EE 382V: Social Computing	**Fall 2018**

Lecture 1: August 24

Lecturer: Vijay Garg *Scribe: Patrick Sigourney*

1.1 Introduction

Scribe: Students are required to act as scribe and take notes during one class section. Link to Google Docs sheet for scribe duty signups was emailed on Aug 24. The notes need to be formatted using LaTeX (overleaf.com is an online tool for this). The generated PDF will be sent to the class TA for review and revisions, after the student has made all necessary corrections and submitted the final .TEX and .PDF files to the TA, the notes will be posted for student use:

https://github.com/Sunny666/EE382V_Social_Computing

TA: The class TA is Sunny Bhagia (sunnysb@utexas.edu). Office hours are TBA and available either in-person or via Skype.

Assignment Teams: All assignments will be done in self-selected teams of 2, with the exception of the textbook chapter presentation, which will be done in assigned teams of 2. Links to Google Docs sheets for both assignment and presentation teams was emailed on Aug 24.

Exams: There will be 2 midterm exams and a final exam. For each exam you are allowed one 8.5"x11" page of notes, handwritten, both sides.

Textbook Assignments: Each month, 5-6 sections of the textbook will be covered by assigned student presentations. In addition to these presentations, all students will need to select 2 of these sections each month, read them, and write a half-page summary for each section. This will be handed in at the start of the first class session each month.

1.1.1 Topics for the Class

1. Matching: Buyers and sellers, pairing, stable matches.
2. Auctions: Buyer's valuation of item
3. Games: Two-player games, payoff matrix with row player & column player; prisoner's dilemma.
4. Voting: One-to-one and priority list; Arrow's Theorem.
5. Experts Algorithm: Accuracy of past predictions impacts weight applied to future predictions
6. Security: RSA public key algorithm

1.2 Matching

Bipartite graph: A graph $G = (V, E)$ is bipartite if V can be partitioned into A and B such that there

are no edges between vertices in the same partition. ie: Elements in A can only be connected to elements of B and elements in B can only be connected to elements of A.

Matching: A matching M is a subset of edges E ($M \subseteq E$) such that no two edges in M share any vertex. ie: one-to-one relationship; a pairing.

Perfect Matching: Perfect matching occurs when the vertex partitions A and B are of the same cardinality (number of elements) ($|A| == |B|$), each element of A is connected to exactly one element of B, and each element of B is connected to exactly one element of A.

Let $G = (A, B, E)$ be a bipartite graph such that $|A| == |B|$.
$M \subseteq E$ is a perfect matching if $|M| == |A|$

Maximal Matching: A matching M is maximal if it cannot be extended to include more matches.

Maximum Matching: A matching M is maximum in a graph if it has the largest cardinality possible. A maximum matching is always a maximal matching, whereas maximal matching is not always maximum matching and may have scope for improvement.

1.2.1 Hall's Marriage Theorem - Conditions for Perfect Matching (PM)

For a vertex a in set A, we define the neighboring set of a as:
$N(a) = \{b \in B | (a, b) \in E\}$
(B is a neighboring set of a which contains all b where there is an edge between a and b)

Let s be a subset of vertex set A; Then the neighboring set of subset s is defined as:
$N(s) = \bigcup_{a \in s} N(a)$
(Neighbor of subset s is the union of all neighboring sets $N(a)$ of each element a in s)

Constricted Set: A constricted set is one where there are more elements in the set s than there are elements in the associated neighbor set $N(s)$.
$\exists s : |s| > |N(s)|$
(There exists a subset s where the number of elements in s is greater than the number of elements in the neighbor set $N(s)$)

Perfect Matching is not possible with a constricted set.

Perfect Matching is possible if: $\forall s \subseteq A : |s| <= |N(s)|$
(For all subsets s in A, the cardinality of s is less than or equal to the cardinality of the neighbor set)

EE 382V: Social Computing **Fall 2018**

Lecture 1: August 24

Lecturer: Vijay Garg *Scribe: Jesus Valenzuela Ochoa*

1.1 Proof that no constricted set $\Rightarrow$ a Perfect Match (PM)

Use induction on n $|A| = |B| = n$
Base Case $n = 1$

$$a_1 \odot \leftrightarrow \odot b_1, \text{ There is } PM$$

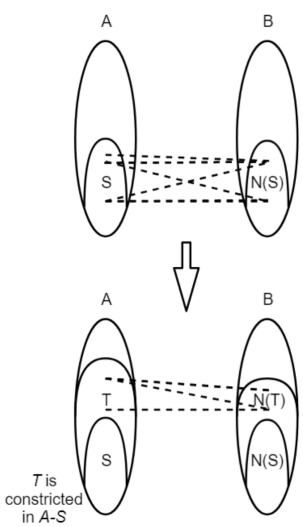
Induction Case
Assume that the theorem is true for all values above $|A| < n$

Case 1
$\exists S$ so that $|S| \equiv |N(S)|$

Case 2
$\forall S : |S| < |N(S)|$

1.1.1 Case 1.

For a selected subset S of vertices from set A, the size of a subset created with the neighbors of the elements of S, i.e. $N(S)$, equals the size of subset S.



It is needed to confirm that for the selected subsets $(S, N(S), E_s)$, the remaining subset $A - S$ does not have any constricted set when matched to subset $B - N(S)$.

This allows to iteratively break the problem into smaller parts, checking that remainder is not constricted after removing any given subset of A and its their neighbors.

If T is constricted in A-S(as shown in the lower section of Figure 1.1)
$\Rightarrow S \cup T$ is constricted in A $\Rightarrow$ **contradiction**

Figure 1.1: Splitting sets.

1.1.2 Case 2.

For a selected subset S of vertices from set A, the size of a subset created with the neighbors of the elements of S, i.e. $N(S)$, is less than the size of subset S.

Pick any vertex a in A and match to any vertex b in $N(a)$, remove both a and b from the consideration.

- Verify that there is no constriction after removal.
- Now use induction as the set size has decreased.

In implementation, you want your algorithm to make matching maxim*um* by the time it is maxim*al*.

1.2 Alternating Path

An alternating path is one that alternates between matched and non-matched edges.

