Advanced Algorithms

Valerio Venanzi

Abstract

This document contains notes taken during the *Advanced Algorithms* course, held by Flavio Chierichetti during the academic year 2022-23.

Contents

1 Introduction **2**
1.1 Categorization of algorithms . . . 2
1.2 Graphs . . . 2
1.3 Approximations . . . 2
1.4 Math and CS notions . . . 3

2 Problems on graphs **3**
2.1 Max-Cut problem . . . 3
2.2 Vertex Cover problem . . . 5

3 Mathematical programming **8**
3.1 Some examples . . . 9
3.2 Hardness of LPs . . . 9
3.3 Solution for Vertex Cover . . . 9
3.4 Set Cover problem . . . 12
3.5 Densest subgraph problem . . . 15
3.6 Primal and Dual . . . 18
3.7 Greedy algorithms for Densest Subgraph . . . 21

4 Math and CS notions **23**
4.1 Tail inequalities . . . 23
4.1.1 Markov inequality . . . 24
4.2 NP . . . 24
4.2.1 NP-Hardness . . . 25
4.2.2 NP-Completeness . . . 25
4.3 Graphs . . . 25
4.3.1 Induced subgraph . . . 25

1 Introduction

1.1 Categorization of algorithms

Algorithms can be categorized by a number of criteria:

- Whether the input is all available at the beggining or not
  - Offline
  - Online
- Whether some choices are randomized or not
  - Deterministic
  - Randomized
  - Both can be quantum algorithms
- How good the algorithm does on some inputs
  - Worst-case input
  - Average-case input
- Whether we distribute our algorithms execution among multiple processors or not
  - Single processor
  - Parallel/Distributed

1.2 Graphs

If S is a set and k is a non-negative integer, then

$$\binom{S}{k} = \{T \mid T \subseteq S \wedge |T| = k\}$$

The cardinality of such set is $|\binom{S}{2}| = \frac{|S|!}{k!(|S|-k)!} = \binom{|S|}{k}$

Many algorithms seen in this course solve problems on graphs. A graph is $G(V, E)$, where V is the set of vertices and E, the set of edges, is $E \subseteq \{\{u, v\} \mid u, v \in V \wedge u \neq v\} = \binom{V}{2}$.

1.3 Approximations

Most problems during the course are going to be approached in the following manner. We have a problem that is NP-Hard (or perhaps NP-Complete), so we don't know how to solve it efficiently. What we know how to do is to solve such problems efficiently but approximating the solution; that is, the solution proposed by our algorithm is worse than the optimal solution to the problem.

How do we denote such approximations? If a problem is a *maximization* problem (e.g. a set that has some property and its cardinality is as large as possible) then our approximation algorithm will return something that is smaller than the optimal. In this case we denote the approximation as a number smaller than 1, possibly a fraction. For instance, a $\frac{1}{2}$-approximation means that the solution is at least half the optimal solution (e.g. if the optimal is 10, then our solution is at least 5).

If a problem is a minimization problem, then we reason in an analogous way. Our algorithm outputs a solution larger than the optimal. For instance, a 2-approximation means that our solution is at most twice as large as the optimal one.

The approach that will be used in many cases is the following. Given a hard problem, we give a randomized algorithm that approximates, in polynomial time, the solution. The random choices of an algorithm can yield to a solution that not only it's an approximation, even worse it can be wrong. Wwe then prove that the probability that the solution is actually wrong is so small, that we can ignore it.

$\boxed{\text{Check how above will hold for future algorithms.}}$

1.4 Math and CS notions

There are some notions about mathematics or computer science, used during the course, that are not part of the program, but are strictly related to many of the things seen, and may be worth explaining quickly.

Such notions are explained at the end of this document, in section Math and CS notions.

2 Problems on graphs

2.1 Max-Cut problem

Input: undirected graph $G(V, E)$

Question: what is the bipartition $(S, V \setminus S)$ of V that maximizes $|Cut(S, V \setminus S)|$?

Where a *cut* is defined as follows:

$$Cut(S, V \setminus S) = \{e \mid e \in E \wedge e \cap S \neq \emptyset \wedge e \cap (V \setminus S) \neq \emptyset\} = \{e \mid e \in E \wedge |e \cap S| = 1\}$$

This problem is NP-Hard, thus we don't know how to solve it efficiently. There exists a $(0.878\ldots)$-approximation based on Semi-Definite Programming. We now see a simple $\frac{1}{2}$-approximation.

Theorem 2.1. If S^* is an optimal solution to Max-Cut on $G(V, E)$, and S is the set returned by Random-Cut, then

$$\mathbb{E}[|Cut(S, V \setminus S)|] \geq \frac{1}{2}|Cut(S^*, V \setminus S^*)|$$

Lemma 2.2. Let S be the set returned by Random-Cut, then $\forall e \in E$, $Pr[e \in Cut(S, V \setminus S)] = \frac{1}{2}$.

Algorithm 1 Approximation of Max-Cut

```
procedure Random-Cut($V, E$)
    $S \leftarrow \emptyset$
    for each $v \in V$ do
        flip an independent/fair coin $c_v$
        if $c_v$ is heads then
            $S \leftarrow S \cup \{v\}$
        end if
    end for
    return $S$
end procedure
```

Proof. By definition, $e \in Cut(S, V \setminus S)$ iff $|S \cap e| = 1$.

If $e = \{v, w\}$, $e \in Cut(S, V \setminus S)$ iff $((v \in S \wedge w \notin S) \vee (v \notin S \wedge w \in S))$.

Let $\xi_1 =$ "$v \in S \wedge w \notin S$" and $\xi_2 =$ "$v \notin S \wedge w \in S$".

$Pr[\xi_1] = Pr[c_v = \text{Heads} \wedge c_w = \text{Tails}] =$ (by independence of the coins) $Pr[c_v = \text{Heads}] \cdot Pr[c_w = \text{Tails}] = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$

With same reasoning, $Pr[\xi_2] = Pr[c_v = \text{Tails}] \cdot Pr[c_w = \text{Heads}] = \frac{1}{4}$.

$Pr[e \in Cut(S, V \setminus S)] = Pr[\xi_1 \vee \xi_2] = Pr[\xi_1] + Pr[\xi_2] - Pr[\xi_1 \wedge \xi_2] = \frac{1}{4} + \frac{1}{4} + 0 = \frac{1}{2}$ □

Lemma 2.3. $\forall T \subseteq V$, $Cut(T, V \setminus T) \subseteq E$, $|Cut(T, V \setminus T)| \leq E$.

Proof. Trivial. □

Corollary 2.3.1. Let S be the set of nodes returned by Random-Cut,

$$\mathbb{E}[|Cut(S, V \setminus S)|] = \frac{|E|}{2}$$

Proof. By Lemma 2.2, $\forall e \in E$ $Pr[e \in Cut(S, V \setminus S)] = \frac{1}{2}$.

$\mathbb{E}[|Cut(S, V \setminus S)|] = \sum_{e \in E} Pr[e \in Cut(S, V \setminus S)] = \sum_{e \in E} \frac{1}{2} = \frac{|E|}{2}$ □

Recall Theorem 2.1.

Proof. The proof follows from Lemma 2.2 and Corollary 2.3.1. □

We now want to exploit a technique often used to improve the expected output of a randomized algorithm. We run the algorithm not once, but multiple times, and get the best result.

Note that, at the end of the algorithm, we don't ask to return *the* largest cut but *a* largest cut, since it could happen that there are multiple cuts with the same cardinality. In fact, it can even happen, in general, that all the cuts found have the same cardinality.

Algorithm 2 Repetition of Random-Cut

```
for $i = 1$ to $t$ do
    run Random-Cut$(V, E)$ independently
    let $S_i, V \setminus S_i$ be the resulting cut
    let $C_i = |Cut(S, V \setminus S)|$
end for
return a largest cut
```

Now, for $i \in [t] = \{1, 2, \dots, t\}$, let us define $N_i = |E| - C_i$ $(N_i \geq 0)$. N_i would be our error, the number of edges we didn't cut.

