Greedy algorithms on graphs

Elia Noris
norisel@student.ethz.ch

S.Davis, R.Impagliazzo
“Models of greedy Algorithms for Graph Problems”

A.Borodin, J.Boyar, K.S.Larsen
“Priority Algorithms for Graph Optimization Problems ”

Motivation

- Greedy algorithm are widely used in almost every kind of problem
- They are quite often the simplest way to solve a problem
- Until now there have been very few tries to analyse them as a class of algorithm
- The definition itself is quite unprecise

What is greed?

Oliver Stone's Wall Street (1987)

The point is, ladies and gentleman, is that greed - for lack of a better word - is good!

Greed is right!

Greed works!

"The point is you can't be too greedy"

Donald J. Trump

Greed /gri:d/

noun [U]

a very strong wish to continually get more of something, especially food or money

(from Cambridge Advance Learner's Dictionary)

The term “greedy algorithm” is didactical, elegant and intuitively understandable, unluckily it lacks the precision needed for a mathematical analysis.

The priority model

Fixed priority algorithm

Determine an allowable ordering of the set of possible input items (without knowing the actual input set S of items)

While S is not empty

> $next :=$ index of input Item $I \in S$ that comes first in the ordering
>
> Make an irrevocable decision concerning I_{next} and remove it from S

Adaptive priority algorithm

While S is not empty

Determine a total ordering of all possible input items (without knowing the input items in S not yet considered)

$next :=$ index of input Item $I \in S$ that comes first in the ordering

Make an irrevocable decision concerning I_{next} and remove it from S

Examples - The Dominating Set Problem

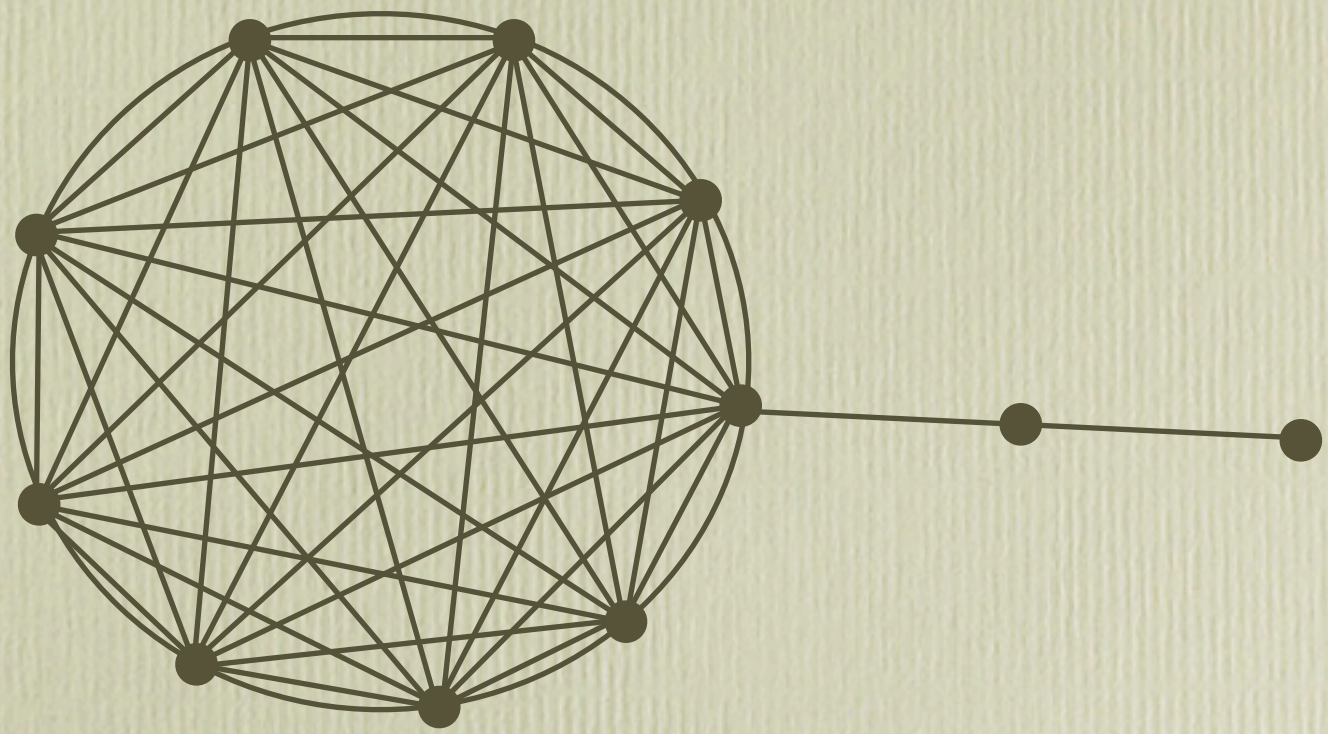
Fixed priority algorithm:

Order the vertices by the number of neighbours.
Add the vertices to the DS until the graph is dominated.

Adaptive priority algorithm

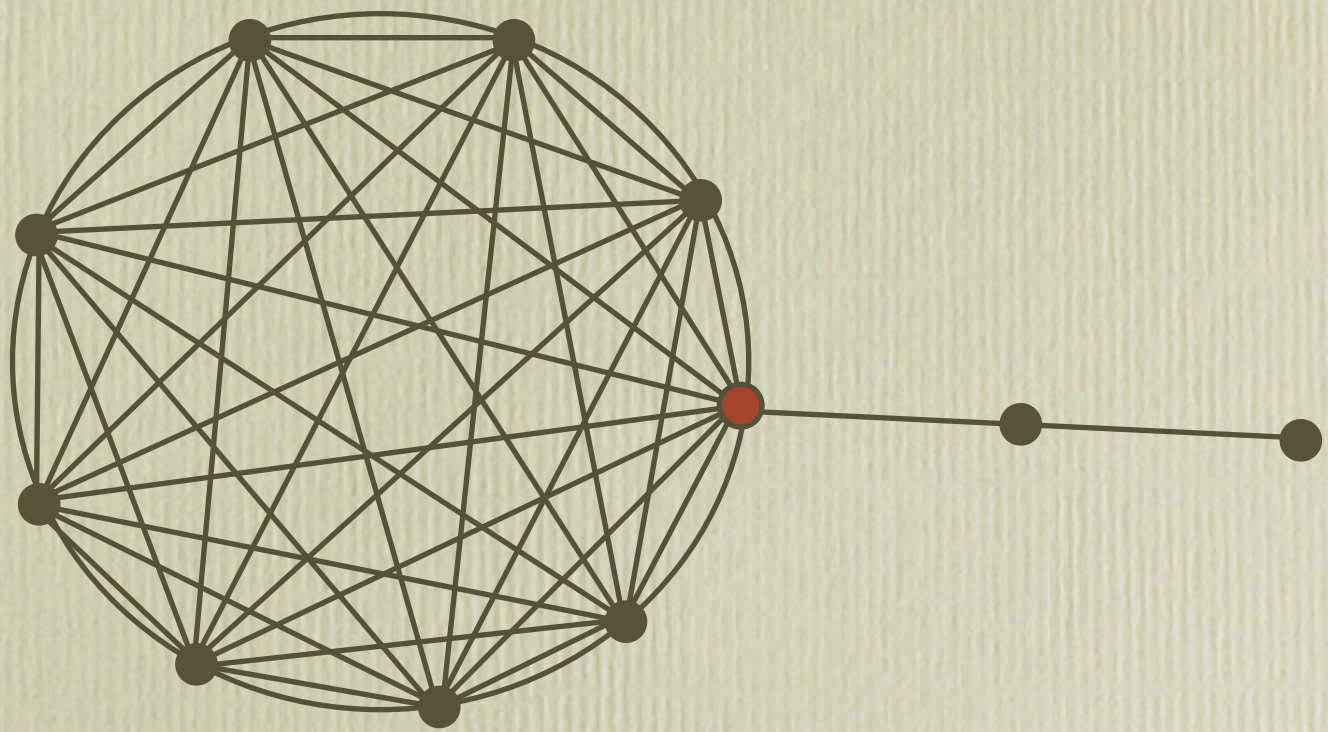
The next vertex will be the one with most not yet covered neighbours.
Vertices are added to the DS until the graph is dominated.

