Lecture #18
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)

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

\begin{itemize}
  \item \text{Bad news:}
  \item \text{Reduce any polynomial time problem to}
  \item \text{Reduce matrix inversion to}
\end{itemize}
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 graph $G = (L, R, E)$
matching $M \subseteq E$ touches any vertex at most once

goal: maximize $|M|$

To solve BM using MF:

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
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 IMI 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 $\rightarrow$ 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 \( x_i \rightarrow \text{AND} \rightarrow x_k = x_i \land x_j \)
  - OR nodes \( x_i \rightarrow \text{OR} \rightarrow x_k = x_i \lor x_j \)
  - 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 $\rightarrow$ CV $\rightarrow$ 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 $i = \text{input for copy } i + 1$

- CV $\rightarrow$ LP
Matrix Multiply (MM) $\leftrightarrow$ Matrix Inversion (MI)

- Each one reduces to other
  - "Fast" algorithm for one $\Rightarrow$ works for other

- Easy direction: MM $\rightarrow$ MI

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

  If inverting $n \times n$ $X$ costs $O(n^3)$, then multiplying $n \times n$ $A \cdot B$ costs
Matrix Multiply (MM) ↔ Matrix Inversion (MI)

- Trickier Direction: MI → MM

2x2 Gaussian Elimination:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} = \begin{pmatrix} I & 0 \\ C \cdot A^{-1} & I \end{pmatrix} \cdot \begin{pmatrix} A & B \\ 0 & D - C \cdot A^{-1} \cdot B \end{pmatrix}$$

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = (A B)^{-1} \cdot \begin{pmatrix} I & 0 \\ Y & I \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} & -A^{-1} \cdot B \cdot S^{-1} \\ 0 & S^{-1} \end{pmatrix} \cdot \begin{pmatrix} I & 0 \\ -Y & I \end{pmatrix}$$

$$= \begin{pmatrix} A^{-1} & -2Y \\ -S^{-1} \cdot Y & S^{-1} \end{pmatrix}$$