Let $0 < \varepsilon < 1$.

$Pr[N_i \geq (1+\varepsilon)\mathbb{E}[N_i]] \overset{\text{(by Markov inequality)}}{\leq} \dfrac{1}{1+\varepsilon} = \dfrac{1+\varepsilon}{1+\varepsilon} - \dfrac{\varepsilon}{1+\varepsilon} = 1 - \dfrac{\varepsilon}{1+\varepsilon} \leq 1 - \dfrac{\varepsilon}{2}$

Note that $|E| = 2\mathbb{E}[C_i]$.

$Pr[N_i \geq (1+\varepsilon)\mathbb{E}[N_i]] = Pr[|E| - C_i \geq (1+\varepsilon)(|E| - \mathbb{E}[C_i])] = Pr[|E| - (1+\varepsilon)|E| \geq C_i - (1+\varepsilon)\mathbb{E}[C_i]] = Pr[-\varepsilon|[|E] \geq C_i - (1+\varepsilon)\mathbb{E}[C_i]] = Pr[-2\varepsilon\mathbb{E}[C_i] + (1+\varepsilon)\mathbb{E}[C_i] \geq C_i] = Pr[C_i \leq (1-\varepsilon)\mathbb{E}[C_i]]$

$C_i \leq (1-\varepsilon)\mathbb{E}[C_i]$ is our "bad event".

$C_i > (1-\varepsilon)\mathbb{E}[C_i]$ is our "good event". $Pr[C_i > (1-\varepsilon)\mathbb{E}[C_i]] \geq \frac{\varepsilon}{2}$.

Suppose we run for $t = \lceil \frac{2}{3} \ln \frac{1}{\delta} \rceil$ (δ will be the probability of error).

$Pr[\forall i \in [t], C_i \leq (1-\varepsilon)\mathbb{E}[C_i]] \overset{\text{(by ind. of the } t \text{ runs)}}{=} \prod_{i=1}^{t} Pr[C_i \leq (1-\varepsilon)\mathbb{E}[C_i]] \leq \prod_{i=1}^{t}(1 - \frac{\varepsilon}{2}) = (1 - \frac{\varepsilon}{2})^t \leq (e^{-\frac{\varepsilon}{2}})^t = e^{-\frac{\varepsilon}{2}t} \leq e^{-\frac{\varepsilon}{2}\frac{2}{\varepsilon}\ln\frac{1}{\delta} = e^{-\ln\frac{1}{\delta}}} = \delta$

2.2 Vertex Cover problem

Input: undirected graph $G(V, E)$.

Output: $S \subseteq V$, of smallest cardinalityt, s.t. $\forall e \in E \; e \cap S \neq \emptyset$.

In other words, we want a subset of the vertices s.t. *every* edge of the graph touches at least one of the chosen vertices.

It is a *minimization* problem; note in fact that we could take every vertex (i.e. $S = V$) and this would clearly cover every edge.

It is an NP-Complete problem.

The way we solve this problem is by actually solving another problem; in fact, we find a *maximal matching*, that happens to be a VC.

Definition 2.1 (Matching)**.** A matching of the graph $G(V, E)$ is a subset $A \subseteq E$ s.t. $\forall \{e, e'\} \in \binom{A}{2}$, $e \cap e' = \emptyset$.

Definition 2.2 (Maximal matching)**.** A maximal matching of $G(V, E)$ is a subset $A \subseteq E$ s.t.

Algorithm 3 Approximation to VC

```
procedure MaximalMatching($V, E$)
    $S \leftarrow \emptyset$
    while $E \neq \emptyset$ do
        pick $e = \{u, v\} \in E$
        $S \leftarrow S \cup e$
        remove all the edges incident on $v$ and $w$ from $E$
    end while
    return $S$
end procedure
```

- it is a matching
- $\forall e \in E \setminus A$, $A \cup \{e\}$ is not a matching

Note that there is a distinction between *maximal* and *maximum* matching. A maximum matching is a matching with the largest possible cardinality. A maximal matching is a matching s.t. we can't add any more edges to it; so if we do so, the set is not a matching anymore.

So it could happen that a matching is maximal, but there exists another matching with larger cardinality.

Lemma 2.4. If $e_1, \ldots, e_t$ are the edges selected by the MaximalMatching algorithms, then $\{e_1, \ldots, e_t\}$ is a maximal matching.

Proof. For brevity, let us call $A = \{e_1, \ldots, e_t\}$. Suppose by contradiction that A is not a maximal matching.

So, either A is not a matching at all, or is not maximal, so there exists an edge removed by the algorithm that should have been added to the set.

When the algorithms picks an edge e, it removes all the edges connected to both endpoints of e. This ensures that $\forall e_1, e_2 \in A$, $e_1 \cap e_2 = \emptyset$. So we know that the set is indeed a matching, and now we have to check if it is maximal.

If it's not maximal, then $\exists e \in E \setminus A$ that has been wrongfully removed by the algorithm. But, following the same reasoning used to show that A is a matching, we know that e shares some endpoints with at least one of the selected edges; i.e. $\exists e_A \in A$ s.t. $e \cap e_A \neq \emptyset$.

So, adding any of the edges removed by the algorithm, would "break" the matching. Thus A is also maximal. □

Lemma 2.5. If MaximalMatching selects t edges, then $|S| = 2t$.

Proof. Trivial. □

Lemma 2.6. Let A be any matching of $G(V, E)$. If S is a VC of $G(V, E)$, then $|S| \geq |A|$.

Proof. First note that a VC for $G(V,E)$ is also a VC for $G(V,B)$, $\forall B \subseteq E$.

Now, pick any $A \subseteq E$. Since, by assumption, S is a VC for $G(V,E)$, then S must also cover each edge in A; this S is a VC for $G(V,A)$.

The graph $G(V,A)$ can only have nodes of degree at most 1 (i.e. either 0 or 1). In fact, suppose A has a node of degree > 1, say 2; then A would have two edges that share a node, thus it wouldn't be a matching.

So, from the point of view of VC, any node in $G(V,A)$ can cover at most one edge, since $\forall v \in V$, $\deg_{G(V,A)}(v) \leq 1$.

Thus, a set of, say, k nodes, can cover $\leq k$ edges of $G(V,A)$. Now, $G(V,A)$ has $|A|$ edges, thus each VC of $G(V,A)$ has to have at least $|A|$ nodes. □

Theorem 2.7. MaximalMatching returns a 2-approximation to VC.

Proof. Let A be the maximal matching produced by the MaximalMatching procedure (the one called $\{e_1, \ldots, e_t\}$ in Lemma 2.4).

This solution contains $|S| = 2|A|$ nodes (Lemma 2.5).

Since A is a matching, if S^* is an optimal solution (smallest solution), we know by Lemma 2.6 that $|S^*| \geq |A|$.

$|S^*| \geq |A| = \frac{|S|}{2} \to |S| \leq 2|S^*|$. □

There are reasons to believe that this is the best possible approximation. In fact, VC is *conjectured* to be NP-Hard to approximate to $2 - \varepsilon$ ($\forall \varepsilon > 0$ constant).

Although, good news, VC has a good property that lets us "easily" (under some conditions) find a solution. It is **fixed parameter tractable**. If $G(V,E)$ has a VC of k nodes, then an optimal VC of $G(V,E)$ can be found in time $n^c \cdot f(k)$; specifically, there's an algorithm that finds a VC in time $O(n^2 \cdot 2^k)$.

So, if the optimal solution is small enough (i.e. $\log n$), then this search only takes polynomial time.

The algorithm makes use of Induced subgraph.

Lemma 2.8. VC$(G(V,E),k)$ takes time $O(n^2 \cdot 2^k)$.

