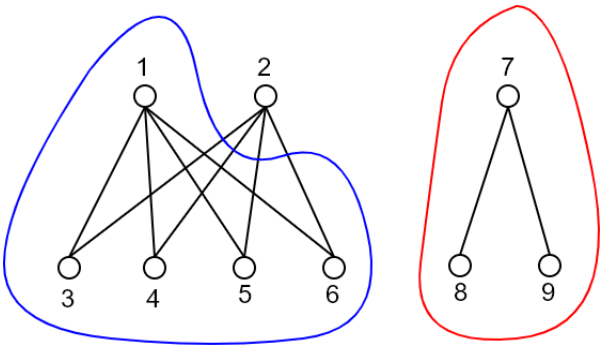


Figure 7.1: An example of max cover on graph G.

$\mathscr{S} = \{\{1,3,4,5,6\},\{2,3,4,5,6\},\{1,3\},\{2,3\},\{1,4\},\{2,4\},\{1,5\},\{2,5\},\{1,6\},\{2,6\},\{7,8,9\},\{7,8\},\{7,9\}\}$ is the set of the neighbors of each node. To cover as many users as possible we choose as the best influencers the nodes 1 and 7.

[1] Part of this chapter is taken from this repo by Cristian Di Pietrantonio.

Note that even if node 2 has more friends than node 7, it would cover the same nodes as 1, so it wouldn't be useful.

Now we present a greedy algorithm that gives an approximation for the max cover problem:

```
Greedy(𝒮, k):
    𝒞_0 ← ∅
    for i = 1, ..., k:
        let S_i ∈ 𝒮 be a set that maximizes f(𝒞_{i-1} ∪ {S_i})
        𝒞_i ← 𝒞_{i-1} ∪ {S_i}
    return 𝒞_k
```

Listing 7.1: The Greedy algorithm to solve the densest subgraph problem

Note that, at each step, we try all the possible sets S_i, and we pick the one that, united to the previous partial solution $\mathscr{C}_{i-1}$, gives the highest value for f, maximizing the next partial solution $\mathscr{C}_i$. Another way of seeing this operation is that, at each step, the algorithm removes the nodes already considered into $\mathscr{C}_{i-1}$, before choosing the possible S_is.

Example 7.1.2. Now we can look at example [7.1.1] with a different prospective:

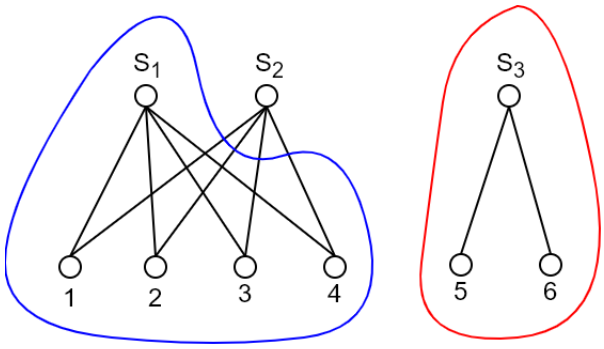


Figure 7.2: An example of application of [7.1].

Now the nodes in the upper part of the graph are the names of the sets we are considering, and the other are the nodes they contain, so $S_1 = S_2 = \{1, 2, 3, 4\}$, $S_3 = \{5, 6\}$ and $k = 2$ as before.

At the beginning we have $\mathscr{C}_0 \leftarrow \emptyset$; then we choose at random between S_1 and S_2, since they give the same result, so we obtain $\mathscr{C}_1 \leftarrow \{S_1\}$; finally, we consider the two possibilities $\{S_1, S_2\}$ and $\{S_1, S_3\}$, but the first one doesn't give any improvement, so we pick S_3 and we obtain $\mathscr{C}_2 \leftarrow \{S_1, S_3\}$, that in this case is the optimal solution.

Theorem 7.1.1. *The algorithm Greedy, described in [7.1], returns a* $\left(1 - \frac{1}{e}\right)$*-approximation for max cover.*

Before going into the proof, let's introduce the incremental return notation:

Definition 7.1.2 (Incremental gain).

$$\Delta(S|A) := f(A \cup \{S\}) - f(A) = \sum_{i \in X} [i \in S \wedge \nexists T \in A, i \in T], \tag{7.2}$$

In other words, $\Delta(S|A)$ *is the number of new elements that we can cover with* S *and that weren't covered before adding* S.
It can be extended to a general gain one obtain with S*, starting at* A.

Proof. Let's fix an optimal solution $\mathscr{C}^* = \{S_1^*, S_2^*, \ldots, S_k^*\}$. Now we'll prove some lemmas that will help us in proving the theorem.

Lemma 7.1.2.

$$\forall\ i \in 0, 1, \ldots, k-1 \quad OPT = f(\mathscr{C}^*) \leq f(\mathscr{C}_i) + k \cdot (f(\mathscr{C}_{i+1}) - f(\mathscr{C}_i)) \tag{7.3}$$

Proof.

$$\begin{aligned}
f(\mathscr{C}^*) &\leq f(\mathscr{C}^* \cup \mathscr{C}_i) && \text{(by monotonicity of } f\text{)} \\
&= f(\mathscr{C}_i) + (f(\{S_1^*\} \cup \mathscr{C}_i) - f(\mathscr{C}_i)) \\
&\qquad + (f(\{S_1^*, S_2^*\} \cup \mathscr{C}_i) - f(\{S_1^*\} \cup \mathscr{C}_i)) \\
&\qquad + \ldots \\
&\qquad + \big(f(\{S_1^*, \ldots, S_{k-1}^*\} \cup \mathscr{C}_i) - f(\{S_1^*, \ldots, S_{k-2}^*\} \cup \mathscr{C}_i)\big) \\
&\qquad + (f(\mathscr{C}^* \cup \mathscr{C}_i) - f(\{S_1^*, \ldots, S_{k-1}^*\} \cup \mathscr{C}_i) && \text{(telescoping series – see } (*^1)\text{)} \\
&= f(\mathscr{C}_i) + \sum_{j=1}^{k} \big(f(\{S_1^*, \ldots, S_j^*\} \cup \mathscr{C}_i) - f(\{S_1^*, \ldots, S_{j-1}^*\} \cup \mathscr{C}_i)\big) \\
& && \text{(just a more compact way to write the same thing)} \\
&= f(\mathscr{C}_i) + \sum_{j=1}^{k} \Delta(S_j^* \mid \{S_1^*, \ldots, S_{j-1}^*\} \cup \mathscr{C}_i) && \text{(by definition of } \Delta \text{ [7.2])} \\
&\leq f(\mathscr{C}_i) + \sum_{j=1}^{k} \Delta(S_j^* \mid \mathscr{C}_i) && \text{(by diminishing returns – see } (*^2)\text{)} \\
&\leq f(\mathscr{C}_i) + \sum_{j=1}^{k} \Delta(S_{i+1} \mid \mathscr{C}_i) && \text{(by } (*^3)\text{)} \\
&\leq f(\mathscr{C}_i) + k \cdot \Delta(S_{i+1} \mid \mathscr{C}_i) \\
&\leq f(\mathscr{C}_i) + k \cdot (f(\mathscr{C}_{i+1}) - f(\mathscr{C}_i))
\end{aligned}$$

In the step marked by $(*^1)$, we "unpack" the function f as a **telescoping series**, where each term, except the first and the last, cancels itself with the next one, so that the result doesn't change, but we can rewrite the expression in a more convenient way for our proof (more about telescoping series on Wikipedia).
In the step marked by $(*^2)$, we exploit the **diminishing returns** property of the function f to give an upper bound, that is, if we already have a big number of sets (i.e., $\{S_1^*, \ldots, S_{j-1}^*\} \cup \mathscr{C}_i$), the gain Δ a new set S_j^* can give us, is less significant then the gain we obtain if we have less sets (i.e., just $\mathscr{C}_i$) (more about diminishing returns on Wikipedia).
The step marked by $(*^3)$ is due to the fact that Greedy maximizes the value of S, so, if S^* had been better then S_{i+1}, Greedy would have kept S^*, instead of picking a new set S_{i+1}. Furthermore, let's note that in the steps $(*^2)$ and $(*^3)$ we are removing the dependencies on the optimal solution, first on the left and then on the right in the argument of Δ. □

With Lemma [7.1.2] we shown that, at each step of Greedy, the partial solution $\mathscr{C}_i$ gives an upper bound to the optimal solution.
Now we want to prove that the difference between the partial and the optimal solution, bounded by $k \cdot (f(\mathscr{C}_{i+1}) - f(\mathscr{C}_i))$, shrinks as the algorithm proceeds with its iterations.

Let's define the difference between the optimal solution and the partial solution at time i as $\delta_i := f(\mathscr{C}^*) - f(\mathscr{C}_i)$.

Observation 7.1.1. $\delta_i - \delta_{i+1} = f(\mathscr{C}^*) - f(\mathscr{C}_i) - (f(\mathscr{C}^*) - f(\mathscr{C}_{i+1})) = f(\mathscr{C}_{i+1}) - f(\mathscr{C}_i)$.

Lemma 7.1.3. *The difference between the optimal solution and the partial solution at time i decreases as i increases:*

$$\delta_{i+1} \leq \left(1 - \frac{1}{k}\right) \delta_i. \tag{7.4}$$

Proof.

$$
\begin{aligned}
& f(\mathscr{C}^*) - f(\mathscr{C}_i) \leq k \cdot (f(\mathscr{C}_{i+1}) - f(\mathscr{C}_i)) && \text{(by lemma [7.1.2])} \\
& \implies \delta_i \leq k \cdot (f(\mathscr{C}_{i+1}) - f(\mathscr{C}_i)) \\
& \implies \delta_i \leq k \cdot (\delta_i - \delta_{i+1}) && \text{(by observation [7.1.1])} \\
& \implies k \cdot \delta_{i+1} \leq k \cdot \delta_i - \delta_i \\
& \implies k \cdot \delta_{i+1} \leq (k-1) \cdot \delta_i \\
& \implies \delta_{i+1} \leq \left(1 - \frac{1}{k}\right) \delta_i
\end{aligned}
$$

□

By applying Lemma [7.1.3], we can finally prove Theorem [7.1.1]:

$$\delta_0 = f(\mathscr{C}^*) - f(\mathscr{C}_0) = f(\mathscr{C}^*)$$

$$\delta_1 \leq \left(1 - \frac{1}{k}\right) \cdot \delta_0 = \left(1 - \frac{1}{k}\right) \cdot f(\mathscr{C}^*)$$

$$\dots$$

$$\delta_i \leq \left(1 - \frac{1}{k}\right) \cdot \delta_{i-1} = \left(1 - \frac{1}{k}\right)^i \cdot f(\mathscr{C}^*)$$

$$\delta_k \leq \left(1 - \frac{1}{k}\right)^k \cdot f(\mathscr{C}^*)$$

Thus $f(\mathscr{C}^*) - f(\mathscr{C}_k) \leq \left(1 - \frac{1}{k}\right)^k \cdot f(\mathscr{C}^*)$

$$\implies f(\mathscr{C}_k) \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \cdot f(\mathscr{C}^*) \geq \left(1 - \frac{1}{e}\right) \cdot f(\mathscr{C}^*) \qquad \text{(by limit [1.3])}$$

Let's note that $f(\mathscr{C}_k)$ is the value returned by Greedy, so the theorem is proved. □

Corollary 7.1.3.1. *The greedy algorithm [7.1] is optimal, that is, it gives the best possible approximation to the max cover problem.*

Observation 7.1.2. The greedy algorithm [7.1] returns a pretty good approximation of the optimal solution even after only half of the k iterations:

$$
\begin{aligned}
& f(\mathscr{C}^*) - f(\mathscr{C}_i) \leq \left(1 - \frac{1}{k}\right)^i \cdot f(\mathscr{C}^*) \\
& \implies f(\mathscr{C}_i) \geq \left(1 - \left(1 - \frac{1}{k}\right)^i\right) \cdot f(\mathscr{C}^*) \\
& \implies \text{if } i = \frac{k}{2},\ f(\mathscr{C}_i) \geq \left(1 - e^{-1/2}\right) \cdot f(\mathscr{C}^*)
\end{aligned}
$$

This means that taking $k/2$ sets instead of k gives us a worse approximation, but not so worse: note that the approximation we obtain using k is $1 - \frac{1}{e} = 0.632$, while the approximation obtained using $k/2$ is $1 - e^{-1/2} = 0.393$.

This result can be extraordinarily useful if we don't actually know the value of k.

An other and even more surprising property is the following one:

Observation 7.1.3. Already at the first iteration, Greedy gives us a useful approximation of the optimal solution:

$$
\begin{aligned}
& f(\mathscr{C}^*) - f(\mathscr{C}_1) \leq \left(1 - \frac{1}{k}\right) \cdot f(\mathscr{C}^*) \\
& \implies f(\mathscr{C}_1) \geq \frac{1}{k} \cdot f(\mathscr{C}^*)
\end{aligned}
$$

This strong and rare property is due to the fact that the first set we pick is the most significant one, since it has the biggest cardinality.