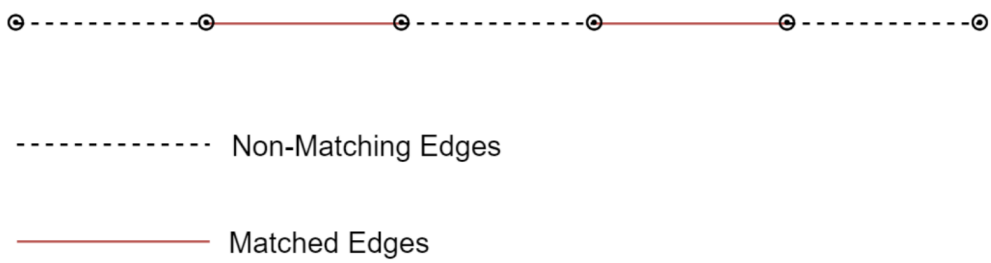


Figure 1.2: Alternating path.

1.2.1 Augmenting Path

An *augmenting path* is an alternating path starts with an exposed vertex in A and ends with and exposed vertex in B. An **exposed vertex** is one that has only non-matching edges attached to it.

If an augmenting path is found, the size of the matching can be augmented (incremented by 1) by exchanging the matched edges for the non matched edges.

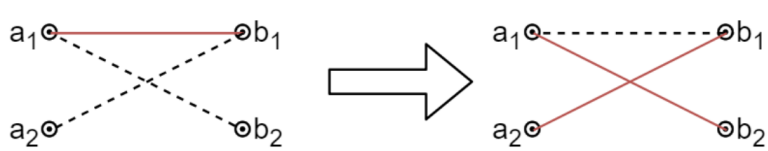


Figure 1.3: Augmenting path edge exchange.

All-edges start as unmatched. Consider using the B.F.S. algorithm to find a match. Always try to find an augmenting path.

B.F.S. Tree(with a twist)

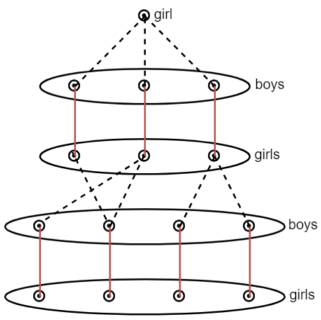


Figure 1.4: Example of a B.F.S.-Tree

This particular example has a constricted set, since there is one more girl than boys.

With B.F.S.-Tree construction, either an augmenting path is found or there is a constricted set. (By B.F.S.-Tree property).

TIP
If there is no constricted set, then you know there is an augmenting path.

Theorem 1.1 (Theorem) *A matching M is maxim*um *if and only if there are no augmenting paths with respect to M.*

Proof:

- The existence of augmenting path indicates matching is not maxim*um*, since matched and non-matching edges can be inverted, increasing matching size by 1.

- To prove that M is not maximum $\Rightarrow \exists$ an augmenting path:

Let M' be a maximum matching,

- Compute $Q = M \triangle M' \leftarrow symmetrical$ ($\triangle \equiv xor$ of M and M')

$$M \triangle M' = (M' - M) \bigcup (M - M')$$

- Since $|M'| > |M|$, Q has more edges from M' than from M.

- Each vertex is adjacent to at most one edge in $M \cap Q$ and at most one edges in $M' \cap Q$

- Q is composed of cycles(even number of edges) and paths(odd number of edges) that alternate between edges from M and M'. Cycles will always be constructed by an even number of edges since we are working with bipartite graphs and a cycle with an odd number of edges would only be achievable by matching elements of the same set. Paths found must have more edges from M' since its size is larger than that of M, hence representing the existence of an alternating augmenting path with reference to M, such as the one shown in Figure 1.5.

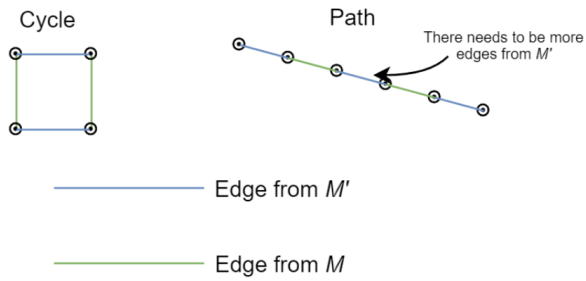


Figure 1.5: Cycle vs path construct

■

1.3 Weak Duality

Maximum matching problem with bipartite graphs:

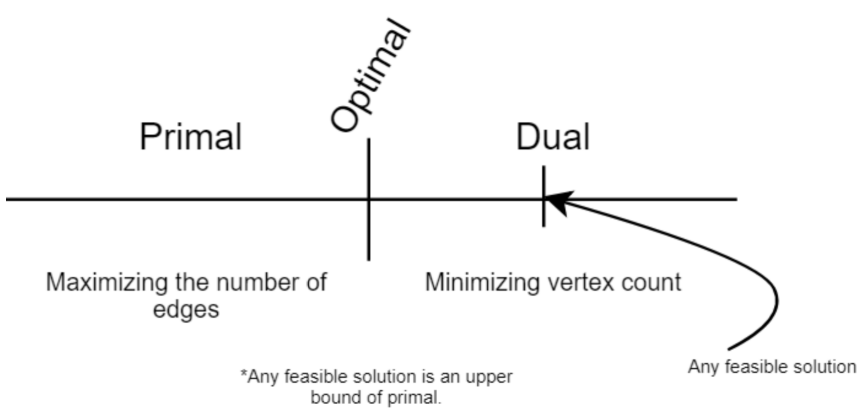


Figure 1.6: Weak duality in matching.

Let G be a bipartite graph, described as $G = (A, B, E)$. $X \subseteq A \cup B$ is a vertex cover of $G = (A, B, E)$ if every edge is adjacent to at least one vertex in X.

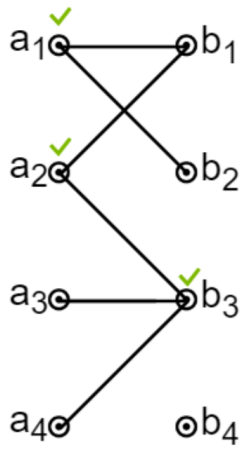


Figure 1.7: Example of a vertex cover. Analyzing what vertices combined are attached to all edges, we can establish $X = (a_1, a_2, b_3)$ for a cover of size 3.