Proof. By induction, running VC$(G(V,E),l)$ causes at most $2^l - 1$ calls to the function VC.

Base case ($l = 0$). Only the first call to VC$(\cdot, 0)$ will be generated. $2^{l-1} - 1 = 2^1 - 1 = 1$.

Inductive step ($l + 1$). Assume the claim holds for l. The number of calls generated by VC$(\cdot, l+1)$ is equal to no more than twice the numner of calls generated by VC$(\cdot, l)$+1. The total number of calls, by induction, is then $\leq 1 + 2(2^{l+1} - 1) = 1 + 2^{l+2} - 2 = 2^{l+2} - 1$.

Each single casll takes time $O(n^2)$; there a few checkes that can be done in constant time, we have to fix an edge, and we have to build the induced subgraph

Algorithm 4 Search of a VC

```
procedure VC($G(V,E), k$)
    if $E = \emptyset$ then
        return true
    end if
    if $k = 0$ then
        return false
    else
        fix $\{u, v\} \in E$
        if VC($G[V \setminus \{u\}], k-1$) then
            return true
        end if
        if VC($G[V \setminus \{v\}], k-1$) then
            return true
        end if
    end if
    return false
end procedure
```

(which take $O(n^2)$). Note that, fixing an edge might take $O(n^2)$; worst case we use an adjacency matrix and we visit it until we find a fixable edge.

Thus, the claim follows. □

Lemma 2.9. VC$(G(V,E),k)$ == true iff $G(V,E)$ has a VC of size k.

Exercise 2.9.1. The proof of above lemma is left as an exercise

3 Mathematical programming

We can divide mathematical programs in:

- Linear Programming (LP)
- Semi-Definite Programming (SDP)

A mathematical program contains:

- varibles $x_1, \dots, x_n$ ($x_i \in \mathbb{R}$ or $x_i \in \mathbb{Q}$)
- objective function $f(x_1, \dots, x_n)$ (either to minimize or to maximize)
- constraint functions $g_i(x_1, \dots, x_n) \leq b_i$ (or $g_i(x_1, \dots, x_n) \geq b_i$), for $i = 1, \dots, n$

In a linear program, the objective function f is linear and each constraint function g_i is linear as well.

When we force each variable to be an integer, the progam is called **Integer Program**.

Definition 3.1 (Feasible solution)**.** A solution is said to be feasible if it satisfied all constraints.

3.1 Some examples

$$\begin{cases} \max x_1 + x_2 \\ x_1 \geq 3 \\ x_2 \geq 5 \\ x_1 \leq 2.5 \end{cases} \tag{1}$$

This program clearly has no solutions, since the constraints $x_1 \geq 3$ and $x_1 \leq 2.5$ can't be both satisfies ad the same time.

$$\begin{cases} \max x_1 + x_2 \\ x_2 \geq 5 \\ x_1 \leq 2.5 \end{cases} \tag{2}$$

There are infinitely many solutions, and the "optimal value" of this LP is unbounded (∞).

$$\begin{cases} \max x_1 + x_2 \\ 2x_1 + x_2 \leq 10 \\ x_1 \geq 0 x_2 \geq 0 \end{cases} \tag{3}$$

We can set $x_1 = 0$ and $x_2 = 10$. All the constraints are satisfied and our objective function would have value 10.

3.2 Hardness of LPs

If $f(x_1, \ldots, x_n) = \sum_{j=1}^{n} c_j x_j$ and $g_i(x_1, \ldots, x_n) = \sum_{j=1}^{n} a_{ij} x_j$, then if each coefficient c_j and a_{ij} and each term b_i is a rational number, representable with at most t bits, then the LP can be optimized in time $O(n \cdot m \cdot t)^c$, for come constant $c > 0$.

IPs are instead harder to solve than LPs. In general, we can't solve an IP in polynomial time. What can be done, and will be done later for some problems, is to give an IP program for a problem, that will be tranformed into an LP, easier to solve.

3.3 Solution for Vertex Cover

We now see a pratical use of LPs to solve the Vertex Cover problem.

Let our set of variables be $\{x_v \mid v \in V\}$. We define our IP, for a graph $G(V, E)$ to be the following:

$$\begin{cases} \min \sum_{v \in V} x_v \\ x_u + x_v \geq 1 \qquad \forall \{u, v\} \in E \\ x_v \in \{0, 1\} \qquad \forall v \in V \end{cases} \tag{4}$$

Solving this IP would give us an optimal VC.

Let $x_{v_1}^*, x_{v_2}^*, \ldots, x_{v_n}^*$ be an optimal solution to the IP. Define $S^* = \{v \mid v \in V \wedge x_v^* = 1\}$.

Theorem 3.1. *S^* is an optimal VC of $G(V, E)$.*

Lemma 3.2. *S^* is a vertex cover.*

Proof. S^* being a VC means that $\forall\{u, v\} \in E$, $u \in S^*$ or $v \in S^*$ or both (i.e. $u, v \in S^*$). Given that $x_u^* + x_v^* \geq 1$, $\forall\{u, v\} \in E$, and that $x_u^*, x_v^* \in B$, at least one of x_u^* and x_v^* is equal to 1. This $u \in S^*$ or $v \in S^*$. $\square$

Lemma 3.3. *If S is a VC, then there exists a feasible solution $\{x_v\}_{v\in V}$ to the IP, s.t. $\sum_{v\in V} x_v = |S|$.*

Proof. $x_v = 1 \leftrightarrow v \in S$. $\square$

Corollary 3.3.1. *An optimal solution to the IP has value equal to*

$$\min_{\substack{S\subseteq V \\ S \text{ is a VC}}} \quad |S|$$

Recall that IPs can't be solve, in general, in polytime, while LPs can. So we *relax* the IP for VC (see equation 4) into an LP.

$$\begin{cases} \min \sum_{v\in V} x_v \\ x_u + x_v \geq 1 & \forall\{u, v\} \in E \\ 0 \leq x_v \leq 1 & \forall v \in V \end{cases} \tag{5}$$

Note the the two programs are almost the same, except that, during the relaxation, we've allowed each variable from being in $\{0, 1\}$, to being in the range $[0, 1]$. Albeit this is not how IPs are always transformed into LPs, it is indeed quite often the case, and it is the way in which most (if not all) of the IPs during this course will be transformed into LPs.

It is easy transforming the IP into a VC solution: for a vertex v, $x_v = 1$ iff v is in the vertex cover. In the LP we have fractional values, so how do we choose vertices? We apply a **rounding rule**, basically an algorithm that, given an LP solutions, it outputs a VC set.

The rounding rule for this problem is quite simple. We define $S = \{v \mid x_v \geq \frac{1}{2} \wedge v \in V\}$.

Lemma 3.4. *S is a vertex cover.*

Proof. Let $\{u, v\} \in E$. The LP contains the constraint $x_u + x_v \geq 1$. Given that our solutions guarantees that $x_u + x_v \geq 1$ (the LP solution is feasible by definition).

$x_u + x_v \geq 1 \rightarrow \max(x_u, x_v) \geq \frac{x_u+x_v}{2} \geq \frac{1}{2}$.

Thus, $u \in S$, or $v \in S$ (or both), thus the edge $\{u, v\}$ is covered. $\square$

Lemma 3.5. *$|S| \leq 2\sum_{v\in V} x_v$.*

Proof. $|S| = \sum_{v\in S} 1 \overset{(v\in S \to x_v \geq \frac{1}{2})}{\leq} \sum_{v\in S} 2x_v = 2\sum_{v\in S} x_v \overset{(x_v \geq 0)}{\leq} 2\sum_{v\in V} x_v$ □

Recall that the LP minimizes $\sum_{v\in V} x_v$. Let $\{x_v^*\}_{v\in V}$ be an optimal solution to the LP. Let S^* be the result of the application of our rounding rule; i.e. $S^* = \{v \mid v \in V \wedge x_v^* \geq \frac{1}{2}\}$.

$$|S^*| \overset{\text{(Lemma 3.5)}}{\leq} 2\sum_{v\in V} x_v^* = 2LP^* \overset{\text{(LP relaxation of IP)}}{\leq} 2IP^* = 2 \min_{\substack{S\subseteq V \\ S \text{ is a VC}}} |S| = 2OPT$$

If Lemma 3.5 could be improved (to, say, $|S| \leq 1.9\sum_{v\in V} x_v$), then the approximation ration would be directly improved.

