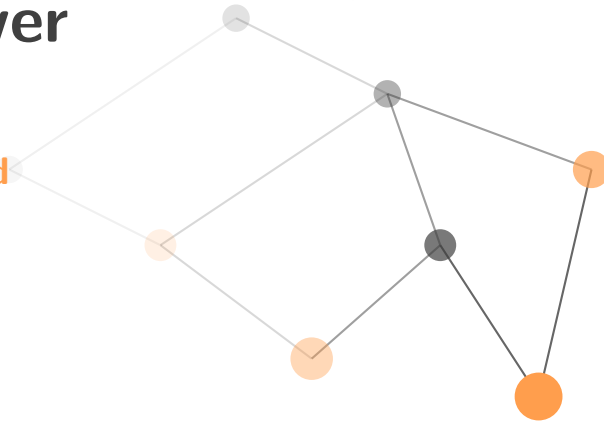
elicsir
Elevating Italian Computer Science Innovation and Research

The Vertex Cover Problem:

Computational Bounds and Studied Approaches



Nicolò Cappa

Mentor: Vincenzo Cutello

March 23, 2026

Table of Contents

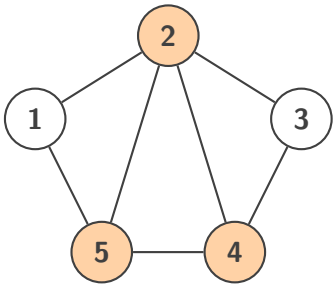
Vertex Cover: Definition

Vertex Cover

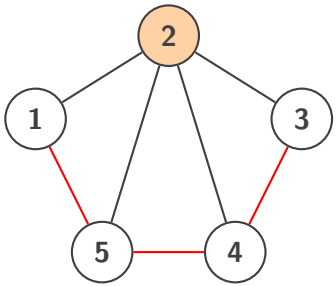
Given an undirected graph $G = (V, E)$, a set $X \subseteq V$ is a **vertex cover** if

$$\forall e \in E : X \cap e \neq \emptyset.$$

$X = \{2, 4, 5\}$ is a vertex cover



$X = \{2\}$ **NOT** a vertex cover



Karp, 1972 "Reducibility among combinatorial problems"

5. NODE COVER
INPUT: graph G', positive integer ℓ
PROPERTY: There is a set R ⊆ N' such that |R| ≤ ℓ and every arc is incident with some node in R.

Given a positive integer $\ell \in \mathbb{N}$, does G have a vertex cover X of size $|X| \leq \ell$?

Karp (1972)

Vertex Cover (decision) is **NP-complete**

Minimum Vertex Cover: Definition

Minimum Vertex Cover π

Input$_\pi$:

- $G = (V, E)$, undirected graph

Amm$_\pi$:

- $X \subseteq V$ s.t $\forall e \in E : X \cap e \neq \emptyset$

Target$_\pi$:

- $|X|$

Type$_\pi$:

- Minimum

From decision to optimization

Iterative decision

The **Minimum Vertex Cover** optimization problem can be solved by iteratively solving its decision version for decreasing values of ℓ.

Karp (1972)

Minimum Vertex Cover (Optimization) is **NPO-complete**

The 2-approximation algorithm

APPROX-VERTEX-COVER

1: $X \leftarrow \emptyset$
2: $E' \leftarrow E$
3: **while** $E' \neq \emptyset$ **do**
4: choose any edge $\{u, v\} \in E'$
5: $X \leftarrow X \cup \{u, v\}$
6: Remove from E'every edge incident on either u *or* v
7: **end while**
8: **return** X

Theorem

Approx-Vertex-Cover is a **polynomial-time 2-approximation** algorithm.

Proof.

