CS711008Z Algorithm Design and Analysis

Lecture 10. ASSIGNMENT, MAXIMUMMATCHING and MAXIMUMWEIGHTEDMATCHING problems [1]

Dongbo Bu

Institute of Computing Technology
Chinese Academy of Sciences, Beijing, China

[1] The slides are made based on Kuhn1955, Egerváry1931, Konig1931, von Neumann1950, Edmonds1965a/b, Munkers1957, and Berge1957.

Outline

- Assignment problem;
- MaximumMatching in bi-partite graph: alternating paths; Berge's lemma, König's theorem; Hopcroft-Karp algorithm; ILP formulation;
- MaximumWeightedMatching in bi-partite graph: Egerváry's idea; Hungarian algorithm;
- MaximumMatching in general graph: Blossom algorithm;
- MaximumWeightedMatching in general graph;

A brief history

- In 1784, G. Monge studied the ASSIGNMENT problem.
- In 1916, D. König studied the matching problem for bi-partite graphs.
- In 1931, J. Egerváry studied the weighted version of the matching problem.
- In 1950, von Neumann proved the equivalence between ASSIGNMENT problem and two players' ZERO-SUM game.
- In 1955, H. Kuhn proposed the HUNGARIAN algorithm.
- In 1957, Munkres gave a analysis of the HUNGARIAN algorithm.
- In 1964, J. Edmonds proposed the BLOSSUM algorithm for matching in general graph (together with the concept of "efficient algorithms").

A brief history

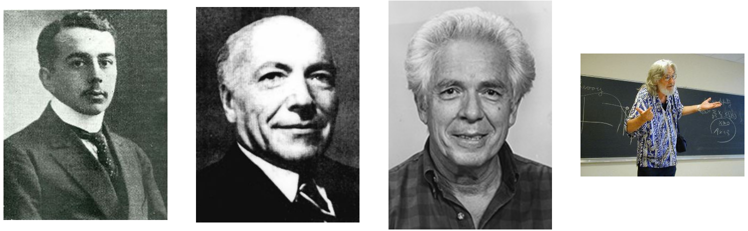


Figure: D. König, J. Egerváry, H. Kuhn, and J. Edmonds

1. ASSIGNMENT problem and MAXIMUMMATCHING in bi-partite graph

ASSIGNMENT problem in terms of matrix

- Let's consider the following ASSIGNMENT problem: there are three individuals (denoted as $I = \{I_1, I_2, I_3\}$), and three jobs $J = \{J_1, J_2, J_3\}$. Any individual might qualify one or more jobs. This information can be presented effectively by a **qualification matrix**, where $q_{ij} = 1$ if I_i qualifies J_j, and $q_{ij} = 0$ otherwise. For example,

$$Q = \begin{bmatrix} 0 & \color{red}{1} & 0 \\ \color{red}{1} & 1 & 1 \\ 0 & 1 & \color{red}{1} \end{bmatrix}$$

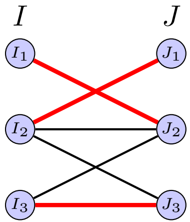
- Question: what is the largest number of jobs that can be assigned to qualified individuals (with at most one job assigned to an individual)? In other words, what is the largest number of 1's that can be chosen, with no two in the same row or column? [2]

[2] This problem was regarded as the most degenerate case of the TRANSPORATION problem, and a relative of the traveling salesman problem.

ASSIGNMENT problem in terms of graph theory

- The ASSIGNMENT problem can be described in the language of graph theory, where nodes represent individuals and jobs, and edges represent qualification.

$$Q = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$



- Thus, the ASSIGNMENT problem essentially attempts to find MAXIMUMMATCHING in bi-partite graph.

INPUT: a bipartite graph $G = (L \cup R, E)$;
OUTPUT: the **matching** M with maximum cardinality. Here, a **matching**, also known as **independent edge set**, refers to a set of **edges without common vertices**.

MaximumMatching in bi-partite graph

- We will talk about the following issues.
  - The alternating path technique
  - Berge's lemma on optimality
  - Maximum-flow approach and Hopcroft-Karp algorithm
  - Linear program formulation and dual problem
  - König's theorem: MaxMatching and MinVertexCover (and Hall's theorem)

1.1 Finding MaxMatching using alternating paths

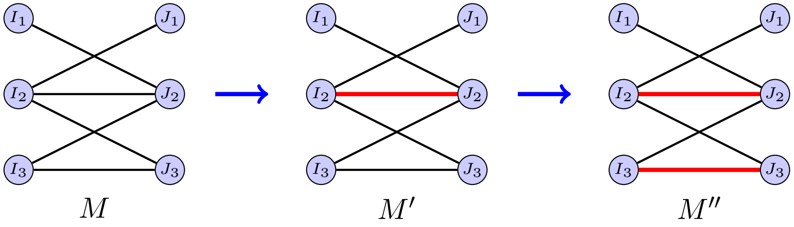
Applying the general IMPROVEMENT strategy

```
$M = M_0$; //set initial solution;
while TRUE do
    $M$ =IMPROVE($M$); //move towards optimum;
    if STOPPING($M$) then
        break;
    end if
end while
return $M$;
```

- The basic idea is to increase cardinality of matching step by step.
- It is trivial to set initial solution, e.g., a null matching. Thus the remaining are how to improve matching and when to stop (i.e., optimality condition).

How to improve a matching? A simple way

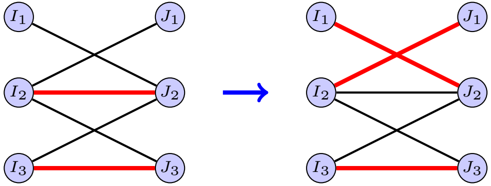
- It is clear that we can increase matching by simply placing an unassigned individual in an unassigned job that the individual qualifies.



- Note that the matching M'' cannot be further improved. In fact, M'' is a **maximal matching** (also called **complete** by H. Kuhn) rather than **maximum matching**. A set is **maximal** w.r.t. a property if it has the property but is not properly contained in any set that has this property.

A more effective improvement strategy: TRANSFER

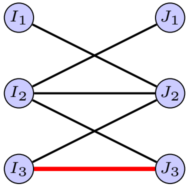
- Note that the error is the assignment of J_2 to I_2. Thus, we can move I_2 from J_2 to J_1, and assign J_1 to I_2, which increases the matching by one.
- This operation (called TRANSFER by H. Kuhn) can be implemented using **augmenting path**.



