Primer on Hardness of Approximation

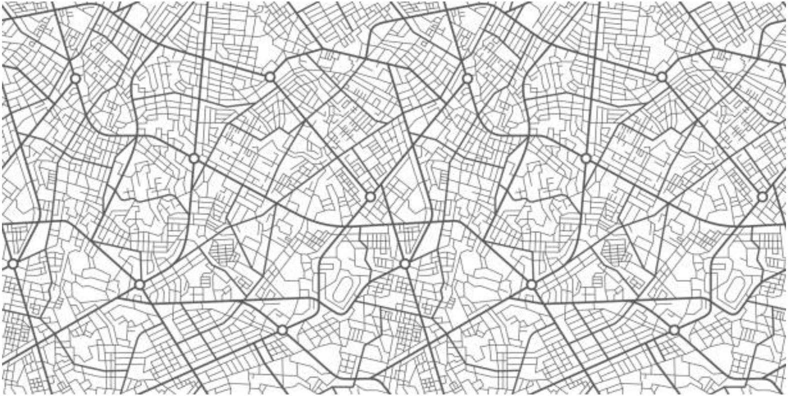
Sofia Vazquez Alferez

LAMSADE Spring School
April 2024

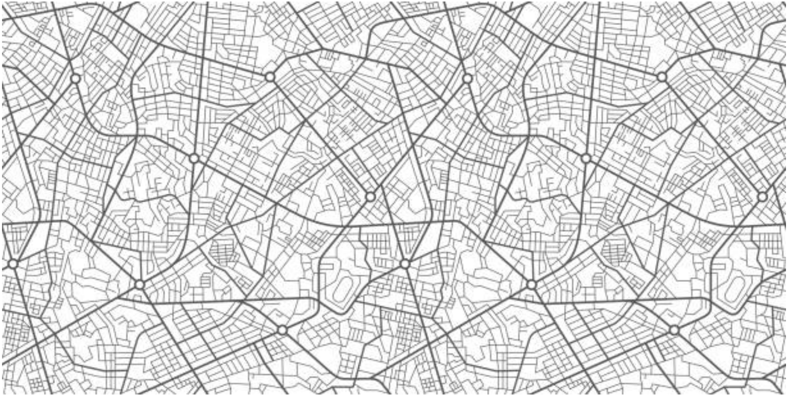
Table of Contents

Two Companion Problems

A problem

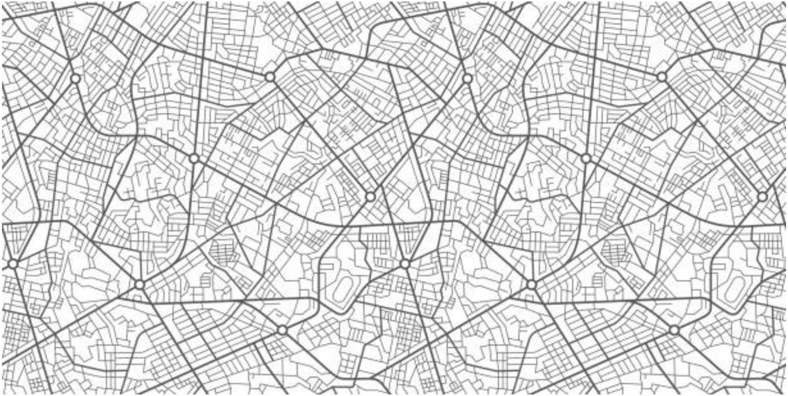


A problem



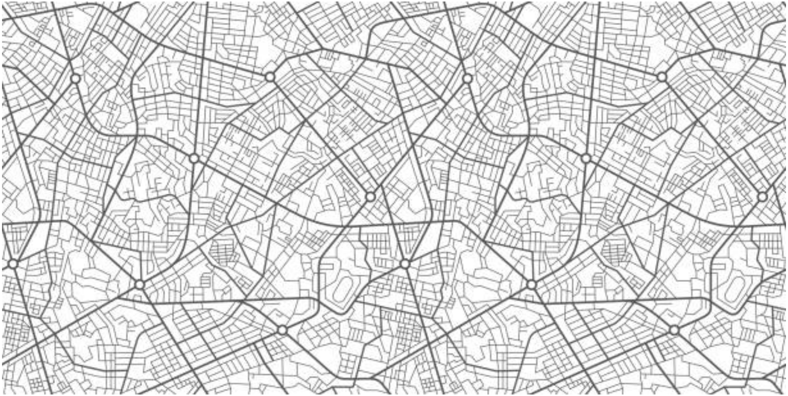
- Monitor street traffic efficiently.