- Let A denote the set of edges selected in line 4
- To cover the edges in A, any vertex cover X (valid also for the optimum X^*) must include an endpoint of each edge in A
- No two edges in A share an endpoint, since once we pick an edge, we delete all edges incident on either of its endpoints from E'
- To cover all edges in A, we need at least one (distinct) endpoint per edge, since two edges in A never share endpoints, hence $|X^*| \geq |A|$
- Each time an edge is selected from E', neither of its endpoints is in X, yielding the exact upper bound $|X| = 2|A|$
- Putting all together $|X| = 2|A| \leq 2|X^*|$

Inapproximability wall

Dinur and Safra (2005)

Unless $P = NP$, Minimum Vertex Cover cannot be approximated in polynomial time within a factor of $\approx \mathbf{1.3606}$

Inapproximability wall

Dinur and Safra (2005)

Unless $P = NP$, Minimum Vertex Cover cannot be approximated in polynomial time within a factor of $\approx \mathbf{1.3606}$

Khot and Regev (2008)

Under the Unique Games Conjecture (**UGC**), Minimum Vertex Cover is hard to approximate within a factor $2 - \epsilon$, for every constant $\epsilon > 0$

Inapproximability wall

Dinur and Safra (2005)

Unless $P = NP$, Minimum Vertex Cover cannot be approximated in polynomial time within a factor of $\approx \mathbf{1.3606}$

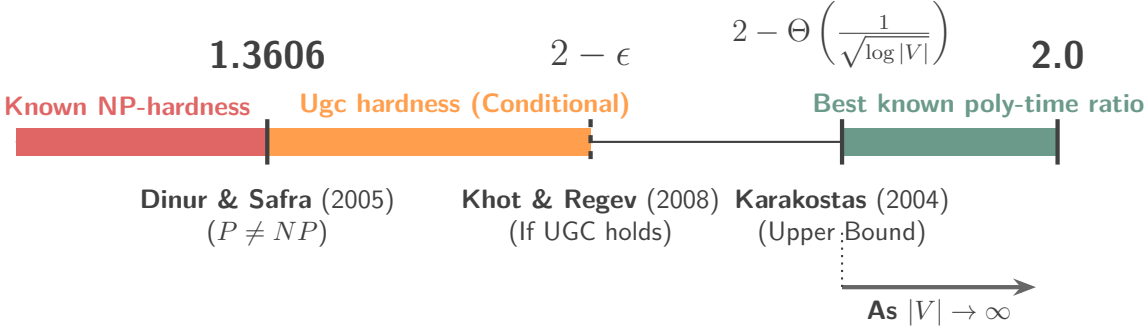
Khot and Regev (2008)

Under the Unique Games Conjecture (**UGC**), Minimum Vertex Cover is hard to approximate within a factor $2 - \epsilon$, for every constant $\epsilon > 0$

The best polynomial-time ratio is essentially **stuck at 2**.

What can we do **in practice**?

Intractability boundary



Landscape

1. Greedy Methods

- **Approx-Vertex-Cover** (maximal matching)
- **MDG** (max-degree greedy)
- **NOVCA** (min-degree + support, add neighbors)

Landscape

1. Greedy Methods

- **Approx-Vertex-Cover** (maximal matching)
- **MDG** (max-degree greedy)
- **NOVCA** (min-degree + support, add neighbors)

2. Evolutionary/Bio-inspired

- $(1+1)$-EA & Populations
- Ant-colony optimization
- Hybrid approaches

Landscape

1. Greedy Methods

- **Approx-Vertex-Cover** (maximal matching)
- **MDG** (max-degree greedy)
- **NOVCA** (min-degree + support, add neighbors)

2. Evolutionary/Bio-inspired

- $(1+1)$-EA & Populations
- Ant-colony optimization
- Hybrid approaches

3. Local Search

- **FastVC** (massive graphs)
- **NuMVC** (edge-weighting LS)
- **RLS** (stochastic local search)

Local search pipeline

Construction trade-off

Quality of initial cover vs **Time to build**

Notations

- For a vertex $v \notin X$, $gain(v)$ is defined as the number of uncovered edges that would become covered adding v to X
- For a vertex $v \in X$, $loss(v)$ is defined as number of covered edges that would become uncovered removing v from X

