SAPIENZA
UNIVERSITÀ DI ROMA

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

Advanced Algorithms

Lecture notes integrated with the book "Algorithm Design",
J. Kleinberg, É. Tardos

Author
Simone Bianco

March 30, 2025

Contents

Information and Contacts **1**

1 Approximation algorithms **2**
1.1 Coping with untractability 2
1.2 Approximations through randomness 3
1.2.1 The Maximum Cut problem 3
1.3 Approximations through problem reduction 7
1.3.1 The Minimum Vertex Cover problem 7

2 Mathematical programming **11**
2.1 Approximations through Linear programming 11
2.2 Integrality gap 15
2.2.1 The Minimum Set Cover problem 18
2.2.2 The Densest Subgraph problem 22
2.3 Approximations through duality 27
2.4 Approximation and runtime trade-offs 33
2.5 Approximations through Semidefinite programming 36
2.6 The Unique Games Conjecture 44

3 Metric Geometry **47**
3.1 The Sparsest Cut problem 47
3.2 Cut metrics 48
3.3 ℓ_1 metrics and isometric embeddings 51

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/Exyss/university-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: **bianco.simone@outlook.it**
- LinkedIn: **Simone Bianco**

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

Suggested prerequisites:

Sufficient knowledge of computability theory, algorithm complexity, number theory and probability

Licence:

1

Approximation algorithms

1.1 Coping with untractability

In computer science and optimization, approximation algorithms are algorithms designed to find near-optimal solutions to computational problems that are NP-hard, i.e. every problem that is verifiable in polynomial time can be reduced to them. Even though the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ question is still unsolved – which corresponds to determining if every problem efficiently verifiable is also efficiently solvable – lots of theoretical results make us believe that $\mathsf{P} \neq \mathsf{NP}$. Hence, we usually assume that the conjecture has actually been proven as false. This means that every NP-hard problem is **untractable**, meaning that there is no polynomial-time algorithm that can solve them exactly in all cases. Hence, **approximation algorithms** are used to find solutions that are "good enough" or "close enough" to the optimal, with a known error bound, sacrificing exactness for efficiency.

In particular, approximation algorithms are used for **optimization problems**, i.e. every type of problem that asks to find a structure that maximizes or minimizes a property. An approximation algorithm is typically evaluated based on how close its solution is to the optimal solution. The approximation ratio is defined as the worst-case ratio between the cost of the solution produced by the algorithm and the cost of the optimal solution.

For instance, if an algorithm for a minimization problem has an approximation ratio of ρ, then the value ℓ of the approximate solution is guaranteed to be at most ρ times as large as the optimal solution value ℓ^*.

$$\ell \leq \rho\ell^* \implies \rho = \frac{\ell}{\ell^*}$$

For a maximization problem, instead, the solution is guaranteed to be at least ρ times as large as the optimal solution value ℓ^*.

$$\ell \geq \rho\ell^* \implies \rho = \frac{\ell^*}{\ell}$$

1.2 Approximations through randomness

1.2.1 The Maximum Cut problem

The **Maximum Cut** problem is a fundamental optimization problem in graph theory and combinatorial optimization. In particular, the problem has numerous practical applications, including in network design, statistical physics (particularly in the study of spin glasses), and in various areas of machine learning, where it is used to model problems such as clustering and data partitioning. Given an undirected graph, the goal of the Max-cut problem is to **partition** the graph's vertices into two disjoint subsets such that the number of edges between the two subsets, i.e. outgoing from one subset to the other, is maximized. This partition is referred to as a **cut**, while the set of edges whose endpoints don't lie in the same subset is called **cut-set**.

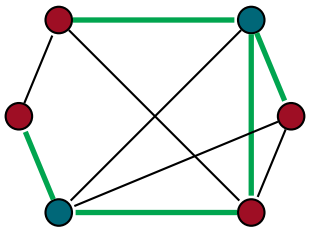


Figure 1.1: The red vertices and the blue vertices form a cut of the graph. The green edges are the edges of the cut-set.

To distinguish between directed and undirected graphs, in the undirected case we'll define the set of edges E of a graph $G = (V, E)$ as $E \subseteq \binom{V}{2} = \{\{u, v\} \mid u, v \in V\}$, while in the directed case we have that $E \subseteq \{(u, v) \mid u, v \in V\}$, where (u, v) represents and edge $u \to v$.

Definition 1.1: Cut of a graph

Given an undirected graph G, a cut of G is a bipartition (S, T) of G where $T = V - S$ and $S \neq \varnothing$. The cut-set of a cut (S, T) is defined as $\text{cut}(S, T) = \{e \in E(G) \mid |S \cap e| = 1\}$

The MC problem is concerned with finding the cut that maximizes the number of edges crossing between the two subsets of the cut (or the total weight of the edges in the cut-set in the weighted case). In particular, we'll focus on the unweighted case of this problem. Unlike its minimization counterpart, i.e. the Min-cut problem, the Max-cut problem is notable for being NP-hard by reduction from the Maximum Independent set problem [GJ90]. The Min-cut problem, instead, is known to lie in P by reduction to the s-t Maximum Cut problem, which is equivalent to the maximum network flow problem.

While finding the optimal solution for the Max-cut problem is computationally intractable for large graphs, significant progress has been made in designing algorithms that can find near-optimal solutions efficiently. One such approach is the famous Goemans-Williamson algorithm, which provides a $(0.878\ldots)$-approximation for MC, i.e. a solution that has at

least a $(0.878\ldots)$-th of the edges of the optimal solution. using semidefinite programming and randomization. We'll see this algorithm in later sections. It is known that there exists a constant $c < 1$ such that there cannot exist any c-approximation algorithm for MC unless $\mathsf{P} = \mathsf{NP}$ is true [ALM+98], which we assume to be false. For MC, this constant is known to be as small as $\frac{83}{84} \approx 0.988$.

For now, we'll focus on showing that **randomness** can be used to get a trivial expected $\frac{1}{2}$-approximation of the problem in polynomial time with a sufficiently high probability. This represents the typical case where randomness can be used to get a good enough polynomial time solution with the small trade-off of having a low probability of getting a solution that is below-expectations.

Algorithm 1.1 The random-cut algorithm

Input: an undirected graph G
Output: a cut (S, T) of G

```
1: function RANDOM-CUT(G)
2:     S ← ∅
3:     for v ∈ V(G) do
4:         Flip a fair independent coin and set c_v as the outcome
5:         if c_v = 1 then                          ▷ 1 is heads, 0 is tails
6:             S ← S ∪ {v}
7:         end if
8:     end for
9:     Return (S, S̄)
10: end function
```

The runtime of this algorithm is clearly $O(n)$ if S is stored using a set data structure. We also notice that the RANDOM-CUT algorithm actually doesn't even care about the graph structure: we're just flipping coins. This idea can be used for many other problems. We now prove that it yields an expected $\frac{1}{2}$-approximation of MC.

Theorem 1.1

Given a graph G, let (S^*, T^*) be an optimal solution to MC(G). Given the output (S, T) of RANDOM-CUT(G), it holds that:

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

Proof. For any edge $e \in E(G)$, we know that $e \in \text{cut}(S, T)$ if and only if $|S \cap e| = 1$. If $e = \{u, v\}$, this is also also equivalent to saying that $u \in S, v \notin S$ or $u \notin S, v \in S$. We notice that:

$$\begin{aligned}
\Pr[e \in \text{cut}(S, T)] &= \Pr[(u \in S, v \notin S) \vee (u \notin S, v \in S)] \\
&= \Pr[u \in S, v \notin S] + \Pr[u \notin S, v \in S] - \Pr[u \in S, v \notin S, u \notin S, v \in S] \\
&= \frac{1}{4} + \frac{1}{4} + 0
\end{aligned}$$

thus, we get that:

$$\mathbb{E}[|\text{cut}(S,T)|] = \sum_{e \in E(G)} 1 \cdot \Pr[e \in \text{cut}(S,T)] = \frac{|E(G)|}{2}$$

Finally, since each cut-set is by definition a subset of $E(G)$, we know that $|\text{cut}(S^*,T^*)| \leq E(G)$, concluding that:

$$\mathbb{E}[|\text{cut}(S,T)|] = \frac{|E(G)|}{2} \geq \frac{|\text{cut}(S^*,T^*)|}{2}$$

□

On first impact, this algorithm may seem useless: the solution is only *expected* to be a $\frac{1}{2}$-approximation of the optimal maximum cut of the input graph. In fact, if we are very unlucky, the solution could contain all the edges or the graph or even no edges at all. However, this algorithm is actually enough. In fact, we can show that, by running this algorithm a sufficient amount of times, the probability of getting a bad solution can be highly reduced.

Algorithm 1.2 The t-times random-cut algorithm

Input: an undirected graph G and a non-negative integer t
Output: a cut (S,T) of G

```
function t-RANDOM-CUT(G, t)
    for i ∈ [⌈t⌉] do
        (S_i, T_i) ← RANDOM-CUT(G)
    end for
    Return (S, S̄) ∈ arg max_{i∈[⌈t⌉]} |cut(S_i, T_i)|
end function
```

Theorem 1.2

Given a graph G and a non-negative integer t, let (S^*,T^*) be an optimal solution to MC(G). Given the output (S,T) of t-RANDOM-CUT(G), it holds that:

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

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

Proof. For each $i \in [\lceil t \rceil]$, let $C_i = \text{cut}(S_i,T_i)$, where $(S_1,T_1),\ldots,(S_t,T_t)$ are the cuts yielded by the algorithm, and let $N_i = |E(G)| - C_i$. Since N_i is a non-negative random variable, by Markov's inequality we have that:

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

Through some algebraic manipulation, and by linearity of the expected value operator, we get that:

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

Using the same argument of the previous theorem, we know that $\mathbb{E}[C_i] = \frac{|E|}{2}$. Hence, we get that:

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

We notice that the event of the last probability corresponds to a "bad solution", i.e. one whose value is at most $(1-\varepsilon)$-th of the expected value. Since each run of RANDOM-CUT is independent from the others, the probability of all the solutions being bad is bounded by:

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

Since $0 < 1 - \frac{\varepsilon}{2} < 2$ and $1 - \frac{\varepsilon}{2} \leq e^{-\frac{\varepsilon}{2}}$ (this last fact comes from the definition of e itself), we get that:

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

Hence, the probability of at least one solution being good is bounded by:

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

Finally, since the argmax operation inside the t-RANDOM-CUT algorithm will select (in the worst case) such good solution, we conclude that:

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

□

We observe that the result that we have just proved is very powerful. For instance, by choosing $\varepsilon, \delta = 0.1$, we get that:

$$\Pr\left[|\mathrm{cut}(S,T)| > (0.45)\,|\mathrm{cut}(S^*,T^*)|\right] \geq 0.9$$

and $t \approx 46$, meaning that we have to run RANDOM-CUT approximately 46 times in order to almost certainly get a solution that is better than a (0.45)-approximation. We also notice that notice that $0 < \frac{1-\varepsilon}{2} < 0.5$ since $\forall \varepsilon > 0$, meaning that we will always sacrifice some optimality to boost our probability.

1.3 Approximations through problem reduction

1.3.1 The Minimum Vertex Cover problem

The **Minimum Vertex Cover** problem is a well-known optimization problem in graph theory and combinatorial optimization. It involves finding the smallest subset of vertices in a graph such that every edge is incident to at least one vertex in the set. Like the Max-cut problem, the Minimum Vertex Cover problem is also NP-hard by reduction from the Maximum Clique problem.

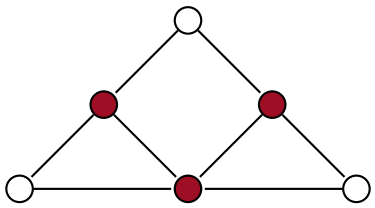


Figure 1.2: The red nodes are the smallest possible vertex cover of the graph.

Definition 1.2: Vertex Cover

Given an undirected graph G, a vertex cover over G is a subset $C \subseteq V(G)$ such that $\forall e \in E(G)$ there is a vertex $v \in C$ such that $v \in e$.

Before proceeding, it's important to distinguish between the concepts of *minimal* and *minimum*. In general, given a property P, a sub-structure X of a structure S is said to be minimal for P over S if $P(X)$ is true and there is no other sub-structure X' of S such that $P(X')$ is true and X' is contained inside X. Instead, X is said to be the minimum for P over S if $P(X)$ is true and there is no other sub-structure X' of S with a lower value for the property $P(X)$. For instance, a minimal vertex cover is a vertex cover that doesn't contain another vertex cover inside it – meaning that we cannot remove vertices and keep the property true – while a minimum vertex cover is a vertex cover with the lowest possible cardinality.

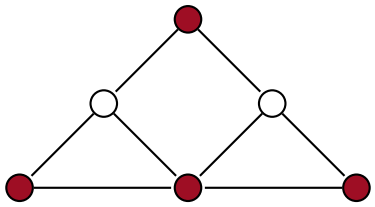


Figure 1.3: The red nodes form a minimal vertex cover of the graph since removing any of them wouldn't preserve the cover property. This vertex cover is not a minimum one.

Vertex covers are highly related to the concept of **matching**. In fact, an approximation for the Minimum Vertex Cover problem can be achieved through the Maximal Matching problem. A matching over a graph is a subset of edges that share no common endpoint. The difference between maximality and maximum is the same as the one between minimality and minimum.

Definition 1.3: Matching

Given an undirected graph G, a matching over G is a subset $M \subseteq E(G)$ such that $\forall e, e' \in M$ it holds that $e \cap e' = \varnothing$

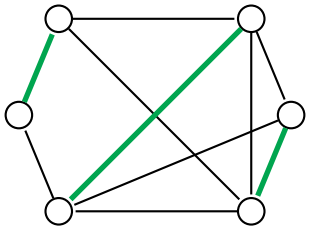


Figure 1.4: The green edges form a maximal matching of the graph.