A problem



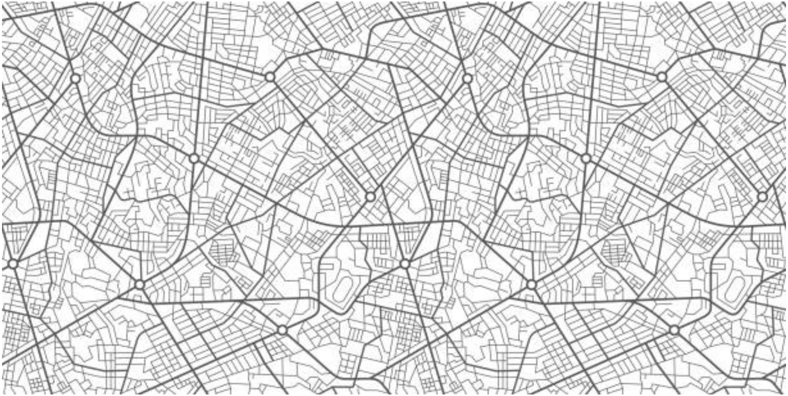
- Monitor street traffic efficiently.
- Goal: use the smallest number of cameras whilst ensuring every junction is covered.

A problem



This task resembles the Minimum Vertex Cover problem!

A problem



This task resembles the Minimum Vertex Cover problem! (Junctions are edges and cameras are vertices)

Minimum Vertex Cover

Minimum Vertex Cover

Given: A graph $G = (V, E)$.

Minimum Vertex Cover

Given: A graph $G = (V, E)$.

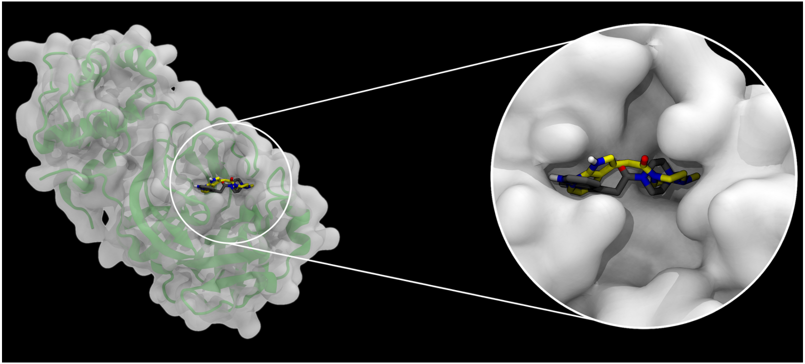
Find: A minimum subset $C \subseteq V$, such that C "covers" all edges in E.

Minimum Vertex Cover

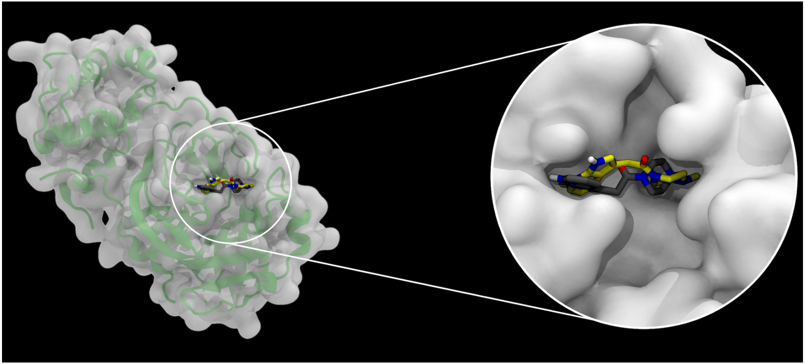
Given: A graph $G = (V, E)$.

Find: A minimum subset $C \subseteq V$, such that C "covers" all edges in E.
i.e., for every edge $uv \in E$ either $u \in C$ or $v \in C$, or both.