Theorem 1.2 *Given any vertex cover X, and any matching M, $|M| \leq |X|$*

Proof: Each element of X can only have one match. All its other edges are removed once it matches. ■

Theorem 1.3 (König, Egerváry Theorem) *For any bipartite graph, the maximum size of a matching equals the size of the minimum vertex cover.*

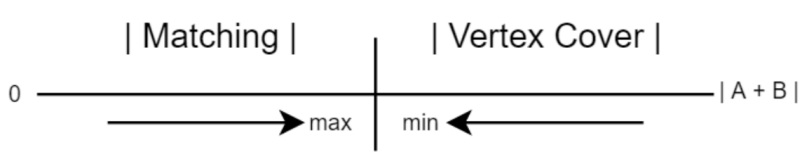


Figure 1.8: Min-Max theorem applied to weak duality in matching.

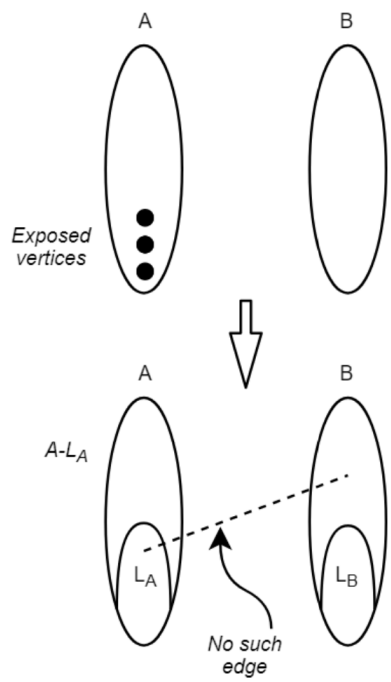


Figure 1.9: Abstraction of sets into L. There may not be any edges between L_A and $B - L$ as all the vertices in L_A are either unmatched or have categorized edges they are connected to in B into L_B, per L's definition.

Let
L: Vertices that can be reached by B.F.S.-tree from an exposed vertex.

M' : maximum matching of a bipartite graph.

$C' = (A - L_A) \cup L_B$

$L = L_A \cup L_B$
$L_A = A \cap L$
$L_B = B \cap L$

Claim

- C' is a vertex cover.
  Every edge in a bipartite graph belongs to either an alternating path, which places its b_x endpoint in $B \cap L$, or it has an a_x endpoint in $A - L_A$. If said edge is matched but not in an alternating path, then its a_x endpoint cannot be in an alternating path, since said path must have included the edge and thus belongs to $A - L_A$. If the edge is unmatched but not in an alternating path, then its a_x endpoint cannot be in an alternating path, for such a path could be extended by adding the edge to it. In consequence, C' forms a vertex cover.

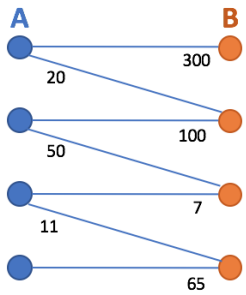
- $|C'| = |M'|$
  Every vertex in C' is an endpoint of a matched edge: every vertex in $A - L_A$ is matched because L includes all unmatched A vertices, and every vertex in $B \cap L$ must also be matched, for if there existed an alternating path to an unmatched vertex then changing the matching by removing the matched edges from this path and adding the unmatched edges in their place would increase the size of the matching. However, no matched edge can have both of its endpoints in C'. Thus, C' is a vertex cover of size equal to M', and must be a minimum vertex cover.

Scribe Notes: Introduction to Weighted Bipartite Matching
August 24th 11:25am-12:00pm

Kelsey Sandlin

1 Introduction to Weighted Bipartite Matching

A **weighted bipartite graph** is a bipartite graph in which the edges between A and B have weights assigned to them. Example:



Weighted Bipartite Graphs turn out to be more useful than bipartite graphs. The goal is to find a *perfect matching** that maximizes the weight:

$$max(\sum_{e \in M} w(e))$$

1.1 A note on *perfect matching** in weighted bipartite graphs

There was a handful of student questions regarding selecting an *imperfect* match to get a larger maximum of matched edge weights.

In a weighted bipartite graph, one can always add zero-weight edges between points in A and points in B. Therefore if the solution is not a perfect matching, we can make it one at no additional cost by matching any un-matched pairs with zero-weight edges. For the examples today, you can assume |A| = |B|.

1.2 Realistic Examples

Weighted bipartite graphs make are more useful for solving realistic problems than non-weighted bipartite graphs – you can think of set A as real estate sellers and set B as potential buyers where the edge weights are the prices of the houses. The sellers want to maximize profits, so the goal is to maximize matched edge weights:

$$max(\sum_{e \in M} w(e))$$

You could also use set A to represent Jobs and set B to represent Workers – with the assumption that each worker can only perform one job. The goal is to finish all the jobs (therefore perfect matching is required) while minimizing the amount paid to the workers (matched edge weights):

$$min(\sum_{e \in M} w(e))$$

Aside: Are the maximization and minimization problems the same problem?

Yes – you can convert from maximization to minimization (or vice-versa) by multiplying the edge weights by -1:

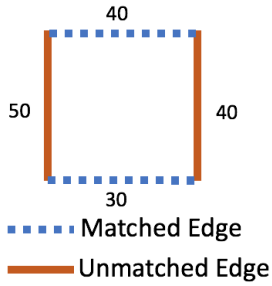
$$max(\sum_{e \in M} w(e)) = -min(\sum_{e \in M} -w(e))$$

1.3 Introduction to the Algorithm

How can we solve this problem to find the optimal perfect matching that maximizes edge weights?

- We will use some ideas about augmenting paths from the previous lecture
- Apply the Alternating Path Algorithm, but only flip the path if the weight of the flipped version is greater than the current weight

This algorithm introduces the concept of a negative cycle. A **Negative Cycle** is an alternating cycle that when flipped improves the weight. In the below example a negative cycle, you can see that if you flip, you'll get an increased total weight.



Either there is a negative cycle in the graph, or you are at an optimal solution.

Unweighted Bipartite Matching Algorithm is O(mn) where m is number of edges and n is number of vertices.

Introduction to Kuhn-Munkres Algorithm: There might exist a more efficient algorithm based on Duality. This dual algorithm will deal with minimizing labels assigned to each vertex.

EE 382V: Social Computing **Fall 2018**

Lecture 2, Session 1: August 25

Lecturer: Vijay Garg *Scribe: Matt Wey*

2.1 Introduction

In this section we will be solving the Assignment Problem specifically using the *Kuhn-Munkres* (Hungarian) Algorithm.