Clearly, a maximal matching can be constructed in polynomial time through by simply adding edges until the property is preserved. In an even more efficient way, it can be computed by the following algorithm.

Algorithm 1.3 The maximal matching algorithm

Input: an undirected graph G
Output: a maximal matching M of G

```
1:  function MAXIMAL-MATCHING(G)
2:      M ← ∅
3:      E' ← E(G)
4:      while E' ≠ ∅ do
5:          Choose e ∈ E'
6:          S ← S ∪ {e}
7:          E' ← E' − {f ∈ E' | e ∩ f ≠ ∅}
8:      end while
9:      Return M
10: end function
```

We observe that the edges $e_1, \ldots, e_t$ picked by the algorithm are always a maximal matching: if $M = \{e_1, \ldots, e_t\}$ is not maximal then at least one edge could still get picked by the algorithm, meaning that it shouldn't have stopped.

Lemma 1.1

Let G be an undirected graph. For any matching M of G and any vertex cover C of G it holds that $|M| \leq |C|$.

Proof. By definition, we observe that if C is a vertex cover for G then it is also a vertex cover for $G' = (V, E')$, where $E' \subseteq E(G)$. Hence, C is also a vertex cover for any $G_M = (V, M)$, where M is a matching of G. By definition of matching, in G_M any vertex has either degree 0 or 1. Thus, each vertex of C can cover at most one edge of M, meaning that C has to have at least $|M|$ vertices to cover all the edges of M. □

We observe that the lemma above is valid for any matching and any vertex cover, not only maximal and minimum ones, making it less specific for our situation. Nonetheless, we can use it to show that the following algorithm is actually a 2-approximation of VC.

Algorithm 1.4 2-approximation of VC

Input: an undirected graph G
Output: a vertex cover for G

1: **function** 2-APPROX-VC(G)
2: $M \leftarrow$ MAXIMAL-MATCHING(G)
3: Return $C = \bigcup_{e \in M} e$
4: **end function**

Theorem 1.3

Given a graph G, let C^* be an optimal solution to VC(G). Given the output C of 2-APPROX-VC(G), it holds that $|C| \leq 2\,|C^*|$.

Proof. Let $M = \{e_1, \ldots, e_t\}$ be the maximal matching returned by MAXIMAL-MATCHING(G). Since $\forall e, e' \in M$ it holds that $e \cap e' = \varnothing$ by definition of matching, it holds that $|C| = 2\,|M|$. Hence, since C^* is a vertex cover, by the previous lemma we get that $|C| = 2\,|M| \leq 2\,|C^*|$. □

This result look quite easy, making us believe that this bound can be highly improved. However, it is conjectured that VC may be NP-hard to approximate to any ratio $2 - \varepsilon$ for any constant $\varepsilon > 0$ – the Unique Games Conjecture implies this result, which is conjectured to be true. Hence, this simple approximation algorithm may actually be the best we can achieve.

The vertex cover is also known to lie in the class of **Fixed-parameter Tractable (FPT)** problems, i.e the set of problems that can be solved in time $f(k) \cdot n^{O(1)}$, where f is a computable function and k is a fixed input parameter. We observe that, since k is fixed, the value $f(k)$ becomes a "constant", making the running time polynomial with respect to the size of the input. The crucial part of the definition is to exclude functions of the form $f(k, n)$, such as k^n.

Algorithm 1.5 Existence of a vertex cover

Input: an undirected graph G an a non-negative integer k
Output: True if G has a vertex cover with at most k vertices, False otherwise

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

Here, the notation $G[V - \{u\}]$ corresponds to the **induced subgraph** by $V - \{u\}$ on G, i.e. the graph obtained by removing the vertices in $V - \{u\}$ from G and all the edges that had one of such vertices as endpoints.

Definition 1.4: Induced subgraph

Given a graph G and a subset $S \subseteq V(G)$, the subgraph induced by S on G is the graph $G[S] = (S, E')$ such that $E' = \{e \in E(G) \mid S \cap e \neq \varnothing\}$.

Figure 1.5: The original graph (left) and the subgraph induced by the blue vertices (right).

To analyze the cost of the previous algorithm, we observe that the computational cost $T(n, m, k)$, where n is the number of nodes and m is the number of edges, is upper bounded by

$$T(n, m, k) \leq 2T(n, m, k - 1) + O(n + m)$$

where $T(n, m, 0) = T(n, 0, k) = \Theta(1)$. Hence, we get that $T(n, m, k) = O(2^k(n + m))$. To get a more precise computational cost, we observe that if a graph G has a vertex cover C with k vertices then $|E(G)| \leq (n - 1)k$ since in the worst case each of the k vertices has degree $n-1$. Since $O(nk) = O(n)$ when k is fixed, we get that $T(n, m, k) = O(2^k n)$.

2

Mathematical programming

2.1 Approximations through Linear programming

Mathematical Programming involves using mathematical models and optimization techniques to solve problems that require finding the best solution from a set of possible choices, subject to constraints. It plays a key role in areas like operations research, artificial intelligence, machine learning, and systems design. In mathematical programming, an optimization problem is typically expressed as:

$$\text{Minimize (or Maximize)} \quad f(x)$$

subject to:

$$\begin{aligned} g_i(x) &\leq b_i \quad \forall i \in [m] \\ h_i(x) &= t_i \quad \forall i \in [m] \\ x &\in V \end{aligned}$$

Here, x is a vector of decision variables inside the vector space V (usually $\mathbb{Q}^n$ or $\mathbb{R}^n$) and $f(x)$ is an objective function, while $g_i(x) \leq b_i$ and $h_i(x) \leq t_i$represent inequality and equality constraints. More specifically, we'll focus on **Linear Programming (LP)** and **Semi-Definite Programming (SDP)**. In linear programs, both the objective function and constraints are linear with respect to $\mathbb{Q}^n$ (or $\mathbb{R}^n$). Moreover, there is no equality constraints and each variable x_i must be non-negative.

$$\begin{aligned} \max\, & x_1 + x_2 \\ & x_1 + 6x_2 \leq 15 \\ & 4x_1 - x_2 \leq 10 \\ & x \geq 0 \\ & x \in \mathbb{R}^n \end{aligned}$$

Figure 2.1: Example of a linear program.

We observe that linear programs can be described in a compact matricial formulation.

$$\begin{aligned} \max\, & c^T x \\ & Ax \leq b \\ & x \geq 0 \\ & x \in \mathbb{R}^n \end{aligned}$$

Figure 2.2: Standard matricial formulation of a linear program.

where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$.

$$\begin{aligned} \max\, & \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ & \begin{bmatrix} 1 & 6 \\ 4 & -1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 15 \\ 10 \\ 1 \end{bmatrix} \\ & x \geq 0 \\ & x \in \mathbb{R}^n \end{aligned}$$

Figure 2.3: Matricial formulation of the previous a linear program example.

Linear programs can be solved in polynomial time. In particular, if a linear program has n variables, m constraints and each coefficient is representable as the ratio of two t-bits integers (real numbers get approximated as rational number) then the LP can be solved through the *Ellipsoid method* in time $O((nmt)^c)$ for some $c > 0$ (this can be in someway extended also to SDPs). Even though it is theoretically guaranteed to have polynomial time, the Ellipsoid method becomes useful only for very large inputs. For more practical cases, the *Simplex method* is used, which is based on *pivot rules*. All of the pivot rules known for the Simplex method have a theoretical exponential lower bound through some particular programs that "fool" the rule, but they have an average complexity that is way better than the Ellipsoid method. For our purposes, we do not care about how these methods work: we're only interested in knowing that LPs and SDPs can be solved in polynomial time.

Integer programs are a particular type of linear program. Here, the vector space of interest is $\{0,1\}^n$. Lots of problem of common interested can be reduced to a linear program. For instance, given a graph G, consider the following integer program defined over the variables $x_{v_1}, \ldots, x_{v_n}$ where $V(G) = \{v_1, \ldots, v_n\}$.

$$\min \sum_{v \in V(G)} x_v$$

$$\begin{aligned} x_u + x_v &\geq 1 \quad \forall \{u, v\} \in E(G) \\ x &\in \{0,1\}^n \end{aligned}$$

Figure 2.4: Integer program for the vertex cover problem.

Its easy to see that the above program perfectly describes the VC problem. In fact, the optimal integral solution to this IP actually gives us an optimal minimal vertex cover.

Lemma 2.1

Given a graph G, if x^* is an optimal solution to the VC integer program then $C^* = \{v \mid v \in V(G), x_v^* = 1\}$ is a minimum vertex cover for G

Proof. Any feasible solution to the VC integer program corresponds to a vertex cover for G. In particular, in C^* for all $\{u, v\} \in E(G)$ we have that:

$$\begin{aligned} x_u^* + x_v^* &\geq 1 \\ x_u^*, x_v^* &\in \{0,1\} \end{aligned} \iff x_u^* = 1 \vee x_v^* = 1 \iff u \in C^* \vee v \in C^*$$

Claim 1: If C' is a vertex cover for G then there is a feasible x' to the VP integer program such that $\sum_{v \in V(G)} x'_v = |S|$.

Proof of the claim. Given a vertex cover C', set $x'_v = 1$ if and only if $v \in S$. Then, since for all $\{u, v\} \in E(G)$ we have that $u \in S$ or $v \in S$, it must hold that $x_u = 1$ or $x_v = 1$, satisfying the constraint $x_u + x_v \geq 1$. Since each constraint is satisfied, x' is a feasible solution. Moreover, by construction we have that $\sum_{v \in V(G)} x'_v = |S|$. □

By way of contradiction, suppose that C^* is not a minimum vertex cover. Then, there must be another vertex cover $V^\star$ such that $|V^\star| < |V^*|$. Thus, through the claim we know that there must another feasible solution $x^\star$ for which $\sum_{v \in V(G)} x^\star = |V^\star| < |V^*| = \sum_{v \in V(G)} x^*$, contradicting the fact that x^* is optimal. Hence, C^* must be a minimum vertex cover. □

The above lemma implies that the Minimum Vertex Cover problem can be reduced to Integer programming. Hence, solving IPs is actually NP-hard compared to solving LPs. This difference may seem counterintuitive: shouldn't the program be easier since we're working with way less feasible solutions? To give an intuition behind this hardness-gap, we can consider the fact that, since we're working over $\{0,1\}^n$ instead of $\mathbb{Q}^n$ or $\mathbb{R}^n$, we're intrinsically imposing lots of strong constrains over the space of feasible solutions, which becomes a lattice of integral vectors.

Hence, IPs cannot be used to get perfect solutions. However, they can be used to get approximate solutions through **LP relaxation**. The idea is simple: we replace the constraints $x \in \{0,1\}^n$ and $x \geq 0$ with the constraints $x \in \mathbb{R}^n$ and $0 \leq x \leq 1$, transforming the IP into an LP.

$$\min \sum_{v \in V(G)} x_v$$

$$\begin{aligned} x_u + x_v &\geq 1 \quad \forall \{u,v\} \in E(G) \\ 0 \leq x &\leq 1 \\ x &\in \mathbb{R}^n \end{aligned}$$

Figure 2.5: LP relaxation for the vertex cover problem.

First of all, we observe that (generally) the space of feasible solution gets enlarged when the IP gets relaxed. In fact, every feasible solution to the original IP is a solution to the relaxed LP, but the optimal integral solution is not guaranteed to be an optimal solution to the LP. Moreover, we also observe that the optimal non-integral solution may not make sense for the original transformation.

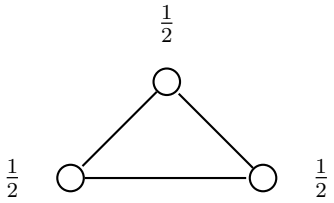


Figure 2.6: The non-integral optimal solution for K_3 is $\overline{x} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix}^T$.

To fix this issue, the LP optimal solution is usually transformed into an approximate solution through techniques such as **rounding**. For instance, in the VS problem the rounding procedure is pretty obvious: in order for the constraints to be satisfied, for each edge at least one the two endpoints must have value greater than $\frac{1}{2}$. Hence, we select all the nodes that have value at least $\frac{1}{2}$. Surprisingly, this simple rounding procedure is guaranteed to yield a 2-approximation for the VC problem.

Theorem 2.1

Given a graph G, if $\overline{x}$ is an optimal solution to the LP relaxation of the VC integer program then $\overline{C} = \{v \mid v \in V(G), \overline{x}_v \geq \frac{1}{2}\}$ is a 2-approximation for VC(G).

Proof. First of all, we observe that any feasible solution to the VC relaxed program still corresponds to a vertex cover for G. In particular, in $\overline{C}$ for all $\{u,v\} \in E(G)$ we have

that:

$$\begin{matrix} \overline{x}_u + \overline{x}_v \geq 1 \\ 0 \leq \overline{x}_u, \overline{x}_v \leq 1 \end{matrix} \implies \overline{x}_u \geq \frac{1}{2} \vee \overline{x}_v \geq \frac{1}{2} \iff u \in \overline{C} \vee v \in \overline{C}$$

Claim: $\left|\overline{C}\right| \leq 2 \sum_{v \in V(G)} \overline{x}_v$.

Proof of the claim. We observe that:

$$\left|\overline{C}\right| = \sum_{v \in \overline{C}} 1 \leq \sum_{v \in \overline{C}} 2\overline{x}_v = 2 \sum_{v \in \overline{C}} \overline{x}_v \leq 2 \sum_{v \in V(G)} 2\overline{x}_v$$

□

Consider now an optimal solution x^* to the IP version of the problem. Since x^* is also a solution of the LP, but it isn't guaranteed to be optimal for the LP. Moreover, through Lemma 2.1 we know that x^* describes a minimum vertex cover C^*. Hence, we conclude that:

$$\left|\overline{C}\right| \leq \sum_{v \in V(G)} 2\overline{x}_v \leq \sum_{v \in V(G)} 2x_v^* = 2\left|C^*\right|$$

□

2.2 Integrality gap

In the proof of Theorem 2.1, we were able to claim that the simple rounding procedure based on LP relaxation that we used gives us a vertex cover $\overline{C}$ such that $\left|\overline{C}\right| \leq 2\text{LP}^*$, where LP^* is the optimal value of the objective function for the relaxation. If the claim inside the proof could be improved to $\left|\overline{C}\right| \leq (2-\varepsilon)\text{LP}^*$, we would get a better approximation ratio for the theorem since $\left|\overline{C}\right| \leq (2-\varepsilon)\text{IP}^*$ naturally follows from it. However, it can be show that the claim's bound is the best we can achieve through simple LP relaxation.

