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

Advanced Algorithms

Lecture notes integrated with the book TODO

Author
Alessio Bandiera

March 11, 2025

Contents

Information and Contacts

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

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

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

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

Suggested prerequisites:

- Progettazione degli Algoritmi

Licence:

1

TODO

placeholder

missing intro- duction

1.1 TODO

1.1.1 The Max Cut problem

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

Definition 1.1: Cut

Given an undirected graph $G = (V, E)$, and a subset of its vertices $S \subseteq V$, the **cut** induced by S on G is defined as follows

$$\mathrm{cut}(S) := \{e \in E \quad |S \cap e| = 1\}$$

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

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

Therefore, given a set of vertices S, the cut induced by S is simply the set of edges that have only one endpoint in S (implying that the other one will be in $V - S$).

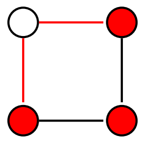


Figure 1.1: Given the set of red vertices S, the green edges represent $\mathrm{cut}(S)$.

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

Definition 1.2: Max Cut problem

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

Although this problem is known to be APX-Hard [1], approximation algorithms and heuristic methods like greedy algorithms and local search are commonly used to find near-optimal solutions.

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

Algorithm 1.1: Random Cut

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

---

function RANDOMCUT(G)
    $S := \varnothing$
    for $v \in V$ **do**
        Let c_v be the outcome of the flip of an independent fair coin
        if $c_v ==$ H **then**
            $S = S \cup \{v\}$
        end if
    end for
    return S
end function

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

Theorem 1.1: Expected approximation ratio of RandomCut

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

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

Proof. By definition, note that

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

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

$$\{v, w\} \in \mathrm{cut}(S) \iff (v \in S \land w \notin S) \lor (v \notin S \land w \in S)$$

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

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

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

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

Analogously, we can show that

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

This implies that

$$\Pr[e \in \mathrm{cut}(S)] = \Pr[\xi_1 \lor \xi_2] = \Pr[\xi_1] + \Pr[\xi_2] - \Pr[\xi_1 \land \xi_2] = \frac{1}{4} + \frac{1}{4} - 0 = \frac{1}{2}$$

Hence, we have that

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

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

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

Algorithm 1.2: t-times Random Cut

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

```
function t-TIMESRANDOMCUT(G, t)
    for i ∈ [t] do
        S_i := RANDOMCUT(G)
    end for
    return S ∈ arg max_{i∈[t]} |cut(S_i)|
end function
```

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

Theorem 1.2

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

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

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

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

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

In particular, this inequality can be rewritten as follows:

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

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

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

Note that the event in the last probability, namely

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

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

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

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

Using the fact that

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

we have that

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

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

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

placeholder □ last part

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

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

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

1.1.2 The Vertex Cover problem

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

Definition 1.3: Vertex cover

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

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

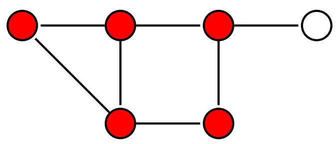


Figure 1.2: An example of a vertex cover.

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

Definition 1.4: Vertex Cover problem

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

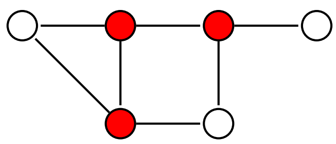


Figure 1.3: This is the *minimum vertex cover* of the previous graph.

As famously proved by Karp [2] in 1972, this problem is NP-Complete, hence we are interested in finding algorithms that allow to find approximations of optimal solutions. For instance, an algorithm that is able to approximate VC concerns the matching problem.

Definition 1.5: Matching

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

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

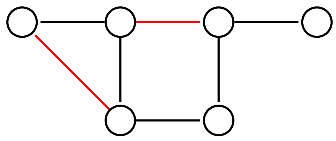


Figure 1.4: A *matching* of the previous graph.

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

Definition 1.6: Maximal matching

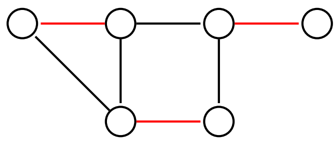
A **maximal matching** is a matching that cannot be extended any further.

