Notes on Social and Behavioral Networks

Cristian Di Pietrantonio

November 22, 2017

Contents

Chapter 1

Meme flow

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:

> 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 a mathematical model that describes the spread of information. We call *node* an entity on the web that holds the information of interest; it can be a blog, a website or a social network profile. Nodes can receive or take information from other nodes, enabling the spread of the meme. Nodes 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 some 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.

1.1 The Trace Spreading Model

Authors in [3] introduce the Trace Spreading Model, where a *trace* is a sequence of nodes ordered by their timestamp.

Define an undirected graph $G = (V, E)$, where:

- V is the set of nodes that holds the information of interest;
- There is only a source of information, chosen uniformly at random;
- $e = \{v, v'\} \in E$ iif information propagated from v to v', $v, v' \in V$ (the actual direction is not important, as you will notice later).

- Each edge $e = \{v, v'\}$ has a *weight* $w(e)$ sampled i.i.d. in $Exp(\lambda)$, which represent the time needed for the information to propagate from v to v'. In fact, for our purpose, any distribution will do (i.i.d. is the important part).

In this model, the trace follows the shortest path tree from the source.

So, having this model and a set of traces (and every trace contains all the nodes), is it possible to reconstruct the unknown social graph G?

1.2 The First-Edge Algorithm

Authors in [1] shows that we can reconstruct (in most cases) the graph $G = (V, E)$ with high probability with a simple algorithm.

Input: T, the set of traces
$E \leftarrow \{\}$ //the set of edges in the graph ;
for $t \in T$ **do**
  $u \leftarrow t[0]$ //the first node in t;
  $v \leftarrow t[1]$ //the second node in t;
  $E \leftarrow E \cup \{(u, v)\}$;
end

Algorithm 1: The first-edge algorithm.

Consider Algorithm 1. What it 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.

Theorem 1.1. *If there are $|T| \geq cn\Delta \log n$ traces, the first-edge algorithm can reconstruct the unknown graph $G = (V, E)$ with probability $p \geq 1 - \frac{1}{n^{2(c-1)}}$, where $n = |V|$, $\Delta = \max_v deg(v)$ and c is a constant.*

Proof. An edge $\{u, v\}$ will be in the graph if and only if u and v appears at the beginning of at least one trace. We will prove that it happens with high probability.

Consider the following event.

$$\xi_1 = \text{“A given edge } \{u, v\} \in E \text{ is inferred using the trace } t.\text{”}$$

The probability of ξ_1 is given by

$$\begin{aligned} Pr\{\xi_1\} &= Pr\{t[0] = x\} Pr\{t[1] = y, y \in \{u, v\} \setminus \{x\} \mid t[0] = x\},\ x \in \{u, v\} && (1.1)\\ &= \frac{2}{n} \frac{1}{deg(x)}, x \in \{u, v\} && (1.2)\\ &\geq \frac{2}{n\Delta}. && (1.3) \end{aligned}$$

In other words it is the probability of one of the two nodes u, v to appear as first node in the trace multiplied by the probability that the other appears as second node in the trace. Since the source is selected uniformly at random, every node has the same probability $\frac{1}{n}$ to be picked; for us, either u or v will do. Next, assume without loss of generality (wlog) that u is picked as first node. The second node will now be drawn from the neighbors of u (convince yourself that there can't be a node y after the first node in the trace without it being one of its neighbors). The probability that the neighbor picked is u is $\frac{1}{deg(u)}$ that can be overestimated using Δ.

Now, the complementary event of ξ_1 is "the edge $\{u, v\}$ is not inferred using trace t" and its probability is $(1 - \frac{2}{n\Delta})$. For the edge not to be inferred at all, there should be no trace from which that edge can be extracted.

$$Pr\{\textit{An edge in the graph is not inferred}\} = \left(1 - \frac{2}{n\Delta}\right)^{|T|} \leq \left(1 - \frac{2}{n\Delta}\right)^{cn\Delta \log n} \tag{1.4}$$

We will show that this probability is tiny. To do so, we must use the following lemma.

Lemma 1.1.

$$1 - x \leq e^{-x} \tag{1.5}$$

Proof. Studying the function we see that

$$e^{-x} = (-e^{-x})' = (e^{-x})'' > 0$$

so its tangent in every point is below the function. Taking the tangent in $x_0 = 0$ we have

$$\begin{aligned} e^{-x} &\geq (e^{-x_0})'x + e^{-x_0} \\ &= -e^{-x_0} + 1 \\ &= -x + 1 \\ &= 1 - x. \end{aligned}$$

□

Going back to our proof we have

$$\begin{aligned} Pr\{\textit{Edge } \{u, v\} \textit{ in the graph is not inferred}\} &= \left(1 - \frac{2}{n\Delta}\right)^{|T|} && (1.6) \\ &\leq \left(1 - \frac{2}{n\Delta}\right)^{cn\Delta \log n} && (1.7) \\ &\leq \left(e^{-\frac{2}{n\Delta}}\right)^{cn\Delta \log n} && (1.8) \\ &= e^{\log n^{-2c}} && (1.9) \\ &= \frac{1}{n^{2c}}. && (1.10) \end{aligned}$$

Finally, we want to compute the probability that the algorithm fails. Let

$$\xi_{a,b} = \text{“}\textit{Edge } \{a,\ b\} \textit{ is not inferred by the algorithm”}.$$

Then

$$\begin{aligned}
Pr\{\textit{The algorithm fails}\} &= Pr\left\{ \bigcup_{\{a,b\}\in E} \xi_{a,b} \right\} && (1.11)\\
&\leq \sum_{\{a,b\}\in E} Pr\{\xi_{a,b}\} && \text{(by union bound)}\\
&= |E| Pr\{\xi_{a,b}\} && (1.12)\\
&\leq n^2 \frac{1}{n^{2c}} && (1.13)\\
&= \frac{1}{n^{2(c-1)}}. && (1.14)
\end{aligned}$$

Notice that we can make this probability tiny as we want by increasing c, i.e. by increasing the number of traces. □

Chapter 2

Chernoff Bound

In this Chapter we see a way to bound tail distributions, that is, the probability that a random variable assumes values that are far from its expectation.

2.1 First-Edge Algorithm, revised

In the last chapter, we studied the First-Edge Algorithm for reconstructing the social graph given a set of traces. The way we proved the algorithm to be valid can be abstracted in the following way. For each trace i, and a given edge $\{u, v\} \in E(G)$, define

$$X_i = \begin{cases} 1 & \{u, v\} \text{ are not the first two nodes in trace } i \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

We considered only a given edge and we computed the probability that the nodes involved in it were never at the beginning of a trace. The reason we wanted to do that was that if $\{u, v\}$ were the first two nodes in a trace then the edge was indeed in the graph.

What we were really asking for is the following:

$$Pr\left\{\prod_i X_i = 1\right\}. \tag{2.2}$$

If all X_is are one then no trace had $\{u, v\}$ as first nodes. Otherwise, the product is zero and the edge is in the graph. Essentially we studied the product of random variables.

When there are a bunch of independent variables and one wants to compute, say, when one of them is different from the others, then the analysis is usually pretty simple. In most cases, though, is not so easy. Usually what is studied is the random variable X defined as

$$X = \sum_i X_i, \tag{2.3}$$

that is, the sum of the X_is as opposite to the product. In most cases, people study this random variable, defining if from a set of variables, which may be independent, and with the aim to compute statistics like the expectation, the second moment, or the probability that the X deviate from its expectation by more than a certain amount. If X is always (or with high probability) close to its expectation we say the variable is *concentrated*. This specific random variable is ubiquitous in algorithms and random processes. We will start looking at this type of variable and its use in modeling social network processes.

2.2 Chernoff bound

We state (without proof) the Chernoff Bound, which is a powerful tool to bound a tail distribution with exponentially decreasing values.

Theorem 2.1 (Chernoff Bound)**.** *Let $X_1, X_2, \ldots, X_n$, $0 \leq X_i \leq 1, \forall i$, be a set of mutually independent random variables, and $X = \sum_{i=1}^{n} X_i$. Then*

$$\Pr\{X \leq (1-\varepsilon)E[X]\} \leq e^{-\frac{\varepsilon^2 E[X]}{3}} \tag{2.4}$$

and

$$\Pr\{X \geq (1+\varepsilon)E[X]\} \leq e^{-\frac{\varepsilon^2 E[X]}{3}}. \tag{2.5}$$

Consider the following situation. We flip n fair coins. Define

$$X_i = \begin{cases} 1 \textit{ if the i-th flip results in heads} \\ 0 \textit{ otherwise} \end{cases}$$

then

$$X = \sum_{i=1}^{n} = \textit{ \# of heads.}$$

What is the probability of the event $\{X = k\}$? For X to be k there must be exactly k coins which resulted in heads. We can choose among $\binom{n}{k}$ compatible events (set of variables), each having the same probability 2^{-n} to happen. It follows that

$$Pr\{X = k\} = \binom{n}{k} 2^{-n}.$$

What if the coin is not fair? We have the binomial distribution.

$$Pr\{X = k\} = \binom{n}{k} p^k (1-p)^{n-k}. \tag{2.6}$$

We would like to know what is the probability that X assumes a value far from the expected one. We can use the Chernoff Bound.

$$Pr\{X \leq (1-\varepsilon)np\} \leq e^{-\frac{\varepsilon^2 pn}{3}}, \tag{2.7}$$

The quantity is going down exponentially with n. Why is this useful? ϵ can go down to $\frac{1}{\sqrt{(n)}}$ and we would still get something that is going down to 0 very quickly. Suppose we choose $p = \frac{1}{2}$ and

$$\varepsilon = \sqrt{\frac{\log n}{n}} \tag{2.8}$$

Then we have

$$e^{-\frac{\frac{\log n}{n}\frac{1}{2}n}{3}} = e^{-\frac{1}{6}\log n} = \frac{1}{\sqrt[6]{n}}.$$

That is, the probability of error can be made as tiny as we want by increasing n.

The Chernoff bound is an extremely important tool that is used, as an example, in all applications that deal with large datasets. This is because it states that it is not necessary to look at all the values to get a sense about what the mean is.

Consider the following scenarion. There are a bunch of people that has to vote of one candidate or the other. A first attempt to predict the winner could be to ask to every person for his/her preference. Then we one would have all the necessary information, but it cannot be done in practice. A more realistic way of doing it is, because of the existence of the Chernoff bound, to pick random people. If there is some gap between the two candidates, maybe 51 vs. 49 or more, then after maybe few cases (maybe 2000 people in this example) one would be able to tell the candidate that is more likely to win. The error may be proportional to $\sqrt{n}$ but, as long as the gap is linear in n, the prediction can be considered valid.

2.3 Chain letters

Chain letters are (were) online petitions that spread virally through emails. They work in the following way. The petition's creator writes an email explaining a certain problem he or she wants to address. Then he/she signs the petition, sends it to a set of recipients and ask them to sign and forward the email. Some recipients may ignore the email, others may sign and forward it instead. One can model the spread using a tree T, where the root is the petition's creator and every node has a child for each person he forwarded the email to. Of course, email are privates. For an external observer to know the petition, at least one person must post the email on a public accessible place on the Web (e.g. public mailing lists). When a node v (person) *exposes* itself (i.e. publish the email), all its ancestors in the tree are revealed too (the path from the root to v).

We would like to know some information about the spread process, such as the reach of the petition. The full tree T is in general unknown, and what can be observed of it depends on how many and which nodes get exposed. We can assume that there exists a small probability $\delta > 0$ such that every node $v \in T$ gets exposed independently with that probability. Define T_δ to be the visible part of T reconstructed though the exposed nodes. Let V be the set of nodes in T_δ. Define $E \subseteq V$ to be the set of nodes that exposed themselves by disclosing their email to the public. Finally, let $L \subseteq E$ be the set of leaves in T_δ.

As we said, we would like to know $|V| = n$, but we only have E. Can we estimate n? If we assume that each node in E exposed itself independently from the others, we can make a reasonable guess about n by looking at T_δ [2]. In fact

$$n\delta = |E| \tag{2.9}$$

so

$$n = \frac{|E|}{\delta}. \tag{2.10}$$

At this point, we need to find δ' such that $\delta' \approx \delta$. One can wrongly think that

$$\delta' = \frac{|E|}{|T_\delta|}. \tag{2.11}$$

The problem here are the leaves of T_δ. If they hadn't exposed themselves, we wouldn't know about them (or other nodes). From the point of view of T_δ, they are *not* exposed independently with probability δ'; rather, they are exposed with probability 1. To compute δ' then, we exclude T_δ's leaves. What follows is an algorithm that outputs δ'.

Input: E, L, V.
if $|V| = 0$ **then**
  return 0
end
if $|V| = 1$ **then**
  return 1
end
return $\frac{|E|-|L|}{|V|-|L|}$

Algorithm 2: Estimation of δ

Finally, if $|V| \geq 2$,

$$n \approx \frac{|E|}{\delta'}. \tag{2.12}$$

Lemma 2.1. *If* $|V| \geq 2$ *then*

$$\Pr\left\{|\delta' - \delta| \geq \epsilon\delta\right\} \leq 2e^{\frac{1}{3}\epsilon^2\delta|V-L|}. \tag{2.13}$$

Proof. For each node i, let X_i be a random variable such that

$$X_i = \begin{cases} 1 \text{ if node } i \text{ exposes itself} \\ 0 \text{ otherwise.} \end{cases} \tag{2.14}$$

Consider the three sets A, B and C built in the following way. We can scan the original tree in a bottom up fashion. At a certain time, node i is considered. If $i \notin A$, check if the node was exposed. If $X_i = 1$, add i to B and every its ancestor to A;

otherwise, add i to C. Observe that the tripartition process does not disclose the value of X_i for all $i \in A$, and that $A = V - L$ and thus the set $E - L$ coincides with the set of exposed nodes in A. Then, we can apply the Chernoff bound, having $X = |E - L|$.

$$
\begin{aligned}
Pr\{X \geq (1+\epsilon)\delta|A|\} &= Pr\{|E-L| \geq |A|\delta + \epsilon\delta|A|\} && (2.15)\\
&= Pr\left\{\frac{|E-L|}{|A|} \geq \delta + \epsilon\delta\right\} && (2.16)\\
&= Pr\left\{|\delta' - \delta| \geq \epsilon\delta\right\} \quad \text{(We want the difference small enough)}\\
&= Pr\{|X - |A|\delta| \geq \epsilon\delta|A|\} && (2.17)\\
&\leq 2e^{-\frac{1}{3}\epsilon^2 E[X]} && (2.18)\\
&= 2e^{-\frac{1}{3}\epsilon^2\delta|V-L|}. && (2.19)
\end{aligned}
$$

□

Chapter 3

Submodular functions

In this chapter we first look at the k-max cover problem. Then, we will se that this problem is part of a sequence of problems having the same structure and that can be solved essentially with the same algorithm. This class of problems is the one of submodular function optimization problems.

3.1 k-max cover

Given $k \geq 1$ and a set of sets S, find a $S_k \subseteq S$, $|S_k| = k$, such that $|\bigcup_{s \in S_k}|$ is maximum. There is a similar problem, the set-cover problem, where we want to cover all the elements with the minimum number of sets. Here we have a budget of k sets, and we want to cover as many elements as possible. The following algorithm gives a greedy solution (in fact, in approximation algorithms there are not so many techniques you can use).

Input: S, k
$X_0 \leftarrow \emptyset$;
for $i = 1, \ldots, k$ **do**
  $s \leftarrow \arg\max_{s \in S} |X_{i-1} \cup s|$;
  $X_i \leftarrow X_{i-1} \cup s$
end
return X_k

Algorithm 3: A greedy algorithm for k-max cover.

