SAPIENZA
UNIVERSITÀ DI ROMA

"Sapienza" University of Rome
Faculty of Information Engineering,
Informatics and Statistics
Department of Computer Science

Advanced Algorithms

Author
Alessio Bandiera

June 22, 2025

Contents

Information and Contacts

Personal notes and summaries collected as part of the *Advanced Algorithms* course offered by the degree in Computer Science of the University of Rome "La Sapienza".

Further information and notes can be found at the following link:
https://github.com/aflaag-notes. Anyone can feel free to report inaccuracies, improvements or requests through the Issue system provided by GitHub itself or by contacting the author privately:

- Email: **alessio.bandiera02@gmail.com**
- LinkedIn: **Alessio Bandiera**

The notes are constantly being updated, so please check if the changes have already been made in the most recent version.

Suggested prerequisites:

- Progettazione degli Algoritmi

Licence:

1 Approximation algorithms

TODO

missing introduction

Lastly, *unless specified* we will adopt the following convention in these notes, when talking about **approximation algorithm**. Given a problem P for which there is an optimal solution OPT, and an algorithm that solves P which outputs a solution ALG to P

- if P is a **minimization** problem, we say that A yields an α-approximation of P if

$$\text{ALG} \leq \alpha \cdot \text{OPT}$$

  in which $\alpha \in \mathbb{R}_{>1}$

- if P is a **maximization** problem, we say that A yields an α-approximation of P if

$$\text{ALG} \geq \alpha \cdot \text{OPT}$$

  in which $\alpha \in (0, 1]$

1.1 Approximation through randomness

1.1.1 The Maximum Cut problem

The first problem that will be discussed is the Maximum Cut problem. The **Maximum Cut problem** — in the unweighted case — is a classic combinatorial optimization problem in the branch of graph theory, in which we seek to partition the vertices of an undirected graph into two disjoint subsets, while maximizing the number of edges that have endpoints in both subsets. More formally, we will define a **cut** of a graph as follows.

Definition 1.1: Cut

Given an undirected graph $G = (V, E)$, and two disjoint subsets $S, T \subseteq V$, the **cut** induced by S and T on G is defined as follows

$$\text{cut}(S, T) = \{e \in E : |S \cap e| = |T \cap e| = 1\}$$

In the case in which $T = \overline{S} = V - S$ we will simply write $\text{cut}(S) = \text{cut}(S, \overline{S})$.

Note that in the definition above we are defining the cut of a graph through the intersection between a set of vertices and edges in E; this is because, in the undirected case, we will consider the edges of a graph $G = (V, E)$ as sets of 2 elements

$$E = \{\{u, v\} \mid u, v \in V\}$$

Therefore, given two sets of vertices S and T, the cut induced by S and T is simply the set of edges that have only one endpoint in S and the other one in T.

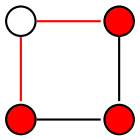


Figure 1.1: Given the set of red vertices S, the green edges represent $\text{cut}(S)$.

With this definition, we can introduce the **Maximum Cut** problem, which is defined as follows.

Definition 1.2: Maximum Cut problem

The **Maximum Cut** (MC) problem is defined as follows: given an undirected graph $G = (V, E)$, determine the set $S \subseteq V$ that maximizes $|\text{cut}(S)|$.

Although this problem is known to be APX-hard [ALM+98], approximation algorithms and heuristic methods like *greedy algorithms* and *local search* are commonly used to find near-optimal solutions.

For now, we present the following **randomized algorithm**, which provides a straightforward $\frac{1}{2}$-approximation for MC. This algorithm runs in polynomial time and achieves the approximation guarantee with high probability.

Algorithm 1.1: Random Cut

Given an undirected graph $G = (V, E)$, the algorithm returns a cut of G.

```
1: function RandomCut(G)
2:     S := ∅
3:     for v ∈ V do
4:         Let c_v be the outcome of the flip of an independent fair coin
5:         if c_v == H then
6:             S = S ∪ {v}
7:         end if
8:     end for
9:     return S
10: end function
```

Note that this algorithm is powerful, because *it does not care about the structure of the graph in input*, since the output is completely determined by the coin flips performed in the `for` loop. Now we will prove that this algorithm provides a correct **expected $\frac{1}{2}$-approximation** of MC.

Theorem 1.1

Let $G = (V, E)$ be a graph, and let S^* be an optimal solution to MC on G. Then, given $S = \text{RandomCut}(G)$, it holds that

$$\mathbb{E}[|\text{cut}(S)|] \geq \frac{1}{2} |\text{cut}(S^*)|$$

Proof. By definition, note that

$$\forall e \in E \quad e \in \text{cut}(S) \iff |S \cap e| = 1$$

Consider an edge $e = \{v, w\} \in E$; then, by definition

$$\{v, w\} \in \text{cut}(S) \iff (v \in S \wedge w \notin S) \vee (v \notin S \wedge w \in S)$$

and let ξ_1 and ξ_2 be these last two events respectively. Then

$$\Pr[\xi_1] = \Pr[c_v = \text{heads} \wedge c_w = \text{tails}]$$

by definition of the algorithm, and by independence of the flips of the fair coins we have that

$$\Pr[\xi_1] = \Pr[c_v = \text{heads}] \cdot \Pr[c_w = \text{tails}] = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$$

Analogously, we can show that

$$\Pr[\xi_2] = \frac{1}{4}$$

This implies that

$$\Pr[e \in \text{cut}(S)] = \Pr[\xi_1 \vee \xi_2] = \Pr[\xi_1] + \Pr[\xi_2] - \Pr[\xi_1 \wedge \xi_2] = \frac{1}{4} + \frac{1}{4} - 0 = \frac{1}{2}$$

Hence, we have that

$$\mathbb{E}[|\text{cut}(S)|] = \sum_{e \in E} 1 \cdot \Pr[e \in \text{cut}(S)] = \frac{|E|}{2} \geq \frac{|\text{cut}(S^*)|}{2}$$

where the last inequality directly follows from the definition of cut of a graph. □

As previously mentioned, this algorithm has an **expected approximation ratio** of $\frac{1}{2}$, which implies that it may return very bad solutions in some cases, depending on the outcomes of the coin flips. However, thanks to the following algorithm, we can actually transform the **guarantee of expectations** into a **guarantee of high probability** — note that it is possible to show that the previous algorithm provides guarantee of high probability as well, but the proof is much more complex.

Algorithm 1.2: t-times Random Cut

Given an undirected graph $G = (V, E)$ and an integer $t > 0$, the algorithm returns a cut of G.

1: **function** t-TIMESRANDOMCUT(G, t)
2: **for** $i \in [t]$ **do**
3: $S_i :=$ RANDOMCUT(G)
4: **end for**
5: **return** $S \in \arg\max_{i \in [t]} |\text{cut}(S_i)|$
6: **end function**

The algorithm above simply runs the RANDOMCUT algorithm t times, and returns the set S_i that maximizes the cut, among all the various $S_1, \ldots, S_t$. The following theorem will show that a *reasonable number* of runs of the RANDOMCUT algorithm suffices in order to almost certainly obtain a $\approx \frac{1}{2}$-approximation of any optimal solution.

Theorem 1.2

Let $G = (V, E)$ be a graph, and let S^* be an optimal solution to MC on G. Then, given $S = t$-TIMESRANDOMCUT(G, t), it holds that

$$\Pr\left[|\text{cut}(S)| > \frac{1-\varepsilon}{2} |\text{cut}(S^*)|\right] > 1 - \delta$$

where $t = \frac{2}{\varepsilon} \ln \frac{1}{\delta}$ and $0 < \varepsilon, \delta < 1$.

Proof. For each $i \in [t]$, let $C_i := |\text{cut}(S_i)|$ for each S_i defined by the algorithm, and let $N_i := |E| - C_i$. Let $0 < \varepsilon < 1$; since N_i is a non-negative random variable, by Markov's

inequality we have that

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

In particular, this inequality can be rewritten as follows:

$$\begin{aligned} 1 - \frac{\varepsilon}{2} &\geq \Pr[N_i \geq (1+\varepsilon)\mathbb{E}[N_i]] \\ &= \Pr[|E| - C_i \geq (1+\varepsilon)(|E| - \mathbb{E}[C_i])] \\ &= \Pr[-\varepsilon\,|E| \geq C_i - (1+\varepsilon)\mathbb{E}[C_i]] \end{aligned}$$

As shown in the proof of Theorem 1.1, we know that $\mathbb{E}[C_i] = \frac{|E|}{2}$, therefore

$$\begin{aligned} 1 - \frac{\varepsilon}{2} &\geq \Pr[-\varepsilon\,|E| \geq C_i - (1+\varepsilon)\mathbb{E}[C_i]] \\ &= \Pr\left[-\varepsilon\,|E| \geq C_i - \frac{1+\varepsilon}{2}\,|E|\right] \\ &= \Pr\left[-\varepsilon\frac{|E|}{2} \geq C_i - \frac{|E|}{2}\right] \\ &= \Pr\left[\frac{1-\varepsilon}{2}\,|E| \geq C_i\right] \\ &= \Pr\left[(1-\varepsilon)\mathbb{E}[C_i] \geq C_i\right] \end{aligned}$$

Note that the event in the last probability, namely

$$|\mathrm{cut}(S_i)| \leq (1-\varepsilon)\mathbb{E}[|\mathrm{cut}(S_i)|]$$

corresponds to a "bad" solution, i.e. one whose cardinality is at most $(1-\varepsilon)$-th of the expected value.

By definition of the algorithm, each of the t runs of the RandomCut algorithm is independent from the others, therefore the probability of *all* the solutions $S_1, \ldots, S_t$ being "bad" is bounded by

$$\Pr[\forall i \in [t] \quad C_i \leq (1-\varepsilon)\mathbb{E}[C_i]] = \prod_{i=1}^{t} \Pr[C_i \leq (1-\varepsilon)\mathbb{E}[C_i]] \leq \left(1 - \frac{\varepsilon}{2}\right)^t$$

Using the fact that

$$\forall x \in \mathbb{R} \quad 1 - x \leq e^{-x} \implies 1 - \frac{\varepsilon}{2} \leq e^{-\frac{\varepsilon}{2}}$$

we have that

$$\Pr[\forall i \in [t] \quad C_i \leq (1-\varepsilon)\mathbb{E}[C_i]] \leq \left(1 - \frac{\varepsilon}{2}\right)^t \leq e^{-\frac{\varepsilon}{2}\cdot t} = e^{-\ln\frac{1}{\delta}} = \delta$$

Therefore, the probability that at least one among $S_1, \ldots, S_t$ is a "good" solution is bounded by

$$\Pr[\exists i \in [t] \quad C_i > (1-\varepsilon)\mathbb{E}[C_i]] = 1 - \Pr[\forall i \in [t] \quad C_i \leq (1-\varepsilon)\mathbb{E}[C_i]] \geq 1 - \delta$$

Finally, since the arg max operation in the algorithm will select in the "worst" case such good solution, we conclude that

$$\begin{aligned}\Pr\left[|\text{cut}(S)| > \frac{1-\varepsilon}{2}|\text{cut}(S^*)|\right] &\geq \Pr\left[\exists i \in [t] \quad C_i > \frac{1-\varepsilon}{2}|\text{cut}(S^*)|\right] \\ &\geq \Pr[\exists i \in [t] \quad C_i > (1-\varepsilon)\,\mathbb{E}[C_i]] \\ &\geq 1-\delta\end{aligned}$$

□

Note that this result is *very powerful*: for instance, if $\varepsilon = \delta = 0.1$, we get that

$$\Pr[|\text{cut}(S)| > 0.45 \cdot |\text{cut}(S^*)|] \geq 0.9$$

and $t \approx 46$, meaning that we just need to run the RandomCut algorithm approximately 46 times in order to get a solution that is better than a 0.45-approximation with 90% probability.

In later sections we will present a 0.878-approximation of MC, based on a *very* different approach.

1.2 Approximation through reduction

1.2.1 The Vertex Cover problem

Another very important problem in graph theory is the Vertex Cover problem, which concerns the combinatorial structure of the **vertex cover**, defined as follows.

Definition 1.3: Vertex cover

Given an undirected graph $G = (V, E)$, a **vertex cover** of G is a set of vertices $S \subseteq V$ such that

$$\forall e \in E \quad \exists v \in S \quad v \in e$$

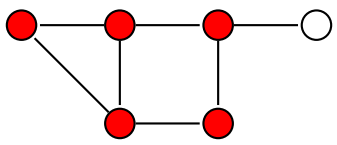


Figure 1.2: An example of a vertex cover.

As shown in figure, a vertex cover is simply a set of vertices that must *cover* all the edges of the graph. Clearly, the trivial vertex cover is representd by $S = V$, but a more interesting solution to the problem is represented by the **minimum vertex cover**.

Definition 1.4: Vertex Cover problem

The **Vertex Cover** (VC) problem is defined as follows: given an undirected graph $G = (V, E)$, determine the vertex cover $S \subseteq V$ of smallest cardinality.

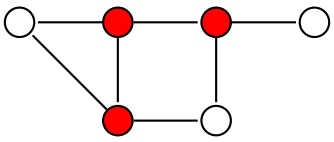


Figure 1.3: This is the *minimum vertex cover* of the previous graph.

As famously proved by Karp [Kar72] in 1972, this problem is NP-complete, hence we are interested in finding algorithms that allow to find approximations of optimal solutions. For instance, an algorithm that is able to approximate VC concerns the matching problem.

Definition 1.5: Matching

Given an undirected graph $G = (V, E)$, a **matching** of G is a set of edges $A \subseteq E$ such that

$$\forall e, e' \in A \quad e \cap e' = \varnothing$$

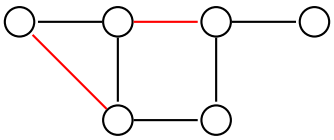


Figure 1.4: A *matching* of the previous graph.

As shown in figure, a matching is nothing more than a set of edges that must not share endpoints with each other — for this reason, in literature it is often referred to as **independent edge set**. Differently from the vertex cover structure, in this context the trivial matching is clearly the set $A = \varnothing$, which vacuously satisfies the matching condition. However, a more interesting solution is represented by the **maximum matching**, but this time we have to distinguish two slightly different definitions, namely the concept of *maximal* and *maximum*.

Definition 1.6: Maximal matching

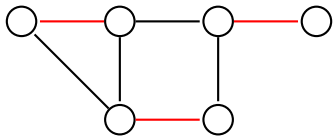
A **maximal matching** is a matching that cannot be extended any further.

For instance, the matching shown in Figure 1.4 is actually a **maximal matching**, because no other edge in E can be added to the current set of edges A of the matching without breaking the matching condition.

Definition 1.7: Maximum matching

A **maximum matching** is a matching that has the largest cardinality.

Clearly, the previous example does not represent a **maximum matching**, because the following set of edges



is still a valid matching for the graph, but has a larger cardinality than the previous set.

Differently from VC, a *maximum matching* can be found in polynomial time. Moreover, the following algorithm can be used to determine a *maximal matching* of a given graph.

Algorithm 1.3: Maximal matching

Given an undirected graph $G = (V, E)$, the algorithm returns a maximal matching of G.

```
1:  function MaximalMatching(G)
2:      S := ∅
3:      while E ≠ ∅ do
4:          Choose e ∈ E
5:          S = S ∪ {e}
6:          Remove from E all the edges incident on u or on v
7:          E = E − {e}
8:      end while
9:      return S
10: end function
```

Idea. The algorithm is very straightforward: at each iteration, a random edge $e = \{u, v\}$ is chosen from E, and then any edge $e' \in E$ such that $e \cap e' \neq \varnothing$ is removed from E.

Clearly, line 6 ensures that the output is a *matching*, and the terminating condition of the `while` loop ensures that it is *maximal*, but since the output depends on the chosen edges, S is not guaranteed to be *maximum*.

Another major reason on why we focus on matchings is the following theorem.

Theorem 1.3: Matchings bound vertex covers

Given an undirected graph $G = (V, E)$, a matching $A \subseteq E$ of G, and a vertex cover $S \subseteq V$ of G, we have that $|S| \geq |A|$.

Proof. By definition, any vertex cover S of $G = (V, E)$ is also a vertex cover for $G^B = (V, B)$, for any set of edges $B \subseteq E$, and in particular this is true for $G^A = (V, A)$.

Now consider G^A, and a vertex cover C on it: by construction we have that $\Delta \leq 1$, therefore any vertex in C will cover at most 1 edge of A. This implies that if $|C| = k$, then C will cover at most k edges of G^A.

Lastly, since G^A has $|A|$ edges by definition, any vertex cover defined on G^A has to contain at least $|A|$ vertices. This implies that no vertex cover S of G smaller than $|A|$ can exist, because S will have to cover at least the edges in A. □

Thanks to this theorem, we can easily show that the algorithm that we just presented in order to find maximal matchings is a 2-approximation of VC.

Theorem 1.4

Given a graph G, and $M = \textsc{MaximalMatching}(G)$, let $S := \bigcup_{e \in M} e$. Then S is a 2-approximation of VC on G.

Proof. Consider an optimal solution S^* to VC on G, and let $M = \{e_1, \ldots, e_t\}$. Note that at each iteration of the algorithm exactly 1 edge is added to $M \subseteq V$, hence it always holds that

$$S_i \cap S_{i+1} = e_i = \{u, v\}$$

for any iteration $i \in [t - 1]$. Moreover, since M is a matching, it holds that $\forall e, e' \in M \quad e \cap e' = \varnothing$, therefore $|S| = 2\,|M|$. Finally, by the previous theorem we have that $|M| \leq |S^*|$, concluding that

$$|S| = 2\,|M| \leq 2\,|S^*|$$

□

This 2-approximation algorithm is conjectured to be optimal, but it has not been proven yet. In fact, VC is conjectured to be NP-hard to $(2-\varepsilon)$-approximate, for any $\varepsilon > 0$.

Interestingly, the decisional version of VC is Fixed Parameter Tractable. This characterization comes from the nature of the problem: for each edge $e = \{u, v\}$ of a given undirected graph $G = (V, E)$, either u or v has to be in the vertex cover, therefore it possible to approach VC by trying all possible choices of set of vertices $S \subseteq V$, and backtrack if necessary. The following algorithm employs this idea.

Algorithm 1.4: Decisional VC

Given an undirected graph $G = (V, E)$, and an integer k, the algorithm returns `True` if and only if G admits a vertex cover of size k.

```
1: function VC(G, k)
2:     if E == ∅ then
3:         return True
4:     else if k == 0 then
5:         return False
6:     else
7:         Choose e = {u, v} ∈ E
8:         if VC(G[V − {u}]), k − 1) then
9:             return True
10:        end if
11:        if VC(G[V − {v}]), k − 1) then
12:            return True
13:        end if
14:        return False
15:    end if
16: end function
```

Note that this algorithm actually solves the *decisional* version of VC. Moreover, the algorithm uses the definition of **induced subgraph**, which is the following.

Definition 1.8: Induced subgraph

Given a graph $G = (V, E)$, and a set of vertices $S \subseteq V$, then $G[S]$ represents the **subgraph induced by** S **on** G, obtained by removing from G all the nodes of $V - S$ and the edges incident on them.

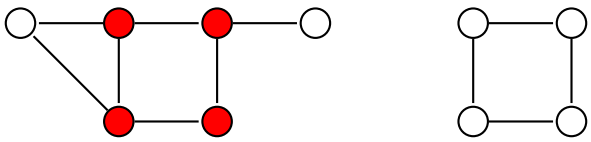


Figure 1.5: On the left: a graph G and a set of vertices S. On the right: the graph $G[S]$.

Idea. The structure of the algorithm consists of a simple backtracking algorithm:

- if the current graph has no edges, we covered every edge of the graph, therefore we return `True`
- if the current graph has some edges, but $k = 0$, then G does not admit a vertex cover of size k, thus we return `False`
- if the current graph has some edges, and $k \neq 0$, then we choose an edge $e = \{u, v\} \in$

E arbitrarily, and we try to consider first u then v in a possible vertex cover — note that $G[V - \{x\}]$ is a graph that does not contain x, neither any edge adjacent to it; if both attempts fail, we return `False`

Cost analysis. It is easy to see that the cost directly depends on the number of recursive calls that the algorithm performs, which is 2^k in the worst case, and the cost of constructing $G[V - \{x\}]$, which we can assume to be $O(n^2)$. Hence, the algorithm has a total cost of $O\left(2^k \cdot n^2\right)$.

2

Mathematical Programming

Up to this point, we have introduced several NP-complete problems and discussed approximation algorithms that provide near-optimal solutions within a certain approximation ratio. We will now explore a new technique that can be leveraged to design approximation algorithms for a broader class of problems, namely **mathematical programming**.

Mathematical programming is a powerful framework used to model and solve **optimization problems** across various fields, including operations research, computer science and engineering. It provides a structured way to *maximize* or *minimize* an **objective function** while satisfying a set of **constraints**. In particular, optimization problems are usually expressed as follows

$$\begin{aligned} &\max \ f(x) \\ &\begin{aligned} g_i(x) &\leq b_i \quad \forall i \in [n] \\ x &\in V \end{aligned} \end{aligned}$$

In this example, we see that

- x is a *vector* that lies inside the vector space V
- the objective function is $f(x)$, which can be either maximized or minimized
- $g_i(x) \leq b_i$ is a constraint — i.e. an inequality — that x must satisfy

Among the most widely studied forms of mathematical programming are **Linear Programming** (LP), **Integer Programming** (IP), and **Semidefinite Programming** (SDP), each with distinct properties and applications.

Starting from linear programming, in LPs both the objective function and the constraints are *linear* w.r.t. V, and there are no *equality* constraints.

$$\max\ x_1 + x_2$$

$$\begin{aligned} 2x_1 + x_2 &\leq 10 \\ x_i &\geq 0 \qquad \forall i \in [n] \\ x &\in \mathbb{R}^n \end{aligned}$$

Figure 2.1: Example of an LP.

LPs can be solved in **polynomial time**: specifically, if an LP has n variables, m constraints, and each coefficient can be expressed as the ratio of two t-bit integers (where real numbers are approximated as rationals), then the Ellipsoid method can solve it in time $O((nmt)^c)$ for some constant $c > 0$. This result extends, to some extent, to SDPs as well.

Despite its theoretical guarantee of polynomial runtime, in practice the Simplex method is often preferred, as it operates based on *pivot rules*. In fact, although all known pivot rules for the Simplex method exhibit a theoretical *exponential* lower bound due to specially constructed worst-case instances, its average performance is significantly better than that of the Ellipsoid method in real-world scenarios.

2.1 IPs and LP relaxation

Differently from LPs, in IPs the vector space of interest is $\{0,1\}^n$. IPs can be used to solve a wide range of problems. For instance, given a graph $G = (V, E)$, the **vertex cover** problem that we discussed in the previous section can be formulated through the following IP:

$$\min \sum_{u \in V}^{n} x_v$$

$$\begin{aligned} x_u + x_v &\geq 1 \quad \forall \{u, v\} \in E \\ x_v &\in \{0,1\} \quad \forall u \in V \end{aligned}$$

Figure 2.2: IP for VC.

It is fairly straightforward to prove that an optimal solution to this IP yields an optimal solution to VC.

Lemma 2.1

Given a graph G, if $\{x_u^*\}_{u \in V}$ is an optimal solution to the previous IP, then

$$S^* := \{v \in V \mid x_v^* = 1\}$$

is a minimum vertex cover for G.

Proof. Consider an optimal solution $\{x_u^*\}_{u \in V}$ to the IP for VC, and define S^* as the set of vertices $v \in V$ such that $x_v^* = 1$. Note that any optimal solution is also a feasible solution, i.e. it satisfies the constraints of the IP.

First, we provide an idea of the proof. The first constraint of the IP forces that for each $\{u, v\} \in E$ the sum between $x_u^* + x_v^*$ is at least one, and the second constraint forces each variable x_v^* to be either 0 or 1. Therefore, together these two constraints imply that for any edge $\{u, v\} \in E$ at least one between x_u^* and x_v^* is 1, and by definition of S^* this means at least one of the endpoints of $\{u, v\}$ is inside S^*. We conclude that S^* is indeed a vertex cover for G, and by its definition note that $|S^*| = \sum_{u \in V} x_u^*$.

This is generalized in the following claim.

Claim: Given a vertex cover S of a graph $G = (V, E)$, there exists a feasible solution $\{x_u\}_{u \in V}$ to the IP having value $|S|$.

Proof of the Claim. Define the solution $\{x_u\}_{u \in V}$ by setting $x_u = 1$ if and only if $u \in S$. Clearly, the value of this solution is indeed $\sum_{u \in V} x_u = |S|$; moreover, by definition of vertex cover, for any edge $\{u, v\} \in E$ at least one between u and v must be in S, therefore at least one between x_u and x_v is set to 1, implying that $x_u + x_v \geq 1$ is always satisfied. □

By way of contradiction, suppose that S^* is not a minimum vertex cover. Hence, there must be another vertex cover S' such that $|S'| < |S^*|$. By the previous claim, this implies that there exists a feasible solution $\{x'_u\}_{u \in V}$ for the IP that has value $|S'|$, but then

$$\sum_{u \in V} x'_u = |S'| < |S^*| = \sum_{u \in V} x_u^*$$

which contradicts the optimality of the solution of $\{x_u^*\}_{u \in V}$ for the IP. □

In particular, this lemma implies that VC can be reduced to Integer programming, indeed solving IPs is actually NP-hard [Kar72], differently from LPs. This result shows that IPs cannot be used *directly* to obtain perfect solutions, but they are still very useful thanks to **relaxation**.

To *relax* an IP, we simply replace the constraint $x \in \{0, 1\}^n$ with $0 \leq x \leq 1$, transforming the IP into an LP.

$$\min \sum_{u \in V}^{n} x_v$$

$$\begin{aligned} x_u + x_v &\geq 1 \quad \forall \{u, v\} \in E \\ 0 \leq x_v &\leq 1 \quad \forall u \in V \end{aligned}$$

Figure 2.3: LP relaxation for the IP of VC.

But solving this LP is not enough to obtain a meaningful solution: in fact, a real-valued solution for this problem does not directly yield a vertex cover for a given graph. To fix this issue, the optimal solution of the LP relaxation is usually transformed through techniques such as **rounding**. Intuitively, the simplest possible type of *rounding rule* is the following: given a solution $\{\overline{x}_u\}_{u \in V}$ to the LP relaxation, to obtain a VC consider the following set

$$S := \left\{ v \in V \mid \overline{x}_v \geq \tfrac{1}{2} \right\}$$

and for VC in particular, we can prove that this rounding rule actually yields a 2-approximation of any optimal solution.

Theorem 2.1

Given a graph $G = (V, E)$, if $\{\overline{x}_u\}_{u \in V}$ is an optimal solution to the LP relaxation of the IP for VC, then

$$\overline{S} := \left\{ v \in V \mid \overline{x}_v \geq \tfrac{1}{2} \right\}$$

is a 2-approximation for VC.

Proof. Since $\{\overline{x}_u\}_{u \in V}$ is an optimal solution to the LP relaxation, it must satisfy the first constraint for which $\overline{x}_u + \overline{x}_v \geq 1$ for any $\{u, v\} \in E$. Moreover, for the second constraint we have that $\overline{x}_u \geq 0$ for all $u \in V$, therefore

$$\forall \{u, v\} \in E \quad \max(\overline{x}_u, \overline{x}_v) \geq \operatorname{avg}(\{\overline{x}_u, \overline{x}_v\}) = \frac{\overline{x}_u + \overline{x}_v}{2} \geq \frac{1}{2}$$

which means that at least one between $\overline{x}_u$ and $\overline{x}_v$ is at least $\frac{1}{2}$, implying that the edge $\{u, v\}$ will be covered by at least one of the two endpoints u and v, by definition of $\overline{S}$. This proves that $\overline{S}$ is a vertex cover of G.

To prove that $\overline{S}$ is indeed a 2-approximation, we just need to show the following: given a minimum vertex cover S^* of G, it holds that $\left|\overline{S}\right| \leq 2\,|S^*|$. By the claim in Lemma 2.1, we know that there exists a feasible solution $\{x^*_u\}_{u \in V}$ for the IP having value $|S^*|$, which must be optimal for the IP since S^* is a minimum vertex cover for G. Therefore, we have

that

$$\begin{aligned}
\left|\overline{S}\right| &= \sum_{v \in \overline{S}} 1 \\
&\leq \sum_{v \in \overline{S}} 2\overline{x}_v \qquad \left(v \in \overline{S} \implies \overline{x}_v \geq \tfrac{1}{2} \iff 2\overline{x}_v \geq 1\right) \\
&= 2 \sum_{v \in \overline{S}} \overline{x}_v \\
&\leq 2 \sum_{v \in V} \overline{x}_v \qquad \left(\overline{S} \subseteq V \wedge \overline{x}_v \geq 0\right) \\
&\leq 2 \sum_{v \in V} x^*_v \\
&= 2\left|S^*\right|
\end{aligned}$$

where the last inequality comes from the fact that the constraints of the LP are *weaker* than the ones of the IP. □

However, note that this result should not come a surprise. In fact, consider a graph G, an optimal solution to the LP relaxation $\{\overline{x}_u\}_{u \in V}$ and $\overline{S}$ defined as previously shown; given an edge $\{u, v\} \in E(G)$, in the worst case we have that

$$\overline{x}_u = \overline{x}_v = \frac{1}{2}$$

which still satisfies both constraints of the LP relaxation, since $\frac{1}{2} + \frac{1}{2} = 1 \geq 1$. This means that, in the worst case, both u and v end up inside $\overline{S}$, which gives an intuitive reason to why this LP relaxation indeed yields a 2-approximation solution.

2.2 Approximation through Linear Programming

2.2.1 Integrality gap

In this section, we are going to use a different definition for the approximation ratio α: in particular, given a problem P

- if P is a **minimization** problem, we say that A yields an α-approximation of P if

  $$\text{ALG} \leq \alpha \cdot \text{OPT}$$

  in which $\alpha \in \mathbb{R}_{\geq 1}$
- if P is a **maximization** problem, we say that A yields an α-approximation of P if

  $$\text{ALG} \geq \frac{\text{OPT}}{\alpha}$$

xuch that, in both cases, it holds that in which $\alpha \in \mathbb{R}_{\geq 1}$. This convention is typically employed for approximation algorithms using LP relaxation.

Consider a problem P, its equivalent IP, and the relative LP relaxation. Given an instance $I \in \mathrm{P}$ of the problem, we will denote with $\mathrm{IP}^*_{\mathrm{P}}(I)$ and $\mathrm{LP}^*_{\mathrm{P}}(I)$ the optimal values for the IP and the LP of the problem P on the instance I — we will omit P and I if the context is clear enough. Note that, in general, for minimization problems it holds that $\mathrm{LP}^* \leq \mathrm{IP}^*$ since the constraints of the LP relaxation are *weaker* than the ones of the IP.

Proposition 2.1

If P is a problem, then

- if it is a **minimization** problem, it holds that $\mathrm{LP}^* \leq \mathrm{IP}^*$
- if it is a **maximization** problem, it holds that $\mathrm{LP}^* \geq \mathrm{IP}^*$

For example, the inequalities discussed in the proof of Lemma 2.1 could be rewritten as follows

$$\begin{aligned} \left|\overline{S}\right| &= \dots \\ &\leq 2 \sum_{v \in V} \overline{x}_v = 2\mathrm{LP}^* \\ &\leq 2 \sum_{v \in V} x^*_v = 2\mathrm{IP}^* = 2\left|S^*\right| \end{aligned}$$

and in particular $\left|\overline{S}\right| \leq 2\mathrm{LP}^* \leq 2\mathrm{IP}^*$. Can we improve this approximation ratio of 2 through LP relaxation? We observe that for any α possible approximation ratio it must hold that

$$\alpha \geq \frac{\mathrm{IP}^*}{\mathrm{LP}^*}$$

because otherwise

$$\alpha < \frac{\mathrm{IP}^*}{\mathrm{LP}^*} \implies \left|\overline{S}\right| \leq \alpha \mathrm{LP}^* < \frac{\mathrm{IP}^*}{\mathrm{LP}^*} \cdot \mathrm{LP}^* = \mathrm{IP}^*$$

meaning that $\overline{S}$ would be a solution better than the optimal solution of the IP, which is impossible — be careful that it is impossible because $\overline{S}$ is a solution to the IP, not the LP. We can generalize this concept as follows.

Definition 2.1: Integrality gap

Given a **minimization** problem P and an instance $I \in \mathrm{P}$, the **integrality gap** between $\mathrm{IP}^*_{\mathrm{P}}(I)$ and $\mathrm{LP}^*_{\mathrm{P}}(I)$ is defined as follows

$$\mathrm{IG}_{\mathrm{P}}(I) = \frac{\mathrm{IP}^*_{\mathrm{P}}(I)}{\mathrm{LP}^*_{\mathrm{P}}(I)}$$

The integrality gap for the problem P is defined as follows

$$\mathrm{IG}_{\mathrm{P}} = \sup_{I \in \mathrm{P}} \mathrm{IG}_{\mathrm{P}}(I) = \sup_{I \in \mathrm{P}} \frac{\mathrm{IP}^*_{\mathrm{P}}(I)}{\mathrm{LP}^*_{\mathrm{P}}(I)}$$

Vice versa, if P is a **maximization** problem, we *invert* the ratio in the definition, as follows

$$\mathrm{IG}_{\mathrm{P}} = \sup_{I \in \mathrm{P}} \mathrm{IG}_{\mathrm{P}}(I) = \sup_{I \in \mathrm{P}} \frac{\mathrm{LP}^*_{\mathrm{P}}(I)}{\mathrm{IP}^*_{\mathrm{P}}(I)}$$

Proposition 2.2

For both minimization and maximization problems P, it holds that $\mathrm{IG}_{\mathrm{P}} \geq 1$.