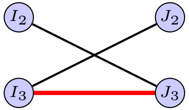
Improvement using augmenting paths

Definition (ALTERNATING PATH and AUGMENTING PATH)

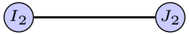
An ALTERNATING PATH w.r.t. a matching M contains edges alternatively in M and $E - M$. An alternating path starting from and ending at unmatched vertex is denoted as AUGMENTING PATH, e.g. $I_1 \rightarrow J_2 \rightarrow I_2 \rightarrow J_1$.



M



augmenting path 1



augmenting path 2

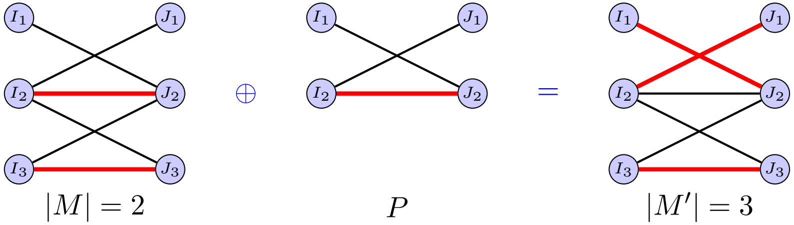
- Note that an augmenting path w.r.t. M has odd number of edges, containing one more edge in $E - M$ than that in M.

Improving matching M through calculating $M \oplus P$

- The **symmetric difference** of a matching M and an augmenting path P leads to **an improved matching**.

Lemma

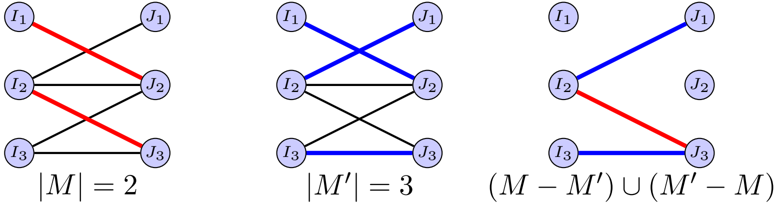
For an augmenting path P w.r.t. a matching M, the **symmetric difference** $M \oplus P = (M - P) \cup (P - M)$ *exchanges the edges in* $P \cap M$ *and those in* $P \cap \overline{M}$*, forming a new matching with one more pair, i.e.,* $|M \oplus P| = |M| + 1$.



Optimality condition Berge's lemma [1957]

Theorem

A matching M in a graph G is maximum iff there is no augmenting path w.r.t. M.



- Key idea: assume for contradiction that there is another matching M' larger than M. The **symmetric difference** $M \oplus M' = (M - M') \cup (M' - M)$ contains **alternating paths**.
- As $|M'| > |M|$, at least one **alternating path** is an **augmenting path** as it contains one more edge in M' than that in M (e.g., $J_1 - I_2 - J_3 - I_3$).

Proof.

- Assume for contradiction there is another matching M' with $|M'| > |M|$. Let's consider the resultant graph G' with edges taking from the **symmetric difference** $M \oplus M' = (M - M') \cup (M' - M)$.
- As both M and M' are matching, each node v in G' is incident with at most one edge in $M - M'$ and at most one edge in $M' - M$. Hence, G' consists of the following three types of connected components:
  1. An isolated vertex,
  2. Alternating path with distinct endpoints, or
  3. Even-length cycle with edges alternating between M and M'.
- Since M' is larger than M, there should be an alternating path with more edges from M' than M, which can be used to improve M. A contradiction.

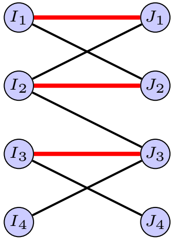
□

Note: Berge's lemma holds for general graphs. We will talk about this issue later.

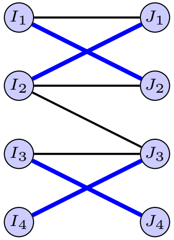
A stronger version by Hopcroft-Karp [1975]

Theorem

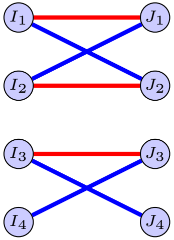
Consider two matching M and M', $|M| = r$, $|M'| = s$, and $s > r$. $M \oplus M'$ contains at least $s - r$ **vertex-disjoint augmenting paths** *w.r.t. M.*



$|M| = 2$



$|M'| = 4$



$M \oplus M'$

Proof.

- Let's consider the resultant graph G' with edges taking from the **symmetric difference** $M \oplus M' = (M - M') \cup (M' - M)$.
- As M and M' are matching, each node v in G' is incident with at most one edge in $M - M'$ and at most one edge in $M' - M$. Hence, G' consists of the following three types of **vertex-disjoint** connected components:
  1. An isolated vertex,
  2. Alternating path with distinct endpoints, or
  3. Even-length cycle with edges alternating between M and M'.
- For each component $C_i = (V_i, E_i)$, we define $\delta(C_i) = |E_i \cap M'| - |E_i \cap M|$. Then $\delta(C_i) \in \{-1, 0, 1\}$. Note that $\delta(C_i) = 1$ iff C_i is an augmenting path relative to M.
- We also have $\sum_i \delta(C_i) = s - r$.
- Hence we have at least $s - r$ augmenting paths relative to M.

□

Note that this theorem also holds even for general graphs.

Algorithm

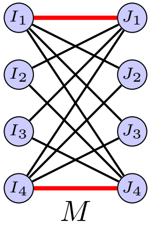
```
$\mathbf{M} = \{\}$; //set initial matching as empty set;
while TRUE do
    find an augmenting path $P$ relative to $M$;
    if no augmenting path found then
        break;
    end if
    augment $M$ using $P$ by $M = M \oplus P$;
end while
return $\mathbf{M}$;
```

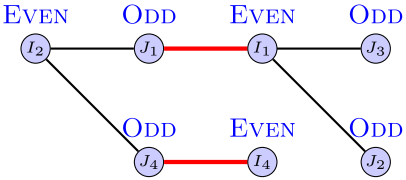
- Time complexity: $O(mn)$
- Reason:
  - $\#$WHILE $= O(n)$ as each augmentation operation increase the matching by at least 1.
  - Finding an augmenting path: $O(m)$ if using BFS.

Question: how to find augmenting paths w.r.t. a matching?

Method 1: extended BFS in the original bipartite graph

- Basic idea: starting from **an unmatched vertex r**, we try to identify **all vertices that have an alternating path to r**. If another unmatched vertex v was visited, we will obtain an augmenting path from r to v.
- These vertices can be identified using **extended BFS**: when a matched vertex is visited, a pair of edges, one from M, and the other from $E - M$, should be added.
- Similar to BFS tree, the extended BFS summarises the explored alternating paths as an **M-alternating tree** rooted at r, where each root-leaf path represents an alternating path.



