Lecture #18

CS 170

Spring 2021

Reductions

- Reducing Problem A to Problem B
  = using a subroutine for solving Problem B to solve Problem A

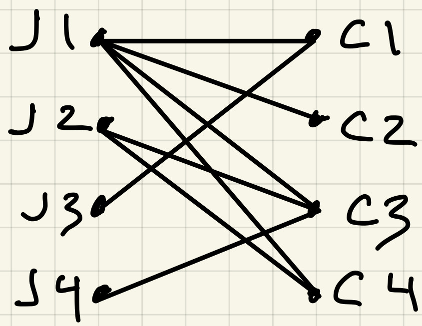
- Good news:

- Bad news:

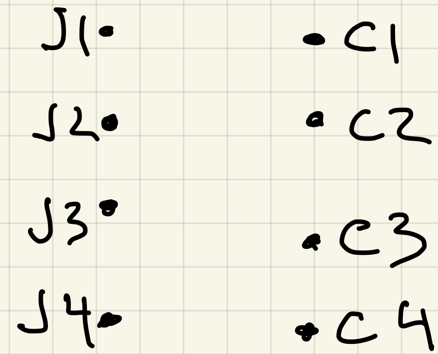
- Assumptions:

Examples

- Good news:
  - Reduce Bipartite Matching (BM) to MaxFlow(MF)



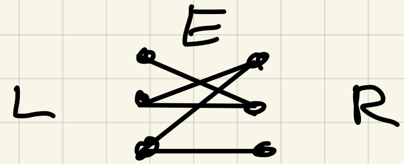
How many jobs and computers can we pair up?



  - Reduce any polynomial time problem to
  - Reduce matrix inversion to
- Bad news:

Bipartite Matching (BM)

Input: Bipartite Graph $G = (L, R, E)$, $E \subseteq L \times R$



Matching:

Goal:

Ex: $L =$, $R =$

$L =$, $R =$

Connect BM and Max Flow (MF) (1/2)

BM:

undirected $G = (L, R, E)$

matching $M \subseteq E$ touches any vertex at most once

goal: maximize $|M|$

MF

directed graph $G = (V, E)$ with source $s \in V$ and sink $t \in V$

edge "capacities" $c_e \geq 0$

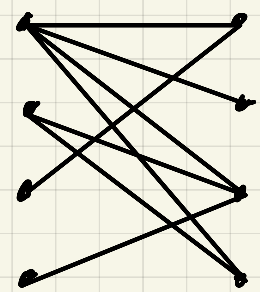
goal: maximize "flow" from s to t subject to capacity limits, conservation of flow

To solve BM using MF:

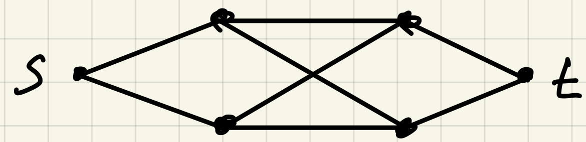
Connect BM and Max Flow (MF) (2/2)

To solve BM using MF:

- need to identify s and t
- need to set capacities c_e
- need to direct edges
- need to connect $|M|$ with flow



What could go wrong?



Recall MF algorithm (Ford-Fulkerson)

repeat
    find a path from s to t with capacity>0
    increase flow along path by maximum amount
until no path from s to t with capacity>0

Claim: There is a 1-1 correspondence between solutions to BM and integer solutions to MF

- Let M be a maximum matching.

- Let $v(E)$ be integer solution to MF where $v(e)$ = flow on edge e

Defining Reductions

Def: Problem A reduces to Problem B ($A \to B$) if there are "efficient" algorithms Preprocess and Postprocess such that solution $A(X)$ is

Ex: $A = BM$ and $B = MF$

Circuit Value Problem (CV)

- Def: A Boolean Circuit is a DAG with
  - input nodes $x_i = 0$ or 1
  - AND nodes $\begin{matrix} x_i \\ x_j \end{matrix} \rightarrow \text{AND} \rightarrow x_k = x_i \wedge x_j$
  - OR nodes $\begin{matrix} x_i \\ x_j \end{matrix} \rightarrow \text{OR} \rightarrow x_k = x_i \vee x_j$
  - NOT nodes $x_i \rightarrow \text{NOT} \rightarrow x_k = \overline{x}_i$
  - output nodes: subset of resulting x_k
- CV: Given a Boolean Circuit, is its output $=1$?

Any efficient algorithm $\longrightarrow$ CV $\longrightarrow$ LP

- Informal argument (CS 172 discusses Turing machines)
  - A computer with poly-sized memory can run algorithm in poly-time
  - Have 1 copy of circuit representing internal state of computer for each time step, with output of copy i = input for copy $i+1$
  -
- CV $\to$ LP

Matrix Multiply (MM) $\longleftrightarrow$ Matrix Inversion (MI)

- Each one reduces to other
  - "Fast" algorithm for one $\Rightarrow$ works for other
- Easy direction: MM $\rightarrow$ MI

Form $X = \begin{bmatrix} I & -A & 0 \\ 0 & I & -B \\ 0 & 0 & I \end{bmatrix}$, compute $X^{-1} = \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}$

If inverting $n \times n$ X costs $O(n^e)$

then multiplying $n \times n$ $A \cdot B$ costs

Matrix Multiply (MM) $\longleftrightarrow$ Matrix Inversion (MI)

- Trickier Direction: MI $\rightarrow$ MM

2×2 Gaussian Elimination:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} I & 0 \\ \underbrace{C \cdot A^{-1}}_{Y} & I \end{bmatrix} \cdot \begin{bmatrix} A & B \\ 0 & \underbrace{D - C \cdot A^{-1} \cdot B}_{S} \end{bmatrix}$$

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} A & B \\ 0 & S \end{bmatrix}^{-1} \cdot \begin{bmatrix} I & 0 \\ Y & I \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} & \overbrace{-A^{-1} \cdot B \cdot S^{-1}}^{Z} \\ 0 & S^{-1} \end{bmatrix} \cdot \begin{bmatrix} I & 0 \\ -Y & I \end{bmatrix}$$

$$= \begin{bmatrix} A^{-1} - Z \cdot Y & Z \\ -S^{-1} \cdot Y & S^{-1} \end{bmatrix}$$