Proof. Follows trivially from Proposition 2.1 and the definition of integrality gap. □

In fact, through the previous argument, we can derive the following property that *must* hold for any approximation ratio.

Proposition 2.3: Limits of LP relaxation

Given a problem P for which there is an α-approximation algorithm which uses LP relaxation, and consider an optimal *rounded* solution $\mathrm{ALG}_{\mathrm{P}}$. Then, it holds that:

- if P is a **minimization problem** and $\mathrm{ALG}_{\mathrm{P}} \leq \alpha \mathrm{LP}^*$, then $\alpha \geq \mathrm{IG}_{\mathrm{P}}$
- if P is a **maximixation problem** and $\mathrm{ALG}_{\mathrm{P}} \geq \frac{\mathrm{LP}^*}{\alpha}$, then $\alpha \geq \mathrm{IG}_{\mathrm{P}}$

Proof. Assume that P is a *minimization* problem for which it holds that $\mathrm{ALG}_{\mathrm{P}} \leq \alpha \mathrm{LP}^*$, and by way of contradiction suppose that $\alpha < \mathrm{IG}_{\mathrm{P}}$. Then, for any instance $I \in \mathrm{P}$ it holds that

$$\mathrm{ALG}_{\mathrm{P}}(I) \leq \alpha \mathrm{LP}^*_{\mathrm{P}}(I) < \mathrm{IG}_{\mathrm{P}}(I) \cdot \mathrm{LP}^*_{\mathrm{P}}(I) = \frac{\mathrm{IP}^*_{\mathrm{P}}(I)}{\mathrm{LP}^*_{\mathrm{P}}(I)} \cdot \mathrm{LP}^*_{\mathrm{P}}(I) = \mathrm{IP}^*_{\mathrm{P}}(I)$$

meaning that $\mathrm{ALG}_{\mathrm{P}} < \mathrm{IP}^*_{\mathrm{P}}(I)$ which is a contradiction because $\mathrm{ALG}_{\mathrm{P}}$ solves the IP.

Now, assume that P is a *maximization* problem for which it holds that $\mathrm{ALG}_{\mathrm{P}} \geq \frac{\mathrm{LP}_{\mathrm{P}}{}^*}{\alpha}$, and by way of contradiction suppose that $\alpha < \mathrm{IG}_{\mathrm{P}} = \frac{\mathrm{LP}^*_{\mathrm{P}}}{\mathrm{IP}^*_{\mathrm{P}}} \iff \frac{1}{\alpha} > \frac{\mathrm{IP}^*}{\mathrm{LP}^*}$. Then, for

any instance $I \in \mathrm{P}$ it holds that

$$\mathrm{ALG}_{\mathrm{P}}(I) \geq \frac{\mathrm{LP}^*_{\mathrm{P}}(I)}{\alpha} > \frac{\mathrm{IP}^*_{\mathrm{P}}(I)}{\mathrm{LP}^*_{\mathrm{P}}(I)} \cdot \mathrm{LP}^*_{\mathrm{P}}(I) = \mathrm{IP}^*(I)$$

meaning that $\mathrm{ALG}_{\mathrm{P}}(I) > \mathrm{IP}^*_{\mathrm{P}}(I)$ which is a contradiction, as before. □

Now, let us analyze again VC and try to bound $\mathrm{IG}_{\mathrm{VC}}$. Consider the following *clique* graph:

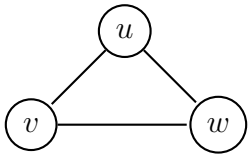


Figure 2.4: The graph K_3.

it is easy to see that

$$\mathrm{IP}^*(K_3) = 1 + 1 = 2$$

because 1 single node is not sufficient to cover all 3 edges in $E(K_3)$. However, since the values in the solution of the LP can be *real-valued*, the value of an optimal solution for the LP relaxation is actually achieved by setting

$$\overline{x}_u = \overline{x}_v = \overline{x}_w = \frac{1}{2} \implies \mathrm{LP}^*(K_3) = 3 \cdot \frac{1}{2} = \frac{3}{2}$$

therefore, by definition of IG we have that

$$\exists I \in \mathrm{P} \quad \mathrm{IG}_{\mathrm{VC}}(I) := \frac{\mathrm{IP}^*_{\mathrm{VC}}(K_3)}{\mathrm{LP}^*_{\mathrm{VC}}(K_3)} = \frac{2}{\frac{3}{2}} = \frac{4}{3} \implies \mathrm{IG}_{\mathrm{VC}} := \sup_{I \in \mathrm{P}} \frac{\mathrm{IP}^*_{\mathrm{VC}}(I)}{\mathrm{LP}^*_{\mathrm{VC}}(I)} \geq \frac{4}{3}$$

Moreover, Theorem 2.1 shows that we already know an algorithm that employs LP relaxation which yields a 2-approximation of VC; therefore, this lower bound on $\mathrm{IG}_{\mathrm{VC}}$ — together with the previous proposition — implies that any possible approximation ratio α on VC must satisfy

$$2 \geq \alpha \geq \mathrm{IG}_{\mathrm{VC}} \geq \frac{4}{3}$$

The following theorem proves that we can actually bound $\mathrm{IG}_{\mathrm{VC}}$ tightly.

Theorem 2.2: Integrality gap for VC

$\mathrm{IG}_{\mathrm{VC}} = 2$

Proof. We already proved that the upper bound is 2, so we just need to prove that the lower bound is 2 as well.

Consider a clique K_n; by the same reasoning presented for the case of K_3, a feasible solution for the LP relaxation over this graph would be

$$x_1 = \ldots = x_n = \frac{1}{2} \implies \mathrm{LP}^*(K_n) \leq n \cdot \frac{1}{2} = \frac{n}{2}$$

Claim: Any minimum vertex cover of K_n has exactly $n-1$ vertices.

Proof of the Claim. Consider a K_n with vertices $V(K_n) = \{v_1, \ldots, v_n\}$, and let $S = \{v_1, \ldots, v_{n-1}\}$ be a set of its vertices. We observe that

- for each edge $\{v_i, v_j\} \in E(K_n)$ such that $i, j \in [n-1]$, clearly S covers the edge
- for each edge $\{v_i, v_n\} \in E(K_n)$, we observe that $v_I \in S$, therefore S still covers the edge

This concludes that $n-1$ vertices are *suffice* to cover $V(K_n)$.

Now, by way of contradiction suppose that there is a vertex cover S' of K_n such that $|S'| \leq n-2$. This implies that there are two vertices $u, v \in V(K_n) - S'$. However, this implies that the edge $\{u, v\} \in E(K_n)$ is not covered — and such always exists since K_n is a complete graph $\lightning$. This proves that $n-1$ vertices are *needed*. □

This claim shows that for any n it holds that $\mathrm{IP}^*(K_n) = n-1$, therefore we have that

$$\mathrm{IG}(K_n) = \frac{\mathrm{IP}^*(K_n)}{\mathrm{LP}^*(K_n)} \geq \frac{n-1}{\frac{n}{2}}$$

which means that

$$\mathrm{IG}_{\mathrm{VC}} = \sup_{I \in \mathrm{VC}} \mathrm{IG}_{\mathrm{VC}}(I) \geq \lim_{n \to +\infty} \frac{n-1}{\frac{n}{2}} = 2$$

□

2.2.2 The Set Cover problem

The next problem that we will study can be seen as a *generalization* of the VC problem, which is the Set Cover problem, defined as follows.

Definition 2.2: Set Cover problem

The **Set Cover** (SC) problem is defined as follows: given a *universe* (or *ground*) set $\mathcal{U} = [n]$, and a collection of sets $C = \{S_1, \ldots, S_m\}$ such that $S_i \subseteq \mathcal{U}$, determine the smallest sub-collection $S \subseteq C$ such that $\bigcup_{S_j \in S} S_j = \mathcal{U}$.

In other words, we are asked to determine the smallest sub-collection of the given C such that we can still cover the whole universe set $\mathcal{U}$. For instance, given $\mathcal{U} = [3]$ and $S_1 = \{1, 2\}$, $S_2 = \{2, 3\}$, $S_3 = \{1, 3\}$, we can cover $\mathcal{U}$ with just $S = \{S_1, S_2\}$.

As for VC, in 1972 [Kar72] proved that SC is NP-complete as well. Moreover, similarly to what we did for VC, we can convert SC into an IP, as follows.

$$\min \sum_{j=1}^{m} x_j$$

$$\sum_{\substack{j \in [m]: \\ i \in S_j}} x_j \geq 1 \quad \forall i \in [n]$$

$$x_j \in \{0, 1\} \quad \forall j \in [m]$$

Figure 2.5: IP for SC.

The first constraint of the IP states that, given an element i in the universe set, at least one of the variables x_j — representing the sets S_j — which contain i, must be set to 1. In other words, we are guaranteeing that all the elements $i \in \mathcal{U}$ are covered by at least one set of C. Lastly, we want to minimize over the size of the sub-collection of C, hence the objective function.

The LP relaxation that we will consider is the same that we defined for VC. However, differently from VC, the *rounding rule* that we applied to obtain an integral solution — namely by defining

$$S = \{v \in V \mid x_v \geq \tfrac{1}{2}\}$$

cannot be applied for this problem. For instance, say that some element $i \in \mathcal{U}$ is contained in 3 sets S_1, S_2 and S_3; hence, the second constraint forces the solution of the LP to satisfy

$$x_1 + x_2 + x_3 \geq 1$$

Nevertheless, by setting

$$x_1 = x_2 = x_3 = \frac{1}{3}$$

we would get a feasible solution for the LP relaxation, but then our rounding rule would return an empty set $S = \varnothing$, which is not a feasible solution for our instance of SC since i would not be covered.

To fix this issue, we are going to present a **randomized rounding algorithm**, which surprisingly seem to be the *best* approach to perform rounding on LP relaxation solutions that we have at our disposal.

Algorithm 2.1: Randomized rounding for SC

Given an instance $(\mathcal{U}, C)$ of SC, the algorithm returns a set cover A for $\mathcal{U}$.

```
1: function RandomizedRoundingSC(𝒰, C)
2:     A := ∅
3:     {x̄_j}_{j∈[m]} := LP_SC(𝒰, C)        ▷ an optimal soluion on the LP relaxation
4:     for k ∈ [⌈2 ln n⌉] do
5:         for j ∈ [m] do
6:             Let c_{k,j} be the outcome of the flip of an ind. coin with H prob. set to x̄_j
7:             if c_{k,j} == H then
8:                 A = A ∪ {S_j}
9:             end if
10:        end for
11:    end for
12: end function
```

First, we are going to prove that the output A of this algorithm is indeed a set cover, with enough probability.

Lemma 2.2

Let $(\mathcal{U}, C)$ be a SC instance, and let $A = \text{RandomizedRoundingSC}(\mathcal{U}, C)$. Then, it holds that

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

Proof. Each iteration of the outermost `for` loop will be referred to as *phase*.

Claim 1: The element i is covered by A in *phase* k with probability at least $1 - \frac{1}{e}$.

Proof of the Claim.

$$\begin{aligned}
\Pr[i \text{ is not covered in } \textit{phase } k] &= \prod_{\substack{j \in [m]: \\ i \in S_j}} (1 - \overline{x}_j) && \text{(the prob. of T)} \\
&\leq \prod_{\substack{j \in [m]: \\ i \in S_j}} e^{-\overline{x}_j} && (1 - x \leq e^{-x}) \\
&= e^{-\sum_{\substack{j \in [m]: \\ i \in S_j}} \overline{x}_j} \\
&\leq e^{-1} && \text{(second constraint of the LP)} \\
&= \frac{1}{e}
\end{aligned}$$

□

Claim 2: The element i is not covered by any set of A with probability at most $\frac{1}{n^2}$.

Proof of the Claim.

$$\begin{aligned}
\Pr[i \text{ is not covered by any set of } A] &= \prod_{k=1}^{\lceil 2\ln n \rceil} \Pr[i \text{ is not covered in } \textit{phase } k] \\
&\leq \prod_{k=1}^{\lceil 2\ln n \rceil} \frac{1}{e} && \text{(by Claim 1)} \\
&= e^{-\lceil 2\ln n \rceil} \\
&\leq e^{-2\ln n} \\
&= \frac{1}{n^2}
\end{aligned}$$

□

Claim 3: A is a set cover with probability at least $1 - \frac{1}{n}$.

Proof of the Claim.

$$\begin{aligned}
\Pr[A \text{ is not a set cover}] &= \Pr[\exists i \quad i \text{ is not covered by any set of } A] \\
&\leq \sum_{i=1}^{n} \Pr[i \text{ is not covered by any set of } A] \\
&\leq \sum_{i=1}^{n} \frac{1}{n^2} && \text{(by Claim 2)} \\
&= \frac{n}{n^2} \\
&= \frac{1}{n}
\end{aligned}$$

□

The statement follows directly from Claim 3. □

Next, we will show that this algorithm yields *on average* a $\lceil 2\ln n \rceil$-approximation of SC.

Lemma 2.3

Let $(\mathcal{U}, C)$ be a SC instance, and let $A = \textsc{RandomizedRoundingSC}(\mathcal{U}, C)$. Then, it holds that

$$\mathbb{E}\left[|A|\right] \leq \lceil 2\ln n \rceil \, \mathrm{IP}^*$$

Proof. Fix a *phase* k, and let A_k be the collection of sets added to A at *phase* k; then it holds that

$$\mathbb{E}\left[|A_k|\right] = \sum_{j=1}^{m} 1 \cdot \Pr[S_j \in A_k] = \sum_{j=1}^{m} \overline{x}_j = \mathrm{LP}^*$$

Moreover, since $A = \bigcup_{k \in \lceil 2 \ln n \rceil} A_k$, we have that

$$\mathbb{E}\left[|A|\right] \leq \mathbb{E}\left[\sum_{k \in \lceil 2 \ln n \rceil} |A_k|\right] = \sum_{k \in \lceil 2 \ln n \rceil} \mathbb{E}\left[|A_k|\right] = \sum_{k \in \lceil 2 \ln n \rceil} \mathrm{LP}^* = \lceil 2 \ln n \rceil \, \mathrm{LP}^* \leq \lceil 2 \ln n \rceil \, \mathrm{IP}^*$$

□

By the law of large numbers, we know that if for each pair $(\mathcal{U}, C)$ we repeatedly compute the previous algorithm we will, eventually, get an output A' such that $|A'| \leq \lceil 2 \ln n \rceil \, \mathrm{IP}^*$. This implies that such procedure yield an approximation ratio $\alpha \leq \lceil 2 \ln n \rceil$, implying that

$$\lceil 2 \ln n \rceil \geq \alpha \geq \mathrm{IG}_{\mathrm{SC}}$$

directly providing an upper bound for SC.

Additionally, we observe that the algorithm can be modified to get a better bound, by simply replacing $\lceil 2 \ln n \rceil$ with $\lceil (1+\varepsilon) \ln n \rceil$, for any $\varepsilon > 0$. In fact, with this modification we would get that $\mathbb{E}\left[|A|\right] \leq \lceil (1+\varepsilon) \ln n \rceil \, \mathrm{LP}^*$, even though we would also have that $\Pr[\mathcal{U} \text{ covered by } A] = 1 - n^{-\varepsilon}$ which is worse for small values of ε — this probability can still be *boosted* by repeatedly computing the output. Hence, we actually get that

$$\mathrm{IG}_{\mathrm{SC}} \leq \lceil \ln n \rceil$$

With VC we were able to provide a tight bound on $\mathrm{IG}_{\mathrm{VC}}$; for SC instead, even though the value of $\mathrm{IG}_{\mathrm{SC}}$ is known [Lov75], the proof is very complex and we will only show a lower bound.

Theorem 2.3: Integrality gap of SC

For any $n \in \mathbb{N}$, it holds that

$$\frac{1}{4 \ln 2} \leq \mathrm{IG}_{\mathrm{SC}} \leq \lceil \ln n \rceil$$

Proof. We already proved the upper bound, so we just need to prove the lower bound. Furthermore, as shown with VC in the previous section, to provide a lower bound on $\mathrm{IG}_{\mathrm{SC}}$ it suffices to show an instance of SC for which the bound holds.

Let $m \geq 2$ be an even integer, and define the following instance of SC: set

$$\mathcal{U}_m := \{e_A \mid A \subseteq [m] \wedge |A| = \tfrac{m}{2}\} \implies n = |\mathcal{U}_m| = \binom{m}{\frac{m}{2}}$$

and define the collection of sets as follows

$$\forall j \in [m] \quad S_j := \{e_A \mid e_A \in \mathcal{U}_m \wedge j \in A\}$$

and set $C_m := \{S_1, \ldots, S_m\}$ For example, if $m = 4$ we have that

$$\mathcal{U}_4 := \{e_A \mid A \subseteq \{1,2,3,4\} \wedge |A| = \tfrac{4}{2} = 2\} = \{e_{\{1,2\}}, e_{\{1,3\}}, e_{\{1,4\}}, e_{\{2,3\}}, e_{\{2,4\}}, e_{\{3,4\}}\}$$

$$S_1 := \{e_{\{1,2\}}, e_{\{1,3\}}, e_{\{1,4\}}\}$$
$$S_2 := \{e_{\{1,2\}}, e_{\{2,3\}}, e_{\{2,4\}}\}$$
$$S_3 := \{e_{\{1,3\}}, e_{\{2,3\}}, e_{\{3,4\}}\}$$
$$S_4 := \{e_{\{1,4\}}, e_{\{2,4\}}, e_{\{3,4\}}\}$$

Note that, thanks to Stirling's approximation we know that

$$n = \binom{m}{\frac{m}{2}} = \Theta\left(\frac{2^m}{\sqrt{m}}\right) \implies m = \log n - \Theta(\log\log n)$$

Claim: $\forall m \geq 2$ even $\quad \mathrm{LP}^*_{\mathrm{SC}}(\mathcal{U}_m, C_m) \leq 2$.

Proof of the Claim. Consider the solution

$$x_1 = \ldots = x_m = \frac{2}{m}$$

for the LP relaxation; clearly, $m \geq 2$ therefore $\forall j \in [m] \quad 0 \leq x_j \leq 1$, and

$$\forall e_A \in \mathcal{U} \quad \sum_{\substack{j \in [m]: \\ e_A \in S_j}} x_j = \sum_{\substack{j \in [m]: \\ e_A \in S_j}} \frac{2}{m} = |A| \cdot \frac{2}{m} = \frac{m}{2} \cdot \frac{2}{m} = 1 \geq 1$$

hence this is a feasible solution to the LP, and its value is simply given by $m \cdot \frac{2}{m} = 2$. □

Claim: $\forall m \geq 2$ even $\quad \mathrm{IP}^*(\mathcal{U}_m, C_m) \geq \frac{1}{2}\log n - O(\log\log n)$.

Proof of the Claim. By way of contradiction, assume that there exists a sub-collection $S = \{S_{i_1}, \ldots, S_{i_k}\} \subseteq C_m$ with $k \leq \frac{m}{2}$ that covers $\mathcal{U}_m$. Consider the following set

$$T := [m] - \{i_1, \ldots, i_k\}$$

thus, we have that

$$|T| \geq m - \frac{m}{2} = \frac{m}{2}$$

hence, we can always define a subset $A \subseteq T$ such that $|A| = \frac{m}{2}$. However, we observe that

$$A \subseteq T \implies i_1, \ldots, i_k \notin A \implies e_A \notin S_{i_1} \cup \ldots \cup S_{i_k}$$

hence e_A is not covered by S ↯.

This means that for any set cover S of $(\mathcal{U}_m, C_m)$ it must hold that

$$|S| > \frac{m}{2} = \frac{1}{2}\log n - \Theta(\log\log n) \implies \mathrm{IP}^*(\mathcal{U}_m, C_m) \geq \frac{m}{2} = \frac{1}{2}\log n - O(\log\log n)$$

□

Finally, from these two claims, it follows that

$$\mathrm{IG}_{\mathrm{SC}}(\mathcal{U}_m, C_m) = \frac{\mathrm{IP}^*(\mathcal{U}_m, C_m)}{\mathrm{LP}^*(\mathcal{U}_m, C_m)} \geq \frac{\frac{1}{2}\log n - O(\log\log n)}{2} = \frac{1}{4}\log n - O(\log\log n)$$

which means that

$$\mathrm{IG}_{\mathrm{SC}} = \sup_{I \in \mathrm{SC}} \mathrm{IG}_{\mathrm{SC}}(I) \geq \mathrm{IG}_{\mathrm{SC}}(\mathcal{U}_m, C_m) \geq \frac{1}{4}\log n = \frac{1}{4\ln 2}\ln n$$

□

2.2.3 The Densest Subgraph problem

Up to this point, we only discussed NP-complete problems, and various techniques used to obtain approximate solutions. The next problem that we are going to introduce, instead, can be solved in **polynomial** time, thanks to linear programming. First, consider the following definition.

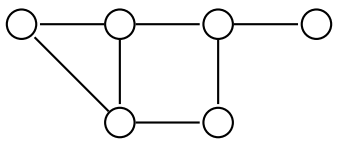
Definition 2.3: Graph density

Given a graph $G = (V, E)$ such that $V \neq \varnothing$, its **density** is defined as follows

$$\rho(G) := \frac{|E|}{|V|}$$

Note that we require $V \neq \varnothing$, but we will omit this detail in the following sections.

For instance, the density of the graph $G = (V, E)$ below



is evaluated through the following ratio

$$\rho(G) := \frac{|E|}{|V|} = \frac{7}{6} = 1.1\overline{6}$$

The problem that we are going to introduce is the following.

Definition 2.4: Densest Subgraph problem

The **Densest Subgraph** (DS) problem is defined as follows: given an undirected graph $G = (V, E)$, determine the induced subgraph of G that maximizes its density. In other words, the problem asks to find a set of vertices S^* in

$$S^* \in \arg\max_{S \subseteq V} \rho(G[S])$$

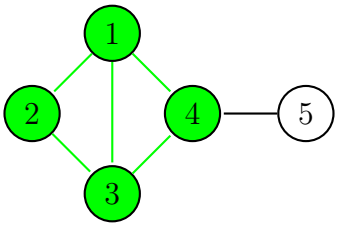


Figure 2.6: For instance, in this graph the set S that maximizes $\rho(G[S])$ is given by $S = \{1, 2, 3, 4\}$, which yields a density of $\frac{5}{4} = 1.25$.

The densest subgraph is relevant because it actually *maximizes* the average degree of its vertices. In fact, thanks to the handshaking lemma we have that

$$\rho(G[S]) := \frac{|E(G[S])|}{|S|} = \frac{1}{2} \cdot \frac{2 \cdot |E(G[S])|}{|s|} = \frac{1}{2} \cdot \frac{\sum_{v \in S} \deg(v)}{|S|} = \frac{1}{2} \operatorname*{avg}_{v \in S} \deg(v)$$

The LP that we will present for this problem is the following. However, to show that this LP actually yields a solution for DS is not trivial, and it will be proven through the following theorems.

$$\max \sum_{e \in E} x_e$$

$$\begin{aligned}
x_{ij} &\leq y_i && \forall ij \in E \\
x_{ij} &\leq y_j && \forall ij \in E \\
\sum_{i \in V} y_i &\leq 1 && \\
x_{ij} &\geq 0 && \forall ij \in E \\
y_i &\geq 0 && \forall i \in V
\end{aligned}$$

Figure 2.7: LP for DS.

Lemma 2.4

Given a graph $G = (V, E)$, for any $S \subseteq V$ there exists a feasible solution $\{x_e, y_i\}_{e \in E, i \in V}$ to the LP having value at least $\rho(G[S])$.

Proof. Given a set $S \subseteq V$, construct the following solution:

- $\forall ij \in E \quad x_{ij} := \begin{cases} \frac{1}{|S|} & i, j \in S \\ 0 & i \notin S \vee j \notin S \end{cases}$
- $\forall i \in V \quad y_i := \begin{cases} \frac{1}{|S|} & i \in S \\ 0 & i \in V - S \end{cases}$

We observe that

- $ij \in E(G[S]) \implies i,j \in S \implies \begin{cases} x_{ij} = \dfrac{1}{|S|} \\ y_i = y_j = \dfrac{1}{|S|} \end{cases} \implies x_i j \leq y_i, y_j$
- $ij \notin E(G[S]) \implies i \notin S \vee j \notin S \implies x_{ij} = 0$, but since $y_i, y_j \geq 0$ for any $i,j \in V$ by definition, it holds that $x_{ij} \leq y_i, y_j$

Additionally, notice that

$$\sum_{i \in V} y_i = \sum_{i \in S} y_i + \sum_{i \in V - S} y_i = |S| \cdot \frac{1}{|S|} + |V - S| \cdot 0 = 1 \leq 1$$

This proves that this is a feasible solution to the LP. Lastly, the value of this solution is simply given by

$$\sum_{e \in E} x_e = \sum_{e \in E(G[S])} e + \sum_{e \notin E(G[S])} x_e = |E(G[S])| \cdot \frac{1}{|S|} + |E - E(G[S])| \cdot 0 = \frac{|E(G[S])|}{|S|} =: \rho(G[S])$$

□

In particular, this lemma proves that if S^* is an optimal solution for DS on a graph G, then there is a solution to this LP such that $\text{LP}^* \geq \rho(G[S^*])$. The next theorem will prove that $\text{LP}^* \leq \rho(G[S^*])$ holds as well, which implies that our LP does in fact produce an optimal solution to DS, and that $\text{IG}_{\text{DS}} = 1$.

Theorem 2.4

Given a graph $G = (V, E)$, for any $\{x_e, y_i\}_{e \in E, i \in V}$ feasible solution to the LP that has value v, there exists a set $S \subseteq V$ such that $\rho(G[S]) \geq v$.

Proof. Consider a feasible solution $\{x'_e, y'_i\}_{e \in E, i \in V}$ of the LP, having value v. We will construct another solution to the LP $\{x_e, y_i\}_{e \in E, i \in V}$ starting from the previous solution, as follows:

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

It is easy to see that this is a feasible solution to the LP, in fact

$$\forall ij \in E \quad x_{ij} := \min(y_i, y_j) \leq y_i, y_j$$

and additionally, by feasibility of $\{x'_e, y'_i\}_{e \in E, i \in V}$, we have that

$$\sum_{i \in V} y_i = \sum_{i \in V} y'_i \leq 1$$

Furthermore, note that

$$v = \sum_{e \in E} x'_e \leq \sum_{e \in E} \min(y'_i, y'_j) = \sum_{e \in E} \min(y_i, y_j) = \sum_{e \in E} x_e$$

meaning that the value of this solution is at least v, but it may be better — this happens if $x_{ij} < y_i$ or $x_{ij} < y_j$ for some $ij \in E$, and if this is the case we say that there is some *slack*.

Next, we will construct a series of sets of vertices starting from this new LP solution we just defined. In particular, given a value $r \geq 0$, let

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

$$E(r) := \{e \mid x_e \geq r\}$$

Notice that $ij \in E(r) \iff i, j \in S(r)$, because

$$ij \in E(r) \iff x_{ij} \geq r \iff \min(y_i, y_j) \geq r \iff y_i, y_j \geq r \iff i, j \in S(r)$$

which means that $S(r)$ and $E(r)$ are well-defined — note that $E(G[S(r)]) = E(r)$.

Claim: There exists a value $r^* \geq 0$ such that $\rho(G[S(r^*)]) \geq v$.

Proof of the Claim. Consider the solution $\{x_e, y_i\}_{e \in E, i \in V}$ to the LP that we constructed previously, and let π be the permutation of the y_i's such that

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

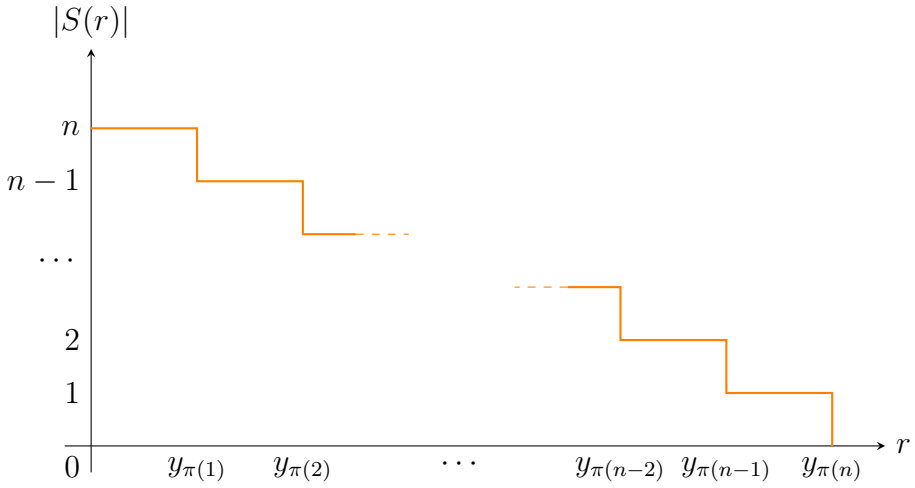
which means that π defines a sorting of the y_i's. Now, consider the possible values of $r \geq 0$, and what happens inside $S(r)$:

- for $r = 0$, by feasibility of our solution we have that

$$S(r) = S(0) := \{i \mid y_i \geq 0\} \implies |S(r)| = n$$

- notice that for any value $0 \leq r \leq y_{\pi(1)} \leq \ldots \leq y_{\pi(n)}$, the value of $|S(r)|$ will still be n, by its definition
- however, if $0 \leq y_{\pi(1)} < r \leq \ldots \leq y_{\pi(n)}$, then the vertex $\pi(1)$ will be the only verterx not contained in $S(r)$, therefore $|S(r)| = n - 1$

Repeating the same argument for all the possible values of r, we obtain the following graph for $|S(r)|$.



From this graph, it is easy to see that computing the integral of $|S(r)|$ is trivial, because

$$\begin{aligned}
\int_0^{y_{\pi(n)}} |S(r)|\, dr &= \int_0^{y_{\pi(1)}} |S(r)|\, dr + \int_{y_{\pi(1)}}^{y_{\pi(2)}} |S(r)|\, dr + \ldots + \int_{y_{\pi(n-1)}}^{y_{\pi(n)}} |S(r)|\, dr \\
&= \int_0^{y_{\pi(1)}} n\, dr + \int_{y_{\pi(1)}}^{y_{\pi(2)}} (n-1)\, dr + \ldots + \int_{y_{\pi(n-1)}}^{y_{\pi(n)}} 1\, dr \\
&= n \cdot (y_{\pi(1)} - 0) + (n-1) \cdot (y_{\pi(2)} - y_{\pi(1)}) + \ldots + 1 \cdot (y_{\pi(n)} - y_{\pi(n-1)}) \\
&= y_{\pi(1)} \cdot [n - (n-1)] + y_{\pi(2)} \cdot [(n-1) - (n-2)] + \ldots + y_{\pi(n)} \cdot (1 - 0) \\
&= y_{\pi(1)} \cdot 1 + y_{\pi(2)} \cdot 1 + \ldots + y_{\pi(n)} \cdot 1 \\
&= \sum_{i=1}^{n} y_{\pi(i)} \\
&= \sum_{i \in V} y_i \\
&\leq 1 \qquad \text{(by feasibility of the solution)}
\end{aligned}$$

By the same argument, we can derive that

$$\int_0^{y_{\pi(n)}} |E(r)|\, dr = \sum_{e \in E} x_e \geq v$$

Note that the last inequality was proved before the claim.

Lastly, we will prove the statement of the claim by way of contradiction. In particular, suppose that for all $r \geq 0$ it holds that $\rho(G[S(r)]) < v$. We can rewrite this inequality as follows

$$\rho(G[S(r)]) := \frac{|E(r)|}{|S(r)|} < v \iff |E(r)| < v \cdot |S(r)|$$

This inequality can be used to upper bound the integral of $|E(r)|$ *point-by-point*, i.e.

$$v \leq \int_0^{y_{\pi(n-1)}} |E(r)|\, dr < \int_0^{y_{\pi(n-1)}} v \cdot |S(r)|\, dr = v \cdot \int_0^{y_{\pi(n-1)}} |S(r)|\, dr \leq v \cdot 1 = v$$

meaning that $v < v$ ↯. □

Finally, since we proved that $G[S(r)]$ is well-defined, and this claim proves that there exists an r^* such that $\rho(G[S(r^*)]) \geq v$, the statement holds. □

2.2.4 Approximation through duality

As we discussed at the beginning of this section, the cost of solving an LP generally depends on the number of *variables* and *constraints* it has. In particular, the LP we presented for solving DS has a *large* number of both, which means that while the cost remains *polynomial*, the degree of the polynomial is too high, making this approach **impractical** for large graphs.

Now, we are going to present a **greedy algorithm**, developed by Charikar [Cha00] in 2000, and we will prove that this algorithm yields a $\frac{1}{2}$-approximation of DS.