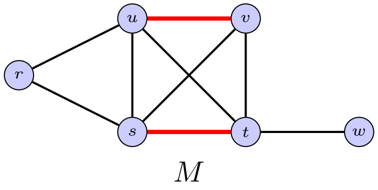


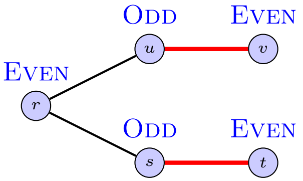
Alternating tree rooted at I_2

The extended BFS method

- The "visited" label in BFS was extended: in an alternating tree, vertices at the even levels are labeled EVEN, while vertices at odd levels are labeled ODD.
- The extended BFS works in $O(m+n)$ time as follows:
  - Initially all vertices are unlabelled.
  - The following step is repeated until all vertices are explored. At each step, we either choose an unmatched and unlabelled vertex r, label it EVEN, create a new tree rooted at r, or choose an EVEN vertex u, and explore all of its adjacent edges (u, v) as follows:
    - If v is unmatched, we report an **augmenting path** from r to v.
    - If v is matched (with w) but unlabelled, we add v, w, and edge (v, w) to the tree, and label v ODD, w EVEN.
    - If v is matched and labelled ODD, we do nothing (as this just means that **another odd-length path** from r to v was found).
    - If v is matched and labelled EVEN, we found an **odd-length cycle** (called **blossom**). Note this cannot occur in bipartite graphs.

An example: Step 1

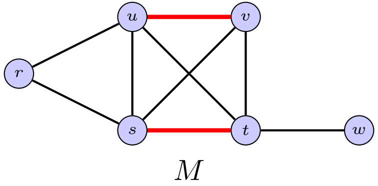


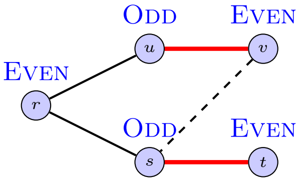


Alternating tree

- Starting from EVEN vertex r, we found an adjacent matched vertex u.
- Thus, both u and v are added to the tree with labels.
- So does another adjacent vertex s.

An example: Step 2

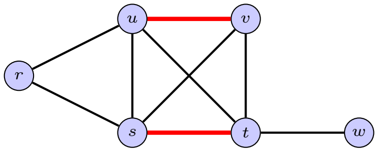




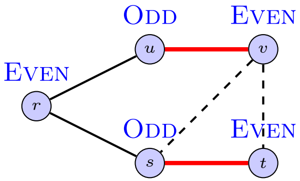
Alternating tree

- Starting from EVEN vertex v, we found an ODD vertex s.
- This just means that besides the alternating path $r - s$, another odd-length alternating path $r - u - v - s$ was found. Thus, nothing needs to do for the alternating tree.

An example: Step 3



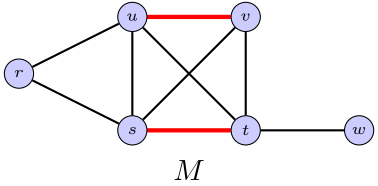
M

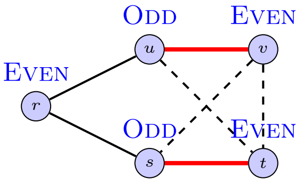


Alternating tree

- Starting from EVEN vertex v, we found an EVEN vertex t.
- We report an odd-length cycle $r - u - v - t - s - r$ (called **blossom**).

An example: Step 4

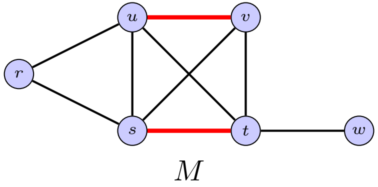


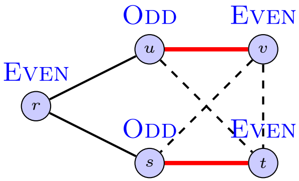


Alternating tree

- Starting from EVEN vertex t, we found an ODD vertex u.
- This just means that besides the alternating path $r - u$, another odd-length alternating path $r - s - t - u$ was found. Thus, nothing needs to do for the alternating tree.

An example: Step 5

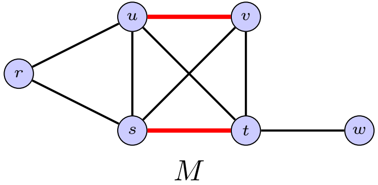


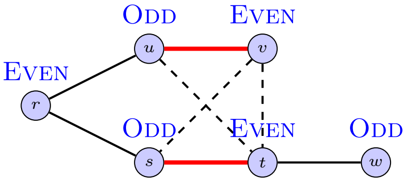


Alternating tree

- Starting from EVEN vertex t, we found an EVEN vertex v.
- We report an odd-length cycle $r - u - v - t - s - r$ (called **blossom**).

An example: Step 5



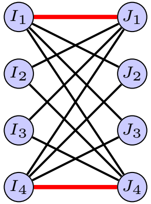


Alternating tree

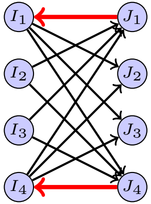
- Starting from EVEN vertex t, we found an unmatched vertex w.
- We report an augmenting path $r - s - t - w$.

Method 2: running standard BFS in a directed graph

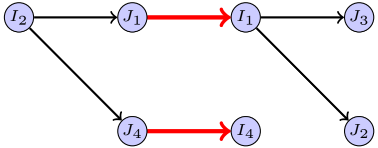
- Consider a matching M in a bi-partite graph $G = (L \cup R, E)$. A corresponding **directed graph** G_M can be constructed by assigning edges in M with direction from R to L, and the other edges in $E - M$ with direction from L to R.
- This way, the problem of finding alternating paths starting from an unmatched vertex r is transformed into finding accessible vertices in G_M, which can be accomplished using BFS in $O(m + n)$ time.



M



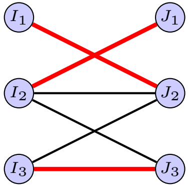
G



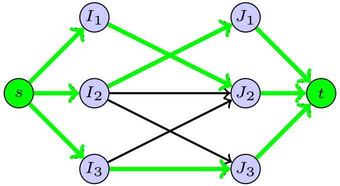
Alternating tree rooted at I_2

1.2 Finding MaximumMatching using the network-flow technique

An alternative strategy: network flow



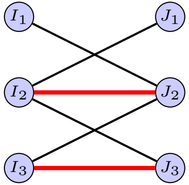
G and matching M



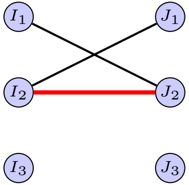
G' and flow f

- Construct a directed network G' by adding a source s and a sink t, and setting capacity as 1 for all directed edges.
- It is easy to see that any matching M corresponds to a flow f with value of $|M|$ (note that all edges have integer flow value if using Ford-Fulkerson algorithm). Conversely, an integer-flow corresponds to a matching M, as no nodes (except s and t) is incident with two edges with flow of 1.
- Thus the maximum matching can be identified by running the Ford-Fulkerson algorithm in $O(mn)$ time.