In Lemma 2.1, we showed how the solution $\overline{x} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix}^T$ is the optimal solution for the relaxed LP for the graph K_3. For the IP, instead, the solution $x^* = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}^T$ is clearly optimal for K_3. Hence, we get that $\text{IP}^* = 2$ and $\text{LP}^* = \frac{3}{2}$. Consider now a value $\alpha > 0$ such that $\left|\overline{C}\right| \leq \alpha\text{LP}^*$. Then, it must hold that:

$$\alpha \geq \frac{2}{\frac{3}{2}} = \frac{4}{3}$$

since otherwise we would get that:

$$\left|\overline{C}\right| \leq \alpha\text{LP}^* < \frac{4}{3} \cdot \frac{3}{2} = 2 = \text{IP}^*$$

meaning that $\overline{C}$ would be a solution better than the optimal solution of the VC integer program, which is impossible. This result implies that there is an **integrality gap** α

between the value of the optimal solution of the IP and the optimal solution of the relaxed LP.

Definition 2.1: Integrality gap

Given a problem P, consider the IP equivalent to P. Given an instance I of P, let $\text{IP}^*_P(I)$ and $\text{LP}^*_P(I)$ be the optimal values of the IP and the LP relaxation of the IP for I. The integrality gap between between $\text{IP}^*_P(I)$ and $\text{LP}^*_P(I)$, denoted as $\text{IG}_P(I)$, is defined as:

$$\text{IG}_P(I) = \frac{\text{IP}^*_P(I)}{\text{LP}^*_P(I)}$$

The integrality gap for P, denoted as IG_P, is defined as:

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

Proposition 2.1: Limits of LP relaxations

Given a problem P, let ALG_P be an algorithm that approximates P. Then:

- If P is a minimization problem and $\text{ALG}_P \leq \alpha \text{LP}^*_P$ then $\alpha \geq \text{IG}_P$
- If P is a maximization problem and $\text{ALG}_P \geq \alpha \text{LP}^*_P$ then $\alpha \leq \text{IG}_P$

Proof. Assume that P is a minimization problem. By way of contradiction, suppose that $\alpha < \text{IG}_P$. Then, for any instance I of P, it holds that:

$$\text{ALG}_P(I) \leq \alpha \text{LP}^*(I) < \text{IG}_P(I) \cdot \text{LP}(I)^*_P = \frac{\text{IP}^*_P(I)}{\text{LP}^*_P(I)} \cdot \text{LP}^*_P(I) = \text{IP}^*_P(I)$$

which is a contradiction. A similar argument can be made for the case where P is a maximization problem. □

When the context makes it clear, we'll refer to $\text{IG}_P, \text{IP}^*_P$ and LP^*_P directly as IG, IP^* and LP^*. For the vertex cover problem, we showed that $\text{IG}(K_3) = \frac{4}{3} = 2 - \frac{2}{3}$, meaning that $\text{IG} \geq 2 - \frac{2}{3}$. By generalizing the argument to K_n, we can show that the integrality gap of the vertex cover problem is exactly 2, meaning that $\forall \varepsilon > 0$ there can be no algorithm ALG such that $\text{ALG} \leq (2 - \varepsilon)\text{LP}^*$ holds for all graphs.

Theorem 2.2: Integrality gap of VC

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

Proof. We already know that the rounding procedure that we used is such that for all graphs G it holds that $\text{ALG}(G) \leq 2\text{LP}^*(G)$. Since the output of such rounding procedure is a (non-optimal) vertex cover, we know that $\text{IP}^*(G) \leq \text{ALG}(G)$. Hence, for all graphs

G we get that:

$$\mathrm{IP}^*(G) \leq 2\mathrm{LP}^*(G) \implies \frac{\mathrm{IP}^*(G)}{\mathrm{LP}^*(G)} \leq 2$$

concluding that $\mathrm{IG} \leq 2$. Consider now the graph K_n. We observe that the vector $\overline{x}$ such that $x_v = \frac{1}{2}$ for all $v \in V(K_n)$ is a feasible solution for the LP relaxation of the VC problem. Hence, we know that:

$$\mathrm{LP}^*(K_n) \leq \sum_{v \in V(K_n)} x_v = \frac{n}{2}$$

Claim: any minimum vertex cover for K_n has exactly $n-1$ nodes.

Proof of the claim. First, we show that there is a vertex cover of size $n-1$ for K_n. Let $C = \{x_1, \ldots, x_{n-1}\}$. For each edge $\{x_i, x_j\} \in E(K_n)$ with $i, j \in [n-1]$ we have that both x_i, x_j are inside C. For each edge $\{x_i, x_n\} \in E(K_n)$ with $i \in [n-1]$, instead, we have that $x_i \in C$. Since every edge is covered by a vertex in C, C is a vertex cover of size $n-1$.

Suppose now by way of contradiction that there is a minimum vertex cover C^* for K_n of size $|C^*| \leq n-2$. Then, we have that $\exists u, v \in V - C^*$ where u, v. Since K_n is a n-clique, we have that $\{u, v\} \in E(K_n)$. However, we know that $u, v \notin C^*$, meaning that C doesn't cover $\{u, v\}$ and thus that it isn't a vertex cover. □

The claim concludes that $\mathrm{IP}^* = n-1$, thus the integrality gap for K_n is bounded by:

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

Hence, we conclude that:

$$\mathrm{IG} = \sup_G \mathrm{IG}(G) \geq \sup_{n \in \mathbb{N}} \mathrm{IG}(K_n) \geq \lim_{n \to +\infty} 2 - \frac{2}{n} = 2$$

□

Even thought the above propositions only refers to algorithms that are bounded by linear relaxations, meaning that other algorithms may indeed produce a better approximation ratio, for many problems we conjecture that the integrality gap is actually the best possible approximation ratio of any algorithm for the problem. For instance, it is conjectured that there is no $(2-\varepsilon)$-approximation of the VC problem unless $\mathsf{P} \neq \mathsf{NP}$, even thought no one has yet proven such result.

2.2.1 The Minimum Set Cover problem

In the previous section we were able to show that the integrality gap of the minimum vertex cover can be exactly computed, concluding that $\mathrm{IG}_{\mathrm{VC}} = 2$. However, the integrality gap cannot be always nicely computed. Nonetheless, finding upper and lower bounds to it still allows us know if our rounding procedure is good enough or not. This is the case of the **Minimum Set Cover** problem.

Given a number $n \in N$, let $\mathcal{U} = [n]$ be the universe set and let $\mathcal{C} = \{S_1, \ldots, S_m\}$ be a collection of subsets $S_i \subseteq \mathcal{U}$. The SC problems asks us to find the smallest sub-collection $\mathcal{S} \subseteq \mathcal{C}$ such that:

$$\mathcal{U} = \bigcup_{S_j \in \mathcal{S}} S_j$$

For instance, given $\mathcal{U} = [4]$ and $S_1 = \{1, 2\}, S_2 = \{2, 3\}, S_3 = \{3, 4\}$, the smallest set cover is given by $\mathcal{S} = \{S_1, S_3\}$. It's easy to see that the VC problem can be reduced to the SC problem by considering the universe set $\mathcal{U} = [|E(G)|]$ and the collection $\mathcal{C} = \{S_{v_1}, \ldots, S_{v_n}\}$ where $S_{v_i} = \{i \in [|E(G)|] \mid v_i \in e_i\}$, making the SC problem NP-hard. The IP program equivalent to the SC problem can be obtained by imposing constraints similar to the ones of the VC problem over the variables $x_1, \ldots, x_m$, where $x_j = 1$ if and only if j is selected in the output cover.

$$\begin{aligned} &\min \sum_{j \in [m]} x_j \\ &\sum_{\substack{j \in [m] \\ \text{s.t. } i \in S_j}} x_j \geq 1 \quad \forall i \in [n] \\ &x \in \{0, 1\}^m \end{aligned}$$

Figure 2.7: Integer program for set cover problem.

We observe that the $\frac{1}{2}$ rounding rule that we used on the optimal solution of the LP relaxation of vertex cover cannot be used for the LP relaxation of set cover. In the VC problem, we had constraints of the form $x_u + x_v \geq 1$, which imply that:

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

guaranteeing that at least one between x_u and x_v can be rounded to 1. Here, we have constraints of the form $\sum_{\substack{j \in [m] \\ \text{s.t. } i \in S_j}} x_j \geq 1$, which imply that:

$$\max(x_{j_1}, \ldots, x_{j_k}) \geq \frac{x_{j_1} + \ldots + x_{j_k}}{k} \geq \frac{1}{k}$$

where $j_1, \ldots, j_k$ are the indexed in $[m]$ for which $i \in S_{j_h}$. Thus, there is no guarantee of at least one between $x_{j_1}, \ldots, x_{j_j}$ to be rounded to 1. This means that we need a new

rounding rule for such problem. In particular, we'll use a **randomized rounding rule**, which, surprisingly, is the best for most problems.

Algorithm 2.1 Randomized rounding for SC

Input: an universe set $\mathcal{U}$ and a collection S
Output: a set cover A for U

function RANDOMIZED-ROUNDING-SC($\mathcal{U}, \mathcal{C}$)
    $A \leftarrow \varnothing$
    $\overline{x} \leftarrow \mathrm{LP}^*_{\mathrm{SC}}(\mathcal{U}, \mathcal{S})$
    for $k \in [\lceil 2 \ln n \rceil]$ **do**
        $A_k \leftarrow \varnothing$
        for $j \in [m]$ **do**
            Flip a coin with head probability set to $\overline{x_j}$ and set $c_{k,j}$ as the outcome
            if $c_{k,j} = 1$ **then** ▷ 1 is heads, 0 is tails
                $A_k \leftarrow A_k \cup \{S_j\}$
            end if
        end for
        $A \leftarrow A \cup A_k$
    end for
    Return A
end function

Lemma 2.2

Let $(\mathcal{U}, \mathcal{C})$ be an input of the SC problem. Given the output $\mathcal{S}$ of RANDOMIZED-ROUNDING-SC($\mathcal{U}, \mathcal{C}$), it holds that:

$$\Pr\left[\mathcal{U} \text{ covered by } A\right] = 1 - \frac{1}{n}$$

Proof. Fix an iteration $k \in [\lceil 2 \ln n \rceil]$ of the outer for loop and fix an element $i \in [n]$. Recalling that $\forall x \in \mathbb{R}$ it holds that $1 - x \leq e^{-x}$, we observe that:

$$\Pr[i \text{ not cov. by } A_k] = \prod_{\substack{j \in [m] \\ \text{s.c. } i \in S_j}} 1 - \overline{x}_j \leq \prod_{\substack{j \in [m] \\ \text{s.c. } i \in S_j}} e^{-\overline{x}_j} \leq \exp\left(- \sum_{\substack{j \in [m] \\ \text{s.c. } i \in S_j}} \overline{x}_j \right)$$

Since $\overline{x}$ is a solution to LP_{SC}, we have that:

$$\Pr[i \text{ not cov. by } A_k] \leq \exp\left(- \sum_{\substack{j \in [m] \\ \text{s.c. } i \in S_j}} \overline{x}_j \right) \leq e^{-1}$$

Since $A = A_1 \cup \ldots \cup A_k$, by iterating over all $k \in [\lceil 2 \ln n \rceil]$ the probability of i not being covered by A is bounded by:

$$\Pr[i \text{ not cov. by } A] = \Pr[\forall k \;\; i \text{ not cov. by } A_k] \leq \prod_{k \in [\lceil 2 \ln n \rceil]} e^{-1} = e^{-\lceil 2 \ln n \rceil} \leq e^{-2 \ln n} = \frac{1}{n^2}$$

Finally, by iterating over all $i \in [n]$, we get that:

$$\Pr[\mathcal{U} \text{ not cov. by } A] = \Pr[\exists i \in [n] \;\; i \text{ not cov. by } A] \leq \sum_{i \in [n]} \Pr[i \text{ not cov. by } A] \leq \frac{1}{n}$$

□

The above lemma guarantees with enough probability that the output of the randomized rounding rule is indeed a set cover – recall that we're interested to solve the problem for large values of n. We'll now show that the expected value of the solution is not so bad.

Lemma 2.3

Let $(\mathcal{U}, \mathcal{C})$ be an input of the SC problem. Given the output $\mathcal{S}$ of RANDOMIZED-ROUNDING-SC$(\mathcal{U}, \mathcal{C})$, it holds that:

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

Proof. Fix an iteration $k \in [\lceil 2 \ln n \rceil]$. We observe that:

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

Since $A = A_1 \cup \ldots \cup A_k$, by linearity of the expected value operator we get that:

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

□

Since through the above lemma we have implicitly proven that $\mathbb{E}[|A|] \leq \lceil 2 \ln n \rceil \, \mathrm{LP}^*$, through the law of large numbers we know that if for each pair $(\mathcal{U}, \mathcal{C})$ we repeatedly compute the algorithm randomized-rounding-sc we will, eventually, get an output A' such that $|A'| \leq \lceil 2 \ln n \rceil \, \mathrm{LP}^*$. Since $\mathrm{IP}^* \leq |A'|$, we get that:

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

Moreover, we observe that the algorithm can be modified to get a better bound. By simply replacing the number of iterations of the outer loop with $\lceil (1 + \varepsilon) \ln n \rceil$, for any $\varepsilon > 0$, we get that $\mathbb{E}[|A|] \leq \lceil (1 + \varepsilon) \ln n \rceil \, \mathrm{LP}^*$, 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, actually get that:

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

To show a lower bound on the integrality gap of set cover, we proceed in a way similar to what we did for the vertex cover problem. However, in this case we're not able to show a tight bound. The result is due to Lovász [Lov75].