Let us define the **integrality gap** to be the ration between the optimal IP solution and the optimal LP solution. That is,

$$IG = \frac{IP^*}{LP^*}$$

Now, let us consider a particular graph s.t. we can use it show an upper bound on the integrality gap. This approach will be used also in the future: build a special instance of a problem, s.t. the instance can be used to show lower or upper bound to the optimality of approximations.

Consider the graph $G(V, E) = K_3$ (i.e. the complete graph with 3 vertices).

Lemma 3.6. Each optimal solution for $G(V, E)$ contains two nodes.

Lemma 3.7. $x_1 = x_2 = x_3 = \frac{1}{2}$ is a feasible solution of value $\frac{3}{2}$.

Proof. $\forall\{i, j\} \in E$, $x_i + x_j = 2\frac{1}{2} = 1 \geq 1$. Thus each constraint is satisfied.

The objective value is $3\frac{1}{2} = \frac{3}{2}$. □

Lemma 3.8. There exists no feasible LP solution of value $< \frac{3}{2}$.

Proof. For $\{i, j\} \in E$, $x_i + x_j \geq 1$.

$2\sum_{v\in V} x_v = 2x_1 + 2x_2 + 2x_3 = (x_1 + x_2) + (x_2 + x_3) + (x_1 + x_3) \overset{\text{feasibility}}{\geq} 1 + 1 + 1 = 3.$

Thus $\sum_{v\in V} \geq \frac{3}{2}$. □

Corollary 3.8.1. $LP^* = \frac{3}{2}$ and $IP^* = 2$.

Thus, for K_3, $IG = \frac{IP^*}{LP^*} = \frac{2}{\frac{3}{2}} = \frac{4}{3}$. So $IG \leq 2$.

Exercise 3.8.1. Show that $\forall\varepsilon > 0$, $\exists G(V, E)$ s.t. $\dfrac{OPT_{VC}(G(V, E))}{LP^*(G(V, E))} \geq 2 - \varepsilon$.

Hints:

- K_t, large t
- Prove the optimal VC for K_t has $t - 1$ nodes

- What is the optimal LP solution for K_t?

3.4 Set Cover problem

Input: $U = [n] = \{1, \dots, n\}$, $S_1, \dots, S_m \subseteq U$.

We call the input U, meaning our *universe* set.

Question: what is the size of a smallest class of subsets $T \subseteq [m]$ s.t. $\bigcup_{j \in T} S_j = U$?

Let us see an example. Let $U = [4] = \{1, 2, 3, 4\}$, and let $S_1 = \{1, 2\}$, $S_2 = \{2, 3\}$, $S_3 = \{3, 4\}$. A solution, even optimal, is $S_1 \cup S_3 = [4]$.

Set Cover (SC) is NP-Hard. We consider an LP solution to approximate it. We first see an IP problem, translated into an LP.

Let the IP be

$$\begin{cases} \min \sum_{j=1}^{m} x_j & (x_j = 1 \text{ if } S_j \text{ is in the solution}) \\ \sum_{j \in [m] \text{ s.t. } i \in S_j} x_j \geq 1 & \forall i \in [n] \\ x_j \in \{0, 1\} & \forall j \in [m] \end{cases} \tag{6}$$

The LP relaxation is

$$\begin{cases} \min \sum_{j=1}^{m} x_j \\ \sum_{j \in [m] \text{ s.t. } i \in S_j} x_j \geq 1 & \forall i \in [n] \\ 0 \leq x_j \leq 1 & \forall j \in [m] \end{cases} \tag{7}$$

The constraint $\sum_{j \in [m] \text{ s.t. } i \in S_j} x_j \geq 1$ means that each element of $[n]$ has to be picked at least once.

Observation 3.8.1. *$LP^* \leq IP^*$*

Proof. The LP is a relaxation to the IP (the constraint of the LP are less stringent than those of the IP). Thus each IP solution is also an LP solution. $\square$

We now need a rounding rule, to converto the LP solution into a solution for Set Cover. This rounding rule is a randomized procedure, less trivial than the one used for VC. Many rouding rules are actually randomized, like this one.

The first for loop is a repetition of "phases". It represents how many times do we want to repeat the subprocedure in the second loop, the one that loops over sets and adds them to the solution.

As it is going to be clearer later on, since this procedure is randomized, it is not always the case that the output A is indeed a set cover.

The number of times we repeat the first loop actually defines a sort of trade off. The more times we repeat it, the higher the probability that A is a set cover, but also the higher the probability that we put too many sets in the solution, leading to a solution worse than the optimal.

Algorithm 5 Randomized rounding technique

```
  $A \leftarrow \emptyset$
  let $x^*$ be an optimal LP solution
  for $k = 1$ to $\lceil 2\ln n \rceil$ do
      for $j = 1$ to $m$ do
          flip an independent coin with head probability $x_j^*$
          if the coin is heads then
              $A \leftarrow A \cup \{S_j\}$
          end if
      end for
  end for
```

Lemma 3.9. Consider a generic iteration k of the outer loop. Let p_i be the probability in iteration k that at least on of the sets containing i gets added to A. Then $p_i \geq 1 - \frac{1}{e} \approx 0.633\dots$.

Proof. To prove this lemma we are going to use that fact that $1 - x \leq e^{-x}$.

$Pr[i \text{ not covered in iteration } k] = \prod_{j\in[m]:i\in S_j}(1 - x_j^*) \leq \prod_{j\in[m]:i\in S_j} e^{-x_j^*} = e^{-\sum_{j\in[m]:i\in S_j}} \overset{\text{(feasibility of solution)}}{\leq} e^{-1}$.

Thus $Pr[i \text{ covered in iteration } k] \geq 1 - \frac{1}{e}$. □

Lemma 3.10. Pick $i \in [n]$. $Pr[i \text{ not covered by ant set in } A] \leq \frac{1}{n^2}$.

Proof. $Pr[i \text{ not covered by any set in } A] = \prod_{k=1}^{\lceil 2\ln n\rceil} Pr[i \text{ not covered in iteration } k] \leq \prod_{k=1}^{\lceil 2\ln n\rceil} \frac{1}{e} = e^{-\lceil 2\ln n\rceil} = n^{-2}$ □

Lemma 3.11. $Pr[A \text{ is a set cover}] \geq 1 - \frac{1}{n}$

Proof. $Pr[A \text{ is not a set cover}] \overset{\text{(UB)}}{\leq} \sum_{i=1}^{n} Pr[i \text{ not covered by } A] \overset{\text{(Lemma 3.10)}}{\leq} \sum_{i=1}^{n} \frac{1}{n^2} = \frac{n}{n^2} = \frac{1}{n}$ □

Lemma 3.12. $\mathbb{E}[|A|] \leq \lceil 2\ln n\rceil LP^* \leq \lceil 2\ln n\rceil OPT_{SC}$

Proof. Fix an iteration k of the outer loop. Let A_k be the class of sets added to A in iteration k.

$\mathbb{E}[|A_k|] \overset{\text{(linearity of } \mathbb{E})}{=} \sum_{i=1}^{m} x_j^* = LP^* \leq OPT_{SC}$

$A = A_1 \cup A_2 \cup \dots \cup A_k \cup \dots \cup A_{\lceil 2\ln n\rceil}$.