An other problem

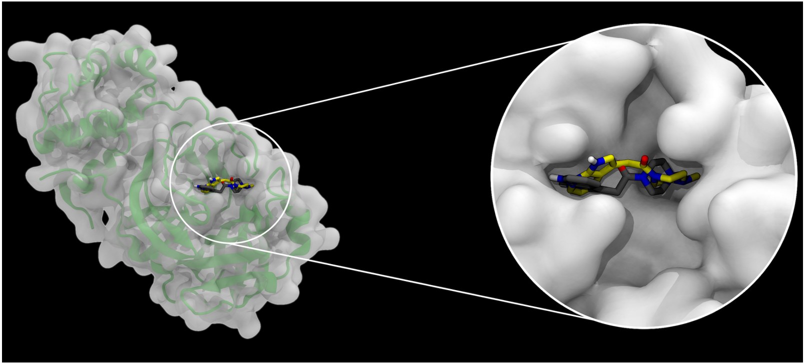


An other problem



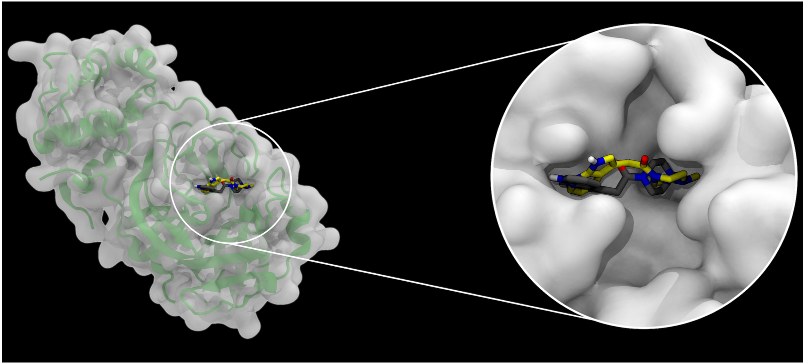
- Predict the mode of binding of a small molecule to a receptor.

An other problem



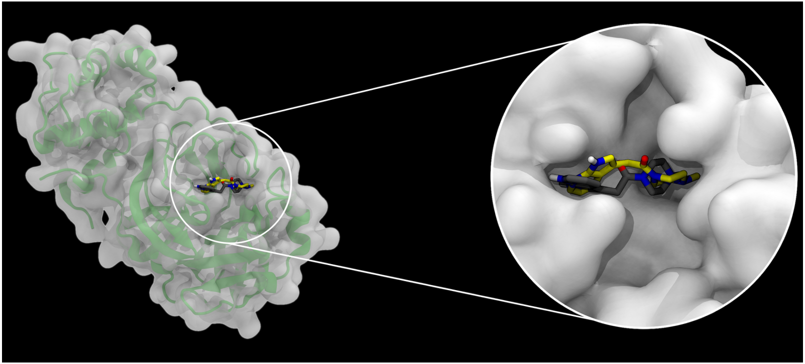
- Predict the mode of binding of a small molecule to a receptor.
- Simplified Model:

An other problem



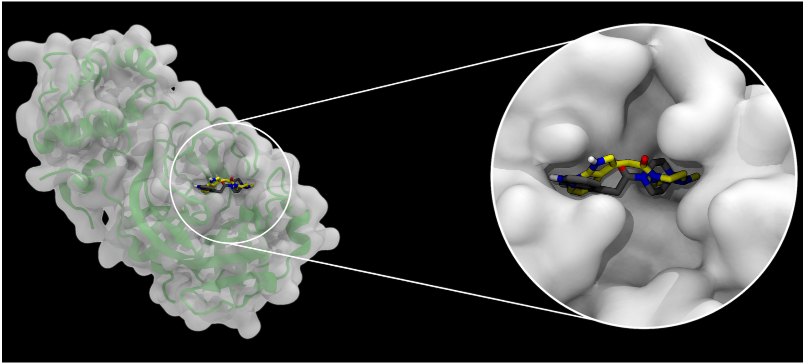
- Predict the mode of binding of a small molecule to a receptor.
- Simplified Model:
  - Vertices: (RECEPTOR POINT, MOLECULE POINT) pairs.

An other problem



- Predict the mode of binding of a small molecule to a receptor.
- Simplified Model:
  - Vertices: (RECEPTOR POINT, MOLECULE POINT) pairs.
  - Edges: (R1,M1)–(R2,M2) if distance(R1,R2) $\approx$ distance(M1, M2)

An other problem



- Predict the mode of binding of a small molecule to a receptor.
- Simplified Model:
  - Vertices: (RECEPTOR POINT, MOLECULE POINT) pairs.
  - Edges: (R1,M1)–(R2,M2) if distance(R1,R2) $\approx$ distance(M1, M2)
- Find largest clique.

Maximum Clique

Maximum Clique

Given: A graph $G = (V, E)$.

Maximum Clique

Given: A graph $G = (V, E)$.

Find: A maximum clique in the graph.

Maximum Clique

Given: A graph $G = (V, E)$.