7.2 Submodular functions

Let's take a function $f : 2^{[n]} \to \mathbb{R}$, we can define three classes of functions:

Definition 7.2.1 (Modular function). *f is modular if,* $\forall\ S, T \subseteq [n]$,

$$f(S) + f(T) = f(S \cup T) + f(S \cap T). \tag{7.5}$$

Definition 7.2.2 (Submodular function). *f is submodular if,* $\forall\ S, T \subseteq [n]$,

$$f(S) + f(T) \geq f(S \cup T) + f(S \cap T). \tag{7.6}$$

Definition 7.2.3 (Supermodular function). *f is supermodular if,* $\forall\ S, T \subseteq [n]$,

$$f(S) + f(T) \leq f(S \cup T) + f(S \cap T). \tag{7.7}$$

Example 7.2.1 (Modular function). The cardinality function is modular:

- $f(S) = |S|$ and $f(T) = |T|$,
- $f(S \cap T) = |S \cap T|$,
- $f(S \cup T) = |S \cap T| + |S \Delta T$,
- $f(S) + f(T) = 2|S \cap T| + |S \Delta T$,
- $2|S \cap T| + |S \Delta T = |S \cap T| + |S \cap T| + |S \Delta T|$.

Example 7.2.2 (Subodular function). The coverage function [7.1] is submodular:

- let $S = \{\{1, 2\}\}$ and $T = \{\{1\}\}$
- $f(S) = 2$ and $f(T) = 1$,
- $f(S \cap T) = 0$,
- $f(S \cup T) = 2$,
- $f(S) + f(T) = 3$,
- $3 > 2$.

Theorem 7.2.1. *$f(S)$ is modular iff* $\exists\ z, w_1, \ldots, w_{|S|}$ *such that*

$$f(S) = z + \sum_{i \in S} w_i. \tag{7.8}$$

7.2.1 Submodular functions properties

Lemma 7.2.2 (Diminishing return property). *If f is submodular and $S \subseteq T$, $i \notin T$, then*

$$\Delta(i|S) \geq \Delta(i|T). \tag{7.9}$$

Before going into the proof, let's recall that $\Delta(i|S)$ is the incremental gain at S of i, as defined in [7.1.2].

Proof. Let's take $A = S \cup i, B = T$;

$$\begin{aligned} f(A) + f(B) &\geq f(A \cup B) + f(A \cap B) && \text{(by submodularity)} \\ f(S \cup \{i\}) + f(T) &\geq f(T \cup \{i\}) + f(S) && \text{(by definition of A,B)} \\ f(S \cup \{i\}) - f(S) &\geq f(T \cup \{i\}) - f(T) \\ \Delta(i|S) &\geq \Delta(i|T) && \text{(by definition of } \Delta) \end{aligned}$$

□

Observation 7.2.1. The lemma is actually an "if and only if"; proof of the inverse implication is left as an exercise.

Observation 7.2.2. If f was modular in lemma 7.2.2 the equality would hold instead.

Lemma 7.2.3. *If $f_1, f_2, \ldots, f_k : 2^x \mapsto \mathbb{R}$ are submodular, and $\alpha_1, \alpha_2, \ldots, \alpha_k \geq 0$, then*

$$g(S) := \sum_{i=1}^{k} \alpha_i f_i(S) \tag{7.10}$$

is submodular as well.

Proof.

$$\begin{aligned} g(S) + g(T) &= \sum_{i=1}^{k} (\alpha_i f_i(S)) + \sum_{i=1}^{k} (\alpha_i f_i(T)) \\ &= \sum_{i=1}^{k} (\alpha_i f_i(S) + \alpha_i f_i(T)) \\ &= \sum_{i=1}^{k} (\alpha_i (f_i(S) + f_i(T))) \\ &\geq \sum_{i=1}^{k} (\alpha_i (f_i(S \cap T) + f_i(S \cup T))) && \text{(by submodularity)} \\ &= \sum_{i=1}^{k} (\alpha_i f_i(S \cap T)) + \sum_{i=1}^{k} (\alpha_i f_i(S \cup T)) \\ &= g(S \cap T) + g(S \cup T) \end{aligned}$$

□

Observation 7.2.3. If one of the αs is negative the lemma 7.2.3 doesn't hold.

7.2.2 One particular submodular function: $f_{\mathscr{C}}(A)$

Definition 7.2.4. *Let $\mathscr{C} \subseteq 2^{[n]},\ w : \mathscr{C} \mapsto \mathbb{R}_{\geq 0},\ A \subseteq [n]$,*

$$f_C(A) = \sum_{S \in \mathscr{C}} (w(S) \cdot [A \cap S \neq \emptyset]) . \tag{7.11}$$

This function looks for all the sets of a class and sums the weights of the ones that intersect the parameter set.

Example 7.2.3. Let $\mathscr{C} = \{\{1,2\},\{2,3\}\},\ w(\{1,2\}) = 1,\ w(\{2,3\}) = 3$, then $f_{\mathscr{C}}(\{1\}) = 1$ and $f_{\mathscr{C}}(\{2\}) = 4$.

Observation 7.2.4. Note that the coverage function [7.1] can be written in this form too.

Lemma 7.2.4. *$f_{\mathscr{C}}(A)$ is submodular.*

Proof. Fix $A, B \subseteq [n]$; let

$$\mathscr{C}_A = \{S | S \in \mathscr{C} \ \wedge \ S \cap A \neq \emptyset \ \wedge \ S \cap B = \emptyset\}$$
$$\mathscr{C}_B = \{S | S \in \mathscr{C} \ \wedge \ S \cap B \neq \emptyset \ \wedge \ S \cap A = \emptyset\}$$
$$\mathscr{C}_\cap = \{S | S \in \mathscr{C} \ \wedge \ S \cap A \neq \emptyset \ \wedge \ S \cap B \neq \emptyset\}$$

i.e., $\mathscr{C}_A$ are the sets that intersect A but not B, $\mathscr{C}_B$ are the sets that intersect B but not A and $\mathscr{C}_\cap$ are the sets that intersect both A and B.

Observation 7.2.5. The conditions are mutually exclusive.

Then,

$$f_{\mathscr{C}}(A) = \sum_{S \in \mathscr{C}_A} w(S) + \sum_{S \in \mathscr{C}_\cap} w(S)$$
$$f_{\mathscr{C}}(B) = \sum_{S \in \mathscr{C}_B} w(S) + \sum_{S \in \mathscr{C}_\cap} w(S)$$
$$f_{\mathscr{C}}(A \cup B) = \sum_{S \in \mathscr{C}_A} w(S) + \sum_{S \in \mathscr{C}_B} w(S) + \sum_{S \in \mathscr{C}_\cap} w(S)$$
$$f_{\mathscr{C}}(A \cap B) \leq \sum_{S \in \mathscr{C}_\cap} w(S) \qquad (*)$$

Note that eq. $(*)$ is true because of cases like the one in fig. 7.3; in general in $\mathscr{C}_\cap$ there may be more sets than in $A \cap B$.

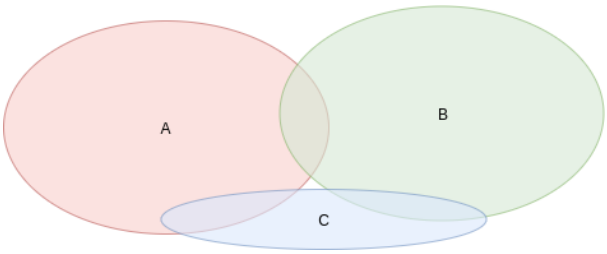


Figure 7.3: Set C has non-empty intersection with both B and A but does not intersect their intersection.

So we have that

$$\begin{aligned} f_{\mathscr{C}}(A) + f_{\mathscr{C}}(B) &= \sum_{S \in \mathscr{C}_A} w(S) + \sum_{S \in \mathscr{C}_B} w(S) + 2 \sum_{S \in \mathscr{C}_\cap} w(S) \\ &= f_{\mathscr{C}}(A \cup B) + \sum_{S \in \mathscr{C}_{cap}} w(S) \\ &\geq f_{\mathscr{C}}(A \cup B) + f_{\mathscr{C}}(A \cap B) \qquad \text{(by inequality eq. } (*)\text{)} \end{aligned}$$

And this finally prove the lemma [7.2.4]. □

Observation 7.2.6. Note that $f_{\mathscr{C}}(A)$ could not be directly computable, since if there is an exponential number of subset $S \in \mathscr{C}$, we must compute a sum for each of them, and this requires exponential time.

7.3 KKT model

KKT stands for Kempe, Kleinsberg and Tardos. Let's assume we have k gift products to give to k persons for free for marketing reasons; let's assume as well that whenever a person buys the product he keeps it forever.

Moreover, let's assume that time is discrete and that the product spreads over time:

- at time t_0 the only persons having the product will be the ones that got it for free;
- at time t_i for each of the edges incident on the nodes having the product we will be flipping a coin:
  - with prob p the product will spread on that edge;
  - else the edge is lost forever.

While this point of view is dynamic, there is a static point of view which states the following: Regardless of which nodes are activated at time 0 what actually happens is that a coin is flipped indipendently for each edge in the graph:

- if the coin is H (head), the edge will stay there;
- else, the edge will be removed (won't be used for spreading the product).

Nature will then make its choices, removing some edges. By looking at the connected components after nature removes the edges we see that:

- each connected component containing at least one seed node will be fully conquered;
- the others will be forever lost.

If we knew the final components, it would be clear which seed set to choose (we would sort the components by decreasing size and pick one node for each of the first k components); sadly, this is not the case because the set of seed nodes must be chosen a priori.

Suppose that, after nature's random choices, A is the set of "*used*" or "*survived*" edges, and $\mathscr{C}_A$ is the partition of the node set V into connected components with A as the edge set.
Let $R(S)$ be the set of nodes "*conquered*" with seed set S; this is clearly a random variable depending on which edges survive.

We can now ask what's the expected cardinality of the reached set $R(S)$ conditioned on the used edges set A; we, in fact, would like to maximize this quantity:

$$E\left[|R(S)|\middle|A\right] = \sum_{T \in \mathscr{C}_A} (|T| \cdot [T \cap S \neq \emptyset]) =: f_A(S) \tag{7.12}$$

Claim 7.3.0.1. *f_A is submodular.*

Proof. Follows from lemma 7.2.4. □

Moreover, we can define the expected value as a function of S without conditioning it on a particular A by using the law of total expectation.

$$f(S) = \sum_{A \subseteq E} (f_A(S) \cdot Pr\{A \text{ is the set of survived edges}\}) = E[|R(S)|] \tag{7.13}$$

Claim 7.3.0.2. *$f(S)$ is submodular.*

Proof. Follows from lemma 7.2.3. □

The problem in optimazing this function is that computing it on a single set takes exponential time, but it can be approximated very well; as an exercise you can try to give the best sampling algorithm to

do it (hint: an approximation can be obtained by using [1.12]). Note that it is possible to compute the average among the various results because the value of $f(S)$ is bounded between k and n.

The good news is that, since $f(S)$ is submodular, we already have an algorithm to optimize it, that is a slight variation of the one we discussed in section Max Cover Problem, as we will see in the next section.

7.4 A greedy algorithm for submodular functions.

Now we present a greedy algorithm that gives an approximation for $f(S)$ [7.13], and – as we will see – for any monotone, non-negative and submodular function:

```
Greedy_{f,ε}(k):
    S_0 ← ∅
    for i = 1, ..., k:
        let x_i be s.t. Δ(x_i|S_{i-1}) ≥ (1 − ε) · max_x Δ(x|S_{i-1})
        S_i ← S_{i-1} ∪ {x_i}
    return S_k
```

Listing 7.2: Greedy algorithm

Note that this is almost the same algorithm as [7.1], with the exception that we don't pick the element that maximize f, but a $(1-\varepsilon)$-approximation ot the optimal element.

Theorem 7.4.1. *If f is monotone, non-negative and submodular Greedy returns a $(1-e^{-1+\epsilon})$-approximation.*

Observation 7.4.1. Note that $1 - e^{-1+\epsilon} = 1 - \frac{1}{e} - O(\varepsilon)$.

Proof of theorem 7.4.1. Let's pick an optimal solution $S^* = \{x_1^*, x_2^*, \ldots, x_k^*\}$, $\forall\, i \in \{0, \ldots, k-1\}$ we have that:

$$
\begin{aligned}
f(S^*) &\leq f(S^* \cup S_i) && \text{(by monotonicity of } f\text{)} \\
&= f(S_i) + (f(\{x_1^*\} \cup S_i) - f(S_i)) \\
&\qquad + (f(\{x_1^*, x_2^*\} \cup S_i) - f(\{x_1^*\} \cup S_i)) \\
&\qquad + \ldots \\
&\qquad + \big(f(\{x_1^*, \ldots, x_k^*\} \cup S_i) - f(\{x_1^*, \ldots, x_{k-1}^*\} \cup S_i)\big) \\
&\qquad + (f(S^* \cup S_i) - f(\{x_1^*, \ldots, x_{k-1}^*\} \cup S_i)) && \text{(telescoping series – see } (*^1)\text{)} \\
&= f(S_i) + \Delta(x_1^*|S_i) + \Delta(x_2^*|S_i \cup \{x_1^*\}) + \ldots + \Delta(x_k^*|S_i \cup \{x_1^*, \ldots, x_{k-1}^*\}) \\
& && \text{(by definition of } \Delta \text{ [7.2])} \\
&= f(S_i) + \sum_{j=1}^{k} \big(\Delta(x_j^*|S_i \cup \{x_1^*, \ldots, x_{j-1}^*\})\big) \\
&\leq f(S_i) + \sum_{j=1}^{k} \big(\Delta(x_j^*|S_i)\big) && \text{(by diminishing returns – see } (*^2)\text{)} \\
&\leq f(S_i) + \sum_{j=1}^{k} \left(\frac{1}{1-\varepsilon}\Delta(x_{i+1}|S_i)\right) && \text{(by greedy choice)} \\
&= f(S_i) + \frac{k}{1-\varepsilon}\Delta(x_{i+1}|S_i)
\end{aligned}
$$

where $(*^1)$ and $(*^2)$ are explained in detail in the proof of lemma 7.1.2. □

Chapter 8

Clustering

In clustering problems, we have a dataset of points in a multi-dimensional space, and we want to partition this dataset into a certain number of clusters, such that a certain distance or cost functions is minimized within the clusters and maximized among them.