Examples - The Dominating Set Problem



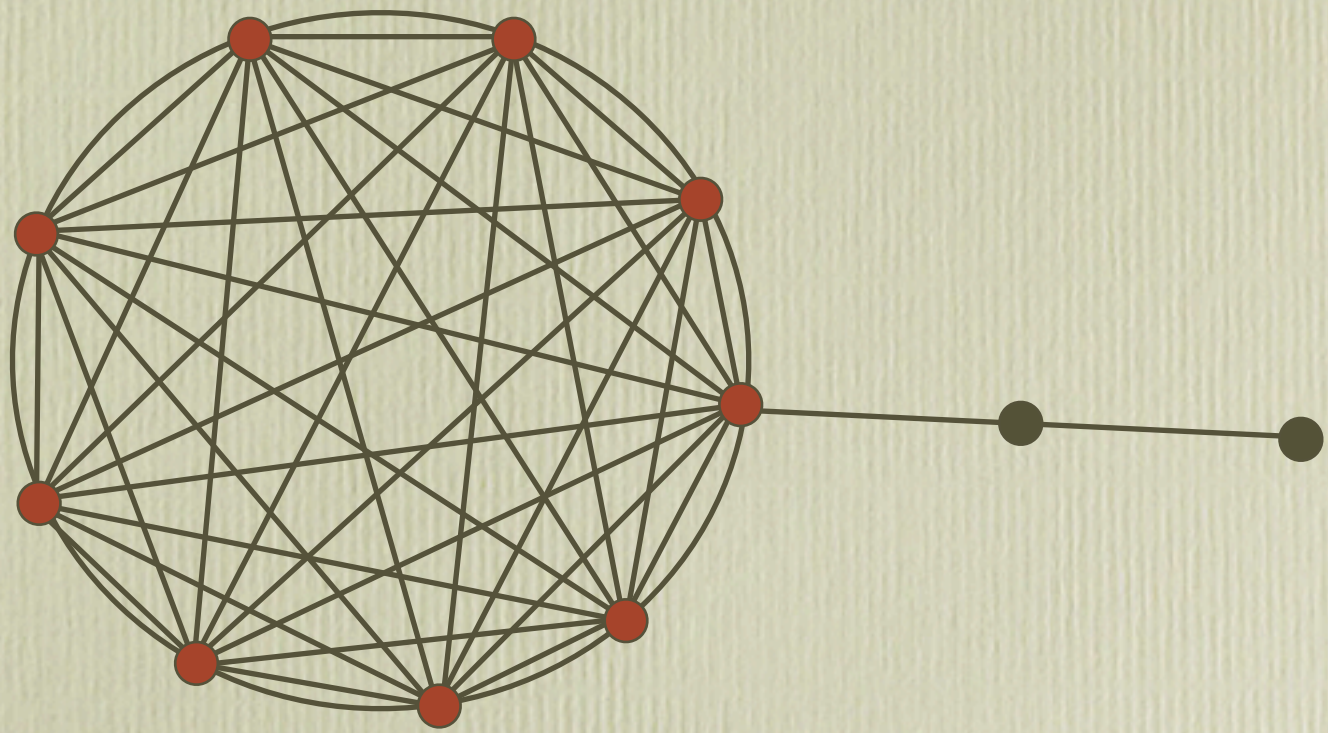
Examples - The Dominating Set Problem

Fixed priority algorithm



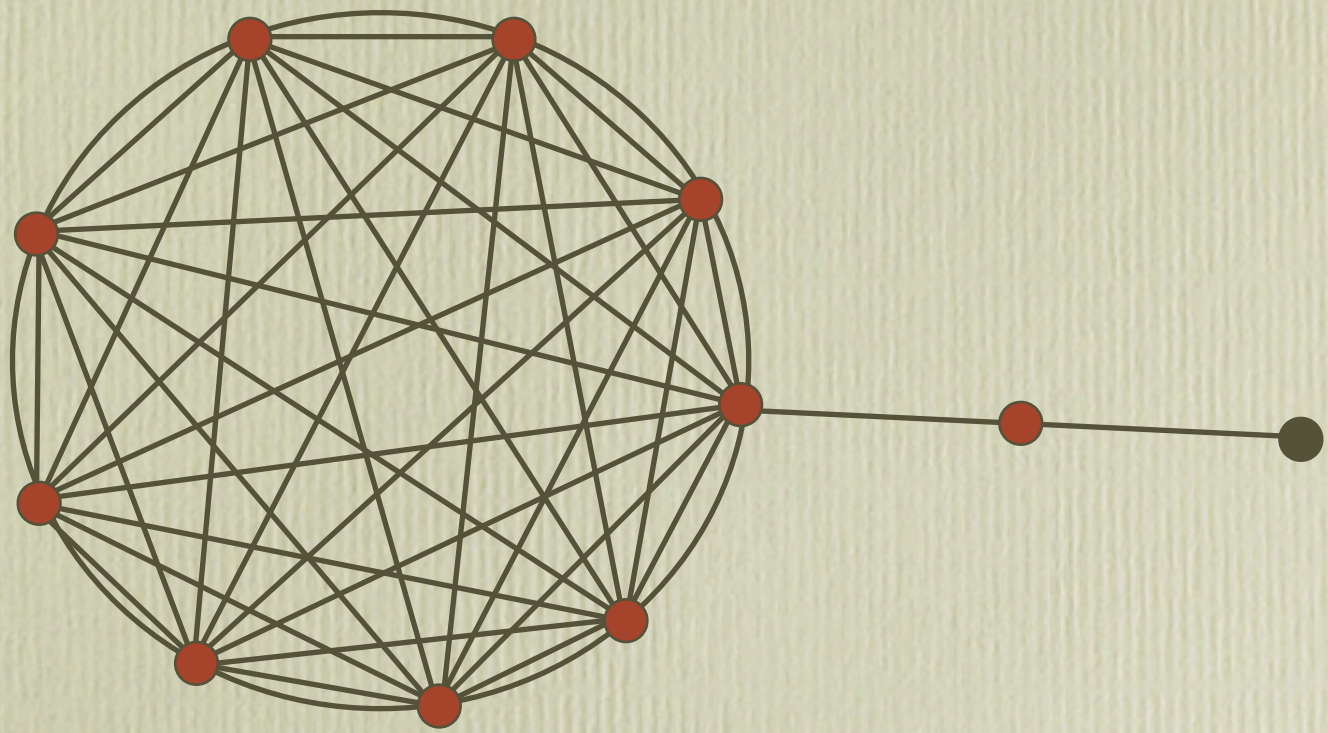
Examples - The Dominating Set Problem

Fixed priority algorithm



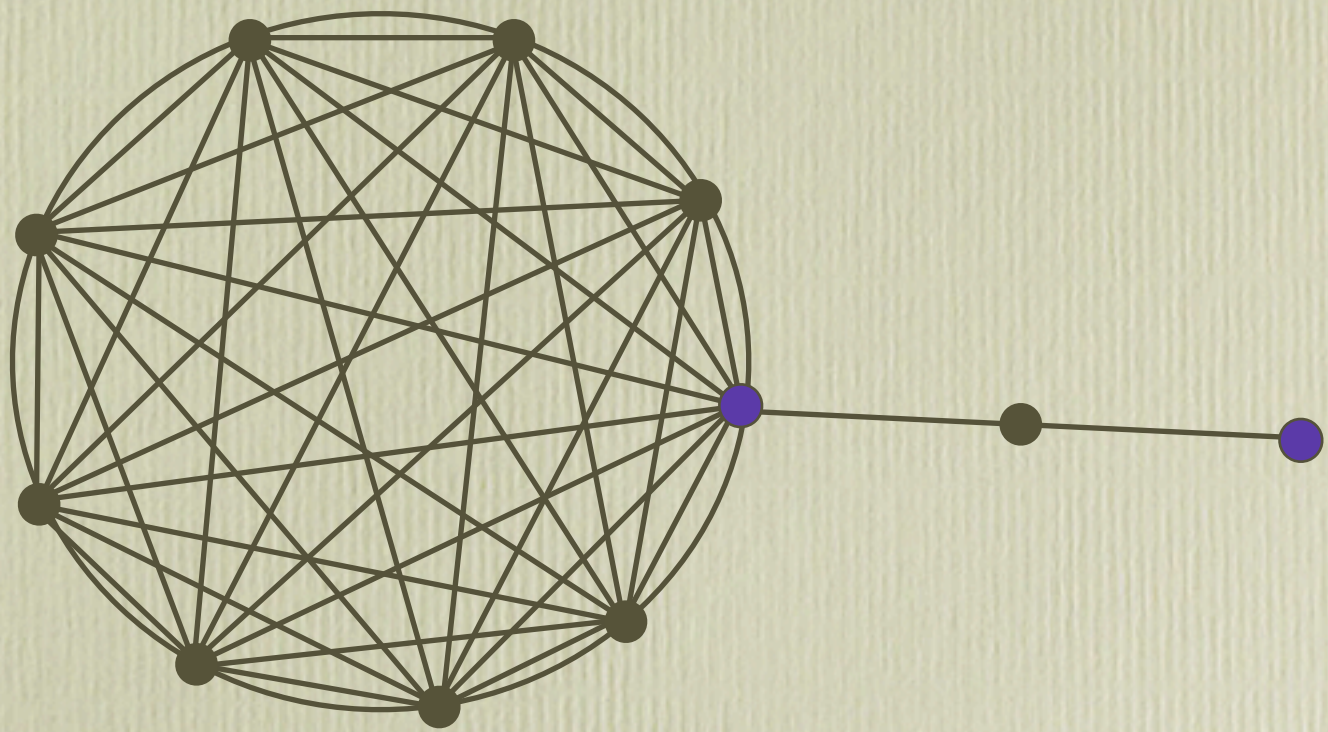
Examples - The Dominating Set Problem

Fixed priority algorithm



Examples - The Dominating Set Problem

Adaptive priority algorithm



A general lower bound technique

Interaction between two entities *Solver* and *Adversary*, Solver tries to solve the problem applying the algorithm, Adversary tries to give Solver the worst possible instance of the problem.

Adversary must be able to provide a solution whose cost/output is used to compute the approximation ratio of Solver's solution.

This is analogous to the competitive analysis of online algorithms.

Approximation ratio