Find: A maximum clique in the graph.
i.e. a subset $C \subseteq V$ of maximum size such that $G[C]$ is a complete graph.

Our two friends:

Minimum Vertex Cover

Given: A graph $G = (V, E)$.

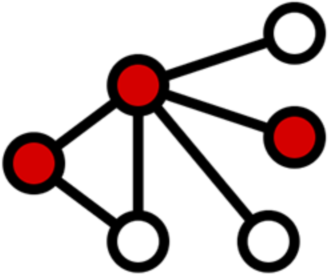
Find: A minimum subset $C \subseteq V$, such that C "covers" all edges in E.

Maximum Clique

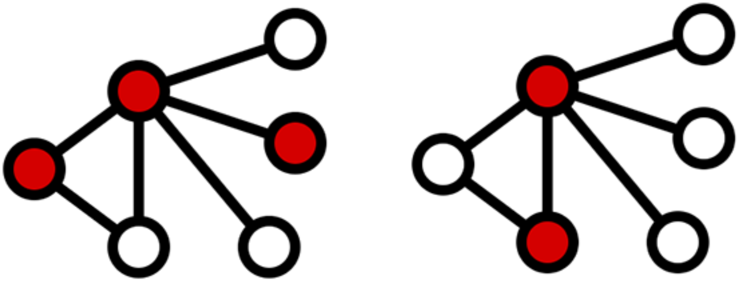
Given: A graph $G = (V, E)$.

Find: A maximum clique in the graph.

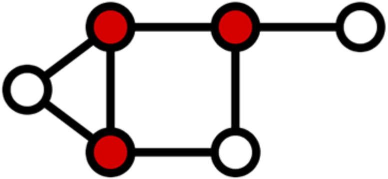
Are they REALLY your friends?



Are they REALLY your friends?

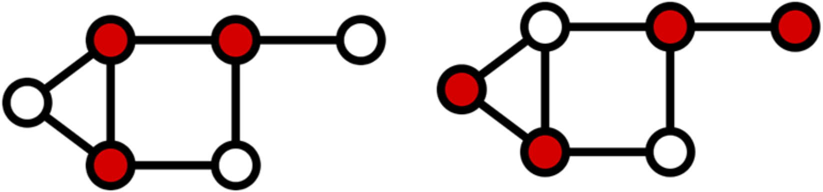


Are they REALLY your friends?



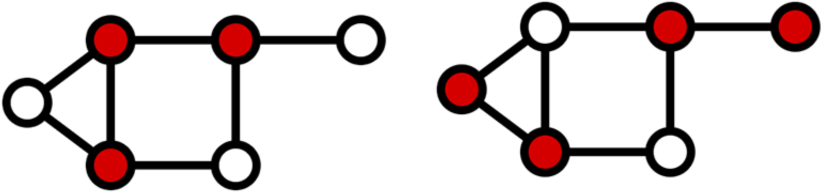
[1] Images: http://isaacsteele.com/cv/edu/college/junior/vertexcover.shtml

Are they REALLY your friends?



[1] Images: http://isaacsteele.com/cv/edu/college/junior/vertexcover.shtml

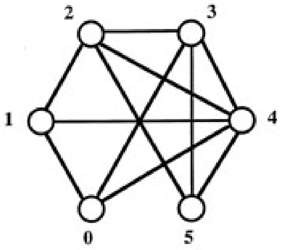
Are they REALLY your friends?



[1]

[1] Images: http://isaacsteele.com/cv/edu/college/junior/vertexcover.shtml

Are they REALLY your friends?



[2]

[2] Images:https://cs.stanford.edu/people/eroberts/courses/soco/projects/2003-04/dna-computing/clique.htm

Some motivation for Hardness of Approximation

Some facts about our friends

Some facts about our friends

- MINIMUM VERTEX COVER and MAXIMUM CLIQUE are both NP-hard.

Some facts about our friends

- MINIMUM VERTEX COVER and MAXIMUM CLIQUE are both NP-hard.
- What do we do when we see hard problems?

Some facts about our friends

- Minimum Vertex Cover and Maximum Clique are both NP-hard.
- What do we do when we see hard problems?
  - Design algorithm that gives optimal solutions but is efficient only on some instances.

Some facts about our friends

- Minimum Vertex Cover and Maximum Clique are both NP-hard.
- What do we do when we see hard problems?
  - Design algorithm that gives optimal solutions but is efficient only on some instances.
  - Design an algorithm that is always efficient but gives sub-optimal solutions.