In traditional approaches to this problem, the number of clusters k is fixed in advance, and the algorithm partitions the dataset in the given number of clusters.
The best known algorithms of this type are:

- *k-medians*: find the location of the k centers as to minimize the average distance between a point and a center;
- *k-centers*: find the location of the k centers as to minimize the maximum distance between a point and a center;
- *k-means*: find the location of the k centers as to minimize the square of the distance between a point and a center;

Observation 8.0.1. The most used is k-means, because its objective functions resembles variance, and so it can be interpreted as the search of Gaussians that randomly generated the points of the dataset.

Theorem 8.0.1. *k-medians, k-centers and k-means are NP-complete problems.*

Example 8.0.1. We have a dataset of vectors representing users of a social network. Each vector could have up to millions dimensions (this is called *curse of dimensionality*), and we could have billions of such vectors.
The distance between any pair of nodes tells us how much they are similar, so we want to cluster similar users to recommend them similar products or movies, or to send them similar ads.

Example 8.0.2. We are Amazon and we have a dataset of users, each one with his/her geographic position in a map, so the distance between two points is the actual distance between two users. We have a fixed budget to position k warehouses, and we want to find the best places where to put them.
A good strategy could be to apply k-medians, so that each client, on average, is at the minimum possible distance from the warehouse.

In real cases, often the number of clusters k is not known *a priori*, so one can use *hierarchical clustering*, but this technique only produces some levels of grouping, so the user of the model still have to decide how many groups to use, and this is not always trivial.

Example 8.0.3. In the example in figure [8.1] we can see that, after the algorithm groups the elements in levels of clusters, the user must decide which level to use, of if an intermediate number of clusters is preferable.

As we will see, in *correlation clustering* model it is the algorithm itself that produces the optimal number of clusters.

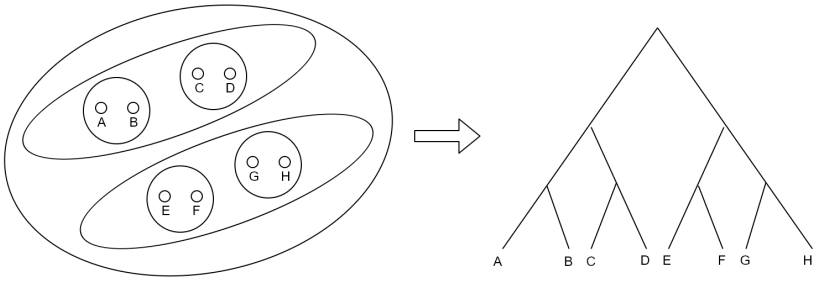


Figure 8.1: An example of hierarchical clustering.

Generally speaking, if the user wants to minimize the distance within clusters, an optimal but useless solution would be to pick a singleton for each data point, whereas if the aim is to maximize the distance among clusters, one could end up with a single cluster that contains all the points, another optimal but useless solution, but these scenarios can't happen with correlation clustering.

8.1 Correlation clustering

We have a certain distance metric d and a graph $G(V, E^+, E^-)$, where E^+ is the set of positive edges, that is, the ones that link nodes that are similar according to d, E^- is the set of negative edges, that is, the ones that link dissimilar nodes according to d.
Note that $E^+ \cup E^- = \binom{V}{2}$ (i.e., the union of the edges in E^+ and E^- is the set of all the edges connecting every pair of distinct vertices) and $E^+ \cap E^- = \emptyset$ (i.e., an edge can't be both positive and negative).
The goal is to put similar nodes together and different nodes separated, that is, we want to minimize the number of errors, given by the number of similar nodes put in different cluster and dissimilar nodes put in the same cluster, as formalized by the *cost* function:

Definition 8.1.1 (Cost). *Given a partition $\mathscr{C} = \{C_1, \ldots, C_i, \ldots\}$ of V (that is, $C_i \cup C_j = \emptyset \ \forall \ i \neq j$, $\bigcup_{C_i \in \mathscr{C}} = V$, $C_i \neq \emptyset \ \forall \ C_i$), the cost function is defined as follows:*

$$Cost(\mathscr{C}) = \sum_{\{i,j\} \in \binom{V}{2}} \Big(\big[i, j \in E^+ \wedge \ i \text{ and } j \text{ are in different clusters of } \mathscr{C} \big] + \big[i, j \in E^- \wedge \ i \text{ and } j \text{ are in the same cluster of } \mathscr{C} \big] \Big) \tag{8.1}$$

N.B.: *Even if there exist an edge connecting any pair of distinct vertices, in the following pictures sometimes we don't draw all the negative edges; so, if there isn't an edge between two nodes, we assume that there exists an implicit negative edge between those nodes.*

Example 8.1.1. Let's look at an example of correlation clustering in picture [8.2], where negative edges are dashed.

In this case, we put the nodes A, B, D, E together in cluster C_1 and we let node C alone in cluster C_2, since the first nodes are all similar to each other, while the other is dissimilar from all the others.

Now we're interested in some **special cases**, shown in figure [8.3]:

a. If the nodes are all similar to each other, a single cluster will be created, and the *cost* is 0;

b. If nodes are all dissimilar to each other, a singleton for each node will be created, and the *cost* is 0;

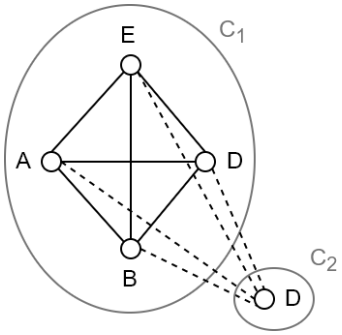


Figure 8.2: An example of correlation clustering.

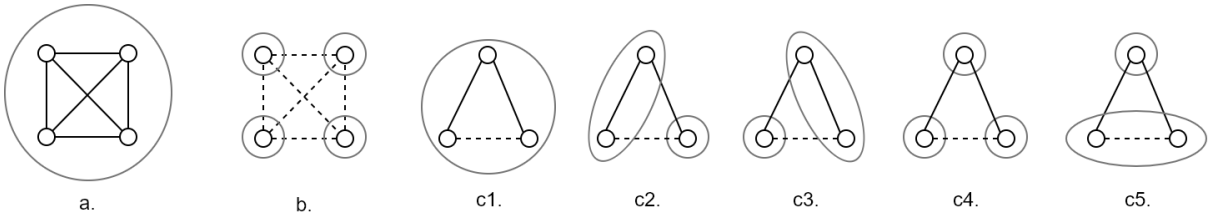


Figure 8.3: Special cases in correlation clustering.

c. If there are "triangles" like the ones in figures c1-c5, whatever clustering we produce the *cost* is at least 1.

These examples allow us to make some observations:

Observation 8.1.1. Since the solution can have $cost = 0$, we can produce a multiplicative approximation only if we manage to always find the optimal solution if it has $cost = 0$, otherwise the approximation will be infinite, even if the cost is small.

Observation 8.1.2. We know that if the graph is composed of disjoint positive cliques, there exists a solution with $cost = 0$, and we can find it by removing negative edges and looking for connected components. Thus, we can produce a multiplicative approximation.

Observation 8.1.3. In whatever way we find triangles made of two positive edges and a negative edge, the *cost* will be at least 1 for each such triangle, and for that reason those triangles are called "*bad triangles*" (we will give a formal definition later).
Furthermore, the *cost* for each bad triangle is at least 1 even if there is a shared node between the bad triangles (see figure [8.4]a.), but it is 1/2 if there is a shared edge (see figure [8.4]b.).

Note that in figure [8.4] the gray circles represent bad triangles instead of clusters.

Definition 8.1.2 (Bad triangle). *If* $\{i,j\} \in E^+ \wedge \{j,k\} \in E^+ \wedge \{i,k\} \in E^-$, *then* $\{i,j,k\}$ *is a bad triangle.*

The set of all the bad triangles of a graph is $\mathscr{T} = \left\{ \{i,j,k\} \;\middle|\; \{i,j,k\} \in \binom{V}{3} \text{ is a bad triangle} \right\}$.

8.1.1 Randomized pivot

Now we present a greedy algorithm by Ailon, Charikar, Newman to approximate the correlation clustering problem:

```
randomized_pivot(V, E+, E−):
    V_1 ← ∅
    i ← 1
    while V_i ≠ ∅:
```

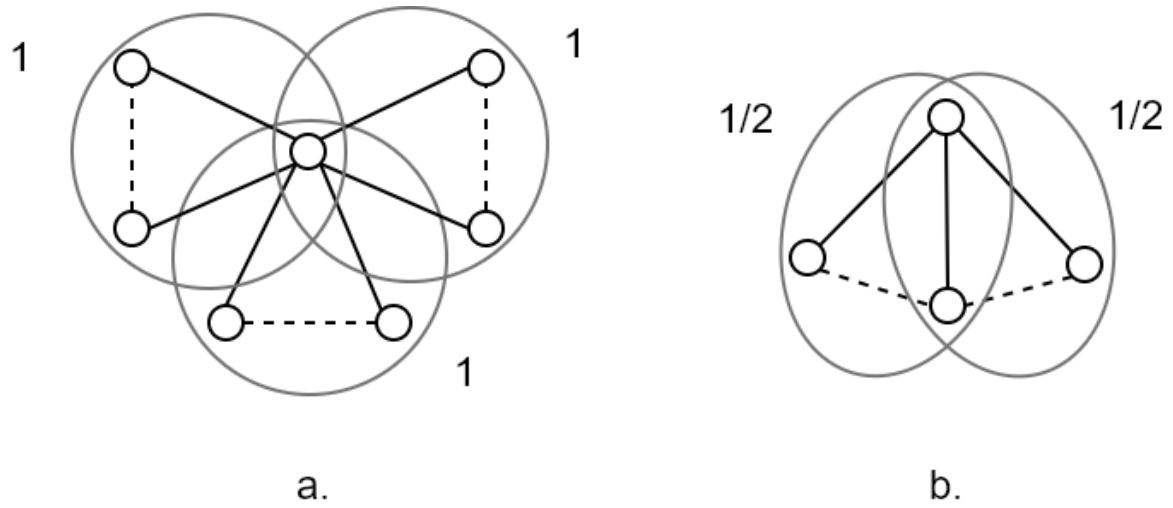


Figure 8.4: Cost of bad triangles.

```
    pick v_i UAR from V_i // we choose the the pivot v_i
    C_i ← {w | w ∈ V_i ∧ {v_i, w} ∈ E^+} ∪ {v_i} // we create a new cluster V_{i+1}
    V_{i+1} ← V_i − C_i // we remove all the nodes of V_{i+1}
    i ← i + 1
return 𝒞_G = {C_1, C_2, ..., C_{i−1}}
```

Listing 8.1: Randomized Pivot

Example 8.1.2. Let's look at some example of execution of Randomized Pivot.

In figure [8.5] we can see all the steps performed by Randomized Pivot to obtain a good approximation of the correlation clustering problem: it produces three clusters, but with one error: there is a positive edge between clusters C_2 and C_2.

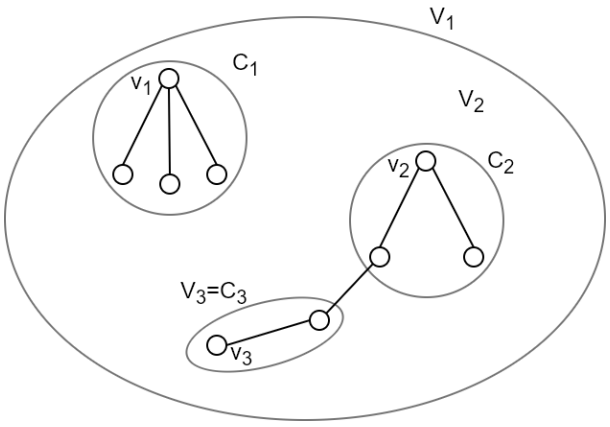


Figure 8.5: Approximated solution of Randomized Pivot.

In figure [8.6] we can see that, if it exists, Randomized Pivot successes to obtain the optimal solution with $cost = 0$: it creates a cluster for each clique, as previously said in observation [8.1.2].

In figure [8.7] we see a case in which Randomized Pivot could find a very bad approximation: if there is a positive star, there are two possibilities:

a. the algorithm chooses a leave of the star as pivot, so the *cost* is only $n - 1$;

b. the algorithm chooses the root of the star as pivot, so the *cost* is even $\binom{n-1}{2} \approx n^2$.