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

Definition 1.7: Maximum matching

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

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



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

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

Algorithm 1.3: Maximal matching

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

1: **function** MAXIMALMATCHING(G)
2: $\quad S := \varnothing$
3: $\quad$ **while** $E \neq \varnothing$ **do**
4: $\qquad$ Choose $e \in E$
5: $\qquad S = S \cup \{e\}$
6: $\qquad$ Remove from E all the edges incident on u or on v
7: $\qquad E = E - \{e\}$
8: $\quad$ **end while**
9: $\quad$ **return** S
10: **end function**

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

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

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

Theorem 1.3: Matchings bound vertex covers

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

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

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

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

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

Theorem 1.4: 2-approximation of VC problem

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

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

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

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

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

□

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

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

Algorithm 1.4: Decisional VC

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

---

function VC(G, k)
    if $E == \varnothing$ **then**
        return `True`
    else if $k == 0$ **then**
        return `False`
    else
        Choose $e = \{u, v\} \in E$
        if VC$(G[V - \{u\}], k-1)$ **then**
            return `True`
        end if
        if VC$(G[V - \{v\}], k-1)$ **then**
            return `True`
        end if
        return `False`
    end if
end function

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

Definition 1.8: Induced subgraph

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

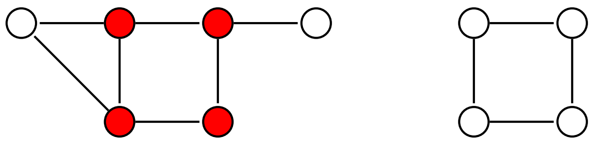


Figure 1.5: On the left: a graph G and a set of vertices S. On the right: the graph $G[S]$.

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

- if the current graph has no edges, we covered every edge of the graph, therefore we return `True`
- if the current graph has some edges, but $k = 0$, then G does not admit a vertex cover of size k, thus we return `False`
- if the current graph has some edges, and $k \neq 0$, then we choose an edge $e = \{u, v\} \in E$ arbitrarily, and we try to consider first u then v in a possible vertex cover — note that $G[V - \{x\}]$ is a graph that does not contain x, neither any edge adjacent to it; if both attempts fail, we return `False`

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

1.2 Mathematical Programming

placeholder

introduction on math programming

IPs can be used to solve a wide range of problems. For instance, given a graph $G = (V, E)$, the **vertex cover** problem that we discussed in the previous section can be formulated through the following IP:

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

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

Figure 1.6: IP for VC.

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

Lemma 1.1

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

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

is a minimum vertex cover for G.

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

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

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

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

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

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

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

In particular, this lemma implies that VC can be reduced to Integer programming, indeed solving IPs is actually NP-Hard [2], differently from LPs. This result shows that IPs cannot be used *directly* to obtain perfect solutions, but they are still very useful thanks to **relaxation**.

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

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

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

Figure 1.7: LP relaxation for the IP of VC.

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

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

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

Theorem 1.5

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

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

is a 2-approximation for VC.

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

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

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

To prove that $\overline{S}$ is indeed a 2-approximation, we just need to show the following: given a minimum vertex cover S^* of G, it holds that $\left|\overline{S}\right| \leq 2\left|S^*\right|$. By the claim in Lemma 1.1, we know that S^* there exists a feasible solution $\{x_u^*\}_{u \in V}$ for the IP, which must be optimal for the IP since S^* is a minimum vertex cover for G. Therefore, we have that

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

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

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

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

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

1.2.1 Integrality gap

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

For example, the inequalities discussed in the proof of Lemma 1.1 could be rewritten as follows

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

and in particular $|\overline{S}| \leq 2\text{LP}^* \leq 2\text{IP}^*$. Can we improve this approximation ratio of 2 through LP relaxation? In general, for any α possible approximation ratio, it must hold that

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

because otherwise

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

meaning that $\overline{S}$ would be a solution better than the optimal solution of the IP, which is impossible. We can generalize this concept as follows.

Definition 1.9: Integrality gap

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

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

The integrality gap for the problem P is defined as follows

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

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

Proposition 1.1: Limits of LP relaxation

Given a problem P for which there is an α-approximation algorithm which uses LP relaxation, it holds that

- if P is a **minimization problem**, then $\alpha \geq \text{IG}_{\text{P}}$
- if P is a **maximization problem**, then $\alpha \leq \text{IG}_{\text{P}}$

Now, let us analyze again VC and try to bound IG_{VC}. Consider the following *clique* graph:

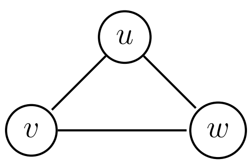


Figure 1.8: The graph K_3.

it is easy to see that

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

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

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

therefore, by definition of IG we have that

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

Moreover, Theorem 1.5 shows that we already know an algorithm that employs LP relaxation which yields a 2-approximation of VC; therefore, this lower bound on $\mathrm{IG}_{\mathrm{VC}}$ — together with the previous proposition — implies that any possible approximation ratio α on VC must satisfy

