

Combinatorial Optimization Course

# **Bipartite Matching Problem**

## *Part II*

Mohammad Hossein Bateni

February 10, 2005

# Topics

- Maximum Matching
  - ★ Direct Algorithm
  - ★ Maximum Flow Application
  - ★ Vertex Cover, Edge Cover, Independent Set

# Topics

- Maximum Matching
  - ★ Direct Algorithm
  - ★ Maximum Flow Application
  - ★ Vertex Cover, Edge Cover, Independent Set
  
- Minimum Weight Perfect Matching
  - ★ Alpha beta Algorithm
  - ★ Min-Cost Flow Algorithm
  - ★ Weighted Vertex Cover

# Similar Problems

- Maximum Weight Perfect Matching

# Similar Problems

- Maximum Weight Perfect Matching
  - ★ *By negating the edge weights*

# Similar Problems

- Maximum Weight Perfect Matching
  - ★ *By negating the edge weights*
- Minimum Weight Matching

# Similar Problems

- Maximum Weight Perfect Matching
  - ★ *By negating the edge weights*
- Minimum Weight Matching
  - ★ *By adding zero edges*

# Similar Problems

- Maximum Weight Perfect Matching
  - ★ *By negating the edge weights*
- Minimum Weight Matching
  - ★ *By adding zero edges*
- Maximum Weight Matching

# Similar Problems

- Maximum Weight Perfect Matching
  - ★ *By negating the edge weights*
- Minimum Weight Matching
  - ★ *By adding zero edges*
- Maximum Weight Matching
  - ★ *By adding zero edges*
  - ★ *By negating the edge weights*

# Formulation

- Primal Formulation

$$\min \sum_{i,j} c_{ij} x_{ij}$$

$$\sum_{j=1}^n x_{ij} = 1 \quad ; \forall i = 1, \dots, n$$

$$\sum_{i=1}^n x_{ij} = 1 \quad ; \forall j = 1, \dots, n$$

$$x_{ij} \geq 0 \quad ; \forall i, j$$

# Formulation

- Primal Formulation

$$\begin{aligned} \min \quad & \sum_{i,j} c_{ij} x_{ij} \\ \sum_{j=1}^n x_{ij} &= 1 \quad ; \forall i = 1, \dots, n \\ \sum_{i=1}^n x_{ij} &= 1 \quad ; \forall j = 1, \dots, n \\ x_{ij} &\geq 0 \quad ; \forall i, j \end{aligned}$$

- Integrality Constraints are not necessary

# Formulation

- Primal Formulation

$$\begin{aligned} \min \quad & \sum_{i,j} c_{ij} x_{ij} \\ \sum_{j=1}^n x_{ij} &= 1 \quad ; \forall i = 1, \dots, n \\ \sum_{i=1}^n x_{ij} &= 1 \quad ; \forall j = 1, \dots, n \\ x_{ij} &\geq 0 \quad ; \forall i, j \end{aligned}$$

- Integrality Constraints are not necessary
  - ★ using Totally Unimodular Matrices Theorem

# Formulation

- Primal Formulation

$$\begin{aligned} \min \quad & \sum_{i,j} c_{ij} x_{ij} \\ \sum_{j=1}^n x_{ij} &= 1 \quad ; \forall i = 1, \dots, n \\ \sum_{i=1}^n x_{ij} &= 1 \quad ; \forall j = 1, \dots, n \\ x_{ij} &\geq 0 \quad ; \forall i, j \end{aligned}$$

- Integrality Constraints are not necessary
  - ★ using Totally Unimodular Matrices Theorem
  - ★ similarity to Hitchcock problem

# Formulation

- Primal Formulation

$$\begin{aligned} \min \quad & \sum_{i,j} c_{ij} x_{ij} \\ \sum_{j=1}^n x_{ij} &= 1 \quad ; \forall i = 1, \dots, n \\ \sum_{i=1}^n x_{ij} &= 1 \quad ; \forall j = 1, \dots, n \\ x_{ij} &\geq 0 \quad ; \forall i, j \end{aligned}$$

