**Advanced Algorithms** March 15, 2022

# Lecture 7: Approximation Algorithms

*Notes by Ola Svensson*[1]

## 1 When Relaxing Integrality is Not for Free: Approximation Algorithms

Thousands (or millions :)) of optimization problems have been proved to be NP-hard, which means that finding optimal solutions for them is very likely to be intractable. More specifically, unless P = NP, there is no algorithm for an NP-hard optimization problem that satisfies all of the following three desiderata:

1. The algorithm is efficient (runs in polynomial time).
2. The algorithm is reliable (works for any input instance).
3. The algorithm finds an optimal solution (optimality).

Therefore, when confronted which such problems, we need to relax one of the above conditions when dealing with NP-hard optimization problems. If we relax reliability, then we get heuristics that are designed to work well on instances that commonly appear in practice. If we relax the third condition, we obtain *approximation algorithms*. The notion of approximation algorithms allows us to obtain a more fine-grained picture of the difficulty of NP-hard optimization problems: some problems turn out to have very good approximations, whereas others resist such attempts.

The formal definition of approximation algorithms is as follows.

**Definition 1** *An $\alpha$-approximation algorithm for a given optimization problem is an algorithm that runs in polynomial time and outputs a solution $S$ such that:*

- $\frac{cost(S)}{cost(Optimal\ solution)} \leq \alpha$ *if the problem is a minimization problem,*
- $\frac{profit(S)}{profit(Optimal\ solution)} \geq \alpha$ *if the problem is a maximization problem.*

It is clear that we will have $\alpha \geq 1$ for minimization problems and $\alpha \leq 1$ for maximization problems. Moreover, if $\alpha = 1$, then we have an efficient exact algorithm (and the problem is in P). The goal when designing approximation algorithms is to achieve a guarantee $\alpha$ that is as close to 1 as possible. We now introduce a general framework for using linear programming in the design of approximation algorithms followed by an application: vertex cover.

## 2 Using Linear Programming to Design Approximation Algorithms

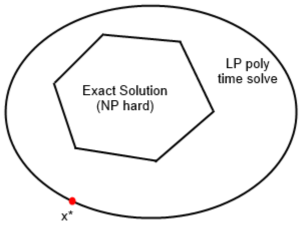
Consider a minimization problem. When considering the definition of approximation algorithms, it seems very hard, even for a specific instance of the problem, to analyze the approximation guarantee $\frac{\text{cost}(S)}{\text{cost(Optimal solution)}}$ of the algorithm on the instance, as we are comparing ourselves with an optimal solution (that is most likely very hard to compute and does not possess any nice structure). The solution to this is to compare ourselves with a *lower bound* on the optimum. Indeed, we then have that

[1]**Disclaimer:** These notes were written as notes for the lecturer. They have not been peer-reviewed and may contain inconsistent notation, typos, and omit citations of relevant works.

$$\frac{\text{cost}(S)}{\text{cost(Optimal solution)}} \leq \frac{\text{cost}(S)}{\text{lower bound on opt}} \leq \alpha\,.$$

From the above, it is clear that we need a good lower bound to be able to claim a good guarantee $\alpha$. To obtain such a lower bound (and to design the approximation algorithm), linear programming is super handy. A popular framework is as follows:

1. Give an exact formulation of the problem as Integer LP – usually with binary variables ($x_i \in \{0,1\}$).
2. Relax to LP – $x_i \in [0,1]$



3. Solve LP to get a optimal solution $x^*$ to the LP which is a lower (upper) bound on the optimal solution to Integer LP and thus the original problem. Then somehow round $x^*$ to an integral solution "without losing too much" (which will determine the guarantee $\alpha$).

We now use the above framework for the vertex cover problem, and then introduce the concept of integrality gap.

## 2.1 Vertex Cover

Here, we apply the framework to get an approximation algorithm for the *Vertex Cover* problem. First recall the definition:

**Definition 2 (Vertex Cover (VC) Problem)** *Given a graph $G = (V, E)$ and a weight function on the vertices $w : V \to \mathbb{R}_+$, output a set $C \subseteq V$ of minimum weight such that for all $\{u, v\} \in E$, $u \in C$ or $v \in C$.*

First, we define an ILP solving VC: For all $v \in V$, we introduce a variable $x_v$, which is 1 if $v \in C$, and 0 otherwise. The objective function is

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

and for each $\{u, v\} \in E$ we introduce the constraint $x_u + x_v \geq 1$. This ensures that for each edge, at least one endpoint is in $C$. Additionally, we require that for each $v \in V$ we have $x_v \in \{0,1\}$. Note that at this point our Integer LP formulation is exactly equivalent to the original problem.