Theorem 2.3: Integrality gap of SC

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

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

Proof. We have already showed that $\mathrm{IG}_{\mathrm{SC}} \leq \lceil \ln n \rceil$, hence proving the lower bound suffices. Fix $n \in \mathbb{N}$ and let $m \in \mathbb{N}$ be the even number such that $n = \binom{m}{\frac{m}{2}}$. We observe that:

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

Let $\mathcal{U}_n$ be the set of all elements representing a subset of $[m]$ with $\frac{m}{2}$ elements:

$$\mathcal{U}_n = \{e_a \mid A \in \binom{[m]}{\frac{m}{2}}\}$$

Let $\mathcal{C}_n = \{S_1, \ldots, S_m\}$, where $S_i = \{e_A \in \mathcal{U}_n \mid i \in A\}$.

Claim 1: $\mathrm{LP}^*(\mathcal{U}_n, \mathcal{C}_n) \leq 2$

Proof of Claim 1. First, we observe that, by construction, every element $e_A \in \mathcal{U}_n$ is in exactly $|A|$ sets. Hence, given the vector $x = \begin{bmatrix} \frac{2}{m} & \cdots & \frac{2}{m} \end{bmatrix}^T$ for each $e_A \in \mathcal{U}_n$ we have that:

$$\sum_{\substack{j \in [m] \\ \text{s.t. } e_A \in S_j}} x_j = \sum_{\substack{j \in [m] \\ \text{s.t. } e_A \in S_j}} \frac{2}{m} = |A| \ \frac{2}{m} = 1$$

meaning that x is a feasible solution for $\mathrm{LP}^*(\mathcal{U}_n, \mathcal{C}_n)$ with objective value 2. □

Claim 2: $\mathrm{IP}^*(\mathcal{U}_n, \mathcal{C}_n) \geq \frac{1}{2} \log n - O(\log logn)$

Proof of Claim 2. By way of contradiction, suppose that there is a sub-collection $\mathcal{S} = \{S_{i_k}, \ldots, S_{i_k}\}$ with $k \leq \frac{m}{2}$ that covers $\mathcal{U}_n$. Given $T = [m] - \{i_1, \ldots, i_k\}$, it holds that $|T| \geq m - \frac{m}{2} = \frac{m}{2}$. Hence, we can always find a subset $A \subseteq T$ such that $|A| = \frac{m}{2}$. Since $A \subseteq T$, we have that $e_A \notin S_{i_1} \cup \ldots \cup S_{i_k}$, contradicting the fact that $\mathcal{S}$ is a cover of $\mathcal{U}_n$.

Hence, there for any set cover $\mathcal{S}$ of $(\mathcal{U}_n, \mathcal{C}_n)$ it must hold that

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

Note: the asymptotic notation changes from Θ to O due to the less than constraint □

From the two claims we get that:

$$\text{IG} = \sup_{(\mathcal{U},\mathcal{C})} \text{IG}(\mathcal{U},\mathcal{C}) \geq \max_{n\in\mathbb{N}} \text{IG}(\mathcal{U}_n,\mathcal{C}_n) \geq \max_{n\in\mathbb{N}} \frac{\frac{1}{2}\log n - O(\log\log n)}{2}$$

which can happen only if:

$$\text{IG} = \sup_{(\mathcal{U},\mathcal{C})} \text{IG}(\mathcal{U},\mathcal{C}) \geq \frac{1}{4}\ \log n = \frac{1}{4\ln 2}\ \ln n$$

□

2.2.2 The Densest Subgraph problem

The **Densest Subgraph** problem asks to find a non-empty subset of vertices S in a given graph G whose induced subgraph $G[S]$ maximizes a measure of **density** $\rho(S)$, defined as the ratio between the number of edges in $G[S]$ and the number of vertices in S.

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

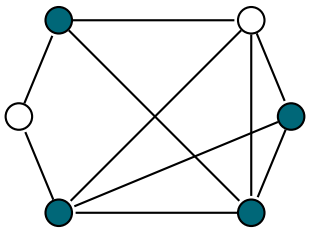


Figure 2.8: The subgraph induced by the blue nodes has density $\frac{4}{3}$

The Densest Subgraph problem became of particular interest in the '00s, mostly in Social Networks. For instance, companies such as Google used dense subgraphs to cut off *spam-networks*, i.e. website networks that used to spam-link each other in order to boost their ranking for the PageRank algorithm used by Google to recommend websites. We observe that the densest subgraph problem is equivalent to the **Maximum Average Degree Subgraph** problem:

$$\arg\max_{S\subseteq V} \frac{|E(G[S])|}{|S|} = \arg\max_{S\subseteq V} \frac{|E(G[S])|}{|S|} = \arg\max_{S\subseteq V} \frac{\sum_{v\in S} \deg_{G[S]}(v)}{2\,|S|} = \arg\max_{S\subseteq V} \frac{\sum_{v\in S} \deg_{G[S]}(v)}{|S|}$$

This equivalence between the two problems allows us to formulate the following IP representation of the problem, where each variable $x_{i,j}$ corresponds to an edge $\{i,j\} \in E(G)$ and each variable y_i corresponds to a vertex $i \in V(G)$.

$$\max \frac{\sum_{\{i,j\}\in E(G)} x_{i,j}}{\sum_{i\in V(G)} y_i}$$

$$\begin{aligned} x_{i,j} &\leq y_i && \forall \{i,j\} \in E(G) \\ x_{i,j} &\leq y_j && \forall \{i,j\} \in E(G) \\ \sum_{i\in V(G)} y_i &\leq 1 \\ x &\in \{0,1\}^m \\ y &\in \{0,1\}^n \end{aligned}$$

Figure 2.9: Integer program for densest subgraph problem.

To get a more standard looking IP program, we can normalize the variables in the following way.

$$\max \sum_{\{i,j\}\in E(G)} x_{i,j}$$

$$\begin{aligned} x_{i,j} &\leq y_i && \forall \{i,j\} \in E(G) \\ x_{i,j} &\leq y_j && \forall \{i,j\} \in E(G) \\ \sum_{i\in V(G)} y_i &\leq 1 \\ x &\in \{0,1\}^m \\ y &\in \{0,1\}^n \end{aligned}$$

Figure 2.10: Normalized integer program for densest subgraph problem.

Charikar [Cha00] proved that the integrality gap of this normalized version is 1, meaning that $IP^*_{\text{DS}} = LP^*_{\text{DS}}$. To show this, we prove the two following lemmas.

Lemma 2.4

Let G be a graph. Then, for each subset $S \subseteq V(G)$ there is a feasible solution to $LP_{\text{DS}}(G)$ with objective value equal to $\rho(S)$.

Proof. Given $S \subseteq V(G)$, let $\overline{x}, \overline{y}$ be defined as:

$$\overline{x_{i,j}} = \begin{cases} \frac{1}{|S|} & \text{if } \{i,j\} \in S \\ 0 & \text{otherwise} \end{cases} \qquad \overline{y_i} = \begin{cases} \frac{1}{|S|} & \text{if } \{i,j\} \in E(G[S]) \\ 0 & \text{otherwise} \end{cases}$$

Claim: $\overline{x}, \overline{y}$ is a feasible solution

Proof of the claim. By construction, we have that:

$$\sum_{i \in V(G)} y_i = \sum_{i \in S} y_i + \sum_{i \in \overline{S}} y_i = \sum_{i \in S} \frac{1}{|S|} + \sum_{i \in \overline{S}} 0 = 1$$

hence the third constraint is satisfied. For each edge $\{i,j\} \in E(G)$, if $\{i,j\} \in E(G[S])$ then $i, j \in S$, implying that $x_i = y_i = y_j = \left|\frac{1}{S}\right|$. If $\{i,j\} \notin E(G[S])$, instead, we have that $x_{i,j} = 0 \leq y_i, y_j$. In both cases, the constraints are satisfied. □

Finally, we observe that the objective value of $\overline{x}, \overline{y}$ is equal to the density of S:

$$\sum_{\{i,j\} \in E(G)} x_{i,j} = \sum_{\{i,j\} \in E(G[S])} x_{i,j} + \sum_{\{i,j\} \in E(G) - E(G[S])} x_{i,j} = \frac{|E(G[S])|}{|S|} = \rho(S)$$

□

Lemma 2.5

Let G be a graph. Then, for each feasible solution to $LP_{\text{DS}}(G)$ with objective value v, there is a subset $S \subseteq V(G)$ such that $\rho(S) \geq v$

Proof. Let x', y' be any feasible solution to $LP_{\text{DS}}(G)$. We observe that this solution may have some "slack" from the constraints, meaning that they are not satisfied at equality. To make things easier, we can transform the solution x', y' to a new solution objective value at least v and without slack. To achieve this, let $\overline{x}, \overline{y}$ such that $\overline{y_i} = y'_i$ for all $i \in V(G)$ and $\overline{x_{i,j}} = \min(y'_i, y'_j)$ for all $\{i,j\} \in E(G)$.

Claim 1: $\overline{x}, \overline{y}$ is a feasible solution with objective value at least v

Proof of Claim 1. Since x', y' is a feasible solution, it satisfies the constraints of the LP. Hence, we get that:

$$\sum_{i \in V(G)} \overline{y}_i = \sum_{i \in V(G)} y'_i \leq 1$$

and that:

$$x_{i,j} = \min(y'_i, y'_j) \leq y'_i, y'_j$$

for each $\{i,j\} \in E(G)$. Moreover, also by feasibility of x', y', we have that:

$$\sum_{\{i,j\} \in E(G)} \overline{x}_{i,j} = \sum_{\{i,j\} \in E(G)} \min(y'_i, y'_j) \geq \sum_{\{i,j\} \in E(G)} x'_{i,j} = v$$

□

For each $r \in \mathbb{R}$, let $S(r) = \{i \mid \overline{y}_i \geq r\}$ and let $E(r) = \{\{i,j\} \mid \overline{x}_{i,j} \geq r\}$.

Claim 2: $\{i,j\} \in E(r)$ if and only if $i,j \in S(r)$, i.e. $E(r) = E(G[S(r)])$

Proof of Claim 2. If $\{i,j\} \in E(r)$, we have that:

$$r \leq \overline{x_{i,j}} = \min(y'_i, y'_j) \leq \overline{y'}_i, \overline{y'}_j$$

hence $i,j \in S(r)$. Vice versa, if $i,j \in S(r)$ then:

$$r \leq \min(\overline{y}_i, \overline{y}_j) = \min(y'_i, y'_j) = \overline{x_{i,h}}$$

hence $\{i,j\} \in E(r)$. □

Now, things will get pretty strange and seemingly pointless – don't worry, keep going with the flow. Let π be a permutation of indices such that:

$$0 \leq \overline{y}_{\pi(1)} \leq \ldots \leq \overline{y}_{\pi(n)}$$

In other words, π is a permutation that sorts $\overline{y}$ in ascending order. Consider now the following integral:

$$\int_0^{\overline{y}_{\pi(n)}} |S(r)| \, dr$$

We notice that, for each $r \in \mathbb{R}$ such that $0 \leq r \leq \overline{y}_{\pi(1)}$ it holds that $|S(r)| = n$. Similarly, for each $r \in R$ such that $\overline{y}_{\pi(1)} < r \leq \overline{y}_{\pi(2)}$ it holds that $|S(r)| = n-1$, and so on (see Figure 2.11 below). Hence, we have that:

$$\begin{aligned}
\int_0^{\overline{y}_{\pi(n)}} |S(r)| \, dr &= \int_0^{\overline{y}_{\pi(1)}} |S(r)| \, dr + \int_{\overline{y}_{\pi(1)}}^{\overline{y}_{\pi(2)}} |S(r)| \, dr + \ldots + \int_{\overline{y}_{\pi(n-1)}}^{\overline{y}_{\pi(n)}} |S(r)| \, dr \\
&= \int_0^{\overline{y}_{\pi(1)}} n \, dr + \int_{\overline{y}_{\pi(1)}}^{\overline{y}_{\pi(2)}} n-1 \, dr + \ldots + \int_{\overline{y}_{\pi(n-1)}}^{\overline{y}_{\pi(n)}} 1 \, dr \\
&= n(\overline{y}_{\pi(1)} - 0) + (n-1)(\overline{y}_{\pi(2)} - \overline{y}_{\pi(1)}) + \ldots + (\overline{y}_{\pi(n)} - \overline{y}_{\pi(n-1)}) \\
&= \overline{y}_{\pi(1)}(n - (n-1)) + \ldots + \overline{y}_{\pi(n)}(2-1) \\
&= \overline{y}_{\pi(1)} + \ldots + \overline{y}_{\pi(n)} \\
&= y'_1 + \ldots + y'_n \\
&\leq 1
\end{aligned}$$

Proceeding in a similar way, we get that:

$$\int_0^{\overline{y}_{\pi(n)}} |E(r)| \, dr = \sum_{\{i,j\} \in E(G)} x_{i,j} \geq v$$

Claim 3: there is an $r \in \mathbb{R}$ with $0 \leq r \leq \overline{y}_{\pi(n)}$ such that $|E(r)| \geq v \, |S(r)|$