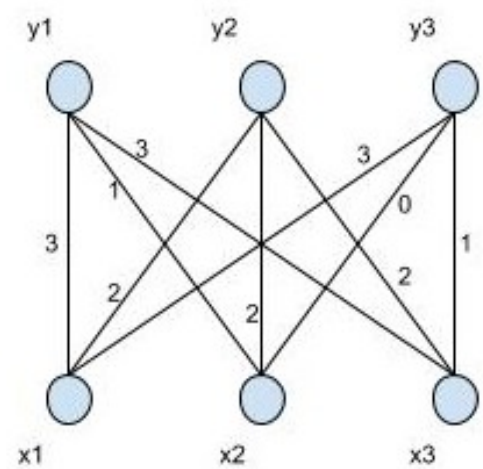


Figure 2.1: For the Assignment Problem we will consider this of bipartite graph.

2.2 Assignment Problem

The goal for the Assignment Problem is to find perfect matching with the maximum weight or the minimum cost. For our example we will be considering the bipartite graph shown in Figure 2.1. The key idea to solving this problem is duality.

2.2.1 Feasibility Constraints

Similar to assigning a weight to an edge, to add a feasibility constraint to the problem we give a label to each vertex. To be considered a feasibility constraint it must meet the following condition.

$$l(x_i) + l(y_j) \geq w(x_i, y_j); \qquad i = 1, 2, \ldots, n; j = 1, 2, \ldots, n \tag{2.1}$$

Where $l(x_i)$ is the label at vertex x_i, and $w(x_i, y_j)$ is the weight from vertex x_i to vertex y_j. The feasibility constraint is considered to create a "tight" edge when

$$l(x_i) + l(y_j) = w(x_i, y_j) \tag{2.2}$$

2.2.2 Equality Graph

When all edges in a given labeling l are tight we get an Equality Graph for that labeling, E_l, which is a subgraph of the original graph (Figure 2.1), an example of which is shown in Figure 2.2.

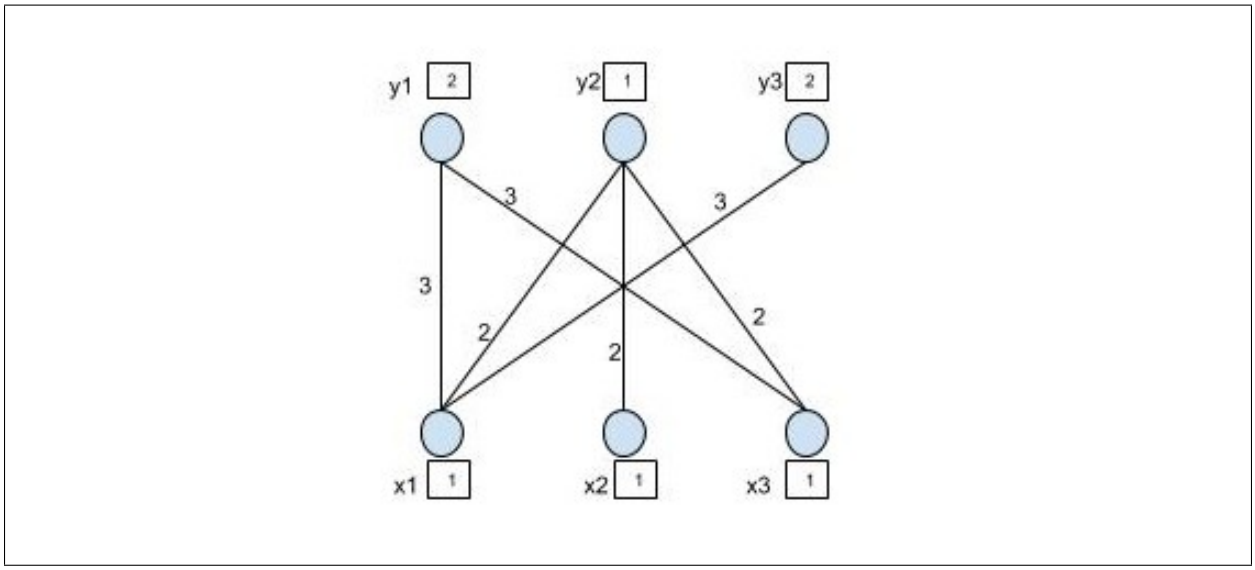


Figure 2.2: An equality graph. This particular equality graph is a subgraph of Figure 2.1.

2.2.3 Kuhn-Munkres Algorithm

As a method to solving the assignment problem we will explore the Kuhn-Munkres (Hungarian) Algorithm.

Theorem 2.1 *If l is feasible and M is a perfect matching in E_l, then M is a max-weight matching.*

Note that this method uses Equality Graphs which require tight edges so for each graph formed in the process the edges should remain tight.

Proof: Let M' be any perfect matching (not necessarily in E_l)

$$w(M') = \sum_{e \in E} w(e) \leq \sum_{e \in M'} l(e_x) + l(e_y) = \sum_{v \in V} l(v)$$

M is a perfect matching in E_l

$$w(M) = \sum_{e \in M} w(e) = \sum_{v \in V} l(v)$$

and finally

$$w(M') \leq w(M)$$

■

We can use Theorem 2.1 in creating the algorithm that will solve the Assignment Problem. It is important to note that the input for our algorithm is a bipartite graph with the connected edges and the weights for the edges. Any non-existent edges can also be considered edges zero weight.

Invariants

l is feasible;

M is some matching in E_l, where E_l is an equality graph with labeling l

The following pseudocode explains how our algorithm will solve the Assignment Problem.