Second, we relax the ILP to an LP by allowing that $x_v \in [0,1]$. Actually, it's sufficient to require that $x_v \geq 0$, because having an $x_v$ greater than 1 will not satisfy any additional constraint, but only increase the value of the objective function, so this will not happen in an optimal solution.

Third, we have to do the rounding: Suppose we've solved the LP and got an optimal solution $x^*$. We will return $C = \{v \in V : x_v^* \geq \frac{1}{2}\}$ as our solution to VC.

**Claim 3** *$C$ is a feasible solution.*

**Proof** Consider any edge $\{u, v\}$ and its constraint. Since $x^*_u + x^*_v \geq 1$, at least one of $x^*_u$, $x^*_v$ is $\geq \frac{1}{2}$ and thus in $C$. ■

**Claim 4** *The weight of $C$ is at most twice the value of the optimal solution of VC.*

**Proof** We have

$$\sum_{v\in C} w(v) = \sum_{v\in V : x^*_v \geq \frac{1}{2}} w(v) \leq \sum_{v\in V : x^*_v \geq \frac{1}{2}} 2x^*_v w(v) \leq \sum_{v\in V} 2x^*_v w(v) = 2\sum_{v\in V} x^*_v w(v) = 2LP_{OPT} \leq 2VC_{OPT}$$

where the first inequality holds because $2x^*_v \geq 1$. ■

So, we have designed a 2-approximation algorithm for Vertex Cover. This is a simple algorithm using linear programming. To appreciate the power of the framework, I challenge you to find a 2-approximation algorithm for Vertex Cover (with node-weights) without the use of LPs. (It is possible but quite difficult.)

## 2.2 Integrality Gap

The notion of the *integrality gap* allows us to bound the power of our linear programming relaxation. Let $\mathcal{I}$ be the set of all instances of a given problem. In the case of a minimization problem, the integrality gap $g$ is defined as

$$g = \max_{I\in\mathcal{I}} \frac{OPT(I)}{OPT_{LP}(I)}\,.$$

As an example, suppose $g = 2$, and that LP found that $OPT_{LP} = 70$. Then, since our problem instance might be the one which maximizes the expression for $g$, all we can guarantee is that $OPT(I) \leq 2 \cdot OPT_{LP}(I) = 140$, so it's not possible to find an approximation algorithm (using only this linear programming relaxation) that approximates better than within a factor of $g = 2$.

### 2.2.1 Integrality Gap for Vertex Cover

**Claim 5** *The integrality gap for vertex cover is at least $2 - \frac{2}{n}$.*

**Proof** Consider the complete graph on $n$ vertices. We have $OPT = n - 1$, because if there are 2 vertices that we don't choose, the edge between them is not covered. However, $LP_{OPT} \leq \frac{n}{2}$, because assigning $\frac{1}{2}$ to each vertex is a feasible solution of cost $\frac{n}{2}$, so the optimum can only be smaller. Now,

$$g \geq \frac{n-1}{\frac{n}{2}} = 2 - \tfrac{2}{n}\,.$$

■

We remark that our 2-approximation algorithm for vertex cover implies that the integrality gap is *at most* 2.

## 3 Set Cover via Randomized Rounding

Let us now apply the framework to the *Set Cover* problem. It can be seen as a generalization of the vertex cover problem and its definition is as follows:

**Definition 6 (Set Cover Problem)** *Given a universe $\mathcal{U} = \{e_1, e_2, \dots e_n\}$, and a family of subsets $\mathcal{T} = \{S_1, S_2, \dots S_m\}$ and a cost function $c : \mathcal{T} \to \mathbb{R}_+$, find a collection $C$ of subsets of minimum cost that cover all elements.*

As for vertex cover, we start by giving an exact Integer LP formulation. For each $i \in \{1, \dots m\}$, define $x_i$, which is 1 if $S_i \in C$, and 0 otherwise. The objective function is

$$\min \sum_{i=1}^{m} x_i \cdot c(S_i)$$

and for each element $e \in \mathcal{U}$, we add the constraint $\sum_{S_i \,:\, e \in S_i} x_i \geq 1$. This ensures that each element is covered by at least one set in $C$. And for each $x_i$, we require that $x_i \in \{0, 1\}$ in the ILP. The LP relaxation is then obtained by replacing the boolean constraints $x_i \in \{0, 1\}$ by $x_i \in [0, 1]$.

Now suppose that each element belongs to at most $f$ sets. Then, as in your exercise on vertex cover on $k$-uniform hypergraphs, we can do the following rounding: $C = \{S_i : x_i^* \geq \frac{1}{f}\}$. In each constraint, there's at least one $x_i^*$ which is at least $\frac{1}{f}$, so each constraint is satisfied. Using the same reasoning as in the analysis of the vertex cover rounding, we can show that this approximation is within a factor of $f$.

### 3.1 A better approximation for Set Cover

If we introduce randomness and allow our algorithm to output non-feasible solutions with some small probability, we can get much better results (in expectation).