Theorem 3.1. *Algorithm 3 returns a* $1 - \frac{1}{e}$ *approximation of the optimal solution.*

Proof. Define OPT to be the value of the optimal solution. Let $t_i = |X_i|$. The value of the greedy solution is going to be t_k. Define

$$n_i = |s_i - X_{i-1}|, \tag{3.1}$$

the number of new elements introduced at iteration i. Finally, define

$$g_i = OPT - t_i, \tag{3.2}$$

that indicates how far the greedy solution is far from the optimal one at iteration i. We want to prove

$$g_k \leq \frac{1}{e} \iff t_k \geq \left(1 - \frac{1}{e}\right) OPT \tag{3.3}$$

Lemma 3.1.

$$n_{i+1} \geq \frac{g_i}{k} \tag{3.4}$$

In words, it looks at how far it is from the optimal solution at iteration i and when it picks the $(i+1)$-th set it gets at least $\frac{1}{k}$ of what it was missing in the previous iteration.

Proof. Let $\{s_1^\star, s_2^\star, \ldots, s_k^\star\}$ be the sets representing the optimal solution and $O^\star$ their union. Let $T \subseteq O^\star$, maybe $T = O^\star - X_i$, for any i.

$$T \subseteq \bigcup_{i=1}^{k} s_i^\star \implies \exists i \ |s_i^\star \cap T| \geq \frac{|T|}{k} \tag{3.5}$$

The fact that $O^\star$ is realized by the choice of k sets that represents the optimal solution implies the existence of at least one set in $O^\star$ that covers at least a $\frac{1}{k}$ fraction of T. This is because if all sets in $O^\star$ covered less than that fraction, then overall they could not cover T and therefore they couldn't cover $O^\star$.

There exists one set in the optimal solution that covers at least $\frac{1}{k}$ fraction of T. Even for the specific T suggested above. But if this is true, then it means that there exists one set in S that covers at least a $\frac{1}{k}$ fraction of $O^\star - X_i = T$, and Algorithm 3 is going to pick at iteration $i+1$ the one set that maximizes the number of new elements it picks. So the new set Algorithm 3 will choose, will have at least as many elements as this one set in the optimal solution, that is $\frac{g_i}{k}$. □

Lemma 3.2.

$$g_i \leq \left(1 - \frac{1}{k}\right)^i OPT, \ \forall i. \tag{3.6}$$

Proof. We prove this by induction. For $i = 0$, $g_0 = OPT$ and so $g_0 \leq (1) OPT$ it is true. Let's assume Lemma 3.2 is true until i and we want to prove it for $i+1$.

$$
\begin{aligned}
g_{i+1} &= g_i - n_{i+1} && (3.7)\\
&\leq g_i - \frac{g_i}{k} && \text{(By Lemma 3.1)}\\
&= g_i \left(1 - \frac{1}{k}\right) && (3.8)\\
&\leq OPT \left(1 - \frac{1}{k}\right) && (g_i \leq OPT)\\
&< OPT \left(1 - \frac{1}{k}\right) \left(1 - \frac{1}{k}\right)^i && (3.9)\\
&= OPT \left(1 - \frac{1}{k}\right)^{i+1}. && (3.10)
\end{aligned}
$$

□

Instantiating Lemma 3.2 with $i = k$ we have

$$g_k \leq \left(1 - \frac{1}{k}\right)^k OPT \leq \frac{1}{e} OPT \tag{3.11}$$

□

In the next section we define what a submodular function is and then why k-max cover is a submodular optimization problem. We will discover that we can get a $(1 - \frac{1}{e})$ approximation for any problem in this class.

3.2 Submodular functions

Given a *ground set* V, a function $f : 2^V \to \mathbb{R}$ is a set function. We would like to maximize the value of this kind of functions, but if we have no information about them, the only thing we can do is to compute f for every set.

Suppose now we know about some of f's structure. The function f is *modular* (or *linear*) if, given some set $A \in 2^V$,

$$f(A) = \sum_{i \in A} w(i), \tag{3.12}$$

where $w(i)$ is the value of element i. The function 3.12 can be maximized easily by simply looking at every one-element set ($n = |V|$ queries or computations) and return the set which contains all the elements with positive weight.

Suppose now that we want to maximize the value of the function under the constraint that the set picked must have cardinality k. We choose the k biggest values. These problems are easy with modular functions, and hard with general set functions.

Definition 3.1 (Submodular function)**.** *Submodular functions are more general than linear functions and are less general than set functions. A function f is submodular if*

$$\forall S, T \subseteq V \;\; f(S) + f(T) \geq f(S \cup T) + f(S \cap T). \tag{3.13}$$

They are a generalization of modular functions. In fact, one can prove that if f is modular then Equation 3.13 holds only with equality.

Let $S = A$, $B \subseteq A$, $x \notin A$, $T = B \cup \{x\}$. Let's apply the submodularity definition

$$f(A) + f(B \cup \{x\}) = f(A \cup \{X\}) + f(B) \tag{3.14}$$

that implies

$$f(B \cup \{x\}) - f(B) \geq f(A \cup \{x\}) - f(A). \tag{3.15}$$

Definition 3.2 (Discrete derivative)**.** *Given a set function $f : 2^V \to \mathbb{R}$, $A \subseteq V$, $x \in V$, we define the* discrete derivative *of f at A with respect to x as*

$$\Delta(x|A) = f(A \cup \{x\}) - f(A). \tag{3.16}$$

In economics, Equation 3.16 is called *marginal return*. Putting together equations 3.15 and 3.16 we get the economics Law of diminishing returns.

Definition 3.3 (Marginal returns property)**.** *Given $B \subseteq A \subseteq V$ and an item $x \in V \setminus B$, adding x to B increase the value of that set more than adding that element to a bigger set A which includes B. Formally*

$$\Delta(x|B) \geq \Delta(x|A). \tag{3.17}$$

We proved that Definition 3.1 implies Definition 3.3. We now prove the opposite direction, to show that the two definitions are equivalent.

Claim 3.1. *Definition 3.3 implies Definition 3.1*

Proof. Let $S, T \subseteq V$ be given. There are two cases, let's look at the simple one first. Assume $S \subseteq T$, then $S \cup T = T$ and $S \cap T = S$. The same conditions of Definition 3.1 are present, so we can just go backwards on the steps we performed to get to Definition 3.3.

So assume $S \not\subseteq T$ and $T \not\subseteq S$. This means that $S - T = \{s_1, \ldots, s_k\}$ is not empty.

Lemma 3.3.

$$\forall 1 \leq i \leq k \;\; f((S \cap T) \cup \{s_1, \ldots, s_i\}) - f(S \cap T) \geq f(T \cup \{s_1, \ldots, s_i\}) - f(T) \tag{3.18}$$

Proof. If $i = 1$ then we get the marginal returns property. The following inequality shows the base step of the induction.

$$f((S \cap T) \cup \{s_1\}) - f(S \cap T) \geq f(T \cup \{s_1\}) - f(T) \tag{3.19}$$

This lemma is all about applying marginal returns property to longer and longer stripes of elements.

What we want to do in the next step is adding s_2. So we have

$$f((S \cap T) \cup \{s_1, s_2\}) - f((S \cap T) \cup \{s_1\}) \geq f(T \cup \{s_1, \dots, s_2\}) - f(T \cup \{s_1\}) \quad (3.20)$$

This is true by the marginal returns property. In fact, all this inequalities will be true up until

$$f((S \cap T) \cup \{s_1, \dots, s_{i+1}\}) - f((S \cap T) \cup \{s_1, \dots, s_i\}) \geq \\ f(T \cup \{s_1, \dots, s_{i+1}\}) - f(T \cup \{s_1, \dots, s_i\}) \quad (3.21)$$

We are assuming that these inequalities hold by induction. Now we want to add up them, and we see that the terms cancel out. We are left with

$$f((S \cap T) \cup \{s_1, \dots, s_{i+1}\}) - f(S \cap T) \geq f(T \cup \{s_1, \dots, s_{i+1}\}) - f(T) \quad (3.22)$$

And so the claim is true. □

Now observe what happens at the last step.

$$\begin{aligned} &f((S \cap T) \cup \{s_1, \dots, s_k\}) - f(S \cap T) \geq f(T \cup \{s_1, \dots, s_k\}) - f(T) \\ &f((S \cap T) \cup (S - T)) - f(S \cap T) \geq f(T \cup (S - T)) - f(T) \\ &f(S) - f(S \cap T) \geq f(T \cup S) - f(T) \\ &f(S) + f(T) \geq f(S \cap T) + f(S \cup T) \end{aligned}$$

And we have proved the equivalence between the two definitions. □

Claim 3.2. *If f and g are submodular, then $f + g$ is submodular.*

Proof. $\forall S, T \subseteq V$

$$f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$$

and

$$g(S) + g(T) \geq g(S \cup T) + g(S \cap T)$$

Define $h(S) = f(S) + g(S)$ and then sum the inequalities. The result is still submodular (the inequality holds). □

If f is submodular, $-f$ is submodular? Only when the property holds at equality (hence, we don't switch sign).

Claim 3.3. *If f is submodular, αf is submodular, $\forall \alpha \geq 0$.*

Claim 3.4. *If f is submodular and g is modular then $f - \alpha g$ is submodular.*

Proof. g is modular, so multiplying it for a negative number results in a modular function, which is also submodular. Finally, the sum of two submodular function is submodular. □

3.3 Facility location

Now we are going to give an example of submodular function optimization problem. Let $\mathcal{F} = \{F_1, \ldots, F_n\}$ be a set of facilities and $\mathcal{U} = \{U_1, \ldots, U_m\}$ a set of users. Maybe $\mathcal{F}$ is the set of universities in Rome and $\mathcal{U}$ the set of (wannabe) students in Rome. Student U_i gets a gain $M_{ij} \geq 0$ for starting a program at University F_j. So what a student would do? He would go to the facility that gives the maximum gain. If that's the protocol, it is all very easy. But suppose you are the government and you can't open all the n universities, but only k of them. We have to select $S = \{i_1, \ldots, i_k : U_{i_k} \in \mathcal{U}\}$ facilities so to maximize $\sum_{i=1}^{m} \max_{j \in S} M_{ij}$.

What is the function we are going to prove to be submodular?

$$f(S) = \sum_{i=1}^{m} \max_{j \in S} M_{ij} \tag{3.23}$$

The function is pretty intricate, but we are going to use the properties we just saw to prove it being submodular. Thanks to Claim 3.2, we can focus only on proving

$$f_i(S) = \max_{j \in S} M_{ij} \tag{3.24}$$

to be submodular.

Claim 3.5. $\forall S, T \subseteq \mathcal{F}$

$$f_i(S) + f_i(T) \geq f_i(S \cup T) + f_i(S \cap T). \tag{3.25}$$

Proof. Let's concentrate on $f_i(S \cup T)$. Observe that, $f_i(S \cup T)$ is going to be larger or at least as large as the other f_is in the inequality 3.25 because f_i is just a max over the elements of the set. Suppose the maximum value of $f_i(S \cup T)$ is realized by some $j^\star$. This element can be in S, T, or both.

$$j^\star \in S \implies f_i(S) = f_i(S \cup T) \tag{3.26}$$

$$j^\star \in T \implies f_i(T) = f_i(S \cup T) \tag{3.27}$$

We are trying to prove that the right hand side is no larger than the left hand side. We have seen that $f_i(S \cup T)$ is the biggest part in the right hand side, and we have shown that, no matter what, there will be one element in the LHS that has the same

value. Now we have to show that the other part of the RHS is not larger than the other element in the LHS.

Suppose $k^\star$ achieves the maximum at $f_i(S \cap T)$. What do we know? $k^\star$ is going to be in both S and T. So the max at S is going to be at least as large as the max obtained with $k^\star$. It follows that

$$f_i(S) \geq f_i(S \cap T) \tag{3.28}$$

and

$$f_i(T) \geq f_i(S \cap T) \tag{3.29}$$

so

$$\min(f_i(T), f_i(S)) \geq f_i(S \cap T). \tag{3.30}$$

□

At this point we can state that K-MAX-COVER is FACILITY LOCATION with $M_{ij} \in \{0, 1\}$. Think of users as elements $E_1, \ldots, E_m$ and facilities as sets $S_1, \ldots, S_n$.

$$M_{ij} = \begin{cases} 1 \text{ if } E_i \in S_j \\ 0 \text{ otherwise.} \end{cases} \tag{3.31}$$

Here, $f_i(S)$ is going to be the maximum across all the sets in the solution S and is going to be 1 if at least one set in S contains element i; zero otherwise. Finally, we are summing up over all the elements we picked.

3.4 Approximation of submodular functions

We are going to see an algorithm that returns a $\left(1 - \frac{1}{e}\right)$ approximation of the optimal solution for a submodular function under a cardinality constraint. It is going to be similar to the one seen for k-max cover.

Input: f, k
$S_0 \leftarrow \emptyset$;
for $j = 1, \ldots, k$ **do**
  Let $v_i \in ([n] - S_{i-1})$ be the element maximizing $\Delta(v_i | S_{i-1})$;
  $S_i \leftarrow S_{i-1} \cup \{v_i\}$;
end
return S_k;

Algorithm 4: Greedy algorithm to approximate submodular functions.

Authors in [4] proved the following theorem.

Theorem 3.2. *If f is submodular, non-negative and monotone then Algorithm 4 returns a $\left(1 - \frac{1}{e}\right)$ approximation of $\max_{S \in \binom{[n]}{k}} f(S)$.*

Proof. Let $S^\star = \{v_1^\star, \dots, v_k^\star\} \in \binom{[n]}{k}$ be the optimal solution.

Lemma 3.4.

$$\forall i \leq k-1 \; f(S^\star) \leq f(S_i) + k\overbrace{(f(S_{i+1}) - f(S_i))}^{new\ step\ gain} \tag{3.32}$$

Proof. We look at the gain obtained by adding a new element at step i. In the following lines we are *telescoping* the first line to get intermediate steps.

$$\begin{aligned}
f(S^\star) &\leq f(S^\star \cup S_i) && \text{(monotonicity)} \\
&= f(S^\star \cup S_i) - f(\{v_1^\star, \dots, v_{k-1}^\star\} \cup S_i) \\
&+ f(\{v_1^\star, \dots, v_{k-1}^\star\} \cup S_i) - f(\{v_1^\star, \dots, v_{k-2}^\star\} \cup S_i) \\
&+ f(\{v_1^\star, \dots, v_{k-2}^\star\} \cup S_i) - \dots \\
&\dots \\
&+ f(\{v_1^\star, v_2^\star\} \cup S_i) - f(\{v_1^\star\} \cup S_i) \\
&+ f(\{v_1^\star\} \cup S_i) - f(S_i) + f(S_i)
\end{aligned}$$

On each row we can see a diminishing return:

$$\begin{aligned}
&\Delta(v_k^\star | S_i \cup \{v_1^\star, \dots, v_{k-1}^\star\}) \\
&+ \Delta(v_{k-1}^\star | S_i \cup \{v_1^\star, \dots, v_{k-2}^\star\}) \\
&+ \dots \\
&+ \Delta(v_2^\star | S_i \cup \{v_1^\star\}) \\
&+ \Delta(v_1^\star | S_i) \\
&+ f(S_i)
\end{aligned}$$

So we have

$$\begin{aligned}
f(S^\star) &\leq f(S^\star \cup S_i) && \text{(monotonicity)} \\
&= f(S_i) + \sum_{j=1}^{k} \Delta(v_j^\star | S_i \cup \{v_1^\star, \dots, v_{j-1}^\star\}) \\
&\leq f(S_i) + \sum_{j=1}^{k} \Delta(v_j^\star | S_i) && \text{(diminishing returns)} \\
&\leq f(S_i) + \sum_{j=1}^{k} \Delta(v_{i+1} | S_i) && \text{(greedy choice)} \\
&= f(S_i) + k\Delta(v_{i+1} | S_i) \\
&= f(S_i) + k(f(S_{i+1}) - f(S_i))
\end{aligned}$$

□

Define $\delta_i = f(S^\star) - f(S_i)$ and also, for simplicity, $\delta_{i+1} = f(S^\star) - f(S_{i+1})$. Remember that we want to claim we don't lose so much in going from one step to the next. By looking at $f(S_{i+1}) - f(S_i)$ we can't say much since we don't know the function f. However, let's look at the difference between δ_i and δ_{i+1}; essentially we used the diminishing return property to forget about the function. Now we can just think about the costs with respect to the optimum that we pay at each step.

$$\begin{aligned}
\delta_i &= f(S^\star) - f(S_i) && (3.33)\\
&\leq k(f(S_{i+1}) - f(S_i)) && \text{(by Lemma 3.4)}\\
&= k(\delta_i - \delta_{i+1}) && (3.34)
\end{aligned}$$

Now we want to show that the cost goes down.

$$k\delta_{i+1} \leq (k-1)\delta_i \implies \delta_{i+1} \leq \left(1 - \frac{1}{k}\right)\delta_i \tag{3.35}$$

So

$$\begin{aligned}
&\delta_0 = f(S^\star) - f(S_0) \leq f(S^\star) && \text{(f non-negative)}\\
&\delta_1 \leq \left(1 - \frac{1}{k}\right)\delta_0 = \left(1 - \frac{1}{k}\right) f(S^\star) && (3.36)\\
&\dots && (3.37)\\
&\delta_k \leq \left(1 - \frac{1}{k}\right)\delta_{k-1} \leq, \dots, \leq \left(1 - \frac{1}{k}\right)^k \delta_0 && (3.38)\\
&\quad = \left(1 - \frac{1}{k}\right)^k f(S^\star) \leq \frac{1}{e} f(S^\star). && (3.39)
\end{aligned}$$

□

This can be though as an example of how one can start by solving a specific problem, creating a solution, and then think of it as a general solution. One can asks what the created algorithm, or the analysis for proving it works, can say about a more general problem.