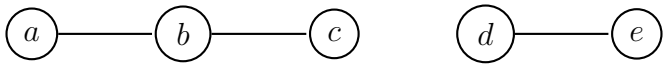
Algorithm 2.2: Charikar's algorithm

Given an undirected graph $G = (V, E)$, the algorithm returns a $\frac{1}{2}$-approximation solution of DS on G.

1: **function** CHARIKAR(G)
2: $S_0 := V(G)$
3: **for** $i \in [n-1]$ **do**
4: $v_i \in \arg\min_{v \in S_{i-1}} \deg_{G[S_{i-1}]}(v)$
5: $S_i := S_{i-1} - \{v_i\}$
6: **end for**
7: **return** $S^* \in \arg\max_{i \in [0,n-1]} \rho(G[S_i])$
8: **end function**

Idea. The algorithm constructs a series $S_0, \ldots, S_n$ of sets of vertices, such that $|S_i| = n-i$, and at each iteration the i-th vertex that gets removed from $G[S_{i-1}]$ is the one having minimum degree. At the end, the algorithm returns the set S^* that maximizes the density of $G[S^*]$.

We observe that this algorithm does not return an optimal solution in general; in particular, problems can arise if there are vertices of equal degree. For instance, consider the following disconnected graph



In this graph, the densest subgraph is induced by the set $S^* = \{a, b, c\}$, because

$$\rho(G[S^*]) = \frac{2}{3}$$

However, since $\deg(a) = \deg(c) = \deg(d) = \deg(e) = 1$, the algorithm may choose $v_1 := a$, meaning that $G[S_1]$ would be

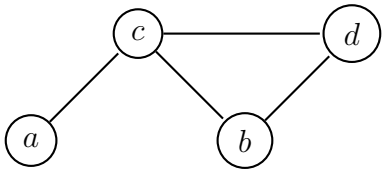
and it is easy to see that $G[S_0] = G$, and any other subgraph of this graph, have lower density than $G[S^*]$.

To prove that this algorithm yields a $\frac{1}{2}$-approximation, we are going to introduce some definitions first.

Definition 2.5: Orientation

Given an undirected graph $G = (V, E)$, an **orientation** of G is a function $\phi : E \to V$ that assigns to each edge $\{u, v\} \in E$ either u or v.

Note that, by definition for any $e \in E$ it holds that $\phi(e) \in e$. For instance, given the following graph



and the following function ϕ

e	$\phi(e)$
ac	c
bc	c
bd	b
cd	d

we would get the following orientation on G

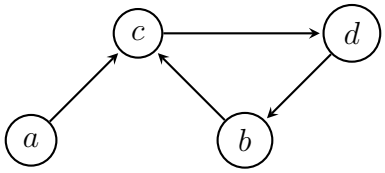


Figure 2.8: The previous graph, oriented through ϕ.

Given an orientation ϕ of G, and a vertex $v \in V(G)$ we will indicate with $\deg_\phi(v)$ the in-degree of v w.r.t. ϕ, i.e.

$$\deg_\phi(v) := |\{e \in E \mid \phi(e) = v\}|$$

For example, in the previous graph we have that $\deg_\phi(c) = 2$, $\deg_\phi(a) = 0$ and $\deg_\phi(b) = \deg_\phi(d) = 1$. By the same argument that proves the handshaking lemma, we observe that

$$\sum_{v \in V(G)} \deg_\phi(v) = |E|$$

Lastly, the maximum in-degree w.r.t. ϕ will be denoted as

$$\Delta_\phi := \max_{v \in V} \deg_\phi(v)$$

We are now ready to prove the approximation ratio of Charikar's algorithm.

Theorem 2.5

Given a graph G, and an optimal solution $\rho(G[S^*])$ for DS on G, let $M := \textsc{Charikar}(G)$; then, it holds that $\frac{1}{2}\rho(G[S^*]) \leq M$.

Proof. We are going to prove this approximation ratio through two claims.

Claim 1: Given an undirected graph G, and an optimal solution $\rho(G[S^*])$ for DS on G, for any orientation ϕ of G it holds that $\rho(G[S^*]) \leq \Delta_\phi$.

Proof of the Claim. By definition of ϕ, each $uv \in E$ is going to be oriented towards either u or v, therefore if $uv \in E(G[S^*])$ then uv is going to be oriented towards a node of S. Hence, we have that

$$|E(G[S^*])| \leq \sum_{v \in V(G[S^*])} \deg_\phi(v) \leq \sum_{v \in V(G[S^*])} \Delta_\phi = |V[G(S^*)]| \cdot \Delta_\phi \iff \rho(G[S^*]) := \frac{|E(G[S^*]|)}{|V(G[S^*])|} \leq \Delta_\phi$$

□

Consider Charikar's algorithm; before proving the second claim, we need to introduce an orientation ϕ_{GR} — where GR stands for *greedy* — of G defined as follows: if v_i is the i-th vertex — removed from $G[S_{i-1}]$ — picked by the algorithm, orient any edge $\{u, v_i\} \in E(G)$ towards v_i, for any $u \in V(G)$. For instance, Charikar's algorithm applied on the graph shown in Figure 2.8 would set $v_1 := a$, therefore $\phi(ca) = a$, and so on. This implies that

$$\forall i \in [n] \quad \deg_{G[S_{i-1}]}(v_i) = \deg_{\phi_{\text{GR}}}(v_i)$$

Claim 2: Given an undirected graph G, and $M := \textsc{Charikar}(G)$, it holds that $\Delta_{\phi_{\text{GR}}} \leq 2M$.

Proof of the Claim. Observe that for any $i \in [n]$ it holds that

$$v_i \in \arg\min_{v \in S_{i-1}} \deg_{G[S_{i-1}]}(v) \iff \deg_{G[S_{i-1}]}(v_i) = \min_{v \in S_{i-1}} \deg_{G[S_{i-1}]}(v)$$

therefore, we have that

$$\begin{aligned}
\deg_{\phi_{\mathrm{GR}}}(v_i) &= \deg_{G[S_{i-1}]}(v_i) \\
&= \min_{v \in S_{i-1}} \deg_{G[S_{i-1}]}(v) \\
&\leq \operatorname*{avg}_{v \in S_{i-1}} \deg_{G[S_{i-1}]}(v) \\
&= \frac{\displaystyle\sum_{v \in S_{i-1}} \deg_{G[S_{i-1}]}(v)}{|S_{i-1}|} \\
&= \frac{2\,|E(G[S_{i-1}])|}{|V(G[S_{i-1}])|} \\
&=: 2\rho(G[S_{i-1}])
\end{aligned}$$

Finally, we get that

$$\begin{aligned}
\Delta_{\phi_{\mathrm{GR}}} &:= \max_{v \in V(G)} \deg_{\phi_{\mathrm{GR}}}(v) \\
&= \max_{i \in [n]} \deg_{\phi_{\mathrm{GR}}}(v_i) \\
&\leq \max_{i \in [n]} 2\rho(G[S_{i-1}]) \qquad \text{(for the previous inequality)} \\
&= 2 \max_{i \in [n]} \rho(G[S_{i-1}]) \\
&\leq 2 \max_{i \in [0,n-1]} \rho(G[S_i]) \\
&=: 2M
\end{aligned}$$

□

Lastly, putting the two claims together, we get that for any optimal solution $\rho(G[S^*])$ of a given undirected graph G, if $M := \textsc{Charikar}(G)$ then

$$\rho(G[S^*]) \leq \Delta_{\phi_{\mathrm{GR}}} \leq 2M \iff \tfrac{1}{2}\rho(G[S^*]) \leq M$$

□

2.3 Approximation through duality

In the last proof of the previous section, we defined ϕ_{GR}, which helped us determine the approximation ratio of the algorithm. But how did Charikar come up with such an orientation of the graph? To answer this question, we need to introduce the concept of the **duality** of linear programs.

Consider the following *maximization* generic linear program:

$$\max\ c_1x_1 + \ldots + c_nx_n$$

$$\begin{gathered} a_{11}x_1 + \ldots + a_{1n}x_n \leq b_1 \\ \vdots \\ a_{m1}x_1 + \ldots + a_{mn}x_n \leq b_m \\ x_1, \ldots, x_n \geq 0 \end{gathered}$$

Figure 2.9: A *primal* LP.

As a convention, we will call the **primal** LP the one that *maximizes* its objective function. Hence, given a primal LP, its **dual** linear program is defined as follows:

$$\min\ b_1y_1 + \ldots + b_my_m$$

$$\begin{gathered} a_{11}y_1 + \ldots + a_{m1}y_m \geq c_1 \\ \vdots \\ a_{1n}y_1 + \ldots + a_{mn}y_m \geq c_n \\ y_1, \ldots, y_m \geq 0 \end{gathered}$$

Figure 2.10: The *dual* of a primal LP.

In other words, given a matrix $A \in \mathbb{R}^{m \times n}$, two vectors $c, x \in \mathbb{R}^n$, and two vectors $b, y \in \mathbb{R}^m$, these two programs can be rewritten as follows:

Primal LP		Dual LP
$\max\ c^Tx$	$\Longleftrightarrow$	$\min\ b^Ty$
$Ax \leq b$		$A^Ty \geq c$
$x \geq 0$		$y \geq 0$

Observe that, by definition, the dual of the dual of a primal LP is the primal LP itself. Moreover, we can relate the solutions of primal LPs and their duals through the following two theorems.

Theorem 2.6: Weak duality theorem

Given a feasible solution x for a primal LP, and a feasible solution y for its dual LP, such that both x and y exist and are finite, it holds that

$$c^T x \leq b^T y$$

Proof. Given any two matrices $N \in \mathbb{R}^{m \times n}$, $M \in \mathbb{R}^{n \times l}$ it holds that

$$(NM)^T = M^T N^T$$

therefore, we have that

$$\begin{aligned} c^T x &= x^T c \\ &\leq x^T A^T y && \text{(by feasibility of } y\text{)} \\ &= (Ax)^T y && \text{(by the previous observation)} \\ &\leq b^T y && \text{(by feasibility of } x\text{)} \end{aligned}$$

□

Theorem 2.7: Strong duality theorem

Given an optimal solution x for a primal LP, and an optimal solution y for its dual LP, such that both x and y exist and are finite, it holds that

$$c^T x = b^T y$$

Going back to our problem, consider again the LP for DS presented in Figure 2.7. We observe that its dual LP is the following linear program:

$$\min \ \delta$$

$$\begin{aligned} \gamma_{ij} + \gamma_{ji} &\geq 1 && \forall ij \in E \\ \delta - \sum_{\substack{j \in V: \\ ij \in E}} \gamma_{ij} &\geq 0 && \forall i \in V \\ \delta &\geq 0 \\ \gamma_{ij} &\geq 0 && \forall ij \in E \\ \gamma_{ji} &\geq 0 && \forall ij \in E \end{aligned}$$

Figure 2.11: The dual of the LP for DS.

Notice that this is the LP that corresponds to the **Minimum Max-Degree Orientation** problem, which asks to find the edge orientation with the lowest possible maximum degree.

In fact, observe that

$$\forall i \in V \quad \delta - \sum_{\substack{j \in V: \\ ij \in E}} \gamma_{ij} \geq 0 \iff \delta \geq \deg(i)$$

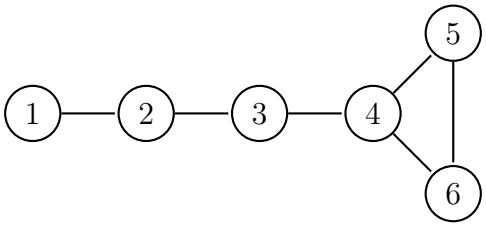
Even though this approach yields a $\frac{1}{2}$-approximation algorithm that runs in only n iterations, it may still be too slow for very large graphs. Can we improve the algorithm further? Consider how CHARIKAR algorithm works: at each iteration, it removes the vertex with the minimum degree from the current graph. What if, instead of removing just one such vertex per iteration, we remove all vertices of minimum degree *at once*?

Algorithm 2.3: Charikar's algorithm (improved version)

Given an undirected graph $G = (V, E)$, the algorithm returns a $\frac{1}{2}$-approximation solution of DS on G.

1: **function** CHARIKARIMPROVED(G)
2: $S_0 := V(G)$
3: $i := 0$
4: **while** $S_i \neq \varnothing$ **do**
5: $A_i := \arg\min_{u \in S_i} \deg_{G[S_i]}(u)$
6: $S_{i+1} := S_i - A_i$
7: **end while**
8: **return** $S^* \in \arg\max_{i \in \mathbb{N}} \rho(G[S_i])$
9: **end function**

This algorithm is able to reduce significantly the number of iterations, by removing the vertices of minimum degree from S_i all at once. However, in the worst case the number of iterations is still n. For instance, consider the following graph



In this graph, our improved version of Charikar would still need n iterations to remove all the vertices — in the figure, the node labeled with the number i is the only node present in A_i at the i-th iteration of the algorithm. Can we do better?

Consider the proof of Theorem 2.5; the second claim of the theorem uses the fact that the vertex v_i has minimum degree in $G[S_{i-1}]$. However, note that the only reason why we

need this fact is to bound

$$\deg_{G[S_{i-1}]}(v_i) \leq \operatorname*{avg}_{v \in S_{i-1}} \deg_{G[S_{i-1}]}(v_i)$$

Hence, we may try to use the *average degree* instead of the *minimum degree*, in the definition of A_i.

Consider a path P_n; observe that CharikarImproved(P_n) would require $\lceil \frac{n}{2} \rceil$ iteration to remove all the vertices from the initial graph, because it would remove at most (for the case when n is odd) 2 vertices per iteration — i.e. the endpoints. What happens if we replace the definition of A_i as follows?

$$A_i := \left\{ v \in S_i \mid \deg_{G[S_i]}(v) \leq \operatorname*{avg}_{u \in S_i} \deg_{G[S_i]}(u) \right\}$$

In the case of P_n, CharikarImproved would still fail at processing the graph all at once, because the average degree is *less than* 2. Nonetheless, this problem suggests the following approach.

Algorithm 2.4: Charikar's algorithm (ε version)

Given an undirected graph $G = (V, E)$, the algorithm returns a $\frac{1}{2(1+\varepsilon)}$-approximation solution of DS on G.

---

function Charikar$_\varepsilon(G)$
  $S_0 := V(G)$
  $i := 0$
  while $S_i \neq \varnothing$ **do**
    $A_i := \left\{ v \in S_i \mid \deg_{G[S_i]}(v) \leq (1+\varepsilon) \operatorname*{avg}_{u \in S_i} \deg_{G[S_i]}(u) \right\}$
    $S_{i+1} := S_i - A_i$
  end while
  return $S^* \in \arg\max_{i \in \mathbb{N}} \rho(G[S_i])$
end function

Theorem 2.8

Given a graph G, and $S :=$ Charikar$_\varepsilon$(G), let S^* be an optimal solution to DS on G. Then, for all $\varepsilon > 0$ it holds that $|S| \geq \frac{1}{2(1+\varepsilon)} |S^*|$. Moreover, the algorithm runs in at most $O(\log_{1+\varepsilon}(n))$ iterations.

Proof. Consider the optimal solution S^*; note that $S^* \neq \varnothing$ because $\rho(G[\varnothing])$ is not defined, and if $|S^*| = 1$, then $|E(G[S^*])| = 0 \implies \rho(G[S^*]) = 0$ meaning that any solution is optimal. Hence, we may assume that $|S^*| \geq 2$.

Claim: $\forall v \in S^* \quad \deg_{G[S^*]}(v) \geq \rho(G[S^*])$.

Proof of the Claim. Fix a vertex $v \in S^*$. By optimality of S, we know that $\rho(G[S^*]) \geq \rho(G[S^* - \{v\}])$. Moreover, note that

$$\rho(G[S^* - \{v\}]) = \frac{|E(G[S^* - \{v\}])|}{|S^* - \{v\}|} = \frac{|E(G[S^*])| - \deg_{G[S^*]}(v)}{|S^*| - 1}$$

Therefore, we have that

$$\rho(G[S^*]) \geq \rho(G[S^* - \{v\}]) \iff \frac{|E(G[S^*])|}{|S^*|} \geq \frac{|E(G[S^*])| - \deg_{G[S^*]}(v)}{|S^*| - 1}$$

and the claim follows by solving the inequality. □

Now, we observe that by construction of the algorithm, at least one node is removed in each iteration, since at least the vertex with minimum degree will be selected, hence

$$\min_{v \in S_i} \deg_{G[S_i]}(v) \leq \operatorname*{avg}_{v \in S_i} \deg_{G[S_i]}(v) \leq (1+\varepsilon) \operatorname*{avg}_{v \in S_i} \deg_{G[S_i]}(v)$$

Thus, the alogrithm will eventualy select a verex of S^*. Let i be the first iteration such that $A_i \cap S^* \neq \varnothing$. Since i is the smallest iteration with such property, it must hold that $S^* \subseteq S_i$. Hence, for any vertex $v_i \in A_i \cap S^*$ it holds that

$$\rho(G[S^*]) \leq \deg_{G[S^*]}(v) \leq \deg_{G[S_i]}(v) \leq (1+\varepsilon) \operatorname*{avg}_{v \in S_i} \deg_{G[S_i]}(v) = 2(1+\varepsilon)\rho(G[S_{i+1}])$$

concluding that

$$\rho(G[S_i]) \geq \frac{1}{2(1+\varepsilon)} \rho(G[S^*])$$

which means that the solution S returned by the algorithm will be at least as good as a $\frac{1}{2(1+\varepsilon)}$-approximation

$$S \in \arg\max_{j \in [n]} \rho(G[S_j]) \geq \rho(G[S_i]) \geq \frac{1}{2(1+\varepsilon)} \rho(G[S^*])$$

Now, fix an iteration $k \in [n]$; we observe that

$$\begin{aligned}
2\,|E(G[S_k])| &= \sum_{v \in S_k} \deg_{G[S_k]}(v) \\
&= \sum_{v \in A_k} \deg_{G[S_k]}(v) + \sum_{v \in S_k - A_k} \deg_{G[S_k]}(v) \\
&\geq \sum_{v \in S_k - A_k} \deg_{G[S_k]}(v) \\
&> \sum_{v \in S_k - A_k} (1+\varepsilon) \operatorname*{avg}_{u \in S_k} \deg_{G[S_k]}(v) \\
&= (|S_k| - |A_k|)(1+\varepsilon) \operatorname*{avg}_{u \in S_k} \deg_{G[S_k]}(u) \\
&= |S_{k+1}|\,(1+\varepsilon) \frac{2\,|E(G[S_k])|}{|S_k|}
\end{aligned}$$

Thus, we get that

$$2\,|E(G[S_k])| > |S_{k+1}|\,(1+\varepsilon)\frac{2\,|E(G[S_k])|}{|S_k|} \implies 1 > \frac{|S_{k+1}|}{S_k}(1+\varepsilon) \implies |S_{k+1}| < \frac{|S_k|}{1+\varepsilon}$$

Finally, let i^* be the last iteration of the algorithm; since $|S_0| = n$ and i^* is reached when $|S_{i^*}| = 1$, we conclude that

$$|S^*| < \frac{n}{(1+\varepsilon)^{i^*}} \implies 1 < \frac{n}{(1+\varepsilon)^{i^*}} \implies i^* \leq \log_{1+\varepsilon} n$$

□

2.4 Approximation through semidefinite programming

In Section 1.1.1 we mentioned that the best known approximation ratio for the Max-Cut problem is around 0.878. We are now ready to discuss this ratio.

First, let's look at the approaches that we used for the approximation we presented: we provided a $\frac{1}{2}$-approximation for Max-Cut that does not even look at the structure of the graph itself, and simply flips coins in order to decide if the corresponding vertex has to be added to the output.

Now, let us build an IP that models the Max-Cut problem: for each edge $\{i, j\}$ define a variable y_{ij}, and for each vertex define a variable x_v.

$$\max \sum_{ij \in E(G)} y_{ij}$$

$$\begin{array}{ll} y_{ij} \leq x_i + x_j & \forall \{i,j\} \in E(G) \\ y_{ij} \leq 2 - (x_i + x_j) & \forall \{i,j\} \in E(G) \\ x_v \in \{0,1\} & \forall u \in V(G) \\ y_{ij} \in \{0,1\} & \forall \{i,j\} \in E(G) \end{array}$$

Figure 2.12: IP for MC.

We observe that the constraints on y_{ij} ensure that $y_{ij} = 1$ if and only if $x_i \neq x_j$. Moreover, we don't need to force $y_{ij} = 1$ when i is in the set and j is not (or vice versa) because the *maximization* will enforce this in the solution automatically.

Now, let us consider the LP relaxation of this IP: the optimal solution of such relaxation will always set every single vertex variable x_v to $\frac{1}{2}$, and every edge variable y_{ij} to 1. In other words, the optimal solution to the LP relaxation gives *no information*. Indeed, this is exactly the reason why the $\frac{1}{2}$-approximation we presented can just flip coints for every vertex without even looking at the structure of the input.

Then, how good can this relaxation get? Since our algorithm is a $\frac{1}{2}$-approximation, we already know that $\mathrm{IG}_{\mathrm{MC}} \leq 2$. Moreover, it can be proven that for any $n \in \mathbb{N}$ the gap of K_n is at least $2 - \varepsilon$ for $\varepsilon > 0$, concluding that $\mathrm{IG}_{\mathrm{MC}} = 2$.

Despite many efforts, for a very long time researchers believed that no improvement over this result could be achieved. However, in a groundbreaking paper published in 1995 Goemans and Williamson [GW95] proved everyone wrong, by completely changing the approach: instead of using an IP, they modeled the Max-Cut problem through a **Quadratic Program** (QP). The following is an example of a generic QP.

$$\max \ \frac{1}{2} x^T Q x + c^T x$$

$$Ax \leq b$$
$$x \in V$$

Figure 2.13: Standard form of a QP.

The QP that exactly models the Max-Cut is very similar to the IP we previously discussed, with the exception of edge constraints being "implicit" in the objective function.

$$\max \sum_{ij \in E(G)} \frac{1 - x_i x_j}{2}$$

$$x_v \in \{-1, 1\} \qquad \forall v \in V(G)$$

Figure 2.14: QP for MC.

In fact, we observe that for any edge $\{i, j\} \in E(G)$ we have that

$$\frac{1 - x_i x_j}{2} = \begin{cases} 0 & x_i = x_j \\ 1 & x_i \neq x_j \end{cases}$$

Nonetheless, solving quadratic programs is also NP-hard. However, the *ingenious* idea of Goemans and Williamson was to *cast* this QP into a **Semidefinite Prorgam** (SDP).

$$\max \sum_{i,j \in [n]} c_{i,j} \langle v_i, v_j \rangle$$

$$\begin{aligned} \max \sum_{i,j \in [n]} a_{i,j,k} \langle v_i, v_j \rangle \leq b_k \quad & \forall k \in [n] \\ v_i \in \mathbb{R}^n \quad & \forall i \in [n] \end{aligned}$$

Figure 2.15: Standard form of an SDP.

Differently from IPs and QPs, SDPs work on *vector variables*, where $\langle v_i, v_j \rangle$ denotes the dot product between the two vectors. However, SDPs can also be solved in polynomial time — like LPs — through an extension of the Ellipsoid method.

The following is Goemans and Williamson's SDP relaxation for the Max-Cut problem:

$$\max \sum_{ij \in E(G)} \frac{1 - \langle v_i, v_j \rangle}{2}$$

$$\begin{aligned} \langle v_i, v_i \rangle = 1 \quad & \forall i \in [n] \\ v_i \in \mathbb{R}^n \quad & \forall i \in [n] \end{aligned}$$

Figure 2.16: SPD for MC.

Let $\overline{x}$ be a feasible solution to the QP. If we map each component $\overline{x}_i$ with the vector $v_i = [\overline{x}_i \ 0 \ \cdots \ 0]^T$, we see that

$$\langle v_i, v_j \rangle = \sum_{k \in [n]} v_k(i) \cdot v_k(j) = \overline{x}_i \overline{x}_j$$

Moreover, we observe that the constraint $\langle v_i, v_i \rangle = 1$ is always satisfied

$$\langle v_i, v_i \rangle = \sum_{k \in [n]} v_k(i)^2 = \overline{x}_i{}^2 = 2$$

This concludes that

$$\mathrm{SDP}^*_{\mathrm{MC}} \geq \mathrm{QP}^*$$

However, the algorithm proposed by the two authors is *very* surprising.

Algorithm 2.5: GW approximation for Max-Cut

Given a graph G, the algorithm returns a cut of G.

1: **function** GWAPPROX(G)
2: $\{v_1, \dots, v_n\} := \text{SDP}_{\text{MC}}(G)$
3: $S_n := \{x \in \mathbb{R}^n \mid ||x|| = 1\}$ ▷ S_n is the n-dimensional hypersphere of radius 1
4: Sample a vector y from S_n UAR
5: $S := \{i \mid \langle v_i, y \rangle \geq 0\}$
6: **return** S
7: **end function**

Theorem 2.9

Given a graph G, let S^* be an optimal solution to MC of G; then, if $S = \text{GWAPPROX}(G)$, it holds that

$$\mathbb{E}\left[|\text{cut}(S)|\right] \geq \alpha_{\text{GW}} \cdot |\text{cut}(S^*)|$$

where

$$\alpha_{\text{GW}} = \frac{2}{\pi} \min_{x \in (-1,1)} \frac{\arccos(x)}{\pi} = 0.878\dots$$

Proof. Omitted. □

We observe that the approximation ratio α_{GW} is actually *tight*. In fact, it can be proven that $\text{IG}_{\text{MC}}^{\text{SDP}} = \alpha_{\text{GW}}$, where the upper bound is given by the graph C_n — details omitted.

Theorem 2.10

$\text{IG}_{\text{MC}}^{\text{SDP}} = \alpha_{\text{GW}}$

This concludes that Goemans and Williamson's algorithm is *the best* rounding algorithm that can be achieved through SDPs. Moreover, researchers believe that this approximation is the best that can be achieved *in general* for MC. In particular, the main reasong behind such hypothesis is the **Unique Games Conjecture**, which will be discussed in the following section.

2.5 The Unique Games Conjecture

Before discussing the conjecture itself, consider the following definition.

Definition 2.6: Unique label cover

Let G be a bipartite graph, bipartitioned through (A, B); given a value $k \in \mathbb{N}$, for each edge $e \in E(G)$ let $\pi_e : [k] \to [k]$ be a permutation. A **unique label cover** (ULC) of G is an assignment $\phi : A \cup B \to [k]$ defining the set S_ϕ of satisfied edges:

$$S_\phi := \{ab \in E(G) \mid a \in A, b \in B, \pi_{ab}(\phi(a)) = \phi(b)\}$$

In other words, an edge $ab \in E(G)$ of G is sait to be *satisfied* by ϕ if the color $\phi(a)$ gets permuted into the color $\phi(b)$ by the permutation π_{ab}.

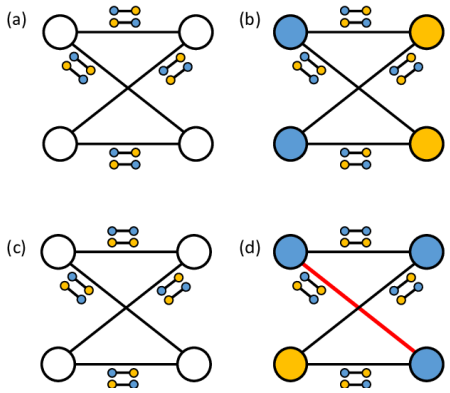


Figure 2.17: In this figure, (a) and (b) are two instances of the UCL problem with 2 colors; (b) is a solution to (a) that satisfies all the edges, while (d) is a solution to (c) with an unsatisfied edge.

Note that UCL instances are *strongly constrainted*, i.e. the color of a vertex uniquely defines the colors of its neighbors, and therefore its entire connected component. Thus, if the input instance admits a valid assignment, such assignment can be found *efficiently* by iterating over all the possible colors of a single node. In particular, this implies that the problem of deciding if a given instance admits a satisfying assignment can be solved in polynomial time.

The **value** of a UCL instance is the ratio of the edges that are satisfiable by any assignment. Hence, for satisfiable instances the ratio is 1, and we can find a satisfiying assignment as described. In contrast, determining the value of an unsatisfiable game — even approximately — appears to be very hard. This difficulty was formalized by Khot [Kho02] in terms of NP-hardness, as follows.

Conjecture 2.1: Unique Games Conjecture (UGC)

It is conjectured that for each $\varepsilon > 0$ there is a value k_ε for which it is $\mathsf{NP} - hard$ to determine if for a UCL instance with k_ε labels one of the following holds:

- at most an ε-fraction of the edges are satisfied
- at least a $(1 - \varepsilon)$-fraction of the edges are satisfied

Furthermore, Khot's conjecture has been shown to be linked with Constraint Satisfaction Problems, a particular type of optimization problems defined as follows.

Definition 2.7: Constraint Satisfaction Problem

Let $\mathcal{P}$ be a set of k-ary predicates defined on $[q]$, for $q, k \in \mathbb{N}$. An instance of a **Constraint Satisfaction Problem** (CSP) is a set of variables $X = \{x_1, \dots, x_n\}$ and a set of constraints $C_1, \dots, C_m$ such that $C_j = \langle I_j, P_j \rangle$ where $I_j \subseteq [n]$ and $P_j \in \mathcal{P}$.

A constraint $C_j = \langle I_j, P_j \rangle$ is said to be *satisfied* by an assignment $\alpha : I_j \to \{0, 1\}$ if $P_j \begin{bmatrix} x_{i_1} & \dots & x_{i_k} \\ \alpha(x_{i_1}) & \dots & \alpha(x_{i_k}) \end{bmatrix}$ evaluates to true.

The goal of a CSP is to find an assignment maximizing the number of satisfied constraints. Many optimization problems can be described in terms of CSP, and in particular the Max-Cut problem — which we mentioned at the end of the previous section. In fact, MC can be described through a CSP with value $q = 2$, and predicate set $\mathcal{P} = \{p(a, b)\}$ with arity 2, where $p(a, b) =$ "$a \neq b$". Then, given a graph G, for each node $x \in V(G)$ we define the variable x_v, and for each edge $uv \in E(G)$ we define the constraint $p(x_u, x_v)$.

Given a problem that can be described through a CSP $\mathcal{C}$, let $\alpha(\mathcal{C})$ be the best possible approximation ratio for $\mathcal{C}$, i.e. the minimum ratio such that the problem does not become NP-hard to approximate. The following theorem, proved by Raghavendra [Rag08], shows that the relationship between approximation algorithms, SDPs, and the UGC is as strong as it can be.

Theorem 2.11: Raghavendra's theorem

For every CSP $\mathcal{C}$ with values in $[q]$ and arity k, the following hold.

- There is an SDP with IG of $\alpha(\mathcal{C})$ and a rounding algorithm that is an $\alpha(\mathcal{C})$-approximation of $\mathcal{C}$.
- If the UGC is true, it is NP-hard to approximate $\mathcal{C}$ with a ratio $\alpha(\mathcal{C}) - \varepsilon$ for any $\varepsilon > 0$.

In particular, if the UGC holds true, Raghavendra's theorem implies that the SDP integrality gap for any problem that can be described through a CSP is the best possible approximation ratio that can be achieved. Although many researchers believe in the validity of the conjecture, it remains unproven.

As a direct corollary of Raghavendra's theorem, we get that the approximation ratios that we previously found for MC and VC are the best possible ones, if the UGC is true.

Corollary 2.1

If the UGC is true, then $\mathrm{IG}_{\mathrm{MC}}^{\mathrm{SDP}} = \alpha_{\mathrm{MC}}$ and $\mathrm{IG}_{\mathrm{VC}} = \alpha_{\mathrm{VC}}$.

3

Metric geometry

TODO

missing introduction

First, consider the following definition.

Definition 3.1: Set spartity

Given a graph $G = (V, E)$, and a set $\varnothing \subset S \subseteq V$, the **sparsity** of S is defined as follows:

$$\psi(S) := \frac{|\text{cut}(S)|}{\left|S \times \overline{S}\right|} = \frac{|\text{cut}(S)|}{|S| \cdot \left|\overline{S}\right|}$$

As for the case of *graph density*, we require $S \neq \varnothing$, but we will omit this detail in the following sections. Moreover, note that by definition of *cut*, we have that for any $S \subseteq V(G)$ it holds that

$$0 \leq |\text{cut}(S)| \leq \left|S \times \overline{S}\right| = |S| \cdot \left|\overline{S}\right|$$

since an edge e is in $\text{cut}(S)$ if and only if $|S \cap e| = 1$, hence the cartesian product $S \times \overline{S}$ represents the edges that have exactly one endpoint in S. Therefore, we have that $0 \leq \psi(S) \leq 1$, and in particular

- $\psi(S) = 0 \implies |\text{cut}(S)| = 0$ which happens if S is *disconnected* from $\overline{S}$
- $\psi(S) = 1 \implies |\text{cut}(S)| = \left|S \times \overline{S}\right|$ which happens if S is "*fully connected*" to $\overline{S}$, i.e. every edge of S is connected to every edge of $\overline{S}$.

Given this metric, we are interested in finding the subset of vertices of a given graph that *minimizes* its sparsity.

Definition 3.2: Sparsest Cut problem

The **Sparsest Cut** (SC) problem is defined as follows: given a graph $G = (V, E)$, determine the subset of vertices that minimizes its sparsity. In other words, the problems asks to finde a set S^* in

$$S^* \in \arg\min_{S \subseteq V} \psi(S)$$

In order to reason about SC, we are going to introduce the following type of graphs, invented by Erdős and Rényi [ER22] in 1959.