We use the same LP as in the previous section, and will run the following algorithm:

1. Solve the LP to get an optimal solution $x^*$.
2. Choose some positive integer constant $d$ (we will see later how $d$ affects the guarantees we get). Start with an empty result set $C$, and repeat step 3 $d \cdot \ln(n)$ times.
3. For $i = 1, \dots m$, add set $S_i$ to the solution $C$ with probability $x_i^*$, choosing independently for each set.

Now let us analyze what guarantees we can get:

**Claim 7** *The expected cost of all sets added in one execution of Step 3 is*

$$\sum_{i=1}^{m} x_i^* c(S_i) = LP_{OPT}$$

**Proof**

$$\mathbb{E}[\text{rounded cost}] = \sum_{i=1}^{m} c(S_i) \Pr[S_i \text{ is added}] = \sum_{i=1}^{m} c(S_i) x_i^* = LP_{OPT}$$

■

From this, we can immediately derive

**Corollary 8** *The expected cost of $C$ after $d \cdot \ln(n)$ executions of Step 3 is at most*

$$d \cdot \ln(n) \cdot \sum_{i=1}^{m} c(S_i) x^* \leq d \cdot \ln(n) \cdot LP_{OPT} \leq d \cdot \ln(n) \cdot OPT$$

Note that we have $LP_{OPT} \leq OPT$ because $LP$ is a relaxation of the original problem, so its optimum can only be better.

That sounds good, but we should also worry about feasibility:

**Claim 9** *The probability that a constraint remains unsatisfied after a single execution of Step 3 is at most* $\frac{1}{e}$*.*

**Proof** Suppose our constraint contains $k$ variables, and let us write it as $x_1 + x_2 + \cdots + x_k \geq 1$. Then,

$$\begin{aligned}
\Pr[\text{constraint unsat.}] &= \Pr[S_1 \text{ not taken}] \ldots \Pr[S_k \text{ not taken}] \\
&= (1 - x_1^*) \ldots (1 - x_k^*) \\
&\leq e^{-x_1^*} \cdot \ldots \cdot e^{-x_k^*} \qquad (1) \\
&= e^{-\sum_{i=1}^{k} x_i^*} \\
&\leq e^{-1} \qquad (2)
\end{aligned}$$

where (1) follows from the inequality $1 - x \leq e^{-x}$ and (2) from the fact that $\sum_i x_i^* \geq 1$. ■

**Claim 10** *The output $C$ is a feasible solution with probability at least* $1 - \frac{1}{n^{d-1}}$*.*

**Proof** Using claim 9, we find that the probability that a given constraint is unsatisfied after $d \cdot \ln(n)$ executions of step 3 is at most

$$\left(\tfrac{1}{e}\right)^{d \cdot \ln(n)} = \frac{1}{n^d}$$

and by union-bound, the probability that there exists any unsatisfied constraint is at most

$$n \cdot \frac{1}{n^d} = \frac{1}{n^{d-1}}$$

■

Now we have an expected value for the cost, and also a bound on the probability that an infeasible solution is output, but we still might have a bad correlation between the two: It could be that all feasible outputs have a very high cost, and all infeasible outputs have a very low cost.

The following claim deals with that worry.

**Claim 11** *The algorithm outputs a feasible solution of cost at most $4d\ln(n)OPT$ with probability greater than* $\frac{1}{2}$*.*

**Proof** Let $\mu$ be the expected cost, which is $d\ln(n) \cdot OPT$ by corollary 8. We can upper-bound the bad event that the actual cost is very high: By Markov's inequality, we have $\Pr[cost > 4\mu] \leq \frac{1}{4}$. The other bad event that we have to upper bound is that the output is infeasible, and by claim 10, we know that this happens with probability at most $\frac{1}{n^{(d-1)}} \leq \frac{1}{n}$. Now in the worst case, these two bad events are completely disjoint, so the probability that no bad event happens is at least $1 - \frac{1}{4} - \frac{1}{n}$, and if we suppose that $n$ is greater than 4, this probability is indeed greater than $\frac{1}{2}$. ■

We have thus designed a randomized $O(\log n)$-approximation algorithm for the set cover problem.

We remark that the used framework has the following general advantage (compared to worst-case guarantees): we can often get better per-instance guarantee than the general approximation factor: Suppose we have an instance where $LP_{OPT} = 100$, and our algorithm found a solution of cost 110. Since we know that $LP_{OPT} \leq OPT$, we can say that our solution on this instance is at most 10% away from the optimal solution for this instance.

## References

[1] Mateusz Golebiewski, Maciej Duleba: *Scribes of Lecture 5 in Topics in TCS 2015. http://theory.epfl.ch/courses/topicstcs/Lecture52015.pdf*