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

February 27, 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/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

Optimization algorithms (?)

1.1 The max cut problem

Introduction to the max-cut Problem

The **max-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**.

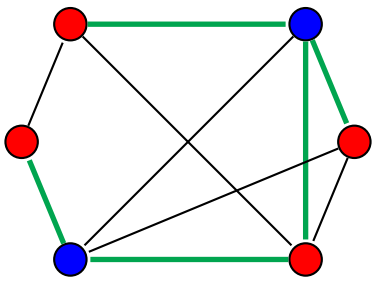

Figure 1.1: The red vertices and the blue vertices form a cut of the graph. The green edges are the edges of the cut-set.

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 $E \subseteq \{\{u, v\} \mid u, v \in V\}$, while in the directed case we have that $E \subseteq \{(u, v) \mid u, v \in V\}$, where (u, v) represents and edge $u \to v$. Instead of an endpoint-based definition, the cut of a graph can also be defined in the following way.

Definition 1: Cut of a graph

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

The MAXCUT 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]. Assuming that $\mathsf{P} \neq \mathsf{NP}$, this means that there is no polynomial-time algorithm that can solve the problem exactly in all cases. 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 MAXCUT, 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 MAXCUT unless $\mathsf{P} = \mathsf{NP}$ is true [ALM+98], which we assume to be false. For MAXCUT, 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: a 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, V − S)
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 MAXCUT.

Theorem 1

Given a graph G, let (S^*, T^*) be an optimal solution to MAXCUT(G). Given the output (S, T) of RandomCut(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.

Theorem 2

Given a graph G and a non-negative integer t, let (S^*, T^*) be an optimal solution to MAXCUT(G). Given the output (S, T) of t-RandomCut(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 $0 < \varepsilon, \delta < 1$.

Algorithm 1.2 The t-times random-cut algorithm

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

1: **function** t-RANDOM-CUT(G, t)
2: **for** $i \in [\lceil t \rceil]$ **do**
3: $(S_i, T_i) \leftarrow$ RANDOM-CUT(G)
4: **end for**
5: Return $(S, V - S) \in \arg\max_{i \in [\lceil t \rceil]} |\text{cut}(S_i, T_i)|$
6: **end function**

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:

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

□

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

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

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 $0 < \varepsilon < 1$, meaning that we will always sacrifice some optimality to boost our probability. This trade-off idea between optimality and probability by running multiple times the same algorithm can be applied to many other problems.

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.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990. ISBN: 0716710455.