CS4234
Optimiz(s)ation Algorithms

L2 – Approximation Algorithms and Linear Programming

https://visualgo.net/en/mvc

(both unweighted and weighted version)

NUS
National University of Singapore

Right Infringements on NUS Course Materials

All course participants (including permitted guest students) who have access to the course materials on Canvas/LumiNUS or any approved platforms by NUS for delivery of NUS modules are not allowed to re-distribute the contents in any forms to third parties without the explicit consent from the module instructors or authorized NUS officials

For CS4234: probably the PS (high-level) solutions, tutorial, and past midterm/final **answers** that are preferably not stored in public domain that will be 'easily indexed by Google'

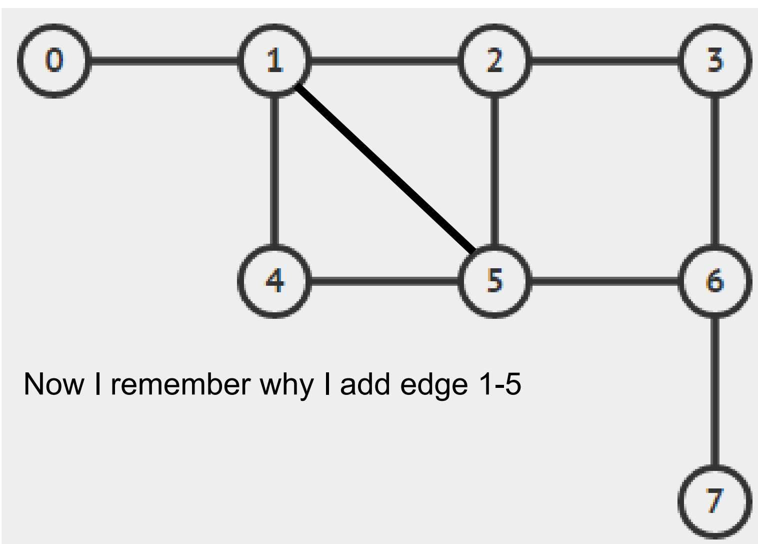
https://www.comp.nus.edu.sg/~stevenha/cs4234.html is public

Lecture 1 Recap (1)

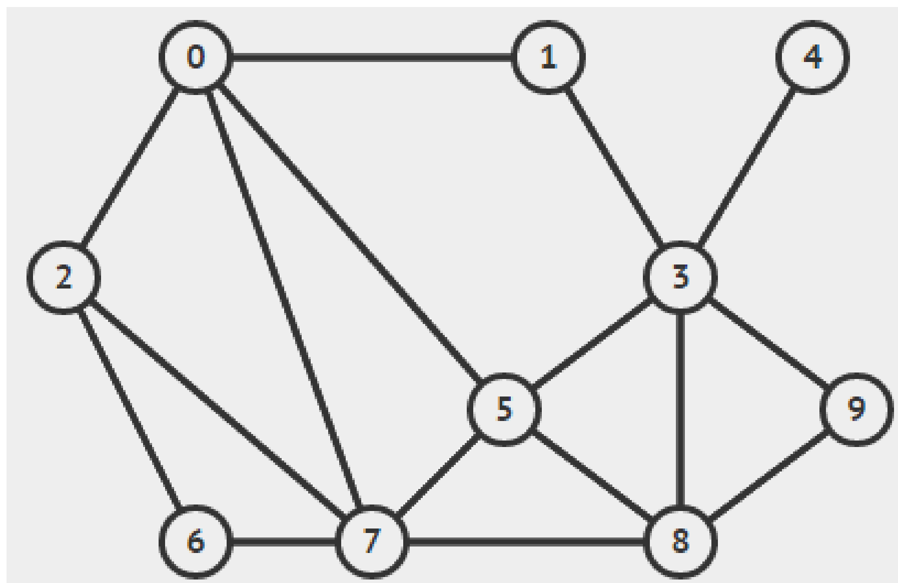
Definition of the first **Combinatorial Optimization Problem (COP)** in CS4234: `Min-Vertex-Cover`

- We revisit CS3230 to (re-)prove that VC is NP-Complete
- Thus MVC is NP-hard (as hard as VC) but not in NP

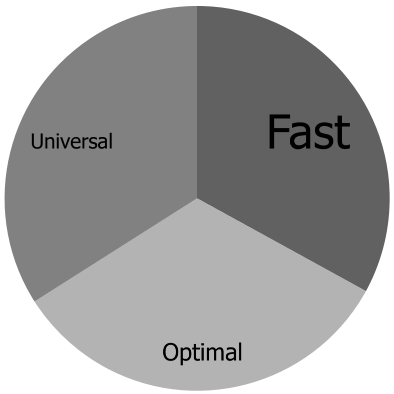
|MVC| = 4
example MVC = {1, 3, 5, 6}



|MVC| = 5 (Prof Halim will improve the bruteforce animation)
example MVC = {3, 0, 7, 8, 2}



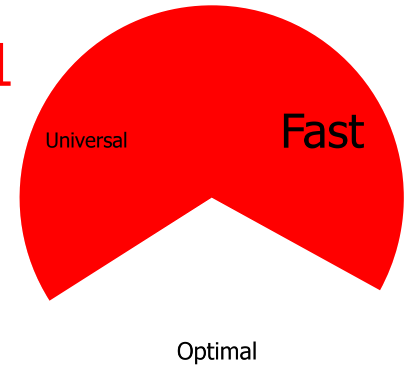
Lecture 1 Recap (2)



What should we do if we know (or we can reduce an known NP-complete problem into our current problem (the decision variant)

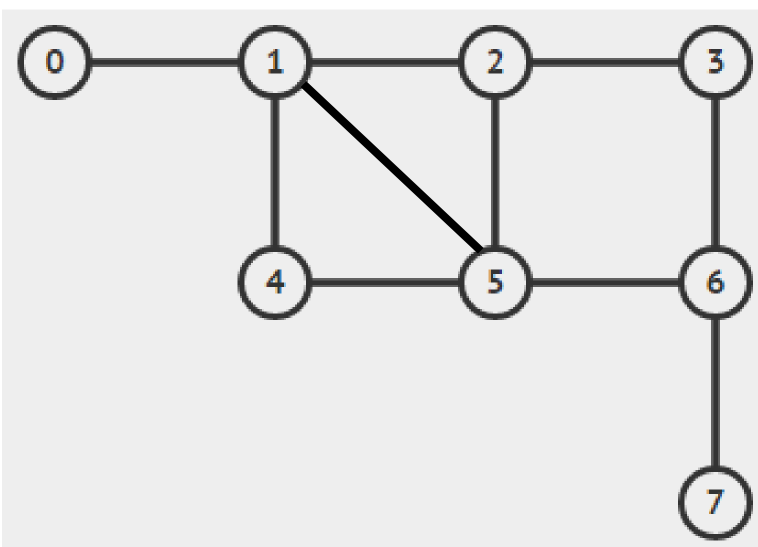
- Hope that it is posed on **special case** that somehow has polynomial solution(s) → lost **universality**
  - e.g., MVC on (Binary) Tree, we have DP solution (or? see T01)
- Hope that it is on **small** **instance/parameter** → still actually **not fast** (**exponential** time)
  - e.g., MVC on small k (or? see future questions)
- Hope that we can get by with **good enough** solution **fast** → good enough may be **non-optimal**
  - e.g., MVC, but we are OK if the answer is at most 2 x OPT
    - To be discussed now ☺