Some facts about our friends

- MINIMUM VERTEX COVER and MAXIMUM CLIQUE are both NP-hard.
- What do we do when we see hard problems?
  - Design algorithm that gives optimal solutions but is efficient only on some instances.
  - Design an algorithm that is always efficient but gives sub-optimal solutions.(Approximation algorithms)

Some facts about our friends

- MINIMUM VERTEX COVER and MAXIMUM CLIQUE are both NP-hard.
- What do we do when we see hard problems?
  - Design algorithm that gives optimal solutions but is efficient only on some instances.
  - Design an algorithm that is always efficient but gives sub-optimal solutions.(Approximation algorithms)
    - Sometimes impossible!

Definition of an approximation algorithm

α-approximation (for minimization)

For $\alpha \geq 1$, an algorithm is an α-approximation algorithm for a minimization problem if on every input instance the algorithm finds a solution with cost $\leq \alpha \cdot OPT$.

Definitions

α-approximation (for minimization)

For $\alpha \geq 1$, an algorithm is an α-approximation algorithm for a minimization problem if on every input instance the algorithm finds a solution with cost $\leq \alpha \cdot OPT$.

α-approximation (for maximization)

For $\alpha \geq 1$, an algorithm is an α-approximation algorithm for a maximization problem if on every input instance the algorithm finds a solution with cost $\geq \frac{1}{\alpha} \cdot OPT$.

Definitions

α-approximation (for minimization)

For $\alpha \geq 1$, an algorithm is an α-approximation algorithm for a minimization problem if on every input instance the algorithm finds a solution with cost $\leq \alpha \cdot OPT$.

α-approximation (for maximization)

For $\alpha \geq 1$, an algorithm is an α-approximation algorithm for a maximization problem if on every input instance the algorithm finds a solution with cost $\geq \frac{1}{\alpha} \cdot OPT$.

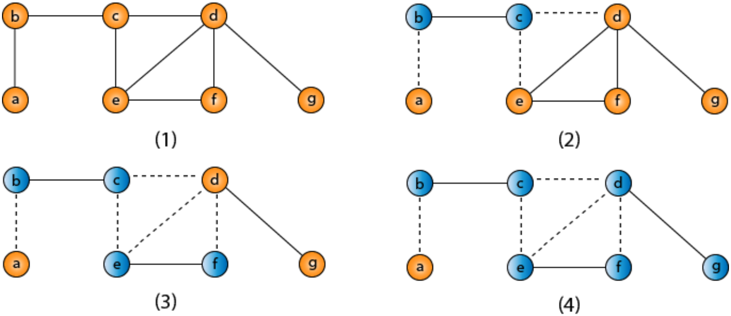
So the smaller α is the better.

Example: VC

Algorithm 1: APPROX-VERTEX-COVER(G)

1 $C \leftarrow \emptyset$
2 while $E \neq \emptyset$
  pick any $\{u, v\} \in E$
  $C \leftarrow C \cup \{u, v\}$
  delete all eges incident to either u or v

return C



Example: VC

This is a 2-approximation algorithm.

Example: VC

This is a 2-approximation algorithm.

- It gives a vertex cover.

Example: VC

This is a 2-approximation algorithm.

- It gives a vertex cover.
- The optimum vertex cover must cover every edge in C. So, it must include at least one of the endpoints of each edge in C. Thus $OPT \geq 1/2|C|$.

How to prove hardness

Proving Hardness - Exact Optimization

When we prove that a combinatorial problem C is NP-hard, we usually pick our favorite NP-complete combinatorial problem L and we show a reduction that:

When we prove that a combinatorial problem C is NP-hard, we usually pick our favorite NP-complete combinatorial problem L and we show a reduction that:

- maps every YES instance of L to a YES instance of C.

Proving Hardness - Exact Optimization

When we prove that a combinatorial problem C is NP-hard, we usually pick our favorite NP-complete combinatorial problem L and we show a reduction that:

- maps every YES instance of L to a YES instance of C.
- maps every NO instance of L to a NO instance of C.

Proving Hardness of Approximation

To prove that a problem C is hard to approximate we need a (more robust) reduction from your favourite NP-hard problem L that:

Proving Hardness of Approximation

To prove that a problem C is hard to approximate we need a (more robust) reduction from your favourite NP-hard problem L that:

- maps every YES instance of L to a YES instance of C

Proving Hardness of Approximation

To prove that a problem C is hard to approximate we need a (more robust) reduction from your favourite NP-hard problem L that:

- maps every YES instance of L to a YES instance of C
- maps every NO instance of L to a VERY-MUCH-NO instance of C.