$|A| \leq \sum_{i=1}^{\lceil 2\ln n\rceil} |A_i|$. Then $\mathbb{E}[|A|] \leq \sum_{i=1}^{\lceil 2\ln n\rceil} \mathbb{E}[|A|] = \sum_{i=1}^{\lceil 2\ln n\rceil} LP^* \leq \lceil 2\ln n\rceil LP^* \leq \lceil 2\ln n\rceil OPT_{SC}$. □

So, let us consider the following statement

$$? \overset{\exists \text{ instance}}{\leq} \frac{OPT_{SC}}{LP^*_{SC}} \overset{\forall \text{ instance}}{\leq} \lceil 2\ln n\rceil$$

If we manage to find an instance with an IG smaller than $\frac{OPT_{SC}}{LP^*_{SC}}$, then we have proved a lower bound on the integrality gap of set cover.

We know of such an instance, that we shall see in a moment. As it is the case, the following will be proved

$$\frac{1}{4\ln 2}\ln n \overset{\exists \text{ instance}}{\leq} \frac{OPT_{SC}}{LP^*_{SC}} \overset{\forall \text{ instance}}{\leq} \lceil 2\ln n \rceil$$

An improvement, albeit a small one.

Let us consider the following universe set, $E = \{e_A \mid A \in \binom{[q]}{\frac{q}{2}},$ for some even $q \geq 2\}$.

For $n = |E| = \binom{q}{\frac{q}{2}} = \Theta(\frac{2^q}{\sqrt{q}}) \Rightarrow q \geq \log n - \Theta(\log\log n)$.

$\forall i \in [q]$, let us define $S_i = \{e_A \mid e_A \in E \wedge i \in A\}$ and $C = \{S_i \mid i \in [q]\}$. $|C| = q \overset{\Delta}{=} m$.

Let us see an example. Given $q = 4$:

- $E = \{e_{12}, e_{13}, e_{14}, e_{23}, e_{24}, e_{34}\}$
- $S_1 = \{e_{12}, e_{13}, e_{14}\}$
- $S_2 = \{e_{12}, e_{23}, e_{24}\}$
- $S_3 = \{e_{13}, e_{23}, e_{34}\}$
- $S_4 = \{e_{14}, e_{24}, e_{34}\}$

Lemma 3.13. $LP^*_{SC} \leq 2$

Proof. Consider the LP solution $_1 = x_2 = \cdots = x_q = \frac{2}{q}$, with $q = n$. This solution has value $\sum_{i=1}^{q} x_i = q\frac{2}{q} = 2$.

The generic LP constraint is $\sum_{s_j : e_a \in S_j} x_j \geq 1$, $\forall e_a \in E$.

$\sum_{s_j : e_a \in S_j} x_j = \sum_{s_j : e_a \in S_j} \frac{2}{q} = |A|\frac{2}{q} = \frac{q}{2}\frac{2}{q} = 1 \geq 1$. Thus the solution is feasible. □

Lemma 3.14. $OPT_{SC} \geq \frac{1}{2}\log_2 n - O(\log\log n)$

Proof. Assume by contradiction that the sets $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$, for $k \leq \frac{q}{2}$, cover each element of the instance. Consider also the set $T = [q] \setminus \{i_1, \ldots, i_k\}$. Then $\exists A \subseteq T$ s.t. $|A| = \frac{q}{2}$ (because $|T| \geq q - k \geq \frac{q}{2}$), then $e_A \in E$.

The element e_A is not covered by $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$, because $A \cap \{i_1, \ldots, i_k\} = \emptyset$. Then $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ is not a set cover if $k \leq \frac{q}{2}$.

It follows that the minimum set cover contains $\geq \frac{q}{2} + 1$ sets. □

Thus $IG = \frac{OPT_{SC}}{LP^*_{SC}} \overset{\text{(on some instances)}}{\geq} \frac{\frac{1}{2}\log_2 n - O(\log\log n)}{2} \approx \frac{1}{4}\log_2 n$

3.5 Densest subgraph problem

Input: $G(V, E)$

Question: what is the subset $S \subseteq V$ having maximum **density**?

We define density as follows:

$$\rho(S) = \frac{|E(S)|}{|S|}$$

Where $E(S) = \{\{u, v\} \mid \{u, v\} \in E \wedge u, v \in S\}$.

A variation of the problem is the following. For k-densest subgraph we want $S \subseteq V$, $|S| = k$, s.t. $|E(S)|$ is as large as possible.

To solve this problem, we give an LP program.

$$\begin{cases} \max \sum_{\{i,j\} \in E} x_{\{i,j\}} & \\ x_{\{i,j\}} \leq y_i & \forall \{i,j\} \in E \\ x_{\{i,j\}} \leq y_j & \forall \{i,j\} \in E \\ \sum_{i \in V} y_i \leq 1 & \\ x_{\{i,j\}} \geq 0 & \forall \{i,j\} \in E \\ y_i \geq 0 & \forall i \in V \end{cases} \tag{8}$$

Lemma 3.15. For any $G(V, E)$, $\forall S \subseteq V$, $\exists$ feasible solution to the LP, having value $\geq \frac{|E(S)|}{|S|} = \rho(S)$.

Proof. For $i \in S$, set $y_i = \frac{1}{|S|}$. For $i \in V \setminus S$, set $y_i = 0$. For each $\{i, j\} \in E$, set

$$x_{\{i,j\}} = \begin{cases} \frac{1}{|S|} & \text{if } \{i,j\} \in E(S) \\ 0 & \text{o/w} \end{cases} \tag{9}$$

The constraint $\sum_{i \in V} y_i \leq 1$ is satisfied, given that $\sum_{i \in V} y_i = \sum_{i \in S} y_i + \sum_{i \in V \setminus S} y_i = \sum_{i \in S} \frac{1}{|S|} + \sum_{i \in V \setminus S} 0 = \frac{|S|}{|S|} = 1$.

Take any $\{i, j\} \in E(S)$. We have $x_{\{i,j\}} = \frac{1}{|S|}$. But, if $\{i, j\} \in E(S)$, then $i, j \in S$. Thus $y_i = y_j = \frac{1}{|S|}$. Then, $\forall \{i, j\} \in E(S)$, the constraints $x_{\{i,j\}} \leq y_i$ and $x_{\{i,j\}} \leq y_j$ are both satisfied.

If instead $\{i, j\} \notin E(S)$, then $x_{\{i,j\}} = 0$. Thus $x_{\{i,j\}} \leq y_i$ and $x_{\{i,j\}} \leq y_j$ are both satisfied by $y_i, y_j \geq 0$.

Thus, our solution is feasible.

The value of the solution is:

$$\sum_{\{i,j\} \in E} x_{\{i,j\}} = \sum_{\{i,j\} \in E(S)} x_{\{i,j\}} = \sum_{\{i,j\} \in E(S)} \frac{1}{|S|} = \frac{|E(S)|}{|S|} = \rho(S)$$

□

Lemma 3.16. For any feasible LP solution of value v, $\exists S \subseteq V$ s.t. $\rho(S) \geq v$.

Proof. Let Y', X' be a feasible LP solution of value v. Let Y, X be the solution to the LP s.t.

- $y_i = y'_i$, $\forall i \in V$
- $x_{\{i,j\}} = \min(y_i, y_j)$, $\forall \{i,j\} \in E$

The Y, X solution is feasible, indeed:

- $\sum_{i \in V} y_i = \sum_{i \in V} y'_i \leq 1$
- $\forall \{i,j\} \in E$, it holds $x_{\{i,j\}} \leq y_i$ and $x_{\{i,j\}} \leq y_j$

We have to provide a set S having density at least v. We are going to provide a number of S's, one of which will have the desired property. $\forall r \geq 0$, let us consider

$$S(r) = \{i \mid y_i \geq r\}$$

$$E(r) = \{\{i,j\} \mid x_{\{i,j\}} \geq r\}$$

We can observe that $\{i,j\} \in E(r) \leftrightarrow i \in S(r)$ and $j \in S(r)$. The proof is left as an exercise.