- Integrality Constraints are not necessary

- ★ using Totally Unimodular Matrices Theorem
- ★ similarity to Hitchcock problem
- ★ Direct proof

# Solving The Problem

- This is a special case of Hitchcock Problem

# Solving The Problem

- This is a special case of Hitchcock Problem
- Can be solved using Alphabeta

# Solving The Problem

- This is a special case of Hitchcock Problem
- Can be solved using Alphabeta
- Simplicity and improvement in running time

# Solving The Problem

- This is a special case of Hitchcock Problem
- Can be solved using Alphabeta
- Simplicity and improvement in running time
- The DRP becomes an unweighted matching problem using the **admissible** edges

# Solving The Problem

- This is a special case of Hitchcock Problem
- Can be solved using Alphabeta
- Simplicity and improvement in running time
- The DRP becomes an unweighted matching problem using the **admissible** edges
- **admissible edges** are those edges for which we have

$$c_{ij} = \alpha_i + \beta_j$$

# AlphaBeta

- RP is like this

$$\min \zeta = \sum_i \pi_i + \sum_j \gamma_j$$

$$\sum_j x_{ij} + \pi_i = 1 \quad ; \quad \forall i$$

$$\sum_i x_{ij} + \gamma_j = 1 \quad ; \quad \forall j$$

$$x_{ij} = 0 \quad ; \quad \forall i, j : \alpha_i + \beta_j \leq c_{ij}$$

$$x_{ij} = 1 \quad ; \quad \forall i, j : \alpha_i + \beta_j = c_{ij}$$

$$\pi_i, \gamma_j \geq 0 \quad ; \quad \forall i, j$$

inadmissible edges

admissible edges

# AlphaBeta

- RP is like this

$$\min \zeta = \sum_i \pi_i + \sum_j \gamma_j$$

$$\sum_j x_{ij} + \pi_i = 1 \quad ; \quad \forall i$$

$$\sum_i x_{ij} + \gamma_j = 1 \quad ; \quad \forall j$$

$$x_{ij} = 0 \quad ; \quad \forall i, j : \alpha_i + \beta_j \leq c_{ij}$$

inadmissible edges

$$x_{ij} = 1 \quad ; \quad \forall i, j : \alpha_i + \beta_j = c_{ij}$$

admissible edges

$$\pi_i, \gamma_j \geq 0 \quad ; \quad \forall i, j$$

- Note that

$$\zeta = 2n - 2 \sum_{i,j} x_{ij}$$

# Alphabet (Cont'd)

- So, RP simplifies as follows

$$\max \sum_{ij} x_{ij}$$

$$\sum_j x_{ij} \leq 1 \quad ; \quad \forall i$$

$$\sum_i x_{ij} \leq 1 \quad ; \quad \forall j$$

$$x_{ij} = 0 \quad ; \quad \forall i, j : \alpha_i + \beta_j \leq c_{ij} \quad \text{inadmissible edges}$$

$$x_{ij} = 1 \quad ; \quad \forall i, j : \alpha_i + \beta_j = c_{ij} \quad \text{admissible edges}$$

# Alphabeta (Cont'd)

- So, RP simplifies as follows

$$\begin{aligned} & \max \sum_{ij} x_{ij} \\ & \sum_j x_{ij} \leq 1 \quad ; \quad \forall i \\ & \sum_i x_{ij} \leq 1 \quad ; \quad \forall j \\ & x_{ij} = 0 \quad ; \quad \forall i, j : \alpha_i + \beta_j \leq c_{ij} \quad \text{inadmissible edges} \\ & x_{ij} = 1 \quad ; \quad \forall i, j : \alpha_i + \beta_j = c_{ij} \quad \text{admissible edges} \end{aligned}$$