Do we need the properties we assumed, i.e. submodularity, non-negativity and monotonicity?

So, non-negativity is not that important. For one thing, if the function can be negative, what does it mean to optimize it and to give a constant approximation? Maybe the optimal is negative, in which case, say, a $\left(1 - \frac{1}{e}\right)$ approximation would be impossible. In the context of maximization, multiplying the optimal (which we assume negative) for a constant $0 < x < 1$ gives a greater value than the optimal (contradiction). Thus, multiplying by a constant can change whether you are maximizing or minimizing. For what we have just said, non-negativity is necessary. On the other hand, you could remove

that assumption keeping, in this particular case, a bound with $f(S^\star) - f(S_0)$. It won't yield a constant approximation, because it doesn't make sense with negative functions, but you would get something that depends on this gap. Essentially, non-negativity is used to get a nice, clean theorem.

What about monotonicity? That actually helps. If it removed, then what we can get is a $\frac{1}{2}$ approximation, under some computational assumptions, but not a $\left(1 - \frac{1}{e}\right)$ approximation. It would require a different way of reasoning about the problem and a much more complicated algorithm. So removing it means getting a worst approximation. Finally, we don't want to remove submodularity, we started from there.

Other questions are about constraints. Why do we care about optimizing the value of the function at sets of cardinality k? One could say "I care about other sets". If the function is monotone and we want to maximize it without any constraint on the set, what should I do? Return the full set. Monotonicity is actually very important if the optimization problem comes with some constraint. As example, what if we do not want exactly k elements but at most k elements. In this case not having monotonicity is an important issue (maybe the empty set is the one that has more value).

3.5 Learning a submodular function

Algorithm 4 is able to approximate the value of the function by looking at $O(kn)$ values instead of $\Theta(2^n)$. It performs a number of steps that is logarithmic in the size of the input. In cases when there is no need to look at all the input, the algorithm is said to be sublinear.

We saw that linear (modular) functions can be understood by only looking at singleton sets (plus the empty set), that is, $n + 1$ values. It is natural to think that we can't do the same for submodular functions, and what follows is the idea to prove it.

What is a construction that can actually prove that, while there is an algorithm that efficiently optimize the function, there is no algorithm that can learn the function efficiently? As usual we want to use easy building blocks, and then mix up them according to the claims we saw earlier (summing them up, essentially).

For $T \subseteq [n]$, define the function

$$f_T(S) = \begin{cases} 0 \ \textit{if } S \subseteq T, \\ 1 \ \textit{otherwise.} \end{cases} \tag{3.40}$$

The function f_T is a building block that we want to use to build the final submodular function F. Consider

$$\mathcal{S} \subseteq U = \binom{[n]}{\frac{n}{2}} = \left\{ Q : Q \subseteq [n] \wedge |Q| = \frac{n}{2} \right\}$$

F will depend on $\mathcal{S}$, but the fact we don't know actually what $\mathcal{S}$ is will lead us to

the result we want to prove. So, let's define

$$F_{\mathcal{S}}(A) = \sum_{T \in \mathcal{S}} f_T(A). \tag{3.41}$$

There are exponentially many functions f_T, which are all the possible building blocks. We are looking for a subset of them, and one subset is be picked without it being disclosed; now, the value of the submodular function F is the sum of the values of the functions at a particular set.

Claim 3.6. *f_T is submodular.*

Proof. Let's pick two sets $A, B \subseteq [n]$, and apply the definition of submodularity.

$$f_T(A) + f_T(B) \geq f_T(A \cup B) + f_T(A \cap B). \tag{3.42}$$

There are four cases to be considered.

1. $A, B \subseteq T$. Then we have $0 + 0 \geq 0 + 0$, which holds.
2. $A \subseteq T$ and $B \not\subseteq T$ (and the specular case). Computing the function values we have that $0 + 1 \geq 0 + 1$.
3. $A, B \not\subseteq T$. In this case we have $1 + 1 \geq 0 + x$. We can't tell exactly the value x of the function computed at intersection, but it can be at most 1, so the inequality holds.

□

Since the sum of submodular functions is submodular, $F_{\mathcal{S}}(A)$ is submodular. We want to prove that knowing F requires a lot of queries. Doing it formally would require many definitions, but it is a pretty easy algorithm. Your algorithm's call is to learn F by querying it. The answer it gets at each query is some integer representing the number of functions f_T that evaluates to 1 at set A.

How big and how small that number can be? The function F is nonnegative, so the minimum is 0 and the maximum is the cardinality of S, which can be as large as 2^n. $F_S(A)$ is an integer from 0 to 2^n, and thus requires n bits for each query. We will show the that the number of bits needed to represent F is much larger, exponential. Why is it the case? How big is the set of sets U?

$$|U| = \binom{n}{\frac{n}{2}} = \Theta\left(\frac{2^n}{\sqrt{n}}\right). \tag{3.43}$$

Since we are picking any subset $\mathcal{S}$ of this big set of sets, to be able to know which subset $\mathcal{S}$ is, we actually have to know for each set in U whether i picked it or not. One bit is needed for each set, summing up to $|U|$ bits. At each query we get only n bits, therefore at least $\frac{|U|}{n}$ queries are needed (i.e., an exponential number).

Why do we need to know exactly S? Well, if we don't know S, the value of F can't be known.

This is why learning submodular functions is hard. But we can optimize them with efficient algorithms, doing leverage on certain properties as we saw.

Chapter 4

Random Graph Models

In this chapter we are going to take a look at random graph models developed to represent properties of social networks. The first one is the Preferential Attachment model, a very famous one, introduced to explain the *power law* that characterize the degree distribution in big social graphs. The other one justifies the fact that between any two people there is a minimal path with really few "hops", fact known as *small world phenomenon*. One thing that we anticipate is that is really difficult to find a single model that contains all the social networks properties; instead researchers were been able to produce algorithms that in general won't work, but on special cases they perform really very well.

4.1 Preferential Attachment Model

It has been observed that the degree distribution in big social graphs, like the one of Facebook or Twitter, is really close to a *power law* distribution.

Definition 4.1. *A probability distribution P follows a power law if*

$$P(x) = x^{-\alpha}, \alpha > 1. \tag{4.1}$$

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".

It turns out that the Preferential Attachment Model is able to capture this social networks' aspect, whereas others random models like $G(n, p)$ (the one proposed by Edgar Gilbert) do not show this regularity. In $G(n, p)$, for example, we expect each node to have the degree very close to the mean, because of the Chernoff Bound.

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.

If we plot a power law distribution on a *loglog plot*, that is, a plot where on both axes are represent the log of coordinates of each point, we get a straight line that says

how many nodes for each degree one can get, which is kind of unusual to observe in real world phenomena, thought to be more "chaotic". A first scientific question one can ask is how can it be that something so striking.

We would rather expect that there exists some average degree from which values would deviate with a an exponential law, resulting in very small deviations from the mean; something similar to what the Chernoff bound states. And yet is not what happens here. In understanding why this the case, the Preferential Attachment Model is really important. It was rediscovered by many people. In physics, this model is usually attributed to Barabási and Albert, but also discovered in mathematical biology, and in math. It is called also called "rich get richer" model, and we will see why.

First of all, it differs from $G(n,p)$ because it is evolutionary, not fixed. It is possible to stop at n nodes if needed, but it is not required. There are many variants. In our discussion, the model starts with one node with a self loop. Then, at each time step t, a new node is added, which then is linked to an existing node in the graph. The new node will choose its neighbor in some random way. In the simplest model it chooses one neighbor, but it can be generalized to k neighbors (this is not considered here).The neighbor is selected with a *rich get richer* policy. This means that a node is chosen with a probability proportional to its degree. Richer nodes, the ones with high degree, have a greater probability to been chosen as neighbor of a new node.

Formally, given the graph G_{t-1}, the probability of a node $v \in V(G_{t-1})$ to be chosen at time t to be the neighbor of the new node is

$$P(v) = \frac{deg(v)}{\sum_{v' \in V(G_{t-1})} deg(v')} \tag{4.2}$$

It follows that the probability distribution may change every time a new node is added. It is evolutionary, and we can't make and independent choice for each edge. When we are in such a situation, studying the graph is hard, in fact there are many things we don't know about this model. But it is possible to study the degree distribution and to show that it is actually a power law distribution.

At fist glance, assuming a graph with n nodes, it seems that to add the $(n+1)$-th node one has to look at the full graph. This means that, if $n = 1000$, then we have to look at 1000 nodes to decide where to put the edge. The first observation is, if we only care about the degree distribution, we can reduce the knowledge about the graph that we care about; in particular, if we know the vector of degree counts at time $t-1$ then it is possible compute the new vector of degrees at time t. This is the main trick we will be using.

Let's define some notation. Let

$$X_i^{(t)} = \textit{"Number of nodes with degree } i \textit{ in } G_t\textit{"}. \tag{4.3}$$

The following properties are always true (we start with one node with a self loop):

$$X_2^{(1)} = 1,\ X_i^1 = 0\ \forall i\ i \neq 2. \tag{4.4}$$

$$|E(G_t)| = t,\ |V(G_t)| = t \tag{4.5}$$

Furthermore. the sum of all degrees in the graph is twice the number of edges.

Can we compute the expected number of nodes with degree, say, one? In the end, we want to be able to know the expected number of nodes of any degree, but now let's focus on the base case. That number depends on the past, since the process is evolutionary, which we can represent as the vector $\overline{X^{(t-1)}}$ containing the degree distribution at $t-1$ time step. We first assume a given realization of the vector $\overline{X^{(t-1)}}$.

Define

$$\xi_1 = \textit{“The new node chooses a degree } 1 \textit{ neighbor”}. \tag{4.6}$$

Then, when adding a new node, either ξ_1 or its complementary will happen. In the first case, the number of nodes stays the same, since we add and remove one; in the second case, we increase the count by one.

$$\begin{aligned}
E\left[X_1^{(t)}|\overline{X_1^{(t-1)}}\right] &= (X_1^{(t-1)}+1)(1-Pr\{\xi_1\}) + X_1^{(t-1)}Pr\{\xi_1\} \\
&= (X_1^{(t-1)}+1)\left(1-\frac{X_1^{(t-1)}}{2(t-1)}\right) + X_1^{(t-1)}\frac{X_1^{(t-1)}}{2(t-1)} \\
&= X_1^{(t-1)} - \frac{(X_1^{(t-1)})^2}{2(t-1)} + 1 - \frac{X_1^{(t-1)}}{2(t-1)} + \frac{(X_1^{(t-1)})^2}{2(t-1)} \\
&= \left(1-\frac{1}{2(t-1)}\right)X_1^{(t-1)} + 1.
\end{aligned}$$

So now, by the law of total probability,

$$\begin{aligned}
E\left[X_1^{(t)}\right] &= \sum_{c\geq 0} Pr\left\{X_1^{(t-1)}=c\right\} E\left[X_1^{(t)}|X_1^{(t-1)}=c\right] \\
&= \sum_{c\geq 0} Pr\left\{X_1^{(t-1)}=c\right\}\left(\left(1-\frac{1}{2(t-1)}\right)c+1\right) \\
&= \sum_{c\geq 0} Pr\left\{X_1^{(t-1)}=c\right\}\left(1-\frac{1}{2(t-1)}\right)c + \sum_{c\geq 0} Pr\left\{X_1^{(t-1)}=c\right\} \\
&= \left(1-\frac{1}{2(t-1)}\right)\sum_{c\geq 0} Pr\left\{X_1^{(t-1)}=c\right\}c + 1 \\
&= \left(1-\frac{1}{2(t-1)}\right)E\left[X_1^{(t-1)}\right] + 1.
\end{aligned}$$

Lemma 4.1.

$$E\left[X_1^{(t)}\right] = \left(1-\frac{1}{2(t-1)}\right)E\left[X_1^{(t-1)}\right] + 1. \tag{4.7}$$

What we get is a recursive formula to compute the expected number of nodes with degree one. The "plus one" is the new node we add, and the others nodes with degree one will almost be preserved except for the $\frac{1}{2(t-1)}$ fraction we might end up losing because of a one degree node removal (by attaching the new edge to it).

How can we hope to do the same thing for degrees larger than one? In this case, the difference is that the new node won't have degree i, so what could happen? The number of nodes having degree i might not be affected, or can increase or decrease by one. If, for example, we cared about nodes with degree three, and the new one attaches to one of such nodes, then the count decreases. If the new node attaches to a previously degree two node, then the count increases. Formally,

$$\begin{aligned}
E\left[X_i^{(t)}|\overline{X^{(t-1)}}\right] &= X_i^{(t-1)} + Pr\left\{X_i^{(t)} - X_i^{(t-1)} = 1\right\} - Pr\left\{X_i^{(t)} - X_i^{(t-1)} = -1\right\} \\
&= X_i^{(t-1)} + \frac{i-1}{2(t-1)}X_{i-1}^{(t-1)} - \frac{i}{2(t-1)}X_i^{(t-1)} \\
&= \left(1 - \frac{i}{2(t-1)}\right)X_i^{(t-1)} + \frac{i-1}{2(t-1)}X_{i-1}^{(t-1)} \\
&= E\left[X_i^{(t)}|X_i^{(t-1)}, X_{i-1}^{(t-1)}\right].
\end{aligned}$$

It follows that

$$\begin{aligned}
E\left[X_i^{(t)}\right] &= \sum_{c,d\geq 0} Pr\left\{X_i^{(t-1)} = c \wedge X_{i-1}^{(t-1)} = d\right\} E\left[X_i^{(t)}|X_i^{(t-1)} = c, X_{i-1}^{(t-1)} = d\right] \\
&= \sum_{c,d\geq 0} Pr\left\{X_i^{(t-1)} = c \wedge X_{i-1}^{(t-1)} = d\right\} \left(\left(1 - \frac{i}{2(t-1)}\right)c + \frac{i-1}{2(t-1)}d\right) \\
&= \sum_{c\geq 0} Pr\left\{X_i^{(t-1)} = c\right\} \left(1 - \frac{i}{2(t-1)}\right)c + \sum_{d\geq 0} Pr\left\{X_{i-1}^{(t-1)} = d\right\} \frac{i-1}{2(t-1)}d \\
&= \left(1 - \frac{i}{2(t-1)}\right)E\left[X_i^{(t-1)}\right] + \frac{i-1}{2(t-1)}E\left[X_{i-1}^{(t-1)}\right].
\end{aligned}$$

Lemma 4.2.

$$\forall i\ i \geq 2,\ E\left[X_i^{(t)}\right] = \left(1 - \frac{i}{2(t-1)}\right)E\left[X_i^{(t-1)}\right] + \frac{i-1}{2(t-1)}E\left[X_{i-1}^{(t-1)}\right]. \tag{4.8}$$

Our original goal was to prove that

$$E\left[X_i^{(t)}\right] \propto i^{-\alpha}t \tag{4.9}$$

that is, a power law behavior. This is a very rough estimate but it is representative of what we would like to achieve; in other words, to bound the expectation with the

number of steps we had so far. Currently we cannot achieve this because we can express the expectation at time t only as a function of the expectation at the previous time step.

