

CS4234

Optimiz(s)ation Algorithms

L2 – Approximation Algorithms
and Linear Programming

<https://visualgo.net/en/mvc>

(both unweighted and weighted version)



Right Infringements on NUS Course Materials

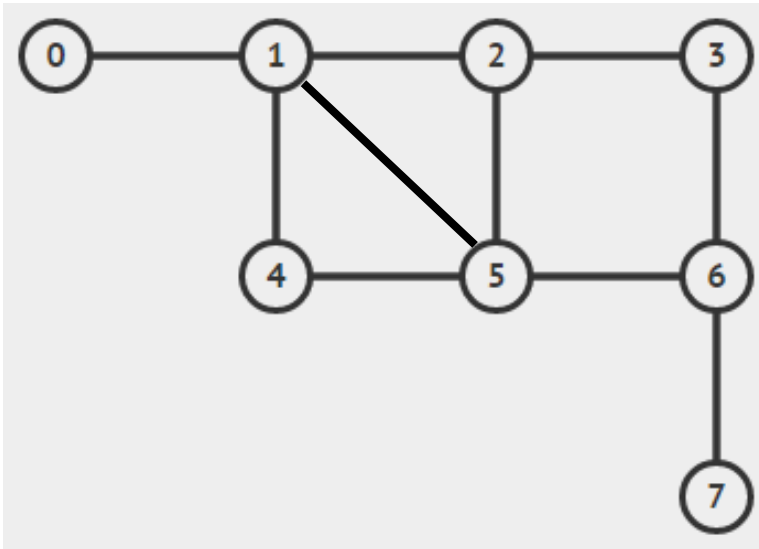
All course participants (including permitted guest students) who have access to the course materials on LumiNUS or any approved platforms by NUS for delivery of NUS modules are not allowed to re-distribute the contents in any forms to third parties without the explicit consent from the module instructors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

For CS4234: probably the tutorial and past midterm/final answers that are preferably not stored in public domain that will be 'easily indexed by Google'

Lecture 1 Recap (1)

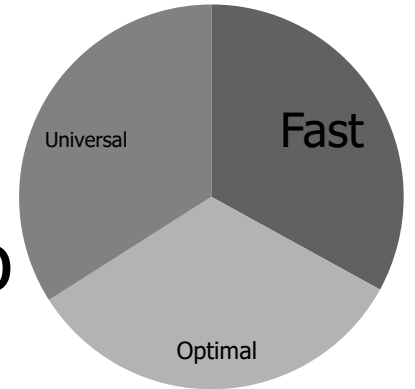
Definition of the first Combinatorial Optimization Problem (COP) in CS4234: `Min-Vertex-Cover`

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



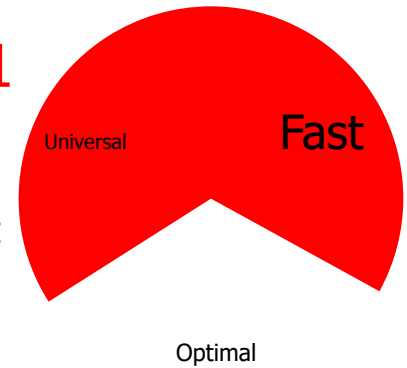
Lecture 1 Recap (2)

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 → lost universality
 - e.g., MVC on (Binary) Tree, we have DP solution (or? see T01)
- Hope that it is on small instance/small 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, but non-optimal solution that can be computed fast
 - e.g., MVC, but we are OK if the answer is at most $2 \times \text{OPT}$
 - 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. */
```

```
1 Algorithm: ApproxVertexCover-1( $G = (V, E)$ )
```

```
2 Procedure:
```

```
3  $C \leftarrow \emptyset$ 
```

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