Approx Algo for MVC – Deterministic 1



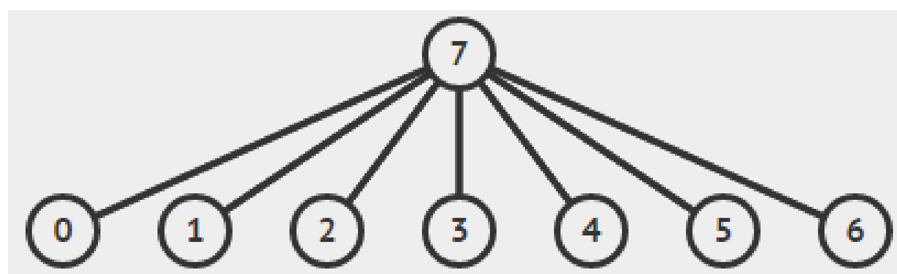

```
/* This algorithm adds vertices greedily, one at a time, until everything
   is covered.  The edges are considered in an arbitrary order, and for
   each edge, an arbitrary endpoint is added.                          */
Algorithm: ApproxVertexCover-1(G = (V, E))
Procedure:
    C ← ∅
    /* Repeat until every edge is covered:                             */
    while E ≠ ∅ do
        Let e = (u, v) be any edge in G.
        C ← C ∪ {u}
        G ← G_{-u} // Remove u and all adjacent edges from G.
    return C
```

Q: how to efficiently implement this?

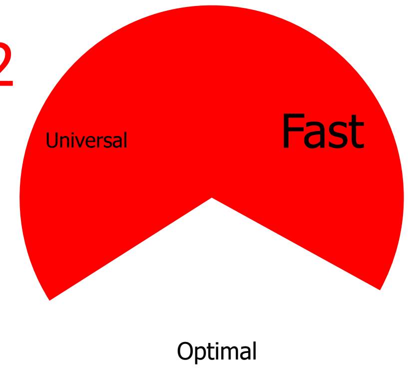


Can be very bad:

- **??**-approximation **(no bound)**



Approx Algo for MVC – Deterministic 2




```
/* This algorithm adds vertices greedily, two at a time, until everything
   is covered. The edges are considered in an arbitrary order, and for
   each edge, both endpoints are added.                                   */
```

Algorithm: ApproxVertexCover-2($G = (V, E)$)

Procedure:

$C \leftarrow \emptyset$

`/* Repeat until every edge is covered: */`

while $E \neq \emptyset$ **do**

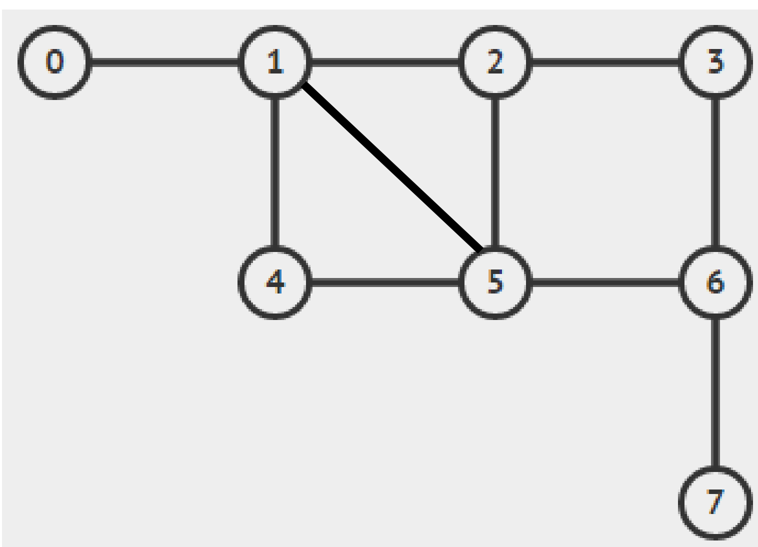
Let $e = (u, v)$ be any edge in G.

$C \leftarrow C \cup \{u, v\}$

$G \leftarrow G_{-\{u,v\}}$ `// Remove` u `and` v `and all adjacent edges from` G.

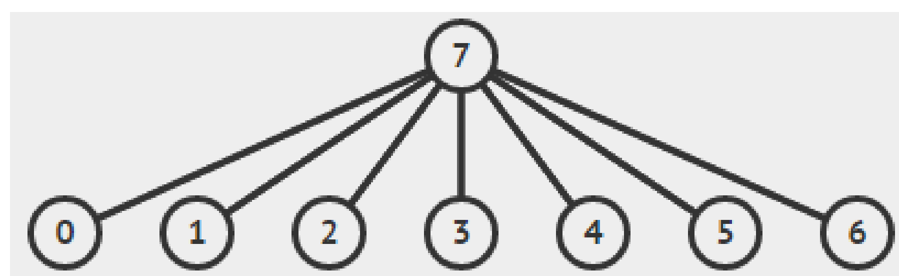
return C

Q: how to efficiently implement this?

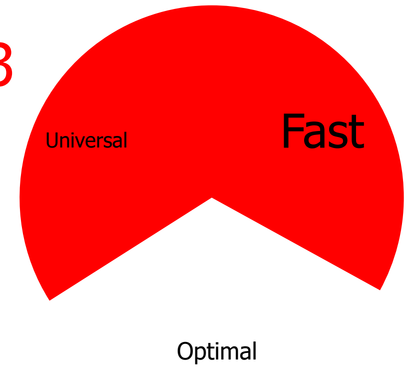


Not that bad?