We need to solve recurrent linear systems (Lemma 4.1, Lemma 4.2). Lemma 4.1 is easy, since it has only one free variable; Lemma 4.2 is not as easy. It ia actually pretty much complicated, and exists general, hard, techniques to solve it. In such situations, the best thing to do is to guess the solution and check that it works. Still, if one wants to solve the recursion, you can evaluate the base case (Lemma 4.1), and then plug the solution into the general case (Lemma 4.2) iteratively.

We will give a solution to bound $E\left[X_i^{(t)}\right]$ and check if it is correct.

Theorem 4.1. *Let* $E_i = \frac{4}{i(i+1)(i+2)}$ *be our guess on the expectation* $E\left[X_i^{(t)}\right]$.

$$E_i t - 1 \leq E\left[X_i^{(t)}\right] \leq E_i t + 1 \tag{4.10}$$

It is not the exact solution, but it is within a constant from it, and since E_i grows with t, the constant factor doesn't matter in practice.

Proof. As said previously hinted, we will do a double induction to prove the theorem. The outer one is on i, and the inner one on t. So, assume $i = 1$, the base case of the outer induction. Let's proceed with the inner one.

Since $i = 1$, the graph has only one node with a self loop, by construction. Why 1 was chosen as error constant? We started from one node with degree two. If you think about, we could have started from any other graph, but we had to change the error factor.

In the inner base case, $t = 1$. Then

$$E\left[X_1^{(1)}\right] = 0 \tag{4.11}$$

and

$$\frac{2}{3} - 1 \leq E\left[X_1^{(1)}\right] \leq \frac{2}{3} + 1. \tag{4.12}$$

Actually, Inequality 4.12 is true for any i. Now we assume that Equation 4.10, with $i = 1$, is true up until $t - 1$. Let's prove that it is true also at time t. We prove the upper bound first.

$$E\left[X_1^{(t)}\right] = \left(1 - \frac{1}{2(t-1)}\right) E\left[X_1^{(t-1)}\right] \tag{4.13}$$

$$\leq \left(1 - \frac{1}{2(t-1)}\right)\left(\frac{4}{6}(t-1)+1\right)+1 \qquad \text{(By inductive hypotesis)}$$

$$= \frac{2(t-1)}{3} - \frac{2(t-1)}{6(t-1)} - \frac{1}{2(t-1)} + 1 \tag{4.14}$$

$$= \frac{2(t-1)}{3} + \left(1 - \frac{1}{3}\right) + \left(1 - \frac{1}{2(t-1)}\right) \tag{4.15}$$

$$= \frac{2}{3}t + \left(1 - \frac{1}{2(t-1)}\right) \leq \frac{2}{3}t + 1. \tag{4.16}$$

Now, the lower bound.

$$E\left[X_1^{(t)}\right] = \left(1 - \frac{1}{2(t-1)}\right) E\left[X_1^{(t-1)}\right] \tag{4.17}$$

$$\geq \left(1 - \frac{1}{2(t-1)}\right)\left(\frac{4}{6}(t-1)-1\right)+1 \qquad \text{(By inductive hypotesis)}$$

$$= \frac{2(t-1)}{3} - \frac{2(t-1)}{6(t-1)} + \frac{1}{2(t-1)} + 1 \tag{4.18}$$

$$= \frac{2}{3}t - \frac{2}{3} - \frac{1}{3} + \frac{1}{2(t-1)} + 1 \tag{4.19}$$

$$= \frac{2}{3}t + \frac{1}{2(t-1)} \geq \frac{2}{3}t - 1. \tag{4.20}$$

The inner induction is terminated. We have proved for the base case $i = 1$. Assume Equation 4.10 is true up until $i - 1$, for any t. We what to show that it is true also for the number of nodes with degree i. Let's now prove the upper bound.

$$E\left[X_i^{(t)}\right] = \left(1 - \frac{i}{2(t-1)}\right) E\left[X_i^{(t-1)}\right] + \frac{i-1}{2(t-1)} E\left[X_{i-1}^{(t-1)}\right] \tag{4.21}$$

$$\leq \left(1 - \frac{i}{2(t-1)}\right) (E_i(t-1)+1) + \frac{i-1}{2(t-1)}(E_{i-1}(t-1)+1) \tag{4.22}$$

$$= \left(1 - \frac{i}{2(t-1)}\right) \left(\frac{4(t-1)}{i(i+1)(i+2)} + 1\right) \tag{4.23}$$

$$+ \frac{i-1}{2(t-1)} \left(\frac{4(t-1)}{(i-1)(i)(i+1)} + 1\right) \tag{4.24}$$

$$= \frac{4(t-1)}{i(i+1)(i+2)} + 1 - \frac{2}{(i+1)(i+2)} - \frac{i}{2(t-1)} + \frac{2}{i(i+1)} + \frac{i-1}{2(t-1)} \tag{4.25}$$

$$= \frac{4(t-1)}{i(i+1)(i+2)} + 1 + \frac{4}{i(i+1)(i+2)} - \frac{1}{2(t-1)} \tag{4.26}$$

$$= \frac{4t}{i(i+1)(i+2)} + 1 - \frac{1}{2(t-1)} \leq E_i t + 1. \tag{4.27}$$

The lower bound proof have essentially the same steps. We show it for completeness.

$$E\left[X_i^{(t)}\right] = \left(1 - \frac{i}{2(t-1)}\right) E\left[X_i^{(t-1)}\right] + \frac{i-1}{2(t-1)} E\left[X_{i-1}^{(t-1)}\right] \tag{4.28}$$

$$\geq \left(1 - \frac{i}{2(t-1)}\right) (E_i(t-1)-1) + \frac{i-1}{2(t-1)}(E_{i-1}(t-1)-1) \tag{4.29}$$

$$= \left(1 - \frac{i}{2(t-1)}\right) \left(\frac{4(t-1)}{i(i+1)(i+2)} - 1\right) \tag{4.30}$$

$$+ \frac{i-1}{2(t-1)} \left(\frac{4(t-1)}{(i-1)(i)(i+1)} - 1\right) \tag{4.31}$$

$$= \frac{4(t-1)}{i(i+1)(i+2)} - 1 - \frac{2}{(i+1)(i+2)} + \frac{i}{2(t-1)} + \frac{2}{i(i+1)} - \frac{i-1}{2(t-1)} \tag{4.32}$$

$$= \frac{4(t-1)}{i(i+1)(i+2)} - 1 + \frac{4}{i(i+1)(i+2)} + \frac{1}{2(t-1)} \tag{4.33}$$

$$= \frac{4t}{i(i+1)(i+2)} - 1 + \frac{1}{2(t-1)} \geq E_i t - 1. \tag{4.34}$$

□

As final thought one can observe that this model is very linear, since we get the recursive equations; also, if you care only about the degree distribution, then you only need the degree count vector. In general, knowing a model well enough can make run your computations faster, discarding information you don't need. There are some things we can carve out of this lesson. As example, how big a clique can be in social network? Say it can contain 20 people more or less (compute the largest clique is NP-complete).

In this model, the biggest clique is of size two, since it is a tree. It follows that it isn't a great representative of a social network under this aspect. Yet, it can give us one of the most striking properties, that is the power law degree distribution.

4.2 Small World Phenomenon

In the Sixties Stanley Milgram and other researchers conducted an experiment on the average path length for social networks of people (in the physical world, of course) in the United States. He wanted to claim that the six degree of separation property is valid (actually, he only thought that the average path length was small, then discovered the mean value).

Here's what he did. He picked something like a thousand people in Boston, Massachusetts, from the address book in a random fashion; then, he selected one guy in New York (a friend of his), let's call him *target*. He sent a letter to each of the 1000 people along the following lines:

> I would like this message to reach *target*, which resides in NYC, but you can only forward this message to people whom you know personally, and asking them to do the same if they are not *target*.

Each guy tries to reach *target* by sending the letter to the closest person he knows. What Milgram observed is that the average length of the chains of letters was 6.

In the following decades, this experiment was confirmed by other studies. The main consequence is the understanding of fact that the social graph diameter is small. In fact Jon Kleinberg observed something more striking: not only the chain among two random people is small in average, but people can find the next step without looking at the full graph. Each of the guys didn't know the shortest path, but only a sense of where their friend were.

Kleinberg tried to find a model that could explain the small diameter and allow the discovery of the chain efficiently.

In the model Kleinberg proposed, people live on a bi-dimensional grid, $n \times n$, and each person lives in exactly one cell of the grid, denoted by some coordinates (i, j). We can think of it also as a graph where the nodes are

$$V = \{(i, k) : 1 \leq i, j \leq n\}. \tag{4.35}$$

The simples model works as follows. Each person has short links to its geographical neighbor people. In addition, each person u has one long range contact v (denoted by $u \rightsquigarrow v$), where u and v are not geographical neighbors; this can be motived by the fact that sometimes people travel and become friends with people who live far away. A person (i, j) chooses as long range friend (k, l) (thus establishing a link) as follows.

$$Pr\{(i, j) \rightsquigarrow (k, l)\} \propto (|i - k| + |j - l|)^{-2} = l_1((i, j), (k, l))^{-2}, \tag{4.36}$$

where l_1 is the Manhattan distance. The exponent might feel arbitrary but we will show that, if the long range links are chosen in this way, then the expected length of the chains will be polylogarithmic. Any other value different from -2 will lead to a polynomial length. In a sense, one of the most striking practical observations Kleinberg made is that Equation 4.36 is the density at which long links should exists for this property to hold, and people have observed that the same type of density is what happens to exist in real social networks. The back-then Yahoo Messenger researchers, analyzing all the users' connections, confirmed that Equation 4.36 represents the correct guess form which links distribution is induced.

In terms of the graph, the actual probability is

$$Pr\{u \rightsquigarrow v\} = \frac{d(u,v)^{-2}}{\sum_{z \neq u} d(u,z)^{-2}} \tag{4.37}$$

where $d = l_1$. Observe that the long range links are chosen independently.

We would like to prove that chains have length $\log^2(n)$ in average. This isn't the same as saying that there exist one path of $\log^2(n)$; that wouldn't give us what we want. In fact, it can be shown that there always exists a path of length $\log(n)$, that is shorter. Rather, we want to claim that the algorithm we are going to see requires $\log^2(n)$ many steps (where each step is associated to a link) to find chains in practice. Also it is a tight bound, no other algorithm can do better.

Input: a node s, destination t, the message M;
$c \leftarrow s$;
while $c \neq t$ **do**
  $w \leftarrow$ neighbor of c that has the smallest l_1 distance to t (break ties arbitrarily);
  c forwards (M, t) to w;
  $c \leftarrow w$;
end

Algorithm 5: GEO-GREEDY algorithm

Algorithm 5 resembles the Milgram experiment. At each iteration, the current person c forwards the letter to the closest friend to the target t he knows.

Before proving any theorem, let's discuss a proof technique and make some observations along the way. First of all, the process we are studying can be in principle hard to study, but there are some properties of the random graph that help a lot.

The first one is that each node chooses its long range contact independently at random. Equation 4.37 can be thought as a distribution over random graphs. Why this can help us? If we get to a new node, then we can assume that exactly at the time we get there the long range contact is chosen, since the choice of the long range contact is independent of other random choices we made so far. The other thing is that we never get to the same node twice. You can always make one step closer to the destination. The distance of of the current node in the Algorithm 5 decreases by at least one at each step.

These properties allow us to study the algorithm and produce the random graph as the algorithm proceeds. As Algorithm 5 goes to a new node, a random independent choice is made at random. So it is easier than what happened in the Preferential Attachment model.

What we want to prove is that, no matter which node we are, there is a reasonable chance that we will be able to halve the distance from ourselves to the final destination. Suppose that, using the random choice, we can prove that the probability of halving the distance is at least some p (think of it as a large number). Then, after $\frac{1}{p}$ steps we will have divided by two the distance to the destination with constant probability. This proof idea has been seen different times by now, in the chain letters and coin tossing, as examples.

If t is the final destination, lets look at a large ball around t. If we are some node c there is a reasonable chance that our long range contact will end up inside this ball. The radius of the ball is half of the distance from t that we are currently at.

Theorem 4.2. *For any source s and destination t,* GEO-GREEDY *will make an expected number of steps to go from s to t of $O\left(\log^2(n)\right)$.*

Proof. Remember that, in the end, we want to show that the probability for our long range link to end up in the ball is high. For this purpose, it is necessary to estimate the number of nodes in the ball. But first, let's give a bound to the normalizing factor in Equation 4.37. By looking closer to it, we can derive the following statement.

$$\sum_{z \neq u} d(u,z)^{-2} = \sum_{i \geq 1} i^{-2} |\{z : d(z,u) = i\}| \tag{4.38}$$

We actually have to bound the number of nodes at any geographical distance from any given node. Let $S_u^r = \{z : d(z,u) = r\}$. What is the value of $|S_u^r|$? It is the number of nodes on the surface of the ball of radius r and centered in u. We will upper bound the size of the surface in the following way.

$$\begin{aligned}
S_u^r &= \{z : d(z,u) = r\} && (4.39)\\
&\subseteq \{(i+k, j+r-k) : 0 \leq k \leq r\} && (4.40)\\
&\quad \cup \{(i+k, j-r-k) : 0 \leq k \leq r\} && (4.41)\\
&\quad \cup \{(i-k, j-r-k) : 0 \leq k \leq r\} && (4.42)\\
&\quad \cup \{(i-k, j+r-k) : 0 \leq k \leq r\} && (4.43)
\end{aligned}$$

The reason why we partitioned the edges in this way is that it easy to compute the cardinalities. In fact we easily state that

$$|S_u^r| \leq 4(r+1). \tag{4.44}$$

We can improve the upper bound a little bit, because we are overcounting by an additive factor of four. By removing the elements in the intersections (the elements "at the corners") we can state the following Lemma.

Lemma 4.3.

$$|S_u^r| \leq 4(r+1) - 4 = 4r. \tag{4.45}$$

But is this bound tight for any node in the graph (or grid)? The nodes at the edges can get a better upper bound. A node in the corner will have only one of the above sets, while a node on the edge will have two. As r becomes bigger and bigger, we will exceed the size of the grid at some point in time, since it is finite, and the surface will be empty. This leads us to the following statement.