```
4 while  $E \neq \emptyset$  do
```

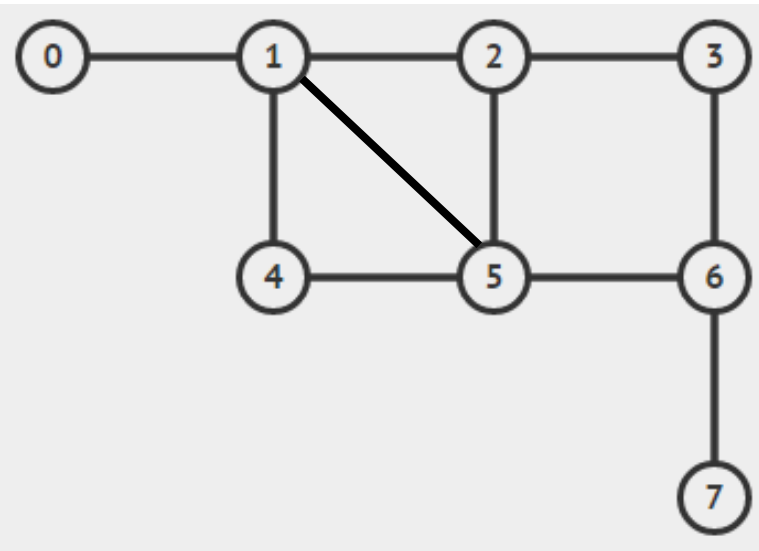
```
5   Let  $e = (u, v)$  be any edge in  $G$ .
```

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

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

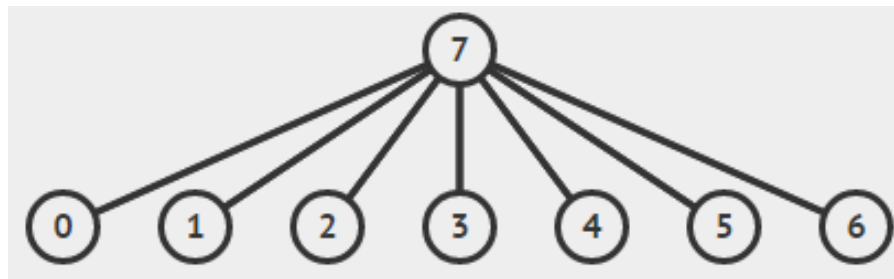
```
8 return  $C$ 
```

Q: how to efficiently implement this?

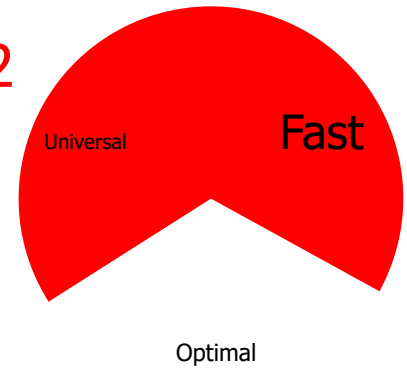


Can be very bad:

– **??-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. */
```

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

```
2 Procedure:
```

```
3  $C \leftarrow \emptyset$ 
```

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

```
4 while  $E \neq \emptyset$  do
```

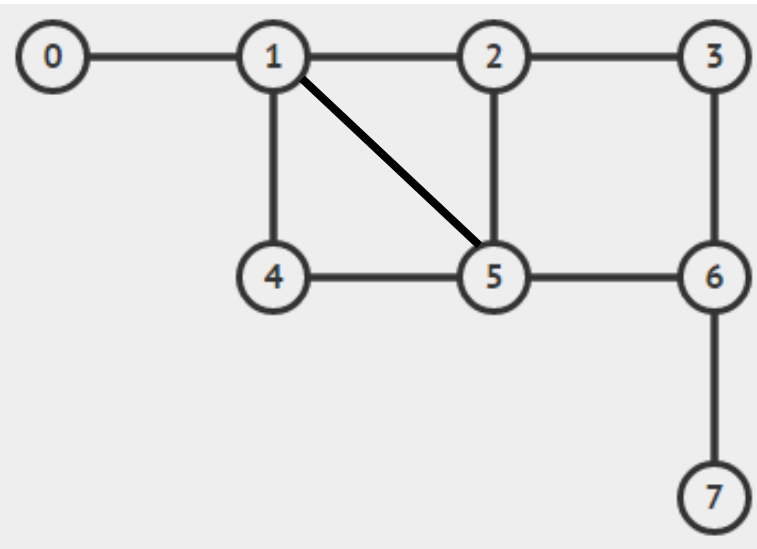
```
5   Let  $e = (u, v)$  be any edge in  $G$ .
```

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

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

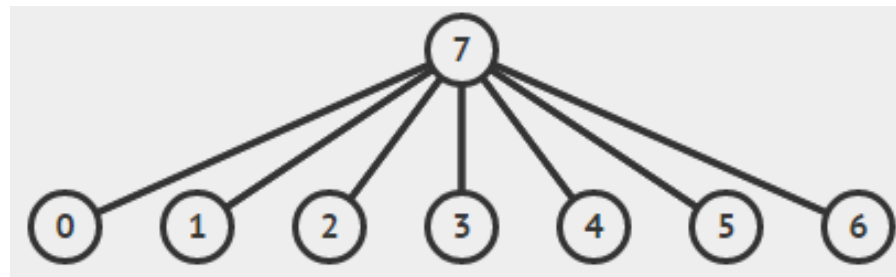
```
8 return  $C$ 
```

Q: how to efficiently implement this?

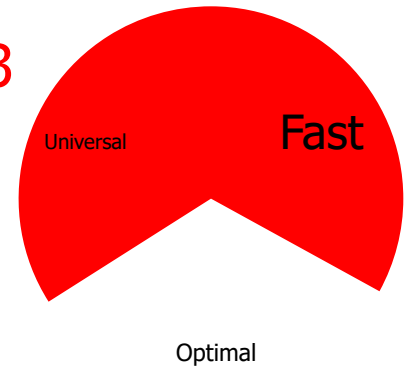


Not that bad?