Proof. Proof of Claim 3. By way of contradiction, suppose that for all $r \in \mathbb{R}$ with $0 \leq r \leq \overline{y}_{\pi(n)}$ it holds that $|E(r)| < v\,|S(r)|$. Then, the two previous integrals are bounded in the following way:

$$v \leq \int_0^{\overline{y}_{\pi(n)}} |E(r)|\ dr < \int_0^{\overline{y}_{\pi(n)}} v\,|S(r)|\ dr \leq v$$

raising a contradiction. □

Let r^* be a value satisfying Claim 3. Then, we have that:

$$\rho(S(r^*)) = \frac{|E(G[S(r^*)])|}{|S(r^*)|} = \frac{|E(r^*)|}{|S(r^*)|} \geq v$$

□

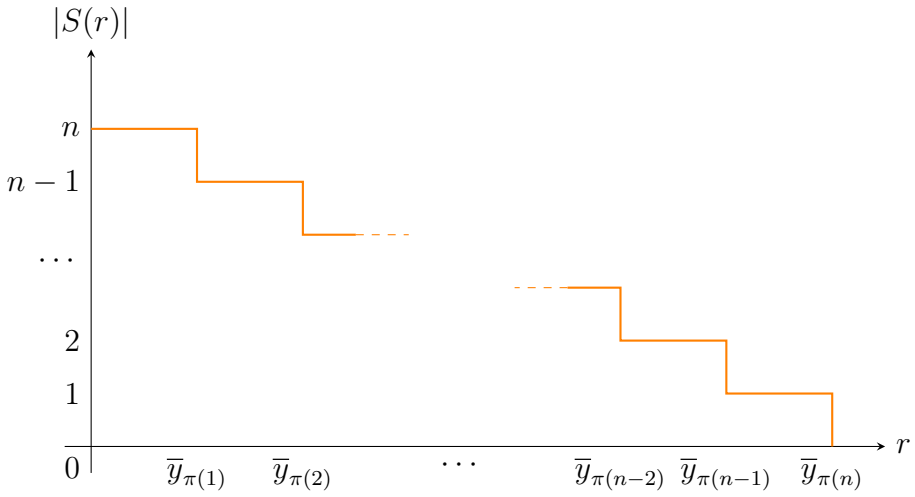


Figure 2.11: Plot of the value $|S(r)|$ with increasing values of r

Theorem 2.4: Integrality gap of DS

$\text{IG}_{\text{DS}} = 1$

Proof. Follows from Lemma 2.4 and Lemma 2.5. □

We observe that the above theorem implies only that the optimal value of the IP and the LP relaxation is equal, not that every solution to the LP is an maximum densest subgraph. In fact, the optimal solution of the LP found by the Ellipsoid method or the Simplex method may be fractional: every optimal integral solution is also an optimal linear solution, but not vice versa. Hence, we still need to apply some rounding procedure. Luckily, the proof of Lemma 2.5 implicitly gives us a perfect rounding procedure, one for which the output value is perfectly optimal and non-approximated.

Algorithm 2.2 Charikar's optimal program for the DS problem

Input: an undirected graph G
Output: the densest subgraph of G

1: **function** DENSEST-SUBGRAPH(G)
2: $\overline{x}, \overline{y} \leftarrow \text{LP}^*_{\text{DS}}(G)$
3: $S \in \arg\max_{i \in [n]} \rho(S(\overline{y}_i))$
4: Return $G[S]$
5: **end function**

Even thought this program is optimal, polynomial-time and deterministic, its runtime is actually $O(n^9)$, making it **unusable** for real applications such as spam-network detection, where the input graph has more than 10^24 vertices.

2.3 Approximations through duality

As we discussed, LPs can be solved in polynomial time through the Ellipsoid method. However, such polynomial is way too high to make it practical. For instance, Charikar's optimal algorithm for the DS problem has a runtime of $O(n^9)$, making this approach impractical for large graphs. To fix this issue, Charikar [Cha00] developed a greedy $\frac{1}{2}$-approximation for the problem that actually runs in $O(n)$. The idea used by Charikar is based on the concept of **linear program duality**.

Consider a maximization problem in standard matricial form (see Section 2.1)

$$\begin{aligned} \max & \, c^T x \\ Ax &\leq b \\ x &\geq 0 \\ x &\in \mathbb{R}^n \end{aligned}$$

The **dual** of such problem is the linear program defined as:

$$\begin{aligned} \max & \, b^T y \\ A^T y &\geq c \\ y &\geq 0 \\ y &\in \mathbb{R}^m \end{aligned}$$

while the original program is referred to as the **primal**.

$$\max \begin{bmatrix} 2 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$
$$\begin{bmatrix} 4 & 8 \\ 2 & 1 \\ 3 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 12 \\ 3 \\ 4 \end{bmatrix}$$
$$x_1, x_2 \geq 0$$

$$\min \begin{bmatrix} 12 & 3 & 4 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$
$$\begin{bmatrix} 4 & 2 & 3 \\ 8 & 1 & 2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \geq \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$
$$y_1, y_2, y_3 \geq 0$$

Figure 2.12: Example of a primal program (left) and its dual program (right).

We observe that the dual of the dual program is the primal program. Duality plays a fundamental role in linear programming and algorithm design. For the former, the importance comes from the **Weak Duality Theorem**.

Theorem 2.5: Weak Duality Theorem

Let (P, D) be a pair of primal-dual programs. If both P and D are feasible then for any feasible solution $\overline{x}$ of P and any feasible solution $\overline{y}$ of D it holds that $c^T\overline{x} \leq b^T\overline{y}$.

Proof. By feasibility of $\overline{x}$ and $\overline{y}$, we know that $A\overline{x} \leq b$ and $A^T\overline{y} \geq c$. Hence, we get that:

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

□

The Weak Duality Theorem implies that any solution of the primal program is bounded by the dual program. When both the primal and dual program have a feasible solution, the Weak Duality Theorem can be used to efficiently find the optimal solution of the primal and the dual. But there's more to it: the primal and dual program are intrinsically related to each other. This powerful result is expressed through the **Strong Duality Theorem**.

Theorem 2.6: Strong Duality Theorem

Let (P, D) be a pair of primal-dual programs. Then, exactly one of the following holds:

1. Both P and D are infeasible
2. P is feasible but unbounded and D is infeasible
3. P is infeasible and D is feasible but unbounded
4. Bot P and D are feasible, bounded and they share the same optimal value.

Proof. Omitted. □

The fourth case of the Strong Duality Theorem directly implies that. when the dual program is easier than the primal, we can solve the primal by solving the dual. This deep connection between primal and dual programs is often used for algorithm design, mostly in approximation algorithms: if the dual is easier to approximate, we're good to go.

This powerful idea is the core of many approximation algorithms. In fact, we observe that we have already secretly used duality in some of the previous algorithms, in particular the 2-approximation of the minimum vertex cover problem. In that approximation, we proved that each matching is upper bounded by each vertex cover. This result immediately follows from the weak duality theory. In fact, the minimum vertex cover problem is actually the dual of the maximum matching problem.

$$\begin{aligned} \max \quad & \sum_{\{i,j\} \in E(G)} x_{i,j} \\ & \sum_{\{i,j\} \in E(G)} x_{i,j} \qquad \forall i \in V(G) \\ & x \geq 0 \\ & x \in \mathbb{R}^m \end{aligned} \qquad \begin{aligned} \min \quad & \sum_{i \in V(G)} y_i \\ & y_i + y_j \qquad \forall \{i,j\} \in E(G) \\ & y \geq 0 \\ & y \in \mathbb{R}^n \end{aligned}$$

Figure 2.13: The primal maximum matching problem (left) and the dual minimum vertex cover problem (right).

Consider now the normalized LP relaxation for the densest subgraph problem.

$$\begin{aligned} \max \quad & \sum_{\{i,j\} \in E(G)} x_{i,j} \\ & x_{i,j} - y_i \leq 0 \qquad \forall \{i,j\} \in E(G) \\ & x_{i,j} - y_j \leq 0 \qquad \forall \{i,j\} \in E(G) \\ & \sum_{i \in V(G)} y_i \leq 1 \\ & x, y \geq 0 \\ & x \in \mathbb{R}^m \\ & y \in \mathbb{R}^n \end{aligned}$$

Figure 2.14: The densest subgraph problem.

The dual program is given by:

$$\min\ \delta$$

$$\begin{aligned} \gamma_{i,j} + \gamma_{j,i} &\geq 1 && \forall \{i,j\} \in E(G) \\ \delta - \sum_{\substack{j \in V(G) \\ \text{s.t. } \{i,j\} \in E(G)}} \gamma_{i,j} &\geq 0 && \forall i \in V(G) \\ \gamma, \delta &\geq 0 \\ \gamma &\in \mathbb{R}^m \\ \delta &\in \mathbb{R} \end{aligned}$$

Figure 2.15: The dual program of the densest subgraph problem.

After rewriting it in a more convenient way, Charikar observed that the dual program corresponds to the **Minimum Max-in-degree Orientation problem**, i.e. the problem that asks to find an orientation of the edges in an undirected graph whose maximum in-degree is as low as possible.

$$\min\ \delta$$

$$\begin{aligned} \gamma_{i,j} + \gamma_{j,i} &\geq 1 && \forall \{i,j\} \in E(G) \\ \delta &\geq \deg(i) && \forall i \in V(G) \\ \gamma, \delta &\geq 0 \\ \gamma &\in \mathbb{R}^m \\ \delta &\in \mathbb{R} \end{aligned}$$

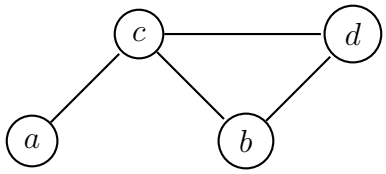
Figure 2.16: The Minimum Max-in-degree Orientation problem.

In fact, the first claim of the proof of Theorem 2.7 is directly implied by the Weak Duality Theorem, whole the orientation ϕ defined inside of it is always a solution to the dual problem. Hence, the real idea behind Charikal's $\frac{1}{2}$-approximation is to yield a solution to the dual program, hence an orientation, whose value is upper bounded by twice the density of the densest set. First, we give a formal definition of edge orientation.

Definition 2.2: Edge orientation

Let G be an undirected graph. An **edge orientation** on G is a function $\phi : E \to V$ such that $\phi(\{u,v\}) \in \{u,v\}$ for all $\{u,v\} \in E(G)$. If $\phi(\{u,v\}) = u$ then the edge $\{u,v\}$ gets oriented towards u, otherwise it gets oriented towards v.

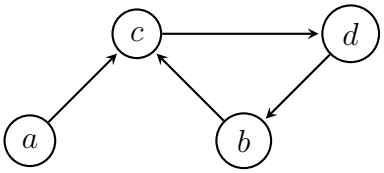
For instance, given the following graph:



and the following edge orientation ϕ:

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

we get the following directed graph:



We observe that every graph has $2^{|E(G)|}$ possible orientation. We denote with $\deg_\phi(v)$ the in-degree of a vertex v over an orientation ϕ, while Δ_ϕ denotes the maximum degree under the orientation.

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

We observe that, similarly to the Handshaking Lemma, we have that:

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

After analyzing all the details and relations, Charikar came up with the following $\frac{1}{2}$-approximation.

Algorithm 2.3 Charikar's $\frac{1}{2}$-approximation for DS

Input: an undirected graph G
Output: a subset of vertices

1: **function** $\frac{1}{2}$-APPROX-DS(G)
2: $S_0 \leftarrow V(G)$
3: **for** $i \in [n]$ **do**
4: Find $v_i \in \arg\min_{v \in S_{i-1}} \deg_{G[S_{i-1}]}(v)$
5: $S_i \leftarrow S_{i-1} - \{v\}$
6: **end for**
7: Return $S \in \arg\max_{i \in [n]} \rho(S_i)$
8: **end function**

Theorem 2.7

Given a graph G, let S^* be an optimal solution to DS(G). Given the output S of $\frac{1}{2}$-APPROX-DS(G), it holds that $|S| \geq \frac{1}{2}|S^*|$.

Proof. Since the orientation problem is the dual of the densest subgraph, by the weak duality theorem we expect any orientation to give an upper bound on the density. In fact, the following claim holds.

Claim 1: for any orientation ϕ on G it holds that:

$$\max_{S' \subseteq V(G)} \rho(S') \leq \Delta_\phi$$

Proof. Since any orientation can be restricted to an orientation over any induced subgraph of G, we get that:

$$\max_{S' \subseteq V(G)} \frac{|E(G[S'])|}{|S'|} = \max_{S' \subseteq V(G)} \frac{\left|\sum_{v \in S'} \deg_{\phi, G[S']}\right|}{|S'|} \leq \frac{\left|\sum_{v \in S'} \Delta_\phi\right|}{|S'|} = \Delta_\phi$$

□

Let $S_0, \ldots, S_n$ and $v_1, \ldots, v_n$ respectively be the subsets and vertices defined inside $\frac{1}{2}$-APPROX-DS(G). Consider the orientation ϕ on G defined as follows: for each $i \in [n]$ and for each edge $e \in G[S_{i-1}]$, if $v_i \in e$ then $\phi(e) = v_i$. In other words, ϕ is the orientation on G such that each edge incident to the vertex v_i removed in the i-th iteration gets oriented towards v_i.