Lemma 4.4.

$$\forall r \leq \frac{n}{2}\ |S_u^r| \geq r \tag{4.46}$$

We give the idea to prove this lemma. Divide the grid in 4 parts. A node u is in one of those 4 parts; then it means we can pick a node at distance $\frac{n}{2}$ in both the adjacent quadrants' direction, defining a set of $r+1 \geq r$ nodes on the segment that connect the two picked nodes. We move $\frac{n}{2}$ steps in both direction, and pick the nodes that define the edge set.

To upper bound the normalizing factor shown in Equation 4.38, we will make use of the following observations.

Observation 4.1.

$$\sum_{i=1}^{t} \frac{1}{i} \leq 1 + \ln t + 1 \tag{4.47}$$

Observation 4.2.

$$1 + \log_b x = \log_b bx \tag{4.48}$$

Proof.

$$\begin{aligned} 1 + \log_b(x) &= \log_b(b) + \log_b(x) && (4.49)\\ &= \log_b(bx) && (4.50) \end{aligned}$$

□

From Equation 4.38 it can be derived that

$$\begin{aligned} \sum_{z \neq u} d(u,z)^{-2} &= \sum_{i \geq 1} i^{-2} |\{z : d(z,u) = i\}| && (4.51)\\ &\leq \sum_{i \geq 1}^{2n-2} i^{-2} 4i && \text{(Lemma 4.3)}\\ &= 4 \sum_{i \geq 1}^{2n-2} \frac{1}{i} && (4.52)\\ &\leq 4(1 + \ln(2n-1)) && \text{(Observation 4.1)}\\ &\leq 4(1 + \ln(2n)) && (4.53)\\ &= 4 \ln(2en) && \text{(Observation 4.2)}\\ &\leq 4 \ln(6n). && (4.54) \end{aligned}$$

We have a more algebraic bound on the normalizing factor, and we can state the following Lemma.

Lemma 4.5.

$$\Pr\{u \rightsquigarrow v\} \geq \frac{d(u,v)^{-2}}{4\ln(6n)}, \tag{4.55}$$

Now we can go back to the algorithm's behavior, analyzing how much time we need to halve the distance from where we are to final destination. We say that Algorithm 5 is in phase j if its current node c satisfies the constraint

$$2^j \leq d(t,c) \leq 2^{j+1}. \tag{4.56}$$

Observe that phases go in the unusual direction. The algorithm starts from phase j, with j that can be as large as $\ln(n+1)$ in the worst case, and then j decreases. Essentially, we want to show that the time to go from one phase to the other is small.

Define the random variable X_j to be

$$X_j = \textit{number of steps that we spend in phase } j. \tag{4.57}$$

The total number of steps X the algorithm will need, then, is

$$X = \sum_{j=0}^{1+\log(n)} X_j. \tag{4.58}$$

In this way we are bucketing the number of steps the Algorithm 5 takes according to phases. The reason why is that it's easy to upper bound the expected number of steps in a phase, because the distances between a node of a specific phase to the destination are going to be all within a constant factor to each other. If we manage to show that X_j is logarithmic then we are done. What does it mean that a phase, in expectation, will require at most $\log(n)$ many steps? It can happen because we have to take at least one long range link, and we will prove it now.

As we said, we want to hit a ball around the target when choosing the next long range link, and it happens with roughly the probability of being a node in the ball times the number of nodes in the ball. We said "roughly" because each node has not the same probability to be chosen, but they are not so different. We need to know how many nodes are in the ball and, for the probability of getting it the ball to be high, we need to lower bound that number. This is different from what we were doing before. To prove the lower bound on the probability of reaching a specific node, in Lemma 4.5, we had to upper bound the number of nodes in the sphere, whereas here we want the number of nodes in the ball to be large.

Here we care about the content of the sphere too, because it is sufficient to hit any node in the sphere, actually the closer to the center the better.

Definition 4.2. *We define the ball of radius r and centered in u as*

$$B_u^r = \{u : d(u,v) \leq r\}. \tag{4.59}$$

To lower bound the volume of the ball we can observe that

$$|B_u^r| \geq \left|B_u^{\lceil \frac{r}{4} \rceil}\right|, \tag{4.60}$$

since the radius of the ball in the RHS is smaller, making the ball cover less nodes. The reason for $\frac{r}{4}$ is that the lower bound in Lemma 4.4 works for radius values up to $\frac{n}{2}$ and the actual radius of the ball we care about might be large as the diameter of graph, roughly $2n$. What is the size of $B_u^{\lceil \frac{r}{4} \rceil}$? It is the sum of the sizes of the smaller spheres that are inside the ball. This is why we need to use Lemma 4.4. We make the ball smaller so that $2n$ will become $n/2$. Some contribution will be lost, but it has to be done.

$$\left|B_u^{\lceil \frac{r}{4} \rceil}\right| = \sum_{i=0}^{\lceil \frac{r}{4} \rceil} |S_u^i|. \tag{4.61}$$

Let's take out the cardinality of the sphere with radius 0,which is one.

$$\begin{aligned}
\left|B_u^{\lceil \frac{r}{4} \rceil}\right| &= \sum_{i=1}^{\lceil \frac{r}{4} \rceil} |S_u^i| + 1 && (4.62)\\
&\geq \sum_{i=1}^{\lceil \frac{r}{4} \rceil} i + 1 && \text{(Lemma 4.4)}\\
&= 1 + \frac{\lceil \frac{r}{4} \rceil (\lceil \frac{r}{4} \rceil + 1)}{2} && (4.63)\\
&\geq \frac{r^2}{32}. && (4.64)
\end{aligned}$$

Lemma 4.6.

$$|B_u^r| \geq \frac{r^2}{32}. \tag{4.65}$$

We now have all the pieces we need. We know how big the ball is going to be and we have a good lower bound on the probability of our current node to have a long range link to any node in the ball.

Take phase j and consider any node $x \in B_t^{2^j}$. Let c be the current node in Algorithm 5.

$$\begin{aligned}
d(c,x) &\leq d(c,t) + d(t,x) && (4.66)\\
&\leq 2^{j+1} + 2^j && (4.67)\\
&\leq 2^{j+2}. && (4.68)
\end{aligned}$$

What we got is that, for any node in the final ball we are trying to hit, the distance between c and that one node is not that large. Let's apply Lemma 4.5.

$$Pr\{c \rightsquigarrow x\} \geq \frac{d(c,x)^{-2}}{4\ln(6n)} \tag{4.69}$$

$$\geq \frac{2^{-2j-4-2}}{\ln(6n)} \tag{4.70}$$

$$= \frac{2^{-2j-6}}{\ln(6n)}. \tag{4.71}$$

Inequality 4.71 shows the probability of c to have the long range contact inside the ball in phase j. Why is this useful? We have the square of the radius many nodes in the ball. The radius i 2^j, which yields to 2^{2j} by substitution; multiplying that quantity with the lower bound shown in Inequality 4.71 results in a constant over $\log(n)$, which is large enough.

$$Pr\Big\{\exists x \in B_t^{2^j} : c \rightsquigarrow x\Big\} = \sum_{x \in B_t^{2^j}} Pr\{c \rightsquigarrow x\} \tag{4.72}$$

$$\geq \sum_{x \in B_t^{2^j}} \frac{2^{-2j-6}}{\ln(6n)} \tag{4.73}$$

$$\geq \frac{(2^j)^2}{32} \frac{2^{-2j-6}}{\ln(6n)} \tag{4.74}$$

$$= \frac{2^{2j-5-2j-6}}{\ln(6n)} \tag{4.75}$$

$$= \frac{2^{-11}}{\ln(6n)} = \frac{1}{2048\ln(6n)}. \tag{4.76}$$

The constant is large, but we didn't pay too much to make it smaller; it doesn't matter for our purposes anyway. Now, how many steps the algorithm will spend in phase j? We have a reasonable probability of hitting the ball and going to the next phase.

Observation 4.3.

$$\sum_{i=0}^{\infty}(1-x) = \frac{1}{x},\ 0 < x < 1. \tag{4.77}$$

Let's compute the expected number of steps.

$$E[X_j] = \sum_{i=1}^{\infty} iPr\{X_j = i\} \tag{4.78}$$

$$= \sum_{i=1}^{\infty} Pr\{X_j \geq i\} \tag{4.79}$$

$$\leq \sum_{i=1}^{\infty} \left(1 - \frac{1}{2048\ln(6n)}\right)^{i-1} \tag{4.80}$$

$$= \sum_{i=0}^{\infty} \left(1 - \frac{1}{2048\ln(6n)}\right)^{i} \tag{4.81}$$

$$= 2048\ln(6n). \tag{4.82}$$

Finally, notice that we are summing the expected value $\log(n)$ times,yielding to an expected running time of $\log^2(n)$.

□

There are a few things that could be said. One is that this proof only shows that in expectation the algorithm takes $\log^2(n)$ steps; what can be shown is that this happens with high probability. The second thing is, as observed by Kleinberg, if the probability doesn't use the constant -2, then the time to completion becomes polynomial. In the exposed work he proposed a model according to which people chooses long range contacts. The model was validated years later looking at data, a very surprising outcome. Other observations are on the model. We started from a grid, but it is not that important. The proof has been recreated for other geometries. Finally, one can regards this as a beautiful example of using theory to guess about realty, without being able to meet that guess.

Chapter 5

Linear programming for approximation algorithms

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, although the original solution used max flow.

5.1 Linear Programming

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. The following is a famous example for introducing the technique.

Suppose you are a farmer and have a L Km^2 field; you need to decide how to plant crops in your field. Should you plant only potatoes or barleys, or a mix of the two? Each of the crops will make you earn a different amount of money when you will sell it. Define

$$S_p = \text{selling price for potatoes, per Km}^2$$
$$S_b = \text{selling price for barleys, per Km}^2$$

If you had no constraint, you would plant the crop with highest selling price. But the size of the field is limited, as well as the amount F of fertilizer you have.

$$F_p = \text{Weight of fertilizer needed for potatoes, per Km}^2$$
$$F_p = \text{Weight of fertilizer needed for barley, per Km}^2$$

We can solve this introducing two variables X_p and X_b, which are the amount of

square Km that will be allocated to potatoes and barleys respectively.

$$\begin{cases} X_p + X_b \leq L \\ F_p X_p + F_b X_b \leq F \\ X_p, X_b \geq 0 \\ \max(S_p X_p + S_b X_b) \end{cases} \tag{5.1}$$

The goal is to find values of the variables that satisfies the constraints above and maximizes the *objective function*.

In general, a linear program can have zero, one or infinite solutions and zero, one or infinite optimal solutions. To simplify the discussion here, we assume only one optimal solution. A simple LP like the above was very useful in past years when they could solve this by hand or with simple algorithms. For a number of centuries, the *simplex algorithm* was the only algorithm known; it is very inefficient, meaning that in general will require a time exponential in number of variables and constraints. As of today, there are algorithms that run in polynomial time for this kind of linear programs.

The form of a general linear program comprises a variable vector $\overline{x} \in \mathcal{Q}^{n \times 1}$, coefficient and constant matrices $\mathbf{A} \in \mathcal{Q}^{m \times n}$, $\mathbf{b}, \mathbf{C} \in \mathcal{Q}^{1 \times n}$. We are using rationals because we would like to be able to do computations. If they were reals, then it wouldn't be clear even how to perform additions between two numbers (because of the infinite precision required).

$$\begin{cases} \max \mathbf{C}\overline{x} \\ \mathbf{A}\overline{x} \leq \mathbf{b} \\ \overline{x} \geq 0 \end{cases} \tag{5.2}$$

The non-negativity constraints are very important, since otherwise the solution wouldn't make sense (in fact, it could create an unbounded solution). The system of inequalities 5.2 is called a general *primal* linear program. "General" because any LP can be reduced to it, and "primal" because it has a maximization as objective function (by convention). Having a general form is desirable because one can develop algorithms that work for any LP and which running time depends on the parameters of the system. Here, the parameters are the number of constraints, which is m, and the number of variables, which is n. Observe that it is impossible to get an algorithm which runs in $O((mn)^c)$, since the input size is much larger than that. One has to take in account the size of $\mathbf{A}$ in terms of bits. In fact, one can prove that the running time is

$$T(m,n) = O\left((nm \log\left(|A|_\infty\right))^c\right), \tag{5.3}$$

where $\log\left(|A|_\infty\right)$ is the bit cost of the input.

In the *knapsack problem* we have the same situation, since the algorithm to solve it uses dynamic programming and thus a table $\mathbf{A}$. The table contains an element for each sum of the values and each sum of the weights of the elements you can put inside the knapsack. The input to the algorithm is a tuple (t^n, t^n) representing the weights and the

values, and can be represented with polynomially many bits (in fact $O(nt \log t)$). If we were to use dynamic programming, though, then the running time will be exponential, since we have to scan at least t^n many entries. When people say that knapsack is polynomial time solvable they assume the weights' value is bounded by a polynomial. Most ot the time one has to take in account the bit cost of the input.

Notice that the bound expressed in Equation 5.3 is not tight (we don't know the constant c). There is no way to say which is the exact running time because we have algorithms that we don't fully understand, but works well in practice. From an approximation algorithms' point of view we can say that, if the LP has polynomial size, then it is solvable in polynomial time. In fact, one could actually solve an exponential LP in polytime (not shown here).

One important concept we will see in use later is the *dual* linear problem of a primal problem. It is defined as

$$\begin{cases} \min \mathbf{b}'\overline{y} \\ \mathbf{A}'\overline{y} \leq \mathbf{C} \\ \overline{y} \geq 0 \end{cases} \tag{5.4}$$

The primal is a max linear program, whereas the dual is a min linear program obtained using the same (transposed) coefficient and constant matrices of the primal; the variable vector has dimension m instead of n. This connection is useful because of the following theorem.

Theorem 5.1 (Weak Duality)**.**

$$OPT(primal) \leq OPT(dual) \tag{5.5}$$

If we have a primal, then we can guess a solution X, check that it satisfies the constraints and evaluate how good it is by using the objective function. But how do we 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 we are close to the optimal solution.

Theorem 5.2 (Strong Duality)**.**

$$OPT(primal) = OPT(dual) \tag{5.6}$$

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

5.2 K-max cover as LP

To give an example, we will revise the K-max cover as a linear program. The first thing we need to guarantee is that the solution selects only K sets. So lets introduce a variable

X_i for each set, such that

$$X_i = \begin{cases} 1 & \text{If the i-th set is picked,} \\ 0 & \text{otherwise.} \end{cases} \tag{5.7}$$

Therefore their sum must be K. Then, I should be able to tell if I picked or not a specific element, since my objective function in max cover is "pick as many element you can". So

$$Y_i = \begin{cases} 1 & \text{If the i-th element is picked,} \\ 0 & \text{otherwise.} \end{cases} \tag{5.8}$$

Notice that Y_i can be one only if at least one variable X_j such that element i is in set j, is set one. It means we picked a set containing element i, so we picked element i. It follows a linear program, with $i, j \in [n]$.

$$\begin{cases} \sum_{i=1}^{m} X_i \leq k \\ Y_j \leq \sum_{i:j \in S_i} X_i \\ 0 \leq X_i, Y_j \leq 1 \\ \max \sum_j Y_j \end{cases} \tag{5.9}$$

Notice that we can't force the variables to be binary. If we could, then we would get an optimal solution for K-max cover. We cannot do that, since max-cover is NP-Hard. Therefore, what people do is to look for fractional solutions, which are not properly suited for max cover since we would picking, say, a set for $\frac{1}{3}$ of it and, say, we could pick $\frac{2}{3}$ of an element. Of course, it doesn't make sense, but what can be proven is the following theorem.

Theorem 5.3. *The optimal k-max cover solution has a value which is*

$$\left(1 - \frac{1}{e}\right) OPT_{LP} \leq OPT_{MC} \leq OPT_{LP}, \tag{5.10}$$

where OPT_{LP} is the optimum found using the LP and OPT_{MC} is the actual optimal value for k-max cover.