Definition 3.3: Erdős-Rényi random graph

Given two values $n \in \mathbb{N}$ and $\mu \in [0, 1]$, the Erdős-Rényi random graph $G(n, \mu)$ is a graph that has a fixed vertex set $V(G) = [n]$, and a *probabilistic* edge set $E(G)$ such that

$$\forall e \in \binom{V(G)}{2} \quad \Pr[e \in E(G)] = \mu$$

By definition, this model represents an *evenly sparse graph*, because for each $S \subseteq V(G)$ it holds that

$$\mathbb{E}[\psi(S)] = \frac{\mathbb{E}\left[|\mathrm{cut}(S)|\right]}{|S| \cdot \left|\overline{S}\right|} = \frac{\sum_{ij \in S \times \overline{S}} \Pr\left[ij \in \mathrm{cut}(S)\right]}{|S| \cdot \left|\overline{S}\right|} = \frac{\mu \cdot |S| \cdot \left|\overline{S}\right|}{|S| \cdot \left|\overline{S}\right|} = \mu$$

The Maximum Cut problem — which we discussed in Section 1.1.1 — can be reduced to SC, implying that the latter is NP-hard as well. Nevertheless, the problem can be approximated through the **Leighton-Rao algorithm** [LR88], which relies on LP relaxation and a series of reduction steps. In particular, its integrality gap is bounded using *geometric arguments*, therefore in the following sections we will focus on **metric geometry** in order to establish this result.

3.1 Metrics

3.1.1 Cut metrics

A **metric** (or *distance function*) provides a way to quantify how far apart objects are within a given space, formalizing the concept of *distance*. In particular, in geometry we have the following definition.

Definition 3.4: Metric

Given a set S, a **metric** on S is a function $d : S \times S \to \mathbb{R}$ that satisfies

1. *non-negativity*: $\forall x, y \in S \quad d(x, y) \geq 0$
2. *symmetry*: $\forall x, y \in S \quad d(x, y) = d(y, x)$
3. *self-distance*: $\forall x \in S \quad d(x, x) = 0$
4. *triangle inequality*: $\forall x, y, z \in S \quad d(x, y) \leq d(x, z) + d(z, y)$

The typical example of metric is the Euclidean distance, which defines d as follows

$$d(x, y) := ||x - y||$$

In particular, for our discussion we are interested in *cut metrics*, defined as follows.

Definition 3.5: Elementary cut metric

Given a graph $G = (V, E)$, and a set $T \subseteq V$, the **elementary cut metric of** T is a function $d_T : V \times V \to \mathbb{R}$ such that

$$d_T(x, y) := \begin{cases} 1 & |T \cap \{x, y\}| = 1 \\ 0 & \text{otherwise} \end{cases}$$

In other words, for a given set of vertices T, the elementary cut metric $d_T(x, y)$ is equal to 1 if and only if the edge xy is in $\text{cut}(T)$.

Proposition 3.1

Any elementary cut metric is a metric.

Proof. Given a graph G, and a susbset $T \subseteq V(G)$, consider the elementary cut metric d_T; we observe that, by definition

- *non-negativity* is satisfied, because

$$\forall x, y \in V(G) \quad d_T(x, y) = 0 \vee d_T(x, y) = 1 \implies d_T(x, y) \geq 0$$

- *symmetry* is satisfied, because

$$\forall x, y \in V(G) \quad |T \cap \{x, y\}| = |T \cap \{y, x\}| \implies d_T(x, y) = d_T(y, x)$$

- *self-distance* is satisfied, because

$$\forall x \in V(G) \quad \nexists \{x, x\} \in E(G) \implies d_T(x, x) = 0$$

hence, we just need to prove that the *triangle inequality* is also satisfied by d_T. Therefore, fix three vertices $x, y, z \in V(G)$; we have three cases:

- if $x, y, z \in T$ then

$$0 = d_T(x, y) \leq d_T(x, z) + d_T(z, y) = 0 + 0 = 0$$

- if $x, y, z \notin T$ then

$$0 = d_T(x, y) \leq d_T(x, z) + d_T(z, y) = 0 + 0 = 0$$

- if $\exists A \in \{T, \overline{T}\}$ such that *exactly one* of x, y, z lies in A, we have three sub-cases:
  - if $x \in A$ then

$$1 = d_T(x, y) \leq d_T(x, z) + d_T(z, y) = 1 + 0 = 1$$

  - if $y \in A$ then

$$1 = d_T(x, y) \leq d_T(x, z) + d_T(z, y) = 0 + 1 = 1$$

  - if $z \in A$ then

$$0 = d_T(x, y) \leq d_T(x, z) + d_T(z, y) = 1 + 1 = 2$$

□

We observe that, by definition, for any T it holds that $d_T(x, y) = d_{\overline{T}}(x, y)$. In fact, if $\{x, y\} \in \text{cut}(T)$ then $|T \cap \{x, y\}| = 1$, meaning that exactly one endpoint of $\{x, y\}$ lies in T, which must imply that the other endpont lies in $\overline{T}$.

But why are we discussing cut metrics in the first place? We are going to show that the concept of elementary cut metric is *deeply* related to the concept of *sparsity* previously introduced. First, consider the following definition.

Definition 3.6: Cut-ratio

Given a graph G, and a subset $T \subseteq V(G)$, the **cut-ratio** induced by the elementary cut metric d_T is defined as

$$\phi(d_T) := \frac{\sum_{xy \in E(G)} d_T(x, y)}{\sum_{xy \in \binom{V(G)}{2}} d_T(x, y)}$$

Unsurprisingly, the following proposition shows that the cut-ratio of d_T is precisely the sparsity of T.

Proposition 3.2

Given a graph G, and a subset $T \subseteq V(G)$, it holds that $\phi(d_T) = \psi(T)$.

Proof. We observe that

$$\phi(d_T) := \frac{\sum_{ij \in E(G)} d_T(x,y)}{\sum_{ij \in \binom{V(G)}{2}} d_T(x,y)} = \frac{\sum_{xy \in E(G)} \mathbb{1}[x \in T \oplus y \in T]}{\sum_{xy \in \binom{V(G)}{2}} \mathbb{1}[x \in T \oplus y \in T]} = \frac{\text{cut}(S)}{|S| \cdot \left|\overline{S}\right|} = \psi(T)$$

□

We can generalize the concept of elementary cut metric through **cut metrics**, i.e. linear combinations over a given set of elementary cut metrics, as defined below.

Definition 3.7: Cut metric

Given a graph G, a sequence $d_{T_1}, \ldots, d_{T_k}$ of elementary cut metrics on G, and k positive real values $\lambda_1, \ldots, \lambda_k > 0$, a **cut metric** is a function defined as follows

$$d : V(G) \times V(G) \to \mathbb{R} : (x,y) \mapsto \sum_{i \in [k]} \lambda_i d_{T_i}(x,y)$$

As for elementary cut metrics, it can be easily proven that cut metrics are indeed metrics — we will omit the proof. Moreover, the following proposition shows that the cut-ratio of a cut metric d is lower bounded by the smallest cut-ratio among the elementary cut metrics that define d itself.

Proposition 3.3

Given a graph G, and a cut metric d defined through $d_{T_1}, \ldots, d_{T_k}$, it holds that

$$\phi(d) \geq \min_{j \in [k]} \phi(d_{T_j})$$

Proof. First, consider the following property of sums.

Claim: For any $a_1, \ldots, a_k, b_1, \ldots, b_k > 0$, it holds that

$$\frac{\sum_{i \in [k]} a_i}{\sum_{i \in [k]} b_i} \geq \min_{j \in [k]} \frac{a_j}{b_j}$$

Proof of the Claim. We observe that

$$\frac{\sum_{i \in [k]} a_i}{\sum_{i \in [k]} b_i} = \frac{\sum_{i \in [k]} \frac{b_i}{b_i} \cdot a_i}{\sum_{i \in [k]} b_i} \geq \frac{\sum_{i \in [k]} b_i \cdot \min_{j \in [k]} \frac{a_j}{b_j}}{\sum_{i \in [k]} b_i} = \min_{j \in [k]} \frac{a_j}{b_j}$$

□

Let $\lambda_1, \dots, \lambda_k > 0$ be the coefficients that define d; then, thanks to the claim, we get

$$\begin{aligned}
\phi(d) &= \frac{\sum_{xy \in E(G)} d(x,y)}{\sum_{xy \in \binom{V(G)}{2}} d(x,y)} \\
&= \frac{\sum_{xy \in E(G)} \sum_{i \in [k]} \lambda_i d_{T_i}(x,y)}{\sum_{xy \in \binom{V(G)}{2}} \sum_{i \in [k]} \lambda_i d_{T_i}(x,y)} \\
&= \frac{\sum_{i \in [k]} \sum_{xy \in E(G)} \lambda_i d_{T_i}(x,y)}{\sum_{i \in [k]} \sum_{xy \in \binom{V(G)}{2}} \lambda_i d_{T_i}(x,y)} \\
&\geq \min_{j \in [k]} \frac{\sum_{xy \in E(G)} \lambda_j d_{T_j}(x,y)}{\sum_{xy \in \binom{V(G)}{2}} \lambda_j d_{T_j}(x,y)} \\
&= \min_{j \in [k]} \frac{\sum_{xy \in E(G)} d_{T_j}(x,y)}{\sum_{xy \in \binom{V(G)}{2}} d_{T_j}(x,y)} \\
&= \min_{j \in [k]} \phi(d_{T_j})
\end{aligned}$$

□

An important consequence of Proposition 3.2 and Proposition 3.3 is the following corollary, which implies that optimizing over **the sparsest cut** is equivalent to both optimizing over all **elementary cut metrics** and **cut metrics** in general.

Corollary 3.1

Given a graph G, it holds that

$$\min_{T \subseteq V(G)} \psi(T) = \min_{T \subseteq V(G)} \phi(d_T) = \min_{d \text{ cut metric}} \phi(d)$$

We observe that the last equality derives from the fact that elementary cut metrics are *trivial* cut metrics.

3.1.2 The ℓ_1 metric

Up until this point, we have explored cut metrics in great detail. In the following section, we are going to shift our focus towards other types of metrics, in patricular the ℓ_μ type of metrics, which are defined as shown below.

Definition 3.8: ℓ_μ metrics

Given $\mu \in \mathbb{R}_{\geq 1}$, and a set $S \subseteq \mathbb{R}^d$ for some d, the ℓ_μ **metric** is a function defined as follows

$$\ell_\mu(x, y) = \sqrt[\mu]{\sum_{i=1}^{d} |x_i - y_i|^\mu}$$

We will denote $\ell_\mu(x, y)$ also as $|x - y|_{\ell_\mu}$.

In particular, we are interested in the ℓ_1 metric. In fact, consider the following definition.

Definition 3.9: Isometrical embedding

Consider two metrics $d_1 : A \times A \to B$ and $d_2 : X \times X \to Y$; we say that d_1 is **isometrically embedded** into d_2 if there is a function $f : A \to X$ such that

$$d_1(x, y) = d_2(f(x), f(y))$$

Lemma 3.1

Any cut metric defined over k cuts is isometrically embedded into ℓ_1 over $\mathbb{R}^k$.

Proof. Let d be a cut metric defined over $T_1, \dots, T_k$, i.e. $d(x, y) = \sum_{i \in [k]} \lambda_i d_{T_i}(x, y)$ for some $\lambda_1, \dots, \lambda_k > 0$. For each $u \in V(G)$, let $\overline{x_u} \in \mathbb{R}^k$ be a vector defined as follows:

$$\overline{x_u}(i) := \begin{cases} \lambda_i & u \in T_i \\ 0 & u \notin T_i \end{cases}$$

where $\overline{x_u}(i)$ is the i-th component of $\overline{x_u}$.

Now, consider the function

$$f : V(G) \to \mathbb{R}^k : u \mapsto \overline{x_u}$$

Now, fix $i \in [k]$, and two vertices $u, v \in V(G)$; we observe that

- $|\{u, v\} \cap T_i| = 1 \implies (u \in T_i \wedge v \notin T_i) \vee (u \notin T_i \wedge v \in T_i)$; hence, we have that

  $$u \in T_i \wedge v \notin T_i \implies \overline{x_u}(i) = \lambda_i \wedge \overline{x_v}(i) = 0 \implies |\overline{x_u}(i) - \overline{x_v}(i)| = 1$$

  and also that

  $$u \notin T_i \wedge v \in T_i \implies \overline{x_u}(i) = 0 \wedge \overline{x_v}(i) = \lambda_i \implies |\overline{x_u}(i) - \overline{x_v}(i)| = 1$$

- $|\{u, v\} \cap T_i| = 0 \implies u, v \in T_i \vee u, v \notin T_i$; hence, we have that

  $$u, v \in T_i \implies \overline{x_u}(i) = \overline{x_v}(i) = \lambda_i \implies |\overline{x_u}(i) - \overline{x_v}(i)| = 0$$

  and also that

  $$u, v \notin T_i \implies \overline{x_u}(i) = \overline{x_v}(i) = 0 \implies |\overline{x_u}(i) - \overline{x_v}(i)| = 0$$

This means that for any $i \in [k]$ and $u, v \in V(G)$, it holds that

$$\mathbb{1}\left[|\{u,v\} \cap T_i| = 1\right] = |\overline{x_u}(i) - \overline{x_v}(i)|$$

Therefore, we obtain the following

$$\begin{aligned} d(u,v) &= \sum_{i\in[k]} \lambda_i d_{T_i}(u,v) \\ &= \sum_{i\in[k]} \lambda_i \cdot \mathbb{1}\left[|\{u,v\} \cap T_i| = 1\right] \\ &= \sum_{i\in[k]} |\overline{x_u}(i) - \overline{x_v}(i)| \\ &= |\overline{x_u} - \overline{x_v}|_{\ell_1} \end{aligned}$$

meaning that d is indeed isometrically embedded into ℓ_1 over $\mathbb{R}^k$. □

Lemma 3.2

For any d, and any set $X \underset{fin}{\subseteq} \mathbb{R}^d$, the ℓ_1 metric over X is isometrically embedded into a cut metric defined over $d \cdot (|X| - 1)$ cuts.

Proof. We prove the statement for $d = 1$, and then extend the proof to all other values of d.

Let $X \underset{fin}{\subseteq} \mathbb{R}^d$, i.e. $X = \{x_1, \ldots, x_n\}$ for some $n \in \mathbb{N}$; moreover, let $\pi : [n] \to [n]$ be a permutation of the indices ordering the n elements of X such that they are ordered in *ascending order*

$$x_{\pi(1)} \leq \ldots \leq x_{\pi(n)}$$

For each $j \in [n-1]$, let $S_j := \{\pi(1), \ldots, \pi(j)\}$; by construction, we have that

$$\forall j \in [n-1] \quad S_1 \subseteq \ldots \subseteq S_j$$

Moreover, for each $j \in [n-1]$ set $\lambda_j := x_{\pi(j+1)} - x_{\pi(j)}$; since the indices order the elements of X in ascending order, we know that $\lambda_j \geq 0$ for all $j \in [n-1]$.

Define d to be the cut metric over the cuts $S_1, \ldots, S_{n-1}$ with coefficients $\lambda_1, \ldots, \lambda_{n-1}$; hence, for each i and j such that $i < j$ we get that

$$\begin{aligned} d(x_{\pi(i)}, x_{\pi(j)}) &= \sum_{t\in[j]} \lambda_t d_{S_t}(x_{\pi(i)}, x_{\pi(j)}) \\ &= \sum_{t\in[j]} \lambda_t \cdot \mathbb{1}\left[\left|\{x_{\pi(i)}, x_{\pi(j)}\} \cap S_t\right| = 1\right] \\ &= \sum_{i\leq t<j} \lambda_t \\ &= (x_{\pi(j)} - x_{\pi(j-1)}) + (x_{\pi(j-1)} - x_{\pi(j-2)}) + \ldots + (x_{\pi(i+1)} - x_{\pi(i)}) \\ &= x_{\pi(j)} - x_{\pi(i)} \\ &= \left|x_{\pi(j)} - x_{\pi(i)}\right|_{\ell_1} \end{aligned}$$

Lastly, since metrics are symmetric, we conclude that

$$\forall i, j \quad d(x_{\pi(i)}, x_{\pi(j)}) = \ell_1(x_{\pi(1)}, x_{\pi(j)})$$

which means that for $d = 1$ finite ℓ_1 metrics can be embedded into a cut metric with $|X| - 1$ cuts. Now, for any dimension $d > 0$, finite ℓ_1 metrics can be embedded into a cut metric that is the sum of each 1-dimensional embedding, meaning that ℓ_1 can be embedded into a cut metric with $d \cdot (|X| - 1)$ cuts. □

These two lemmas further improve our optimization equalities: optimizing over a cut metric is equal to optimizing over an ℓ_1 **metric**.

Corollary 3.2

Given a graph G, it holds that

$$\min_{T \subseteq V(G)} \psi(T) = \min_{T \subseteq V(G)} \phi(d_T) = \min_{d \text{ cut metric}} \phi(d) = \min_{d \ \ell_1 \text{ metric}} \phi(d)$$

3.2 Metric relaxations and distortions

3.2.1 Metric relaxations

Consider the following program:

$$\max \frac{\sum_{ij \in E(G)} d_{ij}}{\sum_{ij \in \binom{V(G)}{2}} d_{ij}}$$

$$\sum_{S \subseteq V(G)} \lambda_S d_S(i,j) = d_{ij} \quad \forall ij \in \binom{V(G)}{2}$$

$$\lambda \in \mathbb{R}^{2^n}$$

$$d \in \mathbb{R}^{\left|\binom{n}{2}\right|}$$

Figure 3.1: Non-linear program for SC.

In this *non*-linear program, the variables d_{ij} and λ_S — as their names suggest — represent the cut values for the optimal cut metric d^* described by the optimal solution, which corresponds to $\phi(d^*)$. Note that the values $d_S(i,j)$ are elementary cut metrics, hence they act as nothing more than a constant coefficient, while we optimize over the values λ_S defining the *actual* coefficients of the cut.

Additionally, the main constraint of the program optimizes over cut metrics, which we proved to be "isometrically equivalent" to ℓ_1 metrics in the previous lemmas. Therefore, this program can be viewed as optimizing both over cut metrics and ℓ_1 metrics. However, the objective function is clearly *non*-linear, hence the program is *non*-linear as well. To

fix this, we can normalize the denominator of the objective function since it is shared among all feasible solutions, obtaining the following LP

$$\begin{aligned} \max \quad & \sum_{ij \in E(G)} d_{ij} \\ & \sum_{S \subseteq V(G)} \lambda_S d_S(i,j) = d_{ij} \quad \forall ij \in \binom{V(G)}{2} \\ & \sum_{ij \in \binom{V(G)}{2}} d_i j = 1 \\ & \lambda \in \mathbb{R}^{2^n} \\ & d \in \mathbb{R}^{\left|\binom{n}{2}\right|} \end{aligned}$$

Figure 3.2: LP for SC.

By construction, the solution to this LP corresponds to the optimal cut metric d^*, and the optimal solution to SC is given by the minimal elementary cut describing d^*; therefore, this LP could be used to get an *exact* solution to SC. However, we cannot employ this LP because it requires an *exponential* number of variables. To solve this issue, consider the following definition.

Definition 3.10: Metric distortion

Given two metrics d_1 and d_2 over the same vector space V, we say that d_2 has a **distortion** from d_1 of at most $\alpha\beta$ if

$$\forall x, y \in V \quad \frac{d_1(x,y)}{\alpha} \leq d_2(x,y) \leq \beta d_1(x,y)$$

Metric distortion describes the concept of *similarity* between measures. In 1985 Bourgain [Bou85] proved that any finite metric can be isometrically embedded into an ℓ_1 metric, up to some distortion factor.

Theorem 3.1: Bourgain's theorem (computational version)

Any metric d on n points can be isometrically embedded in time $n^O(1)$ into an ℓ_1 metric on $\mathbb{R}^d$, where $d = O(\log^3 n)$ and distortion factor $O(\log n)$.

Thanks to this result, Leighton and Rao [LR88] were able to provide an algorithm that yields an $O(\log n)$-approximation to SC by using the following **metric LP relaxation**: instead of optimizing over ℓ_1 metrics, they constructed a relaxation that optimizes over *all* metrics — introducing the approximation factor of $O(\log n)$, which is precisely the distortion factor.

$$\max \sum_{ij \in E(G)} d_{ij}$$

$$\begin{aligned} x_{ij} + x_{jk} &\geq x_{ik} \quad \forall i, j, k \in V(G) \\ \sum_{ij \in \binom{V(G)}{2}} d_{ij} &= 1 \\ \lambda &\in \mathbb{R}^{2^n} \\ d &\in \mathbb{R}^{|\binom{V(G)}{2}|} \end{aligned}$$

Figure 3.3: Metric LP relaxation for SC.

Algorithm 3.1: Leighton-Rao algorithm

Given a graph G, the algorithm returns a cut of G.

1: **function** LeightonRao(G)
2: $\quad d := \mathrm{LP}_{\mathrm{metric}}(G)$
3: $\quad d' :=$ to-ℓ_1-metric(d) ▷ apply Theorem 3.1
4: $\quad d'' :=$ to-cut-metric(d') ▷ apply Lemma 3.2
5: $\quad$ Let $T_1, \ldots, T_k$ be the k cuts of d''
6: $\quad$ **return** $S \in \arg\min_{i \in [k]} \phi(T_i)$
7: **end function**

Thanks to Corollary 3.2 and Theorem 3.1, we obtain the following result.

Theorem 3.2

Given a graph G, and an optimal solution S^* to SC on G, let $S :=$ LeightonRao(G). Then, it holds that

$$\psi(S) \leq O(\log n) \cdot \psi(S^*)$$

In recent years Arora, Rao, and Vazirani [ARV09] were able to improve the approximation ratio up to $O(\sqrt{\log n})$ through the following SDP.

$$\max \sum_{ij \in E(G)} \langle x_i, x_i \rangle - 2 \langle x_i, x_j \rangle + \langle x_j, x_j \rangle$$

$$\begin{gathered} \langle x_i, x_j \rangle - \langle x_j, x_j \rangle + \langle x_j, x_k \rangle \leq \langle x_i, x_k \rangle \qquad \forall i, j, k \in V(G) \\ \sum_{ij \in \binom{V(G)}{2}} \langle x_i, x_i \rangle - 2 \langle x_i, x_j \rangle + \langle x_j, x_j \rangle = 1 \\ \lambda \in \mathbb{R}^{2^n} \\ d \in \mathbb{R}^{\left|\binom{V(G)}{2}\right|} \end{gathered}$$

Figure 3.4: ℓ_2^2 metric SDP relaxation for SC.

The idea is similar to that of the Leighton-Rao algorithm: instead of relaxing to every metric, they relax the problem to **squared ℓ_2 metrics**. In particular, they observed that

$$\begin{aligned} \ell_2^2(x, y) &= \sum_{i \in [n]} |x_i - y_i|^2 \\ &= \sum_{i \in [n]} \left(x_i^2 - 2 x_i y_i + y_i^2 \right) \\ &= \sum_{i \in [n]} x_i^2 - 2 \sum_{i \in [n]} x_i y_i + \sum_{i \in [n]} y_i^2 \\ &= \langle x, x \rangle - 2 \langle x, y \rangle + \langle y, y \rangle \end{aligned}$$

Note that ℓ_2^2 does not always respect the *triangle inequality*. For instance, in $\mathbb{R}$ we have that $\ell_2^2(-1, 0) = \ell_2(0, 1) = 1$, but $\ell_2^2(-1, 1) = 4 > 1$. In fact, the triangle inequality is forced by the second constraint of the SDP itself, in fact:

$$\ell_2^2(x, z) + \ell_2^2(z, y) \leq \ell_2^2(x, y) \implies$$

$$(\langle x_i, x_i \rangle - 2 \langle x_i, x_j \rangle + \langle x_j, x_j \rangle) + (\langle x_j, x_j \rangle - 2 \langle x_j, x_k \rangle + \langle x_k, x_k \rangle) \geq \langle x_i, x_i \rangle - 2 \langle x_i, x_k \rangle + \langle x_k, x_k \rangle$$

$$\implies -2 \langle x_i, x_j \rangle + 2 \langle x_j, x_j \rangle - 2 \langle x_j, x_k \rangle \geq -2 \langle x_i, x_k \rangle$$

$$\langle x_i, x_j \rangle - \langle x_j, x_j \rangle + \langle x_j, x_k \rangle \leq \langle x_i, x_k \rangle$$

Proposition 3.4

Any ℓ_2^2 metric on n points can be isometrically embedded in time $n^{O}(1)$ into an ℓ_1 metric on $\mathbb{R}^d$ with distortion factor $O(\sqrt{\log n})$.

Proof. Omitted. □

This $O(\sqrt{\log n})$ is the current best known approximation for SC. Moreover, SC cannot be expressed in terms of CSP — recall that the above SDP is a *relaxation* for the minimal ℓ_2^2 problem. Furthermore, even if this approximation ratio is believed to be the best possible one, the UGC only implies that there are no constant factor approximations for SC, meaning that the non-constant ratio could be proved to be lower.

3.2.2 Shortest path metric

Metric distortion has showed to be a useful tool for various approximation algorithms. However, in some instances there is a *lower bound* on the distortion required in order to embed a metric into another. For instance, consider the following metric.

Definition 3.11: Shortest path metric

Given a graph G, the **shortest path metric** is a function $d : V(G) \times V(G) \to \mathbb{R}$ defined as follows

$$d_G(x, y) = |\{e \in E(P) \mid P \text{ shortest } u \to v \text{ path}\}|$$

It can be easily proven that this is indeed a metric — we will omit the proof.

We are going to show that the shortest path metric cannot be embedded into ℓ_2 over $\mathbb{R}^d$ without distortion. In particular, we will show the case for the cycle graph C_4. Consider the embedding $f : V(C_4) \to \mathbb{R}^2$ such that

$$f(1) = \left(-\frac{1}{2}; \frac{1}{2}\right) \quad f(2) = \left(\frac{1}{2}; \frac{1}{2}\right) \quad f(3) = \left(\frac{1}{2}; -\frac{1}{2}\right) \quad f(4) = \left(-\frac{1}{2}; -\frac{1}{2}\right)$$

It is easy to see that this embedding is isometrical from d_{C_4} to ℓ_1, since all distances are preserved. However, for the ℓ_2 metric the diagonals are *not* isometric, in fact

$$2 = d_{C_4}(1, 3) \neq \ell_2(f(1), f(3)) = \ell_2\left(\left(-\frac{1}{2}; \frac{1}{2}\right), \left(\frac{1}{2}; -\frac{1}{2}\right)\right) = \sqrt{2}$$

In particular, this embedding induces a distortion factor of at most $\sqrt{2}$. Furthermore, thanks to the following lemma we prove that this distortion factor is actually the *best* that we can achieve.

Lemma 3.3: Shortest Diagonal lemma

For any $y_1, y_2, y_3, y_4 \in \mathbb{R}^4$ it holds that

$$\ell_2^2(y_1, y_3) + \ell_2^2(y_2, y_4) \leq \ell_2^2(y_1, y_2) + \ell_2^2(y_2, y_3) + \ell_2^2(y_3, y_4) + \ell_2^2(y_4, y_1)$$

Proof. We will employ a technique called *sum of squares proof*: the idea is to show that one equation made of sums of squares is equivalent to the square of an other equation, implying that the first one must be always non-negative. We observe that finding the second equation to use as "comparison" is *very* hard.

We will prove the lemma coordinate-by-coordinate: fix $t \in [d]$, and for each $i \in [4]$ let $z_i := y_i(t)$ be the t-th coordinate of y_i.

Claim: For all $t \in [d]$ it holds that

$$(z_1 - z_4)^2 + (z_2 - z_3)^2 + (z_3 - z_4)^2 + (z_4 - z_1)^2 - (z_1 - z_3)^2 - (z_2 - z_4)^2 = (z_1 - z_2 + z_3 - z_4)^2$$

Proof of the Claim. It can be proven by expanding both the left and right hand side of the quation. □

Because of the claim, we get that

$$(z_1-z_4)^2+(z_2-z_3)^2+(z_3-z_4)^2+(z_4-z_1)^2-(z_1-z_3)^2-(z_2-z_4)^2 = (z_1-z_2+z_3-z_4)^2 \geq 0$$

since the LHS is a square. Now, recalling that $\ell_2^2(y_i, y_j) := \sum_{t\in[d]} (y_t(i) - y_t(j))^2$, we get that

$$\ell_2^2(y_1, y_2) + \ell_2^2(y_2, y_3) + \ell_2^2(y_3, y_4) + \ell_2^2(y_4, y_1) - \ell_2^2(y_1, y_3) - \ell_2^2(y_2, y_4) \geq 0$$

□

Proposition 3.5

Any embedding of the shortest path metric d_{C_4} into ℓ_2 over $\mathbb{R}^d$, for any $d \in \mathbb{N}_{\geq 1}$, has a distortion factor of at least $\sqrt{2}$.

Proof. Fix $d \in \mathbb{N}_{\geq 1}$, and consider an embedding $f : V(C_4) \to \mathbb{R}^d$. Without loss of generality, we may assume that $f(i) = x_i$ for each $i \in [4]$. Let M be the maximum ℓ_2 edge length over all the points embedded into $\mathbb{R}^d$, in other words

$$M_{\ell_2} := \max(\ell_2(x_1, x_2), \ell_2(x_2, x_3), \ell_2(x_3, x_4), \ell_2(x_4, x_1))$$

and let m be the minimum ℓ_2 diagonal length over all the points embedded into $\mathbb{R}^d$, in other words

$$m_{\ell_2} := \min(\ell_2(x_1, x_3), \ell_2(x_2, x_4))$$

Thanks to the Shortest Diagonal lemma, we have that

$$\begin{aligned} 2m_{\ell_2}^2 &\leq \ell_2^2(y_1, y_3) + \ell_2^2(y_2, y_4) \\ &\leq \ell_2^2(y_1, y_2) + \ell_2^2(y_2, y_3) + \ell_2^2(y_3, y_4) + \ell_2^2(y_4, y_1) \\ &\leq 4M_{\ell_2}^2 \end{aligned}$$

implying that $2m_{\ell_2}^2 \leq 4M_{\ell_2}^2 \iff m_{\ell_2} \leq \sqrt{2}M_{\ell_2}$. Now, note that the maximum edge length M_{C_4} w.r.t. d_{C_4} is 1, and the minimum diagonal length m_{C_4} w.r.t. d_{C_4} is 2, which means that $m_{C_4} = 2M_{C_4}$. Therefore, the embedding must have applied a distortion factor of at least $\sqrt{2}$. □

3.3 Exercises

4

Submodular optimization

Submodular optimization plays a central role in discrete optimization, particularly in algorithmic settings where efficiency and approximation guarantees are critical. Submodular functions model a *diminishing returns* property that arises naturally in many computational problems, such as influence maximization, data summarization, and sensor placement. But before introducing submodular optimization in detail, we will discuss the following problem.

Definition 4.1: Max Cover problem

The **Max Cover** (MC) problem is defined as follows: given a *universe* (or *ground*) set $\mathcal{U} = [n]$, a collection of sets $C = \{S_1, \ldots, S_m\}$ such that $S_i \subseteq \mathcal{U}$, and an integer $k \geq 1$, determine the sub-collection $S \subseteq C$ such that $|S| = k$ that maximizes $\left|\bigcup_{S_j \in S} S_j\right|$.

In other words, we are asked to determine the sub-collection of the given C of k sets that covers as many elements of $\mathcal{U}$ as possible. For instance, given $\mathcal{U} = [7]$, $S_1 = \{1, 2\}$, $S_2 = \{2, 3\}$, $S_3 = \{3, 4, 5\}$, $S_4 = \{5, 6, 7\}$, and $k = 2$, an optimal solution would be $S = \{S_1, S_3\}$ because $|S_1 \cup S_3| = |\{1, 2, 3, 4, 5\}| = 5$.

The approximation of MC is *solved*, meaning that the best known approximation ratio of $1 - \frac{1}{e}$ has been proven to be tight under NP-hardness. Such approximation ratio can be achieved through the following greedy algorithm.

Algorithm 4.1: $\left(1-\frac{1}{e}\right)$-approximation for MC

Given an instance of MC $(\mathcal{U}, C, k)$, the algorithm returns a $\left(1-\frac{1}{e}\right)$-approximation for the instance.

1: **function** ApproxMaxCover($\mathcal{U}$, C, k)
2: $S := \varnothing$
3: $T_0 := \varnothing$
4: **for** $i \in [k]$ **do**
5: $S_i \in \arg\max_{S_j \in C - S} |S_j - T_{i-1}|$ ▷ S_i maximizes the number of "new" elements
6: $S = S \cup \{S_i\}$
7: $T_i = T_{i-1} \cup S_i$
8: **end for**
9: **return** S
10: **end function**

Theorem 4.1

Given an instance (U, C, k) of MC, and an optimal solution S^* to MC on $(\mathcal{U}, C, k)$, let $S :=$ ApproxMaxCover$(\mathcal{U}, C, k)$. Then, it holds that

$$\left|\bigcup_{S_j \in S} S_j\right| \geq \left(1-\frac{1}{e}\right)\left|\bigcup_{S_j \in S^*} S_j\right|$$

Proof. Consider an instance $(\mathcal{U}, C, k)$ of MC, and an optimal solution S^* to it; moreover, let $X^* := \bigcup_{S_j^* \in S^*} S_j^*$ be the set of elements of $\mathcal{U}$ covered by S^*.