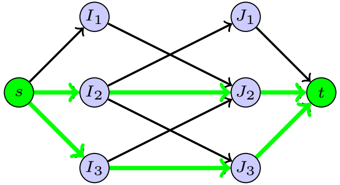
Equivalence between two versions of augmenting paths



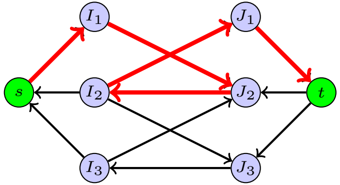
G and matching M



Augmenting path P w.r.t. M



G' and flow f



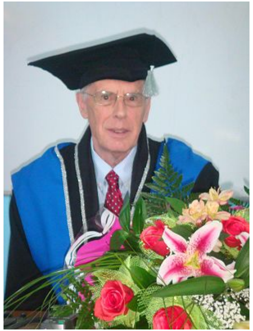
Augmenting path in residual graph G'_f

- The augmenting path $I_1 \to J_2 \to I_2 \to J_1$ w.r.t. M corresponds to the augmenting path in the residual graph G'_f.

1.3 Hopcroft-Karp algorithm: a more efficient algorithm for MaximumMatching

A more efficient algorithm: HOPCROFT-KARP algorithm

- Basic idea: Instead of finding an arbitrary augmenting path, HOPCROFT-KARP algorithm uses the **shortest augmenting path**. Consecutive augmentations using such paths can be divided into **phases**, i.e., at each phase, the shortest augmenting paths have equal length and are vertex-disjoint.
- The finding of **maximal shortest augmenting paths** is essentially what the Dinic's algorithm does.



HOPCROFT-KARP algorithm [1973]

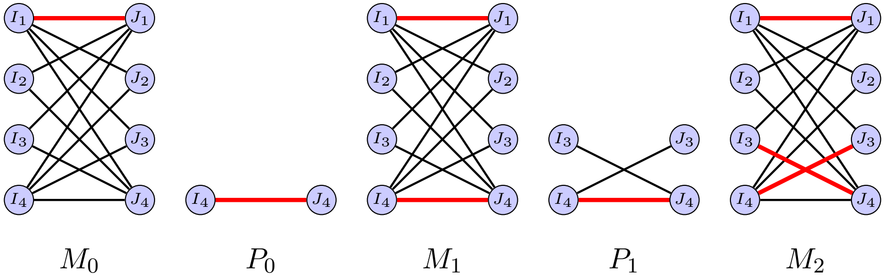
```
$\mathbf{M} = \{\}$; //set initial matching as empty set;
while TRUE do
    find the maximal shortest augmenting paths
    $P = \{P_1, P_2, ..., P_k\}$ w.r.t. to $M$;
    if no augmenting path was found then
        break;
    end if
    augment $M$ using $P$ by $M = M \oplus P_1 \oplus P_2 \oplus ... \oplus P_k$;
end while
return $\mathbf{M}$;
```

- Time complexity: $O(m\sqrt{n})$.
- Key point: instead of considering each augmenting path individually, the algorithm consider the entire phase as **blocking flow** in the DINIC'S algorithm.

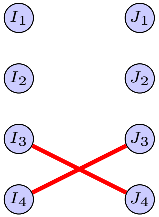
Property of consecutive augmentations

Theorem

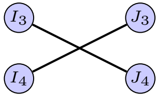
Let M_0 be a matching, P_0 a shortest augmenting path w.r.t. M_0, and P_1 an augmenting path w.r.t. $M_1 = M_0 \oplus P_0$. Then $|P_1| \geq |P_0| + |P_0 \cap P_1|$.



For example, $|P_0| = 1$, $|P_1| = 3$, and $|P_0 \cap P_1| = 1$.



$M_0 \oplus M_2$



$P_0 \oplus P_1$

Proof.

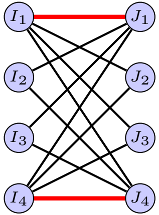
- Let's consider $P_0 \oplus P_1$. We have the following two properties:
    - $P_0 \oplus P_1 = M_0 \oplus M_2$.
    - Since $|M_2| = |M_0| + 2$, $M_0 \oplus M_2$ contains at least 2 vertex-disjoint augmenting paths w.r.t. M_0; call them p and p'. We have $|p| \geq |P_0|$, and $|p'| \geq |P_0|$, as P_0 is the shortest one.
- Thus, $|P_0 \oplus P_1| = |M_0 \oplus M_2| \geq |p| + |p'| \geq 2|P_0|$.
- Since $|P_0 \oplus P_1| = |P_0| + |P_1| - |P_0 \cap P_1|$, we have $|P_1| \geq |P_0| + |P_1 \cap P_0|$.

□

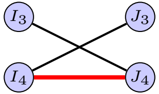
Property of consecutive augmentations (cont'd)

Theorem

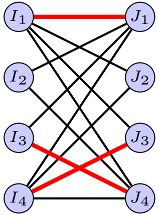
Starting with a matching M_0, *compute a sequence* $\mathbf{M_0}, \mathbf{P_0}, \mathbf{M_1}, \mathbf{P_1}, \mathbf{M_2}, \mathbf{P_2}, ...,$ *where* P_i *represents a* **shortest augmenting path** *w.r.t.* M_i, *and* $M_{i+1} = M_i \oplus P_i$. *We have* $|P_j| \geq |P_i|$ *(for* $j > i$*). If* $|P_j| = |P_i|$, P_j *and* P_i *should be* **vertex-disjoint**, *and thus both of them are augmenting paths w.r.t.* M_0.



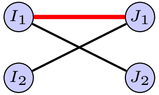
M_0



P_0



M_1

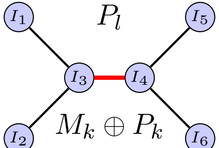


P_1

Proof.