And we have already proved this approximation. In fact, the proof we gave was indirectly based on the linear program, since were proving a bound between the dual of the Linear Program 5.9 and the optimal solution for max cover.

5.3 Densest Subgraph problem

The densest subgraph problem is that of finding a subgraph of maximum density. Formally, give a graph $G = (V, E)$, find $S \subseteq V$ that maximizes

$$f(S) = \frac{|E(S)|}{|S|}, \tag{5.11}$$

where

$$E(S) = \{\{i,j\} : \{i,j\} \in E \wedge \{i,j\} \subseteq S\}. \tag{5.12}$$

In other words, we are looking for a part of the graph that maximizes the average degree (within a factor of 2).

The densest subgraph problem can be formulated as a linear program using some of the ideas used for K-max cover. 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

$$\max \sum_{\{i,j\} \in E} X_{\{i,j\}}. \tag{5.13}$$

For each edge, we cannot pick that edge fractionally more than the amount with which we pick one of its endpoint. 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

$$X_{\{i,j\}} \leq Y_i, \tag{5.14}$$

$$X_{\{i,j\}} \leq Y_j. \tag{5.15}$$

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.

$$\sum_{i=1}^{n} Y_i \leq 1. \tag{5.16}$$

Finally, the variables must be nonnegative. Putting all together we have the following LP.

$$\begin{cases} \max \sum_{\{i,j\} \in E} X_{\{i,j\}} \\ X_{\{i,j\}} \leq Y_i \\ X_{\{i,j\}} \leq Y_j \\ \sum_{i=1}^{n} Y_i \leq 1 \\ X_{\{i,j\}}, Y_j \geq 0 \end{cases} \tag{5.17}$$

What we will be proving is that the optimal solution $\mathrm{OPT}_{\mathrm{LP}}$ of this LP is equal to the optimal solution of the original densest subgraph problem. To do so we need the following set of lemmas.

Lemma 5.1.

$$\forall S \subseteq V \;\; OPT_{LP} \geq f(S). \tag{5.18}$$

Proof. To prove this we have to show a solution of the LP which is feasible and has a value at least $f(S)$. Thus, now we give one and show that it meets the requirements. Let

$$Y_i = \begin{cases} \frac{1}{|S|} & i \in S, \\ 0 & i \notin S. \end{cases} \tag{5.19}$$

Remember that the Y_i's were the variables representing how much of each node in the graph we are taking, and Equation 5.19 states that we split the unit among the nodes in the set S.

The second part is the following:

$$X_{\{i,j\}} = \begin{cases} \frac{1}{|S|} & \{i,j\} \subseteq S, \\ 0 & \{i,j\} \not\subseteq S. \end{cases} \tag{5.20}$$

To show this solution is feasible we check that every constraint is satisfied. Let's start with Inequality 5.16.

$$\sum_i Y_i = |S| \frac{1}{|S|} = 1, \tag{5.21}$$

so far, so good. Let's proceed with Inequality 5.14, 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

$$\begin{cases} X_{\{i,j\}} \leq Y_i \\ X_{\{i,j\}} \leq Y_j \end{cases} \implies X_{\{i,j\}} \leq \min\{Y_i, Y_j\}, \tag{5.22}$$

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

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

$$\frac{1}{|S|} = X_{\{i,j\}} \leq \min\{Y_i, Y_j\} = \frac{1}{|S|}. \tag{5.23}$$

We demonstrated that the solution is feasible. But what is its value?

$$\sum_{\{i,j\} \in E} X_{\{i,j\}} = |E(S)| \frac{1}{|S|} = f(S). \tag{5.24}$$

□

What we have shown is that the LP's optimal value is never worst than the original densest subgraph problem's optimum. Now we want to prove the other direction to get equality.

Lemma 5.2. *Let $\{X_{\{i,j\}}, Y_i\}$ be an optimal solution to the Linear Program 5.17, having value v. Then,*

$$\exists S \subseteq V \ f(S) \geq v. \tag{5.25}$$

Proof. Things are a little trickier here. In fact, what Lemma 5.1 states is that our LP's optimum is larger than any possible solution of densest subgraph problem; but then, of course, we don't expect all the DSG solutions to have the same value, so we have to pick a specific one and show it is as good as our LP's. Why did we assume optimality instead of only feasibility? Because we want the LP solution to have the following property.

$$\forall \{i,j\} \in E \ X_{\{i,j\}} = \min\{Y_i, Y_j\}. \tag{5.26}$$

Proof. Suppose this was not the case, then

$$\exists X_{\{i,j\}} < \min\{Y_i, Y_j\}. \tag{5.27}$$

But now we can increase the value of $X_{\{i,j\}}$ up to the minimum and so the objective function's value increases. It follows that we didn't have them optimal value (contradiction). □

Now we proceed with the Lemma's proof, due to Moses S. Charikar. Let

$$S(r) = \{i : Y_i \geq r\}, \tag{5.28}$$
$$E(r) = \{\{i,j\} : X_{\{i,j\}} \geq r\}. \tag{5.29}$$

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 5.1.

$$\forall r, \ 0 \leq r \leq \max\{Y_i\}, \ \{i,j\} \in E(r) \iff \{i,j\} \subseteq S(r). \tag{5.30}$$

Proof.

$$\begin{aligned}
\{i,j\} \in E(r) &\implies X_{\{i,j\}} \geq r && (5.31)\\
&\implies \min\{Y_i, Y_j\} \geq r && \text{(Equation 5.26)}\\
&\implies Y_i \geq r \wedge Y_j \geq r && (5.32)\\
&\implies \{i,j\} \subseteq S(r). && (5.33)
\end{aligned}$$

Now the other direction.

$$\begin{aligned}
\{i,j\} \subseteq S(r) &\implies r \leq X_{\{i,j\}} = \min\{Y_i, Y_j\} && \text{(Equation 5.26)}\\
&\implies \{i,j\} \in E(r). && (5.34)
\end{aligned}$$

□

No matter which r we pick, it will select a set of nodes and also all the edges induced by that set of nodes. Let's look at the following integral.

$$\int_0^{\max\{Y_i\}} |S(r)|dr = \sum_{i=1}^{n} \int_0^{Y_i} 1dr \tag{5.35}$$

$$= \sum_{i=1}^{n} Y_i. \tag{5.36}$$

Let's do the same for the edges.

$$\int_0^{\max\{X_{\{i,j\}}\}} |E(r)|dr = \sum_{\{i,j\}\in E} \int_0^{X_{\{i,j\}}} 1dr \tag{5.37}$$

$$= \sum_{i=1}^{n} X_{\{i,j\}}. \tag{5.38}$$

Claim 5.2.

$$\exists r \; r \in [0, \max\{Y_i\}] \; s.t. \; |E(r)| \geq v|S(r)|. \tag{5.39}$$

Making this claim actually means saying that

$$f(S(r)) = \frac{|E(r)|}{|S(r)|} \geq v, \tag{5.40}$$

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

Proof. We proceed by contradiction. Assume that

$$\forall r \; r \in [0, \max\{Y_i\}] \text{ s.t. } |E(r)| < v|S(r)|. \tag{5.41}$$

Then,

$$\int_0^{\max\{Y_i\}} |E(r)|dr < v \int_0^{\max\{Y_i\}} |S(r)|dr \tag{5.42}$$

$$\implies \tag{5.43}$$

$$\sum_{\{i,j\}\in E} X_{\{i,j\}} < v \sum_{i=1}^{n} Y_i \tag{5.44}$$

$$\tag{5.45}$$

By optimality of the LP solution it is true that

$$\sum_{i=1}^{n} Y_i = 1 \; \wedge \sum_{\{i,j\}\in E} X_{\{i,j\}} = v. \tag{5.46}$$

It follows a contradiction, $v < v$. □

At this point we have proved Lemma 5.2 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.

5.4 A greedy approach to the Densest Subgraph problem

In the previous section the Densest Subgraph Problem was solved using linear programming. That approach, however, is pretty inefficient, since solving a LP requires a running time of $O(n^{3.5})$. Here, we explore a greedy algorithm to approximate the optimal solution. Let's recall the problem formulation. A graph $G(V, E)$ and the density function $f(S) = \frac{|E(S)|}{|S|}$ are given. We would like to find a set $S \subseteq V$ such that

$$S = \arg\max_{S' \subseteq V} f(S'). \tag{5.47}$$

The greedy algorithm we will analyze is the following.

Input: $G = (V, E)$
$S_0 \leftarrow V$;
for $i = 0, \ldots, |V| - 1$ **do**
  $w_i \leftarrow$ node with minimum degree in the graph induced by S_i;
  $S_{i+1} \leftarrow S_i - \{w_i\}$;
end
return $\arg\max_{S_i} f(S_i)$.

Algorithm 6: DSG-GREEDY algorithm to solve the Densest Subgraph Problem.

As it can be imagined, this algorithm runs very fast, i.e., in $O(|V| + |E|)$ time.

Theorem 5.4. *The solution returned by Algorithm 6 is a 2-approximation to DSG.*

Proof. We will prove the theorem with a combinatorial proof, but it also can be proven using a LP. In order to proceed with the analysis we need to introduce the following definition.

Definition 5.1. *An orientation of am undirected graph $G(V, E)$ is a directed graph $D(V, A)$ where the nodes are the same and every edge in E is given an orientation, forming a set of arches A.*

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 5.3. *If $D(V, A)$ is an orientation (any orientation) of $G(V, E)$ and d_M is the maximum indegree of $D(V, A)$, then,*

$$\forall S \subseteq V \ f(S) \leq d_M. \tag{5.48}$$

Proof. This is actually a pretty simple claim to prove. We know that

$$f(S) = \frac{|E(S)|}{|S|} \tag{5.49}$$

and

$$|E(S)| \leq \sum_{v \in S} indeg_D(v) \tag{5.50}$$

$$\leq \sum_{v \in S} d_M \tag{5.51}$$

$$= d_M |S|. \tag{5.52}$$

Why this is true? Pick any S, there will be some edges is S and each edge will be oriented in exactly one way. So, every edge $E(S)$ will be an arch that enters in some node in S, and therefore it will be counted. The fact is, if we sum up the indegrees, we will be also counting arches in the oriented graph D that come from nodes outside S to nodes in S. We assumed d_M is the maximum indegree in D, so we can upper bound every other indegree with that value.

Dividing by $|S|$ we obtain

$$f(S) = \frac{|E(S)|}{|S|} \leq d_M. \tag{5.53}$$

□

We will be creating an orientation as the algorithm progresses. At the beginning no edge is directed towards anything. Then, when we 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. The algorithm doesn't care about orientations, but we will use this concept in the proof. Let the $D(V, A)$ be the resulting orientation of $G(V, E)$.

Remember, our goal is to study the quality of the S_i having maximum density. Therefore, we are going to study the quality of S_i with respect to the indegree of the w_i we removed (from S_i). The indegree of w_i with respect to D is the degree of w_i in S_i. This is because we are picking the node w_i with minimum degree in S_i and we are directing all the edges in $E(S_i)$, incident on w_i, towards that node. And since we are

picking the node with minimum degree we can state that, $\forall i \in \{0, \ldots, |V|-1\}$,

$$\begin{aligned} indeg_D(w_i) &= deg_{S_i}(w_i) && (5.54)\\ &= \min_{v \in S_i} deg_{S_i}(v) && (5.55)\\ &\leq \operatorname*{avg}_{v \in S_i} deg_{S_i}(v) && (5.56)\\ &= \frac{1}{|S_i|} \sum_{v \in S_i} deg_{S_i}(v) && (5.57)\\ &= 2\frac{|E(S_i)|}{|S_i|} = 2f(S_i). && (5.58) \end{aligned}$$

What we have proved is that the indegree in D of w_i is at most twice the density in S_i. Why do we care? Because we have Lemma 5.3 that gives us the lower bound for the maximum indegree, which is the density, for any set. So what we are going to do is to sandwitch the optimal density first by w_i's degree and finally by the density of one of the set we created.

Let $i^\star$ be an index maximizing $indeg(w_{i^\star})$. If $S^\star$ is the optimal solution then

$$f(S^\star) \leq indeg_D(w_{i^\star}) \leq 2f(S_{i^\star}) \implies f(S_{i^\star}) \geq \frac{1}{2} f(S^\star). \quad (5.59)$$

□

The reason we care about this algorithm is that it can be run efficiently, precisely in time $O(|E|)$.

Remember that an LP has a primal and dual, and the dual gives an upper bound to the primal. What we did is to fix the solution of the dual which in this case was a given orientation. Then we compared the solution of greedy with the dual, analyzing the gap.

5.5 Parallel version of dsg-greedy

We now have a linear algorithm to approximate the DSG solution. 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.

The first observation is do we really need w_i to be the node with minimum degree? Or can we be less strict in choosing w_i? If you look at the previous proof, we used the minimum degree value 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. This observation is the key observation we can use the running time, since we don't have to look for the minimum degree at every step, but we can look for something which is much easier to get. It is a bit hard to give the intuition of why the minimum degree is harder than to find than the average, but you can sort of imagine it in this way. By Markov Inequality, if we pick at random

one node in the graph, the probability that it will have a degree which is less or equal the average degree times a 1.01 is at least a constant. The node of minimum degree will be hidden somewhere while there are many nodes with degree slightly larger than the average, we would lose some constant, but we can make that constant as small as we want.

Studying such an algorithm is going to be a bit more complicated

Input: $G(V, E)$
$S_0 \leftarrow V$;
$i \leftarrow 0$;
while $S_i \neq \emptyset$ **do**
  $A(S_i) \leftarrow \{v : v \in S_i \wedge deg_{S_i}(v) \leq (1+\varepsilon)\,\text{avg}_{w_i \in S_i}\, deg_{S_i}(w_i)\}$;
  $S_{i+1} = S_i / A(S_i)$;
  $i \leftarrow i + 1$;
end
return the "best" S_i

Algorithm 7: QUICK-GREEDY algorithm for densest subgraph problem.

In Algorithm 7 we pick as many nodes as we can because we want to finish quickly. The algorithm is almost the same as the precedent one; in this case we don't care about minimum degree but the average is enough, and if we go slightly above the average we believe it is acceptable (we lose a small constant in the approximation).

Theorem 5.5. *Algorithm* QUICK-GREEDY *will return a* $(2+2\varepsilon)$ *approximation and also will run for at most* $O\left(\frac{1}{\varepsilon}\log(n)\right)$ *iterations.*

Proof. Let $S^\star$ be an optimal solution.

Claim 5.3.

$$\forall v \in S^\star \;\; deg_{S^\star}(v) \geq f(S^\star). \tag{5.60}$$

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. By contradiction, suppose $\exists v \in S^\star \;\; deg_{S^\star}(v) < f(S^\star)$.

$$f(S^\star - \{v\}) = \frac{|E(S^\star - \{v\})|}{|S^\star| - 1} \tag{5.61}$$

$$= \frac{|E(S^\star)| - deg_{S^\star}(v)}{|S^\star| - 1} \tag{5.62}$$

$$\geq \frac{E(S^\star) - f(S^\star)}{|S^\star| - 1} \tag{5.63}$$

$$= \frac{E(S^\star)}{|S^\star|} \frac{|S^\star|}{|S^\star| - 1} - \frac{f(S^\star)}{|S^\star| - 1} \tag{5.64}$$

$$= f(S^\star) \left(\frac{|S^\star|}{|S^\star| - 1} - \frac{1}{|S^\star| - 1} \right) = f(S^\star). \tag{5.65}$$

So $S^\star$ is not the optimal solution (contradiction). □

Before going any further, 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.

Now 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. In the previous proof [insert link] we knew that that iteration existed because we knew in at least one iteration the node we would remove had the maximum indegree in the orientation. Here we don't have orientation, so we need to argue differently.