Claim 2: for each $i \in [n]$ it holds that $\deg_\phi(v_i) \leq 2\rho(S_{i-1})$

Proof. By construction, we have that $\deg_\phi(v_i) = \deg_{G[S_{i-1}]}(v_i)$. Then, by choice of v_i inside the algorithm, we have that:

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

By the Handshaking lemma, we conclude that:

$$\deg_{G[S_{i-1}]}(v_i) \leq \frac{\sum_{v \in S_{i-1}} \deg_{G[S_{i-1}]}(v)}{|S_{i-1}|} = \frac{2\,|E(G[S_{i-1}])|}{|S_{i-1}|}$$

□

Through the two claims, we conclude that:

$$\rho(S^*) \leq \Delta_\phi = \max_{i \in [n]} \deg_\phi(v_i) \leq \max_{i \in [n]} 2\rho(S_{i-1}) = 2\rho(S)$$

□

2.4 Approximation and runtime trade-offs

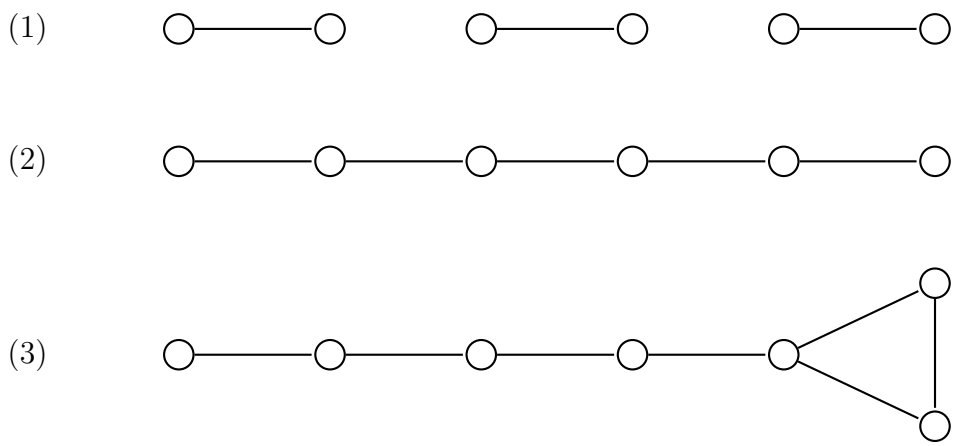
We showed how Charikar came up with an exact solution to the DS problem that runs in $O(n^9)$ and a $\frac{1}{2}$-approximation that is way faster. However, the number of iterations this approximation are still too many: we're iterating over n nodes and each iteration requires $O(n+m)$ steps, making the total cost $O(n(n+m))$. For real-life purposes, this runtime is still too much – for instance, Google could never use such algorithm to detect spam-networks since the underlying graph contains more than 10^30 nodes. What if we modify the algorithm in order to make it remove a all the nodes of minimum degree instead of a single node at a time?

Algorithm 2.4 Charikar's $\frac{1}{2}$-approximation for DS (2nd version)

Input: an undirected graph G
Output: a subset of vertices

1: **function** $\frac{1}{2}$-APPROX-DS(G)
2: $S_0 \leftarrow V(G)$
3: **for** $i \in [n]$ **do**
4: Find $A_i = \{v \in S_{i-1} \mid \deg_{G[S_{i-1}]}(v) = \min_{u \in S_{i-1}} \deg_{G[S_{i-1}]}(u)\}$
5: $S_i \leftarrow S_{i-1} - A_i$
6: **end for**
7: Return $S \in \arg\max_{i \in [n]} \rho(S_i)$
8: **end function**

Even though this second version works better on average, there are still some instances for which it strictly requires almost n iterations. For instance, consider the three graphs in the figure below. For the first graph, the new version requires only one iteration since every vertex has the same degree. For the second graph, it requires $\frac{n}{2}$ iterations, while the third graph requires $n-2$ iterations.



We observe that the proof of Charikar's doesn't require the selected vertices to be the ones with minimum degree: we require the vertices to be just below the average degree.

Algorithm 2.5 Charikar's $\frac{1}{2}$-approximation for DS (3nd version)

Input: an undirected graph G
Output: a subset of vertices

function $\frac{1}{2}$-APPROX-DS(G)
  $S_0 \leftarrow V(G)$
  for $i \in [n]$ **do**
    Find $A_i = \{v \in S_{i-1} \mid \deg_{G[S_{i-1}]}(v) \leq \text{avg}_{u \in S_{i-1}} \deg_{G[S_{i-1}]}(u)\}$
    $S_i \leftarrow S_{i-1} - A_i$
  end for
  Return $S \in \arg\max_{i \in [n]} \rho(S_i)$
end function

We observe however, that this constraint is still "too strict": the third graph now requires 2 iterations, while the second graph still requires $\frac{n}{2}$ iterations since only the endpoints are below the average degree (which has value $\frac{10}{6} = 1.\overline{6}$). To fix this, we can make the boundary more flexible by adding a small bias factor of $(1 + \varepsilon)$ to the selection criteria, where $\varepsilon > 0$. This solutions was proposed by Bahmani, Kumar, and Vassilvitskii [BKV12], obtaining a solid upper bound on the number of iterations, with a small loss on the approximation rate. With this final version, we observe that even a small value such as $\varepsilon = 0.4$ is sufficient to make the algorithm run in 1 iteration even for the second case.

Algorithm 2.6 $\frac{1}{2(1+\varepsilon)}$-approximation for DS

Input: an undirected graph G
Output: a subset of vertices

function $\frac{1}{2(1+\varepsilon)}$-APPROX-DS($G$)
  $S_0 \leftarrow V(G)$
  for $i \in [n]$ **do**
    Find $A_i = \{v \in S_{i-1} \mid \deg_{G[S_{i-1}]}(v) \leq (1+\varepsilon)\,\text{avg}_{u \in S_{i-1}} \deg_{G[S_{i-1}]}(u)\}$
    $S_i \leftarrow S_{i-1} - A_i$
  end for
  Return $S \in \arg\max_{i \in [n]} \rho(S_i)$
end function

Theorem 2.8

Given a graph G, let S^* be an optimal solution to DS(G). Given the output S of $\frac{1}{2(1+\varepsilon)}$-APPROX-DS($G$), for all $\varepsilon > 0$ it holds that $|S| \geq \frac{1}{2(1+\varepsilon)} |S^*|$. Moreover, the algorithm $\frac{1}{2(1+\varepsilon)}$-APPROX-DS does at most $O(\log_{1+\varepsilon} n)$ iterations.

Proof. Let S^* be the optimal solution. If $\rho(S^*) = 0$ then $E(G) = \varepsilon$, so any solution is optimal. Suppose now that $\rho(S^*) > 0$. Then, $E(G) \neq \varepsilon$, implying that $|S^*| \geq 2$. This observation allows us to claim the following.

Claim: for all $v \in S^*$ it holds that $\rho(S^*) \leq \deg_{S^*}(v)$.

Proof of the claim. Fix a vertex $v \in S^*$. By optimality of S^*, we know that $\rho(S^*) \geq \rho(S^* - \{v\})$. Moreover, we observe that:

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

Hence, we get that:

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

The claim follows by solving the inequality. □

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:

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

Hence, the algorithm will eventually select a vertex of S^*. Let i be the first iterations such that $A_i \cap S^* \neq \varepsilon$. Moreover, since i is the smallest iteration with such property, it must hold that $S^* \subseteq S_{i-1}$. Hence, given any vertex $v_i \in A_i \cap S^*$ it holds that:

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

Thus, we conclude that:

$$\rho(S_{i-1}) \geq \frac{1}{2(1+\varepsilon)} \rho(S^*)$$

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

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

Consider now a generic iteration $k \in [n]$. We observe that:

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

Thus, we get that:

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

Let i^* be the last iteration of the algorithm. Since $|S_0| = n$ and since i^* is reached when $|S_{i^*}| = 1$, we get that:

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

□

2.5 Approximations through Semidefinite programming

In Section 1.2.1, we briefly mentioned how the best known approximation ratio for the Max-Cut problem is around 0.878. After developing all the standard tools for approximations through linear programs, we're finally ready to discuss such ratio. First, let's take a step back to were we started. We gave a $\frac{1}{2}$-approximation for Max-Cut that doesn't even care about the structure of the graph itself: the algorithm simply flips a coin and decides if the corresponding vertex has to be added or not.

Using the new tools that we know, we start by formulating an integer program that exactly models the Max-Cut problem. For each edge $\{i,j\}$ we define a variable $y_{i,j}$, while for each vertex v we define the variable x_v.

$$\max \sum_{\{i,j\}\in E(G)} y_{i,j}$$

$$\begin{aligned} y_{i,j} &\leq x_i + x_j && \forall \{i,j\} \in E(G) \\ y_{i,j} &\leq 2 - (x_i + x_j) && \forall \{i,j\} \in E(G) \\ x &\in \{0,1\}^n \\ y &\in \{0,1\}^m \end{aligned}$$

Figure 2.17: Integer program for the Max-Cut problem.

However, we observe that in the LP relaxation of such program the optimal solution will always set every single vertex variable x_v to $\frac{1}{2}$ and every edge variable $y_{i,j}$ to 1. In other words, the optimal solution to the LP relaxation gives us no information at all! In fact, this is the reason why $\frac{1}{2}$-approximation can just flip coins for every vertex without considering the structure of the graph. But how good can this relaxation get? We already know that our algorithm gives implies 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 some $\varepsilon > 0$, concluding that $\mathrm{IG}_{\mathrm{MC}} = 2$ holds for this program.

Since the exact LP relaxation always gives a trivial solution with a gap of 2, researchers tried for many years to improve the gap by considering variants of the program, introducing constraints that may improve the gap. However, each try failed.

$$\max \sum_{\{i,j\} \in E(G)} y_{i,j}$$

$$\begin{array}{rl} y_{i,j} \leq x_i + x_j & \forall \{i,j\} \in E(G) \\ y_{i,j} \leq 2 - (x_i + x_j) & \forall \{i,j\} \in E(G) \\ y_{i,j} + y_{j,k} + y_{k,i} \leq 2 & \forall \{i,j,k\} \in \binom{V(G)}{3} \text{ s.t. } i \sim j \sim k \\ x \in \{0,1\}^n & \\ y \in \{0,1\}^m & \end{array}$$

Figure 2.18: Variant of the IP with an additional "triangle constraint", which forces the solution to have at most 2 edges for each K_3 inside the graph.

In the '90s, many results proved that even by adding a polynomial amount of constraints to the original problem, each with a constant number of variables, while keeping it a Max-Cut relaxation, the integrality gap is still lower bounded by $2 - \varepsilon$ for any $\varepsilon > 0$, making researchers believe that no improvement could be achieved.

Surprisingly, Goemans and Williamson [GW95] proved in a ground breaking result that this gap can indeed be improved, but only by completely changing the approach: instead of using an integer linear program, they modeled the Max-Cut problem through a **Quadratic Program (QP)**.

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

$$\begin{array}{c} Ax \leq b \\ x \in V \end{array}$$

Figure 2.19: Standard form of a quadratic program.

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

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

$$x \in \{1, -1\}^n$$

Figure 2.20: Quadratic program for the Max-Cut problem.

We observe that for any edge $\{i, j\}$ we have that:

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

Nonetheless, solving quadratic programs is also NP-Hard. However, the genius idea of Goemans and Williamson involved *casting* such quadratic program into a **Semidefinite Program (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 2.21: Standard form of a semidefinite program.

We observe that differently from linear and quadratic programs, SDPs work on *vector variables*, where $\langle v_i, v_j \rangle$ denotes the dot product. Like LPs, SDPs can also be solved in polynomial time through an extension of the Ellipsoid method., making them a good tool for algorithm design. In particular, they are mostly used for *computational geometry*.

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

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

Figure 2.22: Goemans and Williamson's SDP relaxation for the Max-Cut problem.

The correctness of this relaxation comes in a pretty natural way. Let $\overline{x}$ be a feasible solution to the QP. If we map each component $\overline{x}_i$ with the vector $v_i = \begin{bmatrix}\overline{x}_i & 0 & \dots & 0\end{bmatrix}^T$, we observe that:

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

Moreover, the constraint $\langle v_i, v_i \rangle = 1$ is always satisfied:

$$\langle v_i, v_i \rangle = \sum_{k \in [n]} v_{k,i}^2 = \overline{x}_i^2 = 1$$

This concludes that $\text{SDP}^*_{\text{MC}} \geq \text{QP}^* = \text{OPT}_{\text{MC}}$. The algorithm proposed by the two authors, however, is not so natural. In fact, it looks pretty much like *black magic*.

Algorithm 2.7 GW 0.878-approximation for MC

Input: an undirected graph G
Output: a cut of G

1: **function** 0.878-APPROX-MC(G)
2: $\{v_1, \dots, v_n\} \leftarrow \text{SDP}_{\text{MC}}(G)$
3: $S_n \leftarrow \{x \in \mathbb{R}^n \mid ||x|| = 1\}$ ▷ S_n is the n-dimensional hypersphere
4: Sample a uniform-at-random vector y from S_n
5: $S \leftarrow \{i \mid \langle v_i, y \rangle \geq 0\}$
6: Return $(S, \overline{S})$
7: **end function**

Theorem 2.9

Given a graph G, let (S^*, T^*) be an optimal solution to MC(G). Given the output (S, T) of 0.878-APPROX-MC(G), it holds that

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

where:

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

Proof. (Sketch). A formal proof of this claim would require advanced analytical arguments that are out of the scope of this course. We give an informal intuition behind how and why the algorithm works by focusing on the 2D case. First, we observe that the dot product between two vectors u, w can be geometrically described as:

$$\langle u, w \rangle = ||u|| \cdot ||w|| \cdot \cos(\theta_{u,w})$$

where $\theta_{u,w}$ is the angle between the two vectors. Since each $v_1, \dots, v_n$ has to satisfy the constraint $\langle v_i, v_i \rangle = 1$, we know that:

$$1 = \langle v_i, v_i \rangle = ||v_i||^2 \implies 1 = ||v_i||$$

Hence, since the vector y is sampled from S_n, we get that $\langle v_i, y\rangle = \cos(\theta_{v_i,y})$. We also observe that for any angle θ it holds that:

$$\cos(\theta) \geq 0 \iff 2k\pi - \frac{\pi}{2} \leq \theta \leq 2k\pi + \frac{\pi}{2}$$

Geometrically, this describes the fact that the two vertices are *close enough* (see Figure 2.23). In other words, the set S contains vectors that are similar to each other (up to some degree of freedom). This vector-comparison metric is often used in Machine Learning.