Proving Hardness of Approximation

To prove that a problem C is hard to approximate we need a (more robust) reduction from your favourite NP-hard problem L that:

- maps every YES instance of L to a YES instance of C
- maps every NO instance of L to a VERY-MUCH-NO instance of C.

Such that if we could approximate C we would be able to distinguish between instances of L

Getting some intuition:

Suppose we had an instance ϕ of SAT and that we had a reduction such that:

Getting some intuition:

Suppose we had an instance ϕ of SAT and that we had a reduction such that:

- If ϕ is satisfiable, it gets mapped to (G, k), where (G, k) is a yes instance of clique (there exists a clique of size k).

Getting some intuition:

Suppose we had an instance ϕ of SAT and that we had a reduction such that:

- If ϕ is satisfiable, it gets mapped to (G, k), where (G, k) is a yes instance of clique (there exists a clique of size k).
- If ϕ is not satisfiable, it gets mapped to instance (H, k) of clique where H has no clique of size $k/3$

Getting some intuition:

Suppose we had an instance ϕ of SAT and that we had a reduction such that:

- If ϕ is satisfiable, it gets mapped to (G, k), where (G, k) is a yes instance of clique (there exists a clique of size k).
- If ϕ is not satisfiable, it gets mapped to instance (H, k) of clique where H has no clique of size $k/3$

If a 2-approximation algorithm A for MAX CLIQUE exists, then:

Getting some intuition:

Suppose we had an instance ϕ of SAT and that we had a reduction such that:

- If ϕ is satisfiable, it gets mapped to (G, k), where (G, k) is a yes instance of clique (there exists a clique of size k).
- If ϕ is not satisfiable, it gets mapped to instance (H, k) of clique where H has no clique of size $k/3$

If a 2-approximation algorithm A for MAX CLIQUE exists, then:

- $A(G) \geq k/2 \leftarrow$ we know $k/2$ is the worst A will return.

Getting some intuition:

Suppose we had an instance ϕ of SAT and that we had a reduction such that:

- If ϕ is satisfiable, it gets mapped to (G, k), where (G, k) is a yes instance of clique (there exists a clique of size k).
- If ϕ is not satisfiable, it gets mapped to instance (H, k) of clique where H has no clique of size $k/3$

If a 2-approximation algorithm A for MAX CLIQUE exists, then:

- $A(G) \geq k/2 \leftarrow$ we know $k/2$ is the worst A will return.
- $A(H) \leq k/3 \leftarrow$ we know $k/3$ is the best A will return.

For exact optimization:

Theorems the heart of Hardness

For exact optimization:

Cook-Levin Theorem

Assuming $P \neq NP$ it is hard to distinguish between:

- an instance ϕ of SAT that has a satisfying assignment.
- an instance ϕ of SAT that has **no** satisfying assignment.

Theorems the heart of Hardness

For exact optimization:

Cook-Levin Theorem

Assuming $P \neq NP$ it is hard to distinguish between:

- an instance ϕ of SAT that has a satisfying assignment.
- an instance ϕ of SAT that has **no** satisfying assignment.

For approximation:

PCP Theorem

There is a constant $\epsilon_M > 0$ for which, assuming $P \neq NP$, it is hard to distinguish between:

- an instance ϕ (on m clauses) of MAX-3SAT that has a satisfying assignment (there is an assignment that satisfies all m clauses)
- an instance ϕ (on m clauses) of MAX-3SAT such that any assignment satisfies at most $(1 - \epsilon_M) \cdot m$ clauses.

An example

VC Example[3]

[3]Known: VC cannot be approximated to a factor of $\sqrt{2} - \epsilon$ for any $\epsilon > 0$

VC Example[3]

It is hard to ϵ_v-approximate VC(30)

There is a gap-preserving reduction from MAX-3SAT(29) to VC(30) that transforms a Boolean formula ϕ to a graph $G = (V, E)$ such that:

[3] Known: VC cannot be approximated to a factor of $\sqrt{2} - \epsilon$ for any $\epsilon > 0$

VC Example[3]

It is hard to ϵ_v-approximate VC(30)

There is a gap-preserving reduction from MAX-3SAT(29) to VC(30) that transforms a Boolean formula ϕ to a graph $G = (V, E)$ such that:

- if $OPT(\phi) = m$, then $OPT(G) \leq \frac{2}{3}|V|$

[3] Known: VC cannot be approximated to a factor of $\sqrt{2} - \epsilon$ for any $\epsilon > 0$