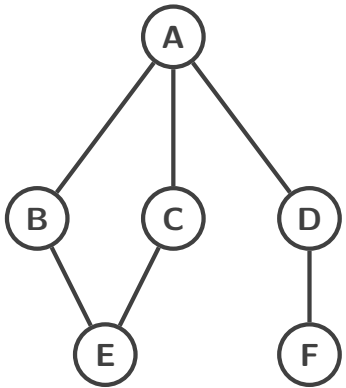
Baseline: greedy construction

Papadimitrious and Steiglitz (1982)

*Repeat the following operations until X becomes a vertex cover: select a vertex $v \notin X$ with the maximum **gain** to add into X, breaking ties randomly.*

Baseline: greedy construction

Algorithm: iteratively pick the vertex with maximum $gain$



Execution state

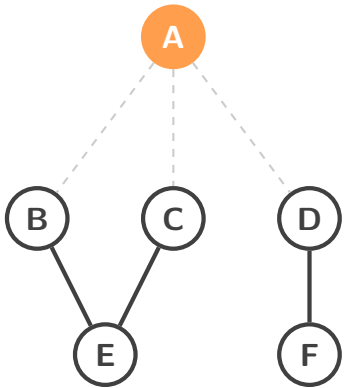
$X = \emptyset$

Current graph: $E' = E$

Max gain: $gain(A) = 3$

Baseline: greedy construction

Algorithm: iteratively pick the vertex with maximum $gain$



Execution state

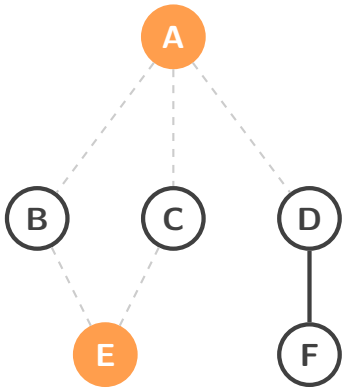
$X = \{\mathbf{A}\}$

Current graph: E' updated

Max gain: $gain(E) = 2$

Baseline: greedy construction

Algorithm: iteratively pick the vertex with maximum $gain$



Execution state

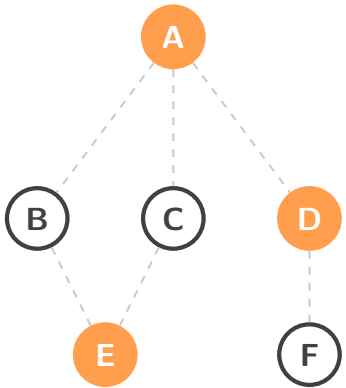
$X = \{A, \mathbf{E}\}$

Current graph: E' updated

Max gain: $gain(D) = 1$

Baseline: greedy construction

Algorithm: iteratively pick the vertex with maximum $gain$



Execution state

$X = \{A, E, \mathbf{D}\}$

Current graph: $E' = \emptyset$

X **is a cover**

Baseline: greedy construction

Complexity: $\mathcal{O}(|V|^2)$

- Scan V in each iteration to find $v = \arg\max_{v \in V \setminus X}(gain(v))$
- It has a complexity of $\Theta(|V|)$ for each iteration
- Let $l = |X|$ be the number of iterations; note that $l \leq |V|$
- Then the complexity is $\Theta(l \cdot |V|) = \mathcal{O}(|V|^2)$

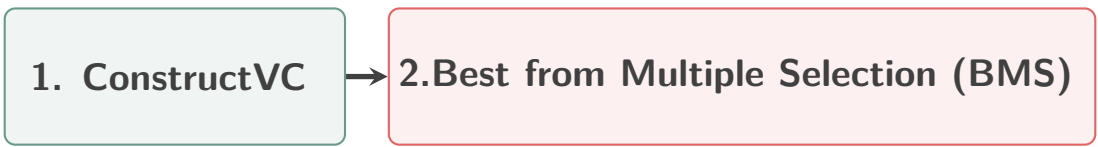
FastVC, Cai (2015)

FastVC

Is a **local search** algorithm for Minimum Vertex Cover, designed for applications in **massive graphs**, based on two low complexity heuristics