– 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. */
```

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

```
2 Procedure:
```

```
3  $C \leftarrow \emptyset$ 
```

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

```
4 while  $E \neq \emptyset$  do
```

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

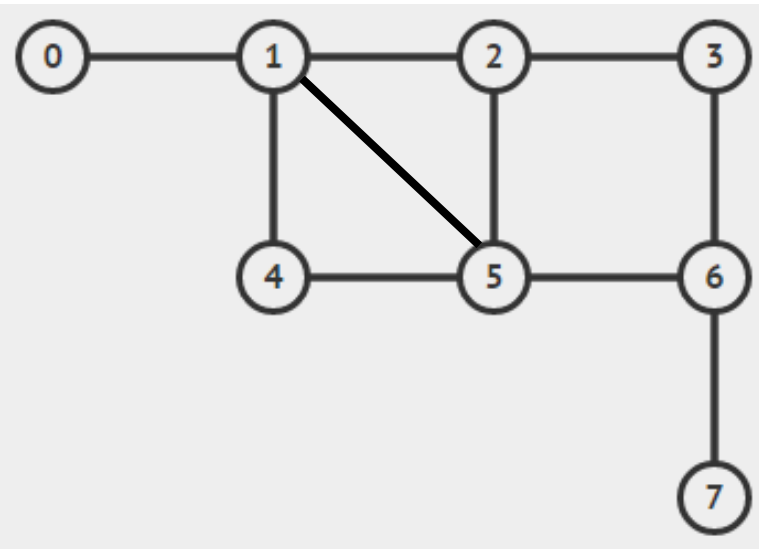
```
6   Let  $u = \operatorname{argmax}_{x \in V} d(x)$ 
```

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

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

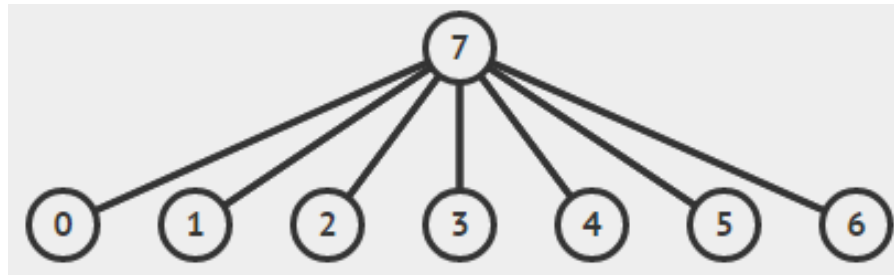
```
9 return  $C$ 
```

Q: how to efficiently implement this?

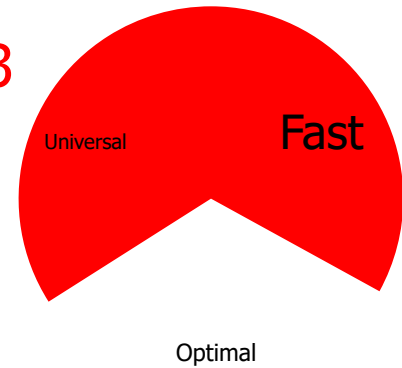


Not that bad?

– 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. */
```

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

```
2 Procedure:
```

```
3  $C \leftarrow \emptyset$ 
```

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

```
4 while  $E \neq \emptyset$  do
```

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

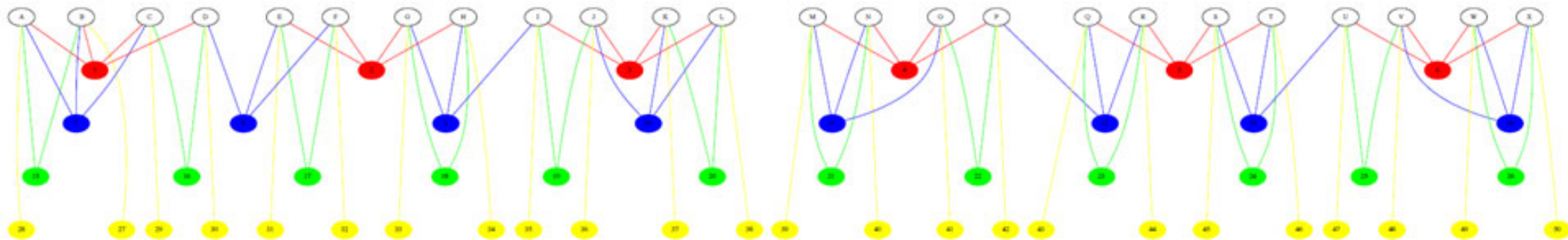
```
6     Let  $u = \operatorname{argmax}_{x \in V} d(x)$ 
```