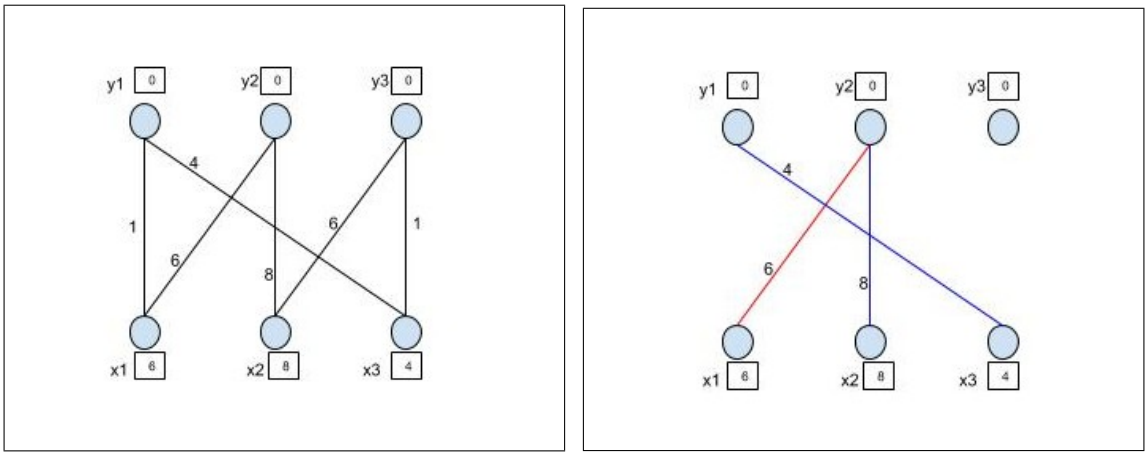
```
while(|M| < n) {
  - if there is an augmenting path in E_l
    - use the augmenting path to augment and increase the size of M
  - else
    - improve l to l' such that E_l exists in E_l'
}
```

Notice each iteration through the while loop will each increase the size of M OR change E_l to include an additional augmenting path. Note that $N_l(S)$ represents the neighbors of set S in labeling the l. The steps of the algorithm are as follows:

1. Initialization: l is initial labeling, $M = \{\}$
2. if M is a perfect matching stop. Otherwise pick a free vertex u such that $u \in X$.
   Assign $S = \{u\}, T = \emptyset$
3. if $N_l(S) = T$
$$\alpha_l = M_{x \in S, y \notin T}(l(x) + l(y) - w(x, y))$$
$$l'(v) = \begin{cases} l(v) - \alpha_l; & v \in S \\ l(v) + \alpha_l; & v \in T \\ l(v); & otherwise \end{cases}$$
4. if $N_l(S) \neq T$ pick $y \in (N_l(S) - T)$
   if y is free then u to y is an augmenting path.
   Augment path and go back to 2.
   if y is match to z
   $S = S \cup \{z\}, T = T \cup \{y\}$. Go to 3.

2.2.4 Hungarian Algorithm Example

- Given a weighted graph, create a basic initialization labeling all the vertices with the max weight of its connected edges as shown in Figure 2.3 (a).
$$\forall y_j : \quad l(y_j) = 0, \quad l(x_i) = \max(w(x_i, y_j))$$
- Select an initial matching as shown in Figure 2.3 (b).



(a) Initialized labeled graph with trivial labeling

(b) Initial equality graph E_l

Figure 2.3

- Assign $S = \{x_1\}, T = \emptyset$. Since $N_l(S) \neq T$ to go step 4.

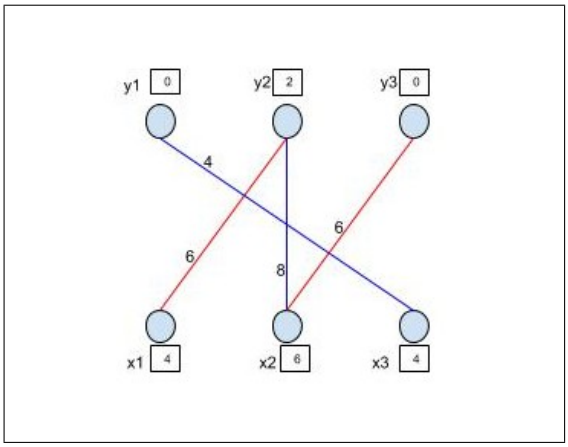
- Assign $y \in (N_l(S) - T)$. We choose y_2, which is matched, and increase the graph by assigning $S = \{x_1\} \cup \{x_2\}, T = \emptyset \cup \{y_2\}$. Go to 3.

- Now $S = \{x_1, x_2\}, T = \{y_2\}, N_l(S) = T$. Calculate α_l

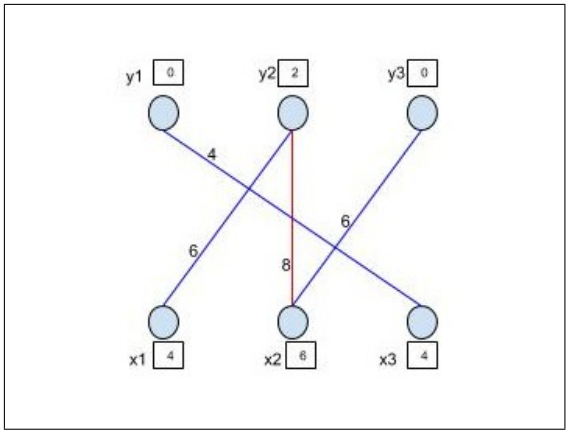
$$\alpha_l = min \begin{cases} (x_1, y_1) & 6 + 0 - 1 \\ (x_1, y_3) & 6 + 0 - 0 \\ (x_2, y_1) & 8 + 0 - 0 \\ (x_2, y_3) & 8 + 0 - 6 \end{cases} = 2$$

- Update Labels by reducing labels of S by 2 and increasing labels of T by 2 and add edge (x_2, y_3) to the equality graph. See Figure 2.4 (a).

- We now have a path starting and ending with unselected edges, so we augment the path and add the updated equality graph to the matching M as shown in Figure 2.4(b).



(a) Equality graph updated to include augmented path.



(b) Graph showing the final optimal matching with total weight 16.

Figure 2.4

- Now that we have complete vertex coverage (perfect matching), by Theorem 2.1 we know we have an optimal (max-weight) solution and can calculate the final value by summing the selected edges or summing the vertex labels.

EE 382V: Social Computing	**Fall 2018**

Lecture 6: August 25

Lecturer: Vijay Garg — *Scribe: Ari Bruck*

6.1 Demange, Gale, and Sotomayor aka `DGS/Auction` Algorithm

After the discussion of the KuhnMunkres algorithm, the professor introduced a new algorithm. The algorithm will provide the best possible assignment of goods and bidders such that the prices are maximized.