Let $S :=$ ApproxMaxCover$(\mathcal{U}, C, k)$; hence, we have that $T_i = \bigcup_{j=1}^{i} S_j$. Additionally, let $t_i := |T_i|$ and $\mu_i := |X^*| - t_i$. Hence, we have that

- $t_0 = |T_0| = |\varnothing| = 0$
- $t_k = |T_k| = \left|\bigcup_{S_j \in S} S_j\right|$ is the value of S
- $\mu_0 = |X^*| - t_0 = |X^*| - 0 = |X^*|$ is the value of the optimal solution S^*

Therefore, to prove the approximation ratio of the statement it suffices to show that $t_k \geq \left(1-\frac{1}{e}\right)\mu_0$.

Claim 1: $\forall i \in [k] \quad |S_i - T_{i-1}| \geq \frac{\mu_{i-1}}{k}$

Proof of the Claim. Since S^* is an optimal solution that covers $|X^*|$ elements of $\mathcal{U}$,

there must be at least one set $S_j^* \in S^*$ such that

$$\left|S_j^* - T_{i-1}\right| \geq \frac{|X^*| - |T_{i-1}|}{k} = \frac{\mu_{i-1}}{k}$$

since otherwise it would be impossible for S^* to cover all of $X^* - T_{i-1}$.

Hence, the set S_i selected at the i-th iteration is at least *as large as* S_j^*, since it is the one maximizing the number of "new" elements added:

$$|S_i - T_{i-1}| \geq \left|S_j^* - T_{i-1}\right| \geq \frac{\mu_{i-1}}{k}$$

□

Claim 2: $\forall i \in [0, k] \quad \mu_i \leq \left(1 - \frac{1}{k}\right)^i |X^*|$

Proof of the Claim. We will prove the claim by induction on i. In particular, if $i = 0$, we have that

$$\mu_0 = |X^*| \left(1 - \frac{1}{k}\right)^0 |X^*| = 1 \cdot |X^*|$$

hence the base case trivially holds. Then, assuming the inductive hypothesis, we can prove the inductive step as follows

$$\begin{aligned}
\mu_{i+1} &= |X^*| - t_{i+1} \\
&= |X^*| - |T_{i+1}| \\
&= |X^*| - \left|\bigcup_{j \in [i+1]} S_i\right| \\
&= |X^*| - \left(\left|\bigcup_{j \in [i]} S_i\right| + \left|S_{i+1} - \bigcup_{j \in [i]} S_i\right|\right) \\
&= |X^*| - (|T_i| + |S_{i+1} - T_i|) \\
&= |X^*| - |T_i| - |S_{i+1} - T_i| \\
&\leq \mu_i - \frac{\mu_i}{k} && \text{(by Claim 1)} \\
&= \left(1 - \frac{1}{k}\right) \mu_i \\
&\leq \left(1 - \frac{1}{k}\right) \left(1 - \frac{1}{k}\right)^i |X^*| && \text{(by inductive hypothesis on } \mu_i\text{)} \\
&= \left(1 - \frac{1}{k}\right)^{i+1} |X^*|
\end{aligned}$$

□

By Claim 2, recalling that for any $k \geq 1$ it holds that $\left(1 - \frac{1}{k}\right)^k \leq e^{-1}$, we get

$$\mu_k \leq \left(1 - \frac{1}{k}\right)^k |X^*| \leq \frac{1}{e} |X^*|$$

Lastly, since $\mu_k = |X^*| - |T_k|$, we get that

$$|X^*| - |T_k| = \mu_k \leq \frac{1}{e}|X^*| \iff |T_k| \geq \left(1 - \frac{1}{e}\right)|X^*| \iff t_k \geq \left(1 - \frac{1}{e}\right)\mu_0$$

□

4.1 Submodular functions

We are now ready to discuss **submodular functions**, which play a central role in *submodular optimization* that we introduced at the beginning of the chapter.

Definition 4.2: Modular functions

Given $n \in \mathbb{N}$, let $f : \mathcal{P}([n]) \to \mathbb{R}$ be a function; given two sets $S, T \subseteq [n]$, we say that f is

- a **submodular function** if $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$
- a **modular function** if $f(S) + f(T) = f(S \cup T) + f(S \cap T)$
- a **supermodular function** if $f(S) + f(T) \leq f(S \cup T) + f(S \cap T)$

For instance, for any $n \in \mathbb{N}$ the *cardinality* function, i.e. $f(S) = |S|$ is modular, because $\forall S, T \subseteq [n] \quad f(S \cup T) = f(S) + f(T) - f(S \cap T)$ by the inclusion-exclusion principle. We observe that submodular and supermodular functions are modular, therefore any property that holds for modular functions is true for the other two types as well.

The main property of modular functions is the property to "know the full description of the function" through at most $n + 1$ values. For example, if f is modular, then the value of $f(\{1, 2, 3\})$ can be computed through $f(\varnothing)$, $f(\{1\})$, $f(\{2\})$ and $f(\{3\})$ as follows

$$f(\{1,2,3\}) = f(\{1,2\}) + f(\{3\}) - f(\varnothing) = (f(\{1\}) + f(\{2\}) - f(\varnothing)) + f(\{3\}) - f(\varnothing)$$

Proposition 4.1

For any $n \in \mathbb{N}$, the function $f : \mathcal{P}([n]) \to \mathbb{R}$ is modular if and only if there exist $z, w_1, \ldots, w_n$ such that for all $S \subseteq [n]$ it holds that $f(S) = z + \sum_{i \in S} w_i$.

Proof. Let f be a modular function defined on $n \in \mathbb{N}$, and fix a set $S \subseteq [n]$; in particular,

let $S = \{1, \dots, k\}$. By modularity of f, we have that

$$\begin{aligned} f(S) &= f(\{1, \dots, k\}) \\ &= f(\{2, \dots, k\}) + f(\{1\}) - f(\varnothing) \\ &= (f(\{3, \dots, k\}) + f(\{2\}) - f(\varnothing)) + f(\{1\}) - f(\varnothing) \\ &= f(\{k\}) + \sum_{i \in S - \{k\}} (f(\{i\}) - f(\varnothing)) \\ &= f(\varnothing) + \sum_{i \in S} f(\{i\}) - f(\varnothing) \\ &= z + \sum_{i \in S} w_i \end{aligned}$$

where $z := f(\varnothing)$ and $w_i := f(\{i\}) - f(\varnothing)$ for all $i \in S$.

Vice versa, suppose that there are values $z', w'_1, \dots, w'_n$ such that for all $S \subseteq [n]$ it holds that $f(S) = z' + \sum_{i \in S} w'_i$. Fix two sets $S, T \subseteq [n]$; then, we have that

$$\begin{aligned} f(S) + f(T) &= \left(z' + \sum_{i \in S} w'_i\right) + \left(z' + \sum_{i \in T} w'_i\right) \\ &= \left(z' + \sum_{i \in S \cap T} w'_i + \sum_{i \in S - T} w'_i\right) + \left(z' + \sum_{i \in S \cap T} w'_i + \sum_{i \in T - S} w'_i\right) \\ &= \left(z' + \sum_{i \in S \cap T} w'_i\right) + \left(z' + \sum_{i \in S - T} w'_i + \sum_{i \in S \cap T} w'_i + \sum_{i \in T - S} w'_i\right) \\ &= f(S \cap T) + f(S \cup T) \end{aligned}$$

hence f is modular. □

The importance of submodularity in optimization stems from its inherent **diminishing returns** property. In economics, diminishing returns describe the phenomenon where the incremental output of a production process decreases as the quantity of a single input increases, while all other inputs are held constant.

Definition 4.3: Diminishing returns

Given $n \in \mathbb{N}$, let $f : \mathcal{P}([n]) \to \mathbb{R}$; given $S \subseteq [n]$ and $x \in [n] - S$, the **return** of x on S for f is defined as

$$\Delta_f(x \mid S) = f(S \cup \{x\}) - f(S)$$

We say that f has **diminishing returns** when it holds that

$$\forall A \subseteq B \subseteq [n], x \in [n] - B \quad \Delta_f(x \mid A) \geq \Delta_f(x \mid B)$$

Theorem 4.2

For any $n \in \mathbb{N}$, the function $f : \mathcal{P}([n]) \to \mathbb{R}$ is submodular if and only if it has diminishing returns.

Proof.

Direct implication. Let $f : \mathcal{P}([n]) \to \mathbb{R}$ be a submodular function for some $n \in \mathbb{N}$, and fix $A \subseteq B \subseteq [n]$ and $x \in [n] - B$; by submodularity of f, we have that

$$f(A \cup \{x\}) + f(B) \geq f(A \cup \{x\} \cup B) + f((A \cup \{x\}) \cap B)$$

Now, since $A \subseteq B$, and $x \notin B$, we have that

$$f(A \cup \{x\}) + f(B) \geq f(B \cup \{x\}) + f(A)$$

Lastly, rearranging the terms, we get that

$$\Delta_f(x \mid A) := f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B) =: \Delta_f(x \mid B)$$

Converse implication. Proved in Problem 6.6.

□

Why did we present the Max Cover problem at the beginning of the chapter? MC is a **submodular optimization** problem, i.e. the problem can be described in terms of the following function, which we will prove is actually *submodular*.

Definition 4.4: Covering function

Given $t \in \mathbb{N}$, let $\mathcal{U} = [t]$ be the universe set, and let $C = \{A_1, \ldots, A_n\}$ be a collection of subsets $A_i \subseteq \mathcal{U}$; the **covering function** $f : \mathcal{P}([n]) \to \mathbb{R}$ is defined as follows:

$$\forall S \subseteq [n] \quad f(S) := \left| \bigcup_{j \in S} A_j \right|$$

For example, given $t = 4$ and $A_1 = \{1, 2, 4\}$, $A_2 = \{2, 3\}$, $A_3 = \{1, 4\}$ — hence $n = 3$ and $C = \{A_1, A_2, A_3\}$ — we have that

$$f(\{1, 3\}) = \left| \bigcup_{j \in \{1,3\}} A_j \right| = |A_1 \cup A_3| = |\{1, 2, 4\}| = 3$$

and so on. In other words, the set S is used to *index* the sets $A_j \in C$.

Proposition 4.2

The covering function is non-negative, monotone increasing and submodular.

Proof. Given a set $X \subseteq [n]$, note that

$$f(X) = \sum_{i \in [t]} \mathbb{1}\left[\exists j \in X \quad i \in A_j\right] = \sum_{i \in [t]} f_i(X)$$

where for each $i \in [t]$ the subfunction $f_i : \mathcal{P}([n]) \to \mathbb{R}$ is defined precisely as

$$f_i(X) := \begin{cases} 1 & \exists j \in X \quad i \in A_j \\ 0 & \text{otherwise} \end{cases}$$

Claim: For all $i \in [t]$, the subfunction f_i is non-negative, monotone increasing and submodular.

Proof of the Claim. Fix $i \in [t]$, and $S, T \subseteq [n]$; by definition, we trivially have that f_i is non-negaitve, therefore $f_i(S) + f_i(T) \geq 0$. Moreover, by definition f_i is also monotone increasing, i.e. $S \subseteq T \implies f_i(S) \leq f_i(T)$ — if $f_i(S) = 0$ then the inequality is always satisfied, and $f_i(S) = 1 \iff \exists j \in S \quad i \in A_j$ but $S \subseteq T \implies j \in T$ hence $f_i(T) = 1$.

To prove submodularity, we observe that since $S \cap T \subseteq S \cup T$ and $f_i \in \{0, 1\}$, we only have two cases:

1. $f_i(S \cup T) = f_i(S \cap T) = 1$. By definition, we have that

   $$f_i(S \cap T) = 1 \iff \exists j \in S \cap T \quad i \in A_j \implies \exists j \in S \wedge j \in T \quad i \in A_j \iff f_i(S) = f_i(T) = 1$$

   Hence, we have that

   $$f_i(S) + f_i(T) = 1 + 1 = f_i(S \cup T) + f_i(S \cap T)$$

2. $f_i(S \cup T) = 1$ but $f_i(S \cap T) = 0$. By definition

   $$f_i(S \cup T) \implies \exists j \in S \cup T \quad i \in A_j \implies \exists j \in S \vee j \in T \quad i \in A_j \iff f_i(S) + f_i(T) \geq 1$$

   hence, we have that

   $$f_i(S) + f_i(T) \geq 1 = f_i(S \cup T) + f_i(S \cap T)$$

□

Lastly, fix $S, T \subseteq [n]$; because of the claim, we know that f is trivially non-negative and monotone increasing. To prove submodularity, thanks to the claim we have that

$$\begin{aligned} f(S) + f(T) &= \sum_{i \in [t]} f_i(S) + \sum_{i \in [t]} f_i(T) \\ &= \sum_{i \in [t]} \left(f_i(S) + f_i(T)\right) \\ &\geq \sum_{i \in [t]} \left(f_i(S \cup T) + f_i(S \cap T)\right) \\ &= \sum_{i \in [t]} f_i(S \cup T) + \sum_{i \in [t]} f_i(S \cap T) \\ &= f(S \cup T) + f(S \cap T) \end{aligned}$$

□

Since MC is NP-hard, this proposition implies that the problem of maximizing non-negative, monotone increasing, and submodular functions over k elements is NP-hard as well. In fact, the $\left(1 - \frac{1}{e}\right)$-approximation for MC that we previously discussed is a specific instance of a more general algorithm, first presented by Nemhauser, Wolsey, and Fisher [NWF78], that aims at maximizing a Non-Negative and Monotone Increasing (NN-MI) submodular function.

Algorithm 4.2: NWF approximation

Given an NN-MI submodular function $f : \mathcal{P}(X) \to \mathbb{R}$, and a value $k \in \mathbb{N}$, the algorithm returns a subset $S \subseteq X$.

1: **function** NWFAPPROX(f, k)
2: $\quad S_0 := \varnothing$
3: $\quad$ **for** $i \in [k]$ **do**
4: $\quad\quad x_i \in \arg\max_{x \in X - S_i} f(S_i \cup \{x\})$ ▷ or, equivalently, maximize $\Delta_f(x_{i+1} \mid S_i)$
5: $\quad\quad S_i := S_{i-1} \cup \{x_i\}$
6: $\quad$ **end for**
7: $\quad$ **return** S_k
8: **end function**

Theorem 4.3

Given an NN-MI submodular function $f : \mathcal{P}(X) \to \mathbb{R}$, a value $k \in \mathbb{N}$, and an optimal set $S^* \in \arg\max_{S \in \binom{X}{k}} f(S)$, let $S := \text{NWFAPPROX}(f, k)$. Then, it holds that

$$f(S) \geq \left(1 - \frac{1}{e}\right) f(S^*)$$

Proof. The following proof is a generalization of the proof of Theorem 4.1, in which we proved MC's approximation ratio. Let $S^* := \{x_1^*, \ldots, x_k^*\}$.

Claim 1: For any $i \in [k-1]$ it holds that $f(S^*) \leq f(S_i) + k(f(S_{i+1}) - f(S_i))$.

Proof of the Claim. Since f is monotone increasing, we know that $f(S^*) \leq f(S_i \cup S^*)$.

Through some algebraic manipulation, we get that

$$\begin{aligned}
f(S^*) &\leq f(S_i \cup S^*) \\
&= f\left(S_i \cup \bigcup_{i=1}^{k} \{x_i^*\}\right) \\
&= f\left(S_i \cup \bigcup_{i=1}^{k} \{x_i^*\}\right) + \sum_{j=0} k-1\left(-f\left(S_i \cup \bigcup_{i=0}^{k} \{x_i^*\}\right)\right) + f\left(S_i \cup \bigcup_{i=0}^{j} \{x_i^*\}\right) \\
&= \sum_{j=0}^{k} \left(f\left(S_i \cup \bigcup_{i=0}^{j} \{x_i^*\}\right)\right) - f\left(S_i \cup \bigcup_{i=0}^{j-1} \{x_i^*\}\right) + f(S_i) \\
&= \sum_{j=0}^{k} \Delta\left(x_j^* \mid S_i \cup \bigcup_{i=0}^{j} \{x_i^*\}\right) + f(S_i)
\end{aligned}$$

By Theorem 4.2, we know that f is submodular if and only if it has diminishing returns, hence

$$\begin{aligned}
f(S^*) &\leq f(S_i) + \sum_{j=0}^{k} \Delta_f\left(x_j^* \mid S_i \cup \bigcup_{i=0}^{j} \{x_i^*\}\right) \\
&\leq f(S_i) + \sum_{j=0}^{k} \Delta_f(x_j^* \mid S_i)
\end{aligned}$$

Finally, by the greedy choice of the algorithm, we conclude that:

$$\begin{aligned}
f(S^*) &\leq f(S_i) + \sum_{j=0}^{k} \Delta_f(x_j^* \mid S_i) \\
&\leq f(S_i) + \sum_{j=0}^{k} \Delta_f(x_{i+1} \mid S_i) \\
&= f(S_i) + k \cdot (f(S_i \cup \{x_i + 1\}) - f(S_i)) \\
&= f(S_i) + k \cdot (f(S_{i+1}) - f(S_i))
\end{aligned}$$

□

Now, for each $i \in [k]$ let $\delta_i = f(S^*) - f(S_i)$.

Claim 2: For all $i \in [k]$ it holds that $\delta_i \geq (1 - \frac{1}{k})^i f(S^*)$.

Proof of the Claim. We proceed by induction on i. For the base case, $i = 0$ therefore we trivially have that

$$\left(1 - \frac{1}{k}\right)^0 f(S^*) = f(S^*) = f(S^*) - f(S_0) = \delta_0$$

Now, assume the claim is true for any $m \leq i$. By Claim 1, we know that

$$\begin{aligned}\delta_{i+1} &= f(S^*) - f(S_{i+1}) \\ &\leq k \cdot (f(S_{i+1}) - f(S_i)) \\ &= k \cdot (f(S_{i+1}) - f(S^*) + f(S^*) - f(S_i)) \\ &= k \cdot (-\delta_{i+1} + \delta_i)\end{aligned}$$

which implies that $\delta_{i+1} \leq (1 - \frac{1}{k})\delta_i$. Therefore, by induction we get that

$$\delta_{i+1} \leq \left(1 - \frac{1}{k}\right)\delta_i \leq \left(1 - \frac{1}{k}\right)^{i+1} f(S^*)$$

□

Lastly, recalling that for all $x \in \mathbb{R}_{\geq 0}$ it holds that $(1 - \frac{1}{k}) \leq e^{-1}$, by Claim 2 we conclude that

$$f(S^*) - f(S_k) = \delta_k \leq \left(1 - \frac{1}{k}\right)^k f(S^*) \leq e^{-1} f(S^*)$$

which implies that

$$f(S_k) \geq \left(1 - \frac{1}{e}\right) f(S^*)$$

□

4.2 Property variations

What happens when we *drop* some of the properties of NN-MI submodular functions? For instance, consider the following function.

Definition 4.5: Cut function

Given a graph G such that $V(G) = [n]$, the **cut function** c is a function defined as follows

$$c : \mathcal{P}([n]) \to \mathbb{R} : S \mapsto |\text{cut}(S)|$$

We observe that this function is trivially non-negative and symmetric, but not monotone increasing: given two nodes $x, y \in V(K_3)$, clearly $\{x\} \subseteq \{x, y\}$ but $c(\{x\}) \geq c(\{x, y\})$. Nonetheless, we can still prove that is is submodular, as described below.

Proposition 4.3

The cut function is non-negative, symmetric and submodular.

Proof. The non-negativity and symmetry of the function trivially hold by definition, hence we are going to prove that it is submodular. Fix two sets $S, T \subseteq [n]$, and define the following 4 sets as follows

$$A_S := S - T \quad A_T := T - S \quad A_{S,T} := S \cap T \quad A := [n] - (S \cup T)$$

Now, that

$$\operatorname{cut}(S) = \operatorname{cut}(A_S, A_T) \cup \operatorname{cut}(A_S, A) \cup \operatorname{cut}(A_{S,T}, A_T) \cup \operatorname{cut}(A_{S,T}, A)$$

meaning that

$$c(S) = |\operatorname{cut}(A_S, A_T)| + |\operatorname{cut}(A_S, A)| + |\operatorname{cut}(A_{S,T}, A_T)| + |\operatorname{cut}(A_{S,T}, A)|$$

Similarly, we get that

$$\begin{aligned} c(T) &= |\operatorname{cut}(A_T, A_S)| + |\operatorname{cut}(A_T, A)| + |\operatorname{cut}(A_{S,T}, A_S)| + |\operatorname{cut}(A_{S,T}, A)| \\ c(S \cup T) &= |\operatorname{cut}(A_S, A)| + |\operatorname{cut}(A_T, A)| + |\operatorname{cut}(A_{S,T}, A)| \\ c(S \cap T) &= |\operatorname{cut}(A_{S,T}, A_S)| + |\operatorname{cut}(A_{S,T}, A_T)| + |\operatorname{cut}(A_{S,T}, A)| \end{aligned}$$

meaning that

$$c(S) + c(T) = c(S \cup T) + c(S \cap T) + 2\,|\operatorname{cut}(A_S, A_T)| \geq c(S \cup T) + c(S \cap T)$$

implying that c is indeed submodular. □

Feige, Mirrokni, and Vondrák [FMV11] proved that the $\frac{1}{2}$-approximation that we discussed in Algorithm 1.1 can be generalized to *any* function that satisfies the same properties of the cut function. Moreover, even though the Max Cut problem has a better approximation ratio given by Algorithm 2.5, for generic functions this bound is actually tight.

Lemma 4.1

Given $n \in \mathbb{N}$, if $g : \mathcal{P}([n]) \to \mathbb{R}$ is submodular, it holds that

$$\forall S \subseteq [n] \quad \operatorname*{avg}_{T \subseteq S} g(T) \geq \frac{g(\varnothing) + g(S)}{2}$$

Proof. We proceed by strong induction on $|S|$. For the base case, if $|S| = 0$ then $S = \varnothing$, therefore

$$g(\varnothing) = \operatorname*{avg}_{T \subseteq \varnothing} g(T) \geq \frac{g(\varnothing) + g(\varnothing)}{2} = g(\varnothing)$$

Now, assume that the statement holds for $|S| \leq i$, and consider $|S| = i + 1$. Fix $x \in S$, and let $S' := S - \{x\}$ — in particular $|S'| = i$. Hence, we get that

$$\begin{aligned}
\sum_{T\subseteq S} g(T) &= \sum_{T'\subseteq S'} g(T') + \sum_{T'\subseteq S'} g(T'\cup\{x\}) \\
&= \sum_{T'\subseteq S'} g(T') + \sum_{T'\subseteq S'} g(T'\cup\{x\}) + \sum_{T'\subseteq S'} g(T') - \sum_{T'\subseteq S'} g(T') \\
&= 2\sum_{T'\subseteq S'} g(T') + \sum_{T'\subseteq S'} (g(T'\cup\{x\}) - g(T')) \\
&= 2\sum_{T'\subseteq S'} g(T') + \sum_{\substack{T\subseteq S:\\ x\in T}} (g(T) - g(T\cap S')) \\
&\geq 2\sum_{T'\subseteq S'} g(T') + \sum_{\substack{T\subseteq S:\\ x\in T}} (g(T\cup S') - g(S')) && \text{(by submodularity of } g\text{)} \\
&= 2\sum_{T'\subseteq S'} g(T') + \sum_{\substack{T\subseteq S:\\ x\in T}} (g(S) - g(S')) && (x\in T \implies T\cup S' = S) \\
&= 2\sum_{T'\subseteq S'} g(T') + 2^i(g(S) - g(S')) && \left(|\{T\subseteq S \mid x\in T\}| = 2^i\right)
\end{aligned}$$

Now, multiplying both sides of the inequality by $2^{-|S|}$, we get that

$$\operatorname*{avg}_{T\subseteq S} g(T) = 2^{-|S|}\sum_{T\subseteq S} g(T) \geq 2^{1-|S|}\sum_{T'\subseteq S'} g(T') + 2^{i-|S|}(g(S) - g(S'))$$

and recalling that $|S| = i+1$, we have that

$$\begin{aligned}
\operatorname*{avg}_{T\subseteq S} g(T) &\geq 2^{-i}\sum_{T'\subseteq S'} g(T') + 2^{-1}(g(S) - g(S')) \\
&= \operatorname*{avg}_{T'\subseteq S'} g(T') + \frac{g(S) - g(S')}{2} \\
&\geq \frac{g(\varnothing) + g(S')}{2} + \frac{g(S) - g(S')}{2} && \text{(by the strong inductive hypothesis)} \\
&= \frac{g(\varnothing) + g(S)}{2}
\end{aligned}$$

□

Lemma 4.2

Given $n\in\mathbb{N}$, if $g:\mathcal{P}([n])\to\mathbb{R}$ is submodular, it holds that

$$\forall S_1, S_2\subseteq[n] \quad \operatorname*{avg}_{T_1\subseteq S_1}\operatorname*{avg}_{T_2\subseteq S_2} g(T_1\cup T_2) \geq \frac{g(\varnothing) + g(S_1) + g(S_2) + g(S_1\cup S_2)}{4}$$

Proof. Given $X\subseteq[n]$, for any $T\subseteq[n]$ let $h_X(T) := g(X\cup T)$.

Claim: For any $X \subseteq [n]$, the function h_X is submodular.

Proof of the Claim. Fix $A, B \subseteq [n]$; we observe that

$$\begin{aligned} h_X(A) + h_X(B) &= g(X \cup A) + g(X \cup B) \\ &\geq g(X \cup A \cup B) + g((X \cup A) \cap (X \cup B)) && \text{(by submodularity of } g\text{)} \\ &= g(X \cup A \cup B) + g(X \cup (A \cap B)) \\ &= h_X(A \cup B) + h_X(A \cap B) \end{aligned}$$

□

Since h_X is submodular, by the previous lemma observe that

$$\forall S \subseteq [n] \quad \operatorname*{avg}_{T \subseteq S} h_X(T) \geq \frac{h_X(\varnothing) + h_X(S)}{2} = \frac{g(X) + g(X \cup S)}{2}$$

Therefore, for any $S_1, S_2 \subseteq [n]$, we have that

$$\begin{aligned} \operatorname*{avg}_{T_1 \subseteq S_1} \operatorname*{avg}_{T_2 \subseteq S_2} g(T_1 \cup T_2) &= \operatorname*{avg}_{T_1 \subseteq S_1} \operatorname*{avg}_{T_2 \subseteq S_2} h_{T_1}(T_2) \\ &\geq \operatorname*{avg}_{T_1 \subseteq S_1} \frac{g(T_1) + g(T_1 \cup S_2)}{2} && \text{(by the previous obs.)} \\ &= \frac{1}{2}\left(\operatorname*{avg}_{T_1 \subseteq S_1} g(T_1) + \operatorname*{avg}_{T_1 \subseteq S_1} g(T_1 \cup S_2)\right) \\ &= \frac{1}{2}\left(\operatorname*{avg}_{T_1 \subseteq S_1} g(T_1) + \operatorname*{avg}_{T_1 \subseteq S_1} h_{S_2}(T_1)\right) \\ &\geq \frac{1}{2}\left(\frac{g(\varnothing) + g(S_1)}{2} + \operatorname*{avg}_{T_1 \subseteq S_1} g_{S_2}(T_1)\right) && \text{(by the previous lemma)} \\ &\geq \frac{1}{2}\left(\frac{g(\varnothing) + g(S_1)}{2} + \frac{g(S_2) + g(S_2 \cup S_1)}{2}\right) && \text{(by the previous obs.)} \\ &= \frac{g(\varnothing) + g(S_1) + g(S_2) + g(S_1 \cup S_2)}{4} \end{aligned}$$

□

Theorem 4.4

Given $n \in \mathbb{N}$, let $f : \mathcal{P}([n]) \to \mathbb{R}$ be a submodular, non-negative and symmetric function. Given $S^* \in \arg\max_{S \subseteq [n]} f(S)$, if R is chosen from $[n]$ uniformly at random (UAR), it holds that

$$\mathbb{E}\left[f(R)\right] \geq \frac{1}{2} f(S^*)$$

Proof. If we apply the previous lemma on S^* and $[n] - S^*$, we get that

$$\begin{aligned}
\frac{f(\varnothing) + f(S^*) + f([n] - S^*) + f([n])}{4} &\leq \operatorname*{avg}_{T_1 \subseteq S^*} \operatorname*{avg}_{T_2 \subseteq [n]-S^*} f(T_1 \cup T_2) \\
&= 2^{-|S^*|} \cdot 2^{-n+|S^*|} \cdot \sum_{T_1 \subseteq S^*} \sum_{T_2 \subseteq [n]-S^*} f(T_1 \cup T_2) \\
&= 2^{-n} \sum_{T_1 \subseteq S^*} \sum_{T_2 \subseteq [n]-S^*} f(T_1 \cup T_2) \\
&= 2^{-n} \sum_{T \subseteq S^* \cup ([n]-S^*)} f(T) \qquad (S^* \cap ([n] - S^*) = \varnothing) \\
&= 2^{-n} \sum_{T \subseteq [n]} f(T) \\
&= \operatorname*{avg}_{T \subseteq [n]} f(T) \\
&= \mathbb{E}[f(R)]
\end{aligned}$$

Lastly, observe that

$$\begin{aligned}
\mathbb{E}[f(R)] &\geq \frac{f(\varnothing) + f(S^*) + f([n] - S^*) + f([n])}{4} \\
&\geq \frac{f(S^*) + f([n] - S^*)}{4} && \text{(by non-negativity of } f\text{)} \\
&= \frac{2f(S^*)}{4} && \text{(by symmetry of } f\text{)} \\
&= \frac{1}{2} f(S^*)
\end{aligned}$$

□

We observe this proof can be also modified for functions that are non-negative, submodular but *not* symmetric, as shown below.

Theorem 4.5

Given $n \in \mathbb{N}$, let $f : \mathcal{P}([n]) \to \mathbb{R}$ be a submodular and non-negative function. Given $S^* \in \arg\max_{S \subseteq [n]} f(S)$, if R is chosen from $[n]$ UAR, it holds that

$$\mathbb{E}\left[f(R)\right] \geq \frac{1}{2} f(S^*)$$

Proof. Consider the proof of the previous theorem; by simply replacing the application of the symmetry of f in the last step with another application of the non-negativity, we obtain a $\frac{1}{4}$-approximation — leaving the rest of the proof unchanged. □

In conclusion, we showed that

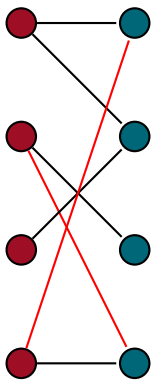
- NN-MI submodular functions have a $\left(1 - \frac{1}{e}\right)$-approximation
- NN-S submodular functions have a $\frac{1}{2}$-approximation
- NN submodular function have a $\frac{1}{4}$-approximation

Submodularity provides a standard approximation bound to many NP-hard *maximization* problems. Furthermore, these same results can be obtained through *supermodularity* for many NP-hard *minimization* problems.

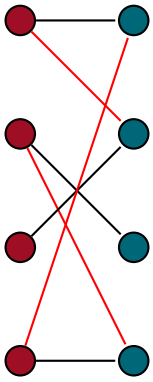
5

Online algorithms

In previous chapters, we discussed the **Maximal Matching** problem primarily through its connections to other computational problems. In this section, however, we shift our focus to *matchings* in **bipartite graphs** — i.e. the Maximal Bipartite Matching (MBM) problem. These matchings are particularly significant due to their applicability in modeling a wide range of real-world scenarios. A classic example is the **task assignment problem**, where the goal is to allocate tasks to employees optimally. This can be framed as a *maximum matching* problem in a bipartite graph: one partition represents the *tasks*, the other the *employees*, and edges connect each employee to the tasks they are qualified to perform.



Consider the matching on the bipartite graph shown in this figure: we observe that it is neither *maximal* nor *maximum*. In fact, we can still add an edge to our matching and obtain a matching of larger cardinality, as shown below:



However, this matching is now *maximal*, but not *maximum*. In fact, the following is a matching of the same bipartite graph but that has larger cardinality than the previous one:

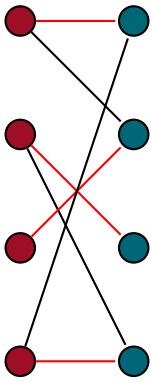


Figure 5.1: A matching on a bipartite graph.

However, let us take a closer look to what differentiates our two matchings: we see that to obtain one from the other we can simply *swap* all the edges of our matching. Moreover, we see that all the edges of the maximal matching actually form a *path*, whose edges alternate between being outside and inside the matching. Furthermore, both the first and the last edge of this path are outside the matching, which must imply that the number of edges *outside* the matching is one more than the number of inside edges.

In general, given a matching M in a graph G, we say that a vertex $v \in V(G)$ is **free** w.r.t. M if there are no edges $e \in M$ such that $e \cap v \neq \varnothing$. An edge $xy \in E(G)$ is said to be *disjoint* from M if both x and y are free w.r.t. M.

Now, we can formalize the concept that we introduced on this small example through the following definitions.

Definition 5.1: Alternating path

Given a graph G, and a matching M on G, an **M-alternating path** is a path of G that starts at a free node w.r.t. M, and is composed of edges that alternate between M and $E(G) - M$.