Claim 1: $\Pr[\{i,j\} \in \text{cut}(S, \overline{S})] = \dfrac{\theta_{v_i,v_j}}{\pi}$

Proof of Claim 1. For each $k \in [n]$, let H_k be the 1-dimensional hyperplane passing through the origin and perpendicular to v_k. This hyperplane splits S_2 in half: one half contains all the vectors x such that $\cos(\theta_{v_k,x}) \geq 0$, while all the others lie on the other half.

Consider now the two hyperplanes H_i, H_j and Figure 2.24. We notice that two hyperplanes induce a another split of S_2 into four subspaces:

1. The green sector contains all the vectors x such that $\cos(\theta_{v_i,x}) \geq 0$ and $\cos(\theta_{v_j,x}) \geq 0$
2. The blue sector contains all the vectors x such that $\cos(\theta_{v_i,x}) \geq 0$ and $\cos(\theta_{v_j,x}) < 0$
3. The yellow sector contains all the vectors x such that $\cos(\theta_{v_i,x}) < 0$ and $\cos(\theta_{v_j,x}) \geq 0$
4. The white sector contains all the vectors x such that $\cos(\theta_{v_i,x}) < 0$ and $\cos(\theta_{v_j,x}) < 0$

We also observe that the angle describing the yellow and blue sectors is exactly equal to θ_{v_i,v_j}. Hence, the portion of the sphere for both sectors is given by $\frac{\theta_{v_i,v_j}}{2\pi}$, implying that:

$$\begin{aligned}
\Pr[\{i,j\} \in \text{cut}(S, \overline{S})] &= \Pr[i \in S, j \notin S \vee i \notin S, j \in S] \\
&= \Pr[i \in S, j \notin S] + \Pr[i \notin S, j \in S] \\
&= \frac{\theta_{v_i,v_j}}{2\pi} + \frac{\theta_{v_i,v_j}}{2\pi}
\end{aligned}$$

□

Through Claim 1, we get that:

$$\begin{aligned}
\mathbb{E}[|\text{cut}(S,T)|] &= \sum_{\{i,j\} \in E(G)} \Pr[\{i,j\} \in \text{cut}(S, \overline{S})] \\
&= \sum_{\{i,j\} \in E(G)} \frac{\theta_{v_i,v_j}}{\pi} \\
&= \sum_{\{i,j\} \in E(G)} \frac{\arccos(\langle v_i, v_j \rangle)}{\pi}
\end{aligned}$$

Claim 2: if $\alpha \in \mathbb{R}$ is a value such that $\alpha \leq \frac{2}{\pi}\frac{\arccos(x)}{1-x}$ for all $x \in (-1,1)$ then

$$\mathbb{E}[|\text{cut}(S,T)|] \geq \alpha \text{SPD}^*_{MC}(G)$$

Proof of Claim 2. Through some algebraic manipulation we get that:

$$\frac{\arccos(x)}{\pi} \geq \alpha \left(\frac{1-x}{2}\right)$$

Hence, since $-1 \leq \langle v_i, v_j \rangle \leq 1$, it holds that:

$$\mathbb{E}[|\text{cut}(S,T)|] \geq \sum_{\{i,j\} \in E(G)} \frac{\arccos(\langle v_i, v_j \rangle)}{\pi} \geq \alpha \sum_{\{i,j\} \in E(G)} \left(\frac{1 - \langle v_i, v_j \rangle}{2}\right) = \alpha \text{SPD}^*_{MC}(G)$$

□

In order to satisfy Claim 2, it's sufficient to consider the value α_{GW} defined as:

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

Moreover, since the SDP is a relaxation of the QP, we get that:

$$\mathbb{E}[|\text{cut}(S,T)|] \geq \alpha_{\text{GW}} \text{SPD}^*_{MC}(G) \geq \alpha_{\text{GW}} \text{QP}^*_{MC}(G)$$

□

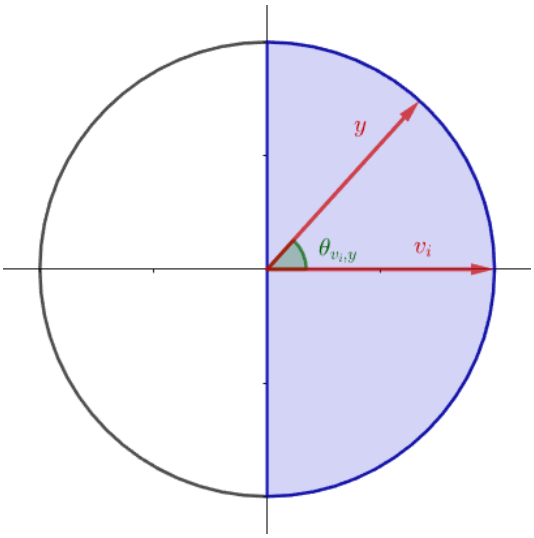


Figure 2.23: If y falls in the blue sector then $\cos(\theta_{v_i,y}) \geq 0$, hence $i \in S$.

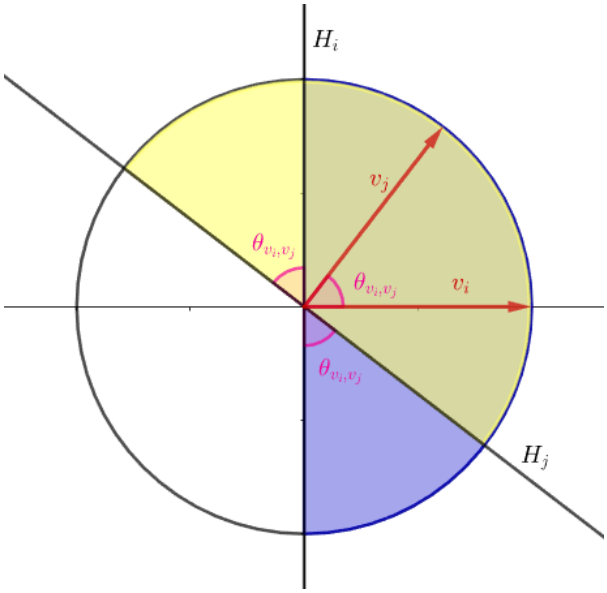


Figure 2.24: Partition of S_2 into the four subspaces induced by H_i, H_j.

We observe that the approximation ratio α_{GW} for the MC problem is actually *tight*. In fact, it can be proven that $\text{IG}^{\text{SDP}}_{\text{GW}} = \alpha_{\text{GW}}$: the above proof of the algorithm implicitly gives the lower bound, while the upper bound is given by the graph C_n. For instance, consider the following C_5 graph.

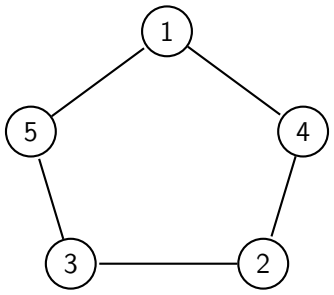


Figure 2.25: The graph C_5 giving the lower bound for $\text{IG}^{\text{SDP}}_{\text{MC}}$.

We observe that the order of the indices labeling such vertices is not arbitrary, but carefully chosen in order to make the following computations easier. First, we notice that the cut $(\{1,2\},\{3,4,5\})$ is optimal for this graph, meaning that $\text{OPT}_{\text{MC}}(C_5) = 4$. We'll now construct a feasible solution to the SDP program in order to get a lower bound on SDP*. For every $j \in [n]$, we set:

$$v_j = \begin{bmatrix} \cos\left(\frac{2j\pi}{5}\right) & \sin\left(\frac{2j\pi}{5}\right) & 0 & 0 & 0 \end{bmatrix}$$

It's easy to see that every vertex satisfies the constraint of the SDP:

$$\langle v_j, v_j \rangle = \cos^2\left(\frac{2j\pi}{5}\right) + \sin^2\left(\frac{2j\pi}{5}\right) = 1$$

Moreover, using the same reasoning as the previous proof we know that:

$$\sum_{\{i,j\}\in E(C_5)} \frac{1-\langle v_i, v_j\rangle}{2} = \sum_{\{i,j\}\in E(C_5)} \frac{1-\cos(\theta_{v_i,v_j})}{2}$$

Now, we observe that the angle between each pair v_i, v_j such that $\{i,j\} \in E(C_5)$ is exactly $\frac{4\pi}{5}$ (see **??**). Hence, we get that:

$$\sum_{\{i,j\}\in E(C_5)} \frac{1-\langle v_i, v_j\rangle}{2} = \sum_{\{i,j\}\in E(C_5)} \frac{1-\cos\left(\frac{4\pi}{5}\right)}{2} = \frac{5(1-\cos\left(\frac{4\pi}{5}\right))}{2}$$

Thus, the integrality gap for C_5 corresponds to:

$$\mathrm{IG}_{\mathrm{MC}}^{\mathrm{SDP}} = \frac{4}{\frac{5(1-\cos\left(\frac{4\pi}{5}\right))}{2}} = 0.884\ldots$$

concluding that $0.878 \approx \alpha_{\mathrm{GW}} \leq \mathrm{IG}_{\mathrm{MC}}^{\mathrm{SDP}} \leq \mathrm{IG}_{\mathrm{MC}}^{\mathrm{SDP}}(C_5) \approx 0.884$.

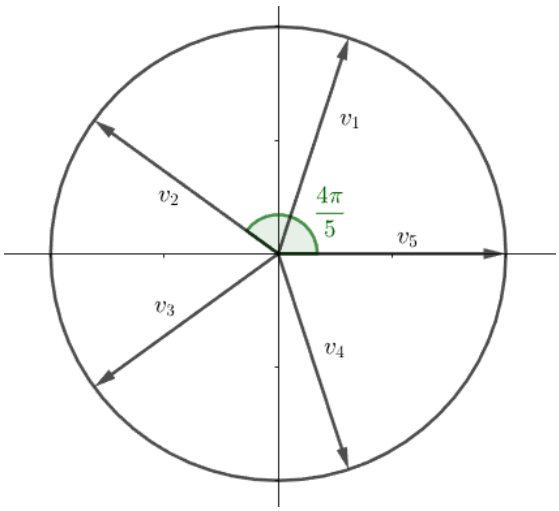


Figure 2.26: The vectors $v_1, \ldots, v_5$ actually describe the 5 roots of unity. For C_n, the argument is generalized to the n roots of unity.

Theorem 2.10: Integrality gap of MC with SDP

$\mathrm{IG}_{\mathrm{MC}}^{\mathrm{SDP}} = \alpha_{\mathrm{GW}} \approx 0.878$

Hence, we conclude that Goemans and Williamson's algorithm is the best rounding algorithm that can be achieved through SDPs. Moreover, researchers believe that this approximation ratio is the best that can be achieved for the MC problem over all techniques. In particular, the main tool of reasoning behind this result is the **Unique Games Conjecture**, which will be discussed in the following section.

2.6 The Unique Games Conjecture

Before diving into the conjecture, we have to define the concept of **Unique Label Cover**, first defined by Khot [Kho02]. We're given a bipartite graph with bipartition (A, B) and a set of k labels (also called *colors*). Each edge $e \in E(G)$ has a permutation $\pi_e : [k] \to [k]$. An ULC is an assignment ϕ that labels each vertex with a color. An edge $\{a, b\}$ of the graph is said to be *satisfied* by ϕ if the color $\phi(a)$ gets permuted into the color $\phi(b)$ by the permutation $\pi_{\{a,b\}}$.

Definition 2.3: Unique Label Cover

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

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

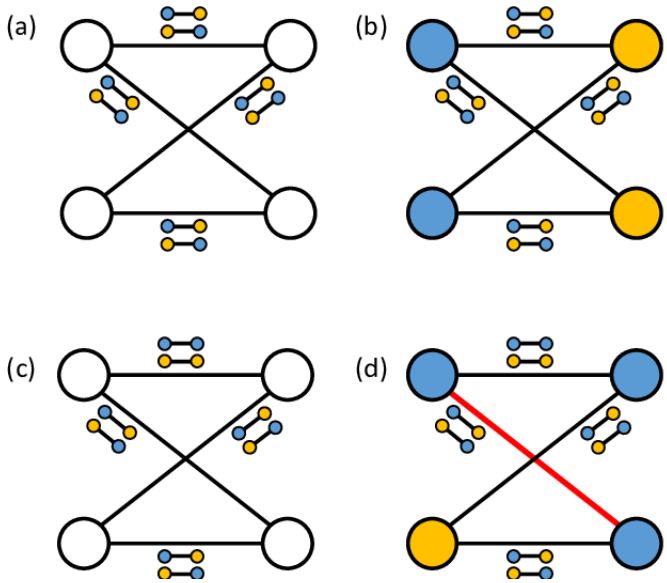


Figure 2.27: (a) and (c) are two instances of the UCL problem with 2 colors. In (b) is a solution to (a) that satisfies all the edges, while (d) is a solution to (c) with an unsatisfied edge.

We observe that the UCL instances are *strongly constrained* in the sense that the color of a vertex uniquely defines the colors of its neighbors and, by propagation, of its entire connected component. Thus, if the input instance admits a valid assignment, then such an assignment can be found efficiently by iterating over all 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 edges that are satisfiable by any assignment. For satisfiable instances, this value is 1 and is easy to find through the above procedure.

On the other hand, it seems to be very difficult to determine the value of an unsatisfiable game, even approximately. Khot's **Unique Games Conjecture** formalizes this difficulty in terms of NP-hardness.

Conjecture 2.1: Unique Games Conjecture

It is conjectured that for each $\varepsilon > 0$ there is a value k_ε for which it is NP-Hard to determine if for an UCL problem with k_ε labels one of the following holds:

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

Khot's conjecture has been proven to be intrinsically linked with **Constraint Satisfaction Problems (CSP)**, a particular subset of optimization problems. CSPs are defined in terms of a set of values $[q]$ and a set of *k-ary* predicates $\mathcal{P}$ on $[q]$ – the *arity* of a predicate on $[q]$ is a function $f : [q]^k \to \{0,1\}$. An instance of a CSP is given by a set of variables $x_1, \dots, x_n$ taking value over $[q]$ and a set of constraints, each taken from $\mathcal{P}$ and redefined over a subset of k variables $x_{i_1}, \dots, x_{i_k}$.