– More analysis later



Approx Algo for MVC – Deterministic 3



/* This algorithm adds vertices greedily, one at a time, until everything is covered. At each step, the algorithm chooses the next vertex that will cover the most uncovered edges. */

Algorithm: ApproxVertexCover-3($G = (V, E)$)

Procedure:

$C \leftarrow \emptyset$

/* Repeat until every edge is covered: */

while $E \neq \emptyset$ **do**

Let $d(x)$ = number of uncovered edges adjacent to x.

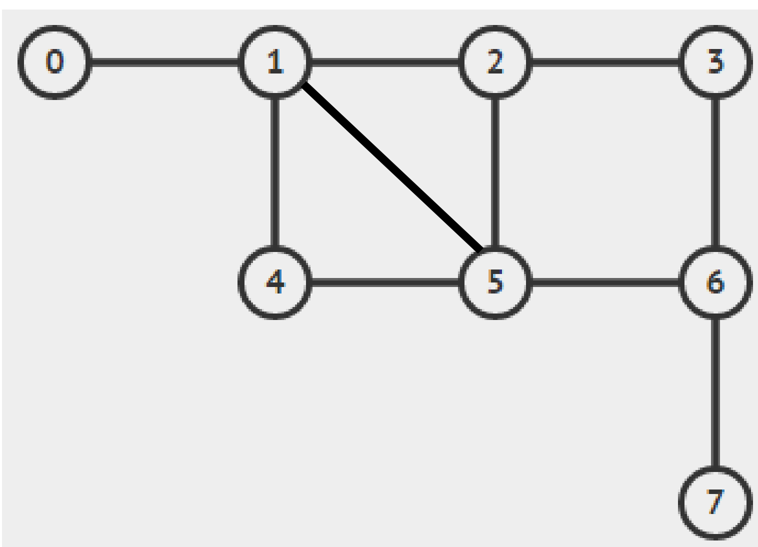
Let $u = \text{argmax}_{x \in V}\, d(x)$

$C \leftarrow C \cup \{u\}$

$G \leftarrow G_{-\{u\}}$ // Remove u and all adjacent edges from G.

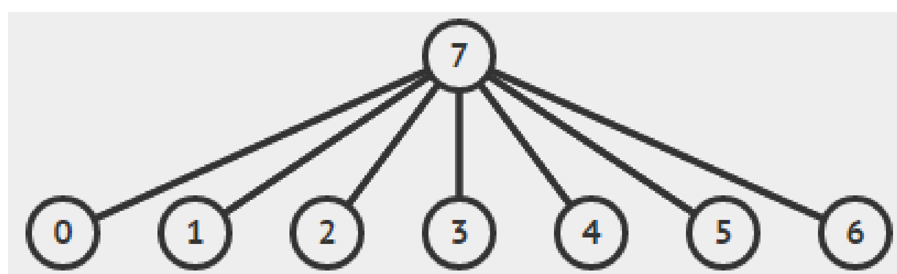
return C

Q: how to efficiently implement this?

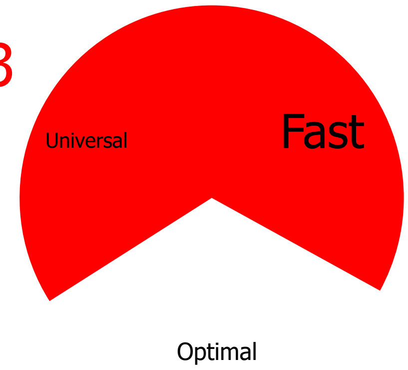


Not that bad?

– More analysis in the next slide



Approx Algo for MVC – Deterministic 3




```
/* This algorithm adds vertices greedily, one at a time, until everything
   is covered. At each step, the algorithm chooses the next vertex that
   will cover the most uncovered edges.                                  */
```

Algorithm: ApproxVertexCover-3($G = (V, E)$)

Procedure:

  $C \leftarrow \emptyset$

  `/* Repeat until every edge is covered: */`

  while $E \neq \emptyset$ **do**

    Let $d(x)$ = number of uncovered edges adjacent to x.

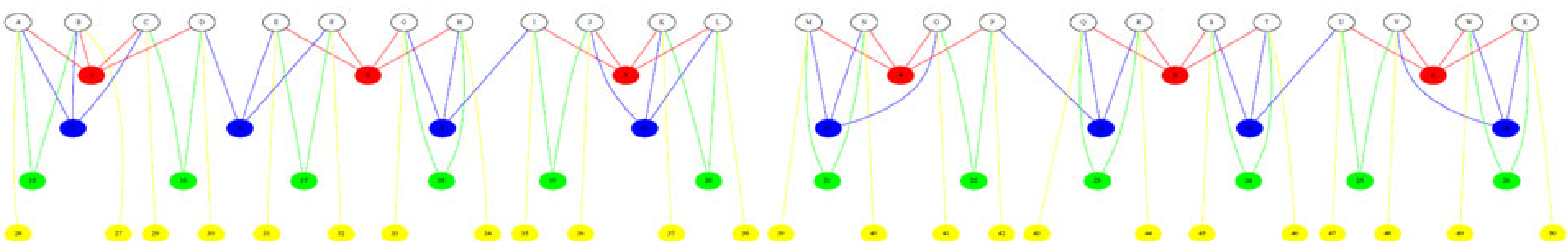
    Let $u = \mathrm{argmax}_{x \in V}\, d(x)$

    $C \leftarrow C \cup \{u\}$

    $G \leftarrow G_{-\{u\}}$ `// Remove` u `and all adjacent edges from` G.