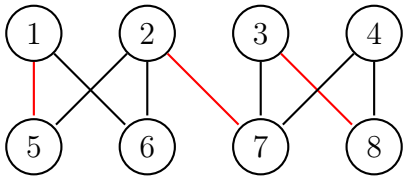


Figure 5.2: For instance, if M is the set of *red* edges — which forms a matching of the graph — then 6 $\{6,2\}$ 2 $\{2,7\}$ 7 $\{7,3\}$ 3 $\{3,8\}$ 8 is an M-alternating path.

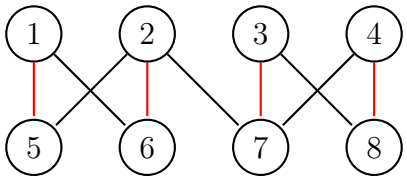
Definition 5.2: Augmenting path

Given a graph G, and a matching M on G, an **M-augmenting path** is an M-alternating path that ends at a free vertex w.r.t. M.

For example, if we consider the path

$$6\ \{6,2\}\ 2\ \{2,7\}\ 7\ \{7,3\}\ 3\ \{3,8\}\ 8\ \{8,4\}\ 4$$

this is actually an M-augmenting path of the previous graph. Augmenting paths are very useful because they can be used to *expand* the cardinality of an initial matching. In fact, in the previous graph we can actually define a *larger* matching by **swapping** the edges of this augmenting path, as shown below



This suggests that the presence of augmenting paths in a graph is a *sufficient* condition for a matching *not* to be maximum, but we can actually prove that it is also *necessary*, as stated in the following theorem, proved by Berge [Ber57] in 1957.

Theorem 5.1: Berge's theorem

Given a graph G, M is a maximum matching of G if and only if in G there are no M-augmenting paths.

Proof.

Direct implication. By contrapositive, consider a graph G and a matching M such that there is an M-augmenting path P in G. Moreover, by way of contradiction assume that M is maximum; however $M\Delta E(P)$ is a larger matching than M — here, Δ is the symmetric difference, therefore the operation $M\Delta E(P)$ has the same effect of *swapping* the edges of P between the ones in M and in $E(P) - M$.

Converse implication. By contrapositive, consider a graph G and a matching M of G that is not maximum, i.e. there exists a matching M^* of G such that $|M| < |M^*|$.

Consider the subgraph of G that has the vertices of $V(G)$ and the edges described by $M \Delta M^*$; the symmetric difference of these two sets will yield the set of edges that are either in M or in M^*, but not in $M \cap M^*$, therefore this subgraph is *not* a multigraph. Moreover, since M and M^* are both matchings, we have that

(1) the degrees of the vertices of this subgraph can be either 0, 1 or 2

(2) in each component of the subgraph the edges *must* alternate between M and M^*

By the observation (1), we have that each component of the subgraph can be either

- a isolated vertex
- a cycle
- a path

and by observation (2), we have that all the cycle components must have *even* length, which implies that they have the same number of edges of M and M^*. On the other hand, path components may have either even or odd length; in particular, even-length paths must have the same number of edges of M and M^* — as for cycle components — while odd-length paths have a different number of edges of M and M^*. However, since $|M| < |M^*|$, there must be at least one path component of this subgraph such that its edges of M are less than the edges of M^*, and this is clearly an M-augmenting path.

□

This theorem has been used to construct many algorithms for finding maximum matchings on bipartite graphs, the easiest one being the following.

Algorithm 5.1: Maximum Bipartite Matching

Given a bipartite graph G, the algorithm returns a maximum matching of G.

```
1: function MaximumBipartiteMatching(G)
2:     M := ∅
3:     while ∃P M-augmenting path do
4:         M = MΔP
5:     end while
6: end function
```

In this algorithm, augmenting paths can be identified using either a DFS or a BFS, resulting in a runtime of $O(mn)$. However, more efficient variants have been developed to reduce this complexity by optimizing the way augmenting paths are found — most notably, the Hopcroft–Karp algorithm [HK73]. Given a bipartite graph G bipartitioned through (A, B), the key idea is to perform a BFS starting simultaneously from all unmatched vertices in A, continuing until at least one free vertex in B is reached; this produces a BFS forest. Then, a DFS is executed over such forest, beginning from the unmatched nodes

in B, to discover augmenting paths. Each such path allows for alternating the matching along its edges, thereby increasing the overall matching size. The algorithm repeats this process until no more augmenting paths can be found. This approach improves the runtime to $O(m\sqrt{n})$, a significant efficiency gain over the basic method.

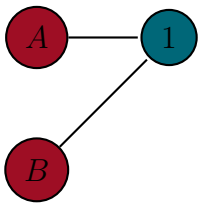
5.1 The online approach

An **online algorithm** processes its input *piece-by-piece* in a sequential manner, without having access to the entire input from the outset. In contrast, a traditional offline algorithm receives *all the input at once* and must then compute a solution. Online algorithms are valuable because they enable solving problems without requiring full knowledge of the data beforehand. The importance of online algorithms has grown significantly with the rise of the Internet and the explosion of big data, where handling massive datasets has made memory management challenging, even for the most advanced supercomputers.

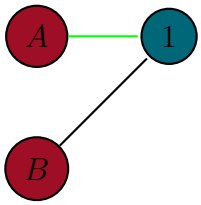
A practical example of online algorithms is seen in MBM, which is critical for example in online **advertising services**. Imagine an advertising platform where each user should be matched to *at most one ad per hour*, and each ad can be assigned to *at most one user per hour*. Based on user interests, the platform generates a list of possible ads for each individual. To maximize revenue, it aims to find the **largest possible matching** between users and ads. This scenario directly corresponds to MBM, where users form one set of the bipartite graph and ads form the other. However, in this context some challenges may arise, for instance

- the sheer volume of users and advertisements makes storing all the data impractical
- the set of available users and ads changes continuously, so earlier matches might no longer be optimal

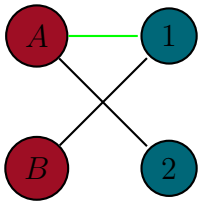
Online algorithms address both of these issues effectively. Still, because they must make decisions without knowing future inputs, they sometimes produce results that are *not optimal in hindsight*. For instance, suppose that our online algorithm currently knows the following association graph — in which the *red* nodes indicate the **ads** and the *blue* ones indicate the **users**



Our algorithm has two options: either match user 1 to ad A or match it to ad B. Without loss of generality, assume that user 1 is matched to ad A, as shown below



Now, suppose that a new user 2 joins our service. After analyzing the preferences of the users, the system determines that only ad A is compatible with user 2, meaning that the latter cannot be assigned any ad since ad A is already matched with user 1.



However, in hindsight we see that this is *not* the optimal solution: in fact, if our algorithm had matched user 1 to ad B, we could've been able to match both users, as shown below

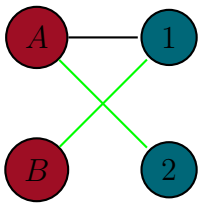


Figure 5.3: The optimal configuration, *in hindsight*.

This tendency toward non-optimal solutions is a *defining feature* of online algorithms and is often *unavoidable*. As a result, online algorithms are typically evaluated through **competitive analysis**, which measures the performance of an online algorithm relative to an optimal offline algorithm. The offline algorithm has the advantage of seeing the entire sequence of requests beforehand, while the online algorithm must respond to an *unpredictable sequence*, making decisions without knowledge of future inputs. Unlike traditional worst-case analysis — which evaluates an algorithm's behavior only on particularly difficult inputs — competitive analysis demands strong performance across both easy and hard inputs, where difficulty is determined based on how well the optimal offline algorithm performs.

To formally assess the competitiveness of online algorithms, researchers employ the **adversary model**, in which an adversary provides *deliberately challenging inputs* to the algorithm. In the case of deterministic algorithms, the adversary is assumed to have *complete knowledge* of the algorithm's internal workings. Given that the online algorithm cannot anticipate future inputs, this all-knowing adversary can exploit its decisions and lead it into making suboptimal choices.

In particular, we are going to employ such adversary model to prove the following result.

Theorem 5.2

There is no deterministic online algorithm for MBM that can match more than $\frac{1}{2}$ of the edges of an optimal solution.

Proof. We prove the result through an adversarial argument. On the first round, the adversary sends three vertices a, b, x, and two edges $\{a, x\}$ and $\{b, x\}$. The online algorithm has then 3 possible choices for x.

1. Match x with a. On the next round, the adversary sends a new vertex y and an edge $\{a, y\}$. Since x has been matched to a, the new vertex y cannot be matched, meaning that the algorithm returns a solution with a matching having size 1. However, an offline optimal solution would have returned the matching $\{\{b, x\}, \{a, y\}\}$, which has size 2. Therefore, the competitive ratio is $\frac{1}{2}$.
2. Match x with b. Similarly to the previous case, on the next round the adversary sends a new vertex y and an edge $\{b, y\}$. Since x has been matched to b, the new vertex y cannot be matched, but an offline solution would have returned the matching $\{\{a, x\}, \{b, y\}\}$. Therefore, the competitive ratio is still $\frac{1}{2}$.
3. Leave x unmatched. On the next round, the adversary sends a new vertex y and an edge $\{a, y\}$. Now our algorithm has two options.
   - Match y with a; this solution is suboptimal, because an offline algorithm would have matched both x and y. Hence, the competitive ratio is again $\frac{1}{2}$
   - Leave y unmatched; this solution is suboptimal, because an offline algorithm would have matched both x and y. Hence, the competitive ratio is 0

This concludes that the adversary can always send a set of vertices and edges such that the competitive ratio is at most $\frac{1}{2}$. □

This result determines an approximation ratio upper bound on *every possible deterministic online algorithm*, concluding that a perfect solution for MBM is **impossible**. Knowing this, we shift our focus on finding a greedy solution that reaches such maximum approximation ratio. In this case, the best solution is the most straightforward: when a new user arrives, we match it if it can be matched, otherwise we leave it unmatched and proceed with the next round.

Algorithm 5.2: The online maximal matching algorithm

Given a graph G fed in an online fashion, the algorithm returns a maximal mathing of G.

```
1: function onlineMaximalMatching(G)
2:     M := ∅
3:     while there is a new user x do
4:         Read 𝒩(x)
5:         U(x) := {v ∈ 𝒩(x) | ∄e ∈ M : v ∈ e}
6:         if U(x) ≠ ∅ then
7:             Choose y ∈ U(x)
8:             M = M ∪ {xy}
9:         end if
10:    end while
11:    return M
12: end function
```

The previous theorem, together with the following proposition, imply that this online greedy algorithm is an optimal approximation for MBM.

Proposition 5.1

Given a graph G, a maximum matching M^* on G, and a maximal matching M of G, we have that

$$|M| \geq \frac{1}{2} |M^*|$$

Proof. Let $S := \{v \in V(G) \mid \exists e \in M : v \in e\}$ be the set of vertices of G that are matched by edges of M. Hence, by definition we have that $|S| = 2\,|M|$.

We observe that:

- each node of S is incident to at most one edge of M^*, otherwise M^* would not have been a matching
- each edge $e \in M^*$ must be incident to at least one node of S, since otherwise $M \cup \{e\}$ would be matching that extends M, contradicting the fact that it was maximal

Now, consider a function $f : M^* \to S$ such that $f(e) \in S \cap e$: the first observation implies that f is *well-defined*, while the second observation implies that f is *injective*. In particular, this concludes that

$$|M^*| \leq |S| = 2\,|M| \iff |M| \geq \frac{1}{2} |M^*|$$

□

To overcome some of the limitations of *deterministic* online algorithms that we presented, **randomness** can be introduced into the algorithm's behavior. In randomized online algorithms, the nature of the adversary varies depending on how much information they possess. A commonly studied scenario involves a randomized adversary, where the input sequence is generated according to a *random distribution*. However, even in this setting we can still prove the following result.

Theorem 5.3

There is no randomized online algorithm for MBM that can expectedly match more than $\frac{3}{4}$ of the edges of an optimal solution for graphs chosen from a random distribution.

Proof. Let G_1 be a graph such that $E(G_1) = \{\{a, x\}, \{b, x\}, \{b, y\}\}$, and let G_2 be the graph such that $E(G_2) = \{\{a, x\}, \{b, x\}, \{a, y\}\}$. Now, consider a distribution that assigns probability $\frac{1}{2}$ to both G_1 and G_2, and probability 0 to all the other possible graphs.

On the first round, the adversary sends the edges $\{a, x\}$ and $\{b, x\}$; then, our algorithm will match x to either a or b , and clearly it will learn nothing about the vertex y whose edges will be sent by the adversary on the next round — namely, they can send either $\{b, y\}$ or $\{a, y\}$. Through a similar argument used to prove Theorem 5.2, we see that the probability of y being impossible to match on the next round is $\frac{1}{2}$. This implies that

what if x unmatched?

$$\mathbb{E}[\text{number of matched edges}] = TODO$$

TODO However, an offline deterministic algorithm (which is a also randomized algorithm that makes no random choices) can always retun an optimal solution with 2 amtched edges, therefore the expected ratio is

prev eq

$$\frac{\frac{3}{2}}{2} = \frac{3}{4}$$

□

5.1.1 The Market Process algorithm

With a similar argument, it can be shown that the bound discussed in the previous section can be improved to $1 - \frac{1}{e}$.

The standard algorithm for the Online Bipartite Matching problem is known as the **Ranking Algorithm**, originally developed by Karp, Vazirani, and Vazirani [KVV90]. The output yielded by this algorithm is guaranteed to be an expected $\left(1 - \frac{1}{e}\right)$-approximation, which is tight considering the bound above.

Over the years, the algorithm has been revised and simplified, while keeping the same approximation ratio. The modern version of this algorithm is known as the **Market Process Algorithm**, developed by Eden, Feldman, Fiat, et al. [EFF+21] — the name comes from its economics-based interpretation.

We will refer to the bipartitions of the graph as the pair (A, U), where A is a set of advertisements and U is a set of users.

Algorithm 5.3: The Market Process algorithm

Given a bipartite graph G with bipartition (A, U), the algorithm returns a matching on G.

---

function MarketProcess(G)
  for $j \in A$ **do**
    Sample w_j from $[0, 1]$ independently and UAR
    $p_j := e^{w_j - 1}$ ▷ the price of the ad j
  end for
  $M := \varnothing$
  for $i \in U$ **do**
    $A' := \{j \in A \mid \nexists i' \in U : \{i', j\} \in M\}$ ▷ set of unmatched ads
    for $j \in A'$ **do**
      $v_i(j) := \mathbb{1}[j \in \mathcal{N}(i)]$ ▷ the value of ad j for user i
      $u_i(j) := v_i(j) - p_j$ ▷ the utility of ad j for user i
    end for
    $u_i(\perp) := 0$ ▷ the utility of "not being matched" for user i
    if $A' \cap \mathcal{N}(i) \neq \varnothing$ **then** ▷ otherwise i remains unmatched
      Choose $j^* \in \arg\max_{j \in A' \cup \{\perp\}} u_i(j)$ ▷ j^* maximizes the utility for user i
      $M = M \cup \{\{i, j^*\}\}$
    end if
  end for
  return M
end function

We observe that, on each iteration, for all $j \in A'$ it holds that $u_i(j) \geq 0$ if and only if $j \in \mathcal{N}(i)$ — since $v_i(j)$ can only be either 0 or 1 and

$$w_j \in [0, 1] \implies \tfrac{1}{e} \leq p_j := e^{w_j - 1} \leq 1$$

Now, before proving the approximation ratio of the algorithm, we are going to introduce some notation:

- given a matching M, we write $M(i) = j$ to express that user i is matched to ad j through M, and $M(i) = \perp$ to express that user i is not matched
- for each ad $j \in A$, let rev_j be the *revenue* for ad j, where

$$\text{rev}_j := \begin{cases} p_j & j \text{ is matched} \\ 0 & \text{otherwise} \end{cases}$$

- for each user $i \in U$, let util_i be the *utility* for user i, where

$$\forall j \in A \quad \text{util}_i := \begin{cases} 1 - p_j & M(i) = j \\ 0 & M(i) = \perp \end{cases}$$

Moreover, we observe that by the bounds on p_j it holds that $0 \leq \text{rev}_j, \text{util}_i \leq 1$.

Lemma 5.1

Given a bipartite graph G with bipartition (A, U), let $w = [w_1\ w_2\ \dots]$ be the vector of the values sampled by running MarketProcess(G); then, for each $ij \in E(G)$ it holds that

$$\mathop{\mathbb{E}}_{w}[\text{util}_i + \text{rev}_j] \geq 1 - \frac{1}{e}$$

Proof. Omitted. □

Theorem 5.4

Given a bipartite graph G with bipartition (A, U), let M^* be an optimal solution to the Online Bipartite Matching problem on G; then, given $M = \text{MarketProcess}(G)$, it holds that

$$|M| \geq \left(1 - \frac{1}{e}\right) |M^*|$$

Proof. Let $w = [w_1\ w_2\ \dots]$ be the vector of the values sampled by running the algorithm on input G, i.e. MarketProcess(G). Given the output M, it holds that

$$\sum_{i \in U} \text{util}_i + \sum_{j \in A} \text{rev}_j = \sum_{ij \in M} (1 - p_j) + p_j = \sum_{ij \in M} 1 = |M|$$

which implies that

$$\begin{aligned}
\mathop{\mathbb{E}}_{w}[|M|] &= \mathop{\mathbb{E}}_{w}\left[\sum_{i \in U} \text{util}_i + \sum_{j \in A} \text{rev}_j\right] \\
&\geq \mathop{\mathbb{E}}_{w}\left[\sum_{ij \in M^*} \text{util}_i + \text{rev}_j\right] \\
&= \sum_{ij \in M^*} \mathop{\mathbb{E}}_{w}[\text{util}_i + \text{rev}_j] && (0 \leq \text{rev}_j, \text{util}_i \leq 1) \\
&\geq \sum_{ij \in M^*} \left(1 - \frac{1}{e}\right) && \text{(previous lemma)} \\
&= \left(1 - \frac{1}{e}\right) |M^*|
\end{aligned}$$

□

5.2 The Expert model

In statistics, economics, and machine learning, the so-called **expert model** is a technique used to combine several *weaker models* into a more *powerful* overall model. This approach is closely related to the concept of **boosting** (or *bootstrapping*). Given a training dataset, the data is divided into n separate chunks, and n different models called **experts** are trained independently one on each chunk. During testing, each expert produces a prediction, resulting in n predictions. The expert model then determines how to *combine* these predictions or selects the most appropriate one to form the final output.

The idea can be summarized as follows:

1. let $m_1, \ldots, m_n$ be the trained experts
2. at each time $t = 1, \ldots, T$ we read a *prediction vector* $y_t = [y_{1,t} \ \cdots \ y_{n,t}]$ where each $y_{i,t} \in \{0, 1\}$ is a prediction of the i-th expert m_i
3. using y_t we produce a final prediction $z_t \in \{0, 1\}$
4. after reading the real outcome x_t, we lose a point if $z_t \neq x_t$, otherwise we gain one

The easiest way to use the prediction vector y_t is to randomly choose which of the experts' prediction will be the final one. If there is at least one **perfect expert** — an expert whose predictions are always correct — we can use a naïve alogritm in order to discard the non-perfect experts until only the perfect ones remain.

Algorithm 5.4: The Naïve Expert model

Given a set M of n trained experts, in which at least one of them is perfect, the algorithm yields predictions over time T.

1: **function** NAÏVEEXPERTMODEL(M, T)
2: $S := [n]$
3: **for** $t \in [T]$ **do**
4: Read $y_t = [y_{1,t} \ \cdots \ y_{n,t}]$
5: Choose $i \in S$ UAR
6: Read the outcome $x_t \in \{0, 1\}$
7: **if** $x_t \neq y_{i,t}$ **then**
8: $S = S - \{j \mid y_{j,t} \neq x_t\}$
9: **end if**
10: **end for**
11: **end function**

We observe that, in the "worst" case at each iteration only the i-th expert has made a mistake, meaning that this procedure will make at most $n - 1$ mistakes.

Proposition 5.2

Given a set of n trained experts M, in which at least one of them is perfect, the procedure NAÏVEEXPERTMODEL(M) makes at most $n-1$ mistakes.

More complex expert systems make use of a *large* number of experts, therefore this naïve procedure requires too many iterations to "reach" the perfect experts. The following approach provides a faster alternative.

Algorithm 5.5: The Halving Expert model

Given a set M of n trained experts, in which at least one of them is perfect, the algorithm yields predictions over time T.

```
1: function HALVINGEXPERTMODEL(M, T)
2:     S := [n]
3:     for t ∈ [T] do
4:         Read y_t = [y_{1,t} ··· y_{n,t}]
5:         Let z_t be the majority of the predictions in y_t
6:         Read the outcome x_t ∈ {0, 1}
7:         if x_t ≠ z_t then
8:             S = S − {j | y_{j,t} ≠ x_t}
9:         end if
10:    end for
11: end function
```

This algorithm is identical to the naïve approach except for a very important detail: instead of choosing the i-th expert UAR, this approach uses a **majority voting system**. In fact, since z_t is the majority of the predictions in y_t, and the values of the outputs are either 0 or 1, at each mistake now *at least half* of the experts must be wrong, thus at least half of the experts are discarded at each error. Hence, we get the following proposition.

Proposition 5.3

Given a set of n trained experts M, in which at least one of them is perfect, the procedure HALVINGEXPERTMODEL(M) makes at most $\lfloor \log n \rfloor$ mistakes.

Both of these algorithms rely on the guarantee that *at least one* perfect model is present in the input, but what if there is no such model? In this case, the algorithm will clearly discard all the experts eventually. Hence, consider the following procedure.

Algorithm 5.6: The Iterative Halving Expert model

Given a set M of n trained experts, the algorithm yields predictions over time T.

```
1: function ITERATIVEHALVINGEXPERTMODEL(M, T)
2:     S := [n]
3:     for t ∈ [T] do
4:         Read y_t = [y_{1,t} ··· y_{n,t}]
5:         Let z_t be the majority of the predictions in y_t
6:         Read the outcome x_t ∈ {0, 1}
7:         if x_t ≠ z_t then
8:             S = S − {j | y_{j,t} ≠ x_t}
9:             if S = ∅ then                ▷ if S became empty, repeat the process
10:                S = [n]
11:            end if
12:        end if
13:    end for
14: end function
```

This procedure is identical to the previous one, but now if S becomes empty we fill it again in order to repeat the process until time T. How many mistakes this algorithm makes at most?

Theorem 5.5

Given a set of n trained experts M, the number m of mistakes made by the procedure ITERATIVEHALVINGEXPERTMODEL(M, T) is such that

$$m \leq (m^* + 1)(\lfloor \log n \rfloor + 1)$$

where m^* is the number of mistakes of the best expert after T rounds.

Proof sketch. We can split the algorithm into *phases*, and each phase begins whenever S is set to $[n]$. Clearly, since there may be no perfect trained experts in the input, each expert could make at least one mistake during each phase. Moreover, in each phase the algorithm will make at most $\lfloor \log n \rfloor + 1$ mistakes, because of the majority rule. Hence, given that there are at most $m^* + 1$ phases, the bound follows. □

Can we improve this idea? Consider the following procedure.

Algorithm 5.7: The Weighted Majority Expert model

Given a set M of n trained experts, the algorithm yields predictions over time T.

---

function WEIGHTEDMAJORITYEXPERTMODEL(M, T)
  for $i \in [n]$ **do**
    $w_i := 1$
  end for
  for $t \in [T]$ **do**
    Read $y_t = [y_{1,t} \ \cdots \ y_{n,t}]$
    $A_t := \sum_{\substack{i \in [n]: \\ y_{i,t}=1}} w_i$
    $B_t := \sum_{\substack{i \in [n]: \\ y_{i,t}=0}} w_i$
    if $A_t \geq B_t$ **then**
      $z_t := 1$
    else
      $z_t := 0$
    end if
    Read the outcome $x_t \in \{0, 1\}$
    if $x_t \neq z_t$ **then**
      for $i \in [n]$ **do**
        if $y_{i,t} \neq x_t$ **then**
          $w_i = \frac{w_i}{2}$
        end if
      end for
    end if
  end for
end function

This procedure is a refined version of the halving model, in which we consider a **weighted majority** instead of a simple majority rule. This allows to *tweak the weights* of the experts — in this case, halving their weight if they were wrong — instead of removing them entirely.

Theorem 5.6

Given a set of n trained experts M, the number m of mistakes made by the procedure WEIGHTEDMAJORITYEXPERTMODEL(M, T) is such that

$$m \leq 2.41 \cdot (m^* + \log n)$$

where m^* is the number of mistakes of the best expert after T rounds.

Proof. Let $w_i^{(t)}$ be the weight of the i-th expert at the beginning of round t, before the

update step, and for each $t \in [T]$ let $W^{(t)} = \sum_{i=1}^{n} w_i^{(t)}$ be the sum of all the weights of the experts. We observe that

- $w_i^{(0)} = 1$ for all $i \in [n]$ by definition, hence $W^{(0)} = n$
- $W^{(t)} \geq W^{(t+1)}$ since on each iteration either all the weights remain the same or some of them get halved

but we can actually find a tighter bound.

Claim: On each round $t \in [T]$, if $z_t \neq x_t$ then $W^{(t+1)} \leq \frac{3}{4} W^{(t)}$.

Proof of the Claim. Let $I^{(t)}$ be the total weight of the wrong experts at the beginning of round t, i.e.

$$I^{(t)} := \sum_{i \in [n] : y_{i,t} \neq x_t} w_i^{(t)}$$

If $z_t \neq x_t$ then at least half of the total weight was placed on experts who made a mistake, hence $I^{(t)} \geq \frac{1}{2} W^{(t)}$. Thus, we have that

$$W^{(t+1)} = \frac{I^{(t)}}{2} + (W^{(t)} - I^{(t)}) = W^{(t)} - \frac{I^{(t)}}{2} \leq W^{(t)} - \frac{W^{(t)}}{4} = \frac{3}{4} W^{(t)}$$

□

Through the claim, we inductively get that

$$W^{(T)} \leq \left(\frac{3}{4}\right)^m W^{(0)} = \left(\frac{3}{4}\right)^m n$$

Now, if i^* is one of the "best" experts — i.e. one of those who made m^* mistakes — it holds that $W^{(T)} \geq w_{i^*}^{(T)}$. Since the i^*-th expert made m^* mistakes, its original weight got halved m^* times, therefore $w_{i^*}^{(T)} = 2^{-m^*}$.

Finally, putting everything together we get that

$$2^{-m^*} = w_{i^*}^{(T)} \leq W^{(T)} \leq \left(\frac{3}{4}\right)^m n$$

and solving for m, we get the upper bound of the claim

$$m \leq \frac{1}{\log \frac{4}{3}} (m^* + \log n) \leq 2.41 \cdot (m^* + \log n)$$

□

Even if it is not known whether this bound is *tight*, the following result can be proven.

Theorem 5.7

Given a set trained experts M, for every deterministic algorithm based on the expert model the total number of mistakes m is such that $m \geq \log n$ and $m \geq 2m^*$, where m^* is the number of mistakced of the best expert after T rounds.

Proof. By way of contradiction, suppose that there is an algorithm A_1 for the expert model that makes $m < 2m^*$ mistakes. Consider two experts E_0 and E_1 such that the first one always returns 0 and the second one always returns 1. Now, suppose that the algorithm makes a mistake on each of the T iterations, i.e. $m = T$.

Then, at time $t = T$ either E_0 or E_1 will have made at most $\frac{T}{2}$ mistakes, which implies that $m^* \leq \frac{T}{2}$. However, this means that

$$T = m < 2m^* \leq T$$

rasing a contradiction — the last inequality follows from the fact that m^* is always upper bounded by the number of rounds T. This proves that $m \geq 2m^*$, which proves the second bound of the statement.

Regarding the first bound, a similar argument could be applied through some 2^n experts $E_0, E_1, \ldots, E_{2^n}$ such that the first n outputs of each i-th expert corresponds to the binary encoding of i — e.g. for $n = 3$ we have that E_0 returns 0, 0, 0 while E_7 returns 1,1,1. □

Nonetheless, this bound can be improved through *randomization*, as presented in the following algorithm.

Algorithm 5.8: The Random Weighted Majority Expert model

Given a set M of n trained experts, the algorithm yields predictions over time T.

---

function RANDWEIGHTEDMAJORITYEXPERTMODEL(M, T)
  for $i \in [n]$ **do**
    $w_i := 1$
  end for
  for $t \in [T]$ **do**
    Read $y_t = [y_{1,t} \ \cdots \ y_{n,t}]$
    Create a distribution P_t such that $P_t(i) = \frac{w_i}{\sum_{j \in [n]} w_j}$
    Sample i form P_t
    Read the outcome $x_t \in \{0, 1\}$
    if $x_t \neq z_t$ **then**
      for $i \in [n]$ **do**
        if $y_{i,t} \neq x_t$ **then**
          $w_i = (1 - \varepsilon) w_i$
        end if
      end for
    end if
  end for
end function

Theorem 5.8

Given a set of n trained experts M, the number m of mistakes made by the procedure RANDWEIGHTEDMAJORITYEXPERTMODEL(M, T) is such that

$$\mathbb{E}[m] \leq (1+\varepsilon)m^* + \frac{1}{\varepsilon}\ln n$$

for each $0 < \varepsilon < \frac{1}{2}$ where m^* is the number of mistakes of the best expert after T rounds.

Proof. Let $w_i^{(t)}$ be the weight of the i-th expert at the beginning of round t, before the update step, and for each $t \in [T]$ let

- $W^{(t)} = \sum_{i=1}^{n} w_i^{(t)}$ be the sum of all the weights of the experts
- $I^{(t)}$ be the weighted fraction of experts that are wrong at rround t

$$I^{(t)} = \frac{\sum_{\substack{i\in[n]:\\ y_{i,t}\neq x_t}} w_i^{(t)}}{\sum_{i\in[n]} w_i^{(t)}} = \frac{1}{W^{(t)}} \sum_{\substack{i\in[n]:\\ y_{i,t}\neq x_t}} w_i^{(t)}$$

Let i^* be one of the best experts, i.e. one of those that make m^* mistakes; then, we have that

$$W^{(T)} \geq w_{i^*}^{T} = w_{i^*}^{(0)}(1-\varepsilon)^{m^*} = (1-\varepsilon)^{m^*}$$

given that $w_i^{(0)} = 1$ for all $i \in [n]$.

Claim 1: For each $t \in [T]$ it holds that $W^{(t)} = n\prod_{s=1}^{t}\left(1-\varepsilon I^{(s)}\right)$.

Proof of the Claim. We observe that for any $t \in [T]$ it holds that

$$\begin{aligned}
W^{(t+1)} &= \sum_{i=1}^{n} w_i^{(t+1)} \\
&= \sum_{i=1}^{n} w_i^{(t)}\left(1-\varepsilon\cdot\mathbb{1}\left[y_{i,t+1}\neq x_{t+1}\right]\right) \\
&= \sum_{i=1}^{n} w_i^{(t)} - \varepsilon\cdot\sum_{\substack{i\in[n]:\\ y_{i,t+1}\neq x_{t+1}}} w_i^{(t)} \\
&= W^{(t)} - \frac{W^{(t)}}{W^{(t)}}\cdot\varepsilon\cdot\sum_{\substack{i\in[n]:\\ y_{i,t+1}\neq x_{t+1}}} w_i^{(t)} \\
&= W^{(t)}\left(1-\varepsilon I^{(t+1)}\right)
\end{aligned}$$

Now, since $W^{(0)} = n$ by definition, we inductively get that

$$W^{(t+1)} = W^{(0)} \prod_{s=1}^{t+1} \left(1 - \varepsilon I^{(s)}\right) = n \prod_{s=1}^{t} \left(1 - \varepsilon I^{(s)}\right)$$

□

Hence, by this claim and the previous observation, we get that

$$n \prod_{s=1}^{T} \left(1 - \varepsilon I^{(s)}\right) = W^{(T)} \geq (1-\varepsilon)^{m^*} \implies \ln n + \sum_{s=1}^{T} \ln\left(1 - \varepsilon I^{(s)}\right) \geq m^* \ln(1-\varepsilon)$$

Now, since $\ln(1-x) \leq -x$ for all $0 \leq x < 1$, we also know that

$$\ln n - \varepsilon \sum_{s=1}^{T} I^{(s)} \geq \ln n + \sum_{s=1}^{T} \ln\left(1 - \varepsilon I^{(s)}\right)$$

thus concluding that

$$\ln n - \varepsilon \sum_{s=1}^{T} I^{(s)} \geq m^* \ln(1-\varepsilon)$$

which in turn implies that

$$\sum_{s=1}^{T} I^{(s)} \geq m^* \frac{1}{\varepsilon} \ln n - \frac{1}{\varepsilon} m^* \ln(1-\varepsilon)$$

Claim 2: $\mathbb{E}[m] = \sum_{s=1}^{T} I^{(s)}$.

