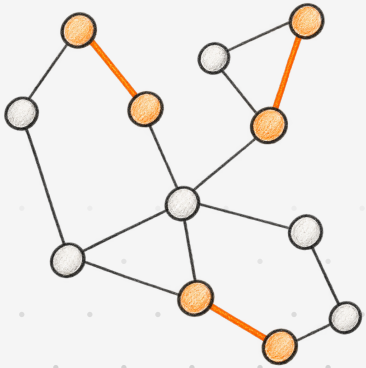
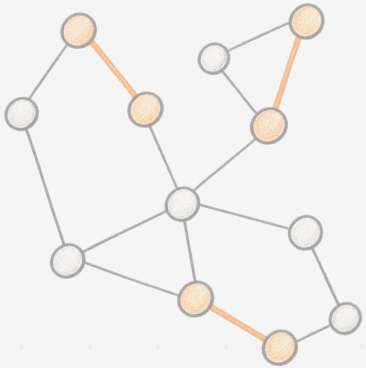


Optimization ▸ Approximation Algorithms

Lecture 11

- Approximating an Optimum
- Lower-Bound Certificates
- Algorithms from Certificates
- How Good Can an Approximation Be?



Optimization ▶ Approximation Algorithms

Lecture 11

- Approximating an Optimum
- Lower-Bound Certificates
- Algorithms from Certificates
- How Good Can an Approximation Be?

Optimization Problems

An optimization problem gives us an input I and a set $\mathcal{F}(I)$ of **feasible solutions**, and asks for the best one:

$$\min_{x \in \mathcal{F}(I)} \text{cost}(x) \qquad \text{or} \qquad \max_{x \in \mathcal{F}(I)} \text{quality}(x).$$

For example, given a directed graph G with capacities c and vertices s, t:

Problem	**Feasible solutions**	**Objective**
Maximum flow	Valid s-t flows f	Maximize $\text{value}(f)$
Minimum s-t cut	Cuts (S, T) with $s \in S$, $t \in T$	Minimize $\text{cap}(S, T)$

When Exact Optimization Is Hard

Both examples admit efficient algorithms: we can compute a maximum flow, and a minimum s-t cut along with it, in polynomial time.

But many natural optimization problems are NP-hard, including the one we will study today.

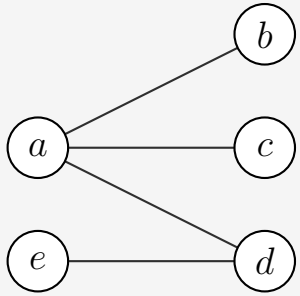
For an NP-hard problem, we cannot expect a polynomial-time algorithm that solves it **exactly**, unless $\mathrm{P} = \mathrm{NP}$.

But it may be possible to relax our goal, and find **approximate** solutions efficiently.

The Minimum Vertex Cover Problem

Definition: Vertex cover

In an undirected graph $G = (V, E)$, a set $S \subseteq V$ is a vertex cover if every edge $\{u, v\} \in E$ has at least one endpoint in S.



Problem

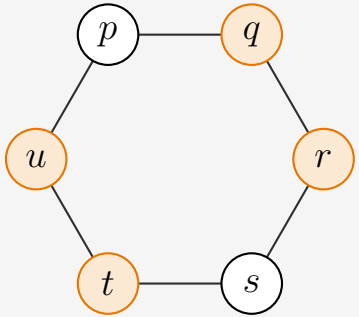
Find a vertex cover with minimum size.

The set V always covers every edge. Finding a **small** cover is the challenge.

Minimum Vertex Cover is NP-hard.

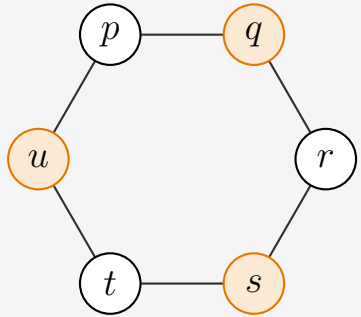
Example: Vertex Covers

A vertex cover: 4 vertices



Every edge has an orange endpoint, and removing any orange vertex leaves an edge uncovered.

A minimum vertex cover: 3 vertices



No cover with fewer than 3 vertices exists.