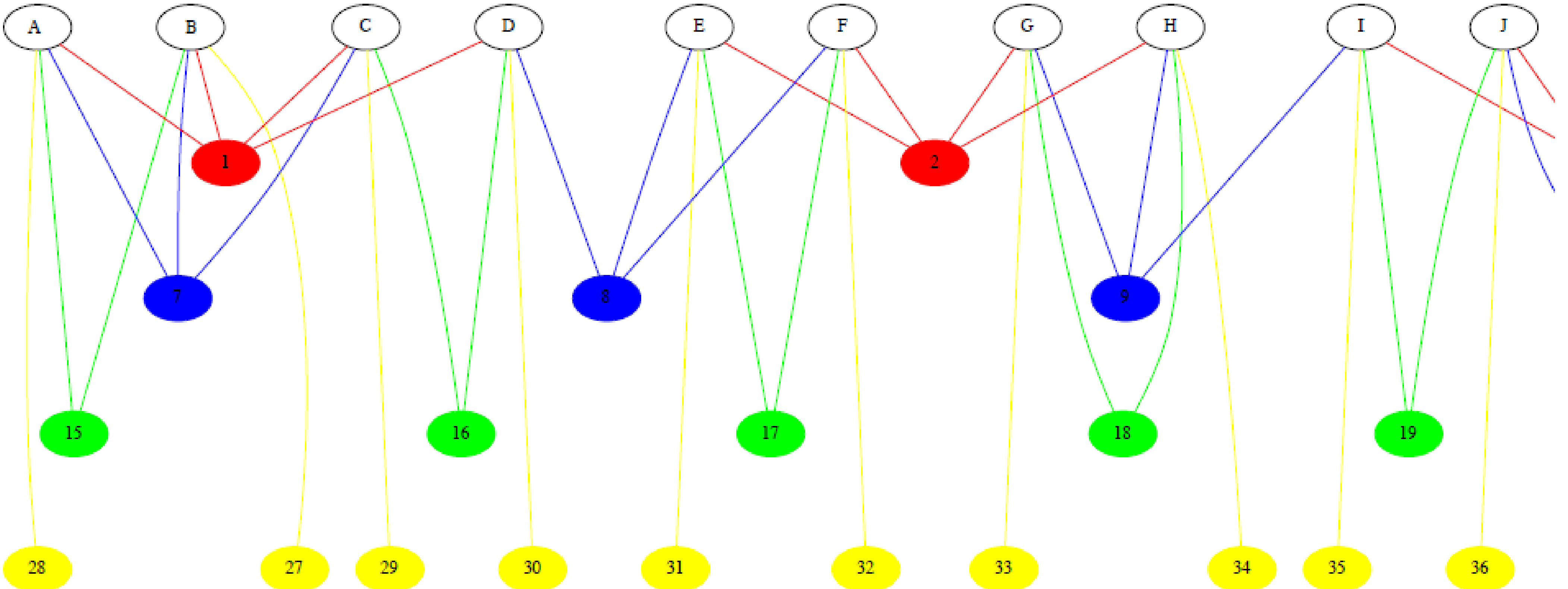
  return C

Below, **n** = 4, there are n! = 4! = 24 whites/yellows, 24/2 = 12 greens, 24/3 = 8 blues, and 24/4 = 6 reds



Zoomed-in

Optimal MVC = n! = 4! whites



The output of ApproxVertexCover-3 *could be*:

n!/n reds + n!/3 blues + n!/2 greens + n!/1 yellows =

n! * (1/n+1/3+1/2+1/1) = n! * (1/1+1/2+1/3+1/n) = n! * **ln n**

(Harmonic series)

This is ln-**n** (or log-**n**, base of log is negligible) factor worse than optimal answer

Approx Algo for MVC – Deterministic 2

/* This algorithm adds vertices greedily, two at a time, until everything is covered. The edges are considered in an arbitrary order, and for each edge, both endpoints are added. */

Algorithm: ApproxVertexCover-2$(G = (V, E))$

Procedure:

$C \leftarrow \emptyset$

/* Repeat until every edge is covered: */

while $E \neq \emptyset$ **do**

Let $e = (u, v)$ be any edge in G.

$C \leftarrow C \cup \{u, v\}$

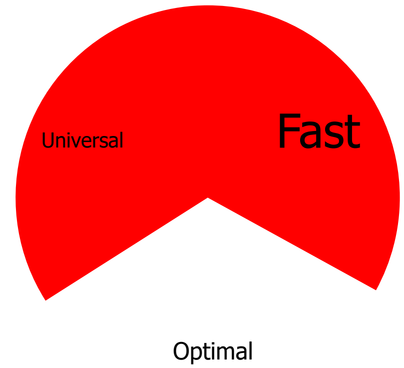
$G \leftarrow G_{-\{u,v\}}$ // Remove u and v and all adjacent edges from G.

return C

- This one has 2-approximation proof
  - Let **E'** be the edges considered by this algorithm, this **E'** is a **matching M** (revisited by Week 06 of CS4234, details in the PDF), **E' = M**
  - **|M|** ≤ **|OPT(G)|** (details in the PDF) and **C = 2*|E'|**
  - So **C = 2*|E'| = 2*|M|** ≤ **2*|OPT(G)|**

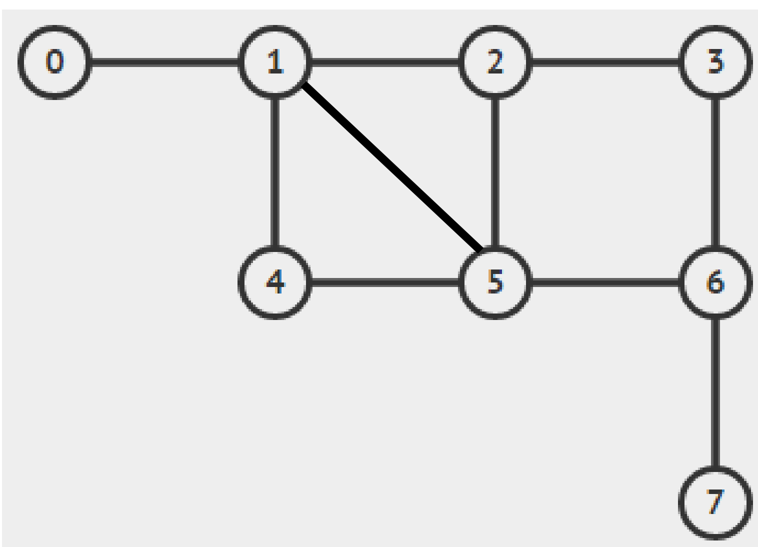
Try Deterministic 2-Opt live at https://visualgo.net/en/mvc

Approx Algo for MVC - Randomized




```
1  Algorithm: RandomizedVertexCover(G = (V, E))
2  Procedure:
3      C ← ∅
       /* Repeat until every edge is covered:                          */
4      while E ≠ ∅ do
5          Let e = (u, v) be any edge in G.
6          b ← Random(0, 1) // Returns {0, 1} each with probability 1/2.
7          if b = 0 then z = u
8          else if b = 1 then z = v
9          C ← C ∪ {z}
10         G ← G_{-z} // Remove z and all adjacent edges from G.
11     return C
```

Q: how to efficiently implement this?