Input :`Bipartite graph with non-negative integer weights on the edges`,
`B` : set of Bidders,
`G` : set of Goods,
$\mathtt{W}_{i,j}$: weight between bidder i and good j
Output :`Maximum weight matching`

Q :Queue of Bidders
P_j :Price of Good j
$Owner_j$:Current winning bidder of Good j
n_m :Size of the matching (In the example in class $n_g \leq n_b$ [more bidders than goods] so $n_m = n_g$)
δ :The incremental price increase in auction price as a result of a matching

Algorithm 1 DGS/Auction

```
for all goods j do
    P_j ← 0                                  ▷ Each good's price starts at 0
    Owner_j ← NULL                           ▷ bidders do not own any good
Q ← B                                        ▷ all bidders are in the queue
n_m ← min(n_g, n_b)
δ ← 1/(n_m + 1)

while Q ≠ ∅ do
    i ← Q.dequeue()
    find j that maximizes W_{i,j} − P_j      ▷ Find the good that has best "effective" payoff
    if W_{i,j} − P_j ≥ 0 then                ▷ If the good adds to the overall weight matching
        Q.enqueue(Owner_j)                   ▷ Replace current owner
        Owner_j ← i                          ▷ with new one
        P_j ← P_j + δ                        ▷ Increase the auction price for that good
return (j, Owner_j)∀j.  ▷ maximum weight matching has been found, return all goods and their owners
```

<u>Correctness:</u> The proof of correctness is based on showing that the algorithm gets into an equilibrium, a situation where all bidders "are happy".

Definition: Bidder i is δ-happy with respect to P if $\exists$ good j s.t. **either**:

$Owner_j = i$
AND
$\forall$ goods j' : $\delta + W_{i,j} - P_j \geq W_{i,j'} - P_{j'}$

OR

$Owner_j \neq i$
AND
$\forall$ goods j : $W_{i,j} \leq P_j$

Loop Invariant: $\forall$ bidders $\not\subset Q$ are δ-happy

Initially: `TRUE`
Q is initialized to all bidders

At Runtime: `TRUE`
For the bidder i dequeued in an iteration, the loop exactly chooses the j that makes him happy, if such j exists, and the δ-error is due to the final increase in P_j.
Therefore this iteration cannot hurt the invariant for any other i': any increase in P_j for j that is not owned by i' does not hurt the inequality while an increase for the j that was owned by i' immediately enqueues i'.

The running time analysis above implies that the algorithm terminates, at which point Q must be empty and thus all bidders must be δ-happy.

■

6.1.1 δ-Happy Bidders

Claim: If all bidders are δ-happy then for every matching M'

$$n\delta + \sum_{i=owner_j} W_{i,j} \geq \sum_{(i,j)\in M'} W_{i,j}$$

Proof: Fix a bidder i and assume i receives item j by this algorithm such that:

$\delta + W_{i,j} - P_j \geq W_{i,j'} - P_{j'}$ where j' is item received in M'

$$\text{Sum over all } i = \sum_{i=owner_j} (\delta + W_{i,j} - P_j) \geq \sum_{i,j'} (W_{i,j'} - P_{j'})$$
$$= n\delta + \sum_{i=owner_j} (W_{i,j} - P_j) \geq \sum_{i,j'} (W_{i,j'} - P_{j'})$$

Since both the algorithm and M' give matchings, each j appears at most once on the left hand side and at most once on the right hand side.
Moreover, if some j does not appear on the left hand side then it was never picked by the algorithm and thus $P_j = 0$. Thus when we subtract $\sum_j P_j$ from both sides of the inequality, the LHS becomes the LHS

of the inequality in the lemma and the RHS becomes at most the RHS of the inequality in the lemma such that

$$= n\delta + \sum_{i=owner_j} W_{i,j} \geq \sum_{i,j'} W_{i,j'}$$

■

Some things to note: In the algorithm, either some bidder gets out of Q or the price of good j goes up by δ. Since all weights (prices) $\in \mathbb{Z}$ (rational prices can be scaled to integers), then $n\delta < 1$.

No P_j can ever increase once its value is above $C = max_{i,j} W_{i,j}$. It follows that the total number of iterations of the main loop is at most $Cn/\delta = O(Cn^2)$ where n is the total number of vertices (goods+bidders).
Each loop can be trivially implemented in $O(n)$ time, giving total running time of $O(Cn^3)$, which for the unweighted case, $C = 1$, matches the running time of the basic alternating paths algorithm on dense graphs.

References

[] Noam Nisan, Auction Algorithm for Bipartite Matching, *Turing's Invisible Hand: Computation, Economics, and Game Theory* (July 2009), https://agtb.wordpress.com/2009/07/13/auction-algorithm-for-bipartite-matching/

EE 382V: Social Computing	**Fall 2018**

Lecture 2: August 25

Lecturer: Vijay Garg — *Scribe: Javier Palomares*

2.1 Linear Programming

Linear Programming involves finding the maximum possible value of some objective subject to some constraints. For example, we might want to maximize the harvest value of some plot of land where we can plant different crops. In linear programming models these type of problems are modeled as a set of decision variables
$\{x_1, x_2, \ldots, x_n\}$
and an objective function
$c_1x_1 + c_2x_2 + \ldots + c_nx_n$
subject to some constraints
$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1$
$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq b_2$
$\vdots$
$a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq b_m$
$x_1, x_2, \ldots, x_n \geq 0$
Linear programming only deals with problems that are linear with respect to the decision variables in the objective function and constraints.

2.1.1 Example of Linear Programming

Maximize the objective function
$x_1 + x_2$
subject to
$2x_1 + x_2 \leq 1$
$x_1 + 2x_2 \leq 1$
$x_1, x_2 \geq 0$

To solve the problem we need to find the point that maximizes the objective function within the union of the spaces defined by each of the constraints. This point is called the optimal solution, and the value the objective function takes at this space is called the optimal value. The space defined by the set of constraints is called the feasible space.

Figure 2.1 shows the feasible space defined by the constraints. Possible values of the objective function correspond to the level sets of $x_1 + x_2$. Since the objective function grows as x_1 and x_2 grow, level sets up and to the right corresponding to a bigger value in the objective function. It turns out that the corner of the feasible space at $(x_1, x_2) = (1/3, 1/3)$ is the optimal point.