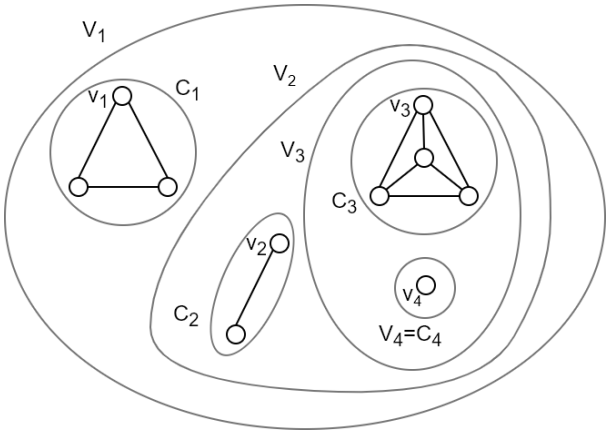


Figure 8.6: Optimal solution of Randomized Pivot.

Note that the second (and worst) possibility happen with probability $1/n$, so it's not so unlikely.

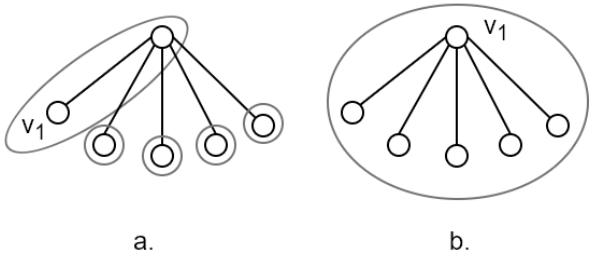


Figure 8.7: Worst case of approximated solution of Randomized Pivot.

Theorem 8.1.1. *Randomized Pivot [8.1] returns an expected 3-approximation:*

$$E\left[cost\left(\mathscr{C}_G\right)\right] \leq 3 \cdot \min_{\mathscr{C}}\{cost(\mathscr{C})\} = 3 \cdot OPT. \tag{8.2}$$

Observation 8.1.4. By using Markov Inequality [1.10], it can be proved that the bad event have small probability, if we execute the algorithm many times:

- Let $X = cost(\mathscr{C})$, X is a not concentrated random variable with expected value $E[X] \leq 3 \cdot OPT$;
- $\Pr\{X \geq 3 \cdot c \cdot OPT\} \leq \Pr\{X \geq c \cdot E[X]\} \leq \dfrac{1}{c}$;
- If we pick $c = 1 + \dfrac{\varepsilon}{3}$, then $\Pr\{X \geq (3+\varepsilon) \cdot OPT\} \leq \dfrac{1}{1+\varepsilon/3} = 1 - \Theta(\varepsilon)$;
- Thus, if we execute the algorithm $\dfrac{1}{\Theta(\varepsilon)}$ times, the probability of the bad event becomes small: $(1-\Theta(\varepsilon))^{\left(\frac{\ln 1/\delta}{\varepsilon}\right)} \to \Theta(\delta)$;
- At this point, it is sufficient to return the best solution among those computed by the algorithm in the various trials, and with high probability it will be a solution that respects the 3-approximation, that is, a solution for which the bad event doesn't happen.

Proof of Theorem 8.1.1. As usual, we will prove the theorem by steps, introducing some lemmas and claims.

Lemma 8.1.2. *Let* $cost_j^+ := \left|\left\{\{v,w\} \mid v \in C_j \wedge w \in V_{j+1} \wedge \{v,w\} \in E^+\right\}\right|$ *and* $cost_j^- := \left|\left\{\{v,w\} \mid v,w \in C_j \wedge \{v,w\} \in E^-\right\}\right|$, *then*

$$cost(\mathscr{C}_G) = \sum_j \left(cost_j^+ + cost_j^-\right). \tag{8.3}$$

Proof. The proof directly follows from the definition of $cost_j^+$ and $cost_j^-$: as we can see in figure [8.8], $cost_j^+$ covers the first part of the definition of *cost* [8.1], that is, the cases in which the algorithm puts two similar nodes in different clusters (edges marked with $*$ in the picture), and $cost_j^-$ covers the second part of the definition, i.e., the cases in which two dissimilar nodes are put in the same cluster (edges marked with $**$ in the picture). Thus, all the errors (red edges) are covered either by $cost_j^+$ or by $cost_j^-$. □

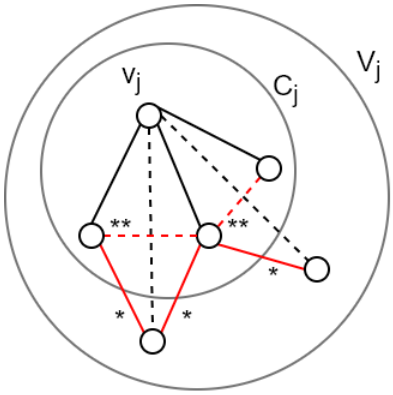


Figure 8.8: Example of application of the definition of $cost_j^+$ and $cost_j^-$.

The following two claims, that follow from Lemma [8.1.2], connect the different types of bad triangles we see in figure [8.8] with the different types of errors, or costs.

Claim 8.1.2.1. $cost_j^+$ *is the number of bad triangles* T *that*

1. *are fully contained in* V_j,
2. *contain* v_j, *and*
3. *for which* $|C_j \cup T| = 2$.

Claim 8.1.2.2. $cost_j^-$ *is the number of bad triangles* T *that*

1. *are fully contained in* $C_j \subseteq V_j$,
2. *contain* v_j.

Now we introduce the concept of "hitting" a bad triangle, that will be useful to prove the next lemma.

Definition 8.1.3 (Hit). *We say that Random-Pivot [8.1]* hits *a bad triangle* T *if* $\exists\ j \mid T \subseteq V_j$ *and* $v_j \in T$.

Corollary 8.1.2.1. $cost(\mathscr{C}_G)$ *is the number of bad triangles hit by Random-Pivot.*

Proof. Corollary [8.1.2.1] follows by the claims [8.1.2.1] and [8.1.2.2] and by the definition of "hit" [8.1.3]. □

We introduced the concept of *hitting* because we want to relate the probability of hitting a bad triangle with the cost paid by the algorithm. We can't directly compute this probability, but we obtain a useful linear relationship.

Lemma 8.1.3. *Given a bad triangle $T \in \mathscr{T}$, let p_T be the (a priori) probability that Random-Pivot will hit T. Then*

$$E[cost(\mathscr{C}_G)] = \sum_{T \in \mathscr{T}} p_T. \tag{8.4}$$

Proof. Lemma [8.1.3] follows by Corollary [8.1.2.1] and the definition of expected value [1.2.1]: we know that $cost(\mathscr{C}_G)$ is a random variable (it depends on $\mathscr{C}_G$, that is the result returned by Randomized Pivot, that in turn is a random algorithm, as the name suggests), and, since its value is the number of bad triangles hit by the algorithm, we can decompose it in a sum of Bernoulli random variables X_T, one for each $T \in \mathscr{T}$, that assume value 1 if the algorithm hits T, with probability p_T, and 0 otherwise. □

So, finally we have all the tools we need to prove that $E\left[cost\left(\mathscr{C}_G\right)\right] \leq 3 \cdot OPT$ (the claim of Theorem [8.1.1]), that is, $\frac{\sum_{T \in \mathscr{T}} p_T}{3} \leq OPT$, and we'll do it with a primal-dual proof, exploiting the linear program underlying Randomized-Pivot.

We begin with the dual LP:

$$\begin{aligned} &\min \sum_{\{i,j\} \in \binom{V}{2}} X_{\{i,j\}} \\ &\left\{ X_{\{i,j\}} + X_{\{j,k\}} + X_{\{i,k\}} \geq 1 \quad \forall \{i,j,k\} \in \mathscr{T} \right. \\ &X_{\{i,j\}} \geq 0 \; \forall \{i,j\} \in \binom{V}{2} \end{aligned} \tag{8.5}$$

Observation 8.1.5. The objective function is the number of errors the algorithm makes (that is, the cost function), while the meaning of the constraint is that the sum of the costs paid for a bad triangle is at least 1, since we know we will make at least one error for each bad triangle.

Lemma 8.1.4. *For each solution $\mathscr{C}$, $cost(\mathscr{C}) \geq DUAL^*$, where $DUAL^*$ is the optimal solution of the dual LP.*

Proof. It is hard to find the optimal solution of the dual, but for our purpose it is sufficient to show a feasible solution and it will be greater or equal than the optimal one.

$$\text{Let } X_{\{i,j\}} := \begin{cases} 1 & \text{if } \left(\{i,j\} \in E^+ \wedge i \text{ and } j \text{ are split by } \mathscr{C}\right) \text{ or} \\ & \quad \left(\{i,j\} \in E^- \wedge i \text{ and } j \text{ are together in } \mathscr{C}\right). \\ 0 & \text{otherwise} \end{cases}$$

Informally, $X_{\{i,j\}} = 1$ iff there is an error in $\mathscr{C}$.

This is a feasible solution because we are counting each bad triangle, and so the value of the dual, with this solution, is equal to $cost(\mathscr{C})$. □

Corollary 8.1.4.1.

$$OPT \geq DUAL^* \tag{8.6}$$

Proof. Since $OPT = \min_{\mathscr{C}}\{cost(\mathscr{C})\}$, the corollary follows directly from Lemma [8.1.4]. □

Now we are going to present the primal LP and to show that $\frac{p_T}{3}$ is a feasible solution. This will allow us to conclude that the following inequality holds:

Claim 8.1.4.1.

$$\frac{1}{3} \cdot E[cost(\mathscr{C}_P)] = \sum_{T \in \mathscr{T}} \frac{p_T}{3} \leq PRIMAL^* \leq DUAL^* \leq OPT \tag{8.7}$$

Observation 8.1.6. If we prove the first inequality, we will have proved that the expected cost of the solution returned by the algorithm is no more than 3 times the cost of the optimal solution. Note that this concludes the proof of Theorem [8.1.1].

Proof. The first equation follows from Lemma 8.1.3, the second inequality is due to the weak duality theorem [6.1.1], the third one to Corollary [8.1.4.1], and we'll prove the first one shortly with Lemma [8.1.5].

Primal LP:

$$\begin{aligned} &\max \sum_{T \in \mathscr{T}} y_T \\ &\begin{cases} \sum_{T \in \mathscr{T},\ \{i,j\} \subset T} y_T \leq 1 \quad \forall \{i,j\} \in \binom{V}{2} \end{cases} \\ &y_T \geq 0 \ \forall \ T \in \mathscr{T} \end{aligned} \tag{8.8}$$

Observation 8.1.7. This kind of constraint is called *fractional packing of triangles*: for each edge, we want to take at most a weight of 1, summing up the weights of the bad triangles that contain that edge.

Consider the primal program, and set $y_T = p_T/3$ for each $T \in \mathscr{T}$. This solution has a value of $\frac{1}{3} \cdot \sum_{T \in \mathscr{T}} p_T$ in the primal so, if we prove it is feasible for the primal, we have shown that $\frac{1}{3} \cdot \sum_{T \in \mathscr{T}} p_T \leq \pi^*$, where π^* is the solution of the primal.

Lemma 8.1.5. $y_T = \dfrac{p_T}{3}$ *is a feasible solution for the primal.*

Proof. To conclude that the solution is feasible for the primal, we only have to prove that, for each $\{i,j\} \in \binom{V}{2}$, it holds $\sum_{\substack{T \in \mathscr{T} \\ T \supset \{i,j\}}} y_T \leq 1$.

Fix an edge $\{i,j\} \in \binom{V}{2}$, and let $A = \bigcup_{\substack{T \in \mathscr{T} \\ T \supset \{i,j\}}} T$ be the set of each node of each bad triangle that includes the edge $\{i,j\}$. Let ξ be the event that, during a run of the algorithm, at least one node of A becomes a pivot.
Note that we need ξ because we know that, if it happens, the algorithm will find at least a triangle in A, otherwise none, and this allows us to divide the probabilities in two parts, by lat of total probability.

Then,

$$\begin{aligned}
\sum_{\substack{T\in\mathscr{T}\\ T\supset\{i,j\}}} y_T &= \sum_{\substack{T\in\mathscr{T}\\ T\supset\{i,j\}}} \frac{p_T}{3} \\
&= \frac{1}{3}\cdot \sum_{\substack{T\in\mathscr{T}\\ T\supset\{i,j\}}} \Pr\{T \text{ is hit}\} \\
&= \frac{1}{3}\cdot \sum_{\substack{T\in\mathscr{T}\\ T\supset\{i,j\}}} \Big(\Pr\{T \text{ is hit} \mid \xi\}\cdot\Pr\{\xi\} + \Pr\big\{T \text{ is hit} \mid \overline{\xi}\,\big\}\cdot\Pr\big\{\,\overline{\xi}\,\big\}\Big) \qquad \text{(by total probability)} \\
&= \frac{1}{3}\cdot \sum_{\substack{T\in\mathscr{T}\\ T\supset\{i,j\}}} \Big(\Pr\{T \text{ is hit} \mid \xi\}\cdot\Pr\{\xi\} + 0\cdot\Pr\big\{\,\overline{\xi}\,\big\}\Big) \\
&= \frac{1}{3}\cdot \sum_{\substack{T\in\mathscr{T}\\ T\supset\{i,j\}}} \Big(\Pr\{T \text{ is hit} \mid \xi\}\cdot\Pr\{\xi\}\Big) \\
&\leq \frac{1}{3}\cdot \sum_{\substack{T\in\mathscr{T}\\ T\supset\{i,j\}}} \Pr\{T \text{ is hit} \mid \xi\}\,.
\end{aligned}$$

Under the conditioning ξ, there will be one node that ends up being the first pivot chosen in A; suppose it is v_k (that is, let k be such that $v_k \in A$ and $v_{k'} \notin A$ fo each $1 \leq k' < k$ — observe that such a k must exists if ξ happens). Moreover, let $B = V_k \cap A$ be the subset of nodes of A that are part of the residual graph when v_k is chosen as a pivot.