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

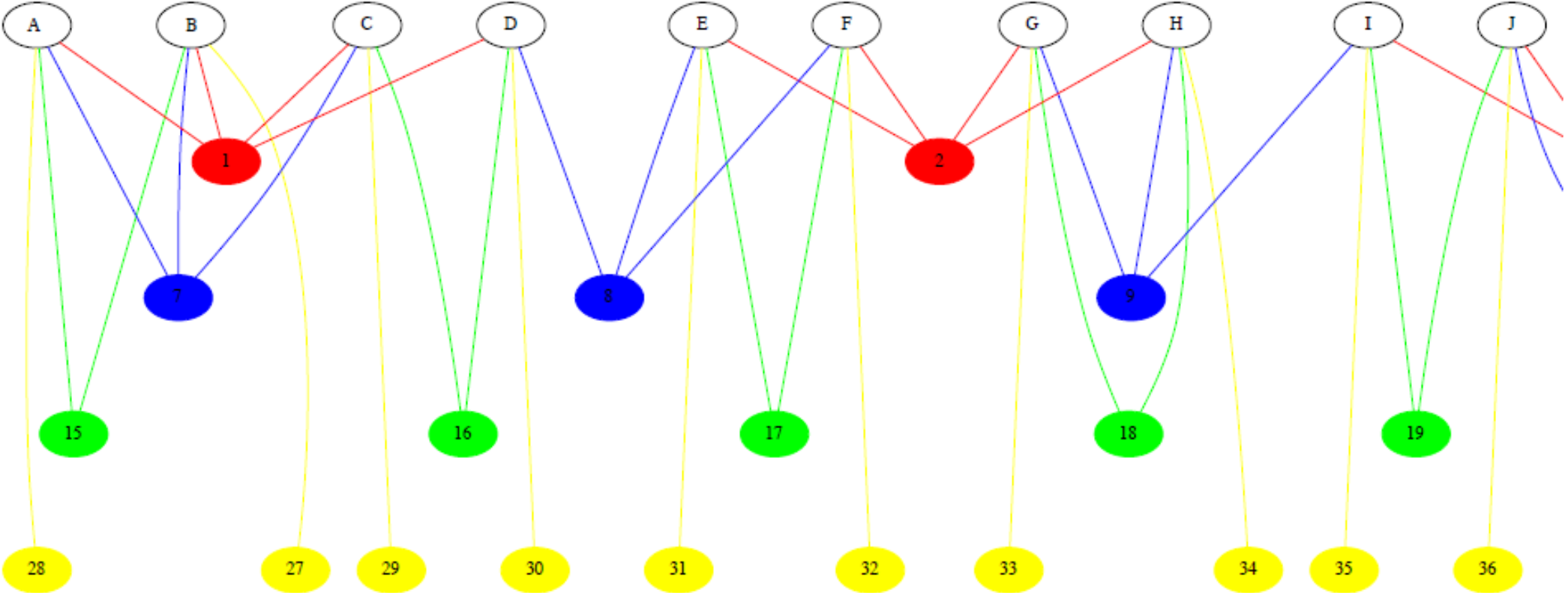
```
8      $G \leftarrow G_{-\{u\}}$  // Remove  $u$  and all adjacent edges from  $G$ .
```

```
9 return  $C$ 
```

This is $\log-n$ factor worse than optimal answer



Zoomed-in



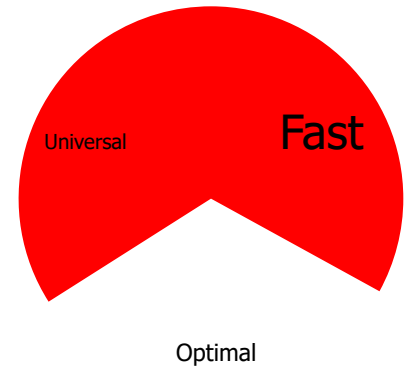
Approx Algo for MVC – Deterministic 2

```
/* 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. */
1 Algorithm: ApproxVertexCover-2( $G = (V, E)$ )
2 Procedure:
3    $C \leftarrow \emptyset$ 
   /* Repeat until every edge is covered: */
4   while  $E \neq \emptyset$  do
5     Let  $e = (u, v)$  be any edge in  $G$ .
6      $C \leftarrow C \cup \{u, v\}$ 
7      $G \leftarrow G_{-\{u, v\}}$  // Remove  $u$  and  $v$  and all adjacent edges from  $G$ .
8   return  $C$ 
```

- This one has 2-approximation proof
 - Let $\mathbf{E'}$ be the edges considered by this algorithm, this $\mathbf{E'}$ is actually a **matching \mathbf{M}** (revisited in the next few weeks of CS4234, details in the PDF), $\mathbf{E' = M}$
 - $|\mathbf{M}| \leq |\mathbf{OPT(G)}|$ (details in the PDF) and $\mathbf{C = 2 * |E'|}$
 - So $\mathbf{C = 2 * |M| \leq 2 * |OPT(G)|}$