We claim that $\exists r \geq 0$ s.t. $\rho(S(r)) = \dfrac{|E(r)|}{|S(r)|} \geq \sum_{\{i,j\} \in E} x_{\{i,j\}} = v$.

Proof of claim. We define a permutation π of the nodes s.t.

$$0 \leq y_{\pi(1)} \leq y_{\pi(2)} \leq \cdots \leq y_{\pi(n-1)} \leq y_{\pi(n)} \leq 1$$

$\int_0^1 |E(r)|dr = \int_0^{y_{\pi(1)}} |S(r)|dr + \int_{y_{\pi(1)}}^{y_{\pi(2)}} |S(r)|dr + \cdots + \int_{y_{\pi(n)}}^1 |S(r)|dr = \int_0^{y_{\pi(1)}} n\,dr + \int_{y_{\pi(1)}}^{y_{\pi(2)}} (n-1)dr + \cdots + \int_{y_{\pi(n-1)}}^{y_{\pi(n)}} 1dr + \int_{y_{\pi(n)}}^1 0dr = (y_{\pi(1)} - 0)n + (y_{\pi(2)} - y_{\pi(1)})(n-1) + \cdots + (y_{\pi(n)} - y_{\pi(n-1)})1 + 0 = y_{\pi(1)}(n-(n-1)) + y_{\pi(2)}((n-1)-(n-2)) + \cdots + y_{\pi(n)}(1-0) = \sum_{i=1}^n y_{\pi(i)} = \sum_{i=1}^n y_i \leq 1.$

$\int_0^1 |E(r)|dr = \sum_{\{i,j\} \in E} x_{\{i,j\}} = \sum_{\{i,j\} \in E} \min(y_i, y_j) = \sum_{\{i,j\} \in E} \min(y'_i, y'_j) \geq \sum_{\{i,j\} \in E} x'_{\{i,j\}} = v$

Then, $\int_0^1 |S(r)|dr \leq 1$ and $\int_0^1 |E(r)|dr \geq v$.

By contradiction, supopose that $\forall r \geq 0$, $\dfrac{|E(r)|}{|S(r)|} < v$. Then $|E(r)| < v|S(r)|$, $\forall r \geq 0$. Thus $v \leq \sum_{\{i,j\} \in E} x_{\{i,j\}} = \int_0^1 |E(r)|dr < \int_0^1 v|S(r)|dr = v \int_0^1 |S(r)|dr \leq v \cdot 1 = v$.

Since $v \not< v$, we have a contradiction. Thus $\exists r \geq 0$ s.t. $|E(r)| \geq v|S(r)| \rightarrow \rho(S(r)) \geq v$. $\square$

Thus, we can get an optimal (densest) subgraph (one of $S(y_1), \ldots, S(y_n)$ will do). We can solve the LP in polytime, and we can transform the LP solution in an optimal subgraph. Is this fast enough? No, the LP has $\geq |E|$ variables.

Corollary 3.16.1. $LP^* = OPT_{DS}$

Proof. By Lemma 3.15, $LP^* \geq OPT_{DS}$. By Lemma 3.16, $LP^* \leq OPT_{DS}$. $\square$

The LP for Densest Subgraph has $\Theta(n^2)$ variables, so albeit being polynomially solvable, the number of variables is not linear. For n very large (e.g. $10^8 \dots 10^9$ nodes), n^2 is too slow.

We now see a greedy algorithm, due to M. Charikar, to approximate Densest Subgraph.

Algorithm 6 Charikar algorithm

```
procedure GREEDY($G(V, E)$)
    $S \leftarrow V$
    for $i = 1, 2, \dots, n$ do
        let $v_i$ be a node having minimum degree in $G[S_{i-1}]$
        $S_i \leftarrow S_{i-1} \setminus \{v_i\}$
    end for
    return a set $S_i$ having larget density $\rho(S_i)$
end procedure
```

Exercise 3.16.1. Prove that Greedy does not always return an optimal solution.

Hint: construct a graph s.t. the algorithm returna a non optimal solution.

Definition 3.2 (Orientation)**.** An orientation ϕ of the edges of an undirected graph $G(V, E)$ is a function that assigns to each $e \in E$ one of its endpoints

$$\phi : E \rightarrow V$$

Note that $\phi(e) \in e$.

Definition 3.3. Given an orientation ϕ of $G(V, E)$, let $d_\phi(v)$ be the in-degree of $v \in V$, according to ϕ.

Observation 3.16.1. If ϕ is an orientation of $G(V, E)$, then $\sum_{v \in V} d_\phi(v) = |E|$.

Definition 3.4. The max-degree of ϕ is $\Delta_\phi = \max_{v \in V} d_\phi(v)$.

Lemma 3.17. $\max_{\emptyset \subset S \subseteq V} \rho(S) = \max_{\emptyset \subset S \subseteq V} \dfrac{|E(S)|}{|S|} \leq \Delta_\phi$, $\forall \phi$.

Proof. Let $\emptyset \subset S \subseteq V$ be given. Each $\{u, v\} \in E(S)$ is going to be oriented towards on of its endpoints; that is, towards a node of S. So $|E(S)| \leq \sum_{v \in S} d_\phi(v) \leq \sum_{v \in S} \Delta_\phi = |S| \Delta\phi$.

Thus, $\dfrac{|E(S)|}{|S|} \leq \Delta_\phi$. $\square$

While running the greedy algorithms, we *build* an orientation. Whenever node v_i (the i^{th} node to be removed) is removed from the graph (which, at that point, has node set S_{i-1}) we orient each edge of $G[S_{i-1}]$ that is incident on v_i towards v_i. The resulting orientation is called ϕ_{GR}.

Observe that, with orientation ϕ_{GR}, $d_{\phi_{GR}}(v)$ equals the degree of v in the graph, when v is about to be removed.

Lemma 3.18. Let M be the maximum $\rho(S_i)$ value; i.e. $M = \max_{i=0,\dots,n-1} \rho(S_i)$. (Note that M is the value of the greedy solution).

Then, $\Delta_{\phi_{GR}} \leq 2M$.

Proof. For $i = 1, 2, \dots, n$, v_i is a node of minumum degree in $G[S_{i-1}]$. In general, the minumum degree in a graph is not larger than the average degree of the same graph. Thus $d_{\phi_{GR}}(v_i) = \min_{v \in S_{i-1}} d_{S_{i-1}}(v) \leq avg_{v \in S_{i-1}} d_{S_{i-1}}(v) = \dfrac{\sum_{v \in S_{i-1}} d_{S_{i-1}}(v)}{|S_{i-1}|} = \dfrac{2|E(S_{i-1})|}{|S_{i-1}|} = 2\rho(S_{i-1})$.

Then

$$\Delta_{\phi_{GR}} = \max_{v_i \in V} d_{\phi_{GR}}(v_i) = \max_{i=1,\dots,n} d_{\phi_{GR}}(v_i) \leq \max_{i=1,\dots,n} 2\rho(S_{i-1}) = 2M$$

□

Corollary 3.18.1. Greedy returns a 2-approximation to Densest Subgraph.

Proof.

$$M \geq \frac{\Delta_{\phi_{GR}}}{2} \geq \frac{\max_{\emptyset \subset S \subseteq V} \rho(S)}{2}$$

Thus the greedy solution is not worse than a 2-approximation. □

3.6 Primal and Dual

We can define an LP, in general, as a **primal** in the following way:

$$\begin{cases} \max c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ (y_1) a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1 \\ (y_2) a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leq b_2 \\ \vdots \\ (y_2) a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leq b_m \\ x_1, \dots, x_n \geq 0 \end{cases} \tag{10}$$

The **dual** of the above LP is:

$$\begin{cases} \min b_1y_1 + b_2y_2 + \cdots + b_my_m \\ (x_1) a_{11}y_1 + a_{21}y_2 + \cdots + a_{m1}y_m \geq c_1 \\ (x_2) a_{12}y_1 + a_{22}y_2 + \cdots + a_{m2}y_m \geq c_2 \\ \vdots \\ (x_n) a_{1n}y_1 + a_{2n}y_2 + \cdots + a_{mn}y_m \geq c_n \\ y_1, \dots, y_m \geq 0 \end{cases} \tag{11}$$

The fact that a primal is always defined as a maximization problem, and a dual as a minimization problem, is conventional.