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

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

Theorem 1.6: Integrality gap for VC

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

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

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

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

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

Proof of the Claim. Consider a vertex cover $S = \{v_1\}$ for any vertex $v_1 \in V(K_n)$; since K_n is a clique, by definition $\deg(v_1) = n-1$, therefore S is able to cover only $n-1$ *uncovered* edges of K_n. Now, consider another vertex $v_2 \in V(K_n)$, and add it to $S = \{v_1, v_2\}$; we observe that $\deg(v_2) = n-1$, but $v_1 \sim v_2$ because K_n is a clique, therefore v_2 will be able to cover only $n-2$ *uncovered* edges of K_n. By the same reasoning, each new vertex $v_i \in V(K_n)$ added to S will be able to cover only $n-i$ *uncovered* edges of K_n. However, note that

$$|E(K_n)| = \binom{n}{2} = \frac{n(n-1)}{2}$$

therefore, to cover all the edges of K_n we need $|S|$ to satisfy the following inequality

$$\sum_{i=1}^{|S|} (n-i) \geq \frac{n(n-1)}{2} \iff n\,|S| - \frac{|S|\,(|S|+1)}{2} \geq \frac{n(n-1)}{2}$$

After some calculations, we derive the following inequality

$$|S|^2 + |S|\,(1-2n) + n(n-1) \leq 0$$

which is satisfied for any value $n-1 \leq |S| \leq n$, therefore a vertex cover of cardinality $|S| = n-1$ suffices to cover all the edges of K_n. $\square$

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

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

which means that

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

$\square$

1.2.2 The Set Cover problem

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

Definition 1.10: The Set Cover problem

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

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

As for VC, in 1972 [2] proved that SC is NP-Complete as well. Moreover, similarly to what we did for VC, we can convert SC into an IP, as follows.

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

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

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

Figure 1.9: IP for SC.

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

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

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

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

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

Nevertheless, by setting

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

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

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

Algorithm 1.5: Randomized rounding for SC

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

---

function RandomizedRoundingSC($\mathcal{U}$, C)
    $A := \varnothing$
    $\{\overline{x}_j\}_{j \in [m]} := \text{LP}_{\text{SC}}(\mathcal{U}, C)$ ▷ an optimal soluion on the LP relaxation
    for $k \in [\lceil 2 \ln n \rceil]$ **do**
        for $j \in [m]$ **do**
            Let $c_{k,j}$ be the outcome of the flip of an ind. coin with H prob. set to $\overline{x}_j$
            if $c_{k,j} ==$ H **then**
                $A = A \cup \{S_j\}$
            end if
        end for
    end for
end function

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

Lemma 1.2

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

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

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

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

Proof of the Claim.

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

□

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

Proof of the Claim.

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

□

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

Proof of the Claim.

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

□

□

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

Lemma 1.3

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

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

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

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

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

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

□

TODO

missing something

Additionally, note that the algorithm can be modified to get a better bound, by simply replacing $\lceil 2 \ln n \rceil$ with $\lceil (1 + \varepsilon) \ln n \rceil$, for any $\varepsilon > 0$. In fact, with this modification we would get that $\mathbb{E}\left[|A|\right] \leq \lceil (1 + \varepsilon) \ln n \rceil \, \mathrm{LP}^*$, therefore

missing something

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

Theorem 1.7: Integrality gap of SC

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

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

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

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

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

and define the collection of sets as follows

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

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

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

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

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

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

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

Proof of the Claim. Consider the solution

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

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

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

hence this is a feasible solution to the LP relaxation, and its value is simply given by

$$m \cdot \frac{2}{m} = 2$$

$\square$

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

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

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

We have that

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

hence, we can always define a subset $A \subseteq T$ such that $|A| = \frac{m}{2}$. However, since $A \subseteq T$, we have that $e_A \notin S_{i_1} \cup \ldots \cup S_{i_k}$ $\square$ da finire

$\square$

Bibliography

[1] Sanjeev Arora et al. "Proof verification and the hardness of approximation problems". In: *Journal of the ACM* 45.3 (May 1998), 501–555. ISSN: 1557-735X. DOI: 10.1145/278298.278306. URL: http://dx.doi.org/10.1145/278298.278306.

[2] Richard M. Karp. "Reducibility among Combinatorial Problems". In: *Complexity of Computer Computations*. Springer US, 1972, 85–103. ISBN: 9781468420012. DOI: 10.1007/978-1-4684-2001-2_9. URL: http://dx.doi.org/10.1007/978-1-4684-2001-2_9.