VC Example[3]

It is hard to ϵ_v-approximate VC(30)

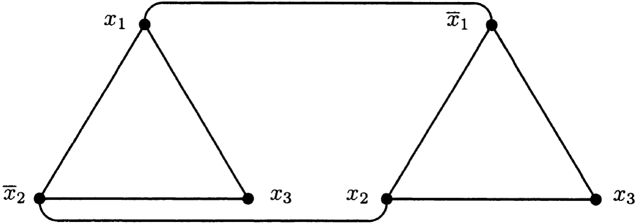
There is a gap-preserving reduction from MAX-3SAT(29) to VC(30) that transforms a Boolean formula ϕ to a graph $G = (V, E)$ such that:

- if $OPT(\phi) = m$, then $OPT(G) \leq \frac{2}{3}|V|$
- if $OPT(\phi) < (1 - \epsilon_b) \cdot m$, then $OPT(G) > (1 + \epsilon_v)\frac{2}{3}|V|$

[3]Known: VC cannot be approximated to a factor of $\sqrt{2} - \epsilon$ for any $\epsilon > 0$

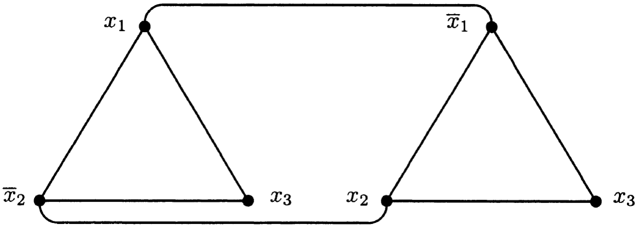
Sketch

$$(x_1 \lor \bar{x_2} \lor x_3) \land (\bar{x_1} \lor x_2 \lor x_3)$$



Sketch

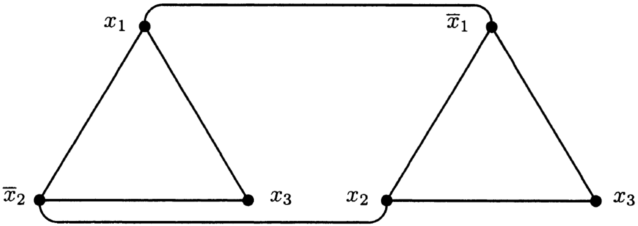
$$(x_1 \;\vee\; \bar{x}_2 \;\vee\; x_3) \;\wedge\; (\bar{x}_1 \;\vee\; x_2 \;\vee\; x_3)$$



The size of a maximum independent set in G is precisely $OPT(\phi)$.

Sketch

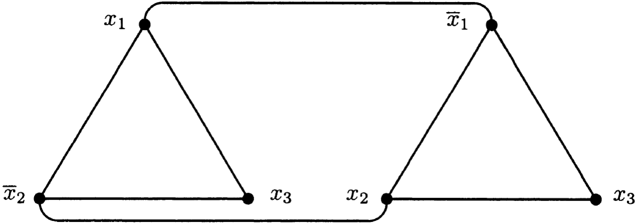
$$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$$



The size of a maximum independent set in G is precisely $OPT(\phi)$.
The complement of a maximum independent set in G is a minimum vertex cover.

Sketch

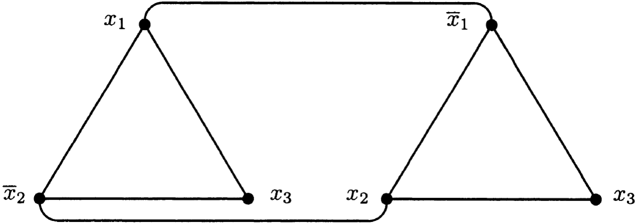
$(x_1 \vee \bar{x_2} \vee x_3) \wedge (\bar{x_1} \vee x_2 \vee x_3)$



The size of a maximum independent set in G is precisely $OPT(\phi)$.
The complement of a maximum independent set in G is a minimum vertex cover.
Therefore, if $OPT(\phi) = m$ then $OPT(G) = 2m$.

Sketch

$$(x_1 \vee \bar{x_2} \vee x_3) \wedge (\bar{x_1} \vee x_2 \vee x_3)$$



The size of a maximum independent set in G is precisely $OPT(\phi)$.
The complement of a maximum independent set in G is a minimum vertex cover.
Therefore, if $OPT(\phi) = m$ then $OPT(G) = 2m$.If $OPT(\phi) < (1 - \epsilon_b) \cdot m$, then $OPT(G) > (2 + \epsilon_b)m$.