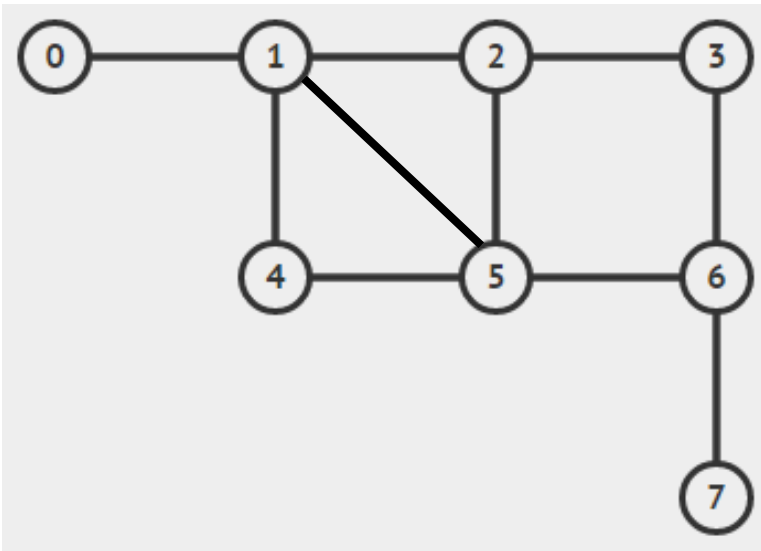
Try Deterministic 2-Opt live at <https://visualgo.net/en/mvc>