- Suppose, for contradiction, that $|P_j| = |P_i|$ (for $j > i$), but P_j and P_i are not vertex-disjoint.
- Then there exist k and l such that $i \leq k < l \leq j$, P_k and P_l are not vertex-disjoint, and for each m, $k < m < l$, P_m is vertex-disjoint from P_k and P_l.
- Then P_l is an augmenting path w.r.t. $M_k \oplus P_k$.
- So $|P_l| \geq |P_k| + |P_k \cap P_l|$.
- But $|P_l| = |P_k|$, so $|P_l \cap P_k| = 0$, i.e., P_l and P_k has no common edge. If P_l and P_k has a common vertex v, they should have in common that edge incident with v which is in $M_k \oplus P_k$. A contradiction.

□



Time complexity of the Hopcroft-Karp algorithm

Theorem

Let s be the cardinality of the maximum matching. The **number of distinct path lengths** $|P_0|, |P_1|, |P_2|, ..., |P_{s-1}|$ *is less than or equal to* $2\lfloor\sqrt{s}\rfloor + 2$.

Proof.

- Let's divide these augmenting steps into two parts:
  - The first $r = \lfloor s - \sqrt{s} \rfloor$ steps: after r steps, the matching has size $|M_r| = r$; however, the path length increases to at most $2\sqrt{s} + 1$, since $|P_r| \leq 2\lfloor\frac{s-\sqrt{s}}{\sqrt{s}}\rfloor + 1 \leq 2\lfloor\sqrt{s}\rfloor + 1$. (Reason: For any r, we have $|P_r| \leq 2\lfloor\frac{r}{s-r}\rfloor$, since $M_s \oplus M_r$ contains at least $s - r$ vertex-disjoint (and hence edge-disjoint) augmenting paths w.r.t. M_r. Altogether these paths contains at most r edges from M_r. So one of them must have less than $\lfloor\frac{r}{s-r}\rfloor$ edges from M_r.)
  - The others $\sqrt{s}$ steps: each step increases path length at most 1.

Time complexity of the Hopcroft-Karp algorithm

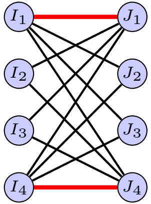
- The fact of at most $2\sqrt{s}+1$ distinct numbers in path length sequence implies that the path sequence can be divided into at most $2\sqrt{s}+1$ phases, each phase containing shortest augmenting paths with identical length.
- The following properties are immediate: the paths in a phase are vertex-disjoint, and they are augmenting paths w.r.t. the matching from which this phase begun.
- Thus, we should not regard successive augmenting steps are independent, but should concentrate on efficient implementation of **the entire phase**, i.e., finding **the maximal collection of augmenting shortest paths**.
- Note that this analysis works for general graphs.
- For bipartite graphs, **the maximal collection of augmenting shortest paths**, forming **M-alternating forest**, can be found using BFS and DFS technique in $O(m)$ time, as **layered network** identified in Dinic's algorithm.
- Thus the total cost is $O(m\sqrt{n})$ for bipartite graphs as each phase cost $O(m)$ time.

HOPCROFT-KARP algorithm: a detailed version

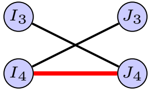
```
$\mathbf{M} = \{\}$; //set initial matching as empty set;
while TRUE do
    Constructing a layered network by running BFS over graph
    $G$, where the edges in $M$ have direction from jobs to
    individuals, while edges in $E - M$ have reverse direction
    (Note: augmenting paths become directed path in $G$);
    Finding the maximal shortest augmenting paths
    $P = \{P_1, P_2, ..., P_k\}$ w.r.t. to $M$ by running DFS guided by
    the layer network; (removing a path as soon as it was
    identified to guarantee them to be vertex-disjoint)
    if no augmenting path was found then
        break;
    end if
    Augmenting $M$ using $P$ by $M = M \oplus P_1 \oplus P_2 \oplus ... \oplus P_k$;
end while
return $\mathbf{M}$;
```

Time complexity: $O(m\sqrt{n})$.

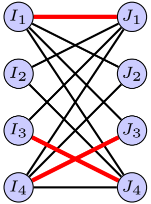
An example: finding P_2 and P_3 in phase 2



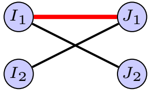
M_2



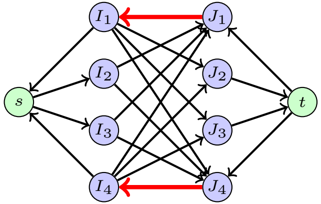
P_2



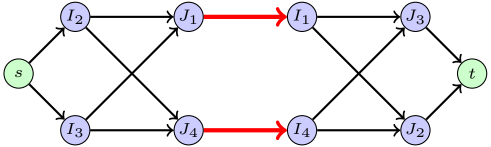
M_3



P_3



G

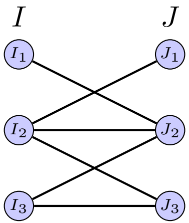


Layered network and shortest $s - t$ paths

Note: the shortest augmenting paths w.r.t. M_2 are in one-to-one correspondence with the shortest $s - t$ paths in G.

1.4 ILP formulation and its dual problem

ILP formulation for maximum-matching problem



$$
\begin{array}{lcccccccrll}
\max & x_{12} & +x_{21} & +x_{22} & +x_{23} & +x_{31} & +x_{32} & +x_{33} & & & \\
s.t. & x_{12} & & & & & & & \leq & 1 & \text{node } I_1 \\
 & & x_{21} & +x_{22} & +x_{23} & & & & \leq & 1 & \text{node } I_2 \\
 & & & & & & x_{32} & +x_{33} & \leq & 1 & \text{node } I_3 \\
 & & x_{21} & & & & & & \leq & 1 & \text{node } J_1 \\
 & x_{12} & & +x_{22} & & & +x_{32} & & \leq & 1 & \text{node } J_2 \\
 & & & & x_{23} & & & +x_{33} & \leq & 1 & \text{node } J_3 \\
 & & & & & & & x_{ij} & = & 0/1 & \\
\end{array}
$$

Totally unimodular property

- Note that the node-arc matrix is totally unimodular i.f.f. the graph is bipartite.
- Thus, by relaxing the integrity constraint to $1 \geq x_{ij} \geq 0$, we have a linear program. Its dual problem is essentially the VERTEX COVER. Here, u_i denotes the dual variable for the individual I_i, while v_i denotes the variable for the job J_i.