Now, we can also define these programs in matrix form.

If

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \tag{12}$$

$$b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} \tag{13}$$

$$c = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} \tag{14}$$

$$X = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \tag{15}$$

$$Y = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \tag{16}$$

Then the **primal** is

$$\begin{cases} \max c^T X \\ AX \leq b \\ X \geq 0 \end{cases} \tag{17}$$

And the **dual** is

$$\begin{cases} \min b^T Y \\ A^T Y \geq c \\ Y \geq 0 \end{cases} \tag{18}$$

Recall that, if N and M are matrices, then $(NM)^T = M^T N^T$.

Theorem 3.19 (Weak Duality)**.** If

- X is a feasible solution to the primal
- Y is a feasible solution to the dual

Then

$$Primal(C) = c^T X \leq b^T Y = Dual(Y)$$

Proof.

$$c^T X = X^T c \leq X^T A^T Y = (AX)^T Y \leq b^T Y$$

□

Recall Densest subgraph problem LP formulation. Let us see it as a Primal LP (in "standard form").

$$\begin{cases} \max \sum_{\{i,j\} \in E} x_{\{i,j\}} & \\ (y_{ij}) : x_{\{i,j\}} - x_i \leq 0 & \forall \{i,j\} \in E \\ (y_{ji}) : x : \{i,j\} - x_j \leq 0 & \forall \{i,j\} \in E \\ (y^*) : \sum_{i \in V} x_i \leq 1 & \\ x_{\{i,j\}} \geq 0 & \forall \{i,j\} \in E \\ x_i \geq 0 & \forall i \in V \end{cases} \tag{19}$$

Let us now consider the equivalent Dual LP.

$$\begin{cases} \min y^* \\ (x_{\{i,j\}}) : y_{ij} + y_{ji} \geq 1 \\ (x_i) : y^* - \sum_{j:\{i,j\} \in E} y_{ij} \geq 0 \\ y^* \geq 0 \\ y_{ij}, y_{ji} \geq 0 \end{cases} \tag{20}$$

Our greedy 2-approximtion for DS is esentially a dual LP solution: set $y^* = \Delta_{\phi_{GR}}$ and

$$y_{ij} = \begin{cases} 1 & \text{if } \{i,j\} \text{ is directed towards } i \text{ in } \phi_{GR} \\ 0 & \text{o/w} \end{cases} \tag{21}$$

For the $x_{\{i,j\}}$, $y_{ij} + y_{ji} \geq 1$ is satisfied, because it is either $1+0 \geq 1$ or $0+1 \geq 1$. For x_i, $y^* \geq \sum_{j:\{i,j\} \in E} y_{ij}$. $\Delta_{\phi_{GR}} = y^* \geq \sum_{j:\{i,j\} \in E} y_{ij} = deg_{\phi_{GR}}(i)$.

Then, each constraint is satisfied by our solution, of value $\Delta_{\phi_{GR}}$.

$$\frac{\Delta_{\phi_{GR}}}{2} \leq \text{greedy solution} \leq PRIMAL^* \leq DUAL^* \leq \Delta_{\phi_{GR}}$$

Now, recall that Solution for Vertex Cover is a *minimization* problem. We saw its dual:

$$\begin{cases} \min \sum_{v \in V} x_v \\ y_{\{u,v\}} : x_u + x_v \geq 1 \\ x_v \geq 0 \end{cases} \tag{22}$$

Let us now see its primal:

$$\begin{cases} \max \sum_{\{u,v\} \in E} y_{\{u,v\}} \\ x_v : \sum_{u:\{u,v\} \in E} y_{\{u,v\}} \leq 1 \\ y_{\{u,v\}} \geq 0 \end{cases} \tag{23}$$

For all maximal matching $M \subseteq E$. For the dual, we set

$$x_u = \begin{cases} 1 & \text{if } e \in M \text{ s.t. } u \in E \\ 0 & \text{o/w} \end{cases} \tag{24}$$

Is this dual solution feasible? Yes. By contradiction, if $x_u + x_v < 1$, for some $\{u, v\} \in E$, we could add $\{u, v\}$ to M. Then M wouldn't be maximal. What's the value of the solution? It is $\sum_{v \in V} x_v = 2|M|$.

Instead, for the primal, we set

$$y_{\{u,v\}} = \begin{cases} 1 & \text{if } \{u, v\} \in M \\ 0 & \text{o/w} \end{cases} \tag{25}$$

In the algorithm, if there exists and edge e, we add both endpoints to the solution, and then we remove them from the graph. THe solution is feasible because M is a matching.

$$|M| \leq PRIMAL^* \leq DUAL^* \leq 2|M|$$

3.7 Greedy algorithms for Densest Subgraph

Going back to Densest subgraph problem, we want to improve the greedy algorithm, by parallelizing it and possibly make it so the number of iterations is reduced.

```
procedure Parallel_Greedy($G(V, E)$)
    $S_0 \leftarrow V$
    $i \leftarrow 0$
    while $S_i \neq \emptyset$ do
        $A_i \leftarrow \{v \mid v \in S_i \wedge \deg_{S_i}(v) = \min_{u \in S_i} \deg_{S_i}(u)\}$
        $S_{i+1} \leftarrow S_i \setminus A_i$
        $i \leftarrow i + 1$
    end while
    return a set $S_i$ having larget density $\rho(S_i)$
end procedure
```

There are instances in which this algorithm saves many iterations, as well as instances in which it doesn't save much time. Let us think for example of a graph in which each component is two vertices connected together; it is easy to see that the graph that the algorithm would stop after only 1 iteration. Let us now consider a path; the algorithm, at each iteration, would remove both endpoints of the path, thus requiring $\approx \frac{n}{2}$ iterations, if n is the number of nodes.

So we need a better algorithm. See the algorithm 7.

Does this solve the problem? Not really. Consider a path with a triangle in one endpoint. For such a graph the algorithm requires $\approx n$ iterations.

See the alorithm 8

Algorithm 7 Greedy average

```
procedure GREEDY_AVG($G(V, E)$)
    $S_0 \leftarrow V$
    $i \leftarrow 0$
    while $S_i \neq \emptyset$ do
        let $v_i$ be a node s.t. $\deg_{S_i}(v_i) \leq avg_{v \in S_i} \deg_{S_i}(v)$
        $S_{i+1} \leftarrow S_i \setminus \{v_i\}$
        $i \leftarrow i + 1$
    end while
    return a set $S_i$ having larget density $\rho(S_i)$
end procedure
```

Algorithm 8 Greedy average

```
procedure GREEDY_AVG($G(V, E)$)
    $S_0 \leftarrow V$
    $i \leftarrow 0$
    while $S_i \neq \emptyset$ do
        $A_i \leftarrow \{v_i \mid \deg_{S_i}(v_i) \leq avg_{v \in S_i} \deg_{S_i}(v)\}$
        $S_{i+1} \leftarrow S_i \setminus A_i$
        $i \leftarrow i + 1$
    end while
    return a set $S_i$ having larget density $\rho(S_i)$
end procedure
```

Still no solution. A path is once once again an instance in which this algortihm doesn't save much time. Though this is not a bad idea per se, so why not try to take a little more than the average?