FastVC

Is a **local search** algorithm for Minimum Vertex Cover, designed for applications in **massive graphs**, based on two low complexity heuristics



ConstructVC: two phases algorithm

1. Extending phase

Goal: Rapidly build a valid cover

- Scan the edge set E exactly once
- Add vertices to X until all edges are covered
- Prioritize high-degree vertices

2. Shrinking phase

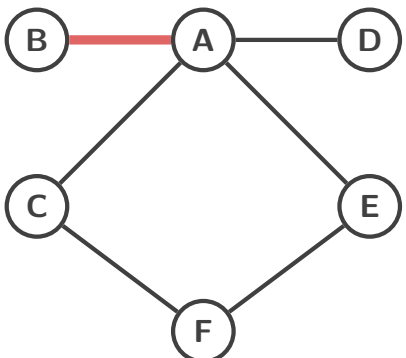
Goal: Remove redundant vertices

- Scan the constructed cover X
- Identify vertices that are no longer strictly necessary
- Remove them to make the cover **minimal**

The Rule

Starting with an empty set X, check each edge $e \in E$ one by one. If the edge is currently **uncovered**, add its endpoint with the **higher degree** into X

Algorithm: Iterate edges, add the max-degree endpoint if uncovered.



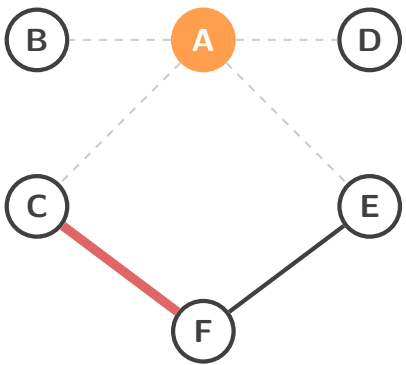
Execution State

$X = \emptyset$

Check Edge: (A,B)
$deg(A) = 4 > deg(B) = 1$

Action: Add **A**

Algorithm: Iterate edges, add the max-degree endpoint if uncovered.



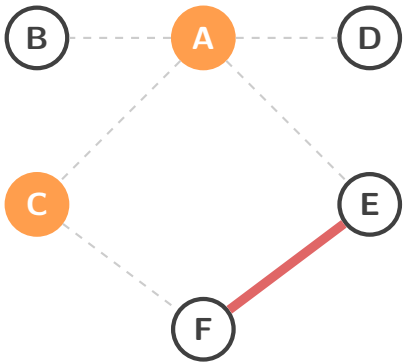
Execution State

$X = \{\mathbf{A}\}$

Check Edge: (C,F)
$deg(C) = 2 == deg(F) = 2$

Action: Add **C**

Algorithm: Iterate edges, add the max-degree endpoint if uncovered.



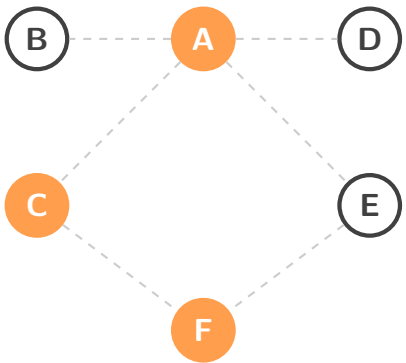
Execution State

$X = \{A, \mathbf{C}\}$

Check Edge: (E,F)
$deg(E) = 2 == deg(F) = 2$

Action: Add **F**

Algorithm: Iterate edges, add the max-degree endpoint if uncovered.



Execution State

$X = \{A, C, \mathbf{F}\}$

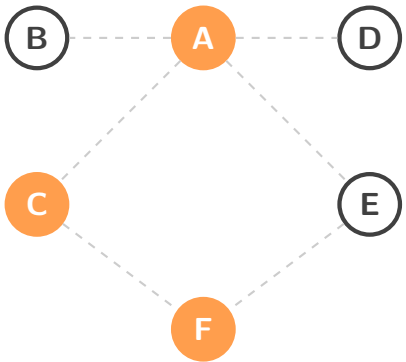
Status: All edges covered.