An algorithm is said to have an approximation ratio of ρ if the expected cost C of the solution is within a factor of ρ of the cost C^* of an optimal solution, i.e. if

$$\frac{C}{C*} \leq \rho$$

holds for every instance of the problem.

Shortest Path

Given a directed Graph $G(V, E)$ and two nodes $s, t \in V$ find a directed tree of edges, rooted at s and with t as a leaf. The objective is to minimize the combined weight of the edges on the path from s to t.

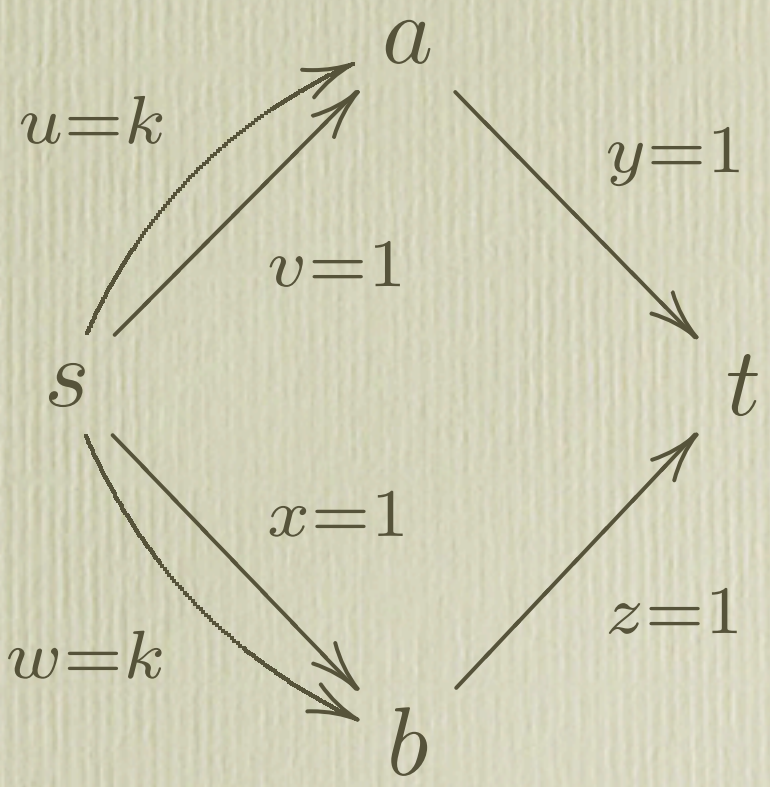
Theorem

No fixed priority algorithm can solve the shortest path problem with any approximation ratio ρ

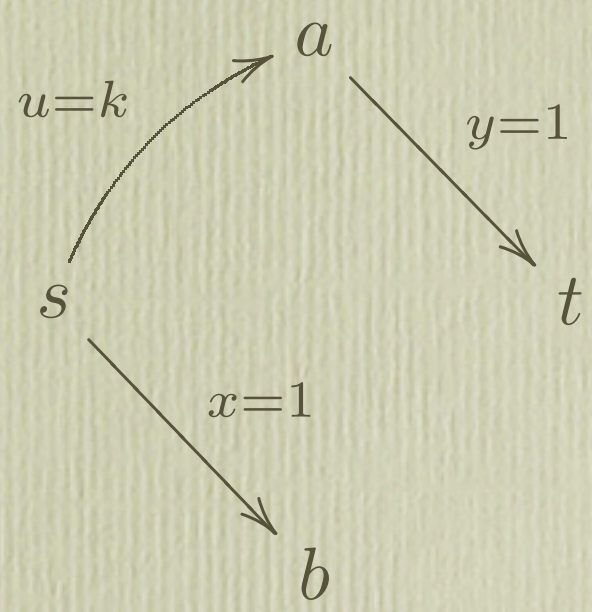
Proof idea:

$k \geq 2\rho$

$y \prec z$

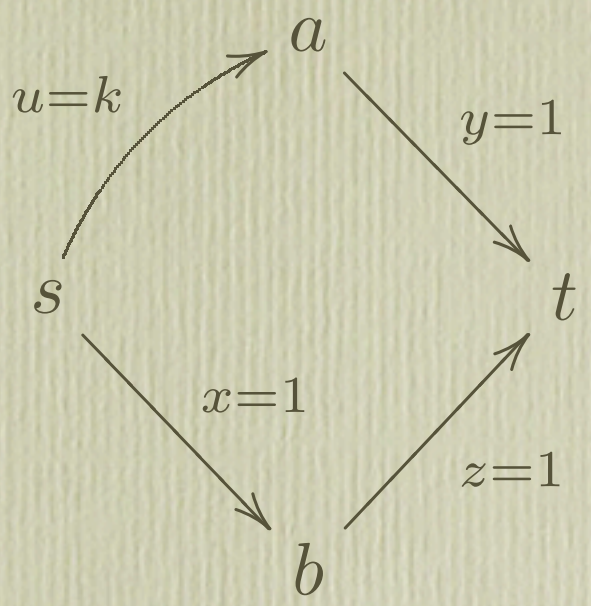


1. If Solver rejects y then Adversary remove z and builds the following instance:



Solver can no longer find a solution whereas Adversary propose $S=\{u,y\}$ and wins.