The Guarantee We Will Aim For

Write $n = |V|$, $m = |E|$, and

$$\mathrm{OPT}(G) = \min\{|S| : S \text{ is a vertex cover of } G\}.$$

Theorem: Today's goal

There is an $O(n + m)$-time algorithm that returns a vertex cover S satisfying

$$|S| \leq 2\,\mathrm{OPT}(G).$$

- The output must really cover **every** edge.
- Its size must be at most twice the best possible size, on **every** input.

Measuring an Approximation

For a minimization problem, let $A(I)$ be the cost of the feasible solution returned by an algorithm on input I.

Definition: Approximation guarantee

An algorithm is an α-approximation, for $\alpha \geq 1$, if

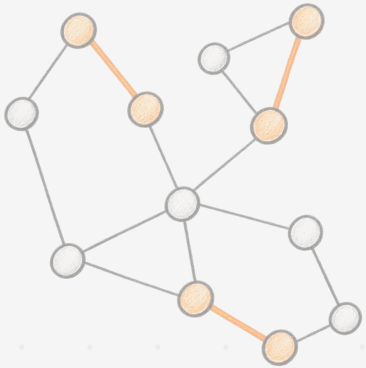
$$A(I) \leq \alpha \operatorname{OPT}(I) \qquad \text{for every input } I.$$

For inputs with positive optimum, compare $A(I)/\operatorname{OPT}(I)$. The approximation ratio is the worst-case ratio.

For a maximization problem, an α-approximation instead guarantees $A(I) \geq \operatorname{OPT}(I)/\alpha$.

A factor of 2 allows cost 20 when the optimum is 10, and cost 2000 when the optimum is 1000.

This is a **multiplicative** guarantee. An additive guarantee would bound $A(I) - \operatorname{OPT}(I)$ instead.



Optimization ▶ Approximation Algorithms

Lecture 11

- Approximating an Optimum
- Lower-Bound Certificates
- Algorithms from Certificates
- How Good Can an Approximation Be?

Comparing with an Unknown Optimum

We want to prove

$$\text{cost of our solution} \leq 2\,\mathrm{OPT}\,.$$

But computing OPT is the problem we are trying to avoid!

Key Idea: Certificates for lower or upper bounds on OPT.

A certificate of a lower bound need not tell us which solution is optimal. It only needs to show that **every** feasible solution costs at least some value L. Then

$$\text{cost of our solution} \leq 2L \leq 2\,\mathrm{OPT}\,.$$

Searching for a Certificate

For vertex cover, a lower-bound certificate is a structure that forces **every** cover to be large.

Problem

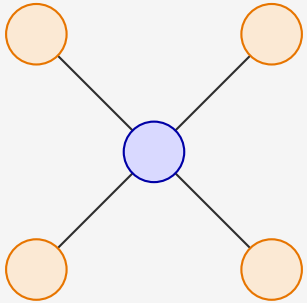
Find structures in G that imply a lower bound on the size of **any** vertex cover.

We do not yet know what such a structure looks like. Let's experiment with small graphs:

- find a small cover;
- then ask how to **prove** that no smaller cover exists.

Whatever we used in that proof is a candidate certificate.

Attempt 1: A Star



The four orange leaves form a cover. So does the blue center by itself.

Claim

The blue center is a minimum vertex cover: $\mathrm{OPT} = 1$.

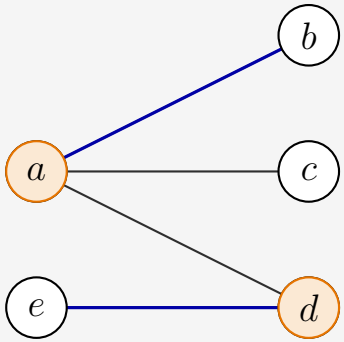
Proof

Take any edge. Every vertex cover must contain one of its endpoints, so every cover has size at least 1.

The center alone is a cover of size 1.

Certificate: a single edge.

Attempt 2: Can One Edge Still Suffice?



Can one vertex cover both $\{a, b\}$ and $\{d, e\}$?

These edges share no endpoint. A cover needs a **different vertex** for each, so $\mathrm{OPT} \geq 2$.

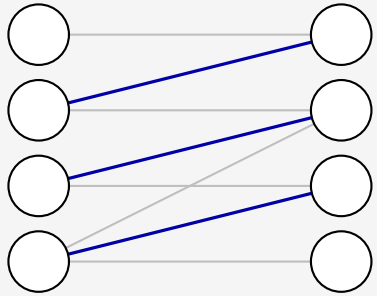
$\{a, d\}$ covers every edge. Thus $\mathrm{OPT} = 2$.