Valid cover obtained.

The Rule

Calculate the **loss** value for every vertex in X. Scan X, and if any vertex v has $\mathbf{loss(v) = 0}$, remove it and update the loss of its neighbors.

Algorithm: Scan the cover X and remove vertices with $loss = 0$.



Execution State

$X = \{A, C, F\}$

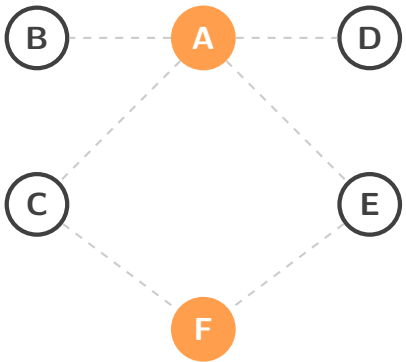
Calculate Loss:

$loss(A) = 3$

$loss(F) = 1$

$loss(C) = \mathbf{0}$

Algorithm: Scan the cover X and remove vertices with $loss = 0$.



Execution State

$\mathbf{X = \{A, F\}}$

Status: Minimal Cover.

Construction Complete.

FastVC construction phase (Cai, 2015)

ConstructVC

1: **Input:** $G = (V, E)$
2: $X \leftarrow \emptyset$
3: **// 1. Extending phase**
4: **for all** $e = (u, v) \in E$ **do**
5: **if** e is uncovered by X **then**
6: add the endpoint of e with higher degree into X
7: **end if**
8: **end for**
9: **// 2. Shrinking phase**
10: **for all** $v \in X$ **do**
11: **if** $loss(v) == 0$ **then**
12: $X \leftarrow X \setminus \{v\}$
13: **end if**
14: **end for**
15: **return** X

FastVC construction phase (Cai, 2015)

Theorem

The set X returned by the *ConstructVC* procedure is a **minimal vertex cover**

FastVC construction phase (Cai, 2015)

Why is it a Cover?

The **extending phase** explicitly checks every edge in E. If uncovered, it adds an endpoint. No edge is left behind

FastVC construction phase (Cai, 2015)

Why is it a Cover?

The **extending phase** explicitly checks every edge in E. If uncovered, it adds an endpoint. No edge is left behind

Why is it Minimal?

The **shrinking phase** only removes a vertex v if $loss(v) = 0$. Removing it produces **zero** uncovered edges.

FastVC construction phase (Cai, 2015)

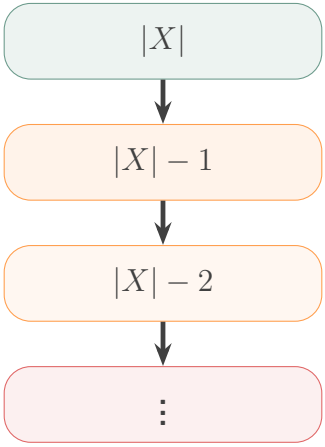
ConstructVC Complexity: $\mathcal{O}(|E|)$

- Let X^+ denote the cover obtained after the Extending phase
- **Extending phase:** scanning the edges takes $\mathcal{O}(|E|)$
- **Loss initialization:** takes $\mathcal{O}(|X^+| + |E|)$. Since we add at most one vertex per iteration, $|X^+| \leq |E|$, making this step $\mathcal{O}(|E|)$
- **Shrinking phase:** the cost is bounded by the total number of loss updates: $\sum_{v \in X^+} deg(v) < \sum_{v \in V} deg(v) = 2|E|$, giving $\mathcal{O}(|E|)$

Massive real-world graphs are **sparse graphs**.

$$|\mathbf{E}| \ll |\mathbf{V}|^{\mathbf{2}}$$

Iterate until time cutoff



NuMVC

Local search solver for MinVC on standard benchmarks. Two key ideas make it effective:

1. Two-stage exchange

- Remove and add are selected **separately**
- Avoids evaluating all exchange pairs
- Reduces move selection from $|R| \cdot |A|$ to $|R| + |A|$