2. If Solver accepts y then Adversary builds the following instance:



Adversary propose *S={x,z}* with cost 2, Solver can't propose *{x,z,y}* because it wouldn't be a directed tree rooted at *s*, hence its solution must contain u and therefore it will cost at least *k+1*.

approximation ratio = $\frac{k+1}{2} > \frac{k}{2} = \rho$

QED

As Dijkstra's Algorithm can solve the Shortest Path Problem exactly and it belongs to the class of ADAPTIVE priority algorithms, we can conclude that the classes of algorithms FIXED and ADAPTIVE are not equivalent.

Weighted Vertex Cover

Given a directed Graph $G(V, E)$ in which every vertex $v \in V$ has an associated positive weight $w(v)$ find a vertex cover (a subset of V whose nodes touch every edge of the graph) of minimum weight.

The weight of a vertex cover V' is defined as:

$$w(V') := \sum_{v \in V'} w(v)$$

It has been shown that is not possible to approximate the weighted vertex cover with an approximation ratio better than

$$10\sqrt{5} - 21 = 1.3606$$

Unless *P=NP*

The best known (non priority) algorithm approximates the weighted vertex cover with an approximation ratio of

$$2 - \theta(\frac{1}{\sqrt{\log n}})$$

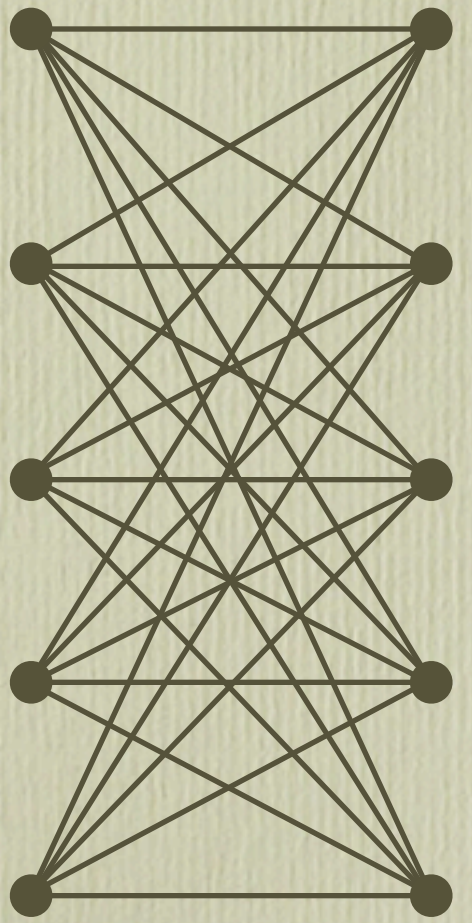
There are quite simple adaptive priority algorithm which solves the weighted vertex cover problem with an approximation ratio of 2.

There's for instance Clarkson's Algorithm that at any iteration picks the node with the minimum weight on number of not yet covered edges ratio.

Theorem

No adaptive priority algorithm can achieve an approximation ratio better than 2 for the Weighted Vertex Cover Problem.

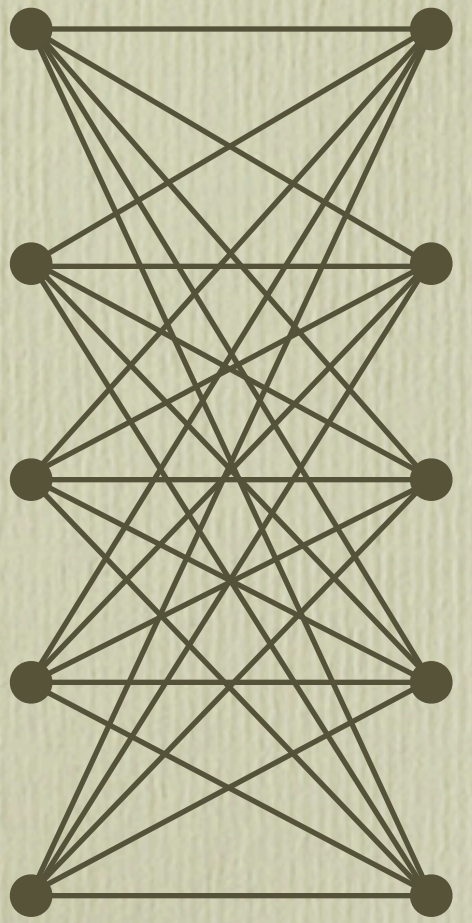
Proof idea:



$K_{n,n}$ *bipartite graph*

Nodes can have a weight of either 1 *or* n^2

One of the following events will eventually occur

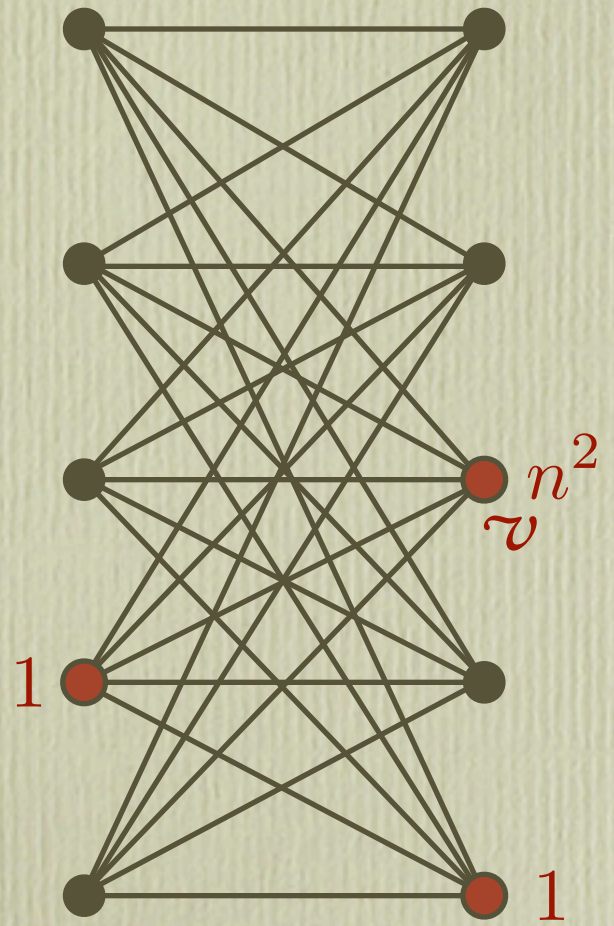


1. the solver accepts a node with weight n^2

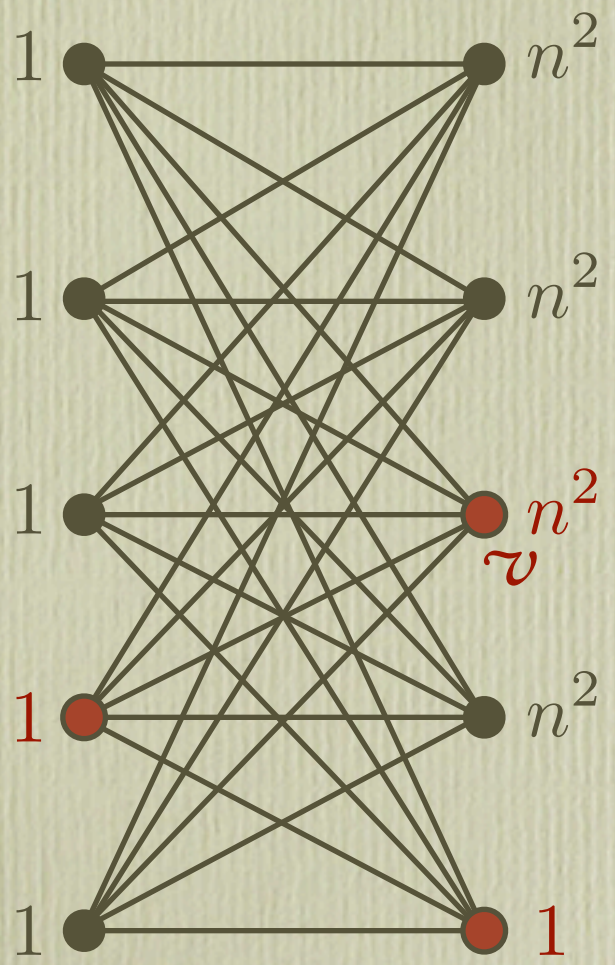
2. the solver accepts $n-1$ nodes of weight 1 from either sides of the bipartite graph

3. the solver rejects a node

case 1 the solver accepts a node v with weight n^2

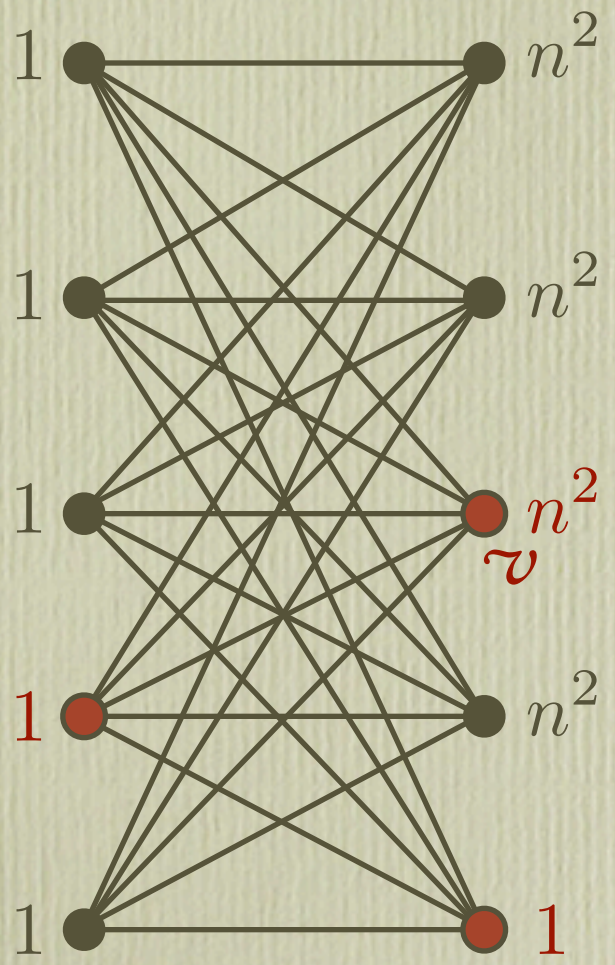


case 1 the solver accepts a node v with weight n^2



Adversary set all the node on the opposite side to 1 and the remaining node to n^2

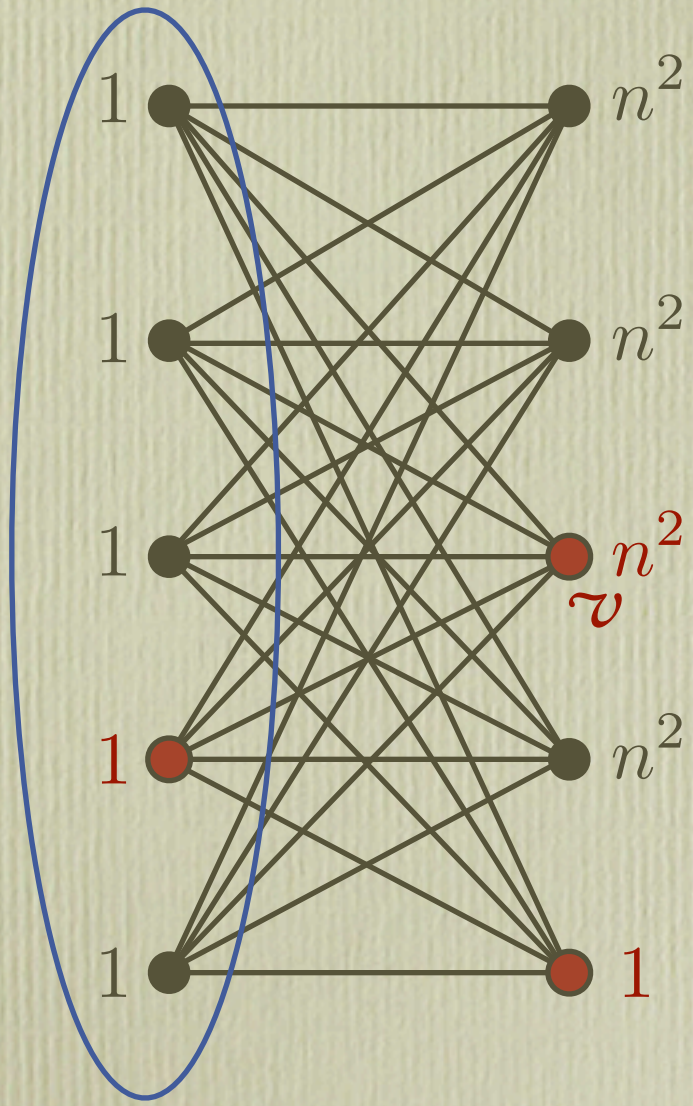
case 1 the solver accepts a node v with weight n^2



Adversary set all the node on the opposite side to 1 and the remaining node to n^2