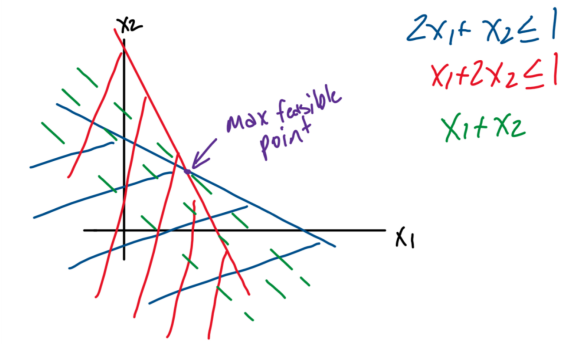


Figure 2.1: Finding optimal solution in the feasible space

2.1.2 Solution at corner points

A lot of Linear Programming was formulated by George Dantzig who found that optimal points are always at the corners of the feasible space. He formulated this into the Simplex algorithm.

2.2 Duality

It turns out that every linear problem has an associated problem called the simplex. If the linear problem (called the primal) is a maximization problem, the dual is a minimization problem, and solutions to the dual problem are an upper limit on the optimal value to the primal problem. If the primal is a minimization problem, then the dual is a maximization problem. Moreover, duality also states that the optimal value of the dual problem equals the optimal value of the primal.

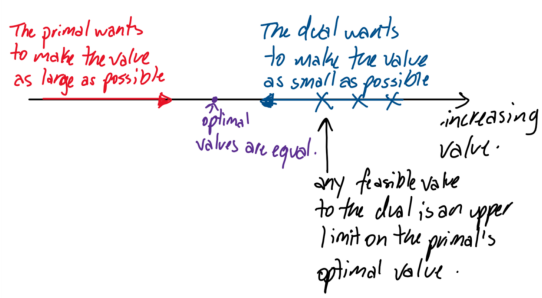


Figure 2.2: The dual places an upper limit on the primal's optimal value

2.2.1 Duality in practice

If we can find a vertex cover for a bipartite graph of size n, then by duality, we can prove that there then there cannot exist a matching of size bigger than n. The minimization problem (finding a vertex cover) places an upper bound on the maximization problem (cardinality of the matching).

2.2.2 Duality example

minimize
$7x_1 + x_2 + 5x_3$
subject to
$x_1 - x_2 + 3x_3 \geq 10$
$5x_1 + 2x_2 - x_3 \geq 6$
$x_1, x_2, x_3 \geq 0$

The dual problem is

maximize
$10p_1 + 6p_2$
subject to
$p_1 + 5p_2 \leq 7$
$-p_1 + 2p_2 \leq 1$
$3p_1 - p_2 \leq 5$
$p_1, p_2, p_3 \geq 0$

Duality tells us that if we find *any* solution to the dual problem, even if it's not the optimal solution, with a value V, then the optimal value of the primal cannot be less than V.

2.2.3 Max-Flow/Min-Cut

In a max flow, the goal is to maximize the flow from a source node to a destination node subject to the constrained capacity of edges between intermediate nodes. Additionally, flow must be conserved, so the flow into an intermediate node is constrained to equal the flow out of a node.

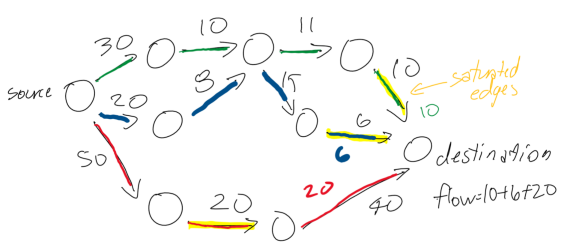


Figure 2.3: Network flow from a source to a destination along edges with capacity

In this example, we see that the edges with capacity 10, 6, and 20 become saturated and constraint the flow from the source to the destination. It turns out that these edges form a min s-t cut according to the min max theorem. In a flow network, a s-t cut is a set of edges that when removed isolates the source and destination. The min s-t cut has a minimum value of weight associated with the set of edges. So if we find the min s-t cut, the weight of the edges is equal to the max flow from the source to the destination.

2.2.3.1 Min-Max Theorem

The max-flow min-cut theorem states that in a flow network, the amount of maximum flow is equal to capacity of the minimum cut.

It turns out that matching problems can be modeled as a max flow problem by adding dummy source and destination nodes, and edges with capacity 1 from the source to the girls and from the boys to the destination, and of infinite capacity between boys and girls.

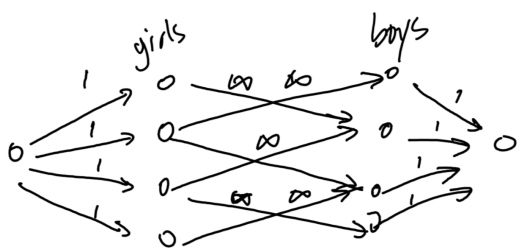


Figure 2.4: Matching Problem as a network flow

2.3 Appendix

In this section, we'll write the primal using vectors and arrays. Note that this was not covered in lecture, and is here just for extra information

We can rewrite the objective function as
$\mathbf{c}'\mathbf{x}$
where $\mathbf{x}$ is the vector of decision variables and $\mathbf{c}$ is the vector of coefficients in the objective function.
Bold face indicates vectors, and $\mathbf{a}'\mathbf{b}$ is the transpose of $\mathbf{a}$ times $\mathbf{b}$.
Then the coefficients of the constraints form the matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & & \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

with rows $\mathbf{a}_i$ and columns $\mathbf{A}_j$
So the primal can be rewritten into the following structure
minimize
$\mathbf{c}'\mathbf{x}$
subject to
$\mathbf{a}_i'\mathbf{x} \geq b_i \in M_1$
$\mathbf{a}_i'\mathbf{x} \leq b_i, i \in M_2,$
$\mathbf{a}_i'\mathbf{x} = b_i, i \in M_3,$
$x_j \geq 0, j \in N_1$
$x_j \leq 0, j \in N_2$
x_j free, $j \in N_3$

The dual then has the structure
maximize
$\mathbf{p}'\mathbf{b}$
subject to
$p_i \geq 0, i \in M_1$
$p_i \leq 0, i \in M_2$
p_i free, $i \in M_3$
$\mathbf{p}'\mathbf{A}_j \leq c_j, j \in N_1$
$\mathbf{p}'\mathbf{A}_j \geq c_j, j \in M_2,$
$\mathbf{p}'\mathbf{A}_j = c_j, j \in N_3,$

EE 382V: Social Computing	**Fall 2018**

Lecture 2: August 25

Lecturer: Vijay Garg — *Scribe: Swapna M Mukrappilly*

2.1 Posets (Partially Ordered Sets)

A partially ordered set is a set P equipped with a binary relation $\leq$ which is a partial order on X.