Approx Algo for MVC - Randomized



```
1 Algorithm: RandomizedVertexCover( $G = (V, E)$ )
2 Procedure:
3    $C \leftarrow \emptyset$ 
4   /* Repeat until every edge is covered: */
5   while  $E \neq \emptyset$  do
6     Let  $e = (u, v)$  be any edge in  $G$ .
7      $b \leftarrow \text{Random}(0,1)$  // Returns  $\{0,1\}$  each with probability  $1/2$ .
8     if  $b = 0$  then  $z = u$ 
9     else if  $b = 1$  then  $z = v$ 
10     $C \leftarrow C \cup \{z\}$ 
11     $G \leftarrow G_{-z}$  // Remove  $z$  and all adjacent edges from  $G$ .
12  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 at least $1/2$
- 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 $E = \emptyset$
 - In expectation *verbal discussion*

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

Before I forget

Time to do attendance taking 😊

- <https://inetapps.nus.edu.sg/ctr/Home>

You need to:

- Declare your temperature regularly
- Do your FET/ART test if you are scheduled for it
- Should *not* be on SHN/LOA at the moment

A high-level tour

Also in CP4 Book 2, p586-589

LINEAR PROGRAMMING

Linear Programming (LP)

A typical LP consists of three components:

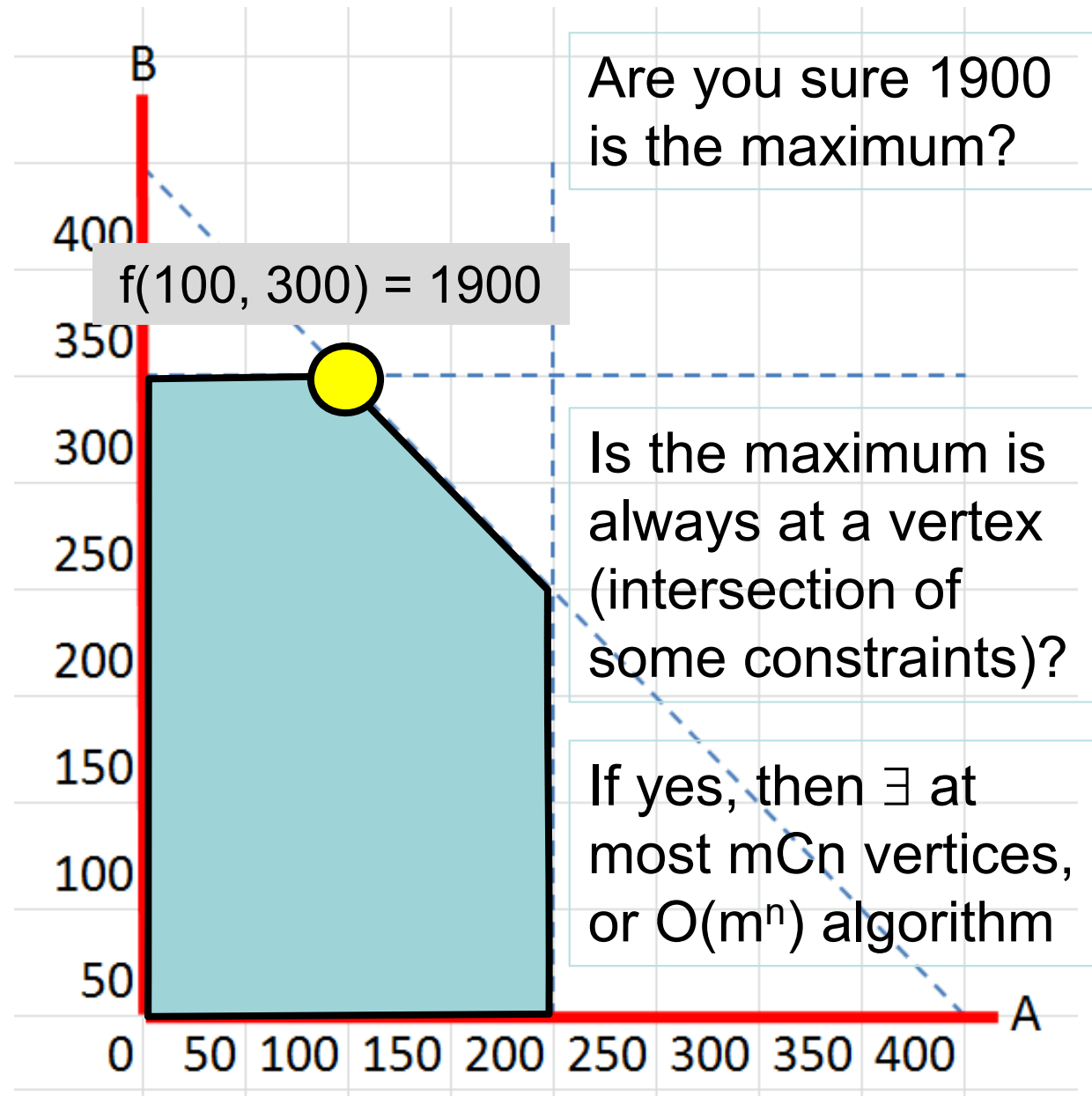
1. A list of (real-valued) variables x_1, x_2, \dots, x_n
 - The goal: To find good values for these variables
2. An objective function $f(x_1, x_2, \dots, x_n)$ that you are trying to maximize or minimize
 - The goal is to find the best values for the variables so as to optimize this function
3. A set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

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

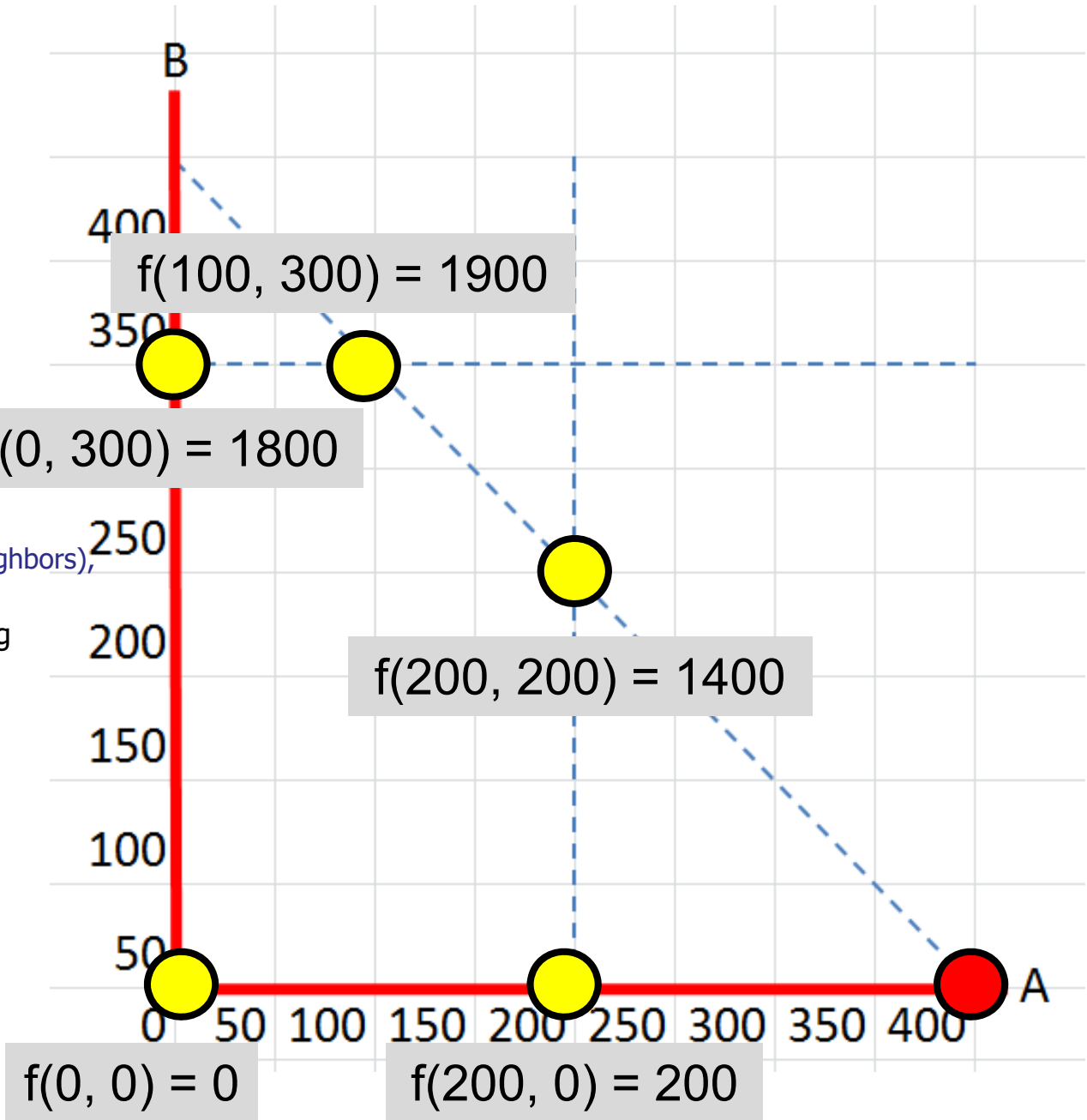


Simplex Method

1. Find any (feasible) vertex v
2. Examine all the neighboring vertices of v :
 v_1, v_2, \dots, v_k
3. Calculate $f(v), f(v_1), f(v_2), \dots, f(v_k)$
 - 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)

Simplex Live Example

1. Find any (feasible) vertex v
2. Examine all the neighboring vertices of v_1, v_2, \dots, v_k
3. Calculate $f(v), f(v_1), f(v_2), \dots, f(v_k)$
 - 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 in LT19 computer can be setup quickly to use Solver Add-in)

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

The Solver Parameters dialog box is shown over an Excel spreadsheet. The spreadsheet has columns A through E and rows 1 through 14. The data in the spreadsheet is as follows:

	A	B	C	D	E
1	Variables	A	B		
2	Coefficients		1	6	
3	Solutions		0	0	<= run Data->Solver to see
4	Z		0		you need to turn on Solver
6	Constraints 1		1	0	<= 200
7	Constraints 2		0	1	<= 300
8	Constraints 3		1	1	<= 400
10		LHS	RHS		
11	Constraints 1		0	200	
12	Constraints 2		0	300	
13	Constraints 3		0	400	

The Solver Parameters dialog box is configured as follows:

- Set Objective: $\$B\4
- To: Max Min Value Of: 0
- By Changing Variable Cells: $\$B\$3:\$C\3
- Subject to the Constraints:
 - $\$B\$11:\$B\$13 \leq \$C\$11:\$C\13
 - $\$B\$3:\$C\$3 \geq 0$
- Make Unconstrained Variables Non-Negative
- Select a Solving Method: Simplex LP
- Solving Method: Select the GRG Nonlinear engine for Solver Problems that are smooth nonlinear. Select the LP Simplex engine for linear Solver Problems, and select the Evolutionary engine for Solver problems that are non-smooth.

The Solver Results dialog box is shown over the same Excel spreadsheet. The spreadsheet data is the same as in the previous screenshot. The Solver Results dialog box is configured as follows:

- Solver found a solution. All Constraints and optimality conditions are satisfied.
- Keep Solver Solution
- Restore Original Values
- Return to Solver Parameters Dialog
- Outline Reports
- Buttons: OK, Cancel, Save Scenario...
- Footer: Solver found a solution. All Constraints and optimality conditions are satisfied. When the GRG engine is used, Solver has found at least a local optimal solution. When Simplex LP is used, this means Solver has found a global optimal solution.

lp_solve in Ubuntu

Cannot do live demonstration in this LT19 desktop computer...
it is on Windows 😞

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