Edges that share no endpoints each need their **own** vertex in any cover. So k such edges certify $\mathrm{OPT} \geq k$.

Edges with No Shared Endpoints: Matchings

Definition: Matching

A matching $M \subseteq E$ is a set of edges with no shared endpoints. A vertex may belong to no edge of M, but cannot belong to two.



The three blue edges form a matching.

Every cover must pay for each blue edge, using distinct vertices.

A matching need not touch every vertex of the graph.

Any Matching Is a Lower-Bound Certificate

Lemma: Matching lower bound

For every matching M in G, every vertex cover S satisfies $|S| \geq |M|$.

Proof

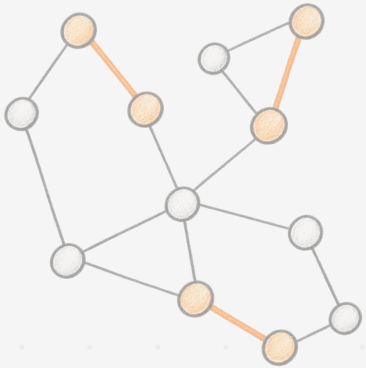
Take any vertex cover S.

It contains at least one endpoint of every edge in M.

The edges of M have disjoint endpoints, so these are at least $|M|$ **distinct** vertices.

Applying the lemma to a minimum vertex cover: for every matching M, $|M| \leq \mathrm{OPT}(G)$.

The largest matching gives the strongest such bound, but **any** matching certifies one.



Optimization ▶ Approximation Algorithms

Lecture 11

- Approximating an Optimum
- Lower-Bound Certificates
- Algorithms from Certificates
- How Good Can an Approximation Be?

Maximum vs. Maximal Matchings

Definition: Maximum matching

A matching with the largest possible number of edges.

Definition: Maximal matching

A matching M to which no further edge can be added:

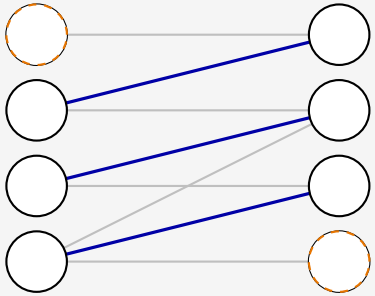
$$\text{for every } e \in E \setminus M, \quad M \cup \{e\} \text{ is not a matching.}$$

Equivalently, every edge outside M touches a vertex already used by M.

Every maximum matching is maximal. A maximal matching can be smaller.

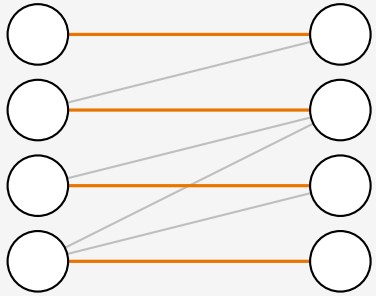
Example: Maximal vs. Maximum

A maximal matching: 3 edges



The two outlined vertices are unused and not adjacent.

A maximum matching: 4 edges



No matching can have more than 4 edges on 8 vertices.

Maximal means no extension by one edge. It does not rule out a better matching.

A Greedy Algorithm for Maximal Matching

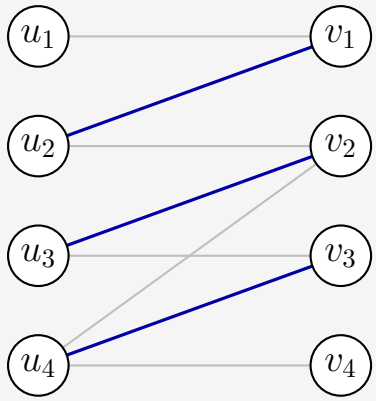
Algorithm: Greedy matching

1. Set $M \leftarrow \varnothing$ and $E_{\text{curr}} \leftarrow E$.
2. While E_{curr} is not empty:
   - Choose any edge $\{u, v\}$ in it and add it to M.
   - Remove every remaining edge touching u or v.
3. Return M.

No search for the best edge: **any** available edge will do.

Removing incident edges ensures that future choices cannot reuse either endpoint.