2-approximation Analysis:

– Watered down proof:
- In line [5-8], the probability that we choose the right z is <u>at least ½</u>
- We include z in the cover and remove z (and 'cover' its edges) in line 9-10
- So C is at most 2 * OPT when E = ∅
  – In expectation *verbal discussion*

Try Probabilistic 2-Opt live at https://visualgo.net/en/mvc

In Practice…

- All approximation algorithms for MVC shown earlier run in polynomial time, i.e., **fast**
- No one is preventing you to run all of them and then report this ☺
  - [see the recording]

A high-level tour
Also in CP4 Book 2, p586-589

LINEAR PROGRAMMING

Linear Programming (LP)

A typical LP consists of three components:

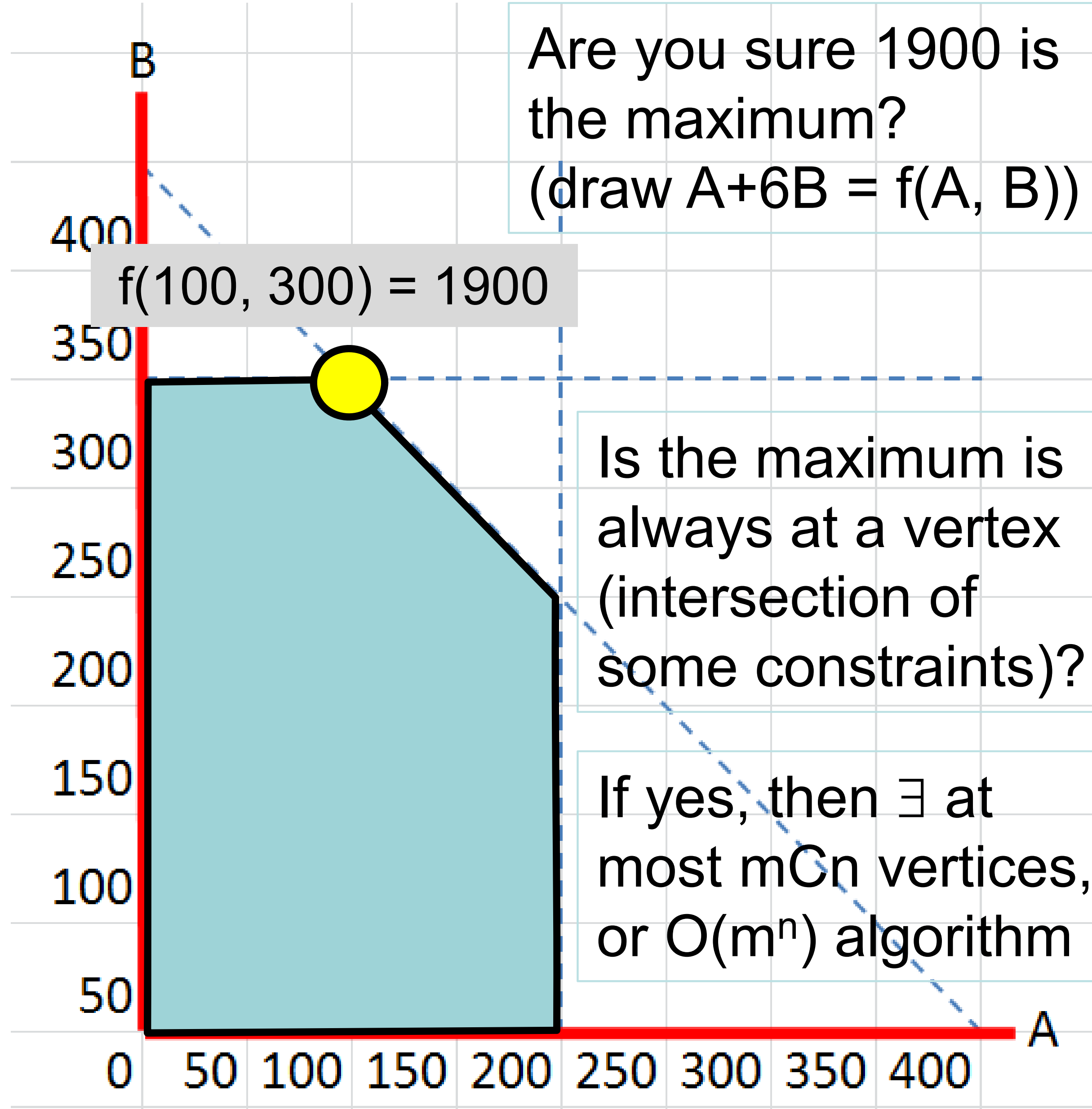
1. A list of (real-valued) variables $x_1, x_2, \ldots, x_n$
   - The goal: To find good values for these variables
2. An objective function $f(x_1, x_2, \ldots, x_n)$ that you are trying to maximize or minimize
   - The goal is to find the best values for the variables so as optimize this function
3. A set of m constraints that limits the feasible solution space
   - Each of these constraints is specified as an inequality

In a(n) LP problem, both the objective function and the constraints **are linear functions** of the variables

LP Example

A slightly different LP will be used in the "live" segment later

$$
\begin{aligned}
\max (A + 6B) \quad & \text{where:} \\
A &\leq 200 \\
B &\leq 300 \\
A + B &\leq 400 \\
A &\geq 0 \\
B &\geq 0
\end{aligned}
$$



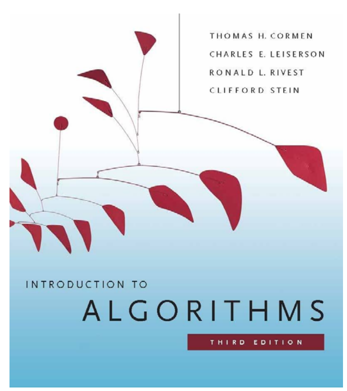
Are you sure 1900 is the maximum?
(draw A+6B = f(A, B))

Is the maximum is always at a vertex (intersection of some constraints)?

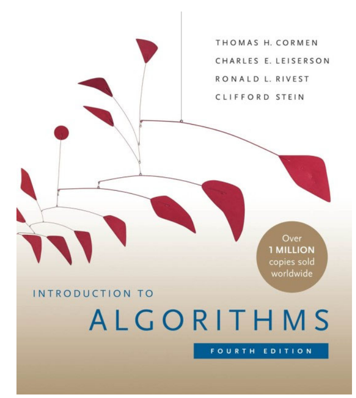
If yes, then ∃ at most mCn vertices, or $O(m^n)$ algorithm

Simplex Method

1. Find any (feasible) vertex v
2. Examine all the neighboring vertices of v: $v_1, v_2, \ldots, v_k$
3. Calculate $f(v), f(v_1), f(v_2), \ldots, f(v_k)$
   – If f(v) is the maximum (among its neighbors), then stop and return v
4. Otherwise, choose **<u>one of</u>** the neighboring vertices v_j where $f(v_j) > f(v)$
   – Let $v = v_j$
5. Go to step (2)