Then,

$$\sum_{\substack{T\in\mathscr{T}\\ T\supset\{i,j\}}} y_T \le \frac{1}{3}\cdot\sum_{\substack{T\in\mathscr{T}\\ T\supset\{i,j\}}} \Pr\{T \text{ is hit} \mid \xi\}$$

$$= \frac{1}{3}\cdot\sum_{S\subseteq A}\left(\Pr\{B=S\mid\xi\}\cdot\sum_{\substack{T\in\mathscr{T}\\ T\supset\{i,j\}}}\Pr\{T \text{ is hit}\mid B=S\wedge\xi\}\right) \qquad (*^1)$$

$$= \frac{1}{3}\cdot\sum_{S\subseteq A}\left(\Pr\{B=S\mid\xi\}\cdot\left(\sum_{\substack{T\in\mathscr{T}\\ \{i,j\}\subset T\subseteq S}}\Pr\{T \text{ is hit}\mid B=S\wedge\xi\}+\sum_{\substack{T\in\mathscr{T}\\ \{i,j\}\subset T\not\subseteq S}}\Pr\{T \text{ is hit}\mid B=S\wedge\xi\}\right)\right) \qquad (*^2)$$

$$= \frac{1}{3}\cdot\sum_{S\subseteq A}\left(\Pr\{B=S\mid\xi\}\cdot\left(\sum_{\substack{T\in\mathscr{T}\\ \{i,j\}\subset T\subseteq S}}\Pr\{T \text{ is hit}\mid B=S\wedge\xi\}+\sum_{\substack{T\in\mathscr{T}\\ \{i,j\}\subset T\not\subseteq S}}0\right)\right) \qquad (*^3)$$

$$= \frac{1}{3}\cdot\sum_{S\subseteq A}\left(\Pr\{B=S\mid\xi\}\cdot\sum_{\substack{T\in\mathscr{T}\\ \{i,j\}\subset T\subseteq S}}\Pr\{T \text{ is hit}\mid B=S\wedge\xi\}\right)$$

$$= \frac{1}{3}\cdot\sum_{S\subseteq A}\left(\Pr\{B=S\mid\xi\}\cdot\sum_{\substack{T\in\mathscr{T}\\ \{i,j\}\subset T\subseteq S}}\frac{3}{|S|}\right) \qquad (*^4)$$

$$= \frac{1}{3}\cdot\sum_{S\subseteq A}\left(\Pr\{B=S\mid\xi\}\cdot(|S|-2)\cdot\frac{3}{|S|}\right)$$

$$\le \frac{1}{3}\cdot\sum_{S\subseteq A}\Big(\Pr\{B=S\mid\xi\}\cdot 3\Big) \qquad \text{(we upper bound } \frac{|S|-2}{|S|} \text{ with 1)}$$

$$= \sum_{S\subseteq A}(\Pr\{B=S\mid\xi\}) = \Pr\{B\subseteq A\mid\xi\} = 1.$$

In $(*^1)$ we apply the law of total probability again, to split the probabilities according to the value of B; in $(*^2)$ we break the sum in two parts, with triangles completely in S and *not* completely in S, the latter will never be hit by the algorithm, by definition of hit [8.1.3], (so their probabilities are 0 in $(*^3)$), the former have probability $3/|S|$ of being hit, since each of their vertexes has the same probability of being chosen as the pivot (since it's picked UAR), and we use it in $(*^4)$.

Thus, the generic constraint of the primal is satisfied: the proposed primal solution is then feasible, which concludes the proof of Lemma [8.1.5]. □

Then, $\frac{1}{3}\cdot\sum_{T\in\mathscr{T}} p_T$ cannot be larger than the optimal primal value π^*. The proof of Claim [8.1.4.1] is concluded. □

By Claim [8.1.4.1], we have $E[\text{cost}(\mathscr{C}_P)] \le 3\cdot\text{cost}(\mathscr{C}^\star)$, that concludes the proof of Theorem [8.1.1], as observed in [8.1.6]. □

8.1.2 Parallel Random Pivot

Observation 8.1.8. The random-pivot algorithm [8.1] can be very slow if there are many isolated nodes (i.e., many negative edges).

The good news is we can obtain a $(3+\varepsilon)-$*approximation* in logarithmic time, if we run a parallel version of the algorithm in a framework such as Pregel or Map-Reduce.
The algorithm is deeply discussed and analyzed in the paper Correlation clustering in MapReduce, by Chierichetti et al., here we just purpose a brief description.

```
parallel randomized pivot:
    while istance is not empty:
        - Δ+ ← max positive degree;
        - activate each element with probability ε/Δ+;
        - deactivate active nodes liked to other active nodes;
        - create a cluster for each remaining active node   // there are many pivots
          (break ties randomly);
```

Listing 8.2: Parallel Randomized Pivot

Theorem 8.1.6. Δ^+ *is halved after* $1/3 \cdot \log(n)$ *iterations.*

Observation 8.1.9. The probability $\dfrac{\varepsilon}{\Delta^+}$ is higher for nodes with higher degree, but those nodes have more neighbors that could deactivate them, so there is a balancing between these two aspects.

Open problem: How to handle the case where there are neither positive nor negative edges? I.e., in which, for some pairs of nodes, we don't know if the two elements are similar or dissimilar.

8.2 K-Centers

As we briefly saw at the beginning of this chapter, the goal of k-centers problem is to find the location of the k centers as to minimize the maximum distance between a point and a center. Now we'll define the problem more formally and we'll give an algorithm to approximate its solution.

Definition 8.2.1 (Minimum k-centers). *The minimum k-centers problem is defined as follows:*
Input: $X = \{x_1, x_2, ..., x_n\}$ *points in a metric space.*
Goal: Find $C \subseteq X$ *such that* $|C| = k$ *and* $\max_{x \in X} d(x, C)$ *is minimized, where* $d(x, C) := \min_{y \in C} d(x, y)$ *is the distance between a point and a set of points.*
Using the value of $d(x, C)$*, we can rewrite the goal as: Minimize* $\max_{x \in X} \min_{y \in C} d(x, y)$.

As we stated in the introduction [8.0.1], minimum k-centers is np-hard, so we look for an approximation. It has been proved that the best possible approximation one can obtain in polynomial time is a 2-approximation:

$$OPT \leq A(X, k) \leq 2OPT \quad \forall\ X, k.$$

Now we present the algorithm to solve the minimum k-centers problem, then we'll prove it gives a 2-approximation.

```
min-k-centers:
    C ← {any point}
    for i = 2, ..., k:
        let x be such that d(x, C) is maximized
        C ← C ∪ {x}
    return C
```

Listing 8.3: Minimum k-centers

Theorem 8.2.1. *Min-k-centers algorithm [8.3] returns a 2-approximation for the minimum k-centers clustering problem.*
In other words, $\forall\ x \in X,\ d(x, C) \leq 2 \cdot OPT$.

Proof. Let $C = \{c_1, c_2, \ldots, c_k\}$ be the set of centers chosen by the algorithm, where the indexes follows the order in which they are chosen.

Assume, by contradiction, that C is not a 2-approximation, then

$$\exists\ z \text{ s.t. } d(z, C) > 2 \cdot OPT. \tag{8.9}$$

If we prove that the following claim holds under this assumption, we will reach an absurd, so we'll prove the theorem.

Claim 8.2.1.1. $d(c_i, c_j) > 2 \cdot OPT\ \forall\ i, j$.

Proof of claim 8.2.1.1.

Observation 8.2.1. $\forall\ A \subset C,\ d(z, A) \geq d(z, C)$, since we have more possibilities to find a point closer to z in C than in A (since it's bigger).

We can use this observation and our assumption [8.9] to show that $d(z, \{c_1\}) \geq d(z, C) > 2 \cdot OPT$. Since the algorithm [8.3] pick as center the furthest point from those already considered, we also know that $d(c_2, \{c_1\}) \geq d(z, \{c_1\})$.

Similarly, we can say that $d(c_3, \{c_1, c_2\}) \geq d(z, \{c_1, c_2\}) \geq d(z, C) > 2 \cdot OPT$. The same holds for c_4: $d(c_4, \{c_1, c_2, c_3\}) \geq d(z, \{c_1, c_2, c_3\}) > 2 \cdot OPT$. And so on for each c_i up to c_k. □

Now we can go back to the proof of Theorem [8.2.1], and show that the claim [8.2.1.1], true under assumption [8.9], produces a contradiction.

Let $O = \{o_1, o_2, \ldots, o_k\}$. Any point in the dataset must be at distance at most OPT from a center in O, by definition of the problem; the same holds for the points in C, so let's pick c_1 from C and let's say that the closest point to c_1 in O is o_1 (it's just a name). Then c_2 can't be at distance $\leq OPT$ from o_1, otherwise we have $d(c_1, c_2) \leq 2 \cdot OPT$, but it's impossible for [8.2.1.1].

The same holds for any $c_i \in C$, thus each c_i is "covered" by a different o_i.

Then, z can't be covered by any o_i, which is a contradiction. □

8.3 K-Means

Definition 8.3.1 (Minimum k-means). *The minimum k-means problem is defined as follows:*
Input: $X \in \mathbb{R}^d = \{x_1, x_2, ..., x_n\}$ *points in a metric space.*
Goal: Find k *centers* $c_1, \ldots, c_k$ *in* $\mathbb{R}^d$ *(the euclidian space) s.t.* $\sum_{x \in X} d(x, C)^2$ *is minimized, where* $d(x, C) := \min_{y \in C} d(x, y)$ *is the distance between a point and a set of points.*

Observation 8.3.1. Differently from k-centers problem [8.2.1], k-means doesn't require that the centers are a subset of the points of the dataset.

Observation 8.3.2. As we stated in the introduction [8.0.1], minimum k-means is np-hard, so we can't actually find a solution for this problem. The following algorithm is frequently used for the *K-means* problem, even though it does not solve it.

1. Phase 1:
   1.1. Choose k centers UAR,
   1.2. Associate each point in X to the closest centers, thus creating clusters;
2. Phase 2:

2.1. Compute the center of mass (barycenter) of each cluster,

2.2. Assign each point to a barycenter to build new clusters,

2.3. Repeat 2.1 and 2.2 until clusters stay the same.

Formally:

```
min-k-means:
    arbitrarily pick c_1,...,c_k
    repeat:
        for i = 1 to k:
            C_i ← set of points in X closest to c_i
            c_i ← 1/|C_i| Σ_{x∈C_i} x // centers of mass, or barycenters
    until C_is and c_is do not change
    return C_1,...,C_k, c_1,...,c_k
```

Listing 8.4: Lloyd's algorithm

This algorithm is a special case of an *EM* algorithm:

Definition 8.3.2 (EM). *An* Expectation-Maximization *algorithm is an iterative method to find maximum likelihood or maximum a posteriori (MAP) estimates of parameters in statistical models, where the model depends on unobserved latent variables. The EM iteration alternates between performing an expectation (E) step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters, and a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the E step. These parameter-estimates are then used to determine the distribution of the latent variables in the next E step.*

Thus, `min-k-means` is sort of a greedy algorithm in two steps, where two different objectives are optimized.

Observation 8.3.3. The algorithm can take exponential time even in $\mathbb{R}^2$.

Observation 8.3.4. Solution can be *very far* from optimal.

Since the algorithm performs some kind of optimization on real numbers, we could wonder whether it will effectively succeed at finding some minimum.

Theorem 8.3.1. *The algorithm effectively finds a local minimum.*

Proof of theorem 8.3.1. To prove this we are going to need two lemmas.

Lemma 8.3.2. *After each iteration of the algorithm, the cost function decreases.*

Proof of lemma 8.3.2. Let's show the first solutions found by the algorithm and the relative costs.

$\mathscr{S}_0 = \{C_1^0, \dots, C_l^0; c_1^0, \dots, c_k^0\}$ is the first solution
$\mathscr{S}_1 = \{C_1^1, \dots, C_l^1; c_1^1, \dots, c_k^1\}$ is the second solution
$cost(\mathscr{S}_0) = \sum_{x \in X} d\left(x, \{c_1^0, \dots, c_k^0\}\right)^2$
$cost(\mathscr{S}_1) = \sum_{x \in X} d\left(x, \{c_1^1, \dots, c_k^1\}\right)^2$

We can see that the algorithm performs a greedy choice since it takes always the lowest cost.

Then, to prove the lemma, we use the following claim:

Claim 8.3.2.1. *Let $C_1, \dots, C_k$ be a partition of X in k sets, b_i be the centers of mass of the C_i, c_i be the centers around which the C_i were built, and $\mathscr{S} = \{C_1, \dots, C_k; c_1, \dots, c_k\}$, $\mathscr{S}' = \{C_1, \dots, C_k; b_1, \dots, b_k\}$. Then*

$$\begin{aligned} cost(\mathscr{S}) &= \sum_{x \in X} d\left(x, \{c_1^1, \dots, c_k^1\}\right)^2 \\ &\geq \sum_{x \in X} d\left(x, \{b_1^1, \dots, b_k^1\}\right)^2 = cost(\mathscr{S}') \end{aligned}$$

Proof of claim 8.3.2.1. Left as an exercise. □

Once we proved claim [8.3.2.1], lemma [8.3.2] follows directly. □

Lemma 8.3.3. *The algorithm terminates.*

Proof of lemma 8.3.3. Let's look at the algorithm from a combinatorial point of view.
The clusterization can be seen as a coloring where we are using k colors. Having k colors and n points there may be k^n colorings, so the number of possible clusterings is finite. □

Then we can finally go back to the proof of theorem [8.3.1].
What the algorithm is doing is a local search; since the cost function decreases at each iteration until it finds a minimum and the algorithm terminates, the algorithm effectively finds a local minimum. □

Despite its long running time and its bad approximation, observed in [8.3.3] and [8.3.4], the algorithm is much used, due to the following facts:

- The algorithm is simple.
- The algorithm's expected running time, computed with a smoothed analysis, is polynomial.