Algorithm 9 Greedy ε (Bahmani, Kumar, Vassilutski, 2012)

```
procedure GREEDY$_\varepsilon$($G(V, E)$)
    $S_0 \leftarrow V$
    $i \leftarrow 0$
    while $S_i \neq \emptyset$ do
        $A_i \leftarrow \{v_i \mid v_i \in S_i \wedge \deg_{S_i}(v_i) \leq (1 + \varepsilon) avg_{v \in S_i} \deg_{S_i}(v)\}$
        $S_{i+1} \leftarrow S_i \setminus A_i$
        $i \leftarrow i + 1$
    end while
    return a set $S_i$ having larget density $\rho(S_i)$
end procedure
```

Now we want to prove two things about this algorithm:

- $GREEDY_\varepsilon$ gives a good approximation to DS
- $GREEDY_\varepsilon$ runs quickly

Lemma 3.20. $GREEDY_\varepsilon$ returns a $2(1 + \varepsilon)$-approximation, $\forall \varepsilon > 0$.

Proof. Let $S^* \neq \emptyset$ be an optimal solution (then, wlog, $|S^*| \geq 2$).

We claim that $\forall v \in S^*$, $\deg_{S^*}(v) \geq \rho(S^*) \triangleq \dfrac{|E(S^*)|}{|S^*|}$.

$\rho(S^*) = \dfrac{|E(S^*)|}{|S^*|} \overset{\text{(by optimality of } S^*)}{\geq} \rho(S^* \setminus \{v\}) = \dfrac{|E(S^* \setminus \{v\})|}{|S^* \setminus \{v\}|} = \dfrac{|E(S^*)| - \deg_{S^*}(v)}{|S^*| - 1}$

So $\dfrac{|E(S^*)| - \deg_{S^*}(v)}{|S^*| - 1} \leq \dfrac{|E(S^*)|}{|S^*|}$.

This implies that $|E(S^*)| - \deg_{S^*}(v) \leq \dfrac{|S^*| - 1}{|S^*|}|E(S^*)| = (1 - \dfrac{1}{|S^*|})|E(S^*)|$.

Thus $|E(S^*)| - \deg_{S^*} \leq |E(S^*)| - \dfrac{|E(S^*)|}{|S^*|}$.

So $\dfrac{|E(S^*)|}{|S^*|} \leq \deg_{S^*}(v)$. And this proves the first claim.

Our second claim is that at least 1 node isremoved in each iteration (this is more technical, note in fact that, if it wasn't like that, the algorithm might run forever).

$$\min_{v \in S_i} \deg_{S_i}(v) \leq avg_{v \in S_i} \deg_{S_i}(v) \leq (1 + \varepsilon) avg_{v \in S_i} \deg_{S_i}(v)$$

Thus $v \in A_i$ (i.e. if $|S_i| \geq 1 \rightarrow |A_i| \geq 1$).

Consider the first iteration i s.t. at least one node of S^* is removed from S_i. That is, let i be the smallest integer s.t. $A_i \cap S^* \neq \emptyset$.

Let $v \in A_i \cap S^*$. Since i is the first iteration in which $A_i \cap S^* \neq \emptyset$, it must be that $S_i \supseteq S^*$.

$\rho(S^*) \overset{\text{(claim 1)}}{\leq} \deg_{S^*}(v) \overset{(S_i \supseteq S^*)}{\leq} \deg_{S_i}(v) \overset{\text{(greedy choice)}}{\leq} (1 + \varepsilon) avg_{u \in S_i} \deg_{S_i}(u) = (1 + \varepsilon)\dfrac{\sum_{u \in S_i} \deg_{S_i}(u)}{|S_i|} = (1 + \varepsilon)\dfrac{2|E(S_i)|}{|S_i|} = 2(1 + \varepsilon)\rho(S_i)$

Thus $\rho(S_i) \geq \dfrac{1}{2(1 + \varepsilon)}\rho(S^*)$.

So $GREEDY_\varepsilon$ returns something no worse than a $2(1 + \varepsilon)$-approximation. $\square$

4 Math and CS notions

4.1 Tail inequalities

- Markov inequality
- Chebyshev inequality
- Chernoff bound

4.1.1 Markov inequality

If Y is a non negative random variable, then $\forall c \geq 1$

$$Pr[Y \geq c \cdot \mathbb{E}[Y]] \leq \frac{1}{c}$$

Proof. Let ξ be an event, and let

$$X_\xi = \begin{cases} 1 & \text{if } \xi \text{ happens} \\ 0 & \text{o/w} \end{cases} \tag{26}$$

In particular, for $a > 0$, consider

$$X_{Y \geq a} = \begin{cases} 1 & \text{if } Y \geq a \text{ happens} \\ 0 & \text{o/w} \end{cases} \tag{27}$$

Then $Pr[a \cdot X_{Y \geq a} \leq Y] = 1$.

So, $\mathbb{E}[a \cdot X_{Y \geq a}] \leq \mathbb{E}[Y]$. Also, $\mathbb{E}[a \cdot X_{Y \geq a}] = a \cdot \mathbb{E}[X_{Y \geq a}] = a \cdot Pr[Y \geq a]$.

$a \cdot Pr[Y \geq a] \leq \mathbb{E}[Y]$, and $Pr[Y \geq a] \leq \frac{\mathbb{E}[Y]}{a}$.

Let us choose $a = c \cdot \mathbb{E}[Y]$, then $Pr[Y \geq c \cdot \mathbb{E}[Y]] \leq \frac{\mathbb{E}[Y]}{c \cdot \mathbb{E}[Y]} = \frac{1}{c}$. $\square$

4.2 NP

Disclaimer: as of now I'm **not** going to be too formal in the definition of NP, since for this course it is important to understand at least that NP problems are hard to solve.

NP stands for *Non-deterministic Polynomial.* It is the set of problems that can be solved in polynomial time by a Non-Deterministic Turing Machine (NDTM). Suppose input size n, we say that an algorithm runs in *polynomial time* if it runs in time $O(n^c)$, for some constant $c > 0$.

These problems capture the idea of puzzles. Given a puzzle, it is hard to solve it, but, given a solution to the puzzle, it is easy to check if the solution is correct.

A bit more formally, given a set S, we want to know if the instance x belongs to S. Given a solution y to the problem, there exists an algorithm $V(\cdot, \cdot)$ that, runs in polynomial time, and given as inputs x and y (i.e. $V(x, y)$) returns true iff $x \in S$; if this is the case, then y is said to be a valid witness.

Let us see an example on *clique.* We want to know if a graph G has a k-clique; so G would be our instance x and S would be the set of all graphs with a k-clique. Finding a k-clique is hard in general, but, given a set of vertices, our witness y, it is quite easy to check if y is a k-clique. Suppose the graph has n node, at most (i.e. for an n-clique) we would have to check if every node is connected to every other node; this procedure would take time $O(n^2)$.

What happens with many (if not all?) of these NP problems is that the solution size is polynomial w.r.t. the input size (possibly even linear), but the number of possible solutions is exponential.

The ways in which we can deal with these problems are the following:

- brute force the optimal solution (not recommended)
- using smart constructs and smart heuristics, some algorithms, although still having exponential time in the worst case, can solve many real life instances really really fast (possibly even in linear or almost linear time)
- find an approximation in polynomial time (which is what we do in this course)

4.2.1 NP-Hardness

A problem is said to be NP-Hard if it is harder to solve than any other problem in NP.

To formally define NP-Hardness we would should also define what a Karp-Reduction is. Very informally, if a problem A is harder to solve than B means that, if we have an algorithm to solve A, then we could also solve B.

4.2.2 NP-Completeness

A problem is said to be NP-Complete if it is both NP-Hard and in NP. So the NP-Complete problems are the hardest to solve in NP.

4.3 Graphs

4.3.1 Induced subgraph

Let $G(V, E)$ be a graph. We define $G[S]$, the subgraph of G induced by S, for $S \subseteq V$, as the graph having S as a set of vertices and edges $\{e \mid e \in E \land e \subseteq S\}$.

Informally, we take we original graph, we cut away any node not in S and all the edges connected to these removed nodes.