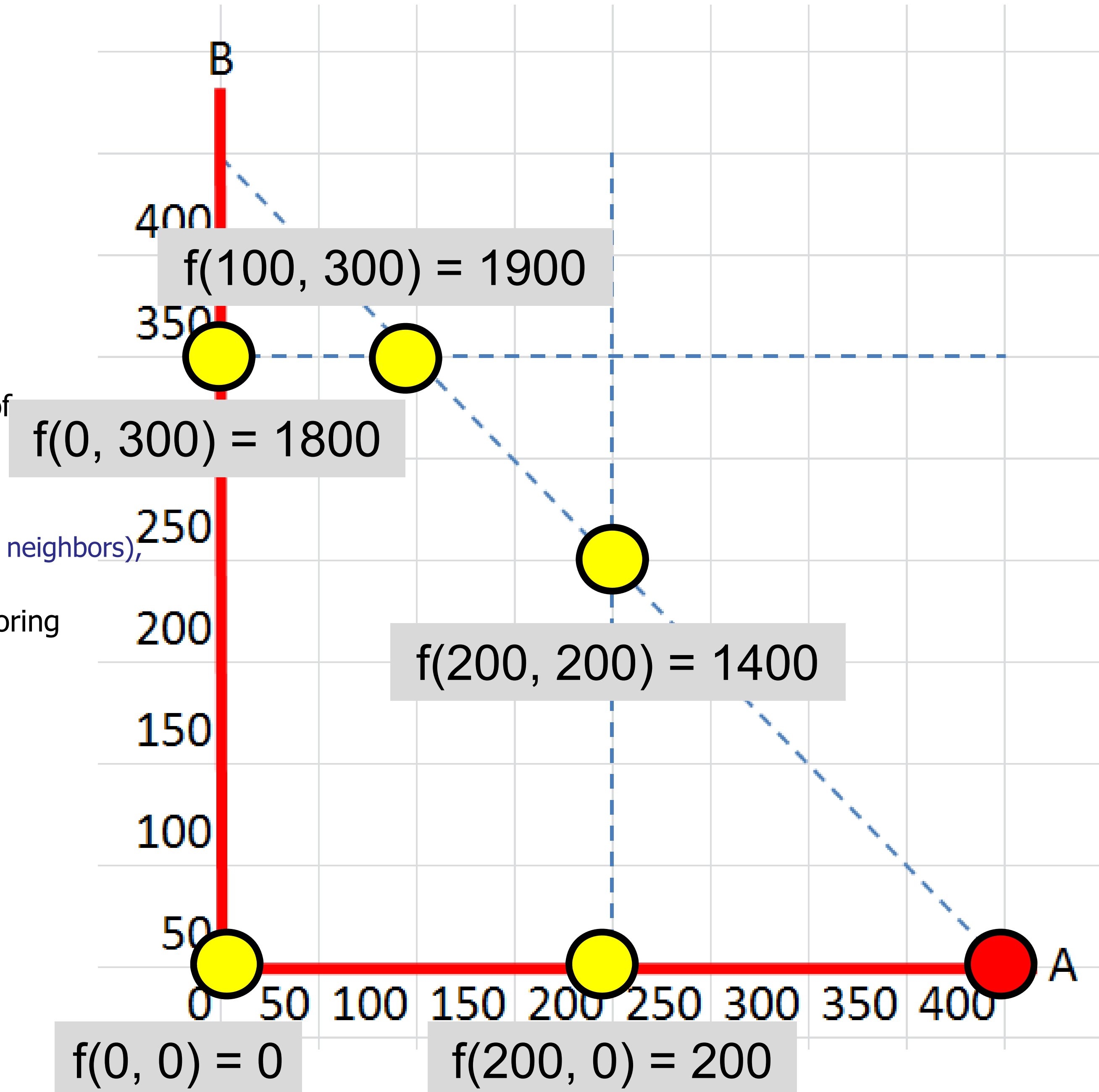


Details of Simplex is “dropped” in CLRS 4th edition…



Simplex Live Example

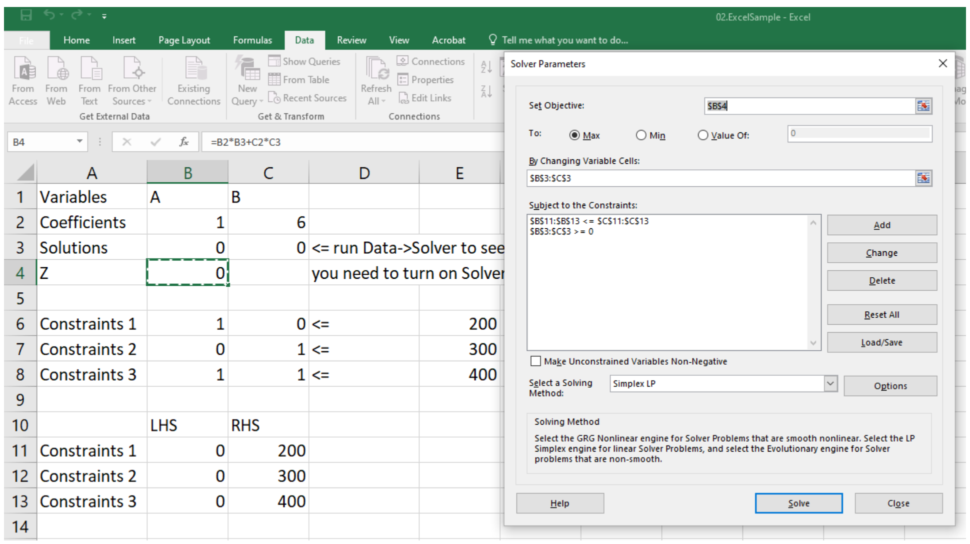
1. Find any (feasible) vertex v
2. Examine all the neighboring vertices of v $v_1, v_2, \ldots, v_k$
3. Calculate $f(v), f(v_1), f(v_2), \ldots, f(v_k)$
   - If f(v) is the maximum (among its neighbors), then stop and return v
4. Otherwise, choose **one of** the neighboring vertices v_j where $f(v_j) > f(v)$
   - Let $v = v_j$
5. Go to step (2)