Let's thus see what a smoothed analysis is.

Definition 8.3.3 (Smoothed analysis). *In a smoothed analysis the running time is computed in the following way:*

1. *The input is adversary;*
2. *The datapoints are perturbed a little bit;*
3. *The expected running time is computed (expected because of the perturbation of the points, which is stochastic).*

The following modification of the Lloyd's algorithm [8.4] by Arthur-Vassilitski guaranties that the algorithm will perform well in any case.

```
new first step:
    pick c_1 ∈ X UAR
    C ← {c_1}
    for i = 2 to k:
        pick c ∈ X w.p. ∝ d(x, C)^2 // ∝ means "proportional to"
        C ← C ∪ {X}
```

Listing 8.5: kmeans++

That is, we can improve the algorithm just picking the first set of centers not uniformly at random but accordingly with a probability distribution that is proportional to the distance between x and C, formally:

$$\Pr\{\text{point } x \text{ is choosen as center}\} = \frac{d(x,C)^2}{\sum_{y \in X} d(x,C)^2}. \tag{8.10}$$

Claim 8.3.3.1. *After the first step, centers are a $O(\log(k))$-approx in expectation.*

Open problems:

- Researches tried to apply the same procedure of k-means++ to other problems that have similar objective function: the distance d with exponents $e \neq 2$. The result was that it works if the exponent is $e \geq 2$, but it is unknown if it could work if $1 \leq e < 2$;
- Since each solution depends on the previous one, the algorithms we saw can't be parallelized without getting a worse approximation, thus researchers are looking for a parallel algorithm that obtain the minimum loose in approximation;
- There are some algorithms that returns a constant approximation of k-means problem, but they aren't used since they are too slow or too hard to implement or use, so maybe some better algorithm can still be found.

Chapter 9

Network motifs

Graphs representing networks, including biological networks and social networks, contain wide variety of subgraphs. One important local property of networks are so-called *network motifs*, which are defined as recurrent and statistically significant sub-graphs or patterns.

Network motifs are sub-graphs that repeat themselves in a specific network or even among various networks. Each of these sub-graphs, defined by a particular pattern of interactions between vertices, may reflect a framework in which particular functions are achieved efficiently. Indeed, motifs are of notable importance largely because they may reflect functional properties.

Let's consider how motifs are used in social networks:

- To understand what is happening in a graph and build better algorithms;
- Classification: to find structural differences among graphs induced by different relations and measured counting the frequency of various motifs;
- Clustering: one can analyze the structure of a graph based on the cuts induced by the motifs instead of those induced by the edges (this approach turns out to be more accurate);
- Coordinate system: each subgraph (defined by some group or event on a social network, for example) becomes a point in the space based on the number of motifs.

Let's start by the simplest network motifs, the triangles.

9.1 Triangles

The triangle is, unsurprisingly, defined as a triple of nodes for which each of the three pairs of node is connected. We would like to compute the *global clustering coefficient*:

$$c(G) = \frac{3 \cdot (\#\text{ triangles in G})}{(\#\text{ connected triples in G})} = \frac{3\mathcal{T}}{w(G)}. \tag{9.1}$$

Let's compute this on an example.

In the graph shown in fig. 9.1 there are 5 connected triples and 1 triangle, so the global clustering coefficient would be 3/5.

Observation 9.1.1. The *g.c.c.* for a clique is 1, because every connected triple is also a triangle; while for a star it is 0 because there are no triangles.

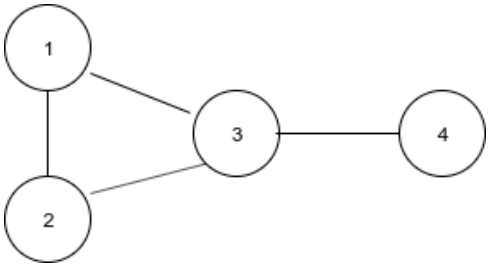
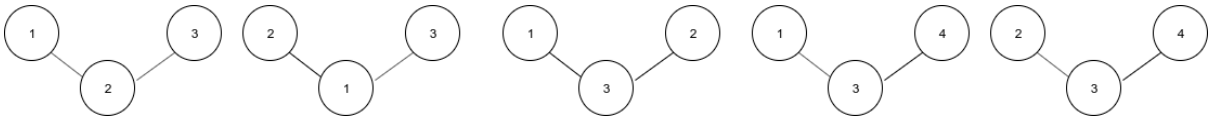


Figure 9.1: Graph example



To compute the denominator for the *g.c.c.*, that is $w(g)$, what we can do is sum all the pairs of nodes in the neighborhood of each node.

$$w(G) = \sum_{u \in V} \binom{deg(u)}{2}$$

taking time $T = O(N)$.
This gives us a first algorithm to compute the *g.c.c.*:

```
Algorithm_1:
    τ ← 0 // the number of triangles
    for u ∈ V:
        for {v, w} ∈ (N_u choose 2): // where N_u is the set of the neighbors of u
            if {v, w} ∈ E:
                τ += 1
    return τ/3
```

Listing 9.1: Algorithm 1

Claim 9.1.0.1. *Algorithm is in* $O(n^3)$.

Proof. First, assume that we have both the adjacency matrix that represents the graph and the degree of each node.
The complexity of the algorithm is given by $w(G)$, since we add at most 1 at each step, for each pair of nodes in $\binom{N_u}{2}$, thus, if the graph is dense – that is the degree for each node u is close to n^2 – we have that:

$$\sum_{u \in V} \binom{deg(u)}{2} = \sum_{u \in V} \Omega(n^2) = \Omega(n^3),$$

while, if the *max-degree* is constant, we have that:

$$\sum_{u \in V} \binom{deg(u)}{2} = \sum_{u \in V} O(1) = O(n).$$

□

Therefore, we would like to find other algorithms, or methods, to compute *g.c.c.* faster, even for dense graphs.

9.1.1 Adjacency Matrix method

One algorithm to compute the number of triangles is based on the adjacency matrix multiplication. The adjacency matrix for the graph seen in fig. 9.1 is the following

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Let's compute the square of it.

$$A^2 = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 3 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}$$

Now, what does the ij cell contain? By definition of matrix multiplication we have that

$$(A^2)_{uv} = \sum_{w \in G} A_{uw} \cdot A_{wv}$$

where $A_{uw} = 1$ iff $(u, w) \in E$ and $A_{wv} = 1$ iff $(w, v) \in E$. In other words, we are counting the number of paths of length 2 that go from u to w.
In cell A_{uu} there is the number of paths of length 2 that go from u to itself, that is equal to its degree because we can go from u to any of its neighbours in 1 step and then use the same edge to go back to u.
For A^3 we have that

$$(A^3)_{uv} = \left(A \cdot A^2\right)_{uv} = \sum_{w \in G} A_{uw} \cdot (A^2)_{wv},$$

thus, intuitively, $(A^3)_{uv}$ contains the number of paths of length 3 that go from u to v, and in general the same reasoning applies to any A^k for paths of length k.

To find triangles we can compute A^3 and, for every node, get its diagonal entry in the matrix, effectively giving us the number of paths of length 3 that go from that node to itself.

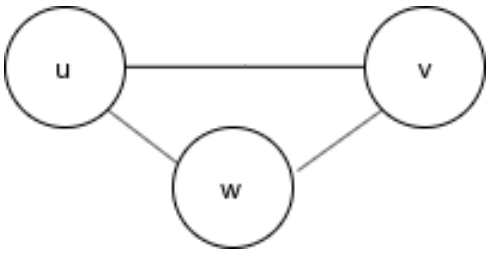


Figure 9.2: A triangle.

Let's compute $(A^3)_{uu}$ in fig. 9.2; there are two paths of length 3 that go from u to itself: uvw, uwv. So $(A^3)_{uu}$ would be 2, but the node is in just one triangle. That is true in general, so every triangle gives a contribute of 2 to the diagonal entry of every node that forms it. Furthermore, this way we will count the same triangle 3 times (1 for each vertex of the triangle).

Definition 9.1.1 (Trace of a matrix). *Let m be a matrix, its trace is*

$$tr(M) = \sum_i M_{ii}. \tag{9.2}$$

We thus have

$$\sum_{u \in G} \left(A^3\right)_{uu} = tr(A^3) = 6 \cdot \tau(G).$$

Theorem 9.1.1. *The number of triangles τ of a graph G is given by*

$$\tau(G) = \frac{tr(A^3)}{6}.$$

This gives us the following algorithm, that uses an algebraic approach based on the adjacency matrix it takes in input:

```
Algorithm_2(A):
    AC ← fast_mat_multiplication(A,3)
    return tr(AC)/6
```

Listing 9.2: Algorithm 2

Observation 9.1.2. In general, to compute A^i, you need $\log_2(i)$ matrix multiplications.

Observation 9.1.3. The execution time is given by the matrix multiplication, for which the best known algorithm takes time $O(n^{2.37})$, where n is the number of nodes. Therefore, the complexity of [9.2] is independent from the number of edges, that is, the algorithm doesn't take into account the structure of the graph.
On the one hand this approach gives an improvement for dense graphs, for which we pass from $O(n^3)$ to $O(n^{2.37})$, while on the other hand we can get a worsening for sparse graphs, such as for the star, for which we pass from $O(n^2)$ to $O(n^{2.37})$.
Note that this is true because for a sparse matrix A, its square A^2 isn't sparse; consider the A^2 for a star, it is dense since there are a lot of possible two-paths.

9.1.2 Structure-based method

Let's consider the star again, in algorithm [9.1] we loose a lot of time by making operations on the root node, that are useless, since we find the same "potential triangles" starting from the leaves.

Idea: If we sort the nodes by their degree, and we follow this ordering when we look for triangles, we won't count the triangles more times, and we'll have less operations to do when we reach the nodes with higher degree (since we'll have already considered most of their neighbors).

Example 9.1.1. The ordering of the nodes of graph in figure [9.1] is: *4, 1, 2, 3*, since the degrees of the nodes are, respectively, *1, 2, 2, 3*.

This gives us the following algorithm:

```
Algorithm_3(G):
    τ ← 0
    order(V) ← sort V so that u < v ⟺ deg(u) ≤ deg(v)
    for each u ∈ order(V):
        N_u^+ ← {v ∈ N_u | v > u}
        for every {v,w} ∈ (N_u^+ choose 2):
            if {v,w} ∈ E:
                τ += 1
```

Listing 9.3: Algorithm 3

Theorem 9.1.2. ***Algorithm_3*** *[9.3] returns exactly $\tau(G)$.*

Proof. Every triangle is counted exactly once by $w = \min\{x \in triangle\}$, that is, the vertex with minimum degree in the triangle. □

Theorem 9.1.3. *Let $\mathcal{E} = |E|$,* ***Algorithm_3*** *[9.3] runs in $O(\mathcal{E}^{3/2})$.*

Example 9.1.2. Let's consider the star again, with Algorithm_3 we need only $O(n^{1.5})$, instead of $O(n^2)$.

Observation 9.1.4. Since we add at most 1 triangle at each step, there will be at most $O(\mathcal{E}^{3/2})$ triangles in the graph, furthermore `Algorithm_3` [9.3] is able to return the whole list of triangles, along with their number.

Example 9.1.3. In a clique there are $\mathcal{E}^{3/2}$ triangles: $\mathcal{E} = \Theta(n^2),\ \tau(G) = \Theta(n^3) \implies \tau(G) = \Theta(\mathcal{E}^{3/2})$. Then we cant do better than this in the worst case, but in general social graphs have less edges and so we can obtain better performances.

Proof of theorem 9.1.3. We prove the theorem putting together some observations and lemmas.

Observation 9.1.5. First of all, consider that $\mathcal{E} = \frac{1}{2}\sum_{u \in G} deg(u)$, thus there are at most $2\sqrt{\mathcal{E}}$ nodes with degree $\geq \sqrt{\mathcal{E}}$.

Let $V_1 = \{v \in V \mid deg(v) \geq \sqrt{\mathcal{E}}\}$ and $V_2 = V - V_1 = \{v \in V \mid deg(v) < \sqrt{\mathcal{E}}\}$.
By observation [9.1.5] we know that $|V_1| \leq 2\sqrt{\mathcal{E}}$, then $E \geq \frac{1}{2}\sum_{u \in V_1} \sqrt{\mathcal{E}} = \frac{1}{2}|V_1|\sqrt{\mathcal{E}}$.

Observation 9.1.6.

$$T = \sum_{u \in V_2} \left| \binom{N_u^+}{2} \right| \leq \sum_{u \in V} \left|N_u^+\right|^2. \tag{9.3}$$

Lemma 9.1.4. $\left|N_u^+\right| \leq 2\sqrt{\mathcal{E}}\ \forall\ u \in V_1$.