A Greedy Run with Three Edges



1. Choose $\{u_2, v_1\}$. Remove its incident edges.
2. Choose $\{u_3, v_2\}$. Remove its incident edges.
3. Choose $\{u_4, v_3\}$. No edges remain available.

The only unused vertices are u_1, v_4, which are not adjacent.

The result is maximal, with 3 edges. Another choice sequence can give 4.

The Greedy Output Is a Matching

Lemma: Greedy produces a matching

The set M returned by the greedy algorithm is a matching.

Proof

By induction on the number of iterations of the loop.

Base case: initially $M = \varnothing$, which is a matching.

Inductive step: assume M is a matching after the first i iterations. In the next iteration, the edge e added must have both endpoints untouched by M: we removed from E_{curr} every edge incident on a vertex touched by M.

So $M \cup \{e\}$ is again a matching. If no edge is added, there is nothing to prove.

The Greedy Output Is Maximal

E_{curr}: remaining edges. Choosing $\{u, v\}$ removes every edge touching u or v.

Lemma: Greedy produces a maximal matching

The matching M returned by the greedy algorithm is maximal.

Proof

Suppose not, for a contradiction. Then some edge $e = \{u, v\} \in E$ has no edge of M incident on u or v.

Edges leave E_{curr} only when they are incident on a vertex touched by M. Writing E_w for the edges incident on w, at termination

$$E_{\text{curr}} = E \setminus \bigcup_{w \text{ touched by } M} E_w \ni e,$$

so $E_{\text{curr}} \neq \varnothing$.

This contradicts the algorithm's termination condition, $E_{\text{curr}} = \varnothing$.

A Linear-Time Implementation

We do not need to physically delete all incident edges.

Algorithm: Scan the edges once

Set $M \leftarrow \varnothing$ and mark every vertex unused. For each edge $\{u, v\}$, in any fixed order:

- If both endpoints are unused, add the edge to M and mark both used.
- Otherwise, skip the edge.

Return M.

Proof: running time

Initializing the vertex marks takes $O(n)$ time. Each edge needs two mark checks and at most two updates, taking $O(1)$ time. The total is $O(n + m)$ with adjacency-list input.

An edge skipped because an endpoint is used can never become available again. This implements the same greedy rule.

Both Endpoints Now Give a Vertex Cover

Lemma: Cover from a maximal matching

If M is a maximal matching, then the set S of all endpoints of edges in M is a vertex cover.

Proof

Suppose not, for a contradiction. Then some edge $\{u, v\} \in E$ has neither endpoint in S.

Every vertex touched by an edge of M is in S. So no edge of M touches u or v.

Then $\{u, v\}$ shares no endpoint with any edge of M, so $M \cup \{\{u, v\}\}$ is still a matching. This contradicts the maximality of M.

The same matching does two jobs: any matching gives a lower bound on $\mathrm{OPT}(G)$; maximality makes its endpoints a feasible solution.

The Vertex Cover Algorithm

Algorithm: A cover from greedy matching

1. Compute a maximal matching M greedily.
2. Return both endpoints of every edge in M:

$$S = \bigcup_{\{u,v\} \in M} \{u, v\}.$$

- **Feasibility:** Maximality proves that S covers every edge.
- **Running time:** $O(n + m)$ for the matching and its endpoint set.

The Certificate Proves a Factor of Two

Theorem: A linear-time 2-approximation

The algorithm returns a vertex cover S in $O(n+m)$ time, with $|S| \leq 2\,\mathrm{OPT}(G)$.

Proof: solution quality

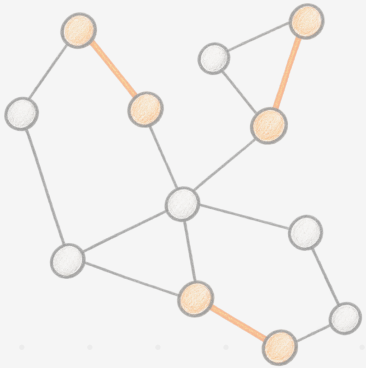
Every cover needs at least one distinct vertex per matching edge:

$$|M| \leq \mathrm{OPT}(G).$$

Our cover takes exactly two vertices per matching edge:

$$|S| = 2|M| \leq 2\,\mathrm{OPT}(G).$$

We never computed the optimum. The matching certifies $|M| \leq \mathrm{OPT}(G) \leq |S| = 2|M|$.