```
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;

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 3 AYs ago)

Any volunteer to convert this to Python?

Now use any tool to solve this

- Maximize $143x+60y$
- Such that:
 - $120x+210y \leq 15000$
 - $110x+30y \leq 4000$
 - $x+y \leq 75$
- And x and y are non-negative

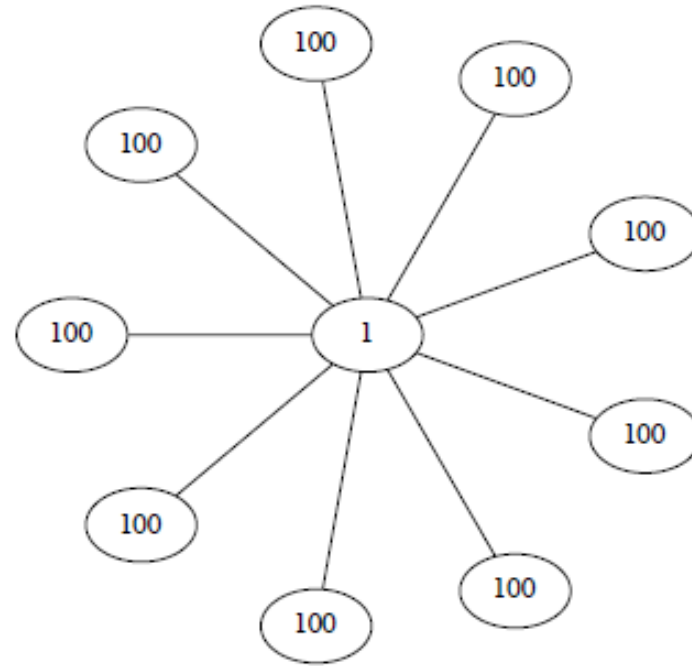
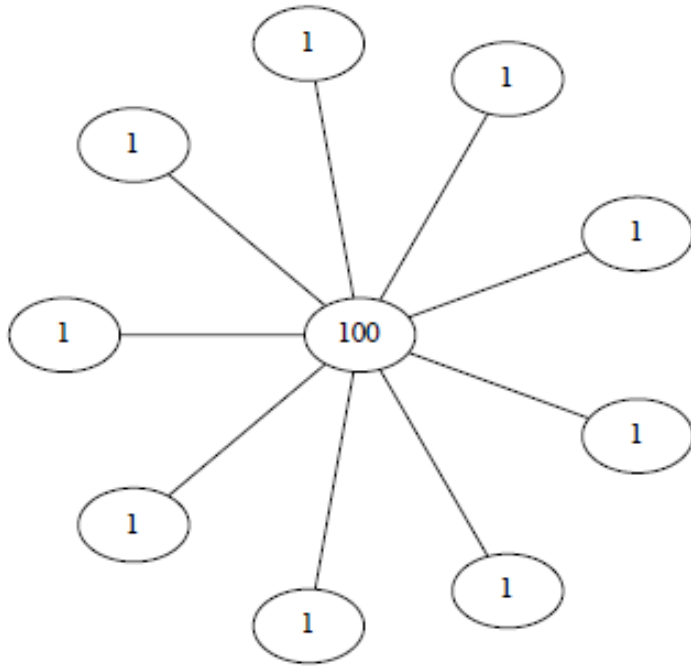
PS: Steven will randomize the LP exercise each year