Proof. Refer to figure [9.3].

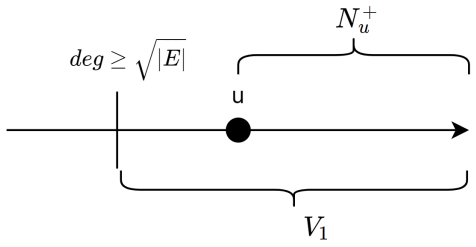


Figure 9.3: A visual proof of lemma [9.1.4]

□

Then we can define T_{V_1} and give it an upper bound (this will allow us to prove an upper bound for T, together with T_{V_2}, that we'll define later):

$$T_{V_1} = \sum_{u \in V_1} \left| \binom{N_u^+}{2} \right| \leq \sum_{u \in V_1} \left|N_u^+\right|^2 \leq 8\mathcal{E}^{3/2}. \tag{9.4}$$

In order to give a similar bound to T_{V_2}, we partition the nodes in V_2 in buckets as follows, according with their degree:
$V_2 = V_2^{(1)} \cup V_2^{(2)} \cup \ldots \cup V_2^{(i)} \cup \ldots \cup V_2^{(k)}$, where $V_2^{(1)} = \{v \mid v \in V \wedge \frac{\sqrt{\mathcal{E}}}{2} \leq deg(v) < \sqrt{\mathcal{E}}\}$, $V_2^{(2)} = \{v \mid v \in V \wedge \frac{\sqrt{\mathcal{E}}}{4} \leq deg(v) < \frac{\sqrt{\mathcal{E}}}{2}\}$, $V_2^{(i)} = \{v \mid v \in V \wedge \frac{\sqrt{\mathcal{E}}}{2^{i+1}} \leq deg(v) < \frac{\sqrt{\mathcal{E}}}{2^i}\}$, $V_2^{(k)} = \{v \mid v \in V \wedge 1 \leq deg(v) < 2\}$.
We know that $\left|V_2^{(1)}\right| \leq 4\sqrt{\mathcal{E}}$ by lemma [9.1.4], then we can compute $\left|V_2^{(2)}\right| \leq \frac{2\mathcal{E}}{\frac{\sqrt{\mathcal{E}}}{4}} = 8\sqrt{\mathcal{E}}$, $\left|V_2^{(i)}\right| \leq \frac{2\mathcal{E}}{\frac{\sqrt{\mathcal{E}}}{2^{i+1}}} = 2^{i+1}\sqrt{\mathcal{E}}$.

We can underline our last result in the following lemma:

Lemma 9.1.5.

$$\left|V_2^{(i)}\right| \leq 2^{i+1}\sqrt{\mathcal{E}}. \tag{9.5}$$

Lemma 9.1.6.

$$T_{V_2} = \sum_{u \in V_2} \left| \binom{N_u^+}{2} \right| \leq \sum_{u \in V_2} \left| N_u^+ \right|^2 \leq \sum_{u \in V_2} deg(u)^2. \tag{9.6}$$

Lemma 9.1.7. *For $u \in V_2^{(i)}$, `algorithm-3` [9.3] does at most $\mathcal{E}2^{-2i}$ steps.*

Thus, we can conclude the proof of theorem by giving an upper bound for T_{V_2}, that – together with the bound for T_{V_1} shown in [9.4] – will give us a bound for T [9.3]:

$$\begin{aligned}
T_{V_2} &= T_{V_2^{(1)}} + T_{V_2^{(2)}} + \ldots + T_{V_2^{(i)}} + \ldots + T_{V_2^{(k)}} \\
&= \sum_{i=1}^{\log_2 \sqrt{\mathcal{E}}} \left(\left| V_2^{(i)} \right| \cdot \left(\text{ work for } u \in V_2^{(i)} \right) \right) \\
&= \sum_{i=1}^{\log_2 \sqrt{\mathcal{E}}} \left(2^{i+2} \sqrt{\mathcal{E}} \cdot \mathcal{E} 2^{-2i} \right) \\
&\leq \mathcal{E}\sqrt{\mathcal{E}} \cdot \sum_{i=1}^{\log_2 \sqrt{\mathcal{E}}} 2^{-2i} \leq 4\mathcal{E}^{3/2}.
\end{aligned}$$

□

9.2 Motifs with more than three nodes

If we want to count the number of motifs with more than three nodes, it becomes increasingly difficult as the number of nodes we consider increases (for example, the problem of counting the number of cliques in a graph is NP-complete). For this reason, generally we look for an approximation of the optimal solution.

Definition 9.2.1. *Given a motif H on k nodes, let*

$$c_H = |\{U \subseteq V : G[U] \simeq H\}| \tag{9.7}$$

be the number of occurrences of the motif we are looking for in the graph.
Note that $G[U] \simeq H$ means "the subgraph on G induced by U is isomorphic to H".

Thus, our goal is to obtain an estimate $\hat{c}_H$ of c_H such that $c_H - t \leq \hat{c}_H \leq c_H + t$ with probability $1 - p$.
Note that this resembles what we've done with Randomized Pivot algorithm for Correlation clustering (see [8.1.1]).

Since it would take an exponential amount of time to pick each subgraph of k nodes and check if it is isomorphic to H, we'll use sampling (see [1.2.2]).

```
Algorithm_1(G, H):
    U ← {k nodes sampled UAR from (V choose k)};
    F ← G[U];
    return Y_H = 𝟙[F ≃ H];
```

Listing 9.4: Algorithm 1

Note that, to evaluate the characteristic function $\mathbb{1}$, that returns 1 if the argument is true and 0 otherwise, it takes only constant time, since we are considering only k nodes, and so we'll need to verify at most $k!$ permutations of the nodes in U.

Now let's analyze the expected value of the output of the algorithm:

$$E[Y_H] = \frac{\sum_{F \subseteq G, F \simeq H} 1}{\binom{|V|}{k}} = \frac{c_H}{\binom{n}{k}} = \sum_{F \subseteq G, F \simeq H} \frac{1}{\binom{n}{k}} \tag{9.8}$$

That is, the number of subgraphs of G that are isomorphic to H over the number of all the subrapgs of G of size k.

Observation 9.2.1. The expected value of the solution returned by Algorithm 1 [9.4] is linear with respect to c_H, but we would like to obtain an unbiased estimator of c_H, i.e., a random variable X_H such that $E[X_H] = c_H$. We can define it as follows:

$$X_H = \binom{n}{k} \cdot Y_H. \tag{9.9}$$

In this way, the expected value is what we want, but the variance is very high: a single sample gives us 0 or $\binom{n}{k}$, nothing very useful, and we can't pick $\binom{n}{k}$ samples.

Observation 9.2.2. Since H is connected, if we apply Algorithm 1 [9.4] on a sparse graph, we'll perform many useless samplings (we'll pick unconnected subgraphs with high probability). For example, the probability of peaking a k-motif on a star is k/n. Thus, we can slightly modify the algorithm as follows:

```
Algorithm_1b(G, H):
    U ← {k nodes sampled UAR from (V choose k)};
    F ← G[U];
    if F is not connected:
        return "FAIL";
    else:
        return Y_H = 𝟙[F ≃ H];
```

Listing 9.5: Algorithm 1b

But this doesn't produce a big improvement, so how can we pick k nodes in such a way that they are connected, and so they have the possibility to form the motif we are looking for? A possibility is to pick a random node, then choose one of its neighbors, then choose a neighbor of the selected neighbor, and so on. We'll describe an algorithm to do this for a special case: the 4-motif.

9.2.1 4-motifs

Let's consider different examples of motifs with 4 nodes, shown in figure [9.4].

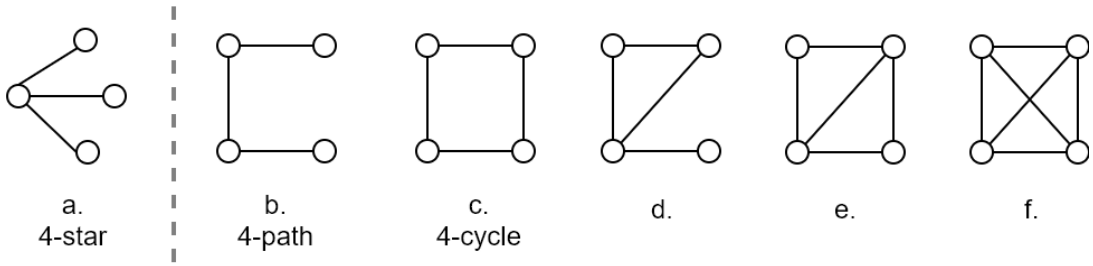


Figure 9.4: Examples of 4-motifs

Observation 9.2.3. The star (a.) can be considered a special case, since it doesn't contain any 4-path.

Observation 9.2.4. Considering 4-motifs from b. to f., if we use the procedure mentioned at the end of the previous section (*sample a random node, choose one of its neighbors, choose a neighbor of the selected neighbor, and so on*), these motifs have different probabilities to be found, since they have different number of 4-paths (increasing from b. to f.).

Let's describe an algorithm for finding the 4-paths (b.) in a graph:

```
Algorithm_2(G):
    choose u_1 UAR from V
    for i = 2,3,4:
        choose u_i UAR from N_{u_{i-1}} // N_x is the set of the neighbors of x
    return Y_H = 𝟙[F ≃ H];
```

Listing 9.6: Algorithm 2 (Path sample)

Let's study the expected value of the result of `Path sample`.
Let a, b, c, d be nodes in G that form a 4-path, note that the algorithm can pick the path $a \to b \to c \to d$ or the path $d \to c \to b \to a$.

$$E[Y_H] = \sum_{F \subseteq G, F \simeq H} \Pr\{\texttt{Path sample} \text{ hits } F\} = \tag{9.10}$$

$$= \sum_{F \subseteq G, F \simeq H} \left(\frac{1}{n} \cdot \frac{1}{deg(a)} \cdot \frac{1}{deg(b)} \cdot \frac{1}{deg(c)} + \frac{1}{n} \cdot \frac{1}{deg(d)} \cdot \frac{1}{deg(c)} \cdot \frac{1}{deg(b)} \right) \tag{9.11}$$

$$= \sum_{F \subseteq G, F \simeq H} \left(\frac{1}{n \cdot \deg(b) \cdot \deg(c)} \left(\frac{1}{deg(a)} + \frac{1}{deg(d)} \right) \right) \tag{9.12}$$

$$= \sum_{F \subseteq G, F \simeq H} \sigma_F \tag{9.13}$$

Observation 9.2.5. While the probability of hitting any node in `Algorithm 1` [9.4] was uniform, and so we had a sum of the same value for each F, the probability of sampling a node in `Path sample` [9.6] depends on the degrees of the previous picked nodes, except for the first chosen node, picked UAR. Thus, the probability of finding a certain path depends on the degrees of the nodes that compose it, so an isolated path, where each node has degree 1 or 2, has higher probability of being chosen than any other path.

To obtain an expected value of c_H, we need to remove this unbalancing, so we apply a compensation factor to Y_H, similarly to what we did in [9.9]:

$$X_H = \frac{1}{\sigma_F} \cdot Y_H. \tag{9.14}$$

Let $Y_F := \mathbb{1}[\texttt{Path sample} \text{ hits } F]$ (bernoullian random variable) and $X_F := Y_F \cdot \frac{1}{\sigma_F}$, then

$$\begin{aligned}
\mathrm{Var}(X_F) &= \mathrm{Var}(Y_F \cdot \frac{1}{\sigma_F}) = \\
&= \frac{1}{(\sigma_F)^2} \cdot \mathrm{Var}(Y_F) \\
&= \frac{1}{(\sigma_F)^2} \cdot (\sigma_F \cdot (1 - \sigma_F)) \\
&= \frac{1}{(\sigma_F)^2} \cdot \sigma_F - 1 \\
&\leq \frac{1}{\sigma_F} = O\left(n \cdot \Delta^3\right)
\end{aligned}$$

where Δ is the maximum degree of the graph. Thus, if the graph is sparse (the degree is constant), $\mathrm{Var}(X_F) = O(n)$.

At this point, it's important to note that there is a problem in the algorithm [9.6]: when we choose a node u_i, a neighbor of u_i, u_{i+1}, and we look for a neighbor of u_{i+1}, we could pick u_i again, thus picking a path with less than 4 nodes. But it isn't difficult to overcome this problem, for example by going back to the second step of the algorithm if F does not contain 4 nodes.
Thus, `Path sample` is a **rejection sampling** algorithm, i.e., one pick new elements at random and increment a counter if they match some criterion, otherwise the elements are thrown away.

Observation 9.2.6. A strength of the algorithm is that it actually produces the list of the motifs it founds, not only their number. This behavior is similar to that of `Algorithm 3` for triangles [9.3] but opposite to that of `Algorithm 2` for triangles [9.2].

Observation 9.2.7. For other kind of motifs, one can use the same algorithm, but the probability σ_F will change.

Chapter 10

Communication Complexity

In a distributed computing framework like MapReduce it is not only important to have computionally efficient algorithms but also not to overload the network with information. This kind of problem is often referred as *Communication Complexity*.