Proof of the Claim. Through algebraic manipulation we get that

$$\begin{aligned}
\mathbb{E}[m] &= \sum_{s=1}^{T} \Pr[\text{mistake at round } s] \\
&= \sum_{s=1}^{T} \Pr[\text{a wrong expert is picked at round } s] \\
&= \sum_{s=1}^{T} \sum_{\substack{i \in [n]: \\ y_{i,s} \neq x_s}} \Pr[\text{expert } i \text{ picked at round } s] \\
&= \sum_{s=1}^{T} \sum_{\substack{i \in [n]: \\ y_{i,s} \neq x_x}} \frac{w_i^{(s)}}{\sum\limits_{i \in [n]} w_i^{(s)}} \\
&= \sum_{s=1}^{T} I^{(s)}
\end{aligned}$$

□

Finally, by the previous equivalence and Claim 2, we conclude that

$$\mathbb{E}[m] = \sum_{s=1}^{T} I^{(s)} \geq m^* \frac{1}{\varepsilon} \ln n - \frac{1}{\varepsilon} \ln(1-\varepsilon) = (1+\varepsilon)m^* + \frac{1}{\varepsilon} \ln n$$

□

5.3 Scoring rules

As previously mentioned, the expert model is commonly used in machine learning to identify the best predictor from a fixed set of models. While effective, this approach does not allow for modifying the experts themselves — they remain unchanged regardless of their performance. But what if we want to adapt the predictor set over time, for example, by removing or replacing models that frequently make mistakes?

This limitation is addressed by the **forecaster model**, which outputs probabilities rather than fixed predictions. For instance, consider a model predicting whether it will rain tomorrow. An expert model would use something like the weighted majority algorithm to output either a 0 (no rain) or a 1 (rain). In contrast, a forecaster model would return a probability distribution such as $(\mu, 1-\mu)$, where μ is the predicted probability of a sunny day.

In expert models, individual predictors adjust their weights after incorrect predictions — effectively incurring a penalty. For forecaster models, however, such **payoff scheme** must be adapted. If the forecaster outputs $(0.5, 0.5)$ and it rains, should that be considered wrong?

A possible payoff scheme works as follows: if the forecaster assigns a probability $x \in [0,1]$ to an event occurring, it earns x points if the event happens, and loses $1-x$ points if it doesn't. This is meant to encourage the forecaster to report $x = \mu$ as the prediction, where μ is what he believes the probability of the event really is. In particular, the **expected gain** $g_\mu(x)$ under this scheme, assuming the forecaster believes the event has probability μ and it claims x as the probability of the event, is given by

$$g_\mu(x) = \mathop{\mathbb{E}}_{\mu}[P_x] = x\mu + (1-x)(1-\mu)$$

However we observe that

$$\frac{d}{dx} g_\mu(x) = \mu + (1-\mu)(-1) = 2\mu - 1$$

which means that the minimum value of the gain is reached on $x = 1$ if $\mu < \frac{1}{2}$, and at $x = 0$ otherwise — recall that $\mu \in [0,1]$. This means that such payoff scheme actually incentivizes the model to *lie*, i.e. to claim that either $x = 1$ or $x = 0$ without values in between!

To fix this problem, we need a payoff scheme based on **scoring rules**. Scoring rules act as a generalization of the payoff scheme, where the gain and the payoff do not have to add up to 1. In other wrods, if the forecaster guesses $x \in [0,1]$ as the probability of the

event, then it gains $f_1(x)$ points if the event really happens, otherwise it loses $f_0(x)$ points — in the pvrious scheme, $f_1(x) = x$ and $f_0(x) = 1 - x$. Hence, the expected gain now becomes

$$g_\mu(x) = \mathop{\mathbb{E}}_{\mu}[P_x] = f_1(x)\mu + f_0(x)(1-\mu)$$

An optimal scoring rule would be one that makes the model converge to $x = \mu$.

Definition 5.3: Scoring rule

Given two functions $f_0, f_1 : X \to [0,1]$, we say that the pair (f_0, f_1) is a **scoring rule** if for all $\mu \in [0,1]$ it holds that

$$\arg\max_{x\in[0,1]} g_\mu(x) = \arg\max_{x\in[0,1]} \left(f_1(x)\mu + f_0(x)(1-\mu)\right) = \mu$$

We observe that this definition implies that there is *only one* maximizing x value, i.e.. μ itself.

Good scoring rules are *hard* to find, and vary depending on the context. However, even finding simple scoring rules is hard. The following is a commonly used scoring rule.

Definition 5.4: Quadratic Scoring Rule

The **Quadratic Scoring Rule** is defined as follows

$$f_0(x) = 1 - x^2$$

$$f_1(x) = 1 - (1-x)^2$$

Theorem 5.9

The Quadratic Scoring Rule is a scoring rule.

Proof. First, we observe that

$$\begin{aligned} g_\mu(x) &= f_1(x)\mu + f_0(x)(1-\mu) \\ &= (1-(1-x)^2)\mu + (1-x^2)(1-\mu) \\ &= \mu - \mu(1-x)^2 + (1-\mu) - (1-\mu)x^2 \\ &= 1 - \mu(1-2x+x^2) - (1-\mu)x^2 \\ &= 1 - \mu - 2\mu x + \mu x^2 - x^2 - \mu x^2 \\ &= 1 - \mu - 2\mu x - x^2 \end{aligned}$$

the derivative of which is

$$\frac{d}{dx} g_\mu(x) = \frac{d}{dx}(1 - \mu - 2\mu x - x^2) = 2\mu - 2x$$

This implies that $g_\mu(x)$ is strictly increasing for $x < \mu$ and strictly decreasing for $x > \mu$. Hence, it must be that the *unique maximum* is reached when $x = \mu$. □

Another known and commonly employed scoring rule is the following.

Definition 5.5: Binary Logarithmic Scoring Rule

The **Binary Logarithmic Scoring Rule** is defined as follows

$$f_0(x) = \ln(1-x)$$

$$f_1(x) = \ln x$$

Theorem 5.10

The Binary Logarithmic Scoring Rule is a scoring rule.

Proof. First, by definition we have that

$$\begin{aligned} g_\mu(x) &= f_1(x)\mu + f_0(x)(1-\mu) \\ &= \mu \ln(x) + (1-\mu)\ln(1-x) \end{aligned}$$

the derivative of which is

$$\begin{aligned} \frac{d}{dx} g_\mu(x) &= \frac{d}{dx}(\mu \ln(x) + (1-\mu)\ln(1-x)) \\ &= \frac{\mu}{x} + \frac{1-\mu}{-(1-x)} \\ &= \frac{\mu - x}{x - x^2} \end{aligned}$$

This implies that $g_\mu(x)$ is strictly increasing for $x < \mu$ and strictly decreasing for $x > \mu$. Hence, it must be that the *unique maximum* is reached when $x = \mu$. □

What if we have more than 2 possible outcomes, such as a set $\{0, 1, \ldots, k-1\}$ of k possible categories having distributions $\overline{\mu} = [\mu_0, \mu_1, \ldots, \mu_{k-1}]$? In this case, we can use the following scoring rule, which is a *generalization* of the Binary Logarithmic Scoring Rule.

Definition 5.6: Cross-Entropy Scoring Rule

Given a vector of distributions $\overline{\mu} = [\mu_0, \mu_1, \ldots, \mu_{k-1}]$, the **Cross-Entropy Scoring Rule** is defined as follows

$$\forall i \in [k] \quad f_i(x) = (x_0, \ldots, x_{k-1}) = \ln(x_i)$$

Clearly, the gain of this function is analogously generalized by extending the sum to all the functions $f_i(x_0, \ldots, x_{k-1})$, as shown below.

Theorem 5.11

Given a vector of distributions $\overline{\mu} = [\mu_0, \mu_1, \ldots, \mu_{k-1}]$, the function

$$g_{\overline{\mu}} = \sum_{i=0}^{k-1} \mu_i f_i(x_0, \ldots, x_{k-1}) = \sum_{i=0}^{k-1} \mu_i \ln(x_i)$$

is uniquely maximized at $\overline{x} = \overline{\mu}$.

Proof. Omitted. □

5.4 Selection processes

Suppose that we have a sequence of items and we wish to select the best one in an *online manner*. In the literature, this is known as the **Secretary Problem**, where we have to select the bast applicant out of a sequence of n applicants, and the candidates are shown UAR in an *online manner*. At any point in time, we can order the applicant we saw w.r.t. their ability. When wee see a candidate we immediately have to decide whether to hire them or not.

Suppose that we have three candidates c_1, c_2 and c_3 whose ability follows the ordering $c_2 > c_1 > c_3$. We observe that there are 3! possible arrival ordering for the three candidates. We propose the following three algorithms:

1. Pick the first candidate you see. This algorithm picks the best candidate — i.e. c_2 in our example — with probability $\frac{2}{3!} = \frac{2}{6} = \frac{1}{3}$.
2. Pick the last candidate you see. This algorithm has the same probability of the first one of picking the best candidate, i.e. $\frac{1}{3}$.
3. Pick the second candidate if they are better than the first candidate, otherwise pick the third candidate. This algorithm picks the best candidate with probability

$$\begin{aligned}
&\sum_{i=1}^{3} \Pr[c_2 \text{ is in the } i\text{-th position}] \cdot \Pr[c_2 \text{ is picked} \mid c_2 \text{ is in the } i\text{-th position}] \\
=&\frac{1}{3} \cdot 0 + \frac{1}{3} \cdot 1 + \frac{1}{3} \cdot \frac{1}{2} \\
=&\frac{1}{2}
\end{aligned}$$

This example clearly shows that *confronting* the various candidates achieves a better result — as the intuition would suggest. However, we recall that in our setting we can only check previous candidates, meaning that we could potentially *miss* the best candidate. Nonetheless, we would still get a "good enough" candidate.

To generalize this idea, we use a **threshold process**:

- we observe the first t candidates, without hiring any of them
- then, we keep observing the following candidates, hiring the first candidate that is better than all of the first t applicants
- if no such candidate is found, we stick with the last applicant seen

What is the *probability of success* of this procedure?

Lemma 5.2

The probability of success of the t-threshold process over n candidates $c_1, \ldots, c_n$ is

$$\Pr[\text{best candidate hired}] = \frac{t}{n}(H_{n-1} - H_{t-1})$$

where $H_k = \sum_{i=1}^{k} \frac{1}{i}$ is the k-th Harmonic number.

Proof. First, we observe that

$$\begin{aligned}
\Pr[\text{best candidate hired}] &= \sum_{i=1}^{n} \Pr[c_i \text{ hired and } c_i \text{ best}] \\
&= \sum_{i=1}^{n} \Pr[c_i \text{ hired} \mid c_i \text{ best}] \cdot \Pr[c_i \text{ best}] \\
&= \frac{1}{n} \sum_{i=1}^{n} \Pr[c_i \text{ hired} \mid c_i \text{ best}] \\
&= \frac{1}{n} \left(\sum_{i=1}^{t} \Pr[c_i \text{ hired} \mid c_i \text{ best}] + \sum_{i=t+1}^{n} \Pr[c_i \text{ hired} \mid c_i \text{ best}] \right) \\
&= \frac{1}{n} \left(0 + \sum_{i=t+1}^{n} \Pr[c_i \text{ hired} \mid c_i \text{ best}] \right) \\
&= \frac{1}{n} \sum_{i=t+1}^{n} \Pr[c_i \text{ hired} \mid c_i \text{ best}]
\end{aligned}$$

However, we observe that the event "c_i is hired $\mid$ c_i best" is equivalent to the event "the second best candidate among $c_1, \ldots, c_i$ is in one of the first t positions $\mid$ c_i is the best". Hence, since we want the second best candidate among $c_1, \ldots, c_i$ to fall within the first t ones the probability of this event is $\frac{t}{i-1}$ — since c_i cannot be the second best candidate,

given that it is *the best*. Therefore, we have that

$$\begin{aligned}\Pr[\text{best candidate hired}] &= \frac{1}{n}\sum_{i=t+1}^{n}\Pr[c_i \text{ hired} \mid c_i \text{ best}] \\ &= \frac{1}{n}\sum_{i=t+1}^{n}\frac{t}{i-1} \\ &= \frac{t}{n}\sum_{j=t}^{n-1}\frac{1}{j} \\ &= \frac{t}{n}(H_{n-1} - H_{t-1})\end{aligned}$$

□

Theorem 5.12

If $t = \left\lfloor \frac{n}{e} \right\rfloor$, the probability of success of the t-threshold process on n candidates $c_1, \ldots, c_n$ is at least $\frac{1}{e} - o(1)$.

Proof. First, observe that

$$H_{n-1} - H_{t-1} = \sum_{k=t}^{n-1}\frac{1}{k} \geq \sum_{k=t}^{n-1}\int_{k}^{k+1}\frac{1}{x}dx = \int_{t}^{n}\frac{1}{2}dx = \ln\left|\frac{n}{t}\right|$$

and since $t = \left\lfloor \frac{n}{e} \right\rfloor$ we have that

$$\ln\left|\frac{n}{t}\right| = \ln\left|\frac{n}{\left\lfloor \frac{n}{e} \right\rfloor}\right| \geq \ln\left|\frac{n}{\frac{n}{e}}\right| = \ln e = 1$$

Hence, by the previous lemma we get that

$$\begin{aligned}\Pr[\text{best candidate hired}] &= \frac{t}{n}(H_{n-1} - H_{t-1}) \\ &\geq \frac{t}{n}\cdot 1 \\ &\geq \frac{\left\lfloor \frac{n}{e} \right\rfloor}{n} \\ &> \frac{\frac{n}{e} - 1}{n} \\ &= \frac{1}{e} - \frac{1}{n}\end{aligned}$$

which clearly goes to $\frac{1}{e}$ as n grows to $+\infty$. □

Now, we are going to consider a variant, known as the **Prophet Inequalities problem**. Suppose that each candidate c_i has a "score" sampled independently from a known random

variable $X_i \geq 0$. The setup is identical: we still get to see the candidates in order and we stil have to select only one of them in an *online fashion*. Additionally, we get to see the distributions of the scores *a priori*.

As an example, suppose that we have two candidates c_1 and c_2 with score variables X_1 and X_2 defined as follows

$$\Pr[X_1 = 0] = \Pr[X_1 = 1] = \Pr\left[X_2 = \tfrac{1}{3}\right] = \Pr\left[X_2 = \tfrac{4}{3}\right] = \tfrac{1}{2}$$

Consider the following algorithm: if $X_1 = 0$ then we accept X_2, otherwise we accept X_1. Let A be the output of this algorithm; what is the expected value of A?

$$\begin{aligned}\mathbb{E}[A] &= \Pr[X_1 = 0]\,\mathbb{E}[X_2] + \Pr[X_1 = 1] \cdot 1 \\ &= \frac{1}{2}\left(\frac{1}{3} \cdot \frac{1}{2} + \frac{4}{3} \cdot \frac{1}{2}\right) + \frac{1}{2} \\ &= \frac{11}{12}\end{aligned}$$

For the expected value of the *best score*, instead, we have that

$$\begin{aligned}\mathbb{E}[\max(X_1, X_2)] &= \Pr\left[X_2 = \frac{4}{3}\right] \cdot \frac{4}{3} + \Pr\left[X_2 = \frac{1}{3}\right] \cdot \Pr[X_1 = 1] \cdot 1 + \Pr\left[X_2 = \frac{1}{3}\right] \cdot \Pr[X_1 = 0] \cdot \frac{1}{3} \\ &= \frac{1}{2} \cdot \frac{4}{3} + \frac{1}{2} \cdot \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{3} \\ &= 1\end{aligned}$$

Therefore, we get that

$$\mathbb{E}[A] \geq \frac{11}{12}\,\mathbb{E}[\max(X_1, X_2)]$$

Can we always be competitive over n candidates? The following theorem shows that no online algorithm can give an approximation better than $\frac{1}{2}$.

Theorem 5.13

There is no randomized online algorithm for the Prophet Inequalities problem that can expectedly return the best candidate with score greater than $\frac{1}{2}$ times the score of the optimal candidate — for scores chosen from a random distribution.

Proof. Fix a small $\varepsilon \in (0, 1]$; let X_1 and X_2 be two random score variables where, such that $\Pr[X_1 = 1] = 1$, $\Pr[X_2 = \frac{1}{\varepsilon}] = \varepsilon$ and $\Pr[X_2 = 0] = 1 - \varepsilon$.

Claim 1: Any online algorithm over X_1 and X_2 will always make a choice with expected value equal to 1.

Proof of the Claim. Fix an online algorithm, and let A be its output. We know that A can either be X_1 or X_2, and by definition it holds that $\mathbb{E}[X_1] = 1$. Moreover, we have that

$$\mathbb{E}[X_2] = \frac{1}{\varepsilon} \cdot \varepsilon + (1 - \varepsilon) \cdot 0 = 1$$

Therefore $\mathbb{E}[A] = 1$ will always hold. □

Claim 2: $\mathbb{E}[\max(X_1, X_2)] = 2 - \varepsilon$.

Proof of the Claim. By definition, we have that

$$\mathbb{E}[\max(X_1, X_2)] = \frac{1}{\varepsilon} \cdot \Pr\left[X_2 = \tfrac{1}{\varepsilon}\right] + 1 \cdot \Pr[X_2 = 0] = \frac{1}{\varepsilon} \cdot \varepsilon + 1 \cdot (1 - \varepsilon) = 2 - \varepsilon$$

□

The two claims directly conclude that for any solution A returned by any algorithm, it holds that

$$\mathbb{E}[A] = \frac{1}{2 - \varepsilon} \mathbb{E}[\max(X_1, X_2)]$$

□

The following is an algorithm that reaches this optimal bound.

Algorithm 5.9: Optimal Prophet Inequalities problem

Given some scoring variables $X_1, \ldots, X_n$, and a value τ, the algorithm returns the best candidate w.r.t. the scoring variables.

```
1: function ProphetIneq(X_1, ..., X_n, τ)
2:     for i ∈ [n] do
3:         Observe an independent sample v_i from X_i
4:         if v_i ≥ τ then
5:             return v_i
6:         end if
7:     end for
8: end function
```

Theorem 5.14

Given n scoring variables $X_1, \ldots, X_n$ for the Prophet Inequalities problem, let $\tau^* = \max(X_1, \ldots, X_n)$. Then, if $A_{\tau^*} = \textsc{ProphetIneq}(X_1, \ldots, X_n, \tau)$, it holds that

$$\mathbb{E}[A_{\tau^*}] \geq \tau^*$$

Proof. Fix a value τ, and let

$$\mu_\tau := \Pr[\exists X_i \geq \tau] = \Pr[\text{a candidate is selected}]$$

Moreover, for each x let $(x)^+$ be defined as $(x)^+ \max(0, x)$ Now, for any τ it holds that

$$\begin{aligned}
\mathbb{E}[A_\tau] &= \mu_\tau \cdot \tau + \sum_{i=1}^{n} \Pr[\max(X_1, \ldots, x_{i-1}) < t] \cdot \mathbb{E}[(X_i - \tau)^+] \\
&\geq \mu_\tau \cdot \tau + \sum_{i=1}^{n} \Pr[\max(X_{\mathbb{1}}, \ldots, x_n) < t] \cdot \mathbb{E}[(X_i - \tau)^+] \\
&= \mu_\tau \cdot \tau + \sum_{i=1}^{n} (1 - \mu_\tau) \, \mathbb{E}[(X_i - \tau)^+] \\
&= \mu_\tau \cdot \tau + (1 - \mu_\tau) \, \mathbb{E}\left[\sum_{i=1}^{n} (X_i - \tau)^+\right] \\
&\geq \mu_\tau \cdot \tau + (1 - \mu_\tau) \, \mathbb{E}\left[\max_{i \in [n]} (X_i - \tau)^+\right] \\
&= \mu_\tau \cdot \tau + (1 - \mu_\tau) \, \mathbb{E}\left[\max_{i \in [n]} \max(0, X_i - \tau)\right] \\
&= \mu_\tau \cdot \tau + (1 - \mu_\tau) \, \mathbb{E}\left[\max(0, \max_{i \in [n]}(X_i) - \tau)\right] \\
&\geq \mu_\tau \cdot \tau + (1 - \mu_\tau) \, \mathbb{E}\left[\max_{i \in [n]} (X_i) - \tau\right] \\
&= \mu_\tau \cdot \tau + (1 - \mu_\tau) \left(\mathbb{E}\left[\max_{i \in [n]} (X_i)\right] - \tau\right)
\end{aligned}$$

Now, given that $\tau^* = \frac{1}{2} \, \mathbb{E}\left[\max_{i \in [n]} (X_i)\right]$, we conclude that

$$\begin{aligned}
\mathbb{E}[A_{\tau^*}] &\geq \mu_{\tau^*} \cdot \tau^* + (1 - \mu_{\tau^*}) \left(\mathbb{E}\left[\max_{i \in [n]} (X_i)\right] - \tau^*\right) \\
&= \mu_{\tau^*} \cdot \tau^* + (1 - \mu_{\tau^*})(2\tau^* - \tau^*) \\
&= \tau^*
\end{aligned}$$

□

6

Solved Exercises

Problem 6.1

Does there exist a polytime algorithm that, given a graph $G = (V, E)$, returns `True` if and only if the maximum cut of G has size $|E|$?

Solution. First, consider the following result.

Claim: G admits a cut of size $|E(G)|$ if and only if G is bipartite.

Proof of the Claim. Let S be a set such that $\left|\text{cut}(S, \overline{S})\right| = |E(G)|$, and consider the two sets $(S, \overline{S})$; then, by definition it holds that

$$\text{cut}(S, \overline{S}) = \left\{uv \mid u \in S \wedge v \in \overline{S}\right\}$$

which implies that $(S, \overline{S})$ is a bipartition of G.

Vice versa, consider a bipartite graph G having a bipartition (A, B); then, we observe that $B = V(G) - A = \overline{A}$, therefore

$$|\text{cut}(A, B)| = \left|\text{cut}(A, \overline{A})\right| = |E(G)|$$

□

Moreover, since G is bipartite if and only if it is 2-colorable, we get that an algorithm which decides if G can be 2-colored suffices to solve the problem.

```
1: function DFS-2-COLORING(G, u, col, c)
2:     if col[u] == c then
3:         return False
4:     end if
5:     col[u] = c
6:     for v ∈ N(v) do
```

```
        if DFS-2-COLORING(G, v, col, 1 − c) then
            return False
        end if
    end for
    return True
end function
function 2-COLORING(G)
    col = [−1, . . . , −1]
    for u ∈ V(G) do
        if col[u] == −1 then
            if DFS-2-COLORING(G, u, col, 0) then
                return False
            end if
        end if
    end for
    return True
end function
```

□

Problem 6.2

Let $G = (V, E)$ be a graph having a vertex cover of size k. Show that the vertices of G can be colored with $k + 1$ colors in such a way that, for each $uv \in E$, u and v have different colors.

Solution. Let V be a vertex cover of size k, and consider the following coloring c of G: assign a different color to each vertex of V, and assign an additional color to all the vertices inside $V(G) - V$.

Claim: c is a proper coloring of G.

Proof of the Claim. Suppose that c is not proper, i.e. there exists two *adjacent* vertices $x, y \in V(G)$ such that $c(x) = c(y)$. By definition of c clearly x and y cannot be both inside V otherwise they would have been assigned different colors; moreover, it cannot happen that one of them is inside V and the other is not, because the vertices in $V(G) - V$ are assigned a color different from all the colors used for V. Hence, it must be that $x, y \in V(G) - V$, but since $x \sim y$ this implies that the edge xy is not covered by the vertex cover V ↯. □

The claim concludes that $|V| + 1 = k + 1$ colors clearly suffice to properly color G. □

Problem 6.3

Show that, for each $\varepsilon > 0$, there exists a graph G such that Algorithm 2.2 returns a subgraph of G having density not larger than $\frac{1}{2} + \varepsilon$ times the optimal — i.e. the approximation bound of $\frac{1}{2}$ is optimal.

Solution. Let G_t be a graph constructed from a $K_{t,2^t}$ and 2^t cliques K_{t+2}.

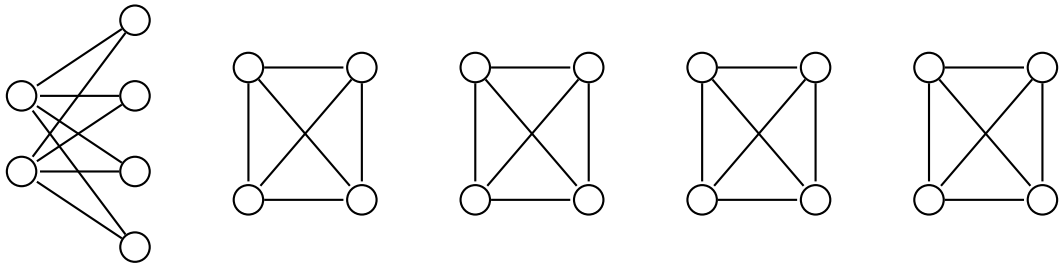


Figure 6.1: An example of the graph G_2, consisting of a $K_{2,4}$ and four K_4's.

How does the algorithm compute when fed with G_t as input? Since the degree of all the vertices of the cliques is exactly $t+1$, and the complete bipartite graph $K_{t,2^t}$ has vertices having degree t — namely, in its biggest partition — the algorithm will compute as follows:

- initially, the algorithm will remove all the vertices of the biggest partition of $K_{t,2^t}$ one by one, since they all have degree t, until only the K_{t+2}-cliques remain
- then, the algorithm will proceed to remove all the vertices of the cliques, removing one K_{t+2}-clique at a time, until no vertices remain

This implies that the possible graphs that the algorithm sees are the following:

- G_t, i.e. the graph in input
- for each iteration i let X_i be the set of vertices of the biggest partition of the complete bipartite graph that have been removed; then, we have the graph $G_t - X_i$, i.e. the remaining complete bipartite graph along with the 2^t K_{t+2}-cliques
- $G - K_{t,2^t}$, i.e. the 2^t K_{t+2}-cliques — we observe that from now on, the algorithm will always see a K_s along with some amount of K_{t+2}-cliques with $s \leq t+2$, which does not increase the density of the total graph, so we may ignore all of these cases

This concludes that we need to check the following densities in order to establish which result the algorithm returns:

- $\rho(G_t) = \dfrac{t \cdot 2^t + 2^t \cdot \binom{t+2}{2}}{t + 2^t + 2^t \cdot (t+2)} \xrightarrow{+\infty} \dfrac{t}{2}$
- $\rho(G_t - X_i) = \dfrac{t \cdot 2^t + 2^t \cdot \binom{t+2}{2} - t \cdot i}{t + 2^t + 2^t \cdot (t+2) - i} \xrightarrow{+\infty} \dfrac{t}{2}$ — we observe that this can be proven through the Squeeze theorem, in fact

$$\frac{t}{2} = \lim_{t \to +\infty} \frac{2^{t-1}(t+2)(t+1)}{2^t(t+3)} = \lim_{t \to +\infty} \rho(G_t - X_{2^t}) \leq \lim_{t \to +\infty} \rho(G - X_i) \leq \lim_{t \to +\infty} \rho(G_t) = \frac{t}{2}$$

- $\rho(G_t - K_{t,2^t}) = \dfrac{2^t \cdot \binom{t+2}{2}}{2^t \cdot (t+2)} = \dfrac{t+1}{2} \xrightarrow{+\infty} \dfrac{t}{2}$

This implies that for any set $S = \textsc{Charikar}(G_t)$ that the algorithm may yield, it holds that

$$\lim_{t \to +\infty} \rho(G_t[S]) = \frac{t}{2}$$

However, it turns out that the densest subgraph of G_t is actually $K_{t,2^t}$, in fact

$$\rho(K_{t,2^t}) = \frac{t \cdot 2^t}{t + 2^t} \xrightarrow{+\infty} t$$

and the algorithm *does not encounter* $K_{t,2^t}$ as we previously mentioned. Hence, we have that

$$\lim_{t \to +\infty} \frac{\rho(G_t[S])}{\rho(K_{t,2^t})} = \frac{\frac{t}{2}}{t} = \frac{1}{2}$$

or, in other words, it holds that

$$\forall \varepsilon > 0 \quad \exists t \in \mathbb{N} \quad \rho(G_t[S]) < (\tfrac{1}{2} + \varepsilon)\rho(K_{t,2^t}) \implies \mathrm{ALG}_{\mathrm{DS}}(G_t) < (\tfrac{1}{2} + \varepsilon) \cdot \mathrm{OPT}_{\mathrm{DS}}(G_t)$$

□

Problem 6.4

Let d be a cut metric. Prove that d satisfies the triangle inequality.

Solution. Let d be defined over $T_1, \ldots, T_k$ and $\lambda_1, \ldots, \lambda_k > 0$, i.e.

$$d(x, y) := \sum_{i \in [k]} \lambda_i d_{T_i}(x, y)$$

Fix x, y and z; we observe that

$$d(x, y) + d(y, z) = \sum_{i \in [k]} \lambda_i d_{T_i}(x, y) + \sum_{i \in [k]} \lambda_i d_{T_i}(y, z) = \sum_{i \in [k]} \lambda_i \left(d_{T_i}(x, y) + d_{T_i}(y, z) \right)$$

and since $d_{T_1}, \ldots, d_{T_k}$ are elementary cut metrics, they must satisfy the triangle inequality, therefore

$$\sum_{i \in [k]} \lambda_i \left(d_{T_i}(x, y) + d_{T_i}(y, z) \right) \geq \sum_{i \in [k]} \lambda_i d_{T_i}(x, z) = d(x, z)$$

which proves that d indeed satisfies the triangle inequality. □

Problem 6.5

Let $G = (V, E)$ be the graph with node set $V = [2n]$ for $n \in \mathbb{N}$, and edge set

$$E = \{\{i, i+1\} \mid 1 \leq i \leq 2n - 1\} \cup \{\{2n, 1\}\}$$

meaning that G is a cycle on $2n$ nodes. Let d_G be the shortest path metric on G. Prove that d_G can be embedded isometrically into ℓ_1.

Solution. First, we observe that the shortest path metric on a $2n$-long cycle graph is defined as follows:

$$\forall j,k \in [2n] \quad d_G(j,k) = \begin{cases} |k-j| & |k-j| \leq n \\ 2n - |k-j| & |k-j| > n \end{cases}$$

Moreover, we observe that ℓ_1 defined on two binary vectors $x, y \in \{0,1\}^h$ is equivalent to their Hamming distance, i.e.

$$\ell_1(x,y) = d_H(x,y) = |\{i : x_i \neq y_i\}|$$

and in particular

$$\forall j,k \in [2n] \quad \ell_1(f(j), f(k)) = d_H(f(j), f(k))$$

Now, consider the function $f : V(C_{2n}) \to \mathbb{R}^n$ defined as follows:

$$\forall i \in [2n] \quad f(i) = \begin{cases} 0^{n-i+1}1^{i-1} & i \leq n \\ 1^{2n-i+1}0^{i-1-n} & i > n \end{cases}$$

where the notation 0^a1^b stands for the vector $(\underbrace{0,\ldots,0}_{a \text{ values}}, \underbrace{1,\ldots,1}_{b \text{ values}}) \in \mathbb{R}^{a+b}$. At first, this definition of f may seem counterintuitive, but if we consider each possible $f(i)$ the following nice pattern of "sliding window" emerges:

i	$f(i)$
1	0000
2	0001
3	0011
4	0111
5	1111
6	1110
7	1100
8	1000

Finally, fix $j,k \in [2n]$, and without loss of generality suppose that $j < k$ – in particular, this implies that $|k-j| = k-j$. We have some cases to handle.

- $j,k \leq n$; therefore, we have that
  - $j,k \leq n \implies k-j \leq n \implies d_G(j,k) = k-j$
  - $j < k \implies n-j+1 > n-k+1 \wedge j-1 < k-1$

  and thus, to evaluate the Hamming distance between $f(j)$ and $f(k)$ we need to consider the "middle" section in which the two binary vectors may differ

$$\begin{aligned} \ell_1(f(j), f(k)) &= d_H(0^{n-j+1}1^{j-1}, 0^{n-k+1}1^{k-1}) \\ &= [n-(j-1)] - [(n-k+1)+1] + 1 \\ &= [n-j+1] - [n-k+2] + 1 \\ &= k-j \\ &= d_G(j,k) \end{aligned}$$

- $j, k > n$; therefore, we have that
  - $j, k > n \implies k - j < n \implies d_G(j,k) = k - j$
  - $j < k \implies 2n - j + 1 > 2n - k + 1 \wedge j - 1 - n < k - 1 - n$

  hence again, we just need to consider the "middle" section in this case too

$$\begin{aligned}\ell_1(f(j), f(k)) &= d_H(1^{2n-j+1}0^{j-1-n}, 1^{2n-k+1}, 0^{k-1-n}) \\ &= [n - (k - 1 - n)] - [(2n - j + 1) + 1] + 1 \\ &= [n - k + 1 + n] - [2n - j + 2] + 1 \\ &= k - j \\ &= d_G(j,k)\end{aligned}$$

- $j \leq n$ and $k < n$. We have two subcases
  1. $k - j \leq n$; therefore, we have that
     - $k - j \leq n \implies d_G(j,k) = k - j$
     - $k - j \leq n \implies n - j + 1 \leq 2n - k + 1 \wedge j - 1 \geq k - n - 1$

     and thus, in this case to evaluate the Hamming distance between $f(j)$ and $f(k)$ we need to consider the sections in which the two vectors can be 0 — since they can only coincide when they are both 1

$$\begin{aligned}\ell_1(f(j), f(k)) &= d_H(0^{n-j+1}1^{j-1}, 1^{2n-k+1}0^{k-1-n}) \\ &= (n - j + 1) + (k - 1 - n) \\ &= k - j \\ &= d_G(j,k)\end{aligned}$$

  2. $k - j > n$; therefore, we have that
     - $k - j > n \implies d_G(j,k) = 2n - k + j$
     - $k - j > n \implies n - j + 1 > 2n - k + 1 \wedge j - 1 < k - n - 1$

     and thus, in this case to evaluate the Hamming distance between $f(j)$ and $f(k)$ instead we need to consider the sections in which the two vectors can be 1 — since they can only coincide when they are both 0

$$\begin{aligned}\ell_1(f(j), f(k)) &= d_H(0^{n-j+1}1^{j-1}, 1^{2n-k+1}0^{k-1-n}) \\ &= (j - 1) + (2n - k + 1) \\ &= 2n - j + k \\ &= d_G(j,k)\end{aligned}$$

□

Problem 6.6

Prove that, for any $n \in \mathbb{N}$, if the function $f : \mathcal{P}([n]) \to \mathbb{R}$ has diminishing returns, it is submodular.