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` "fails" 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 $0 = \text{not in VC}$ and $1 = \text{in VC}$):

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

where:

$$x_i + x_j \geq 1 \quad \text{for all } (i, j) \in E$$

$$x_j \geq 0 \quad \text{for all } j \in V$$

$$x_j \leq 1 \quad \text{for all } j \in V$$

$$x_j \in \mathbb{Z} \quad \text{for all } j \in V$$

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

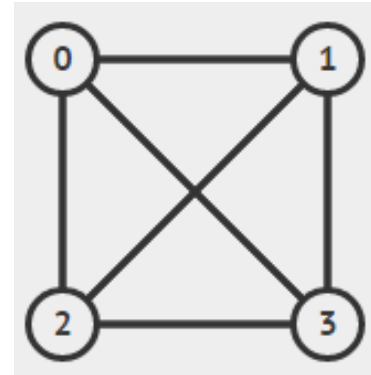
Min-Vertex-Cover (set $w(v_j)$ to all 1) \leq_p ILP

So ILP is also NP-hard

MWVC as a Relaxed Linear Program

Relaxing the Integer constraint

$$\min \left(\sum_{j=1}^n w(v_j) \cdot x_j \right) \quad \text{where:}$$
$$x_i + x_j \geq 1 \quad \text{for all } (i, j) \in E$$
$$x_j \geq 0 \quad \text{for all } j \in V$$
$$x_j \leq 1 \quad \text{for all } j \in V$$



Assume w is all 1

Example LP solution

$$x_0 = x_1 = x_2 = x_3 = 0.5$$

What should we do?

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

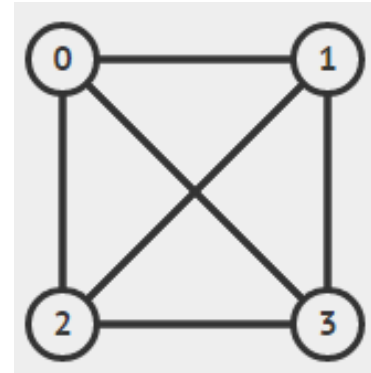
Round up x_j value if it is ≥ 0.5 let $y_j = 1$ if $x_j \geq 1/2$, and let $y_j = 0$ otherwise.

But is this a good approximation?

Analysis

$$\text{OPT}(G) = \text{OPT}(\text{ILP}) \geq \text{OPT}(\text{LP})$$

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



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 $\leq 2 \times \text{OPT}(\text{LP})$

$$\sum_{j=1}^n w(v_j) \cdot y_j \leq \sum_{j=1}^n w(v_j) \cdot (2x_j)$$

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

$$\leq 2 \left(\sum_{j=1}^n w(v_j) \cdot x_j \right)$$

$$\leq 2 \times \text{OPT}(G)$$

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

Linear Programming Summary

General form of an LP:

1. A set of variables: x_1, x_2, \dots, 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: **$\max c^T x$ where $Ax \leq b$ and $x \geq 0$**

LP in Standard Form

Exercise to translate given LP into standard form

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

Details in the PDF

Wait...

- There is an even simpler 2-Approximation algorithm for MWVC... which is...

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++ and Ubuntu Ip_solve
- 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 already out and remains open until this Sat, 21 Aug
 - As of Thursday, 19 August 2021, 11am...
 - 22 with 5/5 ACs (first 15 already reviewed, the rest 'soon'),
 - 10 more with $\geq 2/5$ ACs (should also be on track but will be busy...), but
 - 8 with totally 0 AC (???)
 - Consider these first two lectures and that first PS1 to decide if the (optional) CS4234 is for you...
 - I am OK with if you drop*
 - Terms and conditions apply
- PS2 will out immediately after PS1 ends
 - Still 5 (:O:O:O) NP-hard optimization problems
 - If this is too much, PS3+PS4 will be just 4 problems (lah)...
- Tut01.pdf is out and we will run first tutorial next Mon, 23 Aug

PS1 (More) Hints

- Only shown in the recording