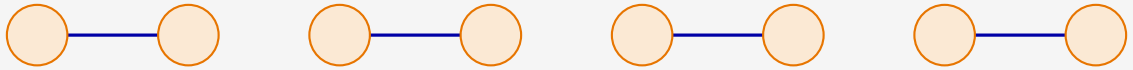
Optimization ▶ Approximation Algorithms

Lecture 11

- Approximating an Optimum
- Lower-Bound Certificates
- Algorithms from Certificates
- How Good Can an Approximation Be?

A Small Example Already Attains the Bound

Suppose the graph itself consists of $r \geq 1$ disjoint edges.



Every maximal matching contains all r edges. Our algorithm takes both endpoints of each:

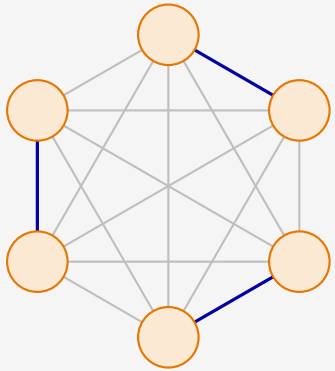
$$|S| = 2r.$$

One endpoint per edge is enough for an optimal cover, so $\mathrm{OPT} = r$.

$$\frac{|S|}{\mathrm{OPT}} = 2.$$

The Same Algorithm Can Be Almost Optimal

Let K_{2r} be the complete graph on $2r$ vertices: every pair is adjacent.



Exercise: Show that

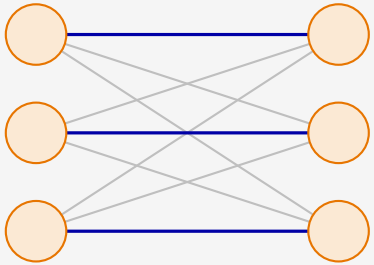
- $\mathrm{OPT}(K_{2r}) = 2r - 1$;
- every maximal matching has r edges.

The algorithm returns all $2r$ vertices:

$$\frac{|S|}{\mathrm{OPT}} = \frac{2r}{2r-1} = 1 + \frac{1}{2r-1} \longrightarrow 1.$$

Tightness on a Connected Graph

Let $K_{r,r}$ have two sets of r vertices, with every edge between the sets and no edges within either set.



Exercise: Show that

- $\mathrm{OPT}(K_{r,r}) = r$;
- every maximal matching has r edges.

Our algorithm takes $2r$ vertices, while one whole side is a cover of size r. The ratio is **exactly** 2.

The bound is tight for this algorithm. This does not prove that every vertex-cover algorithm must lose a factor of 2.

What Is Known About Vertex Cover

- **Upper bound:** simple 2-approximations are known, like ours. The best known ratio is only slightly better: $2 - \Theta(1/\sqrt{\log n})$ (Karakostas, 2009).
- **NP-hardness:** approximating within any factor below $\sqrt{2} \approx 1.414$ is NP-hard (Khot–Minzer–Safra, 2018), improving the earlier factor 1.36 (Dinur–Safra, 2005).
- **Conditional hardness:** under the Unique Games Conjecture, no polynomial-time algorithm achieves ratio $2 - \varepsilon$ for any constant $\varepsilon > 0$ (Khot–Regev, 2008).

News (October 2026): OpenAI proved the Unique Games Conjecture. Combined with Khot–Regev, approximating vertex cover within $2 - \varepsilon$ is NP-hard for every $\varepsilon > 0$: our simple factor of 2 is the best possible unless $\mathrm{P} = \mathrm{NP}$.

Beyond Constant Factors

- **Some problems have no constant-factor approximation** (unless $\mathrm{P} = \mathrm{NP}$). For Set Cover on n elements, a simple greedy algorithm achieves ratio about $\ln n$, and doing better than $(1 - \varepsilon) \ln n$ is NP-hard (Dinur–Steurer, 2014).
- **APX** is the class of problems with a constant-factor approximation. A problem is APX-hard if, for some constant $\varepsilon > 0$, approximating it within $1 + \varepsilon$ is NP-hard. Vertex cover is both in APX and APX-hard.
- **Some NP-hard problems can be approximated arbitrarily well.** Knapsack and Euclidean TSP admit a $(1 + \varepsilon)$-approximation in polynomial time for every fixed $\varepsilon > 0$.

To look up: PTAS, FPTAS, the PCP theorem, hardness of approximation.