Solver's solution contains v and therefore cost at least n^2

case 1 the solver accepts a node v with weight n^2

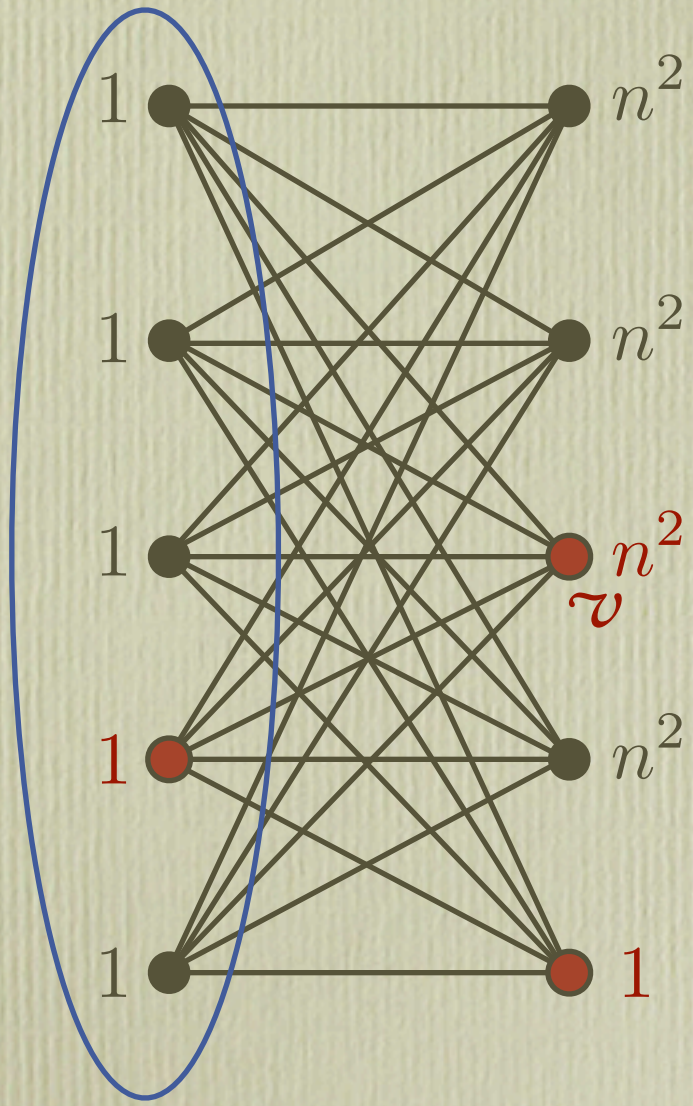


Adversary set all the node on the opposite side to 1 and the remaining node to n^2

Solver's solution contains v and therefore cost at least n^2

Adversary proposes a solution that costs n

case 1 the solver accepts a node v with weight n^2



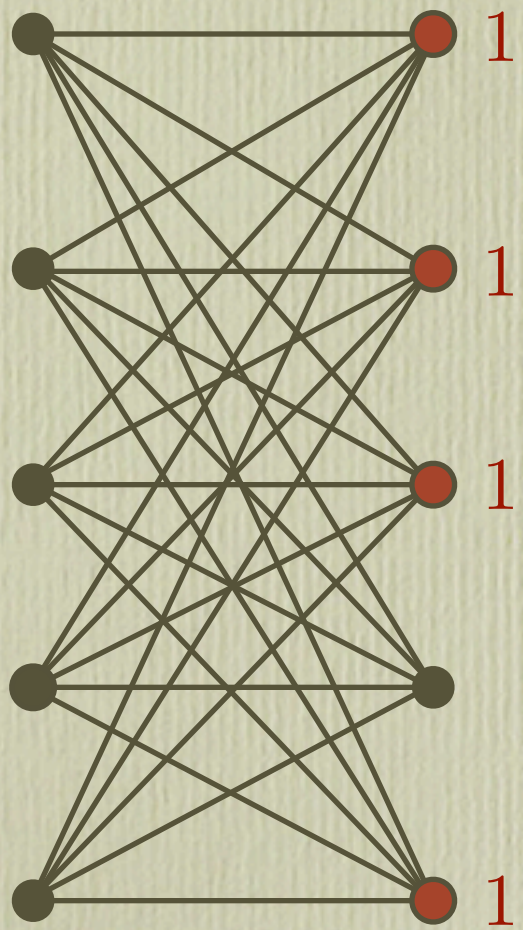
Adversary set all the node on the opposite side to 1 and the remaining node to n^2

Solver's solution contains v and therefore cost at least n^2

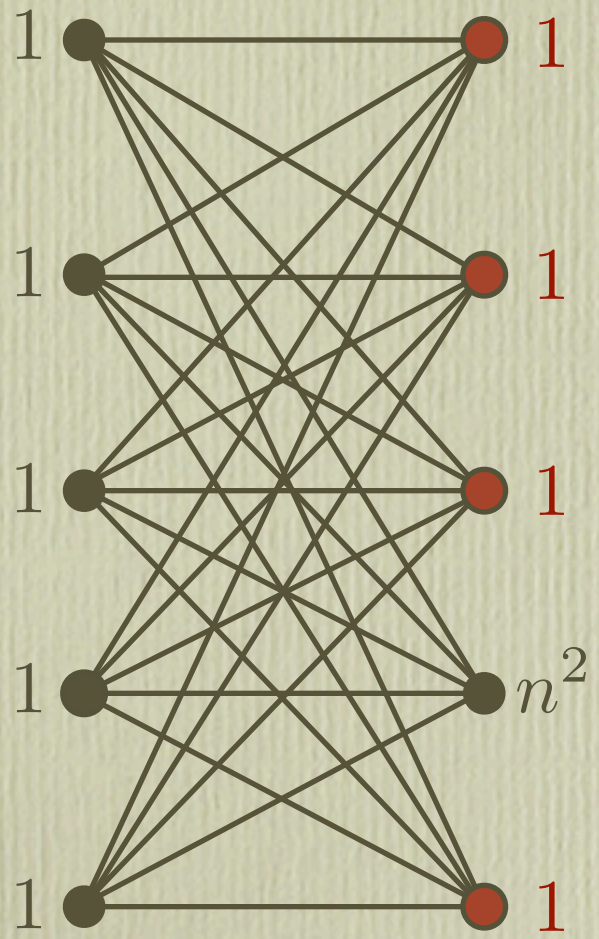
Adversary proposes a solution that costs n

If $\rho < n^2/n = n$ Adversary wins, with $\rho < 2$ this is always the case

case 2 the solver accepts $n-1$ nodes of weight 1 from either sides of the bipartite graph

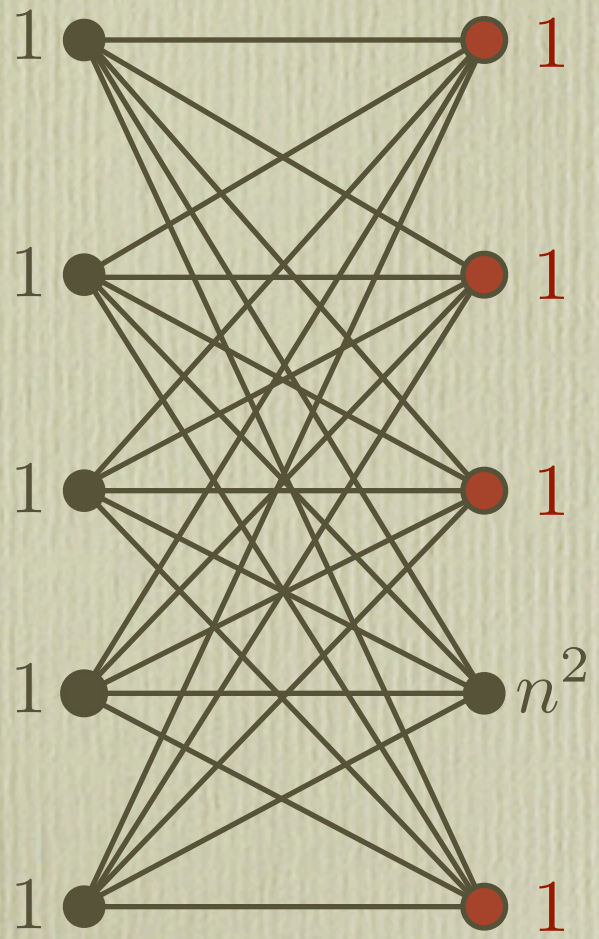


case 2 the solver accepts $n-1$ nodes of weight 1 from either sides of the bipartite graph



Adversary sets the last node on "Solver's" side to n^2 and all the other nodes to 1