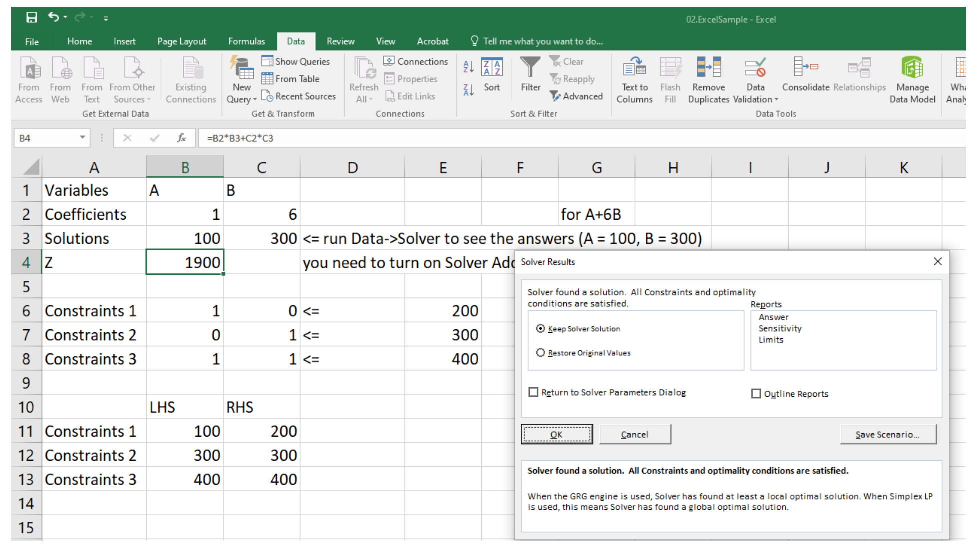


LP Solver in Microsoft Excel

Live Demonstration
(MS Excel can be setup quickly to use Solver Add-in)

See **02.ExcelSample.xlsx** (tab 'LP')





lp_solve in Ubuntu

Cannot do live demonstration on Steven's laptop…
it is on Windows ☹

If I cannot SSH to my DO droplet, just see this screenshot

```
[redacted]l-droplet:/# cat cs4234.lp
/* from CS4234 lecture note 02 */

max: A + 6 B;

A <= 200;
B <= 300;
A + B <= 400;
A >= 0;
B >= 0;
[redacted]l-droplet:/# lp_solve cs4234.lp

Value of objective function: 1900.00000000

Actual values of the variables:
A                              100
B                              300
```

Usable Simplex in C++

https://github.com/jaehyunp/stanfordacm/blob/master/code/Simplex.cc

Good usable Simplex code from Stanford ICPC team ☺

I have a local copy of that Simplex code in Java
(thanks to a senior student from "many" AYs ago)

Any volunteer to convert this to Python?

- Probably ChatGPT can do the translation too
- for non-Kattis projects, you may want to use https://docs.scipy.org/doc/scipy/reference/optimize.linprog-simplex.html

Now use any tool to solve this

- Maximize $777x+7y$
- Such that:
  - $100x \leq 15000$
  - $50y \leq 10000$
  - $x+y \leq 300$
- And x and y are non-negative

Answer for this year
[do it yourself!!]

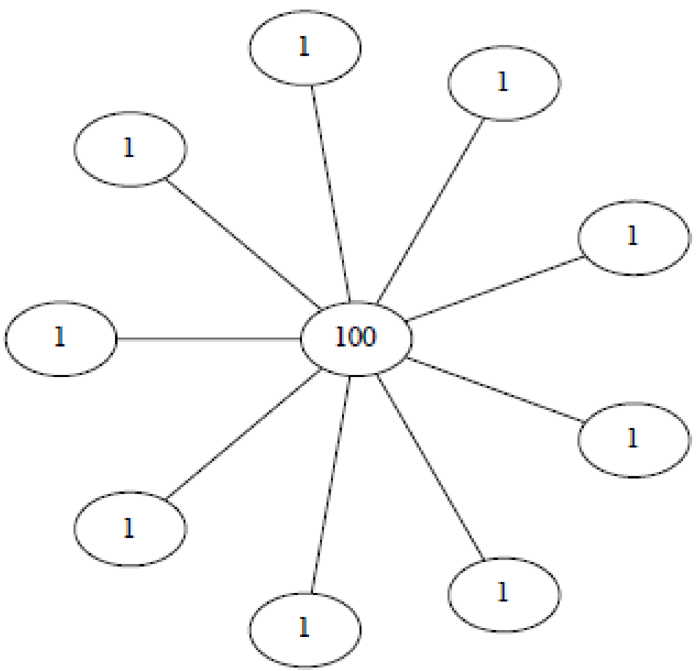
PS: Steven shall randomize the LP exercise each year

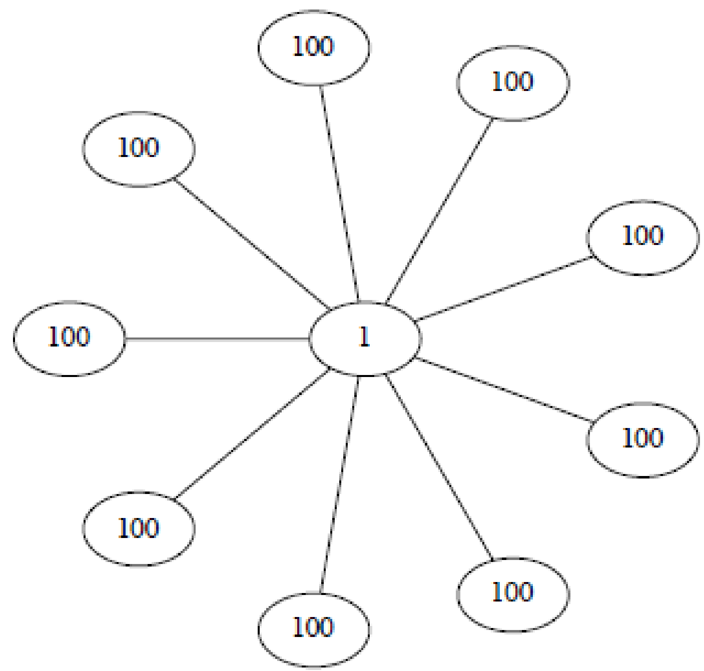
Two ways to ***approximately*** deal with this version…

MIN WEIGHT VERTEX COVER (MWVC)

MIN-WEIGHT-VERTEX-COVER

Both the Deterministic & Randomized 2-approximation algorithm for `Min-Vertex-Cover` "fail" on the weighted version; Do you understand why?