Solution. First, we are going to prove a *stronger* version of the diminishing return property, by extending it to *subsets* of elements — instead of single elements. The notation is a natural extension of the one we used so far.

Claim: $\forall A \subseteq B \subseteq [n], C \subseteq [n] - B \quad \Delta_f(C \mid A) \geq \Delta_f(C \mid B)$.

Proof of the Claim. Fix three sets $A, B, C \subseteq [n]$ such that $A \subseteq B \subseteq [n]$ and $C \subseteq [n] - B$. Moreover, let $C := \{c_1, \ldots, c_k\}$, and for each $i \in [k]$ let $C_i := \{c_1, \ldots, c_i\}$. We proceed by induction on i.

For the base case, we have that $C_1 = \{c_1\}$, hence by the diminishing return property we have that

$$\Delta_f(C_1 \mid A) = \Delta_f(c_1 \mid A) \geq \Delta_f(c_1 \mid B) = \Delta_f(C_1 \mid B)$$

Now, assume inductively that the property holds for C_{i-1}; then, we get that

$$\begin{aligned}
\Delta_f(C_i \mid A) &= f(C_i \cup A) - f(A) \\
&= f(\{c_1, \ldots, c_i\} \cup A) - f(A) \\
&= f(\{c_1, \ldots, c_{i-1}\} \cup \{c_i\} \cup A) - f(A) \\
&= f(C_{i-1} \cup \{c_i\} \cup A) - f(A) \\
&= f(C_{i-1} \cup \{c_i\} \cup A) - f(A) + f(C_{i-1} \cup A) - f(C_{i-1} \cup A) \\
&= \Delta_f(C_{i-1} \mid A) + \Delta_f(c_i \mid C_{i-1} \cup A)
\end{aligned}$$

Now, by applying induction on the first term, and the diminishing return property on the second term, we get that

$$\begin{aligned}
\Delta_f(C_i \mid A) &= \Delta_f(C_{i-1} \mid A) + \Delta_f(c_i \mid C_{i-1} \cup A) \\
&\geq \Delta_f(C_{i-1} \mid B) + \Delta_f(c_i \mid C_{i-1} \cup B) \\
&= f(C_{i-1} \cup B) - f(B) + f(\{c_i\} \cup C_{i-1} \cup B) - f(C_{i-1} \cup B) \\
&= f(C_i \cup B) - f(B) \\
&= \Delta_f(C_i \mid B)
\end{aligned}$$

□

Now, fix two sets $S, T \subseteq [n]$; note that it always holds that $(S \cap T) \subseteq T \subseteq [n]$ and $(S - T) \subseteq [n] - T$ for any choice of S and T. Finally, through some algebraic manipulation, we get that

$$\begin{aligned}
&\Delta_f(S - T \mid S \cap T) \geq \Delta_f(S - T \mid T) \\
\iff &\Delta_f(S - (S \cap T) \mid S \cap T) \geq \Delta_f(S - ((S \cap T) \cup T) \mid (S \cap T) \cup T) \\
\iff &f((S \cap T) \cup S) - f(S \cap T) \geq f((S \cap T) \cup S \cup T) - f((S \cap T) \cup T) \\
\iff &f((S \cap T) \cup S) + f((S \cap T) \cup T) \geq f((S \cap T) \cup S \cup T) + f(S \cap T) \\
\iff &f(S) + f(T) \geq f(S \cup T) + f(S \cap T)
\end{aligned}$$

which proves that f is indeed submodular. □

Problem 6.7

Given $n \in \mathbb{N}$, suppose that the function $f : \mathcal{P}([n]) \to \mathbb{R}$ satisfies the following property:

$$\forall \{i,j\} \in \binom{[n]}{2}, S \subseteq [n] - \{i,j\} \quad f(S \cup \{i\}) + f(S \cup \{j\}) \geq f(S \cup \{i,j\}) + f(S)$$

Prove that f has the diminishing returns property.

Solution. Fix two sets $A \subseteq B \subseteq [n]$, and an element $x \in [n] - B$. Let $B - A = \{b_1, \ldots, b_m\}$ and for each $k \in [m]$ let $B_k := \{b_1, \ldots, b_k\}$. We claim the following.

Claim: For any $k \in [m]$ it holds that $\Delta_f(x \mid A) \geq \Delta_f(x \mid A \cup B_k)$.

Proof of the Claim. We proceed by induction on k; consider $b_1 \in B - A$ — and in particular $b_1 \neq x$; then clearly $A \subseteq [n] - \{x, b_1\}$ which means that we can use the property in the hypothesis as follows

$$\begin{aligned} & f(A \cup \{x\}) + f(A \cup \{b_1\}) \geq f(A \cup \{x, b_1\}) + f(A) \\ \iff & f(A \cup \{x\}) - f(A) \geq f(A \cup \{b_1\} \cup \{x\}) - f(A \cup \{b_1\}) \\ \iff & \Delta_f(x \mid A) \geq \Delta_f(x \mid A \cup \{b_1\}) = \Delta_f(x \mid A \cup B_1) \end{aligned}$$

Now, assume inductively that for $1 < k - 1 \leq m$ it holds that $\Delta_f(x \mid A) \geq \Delta_f(x \mid B_{k-1})$, and consider an element $b_k \in B - (A \cup B_{k-1})$ — and in particular $b_k \neq x$. Then, clearly $A \cup B_{k-1} \subseteq [n] - \{x, b_k\}$ which means that we can use the property in the hypothesis as follows

$$\begin{aligned} & f(A \cup B_{k-1} \cup \{x\}) + f(A \cup B_{k-1} \cup \{b_k\}) \geq f(A \cup B_{k-1} \cup \{x, b_k\}) + f(A \cup B_{k-1}) \\ \iff & f(A \cup B_{k-1} \cup \{x\}) + f(A \cup B_k) \geq f(A \cup B_k \cup \{x\}) + f(A \cup B_{k-1}) \\ \iff & f(A \cup B_{k-1} \cup \{x\}) - f(A \cup B_{k-1}) \geq f(A \cup B_k \cup \{x\}) - f(A \cup B_k) \\ \iff & \Delta_f(x \mid A \cup B_{k-1}) \geq \Delta_f(x \mid A \cup B_k) \end{aligned}$$

and by inductive hypothesis we get that

$$\Delta_f(x \mid A) \geq \Delta_f(x \mid A \cup B_{k-1}) \geq \Delta_f(x \mid A \cup B_k)$$

□

In particular, by the claim we have that for $k = m$ it holds that

$$\Delta_f(x \mid A) \geq \Delta_f(x \mid A \cup B_m) = \Delta_f(x \mid A \cup B_{|B-A|}) = \Delta_f(x \mid A \cup (B - A)) = \Delta_f(x \mid B)$$

which proves that f has the diminishing returns property. □

Problem 6.8

Let $f : \mathcal{P}([n]) \to \mathbb{R}$ be a modular function such that $f(\varnothing) = 0$.

1. Prove that the full description of f (i.e. the value of $f(S)$ for any $S \subseteq [n]$) can be learned by querying the function f on $O(n)$ sets.
2. Let G be a graph with $V(G) = [n]$ and $E(G) \subseteq \binom{V(G)}{2}$. Suppose that our algorithm is able to query f only on the edges of G, i.e. we can query $f(\{i, j\})$ if and only if $\{i, j\} \in E(G)$. Can the algorithm learn the full description of f if G is a complete graph? What if G is a tree? Give a general condition for the learnability of f.

Solution.

1. A function $f : \mathcal{P}([n]) \to \mathbb{R}$ is said to be *learnable* if and only if for any input $S \subseteq [n]$ it holds that the value of $f(S)$ can be known. We claim the following.

   Claim: A modular function f is learnable if and only if $f(\varnothing), f(1), \ldots, f(n)$ are known.

   Proof of the Claim. The direct implication trivially follows from the definition of learnability, so we only need to show the converse implication. We proceed by induction on the length of the input $|S| = k$.

   When $|S| = 1$, we already know the values of $f(S)$ by hypothesis, so the base case trivially holds. Now, assume that the statement holds for $k - 1$ items, and consider a set S of k items. Then, by modularity of f, we know that

   $$f(S) = f(\{x_1, \ldots, x_k\}) = f(\{x_1, \ldots, x_{k-1}\}) + f(\{x_k\}) - f(\varnothing)$$

   Hence, we observe that $f(\{x_1, \ldots, x_{k-1}\})$ can be learned by inductive hypothesis, and since both $f(\{x_k\})$ and $f(\varnothing)$ are known by hypothesis, the claim holds. □

2. We observe that in the previous exercise we did not use the fact that $f(\varnothing) = 0$, but we are going to leverage this property now. Consider a modular function f that can be queried as described in the statement; through the claim of the previous exercise we proved that f is learnable if and only if $f(\varnothing), f(1), \ldots, f(n)$ are learnable. Thus, since $f(\varnothing)$ is given, we need to determine a general condition which allows us to find the values for $f(1), \ldots, f(n)$ given only $f(\{i, j\})$ for each $\{i, j\} \in E(G)$.

   First, by modularity of f, we observe that

   $$\forall \{i, j\} \in E(G) \quad f(\{i, j\}) = f(\{i\}) + f(\{j\}) - f(\varnothing) f(\{i\}) + f(\{j\})$$

   since $f(\varnothing) = 0$. This condition is sufficient to construct a *system of equations*. For instance, consider the graph K_3 with the vertices labeled $V(K_3) = \{1, 2, 3\}$: then, we construct the following:

   $$\begin{cases} f(\{1\}) + f(\{2\}) = f(\{1, 2\}) \\ f(\{2\}) + f(\{3\}) = f(\{2, 3\}) \\ f(\{1\}) + f(\{3\}) = f(\{1, 3\}) \end{cases}$$

We observe that this system can be rewritten through matrices as follows

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} f(\{1\}) \\ f(\{2\}) \\ f(\{3\}) \end{bmatrix} = \begin{bmatrix} f(\{1,2\}) \\ f(\{2,3\}) \\ f(\{1,3\}) \end{bmatrix}$$

Can we generalize this concept? Fix an ordering $e_1, \ldots, e_m$ of $E(G)$, and for each subgraph $H \subseteq G$ let $A^H \in \mathbb{R}^{m \times n}$ be a coefficient matrix such that its k-th row is defined as follows

$$A^H_k(i) = \begin{cases} 1 & i \in e_k \\ 0 & \text{otherwise} \end{cases}$$

Similarly, let $b \in \mathbb{R}^m$ be a vector defined as $b = [f(e_1) \ \cdots \ f(e_k)]^T$, and finally let $x = [x_1 \ \cdots \ x_n]^T$ for each $x_i = f(\{i\})$. It is clear that we can learn f if and only if the system $A^G x = b$ has a unique solution.

Claim: For each component H_i of G, it holds that $A^{H_i} x = b$ has at least one solution if and only if H_i has at least one cycle.

Proof of the Claim. We know that $A^{H_i} x = b$ has aat least one solution if and only if if has at least as many linearly independent rows as linearly independent columnds, which is equal to $|V(H_i)|$ by construction of A^{H_i}. Moreover, each non-empty row of A^{H_i} is already linearly independent from the other ones by construction.

Therefore, we have that $A^{H_i} x = b$ has at least one solution if and only if it has at least $|V(H_i)|$, which can happen if and only if it has at least $|V(H_i)|$ by construction, which implies that H_i has at least one cycle. □

Let S_G be the solution space of $A^G x = b$. Similarly, for each $i \in [t]$ let S_{H_i} be the solution space of $A^{H_i} x = b$. Clearly, it holds that $S_G = S_{H_1} \cap \ldots \cap S_{H_t}$. Hence, by the claim above we conclude that S_G has at least one solution if and only if each component of G has at least one cycle. Moreover, by modularity of f, we know that there can be at most one such solution. Therefore, it holds that f is learnable if and only if every component of G has at least one cycle.

In particular, this also concludes that f can be learned when G is a complete graph with at least 3 vertices, but not then G is a tree.

□

Problem 6.9

Consider the LP relaxation for Set Cover. Suppose that each element in the instance is part of at most t sets. Give a rounding algorithm to the LP that returns a Set Cover solution whose cost is not larger than t times the optimal cost.

Solution. We observe that this exercise is very similar to the proof of Theorem 2.1.

Consider an instance $(\mathcal{U}, C)$ of set cover such that each element $i \in \mathcal{U}$ is in at most t sets. We observe the following.

Claim: For any feasible solution $\{x_j\}_{j\in[m]}$ of the LP relaxation, and for any $i \in [n]$, if $i \in S_{j_1}, \ldots, S_{j_k}$ then there exists $j \in \{j_1, \ldots, j_k\}$ such that $x_j \geq \frac{1}{t}$.

Proof of the Claim. By way of contradiction, suppose that for any $j \in \{j_1, \ldots, j_k\}$ it holds that $x_j < \frac{1}{t}$. However, since $k \leq t$ by the hypothesis on the instance of SC we chose, it holds that

$$\begin{aligned}\sum_{\substack{j\in[m]:\\ i\in S_j}} x_j &= \sum_{h=1}^{k} x_{j_h} \\ &= x_{j_1} + \ldots + x_{j_k} \\ &< k \cdot \frac{1}{t} \\ &\leq t \cdot \frac{1}{t} \\ &= 1\end{aligned}$$

implying that the solution $\{x_j\}_{j\in[m]}$ would violate the constraint of the LP, i.e. such solution would not be feasible $\lightning$. □

Then, consider an optimal solution $\{\overline{x}_j\}_{j\in[m]}$ to the LP relaxation of SC over such instance, and consider the following set

$$\left|\overline{S}\right| = \left\{S_j \in C \mid \overline{x}_j \geq \tfrac{1}{t}\right\}$$

The previous claim immediately proves that $\overline{S}$ is a feasible solution to SC: in fact, by the claim it holds that for each element $i \in [n]$ there exists a set S_j that contains i such that $x_j \geq \frac{1}{t}$, i.e. i will be covered by at least one set.

Finally, to prove that this is a t-approximation of SC, consider an optimal solution $\{x_j^*\}_{j\in[m]}$ to the IP for SC, and construct

$$S^* = \left\{S_j \in C \mid x_j^* = 1\right\}$$

Then, we get that

$$\begin{aligned}\left|\overline{S}\right| &= \sum_{S_j\in\overline{S}} 1 \\ &\leq \sum_{S_j\in\overline{S}} t\overline{x}_j \qquad (S_j \in \overline{S} \implies \overline{x}_j \geq \tfrac{1}{t} \iff t\overline{x}_j \geq 1) \\ &= t \sum_{S_j\in\overline{S}} t\overline{x}_j \\ &\leq t \sum_{S_j\in C} \overline{x}_j \qquad (\overline{S} \subseteq C \wedge \overline{x}_j \geq 0) \\ &\leq t \sum_{S_j\in C} x_j^* \\ &= t\,|S^*|\end{aligned}$$

where the last inequality comes from the fact that the constraints of the LP are *weaker* than the ones of the IP. □

Problem 6.10

Consider the following algorithmic problem. A delivery company has access to a set F of possible stocking facilities, and to a set of consumers C. Each consumer has to be served by one facility, and the company gains $G[c][f]$€ if it serves consumer $c \in C$ with facility $f \in F$.

If the company rents the set of facilities $S \subseteq F$, then the company will serve the generic client $c \in C$ with a facility $f \in S$ that maximizes $G[c][f]$. In particular, if the company rents the facilities of S, then its gain is $\sum_{c \in C} \max_{f \in S} G[c][f]$.

Suppose that the company can rent at most m facilites. Give an algorithm for approximately maximizing the gain of the company.

Solution. We observe that this exercise is very similar to the proof of Proposition 4.2.

Let $g(S) = \sum_{c \in C} \max_{f \in S} G[c][f]$ be the gain of S, and for each customer $c \in C$ construct the following subfunction

$$\forall c \in C \quad g_c(S) = \begin{cases} \max_{f \in S} G[c][f] & S \neq \varnothing \\ 0 & \text{otherwise} \end{cases}$$

This directly implies that

$$g(S) = \sum_{c \in C} g_c(S)$$

Claim: For all $c \in C$, the subfunction g_c is non-negative, monotone increasing and submodular.

Proof of the Claim. First, fix a customer $c \in C$, and consider the subfunction g_c.

Since the values inside the matrix G are €'s, g_c is trivially non-negative.

Now, fix two sets $S, T \subseteq F$ of facilities such that $S \subseteq T$: clearly, since S is contained inside T, and G is non-negative, it holds that

$$\max_{f \in S} G[c][f] \leq \max_{f \in T} G[c][f]$$

This proves that g_c is monotone increasing.

Lastly, fix two sets $S, T \subseteq F$ of facilities; to prove that g_c is also submodular, we have three cases to handle.

- If $S \cap T = \varnothing$, let $M_S := g_c(S)$ and $M_T := g_c(T)$; we observe that clearly $g_c(S \cup T) =$

$\max\{M_S, M_T\}$, therefore it holds that

$$\begin{aligned} g_c(S) + g_c(T) &= M_S + M_T \\ &\geq \max\{M_S, M_T\} + 0 \\ &= g_c(S \cup T) + f(\varnothing) \\ &= g_c(S \cup T) + f(S \cap T) \end{aligned}$$

- If $S \subseteq T$, let $M_S := g_c(S)$ and $M_T := g_c(T)$; we have two cases.
  - If $M_S = M_T$, then $g_c(S \cup T) = M_S = M_T$, therefore

$$\begin{aligned} g_c(S) + g_c(T) &= M_S + M_T \\ &= M_S + M_S \\ &= g_c(S \cup T) + g_c(S) \\ &= g_c(S \cup T) + g_c(S \cap T) \end{aligned}$$

  - If $M_S \neq M_T$, then $M_T = g_c(M - T)$ — otherwise M_T would violate the maximality of M_S — which in turn implies that $M_T > M_S$ — otherwise M_S would have been a maximum for T as well. Therefore, it holds that $g_c(S \cup T) = M_T = g_c(T)$, hence

$$\begin{aligned} g_c(S) + g_c(T) &= M_S + M_T \\ &= g_c(S) + g_c(T) \\ &= g_c(S \cap T) + g_c(S \cup T) \end{aligned}$$

- If $S - T, T - S \neq \varnothing$, let $M_S := g_c(S)$, $M_T := g_c(T)$ and $M_{S \cap T} = g_c(S \cap T)$; we have three cases.
  - If $M_S = M_T = M_{S \cap T}$ then $g_c(S \cup T) = M_S = M_T = M_{S \cap T}$ therefore

$$\begin{aligned} g_c(S) + g_c(T) &= M_S + M_T \\ &= M_{S \cap T} + M_{S \cap T} \\ &= g_c(S \cup T) + g_c(S \cap T) \end{aligned}$$

  - If $M_S = M_{S \cap T} \neq M_T$ — the case in which $M_T = M_{S \cap T} \neq M_S$ is analogous — then $M_T > M_S$ otherwise M_S would have been maximum for T as well. Therefore, it holds that $g_c(S \cup T) = M_T = g_c(T)$, hence

$$\begin{aligned} g_c(S) + g_c(T) &= M_S + M_T \\ &= M_{S \cap T} + g_c(S \cup T) \\ &= g_c(S \cap T) + g_c(S \cup T) \end{aligned}$$

  - If M_S, M_T and $M_{S \cap T}$ are all distinct, then $M_S, M_T > M_{S \cap T}$ otherwise $M_{S \cap T}$ would violate the maximality of M_S and M_T for S and T respectively. Therefore, it holds that $g_c(S \cup T) = \max\{M_S, M_T\} > M_{S \cap T}$, hence

$$\begin{aligned} g_c(S) + g_c(T) &= M_S + M_T \\ &\geq \max\{M_S, M_T\} + M_{S \cap T} \\ &= g_c(S \cup T) + g_c(S \cap T) \end{aligned}$$

□

By the claim, it trivially follows that g is both non-negative and monotone increasing. Finally, fix two sets $S, T \subseteq F$ of facilities; show that it is also submodular, we can leverage the submodularity of g_c for each $c \in C$ as follows:

$$\begin{aligned} g(S) + g(T) &= \sum_{c \in C} g_c(S) + \sum_{c \in C} g_c(T) \\ &= \sum_{c \in C} (g_c(S) + g_c(T)) \\ &\geq \sum_{c \in C} (g_c(S \cup T) + g_c(S \cap T)) \\ &= \sum_{c \in C} g_c(S \cup T) + \sum_{c \in C} g_c(S \cap T) \\ &= g(S \cup T) + g(S \cap T) \end{aligned}$$

Therefore, since the gain of S is non-negative, monotone increasing and submodular, for any m maximum number of facilities that the company can rent we can employ Algorithm 4.2 to return a set of m facilities with an approximation ratio of $1 - \frac{1}{e}$ by Theorem 4.3. □

Algorithm 6.1: The f-Generalized W.M. Expert model

Given a set M of n trained experts, the algorithm yields predictions over time T.

---

function WEIGHTEDMAJORITYEXPERTMODEL(M, T, f)
  for $i \in [n]$ **do**
    $w_i := 0$
  end for
  for $t \in [T]$ **do**
    Read $y_t = [y_{1,t} \ \cdots \ y_{n,t}]$
    $A_t := \sum_{\substack{i \in [n]: \\ y_{i,t}=1}} f(w_i)$
    $B_t := \sum_{\substack{i \in [n]: \\ y_{i,t}=0}} f(w_i)$
    if $A_t \geq B_t$ **then**
      $z_t := 1$
    else
      $z_t := 0$
    end if
    Read the outcome $x_t \in \{0, 1\}$
    if $x_t \neq z_t$ **then**
      for $i \in [n]$ **do**
        if $y_{i,t} \neq x_t$ **then**
          $w_i = w_i + 1$
        end if
      end for
    end if
  end for
end function

Problem 6.11

Consider the algorithm above: this is the f-Generalized Weighted Majority Expert model, which is a generalized version of Algorithm 5.7. In fact, if $f(w_i) = 2^{-w_i}$ then the two algorithms are identical, and it guarantees a number of mistakes not larger than $O(m^* + \log n)$ where m^* is the number of mistakes of the best expert, and n is the number of experts — as we proved in Theorem 5.6.

Prove that, if $f(w_i) = \frac{w_i}{w_i+1}$, then the number of mistakes of the f-Generalized Weighted Majority algorithm is $\Omega(n)$ even in cases where $m^* = 0$.

Moreover, prove that if $f(w_i) = (w_i + 1)^{-\alpha}$ for a generic constant $\alpha > 0$, then there are instances with $m^* = 0$ where the f-Generalized Weighted Majority algorithm makes a number of mistakes polynomial in n.

Solution. The first approach that we may try is the one used in the proof of Theorem 5.6

itself. In fact, in that version of the algorithm we were able to prove that $W^{(t+1)} \leq \frac{3}{4}W^{(t)}$, however the value $\frac{3}{4}$ directly derives from the *updating rule* of the weights. In particular, since the weights of each wrong expert is *halved*, it holds that

$$W^{(t+1)} = \frac{I^{(t)}}{2} + (W^{(t)} - I^{(t)})$$

since the *total* weight of the wrong experts is halved as well, and the second term of the sum is the remaining total weight — i.e. the sum of the weights of the experts who were *right*.

Can we employ a similar approach in this second setting? First, we need to understand how to get $w_i^{(t+1)}$ given $w_i^{(t)}$ if the i-th expert was wrong — which is not explicit in the f-Generalized version. We observe that $f(w_i^{(t)}) = \frac{1}{k}$ and $f(w_i^{(t+1)}) = \frac{1}{k+1}$ — for $k \geq 1$, implying that

$$k = \frac{1}{f(w_i^{(t)})} \implies f(w_i^{(t+1)}) = \frac{1}{k+1} = \frac{1}{\frac{1}{f(w_i^{(t)})} + 1} = \frac{f(w_i^{(t))}}{1 + f(w_i^{(t)})}$$

which means that we can rewrite the f-Generalized version by setting all the weights $w_i^{(0)} := 1$ at the beginning, and then use the following *updating rule*

$$w_i := \frac{w_i}{1 + w_i}$$

From now on, we will use this version of the f-Generalized variant of the algorithm, in order to reason more clearly w.r.t. the updating rule we are using.

In particular, this updating rule implies that the equation for $W^{(t+1)}$ does not hold anymore, for two reasons:

- we are not *halving* the wrong weights anymore, but we are decreasing $I^{(t)}$ by an amount that directly depends on the weights themselves
- even assuming that
  $$W^{(t+1)} = \varepsilon I^{(t)} + (W^{(t)} - I^{(t)})$$
  for some amount $\varepsilon > 0$ would not be correct, because the amount directly depends on the weights, therefore ε *changes over time* and cannot be assumed to be a constant value

In conclusion, this suggests that a different approach is needed in order to prove that $m = \Omega(n)$, even for $m^* = 0$.

In fact, an *adversarial argument* suffices to prove such lower bound on m, and the answer lies in how the updating rule operates. In particular, starting from $w_i = 1$, we get that each time the i-th expert is wrong w_i is transformed as follows:

$$1 \xrightarrow{f} \frac{1}{2} \xrightarrow{f} \frac{1}{3} \xrightarrow{f} \dots$$

which is precisely the harmonic sequence. In other words, after k mistakes of the i-th expert, it holds that $w_i = h_k = \frac{1}{k}$, where h_k is the k-th member of the harmonic sequence.

Hence, consider a set of n experts such that $m^* = 0$ — i.e. the best expert makes no mistakes, which means that there is at least one perfect expert. Let the first expert be the perfect one, and all the other $n-1$ experts be such that they always answer 0. Then, suppose that the correct answer is *always 1*. Thus, we have that

- the perfect expert is always correct, hence $w_1^{(t)} = 1$ for any $t \in [0, T]$
- all the other experts are wrong *at each iteration*, hence $w_i^{(t)} = \frac{1}{t+1}$ for each $i = [2, n]$ and for each $t \in [0, T]$

implying that when $t < n - 2$ it holds that

$$A_t = \sum_{\substack{i \in [n]: \\ y_{i,t}=1}} w_i^{(t)} = 1 = (n-1) \cdot \frac{1}{n-1} < (n-1) \cdot \frac{1}{t+1} = \sum_{\substack{i \in [n]: \\ y_{i,t}=0}} w_i^{(t)} = B_t$$

which means that for the first $n-2$ iterations (we start at $t = 0$) the algorithm is *forced to be wrong*. This implies that $m \geq n - 2$, and by choosing for instance $c = \frac{1}{2}$ and $n_0 = 4$ we get that

$$\forall n \geq n_0 \quad m \geq c \cdot n \implies m = \Omega(n)$$

Lastly, if we assume that $f(w_i) = (w_i + 1)^{-\alpha}$ for some $\alpha > 0$ we observe that the weights now change as follows

$$1 \xrightarrow{f} \frac{1}{2^\alpha} \xrightarrow{f} \frac{1}{3^\alpha} \xrightarrow{f} \dots$$

which implies that $w_i = (h_k)^\alpha = \frac{1}{k^\alpha}$. Therefore, we can employ the same adversary argument to find a lower bound on m, but this time to determine the number of iterations in which our algorithm is forced to be wrong we need t to be such that

$$1 < \frac{n-1}{(1+t)^\alpha} \iff (1+t)^\alpha < n - 1 \implies t < (n-1)^{\frac{1}{\alpha}} - 1$$

and since we start at $t = 0$, this implies that

$$m \geq (n-1)^{\frac{1}{\alpha}} - 1 \implies m = \Omega\left(n^{1/\alpha}\right)$$

□

Bibliography

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, et al. "Proof verification and the hardness of approximation problems". In: *Journal of the ACM* 45.3 (May 1998), 501–555. ISSN: 1557-735X. DOI: 10.1145/278298.278306. URL: http://dx.doi.org/10.1145/278298.278306.

[ARV09] Sanjeev Arora, Satish Rao, and Umesh Vazirani. "Expander flows, geometric embeddings and graph partitioning". In: *Journal of the ACM* 56.2 (Apr. 2009), 1–37. ISSN: 1557-735X. DOI: 10.1145/1502793.1502794. URL: http://dx.doi.org/10.1145/1502793.1502794.

[Ber57] Claude Berge. "TWO THEOREMS IN GRAPH THEORY". In: *Proceedings of the National Academy of Sciences* 43.9 (Sept. 1957), 842–844. ISSN: 1091-6490. DOI: 10.1073/pnas.43.9.842. URL: http://dx.doi.org/10.1073/pnas.43.9.842.

[Bou85] J. Bourgain. "On lipschitz embedding of finite metric spaces in Hilbert space". In: *Israel Journal of Mathematics* 52.1–2 (Mar. 1985), 46–52. ISSN: 1565-8511. DOI: 10.1007/bf02776078. URL: http://dx.doi.org/10.1007/BF02776078.

[Cha00] Moses Charikar. "Greedy Approximation Algorithms for Finding Dense Components in a Graph". In: *Approximation Algorithms for Combinatorial Optimization*. Springer Berlin Heidelberg, 2000, 84–95. ISBN: 9783540444367. DOI: 10.1007/3-540-44436-x_10. URL: http://dx.doi.org/10.1007/3-540-44436-X_10.

[EFF+21] Alon Eden, Michal Feldman, Amos Fiat, et al. "An Economics-Based Analysis of RANKING for Online Bipartite Matching". In: *Symposium on Simplicity in Algorithms (SOSA)*. Society for Industrial and Applied Mathematics, Jan. 2021, 107–110. ISBN: 9781611976496. DOI: 10.1137/1.9781611976496.12. URL: http://dx.doi.org/10.1137/1.9781611976496.12.

[ER22] P. Erdős and A. Rényi. "On random graphs. I." In: *Publicationes Mathematicae Debrecen* 6.3–4 (July 2022), 290–297. ISSN: 0033-3883. DOI: 10.5486/pmd.1959.6.3-4.12. URL: http://dx.doi.org/10.5486/PMD.1959.6.3-4.12.

[FMV11] Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. "Maximizing Non-monotone Submodular Functions". In: *SIAM Journal on Computing* 40.4 (Jan. 2011), 1133–1153. ISSN: 1095-7111. DOI: 10.1137/090779346. URL: http://dx.doi.org/10.1137/090779346.

[GW95] Michel X. Goemans and David P. Williamson. "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite pro-

gramming". In: *Journal of the ACM* 42.6 (Nov. 1995), 1115–1145. ISSN: 1557-735X. DOI: 10.1145/227683.227684. URL: http://dx.doi.org/10.1145/227683.227684.

[HK73] John E. Hopcroft and Richard M. Karp. "An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs". In: *SIAM Journal on Computing* 2.4 (Dec. 1973), 225–231. ISSN: 1095-7111. DOI: 10.1137/0202019. URL: http://dx.doi.org/10.1137/0202019.

[Kar72] Richard M. Karp. "Reducibility among Combinatorial Problems". In: *Complexity of Computer Computations*. Springer US, 1972, 85–103. ISBN: 9781468420012. DOI: 10.1007/978-1-4684-2001-2_9. URL: http://dx.doi.org/10.1007/978-1-4684-2001-2_9.

[Kho02] Subhash Khot. "On the power of unique 2-prover 1-round games". In: *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. STOC02. ACM, May 2002. DOI: 10.1145/509907.510017. URL: http://dx.doi.org/10.1145/509907.510017.

[KVV90] R. M. Karp, U. V. Vazirani, and V. V. Vazirani. "An optimal algorithm for on-line bipartite matching". In: *Proceedings of the twenty-second annual ACM symposium on Theory of computing - STOC '90*. STOC '90. ACM Press, 1990, 352–358. DOI: 10.1145/100216.100262. URL: http://dx.doi.org/10.1145/100216.100262.

[Lov75] L. Lovász. "On the ratio of optimal integral and fractional covers". In: *Discrete Mathematics* 13.4 (1975), 383–390. ISSN: 0012-365X. DOI: 10.1016/0012-365x(75)90058-8. URL: http://dx.doi.org/10.1016/0012-365X(75)90058-8.

[LR88] Tom Leighton and Satish Rao. "An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms". In: *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society. 1988, pp. 422–431.

[NWF78] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. "An analysis of approximations for maximizing submodular set functions—I". In: *Mathematical programming* 14 (1978), pp. 265–294.

[Rag08] Prasad Raghavendra. "Optimal algorithms and inapproximability results for every CSP?" In: *Proceedings of the fortieth annual ACM symposium on Theory of computing*. STOC '08. ACM, May 2008, 245–254. DOI: 10.1145/1374376.1374414. URL: http://dx.doi.org/10.1145/1374376.1374414.