Fix an optimal solution $S^\star$. Let $i^\star$ be the first integer such that QUICK-GREEDY removes some element of the optimal solution $S^\star$ from $S_{i^\star}$ (to get $S_{i^\star+1}$). Notice that there must be such $i^\star$. Formally,

$$|A(S_{i^\star}) \cap S^\star| \geq 1 \ \wedge \ \forall j < i^\star \ |A(S_j) \cap S^\star| = 0 \tag{5.66}$$

It follows that $S^\star \subseteq S_{i^\star}$. Let v be a node in $A(S_{i^\star}) \setminus S^\star$. Then

$$f(S^\star) \leq deg_{S^\star}(v) \qquad \text{(Claim 5.3)}$$

$$\leq deg_{S_{i^\star}}(v) \tag{5.67}$$

$$\leq (1+\varepsilon) \operatorname*{avg}_{v \in S_{i^\star}} deg_{S_{i^\star}}(v) \tag{5.68}$$

$$= (1+\varepsilon) 2 \frac{|E(S_{i^\star})|}{|S_{i^\star}|} \tag{5.69}$$

$$= (1+\varepsilon) 2 f(S_{i^\star}). \tag{5.70}$$

We have proven the approximation. What remains to see is that the algorithm finishes in logarithmic many steps. It all depends on the fact that we take out many node at once, in each iteration.

$$2|E(S_i)| = \sum_{v \in S_i} deg_{S_i}(v) \tag{5.71}$$

$$= \sum_{v \in A(S_i)} deg_{S_i}(v) + \sum_{v \in S_i \setminus A(S_i)} deg_{S_i}(v) \tag{5.72}$$

$$\geq \sum_{v \in S_i \setminus A(S_i)} deg_{S_i}(v) \tag{5.73}$$

$$\geq \sum_{v \in S_i \setminus A(S_i)} \operatorname*{avg}_{w \in S_i} deg_{S_i}(w) \tag{5.74}$$

$$= |S_i \setminus A(S_i)|(1+\varepsilon) \operatorname*{avg}_{w \in S_i} deg_{S_i}(w) \tag{5.75}$$

$$= |S_i \setminus A(S_i)|(1+\varepsilon) 2\frac{|E(S_i)|}{|S_i|}. \tag{5.76}$$

It follows that

$$2|E(S_i)| \geq |S_i \setminus A(S_i)|(1+\varepsilon) 2\frac{|E(S_i)|}{|S_i|} \tag{5.77}$$

$$\implies 1 \geq (1+\varepsilon)\frac{|S_i \setminus A(S_i)|}{|S_i|} \tag{5.78}$$

$$\implies \frac{1}{(1+\varepsilon)} \geq \frac{|S_i \setminus A(S_i)|}{|S_i|} \tag{5.79}$$

$$\implies \frac{1}{(1+\varepsilon)} \geq \frac{|S_i| - |A(S_i)|}{|S_i|} \tag{5.80}$$

$$\implies \frac{1}{(1+\varepsilon)} \geq 1 - \frac{|A(S_i)|}{|S_i|} \tag{5.81}$$

$$\implies \frac{|A(S_i)|}{|S_i|} \geq 1 - \frac{1}{(1+\varepsilon)} \tag{5.82}$$

$$\implies \frac{|A(S_i)|}{|S_i|} \geq 1 - \frac{1}{(1+\varepsilon)} = \frac{1+\varepsilon}{1+\varepsilon} - \frac{1}{1+\varepsilon} = \frac{\varepsilon}{1+\varepsilon}. \tag{5.83}$$

We have proved that the fraction of nodes that are taken out at each iteration is $O(\varepsilon)$.

Observation 5.1.

$$x \leq \frac{1}{2}\ \log(1+x) = \Theta(x). \tag{5.84}$$

It can be proven using Taylor series.

And since we are taking out a constant fraction at each iteration, the running time

is

$$T(n) = O\left(\log_{1+\frac{\varepsilon}{1+\varepsilon}} |V|\right) \tag{5.85}$$

$$= O\left(\frac{\log |V|}{\log\left(1+\frac{\varepsilon}{1+\varepsilon}\right)}\right) \qquad \text{(Assume } \varepsilon \leq \tfrac{1}{2}\text{)}$$

$$= O\left(\frac{\log |V|}{\frac{\varepsilon}{1+\varepsilon}}\right) \tag{5.86}$$

$$= O\left(\frac{\log |V|}{\varepsilon}\right). \tag{5.87}$$

□

If you have a MapReduce cluster you could use the the above explained algorithm to analyze a billion of nodes in 40 iterations or so, meaning that it is pretty efficient. The way it was written down here forces it to remember all the candidates S_i. After we compute S_{i+1}, it could check what is the best set between S_i and S_{i+1} and keep the one with higher density.

Chapter 6

Clustering

In this chapter we present *clustering algorithms*, that is, algorithms that group together similar data items given a dataset and a metric.

Definition 6.1 (Metric)**.** *Given a set $\mathcal{X}$, a* metric *(or distance) $d : \mathcal{X} \times \mathcal{X} \to \mathcal{R}$ is a function that has the following properties:*

1. $\forall i, j \in \mathcal{X}\ d(i, j) \geq 0$.
2. $i, j \in \mathcal{X}\ d(i, j) = 0 \iff i = j$.
3. $d(x, y) \leq d(x, z) + d(y, z)$ *(triangle inequality).*

In clustering problems a metric is a measure of *similarity* between two data items.

6.1 The k-center problem

In this section we consider the k-center problem [5]. Given as input an undirected graph $G = (V, E)$, a metric d, defined as the distance between any two vertices in the graph, and an integer k. The goal is to find k clusters that group similar vertices together, such that the maximum distance between a vertex and its cluster center is minimized. The corresponding geometric intuition is that we are trying to find the set of k balls with the minimum radius r which allow them to cover all the space. Formally, we can define the distance between a vertex i and a set $S \subseteq V$ as

$$d(i, S) = \min_{v \in S} d(i, v). \tag{6.1}$$

Intuitively, each vertex i is associated to the center which is closest. Therefore, given a set S, which can be though as the set of centers, the minimum radius all the balls must have to cover the space is

$$r = \max_{v \in V} d(v, S). \tag{6.2}$$

In other words, the radius must be at least as the maximum distance between a vertex and the closest cluster center.

The following algorithm is a greedy solution which attempts to find the set S of k centers which minimizes r.

Input: A graph $G(V, E)$, a metric d.
$S \leftarrow \{i\}$, $i \in V$;
while $|S| < k$ **do**
  $x \leftarrow \arg\max_{v \in V \setminus S} d(v, S)$;
  $S \leftarrow S \cup \{x\}$;
end
return S

Algorithm 8: K-CENTER-GREEDY algorithm.

Theorem 6.1. *The* K-CENTER GREEDY *algorithm (Algorithm 8) gives a 2-approximation of the optimal solution.*

Proof. Let $S^\star \subseteq V$ be an optimal solution with radius $r^\star$. Assign each vertex $v \in V$ to one of the closest points in $S^\star$. Let $\{C(s_i^\star)\}_{s_i^\star \in S^\star}$ be the resulting clustering. Finally, denote with S the final solution of the greedy algorithm.

Lemma 6.1.

$$\forall x \in S^\star,\ \forall y, z \in C(x)\ d(y, z) \leq 2r^\star. \tag{6.3}$$

Proof. By triangle inequality,

$$\begin{aligned} d(y, z) &\leq d(x, y) + d(x, z) && (6.4)\\ &\leq 2r^\star. && (6.5) \end{aligned}$$

□

We will split the analysis in two cases.

- Suppose that $\forall x \in S^\star, C(x) \cap S \neq \emptyset$. Pick any $y \in C(x)$. By assumption, $\exists s \in S\ s \in C(x)$. It follows that $d(y, s) \leq 2r^\star$.
- Suppose now that $\exists x \in S^\star\ C(x) \cap S = \emptyset$. Then, by the pigeonhole principle, $\exists x' \in S^\star\ |C(x') \cap S| \geq 2$. Let $y, y' \in C(x') \cap S,\ y \neq y'$. Suppose y was added to S by the algorithm before y'. Let S' be the set of points that greedy selected before y'. Pick any $z \in V$.

$$\begin{aligned} d(z, S) &\leq d(z, S') && \text{(since } S' \subseteq S\text{)}\\ &\leq d(y', S') && \text{(greedy choice)}\\ &\leq d(y, y') && \text{(since } y \in S'\text{)}\\ &\leq 2r^\star. && \text{(by Lemma 6.1)} \end{aligned}$$

So we have proved the 2-approximation. □

Can we hope to do better? The following theorem says that the approximation is optimal if $P \neq NP$.

Theorem 6.2. *Proving a $(2-\varepsilon)$-approximation to the k-center problem is NP-hard.*

Proof. We will show a reduction from the *Dominating Set Problem*, which is NP-hard. In the dominating set problem, we are given a graph $G = (V, E)$ and an integer k; the objective is to find a set $S \subseteq V$ of size k such that for every node $w \in V$, $w \in S$ or w has a neighbor in S.

Definition 6.2 ($(1,2)$-metric)**.** *A $(1,2)$-metric is a function $d : V \times V \to \{1,2\}$ such that*

$$d(w,v) \in \{1,2\} \ \forall v, w \in V \wedge w \neq v.$$

Given an instance of dominating set, we build the k-center instance. The set of points is always V. The metric is d such that

$$d(u,v) = \begin{cases} 0 & u = v, \\ 1 & \text{if } u \text{ and } v \text{ are neighbors,} \\ 2 & \text{otherwise.} \end{cases} \tag{6.6}$$

There is a dominating set of size k if and only if the optimal radius for the k-center instance is 1. Furthermore any $(2-\varepsilon)$ approximation algorithm must always produce a solution or radius 1, since the distance can be either 1 or 2. □

6.1.1 A more general formulation

In many cases we are not bound to choose centers in the set of points that have to be clustered. Think of a set of points disposed as a circle in the 2D euclidian space as the metric space in input to our k-center algorithm. If one wants to pick one center with the restriction that it must be part of the given metric, any of them will do. But if you could pick the center of the circle (which is outside the subset of points chosen) then you could cover every point with minimal cost. In general, there is a set of points W we want to cover and another set V which is the set of points we can use as centers. In light of this, we can give another, more general, definition of the k-center problem, call it "formulation B". Given a metric space V and $W \subseteq V$, the goal is to find a set of center $\mathcal{C} \subseteq V$ such that:

$$\mathcal{C} = \arg\min_{C \in \binom{V}{k}} \max_{w \in W} \min_{c \in C} d(w,c). \tag{6.7}$$

If we force the centers to be chosen in W, we get back to the previous formulation, let us call it formulation A. In fact, people usually try to solve a problem by modeling it into formulation B and then claim that the result is a constant approximation to the

optimal solution for the formulation A of k-center. For simplicity, assume that we are trying to find only one center. Furthermore, suppose that

$$\mathcal{C}^\star = \{c^\star\} \subseteq V,\ c^\star \in V. \tag{6.8}$$

If the optimal center is in V, then how much would we pay if we were to substitute that optimal center with a point in W? What we want to claim is that the cost of the optimal solution or, in general, of picking a center in V is not that smaller than the one we would have if we had to pick a node inside W.

Lemma 6.2.

$$\max_{w \in W} d(w, c^\star) \geq \max_{\{w_1, w_2\} \in \binom{W}{2}} \frac{d(w_1, w_2)}{2}. \tag{6.9}$$

Proof. By triangle inequality we can state that

$$d(w_1, c^\star) + d(c^\star, w_2) \geq \max_{\{w_1, w_2\} \in \binom{W}{2}} d(w_1, w_2). \tag{6.10}$$

It follows that

$$\max_{\{w_1, w_2\} \in \binom{W}{2}} d(w_1, w_2) \leq \max_{w \in W} d(w, c^\star) + \max_{w \in W} d(w, c^\star) \tag{6.11}$$

$$= 2 \max_{w \in W} d(w, c^\star). \tag{6.12}$$

Finally, divide both sides by 2. □

So any point in W, picked as center, will give a 2-approximation of the optimal center. This is true for 1-center. What about k-centers? Well, use the K-CENTER-GREEDY algorithm for $V = W$; it will returns a bunch of clusters. It is known, for each of them, that the best point in the metric subspace o can give you a 2 approximation so it gives a factor of 4(?)

6.2 The k-median problem

Let's introduce k-median problem. As usual, given a metric space (V, d) we are looking for the best k centers, but now we care about minimizing the sum of the distances. Formally, the solution the k median problem is a set of centers C such that

$$\min_{C \in \binom{V}{k}} \sum_{v \in V} \min_{c \in C} d(v, c) \leq \delta. \tag{6.13}$$

Intuitively, the objective is to minimize the average distance of the points to the closest center. Why average? If we multiply Inequality 6.13 by $\frac{1}{|V|}$ the sum becomes an average. One issue with the k center is that it is sensible to outliers, which may affect too much the choice of centers; taking the average mitigates the problem.

The algorithm k-median admits a $(3+\varepsilon)$ in polynomial time. This algorithm uses a local search starting from k points; then for each point in the current set and for each point outside the current set, swap the two points and see the resulting improvement. If the cost decreases, the swap is confirmed. In fact, to get the $(3+\varepsilon)$ approximation, it is necessary to look not at a single point swap, but at a $\frac{1}{\varepsilon}$ points swap. We are not going to look at the algorithm, but the one question about 1-median we are talking about, does something like the reduction to $W = V$ to $W \subseteq V$ holds for k-median as well (not so strong though). So, what happens for 1-median?

Suppose $c^\star$ is the optimal solution from the larger metric V.

$$\sum_{w \in W} \min_{c \in \{c^\star\}} d(w, c) = \sum_{w \in W} d(w, c^\star) = \text{OPT}. \tag{6.14}$$

For 1-center, any point in w would have been enough to approximate the solution. For 1-median, the claim is different.

Theorem 6.3. *If w is chosen uniformly at random from W, then the expected 1 median cost of w is at most $2OPT$.*

Proof. Pick a bunch of points from W, say $\frac{1}{\varepsilon}$, we will know that high probability at least one of them give us a $(2+\varepsilon)$ approximation. We don't have to look at all the points in W, but constantly many, and get an approximation which is arbitrarily close to 2. The approximation gets better as the number of points picked grows. Let's see what is the cost of picking a random w as a center

$$\begin{aligned} E[\text{1-median cost of w}] &= \frac{1}{|W|} \sum_{w \in W} \sum_{w' \in W} d(w, w') && (6.15)\\ &= \frac{2}{|W|} \sum_{\{a,b\} \in \binom{W}{2}} d(a, b) && (6.16)\\ &\leq \frac{2}{|W|} \sum_{\{a,b\} \in \binom{W}{2}} (d(a, c^\star) + d(c^\star, b)). && (6.17) \end{aligned}$$

How many times we are considering $d(x, c^\star)$, $x \in W$? $|W| - 1$ times.

$$\begin{aligned} \frac{2}{|W|} \sum_{\{a,b\} \in \binom{W}{2}} (d(a, c^\star) + d(c^\star, b)) &= \frac{2}{|W|}(|W| - 1) \sum_{x \in W} d(x, c^\star) && (6.18)\\ &= \frac{2}{|W|}(|W| - 1)\text{OPT} && (6.19)\\ &= \left(2 - \frac{2}{|W|}\right) \text{OPT}. && (6.20) \end{aligned}$$

□

And so again we have a simple algorithm that given a cluster will find one approximate one median for that cluster. Just pick some random point in the cluster. One could ask if this is tight. What we said is if we pick one point from w the quality decrease buy that factor. It is tight. Think of a star graph, and consider the shortest path metric. V is the full set of nodes, while $W \subseteq V$ is the set of leaves; if we can choose $c^\star$ to be the root node, then

$$\sum_{w \in W} d(w, c^\star) = |W|. \tag{6.21}$$

Now suppose $c^\star \in W$. Then what could happen is that we would have

$$\sum_{w' \in W} d(w, w') = 2(|W| - 1) \tag{6.22}$$

If we take the ratio then we get the bound. So this algorithm gives us a way to approximate a 1-median given a set of points. Just pick a random point. On expectation it is a 2approximation.