$$\begin{array}{lcccccccl}
\min & u_1 & +u_2 & +u_3 & +v_1 & +v_2 & +v_3 & & \\
s.t. & u_1 & & & & +v_2 & & \geq 1 & \text{edge } (I_1, J_2) \\
 & & u_2 & & +v_1 & & & \geq 1 & \text{edge } (I_2, J_1) \\
 & & u_2 & & & +v_2 & & \geq 1 & \text{edge } (I_2, J_2) \\
 & & u_2 & & & & +v_3 & \geq 1 & \text{edge } (I_2, J_3) \\
 & & & u_3 & & +v_2 & & \geq 1 & \text{edge } (I_2, J_2) \\
 & & u_2 & & & & +v_3 & \geq 1 & \text{edge } (I_2, J_3) \\
 & u_1, & u_2, & u_3, & v_1, & v_2, & v_3 & \geq 0 &
\end{array}$$

Dual problem: MinVertexCover

- Note that the node-arc matrix is totally unimodular i.f.f. the graph is bipartite.
- Thus, the dual problem is essentially the MinVertexCover problem.

$$
\begin{array}{lccccccll}
\min & u_1 & +u_2 & +u_3 & +v_1 & +v_2 & +v_3 & & \\
s.t. & u_1 & & & & +v_2 & & \geq 1 & \text{edge } (I_1, J_2) \\
 & & u_2 & & +v_1 & & & \geq 1 & \text{edge } (I_2, J_1) \\
 & & u_2 & & & +v_2 & & \geq 1 & \text{edge } (I_2, J_2) \\
 & & u_2 & & & & +v_3 & \geq 1 & \text{edge } (I_2, J_3) \\
 & & & u_3 & & +v_2 & & \geq 1 & \text{edge } (I_2, J_2) \\
 & & u_2 & & & & +v_3 & \geq 1 & \text{edge } (I_2, J_3) \\
 & u_1, & u_2, & u_3, & v_1, & v_2, & v_3 & = 0/1 &
\end{array}
$$

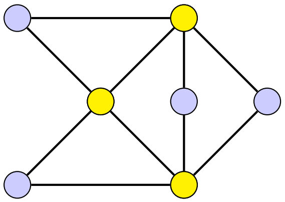
1.5 Konig's theorem: MaxMatching vs. MinVertexCover

MinVertexCover Problem

- Practical problem:
  Given n sites connected with paths, how many guards (or cameras) should be deployed on sites to surveille **all** *the paths?*

MinVertexCover problem

Input: Given a graph $G =< V, E >$,
Output: Find the minimum set of nodes $S \subseteq V$, such that each edge has at least one of its endpoints in S.



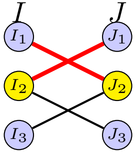
- In this example, the three nodes in yellow cover all edges.

MaxMatching vs. MinVertexCover: Konig's theorem [1931]

- Konig's theorem in terms of graph theory:

Theorem

For a bipartite graph G, the size of maximum matching equals that of the minimum vertex cover.

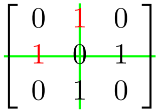


MaxMatching vs. MinVertexCover: Konig's theorem [1931]

- Konig's theorem in the language of matrix:

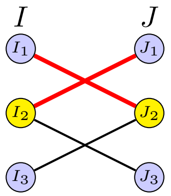
Theorem

If the elements of a matrix are 0's and 1's, then the minimum number of rows and columns that will contains all of the 1's is equal to the maximum number of 1's that can be chosen, with no two in the same row or column.

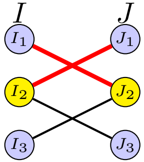


$$\begin{bmatrix} 0 & \color{red}{1} & 0 \\ \color{red}{1} & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

MaxMatching vs. MinVertexCover



- Note that MinVertexCover problem for general graphs is NP-Complete. However, when limited to bipartite graphs, MinVertexCover problem is in P.
- Actually, this theorem is a pre-linear programming example of duality.
- First, it is obvious that any vertex cover provides an upper bound for maximum matching (Reason: the edges in a matching share no common nodes; thus, to cover these edges, at least one of its ends should be selected.)



$$F = \{I_3, J_2, I_1\}$$

Proof.

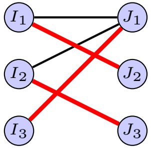
- Basic idea: For each edge in M, exactly one end will be selected. We first divide the edges in M into two categories:
  1. $M \cap F$: edges in **an alternating tree rooted at an unmatched node in I**, say (I_1, J_2). Here, F denotes **the forest** of this type of alternating trees.
  2. $M - F$: the others edges, say (I_2, J_1).
- Next, we select $C = (I - F) \cup (J \cap F)$, i.e., **right end** of edges in $M \cap F$, and **left end** of edges in $M - F$.
- C forms a vertex cover since for any edge $(u, v) \notin M$,
  - If both u and v are matched, both of them will be selected.
  - If u is unmatched, then $(u, v) \in F$, and thus v will be selected.
  - If v is unmatched, then u should be matched and thus be selected.

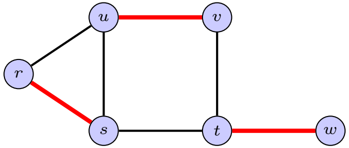
1.6 Perfect matching: Hall's theorem and Tutte's theorem

Perfect matching

Definition (Perfect matching)

In a perfect matching of a graph, each vertex is incident to an edge in the matching.





- Hall's theorem states the sufficient and necessary condition for the existence of perfect matching in bipartite graphs, and Tutte's theorem states this condition for general graphs.
- We will talk about Tutte's theorem later.

Hall's theorem [1935]

Theorem

A bipartite graph $G = (I \cup J, E)$ has a perfect matching iff for any $S \subseteq I$, $|N(S)| \geq |S|$, where $N(S)$ denotes neighbours of vertices in S.

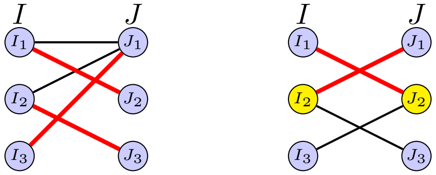
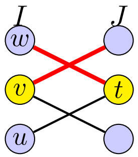


Figure: A graph with perfect matching (left side panel) and a graph without perfect matching (right side panel)



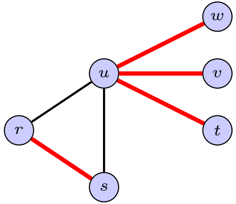
Proof.

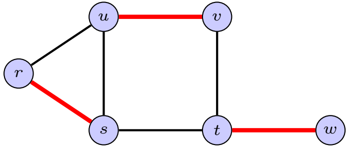
- Here we only prove the existence of a perfect matching if for any $S \subseteq I$, $|N(S)| \geq |S|$. Suppose for contradiction that the maximum matching M is not perfect, i.e., there still exists an unmatched vertex $u \in I$.
- Let F denote the alternating tree rooted at u, $S = I \cap F$, and $T = J \cap F$, e.g., $S = \{u, w\}$, and $T = \{t\}$.
- Since M is maximum matching, every job in T is matched to an individual in $S - \{u\}$, and vice versa. Thus, $|S| - 1 = |T|$.
- Note there is no edge between $S = I \cap F$ and $J - F$ (as these edges cannot be covered by $C = (I - F) \cup (J \cap F)$, making C not the minimum vertex cover).
- Hence $N(S) \subseteq T$, i.e., $|N(S)| \leq |T| = |S| - 1$. Contradiction.