- Which is recognized as a maximum matching using only the admissible edges

# Alphabeta (Cont'd)

- So, RP simplifies as follows

$$\begin{aligned} & \max \sum_{ij} x_{ij} \\ & \sum_j x_{ij} \leq 1 \quad ; \quad \forall i \\ & \sum_i x_{ij} \leq 1 \quad ; \quad \forall j \\ & x_{ij} = 0 \quad ; \quad \forall i, j : \alpha_i + \beta_j \leq c_{ij} \quad \text{inadmissible edges} \\ & x_{ij} = 1 \quad ; \quad \forall i, j : \alpha_i + \beta_j = c_{ij} \quad \text{admissible edges} \end{aligned}$$

- Which is recognized as a maximum matching using only the admissible edges
- If this graph does not have a perfect matching, we have to change the dual variables,  $\alpha_i$  and  $\beta_j$

# Alphabeta (Cont'd)

- Let  $X$  and  $Y$  be the upper and lower partitions of the graph

# Alphabeta (Cont'd)

- Let  $X$  and  $Y$  be the upper and lower partitions of the graph
- Let  $U$  be free vertices of  $X$

# Alphabeta (Cont'd)

- Let  $X$  and  $Y$  be the upper and lower partitions of the graph
- Let  $U$  be free vertices of  $X$
- Let  $S$  and  $T$  be vertices of  $X$  and  $Y$  which are reachable by alternating paths from  $U$

# Alphabeta (Cont'd)

- Let  $X$  and  $Y$  be the upper and lower partitions of the graph
- Let  $U$  be free vertices of  $X$
- Let  $S$  and  $T$  be vertices of  $X$  and  $Y$  which are reachable by alternating paths from  $U$
- Compute  $\theta$  as follows

$$\theta = \frac{1}{2} \min_{\substack{i \in S \\ j \notin T}} \{c_{ij} - \alpha_i - \beta_j\}$$

# Alphabeta (Cont'd)

- Let  $X$  and  $Y$  be the upper and lower partitions of the graph
- Let  $U$  be free vertices of  $X$
- Let  $S$  and  $T$  be vertices of  $X$  and  $Y$  which are reachable by alternating paths from  $U$
- Compute  $\theta$  as follows

$$\theta = \frac{1}{2} \min_{\substack{i \in S \\ j \notin T}} \{c_{ij} - \alpha_i - \beta_j\}$$

- Add  $\theta$  to dual variables of  $S$  and  $Y - T$  and subtract it from dual variables of  $X - S$  and  $T$

# Alphabeta (Cont'd)

- Let  $X$  and  $Y$  be the upper and lower partitions of the graph
- Let  $U$  be free vertices of  $X$
- Let  $S$  and  $T$  be vertices of  $X$  and  $Y$  which are reachable by alternating paths from  $U$
- Compute  $\theta$  as follows

$$\theta = \frac{1}{2} \min_{\substack{i \in S \\ j \notin T}} \{c_{ij} - \alpha_i - \beta_j\}$$

- Add  $\theta$  to dual variables of  $S$  and  $Y - T$  and subtract it from dual variables of  $X - S$  and  $T$
- No matching edge will get inadmissible in the process

# Alphabeta (Cont'd)

- Let  $X$  and  $Y$  be the upper and lower partitions of the graph
- Let  $U$  be free vertices of  $X$
- Let  $S$  and  $T$  be vertices of  $X$  and  $Y$  which are reachable by alternating paths from  $U$
- Compute  $\theta$  as follows

$$\theta = \frac{1}{2} \min_{\substack{i \in S \\ j \notin T}} \{c_{ij} - \alpha_i - \beta_j\}$$

- Add  $\theta$  to dual variables of  $S$  and  $Y - T$  and subtract it from dual variables of  $X - S$  and  $T$
- No matching edge will get inadmissible in the process
- No edge from  $S$  to  $T$  will get inadmissible

# Alphabeta (Cont'd)

- Let  $X$  and  $Y$  be the upper and lower partitions of the graph
- Let  $U$  be free vertices of  $X$
- Let  $S$  and  $T$  be vertices of  $X$  and  $Y$  which are reachable by alternating paths from  $U$
- Compute  $\theta$  as follows

$$\theta = \frac{1}{2} \min_{\substack{i \in S \\ j \notin T}} \{c_{ij} - \alpha_i - \beta_j\}$$