What we would like to prove now is a different type of tightness for 1-center to stress the difference between clustering a bunch of points in the metric and clustering all the points in the metric. Last time we have proved that for k center we have proved a tight 2-approximation. How hard 1 center is? By just looking at the definition we know that it is solvable in $O(|V|^2)$. More in general, if k is a constant, then the problem is solvable in polynomial time. What about the problem of clustering $W \subset V$? If you want to be polynomial in W? Any point from W would give us a 2 approximation, by 1 center reduction. We will prove it is an optimal approximation. This claim seems simple but it is not. Proving something is NP hard requires defining which metric we are looking at and also if we want to prove in terms of size of points and not metric, it means metric given implicitly, because we want the algorithm be polynomial in W and we know we can be polynomial in $|V|$ (try all the possible centers). The reduction is from 3-SAT to 1-median.

3-SAT is the problem of finding an assignment of n variables such that a set $\mathcal{C}$ of m clauses,

$$\mathcal{C} \subseteq \binom{\{x_1, x_2, \ldots, x_n, \overline{x_1}, \overline{x_1}, \ldots, \overline{x_n}\}}{3}, \tag{6.23}$$

are all satisfied. Now we build a 1-median instance starting from 3SAT. Define a metric space which contains a point C_i for each clause, and 2^n points for each of the truth assignments to the n variables. The distance d associated to the metric space is defined as

$$d(a,b) = \begin{cases} 2 & a = C_i,\ b = C_j,\ i \neq j \\ 2 & \text{if } a \text{ and } b \text{ are truth assignments} \\ 1 & a = C_i,\ b \text{ is a truth assignment such that } C_i(b) = true \\ 2 & \text{otherwise.} \end{cases} \tag{6.24}$$

The 1-median instance asks for one point in the metric having an average distance of 1 from points in $W = \{C_1, C_2, \ldots, C_m\}$.

If there exists a truth assignment to the 3-SAT instance, the associated point in the metric will be at average distance of 1 from the points in W. If there exists a point in the metric at average distance ≤ 1 from W, then that point is a truth assignment with satisfies $\mathcal{C}$.

6.3 Correlation Clustering

Correlation clustering (abbreviated as CC) provides a method for clustering a set of objects into the optimal number of clusters without specifying that quantity in advance. Instead, given two objects v, v' an *oracle* will tell us if they are similar or not. In practice, you can think of a *classifier* that takes as input two objects and returns $+1$ if they are similar, or -1 if they are not. We can model this scenario as a graph $G = (V, E^+, E^-)$, such that $E^+ \cap E^- = \emptyset$, $E^+ \cup E^- = \binom{V}{2}$, where there is a positive edge between two nodes if they are similar, a negative edge if they are dissimilar. Note that in the version of CC that we are going to expose the graph is complete, that is, we always know if two nodes are similar or not.

The result of the clustering operation is a partition P of V, where the cost of P is equal to the number of edges of E^+ that are split by P, plus the number of edges of E^- that are not. This is because we want the positive edges to stay inside clusters, and negative edges to exists only between clusters. The problem is to find a partition of V of minimum cost. We are going to show that there exists a 3-approximation for it.

If we have a problem whose cost can shrink to zero, then we have to show that zero cost instances can be solved in polynomial time; otherwise, no approximation is possible. Why is the case we can easily solve zero cost instances? For an instance to have cost zero, you can partition the node such that all positive edges are inside the cluster and all the negative edges are between the cluster. If this is the case, we just pick one node and its positive neighbors and create a cluster. Repeat the operation until no node is left.

The algorithm we present is the following.

Input: G, a fully connected graph.
$\mathcal{C} \leftarrow \emptyset$;
while $V \neq \emptyset$ **do**
  Pick $v \in V$ uniformly at random;
  $C \leftarrow \{v\} \cup \{w | w \in V \wedge \{v, w\} \in E^+\}$;
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\}$;
  $V \leftarrow V \setminus C$;
end
return $\mathcal{C}$;

Algorithm 9: GREEDY-CC algorithm.

We said that the algorithm GREEDY-CC returns a 3 approximation. In fact, we will prove something slightly weaker.

Theorem 6.4.

$$E[\textit{cost of } \text{GREEDY-CC}] \leq 3OPT \tag{6.25}$$

This is a pretty impressive claim; to see what are scenarios where the algorithm fails let's look at the following cases. Consider a star graph with v_1 as root and $v_2, \ldots, v_n$ as leaves. The node v_1 is connected to the other nodes via positive edges; all the other node pairs have negative edges between them. We call *pivot* the current node selected by the algorithm. If we pick v_1 as first pivot, we will create a single cluster with all the nodes. But the cost is pretty large, in fact quadratic. If we pick v_2 as pivot, then the algorithm creates a cluster containing only v_2, v_1. All the positive edges go away, and the remaining nodes will form singleton clusters. Here the cost is $n-2$. There is a big gap between the two possible runs of the algorithm. If the probability of selecting a bad node is high, the algorithm won't work. But, in this case, what is the probability of picking v_1 as pivot?. Since every other node can take away v_1, the optimal solution will be chosen with probability $1 - \frac{1}{n}$. What we want to prove is that the chance of picking a bad solution is small.

Studying this algorithm is hard because a single choice will change what you have to do in the next step. If you think about the Kleinberg model [reference here], we wanted to show that we get to the destination in $\log^2 n$ many steps. The idea there was "well, if we are on a given node there is a small chance, $\frac{1}{\log n}$, to make the right jump and get closer to the destination. If we do that, then great, we are closer to the destination; if we don't, it's fine because at the next step we will have another chance". It is not the case here. But still, this thing works out.

Proof. Define Cost_i^- to be the number of negative edges inside cluster i and Cost_i^+ to be the number of positive edges from cluster i to cluster j, $j > i$. Then, we can define

$$\text{Cost}_i = \text{Cost}_i^- + \text{Cost}_i^+. \tag{6.26}$$

Notice that in this way we partition the cost among clusters.

Lemma 6.3.

$$\sum_i \textit{Cost}_i = \textit{Cost of the greedy solution.} \tag{6.27}$$

Proof. Can be easily seen by the way we defined the cost. □

We introduce the concept of "bad triangle". It is a tool that we will use to upper bound the cost of the algorithm and the cost of the optimal solution. A bad triangle is a triple $B = \{v_1, v_2, v_3\} \in \binom{V}{3}$ such that $|\binom{B}{2} \cap E^-| = 1$. It is "bad" because each time we encounter this triple in out solution we have to pay at least one unit of cost.

Now we bound the cost in terms of bad triangles.

Lemma 6.4. *$Cost_i = |\{B|B$ is a bad triangle containing the i-th pivot, and which is still completely contained in the remaining graph when the i-th pivot is chosen.$\}|$.*

Proof. We know that $\text{Cost}_i = \text{Cost}_i^+ + \text{Cost}_i^-$. Let's look at Cost_i^- first. It counts the number of negative edges inside a cluster. Each negative edge v_i, v_j is added because the i-th pivot v has positive links to v_i and v_j. It follows that $\{v, v_i, v_j\}$ forms a bad triangle. It is trivial to see that this triangle is counted once.

Lets now consider Cost_i^+. There is positive edge between cluster i and cluster j, $i < j$, in the following scenario. The i-th pivot v_i adds v, one of its neighbors, to its cluster; v has also a positive link to the j-th pivot. On one hand, between v_i and v_j there is a negative edge, otherwise v_j would be in cluster i. On the other, v_j can't add v to its cluster since it has already been picked by v_i. So there is the positive link between cluster i and j. We have also a bad triangle $\{v_i, v, v_j\}$ centered in v. This triangle is counted only in the i-th iteration since it is still present in the graph. When iteration j comes, nodes v, v_i have been already taken out.

Remember that the graph is fully connected, and it is the reason why this works. □

The reason why we introduced the concept of bad triangle is that we do kill a lot of bad triangles when selecting a pivot, without having to pay for them. This leads to the following corollary.

Corollary 6.1. *Let $\mathcal{B}$ be the set of all bad triangles. The expected cost of* GREEDY-CC *is the number of bad triangles hit by a pivot when they are still completely in the graph and this is equal to*

$$\sum_{T \in \mathcal{B}} \Pr\{T \text{ is hit}\}. \tag{6.28}$$

In general, different triangles will have different probabilities. If $\{a, b, c\}$ is a bad triangle, let $T_{\{a,b,c\}}$ be the event that c is chosen as pivot while $\{a, b, c\}$ are still all part of the graph. Because of the random choice of the pivot it is true that

$$p_{abc} = Pr\{T_{\{a,b,c\}}\} = p_{acb} = p_{bca}. \tag{6.29}$$

Define q_{abc} to be $Pr\{T_{\{a,b,c\}}\}$. The three events (permutations of a, b and c) are mutually independent, since one excludes the other. The probability that the triangle $\{a, b, c\}$ is hit is equal to $3q_{abc}$.

$$E[\text{Cost of GREEDY-CC}] = \sum_{T \in \mathcal{B}} Pr\{T \text{ is hit}\} = 3 \sum_{T \in \mathcal{B}} q_T. \tag{6.30}$$

Why do we care about q_T? As we will see, $3 \sum_{T \in \mathcal{B}} q_T$ will actually be 3 times of the solution of the dual of the following LP problem. Equation 6.30 will be a 3 approximation, because the OPT has a cost that is at most the sum of the q_Ts.

Let us now introduce a linear relaxation of the correlation clustering problem.

$$\text{LP} = \begin{cases} \min \sum_{\{v,w'\} \in \binom{V}{2}} X_{\{v,v'\}} \\ X_{\{v,v'\}} + X_{\{v,v''\}} + X_{\{v',v''\}} \geq 1 \ \forall \{v, v', v''\} \in \mathcal{B} \\ X_{\{v,v'\}} \geq 0. \end{cases} \tag{6.31}$$

Each variable $X_{\{v,v'\}}$ tells me if the edge $\{v, v'\}$ is wrongly placed. So we want to minimize the number of this kind of edges in the solution. But whatever clustering is, a bad triangle makes us pay at least one.

Lemma 6.5. *Lp is a relaxation of cc. In other words, $LP^\star \leq OPT$, where OPT is the correlation clustering optimum.*

Proof. The Lemma is true because, whatever the CC optimal solution is, for each pair of nodes in it, the corresponding edge can be either correctly or incorrectly placed.

$$X_{\{v,v'\}} = \begin{cases} 1 & \text{if } \{v, v'\} \text{ has a cost of 1 in OPT,} \\ 0 & \text{otherwise.} \end{cases} \tag{6.32}$$

It satisfies the only constraint, since in every solution the cost of every bad triangle is at least one. □

Recall that $\text{DUAL}^\star \leq LP^\star$, if LP is a minimization problem. The dual problem is the following.

$$\text{DUAL} = \begin{cases} \max \sum_{T \in \mathcal{B}} Y_T \\ \sum_{T \in \mathcal{B}: \{v,v'\} \subseteq T} Y_T \leq 1 \quad \forall \{v, v'\} \in \binom{V}{2} \\ Y_T \geq 0. \end{cases} \tag{6.33}$$

Here, we are assigning a weight to each bad triangle so that no edge gets a total weight more than one.

Theorem 6.5. *$Y_T = q_T$ is a feasible solution for the dual with value $\sum_{T \in \mathcal{B}} q_T$.*

Proof.

$$\sum_{T \in B: \{v,v'\} \subseteq T} q_T = \sum_{v'': \{v,v',v''\} \in \mathcal{B}} q_{vv'v''} \tag{6.34}$$

$$= \sum_{v'': \{v,v',v''\} \in \mathcal{B}} p_{vv'v''} \tag{6.35}$$

$$= \sum_{v'': \{v,v',v''\} \in \mathcal{B}} Pr\left\{T_{\{v,v',v''\}}\right\} \tag{6.36}$$

$$\leq 1. \tag{6.37}$$

□

Finally we can claim the following corollary that completes the proof.

Corollary 6.2. $\sum_{T \in \mathcal{B}} q_T \leq DUAL^\star \leq LP^\star \leq OPT.$

□

Chapter 7

Locality Sensitive Hashing

Locality Sensitive Hashing (LSH) is an algorithmic scheme used to reduce the dimensionality of high dimensional data and to find the similar objects. An example of LSH that we will see was introduced in the Altavista search engine back in the Nineties to recognize duplicate pages. As an example consider a web page as a subset of the *universe* U, the set of all the English words. If we assign each word an integer, the universe is the set of the fist N numbers. So, given two sets $A, B \subseteq U$, we want to compute the degree of similarity between them.

7.1 Jaccard and Hamming similarities

We now introduce two similarities over sets, the Jaccard similarity and Hamming similarity.

Definition 7.1. *The Jaccard similarity between two sets A and B is defined as*

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}. \tag{7.1}$$

Think of a page as a set of words. We want to claim that two pages are the same or very similar if their Jaccard similarity is high. It easy to show that the Jaccard similarity is always between 0 and 1.

Definition 7.2. *Given two sets $A, B \subseteq [n]$, the Hamming similarity, or Hamming distance, between A and B is defined as*

$$H(A, B) = \frac{|A \cap B| + |\overline{A \cup B}|}{n}. \tag{7.2}$$

So this similarity gives an additive factor of $\frac{1}{n}$ for each element that is in both sets and for each element which is not in either sets. We can see A and B as characteristic vectors this time, where there is a 1 at the i-th position if i is part of the set, 0 otherwise. There is an increase in similarity whenever the same coordinate i leads to the same value

in both vectors. People haven't used Hamming similarity to compare web pages because these vectors would be huge and most elements would be zero, since the number of words in a web page is much less than the number of words in the English dictionary.

Now we are going to give an *LSH scheme* for each similarity introduce. We haven't defined what is a LSH yet, but for now think of it as an algorithm which takes a set and reduces it to a small number, i.e. $O(\log(n))$, bits; this operation is called *sketching*. Obviously, in order to compress so much information we have to lose something; but we will keep enough information to get a sense of what the similarity of two "fingerprints" is.

7.1.1 Shingles

The LSH for the Jaccard similarity is called *Shingles*. What it does is the following:

1. Sample a permutation π of $[n]$ uniformly at random.
2. Define the hash function $h_\pi : 2^U \to U$ as

$$h_\pi(S) = \text{the element of S having minimum rank in } \pi. \tag{7.3}$$

3. As sets S are shown to the algorithm, sketch them to $h_\pi(S)$.

As an example, consider the set $S = \{1, 2, 5\}$ and the permutation

$$\pi = 3 < 2 < 4 < 5 < 1.$$

Then, $h(S) = 2$.

Bibliography

[1] Abrahao et al. “Trace Complexity of Network Inference”. In: *Knowledge Discovery and Data Mining* (2013).

[2] Chierichetti, Kleinberg, and Liben-Nowell. “Reconstructing Patterns of Information Diffusion from Incomplete Observations”. In: *Advances in Neural Information Processing Systems* (2011).

[3] Gomez-Rodriguez, Leskovec, and Krause. “Inferring Networks of Diffusion and Influence”. In: *Knowledge Discovery and Data Mining* (2010).

[4] Nemhauser, Wolsey, and Fisher. “An analysis of approximations for maximizing submodular set functions”. In: *Mathematical Programming* (1978).

[5] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. URL: `http://www.designofapproxalgs.com/`.