1.7 MaxMatching vs. MinEdgeCover for general graphs

MinEdgeCover

INPUT: A graph $G = (V, E)$
OUTPUT: The minimum subset of edges $S \in E$ cover all nodes, i.e., every node $v \in V$ is incident to at least one edge in S.



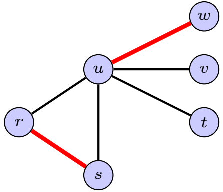


- MinVertexCover is NP-Complete for general graphs whereas MinEdgeCover is in P.

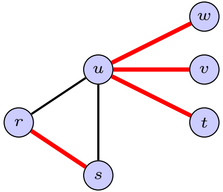
Finding MinEdgeCover through extending MaxMatching

Theorem

Consider a graph $G = (V, E)$ without isolated nodes. Its maximum matching M has size $|M| = s$ iff its minimum edge cover C has size $|C| = n - s$.

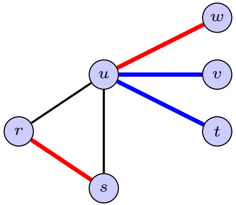


Maximum Matching
$|M| = 2$



Minimum Edge Cover
$|C| = 4$

- MinEdgeCover aims to cover all nodes using as few edges as possible. In contrast, MaxMatching aims to cover as many nodes as possible using **independent edges**.



Proof.

- Let's construct an edge cover through extending a maximum matching.
- Consider a maximum matching M with $|M| = s$. We first select all edges in M, which cover $2s$ nodes.
- Next, for each of the $n - 2s$ unmatched nodes, we arbitrarily choose one of its incident edges.
- Note that these selected edges are distinct, as there is no edge connecting any two unmatched nodes, say v and t (otherwise, adding edge (v, t) will lead to a larger matching).
- Thus, we obtain a total of $n - s$ edges, which covers all nodes.

Proof.

- On the other hand, let's construct a matching through removing edges from a minimum edge cover C with $|C| = t$.
- Initially we set $M = C$. For each node $v \in V$, let d denote the number of its incident edges in M, $d - 1$ edges are removed from M.
- Note the removal of $d - 1$ edges creates $d - 1$ unmatched nodes (w.r.t. M). (Reason: if the removal of an edge creates no unmatched node, then C is not a minimum edge cover.)
- Let y be the total number of edges removed from C. Finally M contains $t - y$ edges.
- M is a matching, as it contains at most 1 incident edge for each node.
- In addition, $|M| = t - y = n - t$. (Reason: Initially M covers all nodes, and the removal of y edges create y unmatched nodes as well $2(t - y)$ matched nodes. Thus, we have $n = y + 2(t - y) = 2t - y$.)

1.x Equivalence of Assignment problem and certain zero-sum two-person game

2. MaxWeightedMatching for bipartite graphs

Weighted ASSIGNMENT problem in terms of matrix

- Suppose 3 individuals $I = \{I_1, I_2, I_3\}$ are available for 3 jobs $J = \{J_1, J_2, J_3\}$, and a rating matrix R of positive integers is given, where r_{ij} represents the rating of individual i for job j.

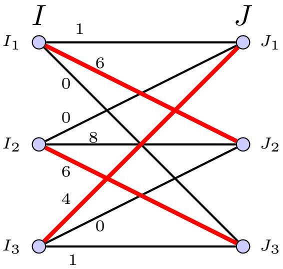
$$R = \begin{bmatrix} 1 & \textcolor{red}{6} & 0 \\ 0 & 8 & \textcolor{red}{6} \\ \textcolor{red}{4} & 0 & 1 \end{bmatrix}$$

- Question: how to assign jobs to maximise the sum of ratings (with exactly one job assigned to an individual)?
- The original paper by H. Kuhn describes the ASSIGNMENT problem as maximization of ratings. An alternative and equivalent problem statement is the minimization of costs. For example, we can set cost $c_{ij} = C - r_{ij}$, where C denotes the largest rating.

Assignment problem in terms of graph theory

- The Assignment problem can also be described in the language of graph theory, where nodes represent individuals and jobs, and edges represent ratings.

$$R = \begin{bmatrix} 1 & 6 & 0 \\ 0 & 8 & 6 \\ 4 & 0 & 1 \end{bmatrix}$$

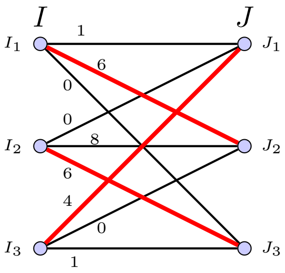


- Thus, the Assignment problem essentially attempts to find MaxWeightedMatching in bi-partite graph.

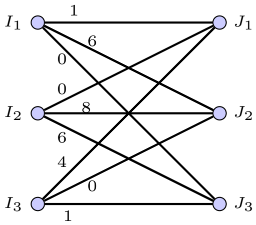
MaxWeightedMatching in bipartite graph

INPUT: a (complete) bipartite graph $G = (I \cup J, E)$, and each edge (i, j) is associated with a weight r_{ij}.
OUTPUT: a **perfect matching** M with maximum sum of weights of matched edges. Here, a **matching**, also known as **independent edge set**, refers to a set of **edges without common vertices**.

$$R = \begin{bmatrix} 1 & 6 & 0 \\ 0 & 8 & 6 \\ 4 & 0 & 1 \end{bmatrix}$$



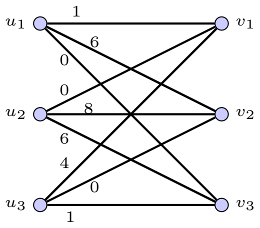
ILP formulation of MAXWEIGHTEDMATCHING problem



- Due to the weight of edges, the network-flow technique doesn't work as in the MAXMATCHING problem. Now we return back to the ILP formulation.
- Primal problem:

$$\begin{array}{rlcrcl}
\max & \sum_{i\in I}\sum_{j\in J} r_{ij}x_{ij} & & & & \\
s.t. & \sum_{i\in I} x_{ij} & = & 1 & & \text{all } j\in J \\
 & \sum_{j\in J} x_{ij} & = & 1 & & \text{all } i\in I \\
 & x_{ij} & = & 0/1 & & \text{all } i\in I, j\in J
\end{array}$$