- Add  $\theta$  to dual variables of  $S$  and  $Y - T$  and subtract it from dual variables of  $X - S$  and  $T$
- No matching edge will get inadmissible in the process
- No edge from  $S$  to  $T$  will get inadmissible
- At least one new admissible edge will be formed from  $S$  to  $Y - T$

# Algorithm

1. Initialize

$$\alpha_i = 0$$
$$\beta_j = \min_i \{c_{ij}\}$$

2. **while** there is an unmatched vertex **do**

(a) Try to find an augmenting path using only admissible edges

(b) **if** no augmenting path is found **then** take this minimum and update the dual variables accordingly

$$\theta = \min_{i,j} \{c_{ij} - \alpha_i - \beta_j\}$$

where  $i$  is reachable through alternating paths and  $j$  is not

# Running Time Analysis

- *Initialize* takes  $\Theta(n^2)$

# Running Time Analysis

- *Initialize* takes  $\Theta(n^2)$
- Loop repeats at most  $\Theta(n)$  times

# Running Time Analysis

- *Initialize* takes  $\Theta(n^2)$
- Loop repeats at most  $\Theta(n)$  times
- At most  $\Theta(n)$  successful search

# Running Time Analysis

- *Initialize* takes  $\Theta(n^2)$
- Loop repeats at most  $\Theta(n)$  times
- At most  $\Theta(n)$  successful search
- At most  $\Theta(n^2)$  unsuccessful search

# Running Time Analysis

- *Initialize* takes  $\Theta(n^2)$
- Loop repeats at most  $\Theta(n)$  times
- At most  $\Theta(n)$  successful search
- At most  $\Theta(n^2)$  unsuccessful search
- Each search takes  $\Theta(n^2)$  time

# Running Time Analysis

- *Initialize* takes  $\Theta(n^2)$
- Loop repeats at most  $\Theta(n)$  times
- At most  $\Theta(n)$  successful search
- At most  $\Theta(n^2)$  unsuccessful search
- Each search takes  $\Theta(n^2)$  time
- Total run-time is  $\Theta(n^4)$

# Faster Algorithm

## 1. Initialize

$$\alpha_i = 0$$

$$\beta_j = \min_i \{c_{ij}\}$$

$$slack_j = 0$$

## 2. **while** there is an unmatched vertex **do**

(a) Try to find an augmenting path using only admissible edges

(b) **if** no augmenting path is found **then**

i. take the minimum and update the dual variables accordingly

ii. *continue* searching for the path

(c) Augment along the path

# Faster Algorithm

## 1. Initialize

$$\alpha_i = 0$$

$$\beta_j = \min_i \{c_{ij}\}$$

$$slack_j = 0$$

## 2. **while** there is an unmatched vertex **do**

(a) Try to find an augmenting path using only admissible edges

(b) **if** no augmenting path is found **then**

i. take the minimum and update the dual variables accordingly

ii. *continue* searching for the path

(c) Augment along the path

- The Crucial point is how to find  $\theta$  quickly

# Faster Algorithm (Cont'd)

- We maintain an array **slack** where for a vertex of  $Y - T$  is defined as

$$\text{slack}_j = \frac{1}{2} \min_{i \in S} \{c_{ij} - \alpha_i - \beta_j\}$$

# Faster Algorithm (Cont'd)

- We maintain an array **slack** where for a vertex of  $Y - T$  is defined as

$$\text{slack}_j = \frac{1}{2} \min_{i \in S} \{c_{ij} - \alpha_i - \beta_j\}$$

- Using this array, finding  $\theta$  can be done in  $\Theta(n)$

# Faster Algorithm (Cont'd)

- We maintain an array **slack** where for a vertex of  $Y - T$  is defined as

$$\text{slack}_j = \frac{1}{2} \min_{i \in S} \{c_{ij} - \alpha_i - \beta_j\}$$

- Using this array, finding  $\theta$  can be done in  $\Theta(n)$
- How to maintain the array?