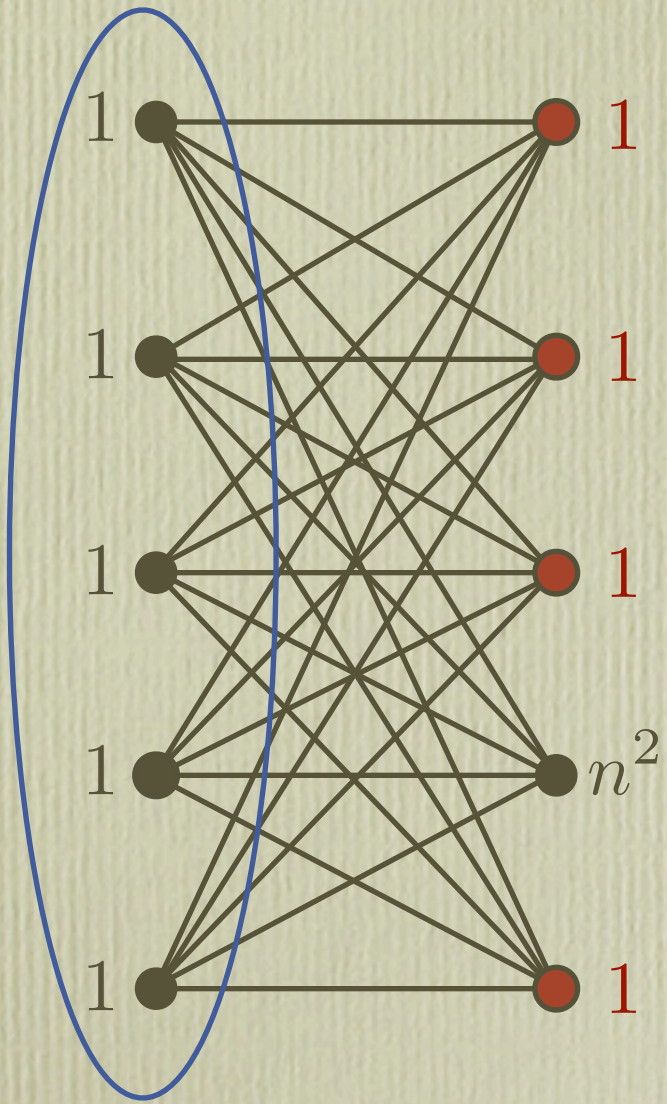
case 2 the solver accepts $n-1$ nodes of weight 1 from either sides of the bipartite graph



Adversary sets the last node on "Solver's" side to n^2 and all the other nodes to 1

Solver's solution either contains the "heavy" node or all the nodes on the other side, hence it will cost at least $2n-1$

case 2 the solver accepts $n-1$ nodes of weight 1 from either sides of the bipartite graph



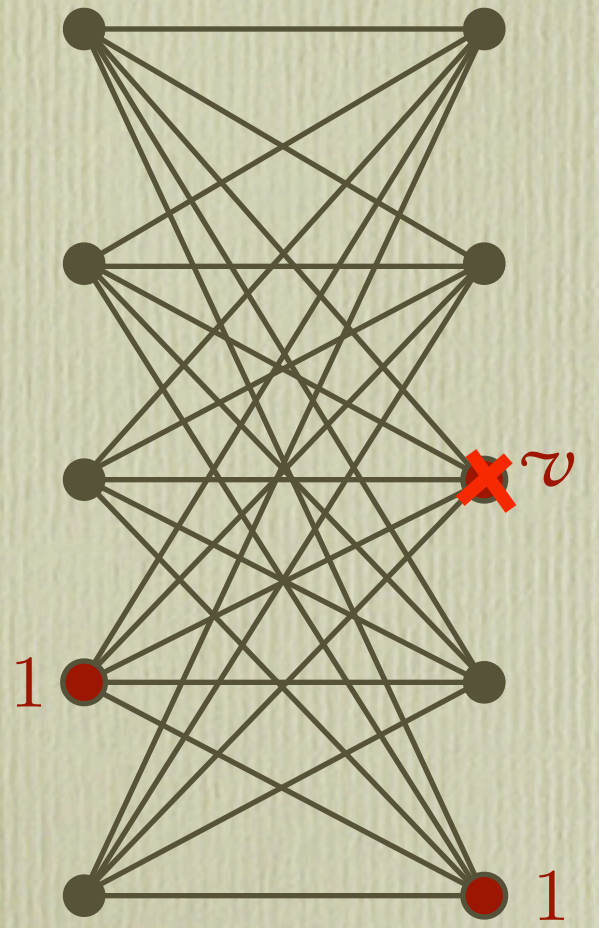
Adversary sets the last node on "Solver's" side to n^2 and all the other nodes to 1

Solver's solution either contains the "heavy" node or all the nodes on the other side, hence it will cost at least $2n-1$

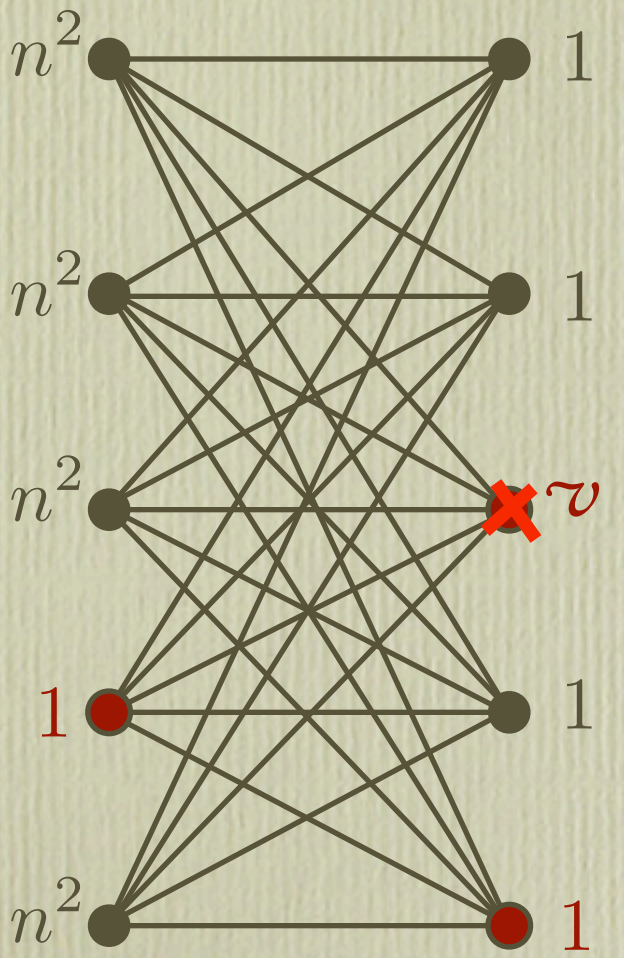
Adversary's solution costs n

If $\rho < \dfrac{2n-1}{n} = 2 - o(1)$
Adversary wins

case 3 the solver rejects a node v (of any weight)