2. Edge weighting with forgetting

- Uncovered edges receive larger weights
- Old weights are periodically decreased (multiplied by ρ with $\rho \in (0, 1)$)
- Diversify search (exploration)

NuMVC: move rule

Weighted score

NuMVC uses edge weights. $loss_w$ and $gain_w$ denote the weighted versions of $loss$ and $gain$

$$dscore(v) = \begin{cases} -loss_w(v), & v \in X, \\ gain_w(v), & v \notin X \end{cases}$$

Two-stage exchange

- **Remove:** choose $u \in X$ with highest $dscore(u)$
- choose one uncovered edge $e = \{x, y\}$ uniformly at random
- **Add:** insert the endpoint of e with higher $dscore$

NuMVC: the bottleneck

What NuMVC achieves

Two-stage exchange reduces move selection from $|R| \cdot |A|$ to $|R| + |A|$. The add step is further restricted to the endpoints of one uncovered edge.

What is still expensive

The **remove step** still scans the entire current cover:

$$u = \arg\max_{v \in X} dscore(v) \qquad \Longrightarrow \qquad \mathcal{O}(|X|) \text{ per iteration}$$

- Good on small/medium benchmarks (DIMACS, BHOSLIB)

Best from Multiple Selection (BMS)

BMS rule

- Sample k vertices uniformly from the current cover X
- Compare only these k candidates
- Remove the one with minimum $loss$

$$u = \arg\min_{v \in S} loss(v), \qquad S \subseteq X,\ |S| = k$$

Approximate best-choice, at much lower cost

BMS: probabilistic quality

Best from Multiple Selection

Sample k vertices from X with replacement, and remove the one with minimum $loss$ among them

Quality guarantee

Let E be the event that the vertex chosen by BMS has $loss$ not greater than $\rho|X|$ vertices in X. Then

$$\Pr(E) \geq 1 - \left(\frac{\rho|X| - 1}{|X|}\right)^k > 1 - \rho^k$$

BMS: probabilistic quality

FastVC setting

In FastVC, $k = 50$. For $\rho = 0.9$,

$$\Pr(E) > 1 - 0.9^{50} > 0.9948$$

so BMS returns a **good-quality** removal with very high probability

BMS: complexity

Exact best-choice

scan all of X

$\mathcal{O}(|X|)$

BMS

scan only k samples

$\mathcal{O}(k)$

FastVC choice

With fixed k (they used $k = 50$), the selection cost becomes

$$\mathcal{O}(k) = \mathcal{O}(1)$$

Three design points

Method	Construction	Remove step	Main idea
MDG	$\mathcal{O}(\|V\|^2)$	–	greedy baseline
NuMVC	$\mathcal{O}(\|V\|^2)$	$\mathcal{O}(\|X\|)$	stronger guidance
FastVC	$\mathcal{O}(\|E\|)$	$\mathcal{O}(k) = \mathcal{O}(1)$	cheap moves

Experimental setting

Setup

- Each algorithm is executed **10 times** on every instance
- Time cutoff: **1000 seconds**
- Reported metrics: $X_{\min}$, X_{avg}

Reported values

- X_{avg}: average cover size over 10 runs
- $X_{\min}$: best cover size found over 10 runs

Representative quality results

Graph	$\lvert V\rvert$	$\lvert E\rvert$	NuMVC $X_{\min}(X_{avg})$	FastVC $X_{\min}(X_{avg})$
web-wikipedia2009	1864433	4507315	n/a	**648317** (648321.8)
sc-msdoor	415863	9378650	381569 (381574.6)	**381558** (381558.9)
sc-shipsec1	140385	1707759	117477 (117536.9)	**117318** (117337.5)
soc-flickr	513969	3190452	153343 (153352.7)	**153272** (153272.0)
web-it-2004	509338	7178413	414741 (414758.7)	**414671** (414676.3)
soc-douban	154908	327162	**8685** (8685.0)	**8685** (8685.0)