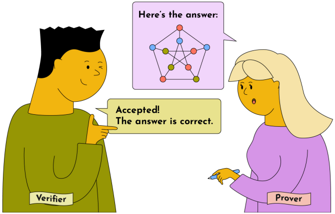
The magic of the PCP theorem

Another formulation of the PCP theorem

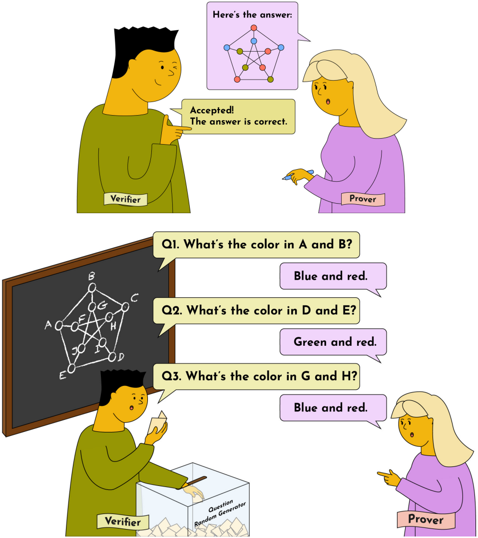
PCP Theorem

$NP = PCP(log, O(1))$

PCP explained

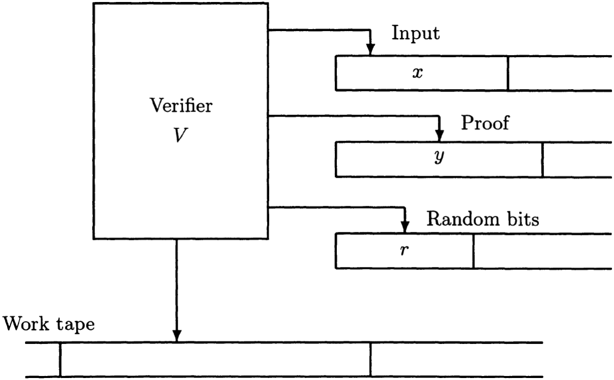


PCP explained



PCP explained

PCP explained



[4]

[4] Image: Vazirani, V. (2001) *Approximation algorithms*. Springer.

Another formulation of the PCP theorem

PCP Theorem

$NP = PCP(log, O(1))$

Another formulation of the PCP theorem

PCP Theorem

$NP = PCP(log, O(1))$

Observation

$NP = PCP(0, poly)$

Conclusion

Conclusion

- Important to study hardness of approximation for NP-hard problems.

Conclusion

- Important to study hardness of approximation for NP-hard problems.
- For hardness of approximation, need more robust reductions between combinatorial problems

Conclusion

- Important to study hardness of approximation for NP-hard problems.
- For hardness of approximation, need more robust reductions between combinatorial problems
- The PCP theorem is cool!

Resources and Aknowledgements

I took a lot of inspiration from these four sources:

- Oliveira, R. (2020) *Lecture 18: Hardness of Approximation*. https://cs.uwaterloo.ca/~r5olivei/courses/2020-fall-cs466/lecture18-hardness-approximation-post.pdf
- Scheideler, C. (2005) *Lecture 9- Approximation and Complexity*. https://www.cs.jhu.edu/~scheideler/courses/600.471_S05/lecture_9.pdf
- Warnow,T. (2005) *Approximation Algorithms (continued).* http://tandy.cs.illinois.edu/dartmouth-cs-approx.pdf
- Vazirani, V. (2001) *Approximation algorithms*. Springer.

I stole the different images from:

- The cool PCP cartoon: https://www.zkcamp.xyz/blog/information-theory
- City map: https://www.istockphoto.com/fr/vectoriel/city-voir-le-plan-gm1095330908-294013033?searchscope=image%2Cfilm
- Molecular docking: https://condrug.com/urun/molecular-docking/
- The VC approx alg: https://www.javatpoint.com/daa-approximation-algorithm-vertex-cover

The idea of molecular docking as clique:

Kuhl, F.S., Crippen, G.M. and Friesen, D.K. (1984), *A combinatorial algorithm for calculating ligand binding*. J. Comput. Chem., 5: 24-34. https://doi.org/10.1002/jcc.540050105

Extras

Most common approximation classes

- $\alpha = O(n^c) \leftarrow$ Clique
- $\alpha = O(\log n) \leftarrow$ Set cover
- $\alpha = O(1) \leftarrow$ Vertex Cover