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 TODO

Author
Alessio Bandiera

March 1, 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

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

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

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

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

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 $V - S$).

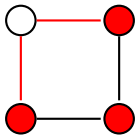


Figure 1.1: Given the set of red vertices S, the green edges represent $\text{cut}(S)$.

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

Definition 1.2: Max Cut problem

Given an undirected graph $G = (V, E)$, determine the set $S \subseteq V$ that maximizes $|\text{cut}(S)|$.

Although this problem is known to be APX-Hard [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 the Max Cut problem. This algorithm runs in polynomial time and achieves the approximation guarantee with high probability.

Algorithm 1.1: Random Cut

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

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

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

Theorem 1.1: Expected approximation ratio of RandomCut

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

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

Proof. By definition, note that

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

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

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

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

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

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

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

Analogously, we can show that

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

This implies that

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

Hence, we have that

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

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

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

Algorithm 1.2: t-times Random Cut

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

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

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

Theorem 1.2

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

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

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

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

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

In particular, this inequality can be rewritten as follows:

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

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

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

Note that the event in the last probability, namely

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

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

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

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

Using the fact that

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

we have that

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

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

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

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, 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

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

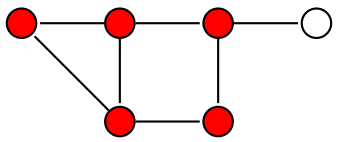


Figure 1.2: An example of a vertex cover.

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

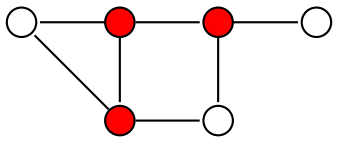


Figure 1.3: This is the *minimum vertex cover* of the previous graph.

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 the VC problem 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

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

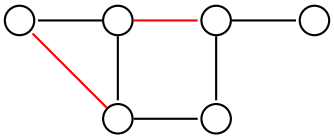


Figure 1.4: A *matching* of the previous graph.

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

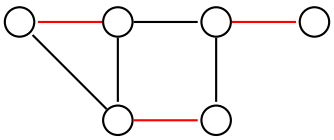
A **maximal matching** is a matching that cannot be extended any further.

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

Algorithm 1.3: Maximal matching

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

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

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

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

Additionally, another major reason 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. □

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 the VC problem.

Theorem 1.4: 2-approximation of VC problem

The MaximalMatching algorithm is a 2-approximation of the Vertex Cover problem.

Proof. Given an undirected $G = (V, E)$, let $S = \text{MaximalMatching}(G)$, and let $e_1, \ldots, e_t$ be the sequence of edges chosen by the algorithm at each iteration of the `while` loop.

Note that, by definition of the algorithm, at each iteration exactly 2 vertices are added to $S \subseteq V$, and it always holds that

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

for any iteration $i \in [t-1]$, because in line 6 the algorithm removes from E all the edges incident on either u or v. This implies that $|S| = 2t$.

placeholder □

the last part is wrong

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

Interestingly, the decisional version of the VC problem 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 the VC problem by trying all possible choices of set of vertices $S \subseteq V$, and backtrack if necessary. The following algorithm employs this idea.

Algorithm 1.4: Decisional VC

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

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

The algorithm uses the definition of **induced subgraph**, which is defined as follows.

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.

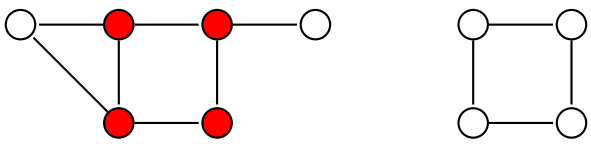


Figure 1.5: On the left: a graph G and a set of vertices S. On the right: the graph $G[S]$.

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 $O\left(2^k \cdot n^2\right)$.

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.