MWVC as an (Integer) Linear Program ILP/IP

The formulation (x_j is a Boolean {0, 1} variable where 0 = not in VC and 1 = in VC):

$$\min \left(\sum_{j=1}^{n} w(v_j) \cdot x_j \right) \quad \text{where:}$$

$$\begin{aligned} x_i + x_j &\geq 1 \quad \text{for all } (i,j) \in E \\ x_j &\geq 0 \quad \text{for all } j \in V \\ x_j &\leq 1 \quad \text{for all } j \in V \\ x_j &\in \mathbb{Z} \quad \text{for all } j \in V \end{aligned}$$

Notice that this line actually has E copies in an actual (I)LP program

Some other textbook says $x_j \in \{0, 1\}$, and there are V copies

The unweighted one
Proven NP-hard last week

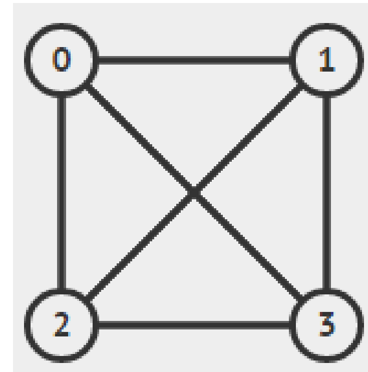
Min-Vertex-Cover (set $w(v_j)$ to all 1) $\leq_p$ ILP

So ILP is also NP-hard

PS: A tool for ILP that I have explored in the past

MWVC as a Relaxed Linear Program

Relaxing the Integer constraint



Assume w is all 1
Example LP solution
$x_0 = x_1 = x_2 = x_3 = 0.5$
What should we do?

$$\min\left(\sum_{j=1}^{n} w(v_j)\cdot x_j\right) \qquad \text{where:}$$

$$\begin{array}{rcll} x_i + x_j & \geq & 1 & \text{for all } (i,j) \in E \\ x_j & \geq & 0 & \text{for all } j \in V \\ x_j & \leq & 1 & \text{for all } j \in V \end{array}$$

[redacted]

PS: LP solution can be equal or better than ILP solution. Why?

Round up $\mathbf{x_j}$ value if it is ≥ 0.5

let $y_j = 1$ if $x_j \geq 1/2$, and let $y_j = 0$ otherwise.

But is this a good approximation?

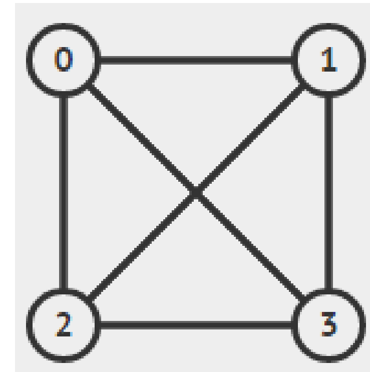
lp_solve output

```
[redacted] # cat k4.lp
min: x0+x1+x2+x3;

x0+x1 >= 1;
x0+x2 >= 1;
x0+x3 >= 1;
x1+x2 >= 1;
x1+x3 >= 1;
x2+x3 >= 1;

x0 >= 0;
x1 >= 0;
x2 >= 0;
x3 >= 0;

x0 <= 1;
x1 <= 1;
x2 <= 1;
x3 <= 1;
[redacted] lp_solve k4.lp

Value of objective function: 2.00000000

Actual values of the variables:
x0                              0.5
x1                              0.5
x2                              0.5
x3                              0.5
```

Analysis

OPT(G) = OPT(ILP) ≥ OPT(LP)

$$cost(OPT) \geq \sum_{j=1}^{n} w(v_j) \cdot x_j$$



Assume w is all 1

Example ILP solution $x_0 = x_1 = x_2 = 1;\ x_3 = 0$

Example LP solution $x_0 = x_1 = x_2 = x_3 = 0.5$

Rounded answer ≤ 2 x OPT(LP)

$$\begin{aligned} \sum_{j=1}^{n} w(v_j) \cdot \boxed{y_j} &\leq \sum_{j=1}^{n} w(v_j) \cdot \boxed{(2x_j)} \\ &\leq 2\left(\sum_{j=1}^{n} w(v_j) \cdot x_j\right) \\ &\leq 2 \times OPT(G) \end{aligned}$$

Notice, however, that $y_j \leq 2x_j$, for all j.

Analysis: This is a **2**-Approximation algorithm

Linear Programming Summary

General form of an LP:

1. A set of variables: $x_1, x_2, \ldots, x_n$
2. A linear objective to maximize (or minimize): $c^T x$
   - c and x as vectors,
   - c^T represents the transpose of c
   - multiplication represents the dot product
3. A set of linear constraints written as a Matrix equation: $Ax \leq b$

Presented as: $\mathbf{max}\ \mathbf{c^T x}$ **where** $\mathbf{Ax \leq b}$ **and** $\mathbf{x \geq 0}$

LP in Standard Form

Exercise to translate given LP into standard form

$$\begin{aligned} \min x_1 + 2x_2 - x_3 & \quad & \text{where // this is a minimization problem} \\ x_1 + x_2 & = & 7 \text{ // this is an equality} \\ x_2 - 2x_3 & \geq & 4 \text{ // this is a } \geq \text{ inequality} \\ x_1 & \leq & 2 \end{aligned}$$

Details in the PDF

Wait…

- There is an even simpler 2-Approximation algorithm for MWVC… which is…
  - See https://visualgo.net/en/mvc, M weighted VC

Summary

- Approximation algorithms for MVC
  - Deterministic, 3 variants, **but only variant 2 is 2-Approximation**
  - Randomized, ***Expected*** 2-Approximation
- Introduction to LP and an overview of Simplex Method
  - Simplex in Excel++, Ubuntu lp_solve, and custom code
- Introducing the weighted MVC (MWVC)
  - Problem with MVC approximation algorithms…
  - Reducing MWVC to ILP
  - Relaxing ILP to LP (we can use Simplex) and rounding up the answer
  - Analysis of that solution: **2-approximation**
  - Plus yet another alternative 2-approximation solution

Admin

- PS1 is open until this **Sun, 27 Aug 23, 11.59pm**
  - As of Thu, 24 August 2023, 11.30am…
    - 47 with ACs on A+B+C+D or more,
      - Ignore E+F+G+H if you have (many) other things to do
    - 7 more with only ≥ 2/4 ACs
      - Should also be on track to complete PS1, but will have a busy day today/tomorrow…
    - **But a staggering 69-47-7 = 15 pax with only 1 or 0 AC…**
      - Are you staying in this course or not?
  - Consider that first PS1, these first two lectures (,plus my Lecture 03a+03b recordings and future Tut01+Tut02) and decide if the (optional/elective) CS4234 is for you…
    - I am OK with if you drop* (Terms and conditions apply)
- PS2 still starts from **Sat, 26 Aug 23, 08.00am**
  - Four (:O) **NP-hard** optimization problems
- Tut01.pdf is out; first tutorial next **Mon, 28 Aug 23**

PS1 (More) Hints

- Only shown in the recording