Selected rows adapted from Tables 3–4 in Cai et al. Bold indicates the smaller reported cover size. $X_{\min}$*: best found over 10 runs;* X_{avg}*: average over 10 runs; true optimum is not reported for these instances.*

Running time on equal-quality instances

Graph	$\|V\|$	$\|E\|$	NuMVC time	FastVC time
soc-BlogCatalog	88784	2093195	448.804	**1.9**
soc-douban	154908	327162	1.374	**0.06**
soc-epinions	26588	100120	95.104	**0.138**
tech-WHOIS	7476	56943	0.454	$<$**0.01**
web-indochina-2004	11358	47606	17.242	**0.091**
web-uk-2005	129632	11744049	0.285	**0.06**

Selected equal-quality instances. Lower is better.

Takeaway

Main message

- For hard optimization problems, we often seek a **good** solution rather than a provably optimal one
- For Minimum Vertex Cover, if **UGC holds** it is hard to approximate within any factor of 2
- **Local search** heuristics trade theoretical guarantees for practical quality
- In this setting, performance depends on a **trade-off** between **solution quality** and the computational cost of the heuristic itself
- **FastVC** replaces expensive heuristics with cheap approximate ones ($\mathcal{O}(|E|)$ construction, $\mathcal{O}(1)$ BMS removal) and dominates on massive real-world graphs

On massive instances, cheaper moves beat smarter ones

Questions?

References I

Complexity and approximation

- R. M. Karp. *Reducibility Among Combinatorial Problems*. In *Complexity of Computer Computations*, 1972.
- I. Dinur and S. Safra. *On the Hardness of Approximating Minimum Vertex Cover*. *Annals of Mathematics*, 2005.
- G. Karakostas. *A Better Approximation Ratio for the Vertex Cover Problem*, 2004.

UGC and hardness

- S. Khot. *On the Power of Unique 2-Prover 1-Round Games*. In *STOC*, 2002.
- S. Khot and O. Regev. *Vertex Cover Might Be Hard to Approximate to Within* $2-\varepsilon$. *JCSS*, 2008.

Local search for MinVC

- S. Cai, K. Su, C. Luo, and A. Sattar. *NuMVC: An Efficient Local Search Algorithm for Minimum Vertex Cover*. *JAIR*, 2013.
- S. Cai, J. Lin, and C. Luo. *Finding a Small Vertex Cover in Massive Sparse Graphs: Construct, Local Search, and Preprocess*. *JAIR*, 2017.

References II

Evolutionary and metaheuristic approaches

- P. S. Oliveto, J. He, and X. Yao. *Analysis of the* $(1+1)$*-EA for Finding Approximate Solutions to Vertex Cover Problems. IEEE Transactions on Evolutionary Computation*, 2009.
- P. S. Oliveto, J. He, and X. Yao. *Analysis of Population-Based Evolutionary Algorithms for the Vertex Cover Problem.* In *CEC*, 2008.
- S. Khuri and T. Bäck. *An Evolutionary Heuristic for the Minimum Vertex Cover Problem*, 1994.
- S. J. Shyu, P.-Y. Yin, and B. M. T. Lin. *An Ant Colony Optimization Algorithm for the Minimum Weight Vertex Cover Problem. Annals of Operations Research*, 2004.

Neural / learned heuristic

- E. Zhu, Q. Bao, Y. Zhang, and C. Liu. *Optimizing Minimum Vertex Cover Solving via a GCN-assisted Heuristic Algorithm*, 2025.

Backup: Unique Games Conjecture

Definition (unique 2-prover 1-round game)

A 2-prover 1-round game is **unique** if, for every question pair and every answer of one prover, there is **exactly one** accepting answer of the other prover.

Unique Games Conjecture (Khot, 2002)

For arbitrarily small constants $\varepsilon, \delta > 0$, there exists a constant

$$k = k(\varepsilon, \delta)$$

such that it is NP-hard to determine whether a unique 2-prover game with answers from a domain of size k has value

$$\mathrm{val}(G) \geq 1 - \varepsilon \qquad \text{or} \qquad \mathrm{val}(G) \leq \delta.$$