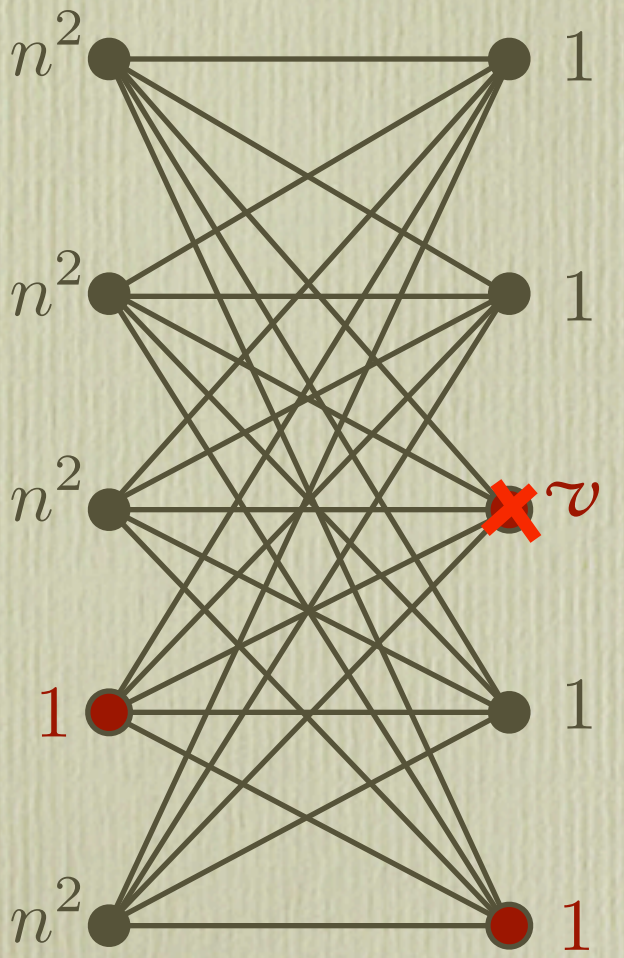


case 3 the solver rejects a node v (of any weight)



Adversary set all the unseen node on the opposite side of v to n^2 and the remaining node to 1

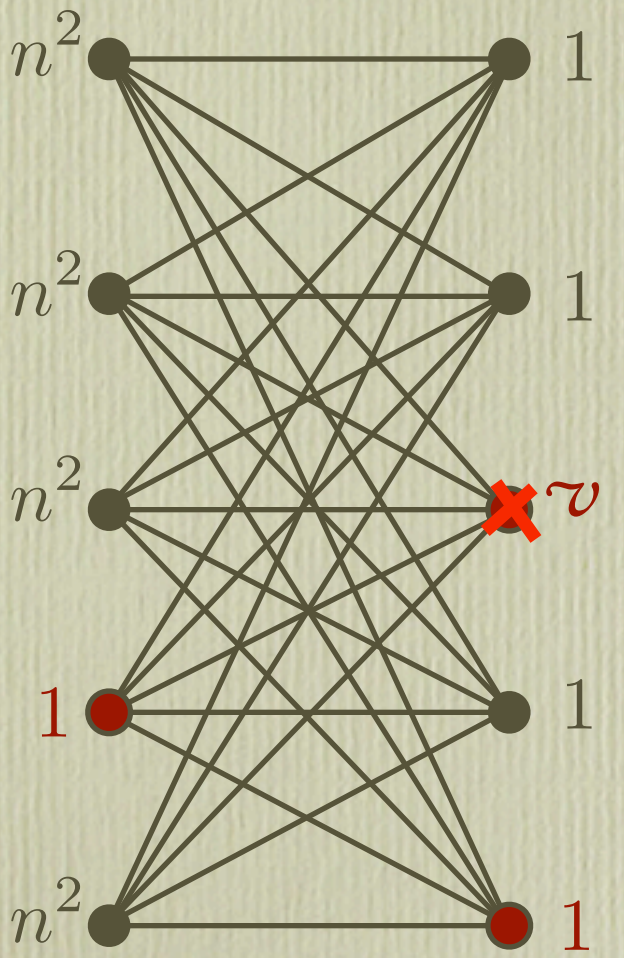
case 3 the solver rejects a node v (of any weight)



Adversary set all the unseen node on the opposite side of v to n^2 and the remaining node to 1

At least 2 nodes are set to n^2

case 3 the solver rejects a node v (of any weight)

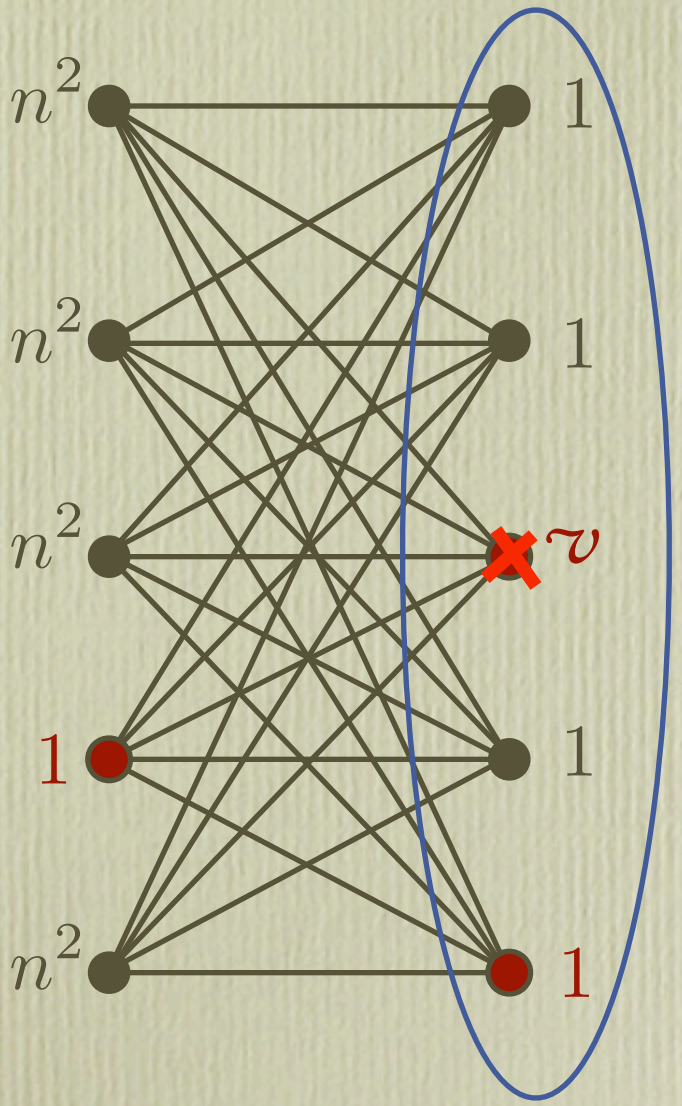


Adversary set all the unseen node on the opposite side of v to n^2 and the remaining node to 1

At least 2 nodes are set to n^2

Solver's solution doesn't contain v, hence it cost at least $2n^2$

case 3 the solver rejects a node v (of any weight)



Adversary set all the unseen node on the opposite side of v to n^2 and the remaining node to 1

At least 2 nodes are set to n^2

Solver's solution doesn't contain v, hence it cost at least $2n^2$

Adversary's solution contains v and costs at most $n^2 + n - 1$

Adversary wins iff $\rho < \dfrac{2n^2}{n^2 + n - 1} = 2 - o(1)$

QED

so far so good!

Acceptances-first algorithm

- Special kind of adaptive priority algorithm
- The decision is an *accept/reject* decision
- After the first rejection the algorithm can no longer accept any other item

Memoryless

- Special kind of adaptive priority algorithm
- The decision is an *accept/reject* decision
- Rejections have no influx on the further decision (i.e. rejections are seen as *no-ops*)

Memoryless algorithm can be simulated by *acceptances-first* algorithms

Node vs. edge model

- In the node model the Graph is represented by lists of adjacent vertices.
- In the edge model the Graph is represented by lists of adjacent edges.
- The two models should be equivalent as they represent the same thing
- unfortunately most results in the two papers require the problem to be formulated in a specified form

Conclusions

- This approach leads to results that hold for whole classes of algorithms including yet to be designed algorithms
- The distinction between edge model and node model is inelegant
- Not all the result are really significant
- The model is promising but still in the alpha version

dulcis in fundo

courtesy by our friend at MSN.COM

Say you want to go from Haugesund to Trondheim

Approximatively 600 km

It doesn't matter if you're looking for the shortest way...

or maybe for the fastest...

the point is....

WHERE DO YOU WANT TO GO TODAY?