10.1 String equality

In the following we see an algorithm to compute strings equality.

```
X̄: bits string of computer A
Ȳ: bits string of computer B
Algo(X,Y):
    let S ⊆ [n] be the set of indexes sampled u.a.r.
    let computer A compute a = ⊕_{i∈S} x_i
    A sends a to B
    B computes b = ⊕_{i∈S} y_i
    if a = b then return EQUAL
    else return DIFF
```

Listing 10.1: Greedy algorithm

Observation 10.1.1. if $a = b$ the algorithm always returns $EQUAL$; if the algorithm returns $EQUAL$ this does not mean necessarily $a = b$. If the algorithm returns $DIFF$ then $a \neq b$.

As we will see, when a and b are different, the algorithm will return $DIFF$ with probability $\frac{1}{2}$. We can thus use the algorithm iteratively k times. If it never returns $DIFF$ we can say that $a = b$ with confidence $1 - (\frac{1}{2})^k$. The first time the algorithm returns $DIFF$ we can stop because $a \neq b$.

Theorem 10.1.1. *If $a \neq b$ the algorithm says $DIFF$ with probability* $\frac{1}{2}$

Proof. Let T be the set of indexes in which $\bar{X}$ and $\bar{Y}$ differ.

$$T = \{i | x_i \neq y_i\}$$

Let S' be the subset of indexes sampled *u.a.r* by the algorithm where $\bar{X}$ and $\bar{Y}$ differ.

$$S' = S \cap T$$

S' is *u.a.r.* because S is *u.a.r.* We know that A will compute $\bigoplus_{i \in S} x_i$, while B will compute $\bigoplus_{i \in S} y_i$.

$$a = \bigoplus_{i \in S} x_i \tag{10.1}$$

$$b = \bigoplus_{i \in S} y_i \tag{10.2}$$

Now, thanks to the commutative and distributive properties of the *xor*, we have that

$$\bigoplus_{i \in S} x_i = \left(\bigoplus_{i \in S'} x_i \right) \oplus \left(\bigoplus_{i \in S - S'} x_i \right) \tag{10.3}$$

$$\bigoplus_{i \in S} y_i = \left(\bigoplus_{i \in S'} y_i \right) \oplus \left(\bigoplus_{i \in S - S'} y_i \right) \tag{10.4}$$

We will now define z as the *xor* over all the indices in $S - S'$ of x_i. It is equal to the *xor* over all the same indices of y_i.

$$z = \bigoplus_{i \in S - S'} x_i = \bigoplus_{i \in S - S'} y_i \tag{10.5}$$

This is true because T is the set of indices where the two strings differ, so S' is the subset of S containing those indices where the two strings differ. It then holds that $S - S'$ will be the set of indices where the two strings are same.

$$\bigoplus_{i \in S} x_i = \bigoplus_{i \in S'} x_i \oplus z \tag{10.6}$$

$$\bigoplus_{i \in S} y_i = \bigoplus_{i \in S'} y_i \oplus z \tag{10.7}$$

We thus have that for a to be different from b, $\bigoplus_{i \in S'} x_i$ must be different from $\bigoplus_{i \in S'} y_i$.

$$\bigoplus_{i \in S} x_i \neq \bigoplus_{i \in S} y_i \iff \bigoplus_{i \in S'} x_i \neq \bigoplus_{i \in S'} y_i \tag{10.8}$$

Reminding that the *xor* of two bits is 1 when the two bits are different, we have that

$$\bigoplus_{i \in S'} x_i \neq \bigoplus_{i \in S'} y_i \iff \left(\bigoplus_{i \in S'} x_i \right) \oplus \left(\bigoplus_{i \in S'} y_i \right) = 1 \tag{10.9}$$

Furthermore, by distributive property, it holds that

$$\left(\bigoplus_{i \in S'} x_i \right) \oplus \left(\bigoplus_{i \in S'} y_i \right) = \bigoplus_{i \in S'} (x_i \oplus y_i) \tag{10.10}$$

According to the definition of S', the two strings differ in each $i \in S'$; it is then true that $x_i \oplus y_i$ must be equal to 1 for $i \in S'$.

$$\bigoplus_{i \in S'} (x_i \oplus y_i) = \bigoplus_{i \in S'} 1 \tag{10.11}$$

The *xor* of a binary string is 0 if the string is composed of a even number of 1s while it is 0 otherwise.

$$\bigoplus_{i \in S'} 1 = \begin{cases} 0 & if\ |S'|\ is\ even \\ 1 & otherwise \end{cases} \tag{10.12}$$

Summing up, we have that

$$Pr\left\{ \bigoplus_{i \in S} x_i \neq \bigoplus_{i \in S} y_i \right\} = Pr\left\{ |S'|\ is\ even \right\} = \begin{cases} 0 & if\ |T| = 0 \\ \frac{1}{2} & if\ |T| \geq 1 \end{cases} \tag{10.13}$$

□

We have shown what happens when $a \neq b$, now we are going to show that the algorithm will never make a mistake when the two strings are effectively equal.

Claim 10.1.1.1. *If $a = b$ the algorithm **always** returns EQUAL.*

Proof. If $a = b$ then $|T| = 0$; we then have that

$$Pr\left\{ \bigoplus_{i \in S} x_i = \bigoplus_{i \in S} y_i \right\} = 1 - Pr\left\{ |S'|\ is\ even \right\} = 1 - 0 = 1$$

□

10.1.1 Model

Let $V = \{1, \ldots, n\}$. Let $E \subseteq \binom{V}{2}$. Let V be partitioned among k disjoint parts $V_1, \ldots, V_k$ ($\bigcup_{i=1}^{k} V_i = V$ and $V_i \cap V_j = \varnothing$ for $1 \leq i < j \leq k$). We will mostly be interested in the case $k = n$.

There are k computers with unbounded memory and computational power. For each $i \in [k]$, computer i knows the set V_i and, for each $v \in V_i$, the set of neighbors of v, $N(v)$. More precisely, computer i has the following set as input:

$$\{(v, N(v)) \mid v \in V_i\}.$$

Computers can communicate in a *point-to-point* manner — that is, the generic computer $i \in [k]$ can send any string of bits $\bar{w} \in \{0,1\}^\star$ to the generic computer $j \in [k] - \{i\}$; the cost of sending a string is equal to its number of bits.

The goal of the $k + 1$ computers is to determine the connected components of G, while minimizing the cost — that is, while minimizing the number of transmitted bits.

10.1.2 A simple algorithm

First, observe that there exists the following trivial algorithm, if $|E| = m$.

Theorem 10.1.2. *The problem can be solved using $O(m \log n)$ bits of communication.*

Proof. For each $i \in \{2, \ldots, k\}$, let computer i sends its input to computer 1. Let computer 1 compute the components. The total number of bits is $O(m \log n)$. □

Theorem 10.1.3. *The problem can be solved using $O((k-1)n \log n)$ bits of communication.*

Proof. At the outset, let each node have a label equal to its identifier. Computation will proceed in iterations:

- in the first step of each iteration, each computer tells Computer 1 one (if any) couple of adjacent (in the graph) and distinct labels that it sees in its subgraph;
- in the second step of each iteration, the central computer selects arbitrarily a couple that it receives, say $x < y$, and tells to every computer that the label y has been changed to x;
- in the third and last step of each iteration, each computer changes the labels as dictated by the central computer (e.g., every label y is changed to x).

The first two steps of each iteration cost $O((k-1)\log n)$ bits of communication each. The third step is free.

The algorithm ends if no distinct adjacent labels are found in the first step of an iteration. When the algorithm ends, Computer 1 can easily guess the number of components: it will be equal to the number of remaining labels (that is: it will be equal to n minus the number of completed iterations).

The number of transmitted bits is then equal to $O((k-1)\log n)$ times the number of completed iterations. In each completed iteration, one label is removed. Since we start with n labels, the total cost is $O(n(k-1)\log n)$ bits. □

10.1.3 A $O\left(n^{3/2}\log^{1/2} n\right)$-bits algorithm

Let us assume, for simplicity, that the computers share a common random source — in particular, let us assume that they can all freely access a function $h : \binom{V}{2} \to \{0,1\}$ chosen uniformly at random in the set $\left\{g \mid g : \binom{V}{2} \to \{0,1\}\right\}$.

The algorithm [**?**] that we will describe can be made to work in the *simultaneous message* model (that is, the communication can end after a single round.)[1]

For simplicity of exposition, we assume that each computer holds exactly one node (so that $k = n$).

Randomized cut checking algorithm

We begin by describing an algorithm that can probabilistically check whether one or more sets of nodes leave some edges in their cuts.

Every node $v \in V$ computes

$$x(v) = \bigoplus_{e \in E | v \in e} h(e),$$

and sends it to the referee.

Algorithm 1: The Cut-Checking Algorithm.

Lemma 10.1.4. *Algorithm 1 uses n bits of communication. Moreover, for any $S \subseteq V$, the following holds. Let $y_S = \bigoplus_{v \in S} x(v)$; then:*

- *if there are no edges in the cut induced by S, then $y_S = 0$ with probability 1;*
- *if there is at least one edge in the cut induced by S, then $y_S = 1$ with probability $1/2$.*

Proof. The output of each application of the hash function h can be represented with 1 bit; the nodes only send n exclusive-ors of outputs of the hash function — thus the total bit cost is n.

Let $I(S) = \{e \mid e \subseteq S\}$ be the set of edges that are completely contained in S, and let $C(S) = \{e \mid |e \cap S| = 1\}$ be the set of edges in the cut $(S, V - S)$.

Then,

$$y_S = \bigoplus_{v \in S} \bigoplus_{\substack{e \in E \\ v \in e}} h(e) = \bigoplus_{e \in I(S)} (h(e) \oplus h(e)) \oplus \bigoplus_{e \in C(S)} h(e) = \bigoplus_{e \in C(S)} h(e).$$

Thus, if $C(S) = \varnothing$ (that is: if S has no edges in its cut) we will have $y_S = 0$ with probability 1.

Now suppose, instead, that $C(S) \neq \varnothing$; choose arbitrarily an edge $e^\star \in C(S)$. We can then write

$$y_S = h(e^\star) \oplus \bigoplus_{e \in C(S) - \{e^\star\}} h(e).$$

By the iid choice of the outputs of h we have that — regardless of $\bigoplus_{e \in C(S) - \{e^\star\}} h(e)$ — the random choice of $h(e^\star)$ will guarantee that y_S will be chosen uniformly at random in $\{0,1\}$. The proof is completed. □

[1] A result of Newman [**?**] can be used to translate any simultaneous message algorithm into a $O(1)$-rounds protocol, using private randomness and the same $O\left(n^{3/2}\log^{1/2} n\right)$ bits of communication. Thus, the assumption that the computers share a common random source can be avoided.

Components-Guessing with $O\left(n^{3/2}\log^{1/2} n\right)$ bits

We now give an algorithm that correctly identifies the components with probability $1-o(1)$ using $O\left(n^{3/2}\log^{1/2} n\right)$ bits of communication, in a single round, with a common source of randomness.

As usual, wlog, we assume that each computer holds exactly one node — and that there exists one (arbitrarily chosen) computer called the *leader* of the computation:

(i) each node v acts independently as follows: if $\deg(v) \leq \sqrt{n/\log n}$, then v (lets its computer) send a complete listing of all its incident edges to the leader; if, instead, $\deg(v) > \sqrt{n/\log n}$, then v (lets its computer) select a UAR sample of $\left\lfloor \sqrt{n/\log n} \right\rfloor$ of the edges incident on v, and sends them to the leader. In both cases, v tells $\deg(v)$ to the leader;

(ii) the leader reconstructs the partial graph H of the edges it received; observe that each component of H is a subcomponent of some component of the graph G; a *complete* component of H is a component containing only nodes whose degrees, as claimed by the computers that contain them, equal their degrees in H;

(iii) the leader runs the Cut-Checking algorithm for $2\sqrt{n\log n}$ times simultaneously on all the subsets of incomplete components[2];

(iv) claim that the minimal sets that were deemed to be correct by all the runs of the Cut-Checking algorithm are the components.

Theorem 10.1.5. *The above Components-Guessing algorithm outputs the correct partition into components with probability $1-o(1)$; the algorithm uses at most $O(n^{3/2}\log^{1/2} n)$ bits of communication.*

Proof. Observe that the algorithm will communicate $O(n^{3/2}\log^{1/2} n)$ bits; indeed, step (i) requires at most $O(n\sqrt{n\log n})$ bits; moreover, the $\Theta\left(\sqrt{n\log n}\right)$ runs of the Cut-Checking algorithm in step (iii) will send $\Theta(n)$ bits each. Step (ii) and (iv) will not send any bit.

Since each incomplete component has at least one node of degree not smaller than $\sqrt{n/\log n}$, each incomplete component contains more than $\sqrt{n/\log n}$ nodes — thus, there are at most $\frac{n}{\sqrt{n/\log n}} = \sqrt{n\log n}$ incomplete components. The number of subsets of incomplete components is then at most $2^{\sqrt{n\log n}}$; the probability that any given set, that has one or more edges in its cuts, passes all the runs of the Cut-Checking algorithm is at most $2^{-2\sqrt{n\log n}}$ — therefore, by the union bound, the probability that some set that strictly includes a connected component passes all the tests is at most

$$2^{\sqrt{n\log n}} \cdot 2^{-2\sqrt{n\log n}} = 2^{-\sqrt{n\log n}} = o(1).$$

The claim is proved. □

[2]Observe that there exist up to $2^{O(n)}$ such subsets — so, this step can require exponential time. We will not care about this particular aspect, because we are only interested in the number of bits exchanged by the computers. Note, though, that this step can be performed in polynomial time using the Gaussian elimination algorithm.