# Faster Algorithm (Cont'd)

- We maintain an array **slack** where for a vertex of  $Y - T$  is defined as

$$\text{slack}_j = \frac{1}{2} \min_{i \in S} \{c_{ij} - \alpha_i - \beta_j\}$$

- Using this array, finding  $\theta$  can be done in  $\Theta(n)$
- How to maintain the array?
- Two occasions that **slack** might need a change
  1. New vertex added to  $S$ 
    - ★ We can update **slack** in  $\Theta(n)$

# Faster Algorithm (Cont'd)

- We maintain an array **slack** where for a vertex of  $Y - T$  is defined as

$$\text{slack}_j = \frac{1}{2} \min_{i \in S} \{c_{ij} - \alpha_i - \beta_j\}$$

- Using this array, finding  $\theta$  can be done in  $\Theta(n)$
- How to maintain the array?
- Two occasions that **slack** might need a change
  1. New vertex added to  $S$ 
    - ★ We can update **slack** in  $\Theta(n)$
  2. changing  $\alpha_i$  or  $\beta_j$ 
    - ★ update in  $\Theta(n)$  by subtracting  $2\theta$  from  $\text{slack}_j \geq 0$

# Faster Algorithm (Cont'd)

- We maintain an array **slack** where for a vertex of  $Y - T$  is defined as

$$\text{slack}_j = \frac{1}{2} \min_{i \in S} \{c_{ij} - \alpha_i - \beta_j\}$$

- Using this array, finding  $\theta$  can be done in  $\Theta(n)$
- How to maintain the array?
- Two occasions that **slack** might need a change
  1. New vertex added to  $S$ 
    - ★ We can update **slack** in  $\Theta(n)$
  2. changing  $\alpha_i$  or  $\beta_j$ 
    - ★ update in  $\Theta(n)$  by subtracting  $2\theta$  from  $\text{slack}_j \geq 0$
- Each case occurs  $O(n)$  in a search

# Faster Algorithm (Cont'd)

- We maintain an array **slack** where for a vertex of  $Y - T$  is defined as

$$\text{slack}_j = \frac{1}{2} \min_{i \in S} \{c_{ij} - \alpha_i - \beta_j\}$$

- Using this array, finding  $\theta$  can be done in  $\Theta(n)$
- How to maintain the array?
- Two occasions that **slack** might need a change
  1. New vertex added to  $S$ 
    - ★ We can update **slack** in  $\Theta(n)$
  2. changing  $\alpha_i$  or  $\beta_j$ 
    - ★ update in  $\Theta(n)$  by subtracting  $2\theta$  from  $\text{slack}_j \geq 0$
- Each case occurs  $O(n)$  in a search
- This gives a total of  $\Theta(n^2)$  for each search
- And  $\Theta(n^3)$  for the whole algorithm

# Duality

- What is the Dual?

# Duality

- What is the Dual?
- Dual of Maximum unweighted matching was minimum vertex cover

# Duality

- What is the Dual?
- Dual of Maximum unweighted matching was minimum vertex cover
- Here, we have minimum weighted vertex cover. Assign nonnegative weights to vertices such that the sum of numbers at the ends of an edge is not less than the edge weight

# Duality

- What is the Dual?
- Dual of Maximum unweighted matching was minimum vertex cover
- Here, we have minimum weighted vertex cover. Assign nonnegative weights to vertices such that the sum of numbers at the ends of an edge is not less than the edge weight
- *Government Problem.* Due to technical problems, the Government wants to stop all farms and plants from working, spending minimum amount of money. It has to give some money to each farmer so that he will not farm anymore, and some to each factory so that it would not work anymore. If a farm and a plant can gain more money as the profit of processing the farm output in the plant, they will not accept government money and continue working. Obviously, the government wants to achieve the minimum cost.

Any Questions?

**THE END**