Dual problem: MinWeightedVertexCover problem



- The coefficient matrix of primal problem is totally-unimodular; thus, we can replace $x_{ij} = 0/1$ with $1 \geq x_{ij} \geq 0$, and write its dual problem (minimum weighted vertex cover problem) as below.
- Dual problem:

$$\begin{array}{lrcrcll} \min & \sum_{i \in I} u_i & + & \sum_{j \in J} v_j & & & \\ s.t. & u_i & + & v_j & \geq & r_{ij} & \text{all } (i,j) \end{array}$$

Optimality criterion for X and (U, V)

- Primal problem:

$$\begin{array}{lrrcll} \max & \sum_{i\in I}\sum_{j\in J} & r_{ij}x_{ij} & & & \\ s.t. & \sum_{i\in I} & x_{ij} & = & 1 & \text{all } j\in J \\ & \sum_{j\in J} & x_{ij} & = & 1 & \text{all } i\in I \\ & & x_{ij} & \leq & 1 & \text{all } i\in I, j\in J \end{array}$$

- Dual problem:

$$\begin{array}{lrcrcll} \min & \sum_{i\in I} u_i & + & \sum_{j\in J} v_j & & & \\ s.t. & u_i & + & v_j & \geq & r_{ij} & \text{all edge } (i,j) \end{array}$$

- Complementary slackness: [3]

$$(u_i + v_j - r_{ij})x_{ij} = 0 \text{ all } (i,j)$$

- Optimality criterion:
  - (1) X is primal feasible;
  - (2) (U, V) is dual feasible;
  - (3) X and (U, V) satisfy complementary slackness condition.

[3] Also called **orthogonality** by M. L. Balinski and R. E. Gomory, which essentially states that primal and dual problems have identical objective value.

Two solving strategies

- Optimality criterion:
  - (1) X is primal feasible: X represents a perfect matching.

$$\begin{array}{rccll} \sum_{i\in I} & x_{ij} & = & 1 & \text{all } j \in J \\ \sum_{j\in J} & x_{ij} & = & 1 & \text{all } i \in I \\ & x_{ij} & \leq & 1 & \text{all } i \in I, j \in J \end{array}$$

  - (2) (U, V) is dual feasible:

$$u_i \quad + \quad v_j \quad \geq \quad r_{ij} \qquad \text{all edge } (i, j)$$

  - (3) X and (U, V) are orthogonal:

$$(u_i + v_j - r_{ij})x_{ij} = 0 \text{ all } (i, j)$$

- Primal method: Initialize X and (U, V) to satisfy conditions (1) and (3), and improve them to make condition (3) hold. This is what M. L. Blinski and R. E. Gomory's method does.
- Dual method: Initialize X and (U, V) to satisfy conditions (2) and (3), and improve them to make condition (1) hold. This is what the Hungarian method does.

2.1 Hungarian method: solve the dual problem

The origin of Hungarian method

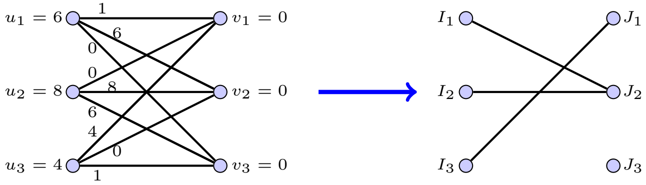
- In 1953, Harold W. Kuhn spent the summer on TSP and assignment problem at the Institute of Numerical Analysis.
- The LP formulation of the assignment problem was well known, but a 10 by 10 assignment problem has 100 variables and so was too large for both SWAC (256 words of 40 bits) and SEAC (25 variables).
- When reading Konig's book on graph theory, H. Kuhn recognised Konig's theorem to be a pre-LP example of duality.
- In a footnote, Konig referred to a paper by J. Egervary (in Hungarian) for a treatment of general case. H. Kuhn taught himself Hungarian and translated the paper, and recognised that J. Egervary gave a computationally trivial method to transform a general assignment problem into a 0-1 problem. Combining these two ideas to explore duality in an effective manner, the Hungarian method was born, which anticipates the primal-dual method.
- H. Kuhn could solve 12 by 12 problem, with pencil and paper, in no more than 2 hours.

Hungarian method: solve the dual problem

- Dual problem:

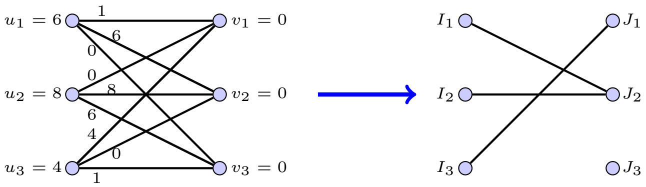
$$
\begin{array}{lrcrcll}
\min & \sum_{i\in I} u_i & + & \sum_{j\in J} v_j & & & \\
s.t. & u_i & + & v_j & \geq & r_{ij} & \text{all edge } (i,j)
\end{array}
$$

- Basic idea: Initially, it sets (U, V) to be dual feasible and sets X and (U, V) to be orthogonal, i.e., $(u_i + v_j - r_{ij})x_{ij} = 0$ for all (i, j). Next, it attempts to make X to be primal feasible, i.e., X forms a perfect matching.



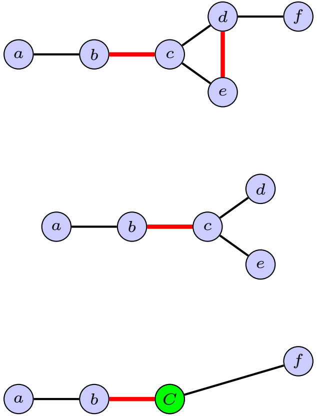
- It is trivial to find a dual feasible (U, V) and orthogonal X (set $x_{ij} = 0$ if $u_i + v_j > r_{ij}$). J. Egervary (and H. Kuhn) removed such edges and focused on the remainder graph (called **equality graph** $G_E(U, V)$).

Hungarian method: outline



```
Set $u_i = \max_j r_{ij}$;
Set $v_i = 0$;
while TRUE do
    Build the equality graph $G_E(U, V)$ with only edges $(i, j)$ if
    $u_i + v_j = r_{ij}$;
    if $G_E(U, V)$ has a perfect matching $\mathbf{M}$ then
        return $\mathbf{M}$;
    end if
    Decrease certain $u_i$ or $v_j$;
end while
```

2.2 Primal method: solve the primal problem


a
b
c
d
e
f
a
b
c
d
e
a
b
C
f