Definition 2.4: Constraint Satisfaction Problem (CSP)

Let $\mathcal{P}$ be a set of k-ary predicates defined on $[q]$, where $q, k \in \mathbb{N}$. 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 $P_j \begin{bmatrix} x_{i_1} & \cdots & x_{i_k} \\ \alpha(x_{i_1}) & \cdots & \alpha(x_{i_k}) \end{bmatrix}$ evaluates as true.

The goal of a CSP problem is to find an assignment maximizing the number of satisfied constraint. Lots of optimization problems, even those that we have already discussed, can be described in terms of CSP.

1. The *max-cut problem* can be described in terms of a CSP with value $q = 2$ and predicate set $\mathcal{P} = \{p(a,b)\}$ with arity 2, where $p(a,b) =$ "$a \neq b$". Given a graph G, for each node $x \in V(G)$ we define the variable x_v and for each edge $\{u,v\} \in E(G)$ we define the constraint $p(x_u, x_v)$
2. The *max k-coloring problem* asks to find a k-coloring of a graph that maximizes the number of well-colored edges, that being the edges whose endpoints have different color. This problem can be described in terms of a CSP with value $q = k$ and predicate set $\mathcal{P} = \{p(a,b)\}$ with arity 2, where $p(a,b) = \llbracket a \neq b \rrbracket$. Given a graph G, for each node $x \in V(G)$ we define the variable x_v and for each edge $\{u,v\} \in E(G)$ we define the constraint $p(x_u, x_v)$ – this is basically a generalization of the max-cut problem.
3. The *max 3-sat problem* asks to find an assignment maximizing the number of clauses

satisfied in a 3CNF formula. This problem can be described in terms of a CSP with value $q = 2$ and predicate set:

$$\begin{aligned}\mathcal{P} = \{&p_0(a,b,c) = \text{“}a \vee b \vee c\text{”}, p_1(a,b,c) = \text{“}a \vee b \vee \overline{c}\text{”},\\ &p_2(a,b,c) = \text{“}a \vee \overline{b} \vee c\text{”}, p_3(a,b,c) = \text{“}a \vee \overline{b} \vee \overline{c}\text{”},\\ &p_4(a,b,c) = \text{“}\overline{a} \vee b \vee c\text{”}, p_5(a,b,c) = \text{“}\overline{a} \vee b \vee \overline{c}\text{”},\\ &p_6(a,b,c) = \text{“}\overline{a} \vee \overline{b} \vee c\text{”}, p_7(a,b,c) = \text{“}\overline{a} \vee \overline{b} \vee \overline{c}\text{”}\}\end{aligned}$$

For instance, given the formula $F = (\overline{x_1} \vee x_2 \vee x_7) \wedge (\overline{x_3} \vee \overline{x_4} \vee \overline{x_{10}})$ we define the constraints $p_4(x_1, x_2, x_7)$ and $p_7(x_3, x_4, x_{10})$

Raghavendra [Rag08] used CSPs to show that the connection between approximation algorithms, SDPs and the Unique Games Conjecture is as deep as possible. In fact, if the conjecture is proven to be true, Raghavendra's theorem directly implies that the SDP integrality gap of every problem describable through a CSP is the best possible approximation ratio. Many researchers believe the conjecture to be true, but none has yet proven it. Given a problem describable through a CSP $\mathcal{C}$, we denote with $\alpha(\mathcal{C})$ the best possible approximation ratio for $\mathcal{C}$, i.e. the minimum ratio such that the problem doesn't become NP-Hard to approximate.

Theorem 2.11: Raghavendra's theorem

For every CSP $\mathcal{C}$ with values in $[q]$ and arity k, it holds that:

- There is an SDP with integrality gap of $\alpha(\mathcal{C})$ and a rounding algorithm that is an $\alpha(\mathcal{C})$-approximation of $\mathcal{C}$
- If the Unique Games Conjecture is true, it is NP-Hard to approximate $\mathcal{C}$ with a ratio $\alpha(\mathcal{C}) - \varepsilon$ for any $\varepsilon > 0$

Proof. Omitted. □

As a direct corollary of Raghavendra's theorem, we get that the previously found approximation ratios for max-cut and vertex cover are the best possible ones.

Corollary 2.1

If the Unique Games Conjecture is true then $\mathrm{IG}_{\mathrm{MC}}^{\mathrm{SDP}} = \alpha(\mathrm{MC})$ and $\mathrm{IG}_{\mathrm{VC}} = \alpha(\mathrm{VC})$

3

Metric Geometry

3.1 The Sparsest Cut problem

At this point, we have widely discussed the max-cut problem, showing that it is NP-Hard and arguing that the Goemans-Williamson 0.878-approximation is probably the best that can be achieved. Moreover, we also briefly mentioned that the min-cut problem can, instead, be solved in polynomial time through a reduction to the maximum flow problem, using the *Max-flow/Min-cut theorem*.

Now, we'll introduce a variant of the Min-cut problem: the *Sparsest Cut* problem. Given a graph G, the **Sparsest Cut** problem asks to find a non-empty subset of vertices S whose *sparsity* $\psi(S)$ is minimized, where:

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

In other words, the goal is to partition a given graph into two large pieces while removing as few edges as possible. The sparsest cut problem plays an important role in theory of network flow, geometric embeddings and form a crucial component of divide-and-conquer approaches in applications such as packet routing, VLSI layout and clustering.

A good tool of reasoning for the S-Cut problem is the **Erdős-Rényi Random Graph**. Given two values n, μ, where $n \in \mathbb{N}$ and $0 \leq \mu \leq 1$, the Erdős-Rényi graph, written as $G(n, \mu)$, is a graph with a fixed vertex set $V(G) = [n]$ and a probabilistic edge set, where each edge $e \in \binom{[n]}{2}$ has probability μ of being added to the edge set.

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

By definition, this model represents a "evenly sparse graph". In fact, we observe that for each non-empty subset $S \subseteq V(G)$ it holds that:

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

For general graphs, we observe that for each non-empty subset $S \subseteq V(G)$ it holds that:

$$0 \leq \left|\sqcup\!\sqcap\!\sqcup(S, \overline{S})\right| \leq |S| \cdot \left|\overline{S}\right|$$

implying that $0 \leq \psi(S) \leq 1$. In particular, we observe that $\psi(S) = 0$ holds when S is disconnected from $\overline{S}$, while $\psi(S) = 1$ when S is "fully connected" to $\overline{S}$, meaning that every edge of S is connected to every edge of $\overline{S}$.

Considering the set S^* inducing the minimal sparsity value $\psi(S^*)$, we have that $\psi(S^*) = 0$ when the whole graph G is disconnected, while $\psi(S^*) = 1$ when G is a complete graph, i.e. $G = K_n$.

Not-so surprisingly, the Max-Cut problem can be reduced to the S-Cut problem, implying that the latter is also NP-Hard. The problem can, however, be approximated through the *Leighton-Rao algorithm*. This algorithm is based on LP relaxation and a series of reduction. In particular, the integrality gap of such algorithm is bounded by *geometrical arguments*, in particular geometric embeddings. With the excuse of proving such result, we'll dive deeply into **metric geometry**.

3.2 Cut metrics

Intuitively, a **metric** (or *distance*) is a tool for measuring objects. In metric geometry, a metric is defined as a symmetric non-negative function from a set S to $\mathbb{R}$ that respects the triangle inequality, where any object in S has distance 0 from itself.

Definition 3.1: Metric

Let S be a set. A **metric** on S is a function $d : S \times S \to \mathbb{R}$ such that:

1. *Non-negativity*: for all $x, y \in S$ it holds that $d(x, y) \geq 0$
2. *Symmetry*: for all $x, y \in S$ it holds that $d(x, y) = d(y, x)$
3. *Self-distance*: for all $x \in S$ it holds that $d(x, x) = 0$
4. *Triangle inequality*: for all $x, y, z \in S$ it holds that $d(x, y) \leq d(x, z) + d(z, y)$

The typical example of metric is the *Euclidean distance*, i.e. $d(x, y) = |x - y|$. In the context of cuts, we're interested in *cut metrics*. In particular, given a graph G, for each subset $T \subseteq V(G)$ we define the **elementary cut metric** on T as the function $d_T : V(G) \times V(G) \to \mathbb{R}$ such that:

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

Proposition 3.1

Each elementary cut metric is a metric.

Proof. Given a graph G and a subset $T \subseteq V(G)$, consider the cut metric d_T. The first three axioms are trivially satisfied by the definition, hence we have to prove that the triangle inequality is also satisfied.

Fix three vertices $x, y, z \in V(G)$. We have three cases:

1. If $x, y, z \in T$ then:

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

2. If $x, y, z \in T$ then:

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

3. If $\exists A \in \{T, \overline{T}\}$ such that exactly one of x, y, z is in A, we have three sub-cases:
   (a) If $x \in A$ then:

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

   (b) If $y \in A$ then:

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

   (c) If $z \in A$ then:

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

□

We notice that, by definition, it always holds that $d_T(x, y) = d_{\overline{T}}(x, y)$. More importantly, we observe that the concept of elementary cut metric is deeply related to the concept of sparsity. In fact, we will show that the S-Cut problem is actually an optimization over elementary cut metrics. Given an elementary cut metric d_T, let $\phi(d_T)$ be the **cut-ratio** of T induced by d_T:

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

Proposition 3.2

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

Proof. Through some algebraic manipulation we have that:

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

□

Elementary cut metrics are generalized through the concept of **cut metric**, i.e. linear combinations over a set of elementary cut metrics.

Definition 3.2: Cut metric

Let G be a graph and let $d_{T_1}, \dots, d_{T_k}$ be elementary cut metrics on G. Given $\lambda_1, \dots, \lambda_k > 0$, a **cut metric** is a function $d : V(G) \times V(G) \to \mathbb{R}$ defined as:

$$d(x, y) = \sum_{i \in [k]} \lambda_i d_{T_i}(x, y)$$

It can be easily proven that cut metrics are indeed metrics. Moreover, we observe that the cut-ratio of each cut metric is lower bounded by the minimal cut-ratio of all the elementary cut metrics that define it.

Proposition 3.3

Let d be a cut metric on the graph G defined through $d_{T_1}, \dots, d_{T_k}$. Then, it holds that:

$$\phi(d) \geq \min_{i \in [k]} \phi(d_{T_i})$$

Proof. Let $\lambda_1, \dots, \lambda_k > 0$ be the scalars such that:

$$d(x, y) = \sum_{i \in [k]} \lambda_i d_{T_i}(x, y)$$

We observe that:

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

Claim: for any $a_1, \dots, a_k, b_1, \dots, b_k > 0$ it holds that:

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

Proof of the claim. Let $p = \min_{i \in [k]} \frac{a_i}{b_i}$. Then, it holds that:

$$\frac{\sum_{i \in [k]} a_i}{\sum_{i \in [k]} b_i} = \frac{\sum_{i \in [k]} \frac{b_i}{b_i} \cdot a_i}{\sum_{i \in [k]} b_i} \geq \frac{\sum_{i \in [k]} b_i p}{\sum_{i \in [k]} b_i} = p$$

□

Through the claim, we get that:

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

□

As a corollary, we get 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:

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

3.3 ℓ_1 metrics and isometric embeddings

Cut metrics are highly related with other types of metrics, in particular the ℓ_1 metric. In general, ℓ_μ metrics represent a generalization of the Manhattan and Euclidean distances, which correspond to ℓ_1 and ℓ_2.

Definition 3.3: ℓ_μ metric

The ℓ_μ **metric** is a function $\ell_\mu : S \times S \to \mathbb{R}$, where $S \subseteq \mathbb{R}^d$, defined as:

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

We observe that ℓ_μ, with $\mu \in \mathbb{R}$, is a metric for each $\mu \geq 1$, while it isn't for $0 < \mu < 1$. The relation between any cut metric d over k cuts and ℓ_1 metrics is stated through the concept of *isometric embedding*, i.e. a map between the topological spaces $(V(G), d)$ and $(\mathbb{R}^k, \ell_1)$ that preserves distances.

Definition 3.4: Isometrical embedding

Let $d_1 : A \to B$ and $d_2 : X \to Y$ be two metrics. We say that d_1 is **isometrically embedded** into d_2 if there is a function $f : S \to X$ such that:

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

Lemma 3.1

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

Proof. Next Lecture. □

Lemma 3.2

If $X \underset{fin}{\subseteq} \mathbb{R}^d$ then the ℓ_1 metric over X is isometrically embedded into a cut metric over $k = d \cdots (|x| - 1)$ cuts

Proof. Next Lecture. □

The two previous 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:

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

Bibliography

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, et al. "Proof verification and the hardness of approximation problems". In: *J. ACM* (1998). DOI: 10.1145/278298.278306.

[BKV12] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. "Densest subgraph in streaming and MapReduce". In: *Proc. VLDB Endow.* (2012). DOI: 10.14778/2140436.2140442.

[Cha00] Moses Charikar. "Greedy Approximation Algorithms for Finding Dense Components in a Graph". In: Springer Berlin Heidelberg, 2000.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

[GW95] Michel X. Goemans and David P. Williamson. "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming". In: *J. ACM* (1995). DOI: 10.1145/227683.227684.

[Kho02] Subhash Khot. "On the power of unique 2-prover 1-round games". In: Association for Computing Machinery, 2002. DOI: 10.1145/509907.510017.

[Lov75] L. Lovász. "On the ratio of optimal integral and fractional covers". In: (1975). DOI: 10.1016/0012-365x(75)90058-8.

[Rag08] Prasad Raghavendra. "Optimal algorithms and inapproximability results for every CSP?" In: Association for Computing Machinery, 2008. DOI: 10.1145/1374376.1374414.