$$P = (X, \leq)$$

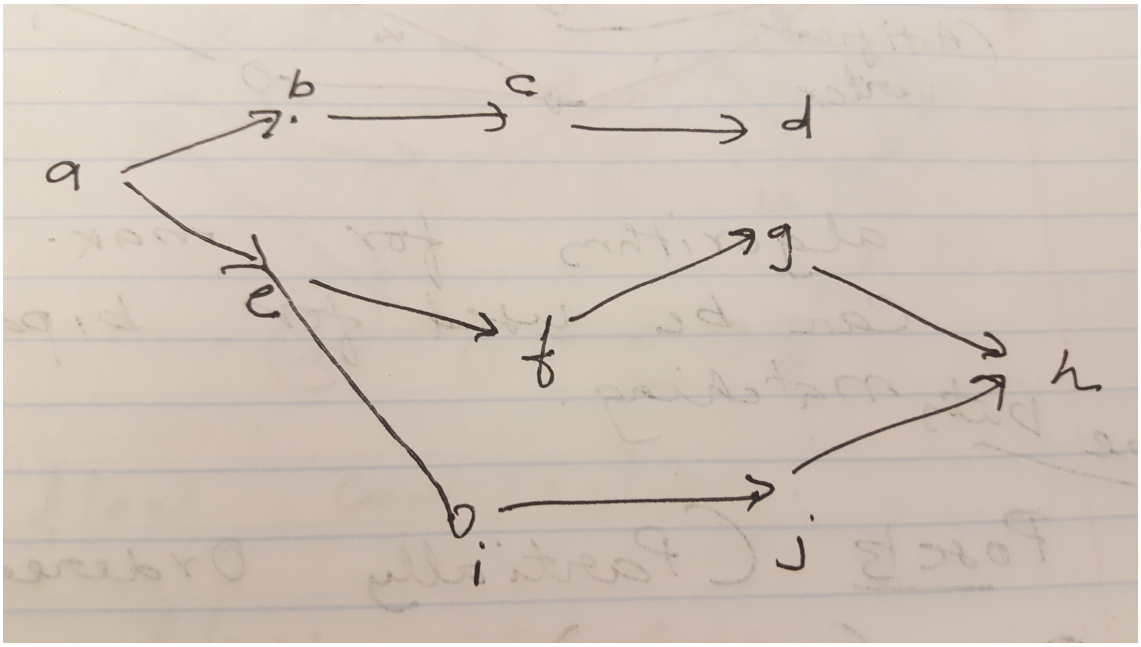
X: Any set.

The relation $\leq$ is **reflexive**, **transitive**, **antisymmetric** relation.

For all a,b,c $\in \mathbb{X}$ it must satisfy,

- a $\leq$ a (reflexivity: every element is related to itself)
- If a $\leq$ b and b $\leq$ a, then a = b (antisymmetry: two distinct elements cannot be related in both directions)
- If a $\leq$ b and b $\leq$ c, then a $\leq$ c (transitivity: if a first element is related to second element and that element is related to a third element, then the first element is related to the third element)

2.2 Chains and Antichains



$C \subseteq X$ is a chain if,

$$\forall x, y \in C, \quad (x \leq y) \quad or \quad (y \leq x)$$

In the above figure, {a,b,c,d} is a chain.

$A \subseteq X$ is an antichain if,

$$\forall x, y \in A, \quad (x \not\leq y) \wedge (y \not\leq x)$$

In the above figure, {c,f,i} is an antichain.

2.3 Dilworth's Theorem

Dilworth's theorem, also called min-max theorem states that,

Minimum number of chains required to cover the poset = maximum sized antichain.

2.4 Answers to problems on Matching

Exercise 1-1

There exists an augmenting path in G with respect to M iff there exists a directed path in D between an exposed vertex in A and an exposed vertex in B

Prove the above claim

Answer:

Proof:-

Let us consider an a maximal matching M and a maximum matching M*. The symmetric difference D is a subgraph between M and M∗, $D = M \bigoplus M*$

As both M and M∗ are matchings, each vertex in D can have degree not greater than 2, because it can be adjacent to at most one edge from each matching. If a vertex has degree equal to 2, then it is adjacent to one edge from M and one edge from M∗. Therefore, all the cycles and paths are alternating but cycles can only be even-length.

As |M∗| > |M| and $D = M * \bigoplus M = (M * \cup M) \backslash (M * \cap M)$ it is also true that $|D \cap M *| > |D \cap M|$, meaning that there are more edges from M* than M in D. Since cycles can only be even length, the augmenting path can only be odd-length paths that start and end with edges from M∗. Hence an augmenting path has to be a directed path and not a cycle.

If the vertices in A and B were not exposed, there would already been an edge between A and B resulting in a non-disjoint set.

Thus we can prove that an augmenting path can exist if and only if there is a directed path between an exposed vertex in A and an exposed vertex in B

Exercise 1-2

An edge cover of a graph G = (V, E) is a subset of R of E such that every vertex of V is incident to at least one edge in R. Let G be a bipartite graph with no isolated vertex. Show that the cardinality of the minimum edge cover R∗ of G is equal to |V | minus the cardinality of the maximum matching M∗ of G. Is this true also for non-bipartite graphs?

Answer:

Let $\nu(G)$ be the size of the maximum matching M*.

Let n =|V| be the number of vertices in the graph G.

We will construct an edge cover as follows,

A maximum matching M* covers $2\nu(G)$ vertices.Since there are no isolated vertices, the vertices that are left uncovered $= n - 2\nu(G)$.

If we use one edge to cover one vertex, the total size of the edge cover |R*|, $\rho(G)$

$$\leq \nu(G) + n - 2\nu(G)$$
$$\leq n - \nu(G)$$
$$\rho(G) + \nu(G) \leq n \qquad .....(1)$$

Let R* be the minimum edge cover. Since removing any edge from R* yields an uncovered vertex, each edge has an endpoint of degree 1 in (V, R*)

This implies that (V, R*) consists of disjoint sets. The number of components in (V, R*)

$$= |V| - |R*|$$
$$= n - \rho(G)$$

Now we construct a matching M of size $n - \rho(G)$ by choosing one edge from each disjoint set in (V, R*)

Maximum matching $|M*| \geq n - \rho(G)$

$$\nu(G) \geq n - \rho(G)$$
$$\nu(G) + \rho(G) \geq n \qquad .....(2)$$

From (1) and (2) we get,

$$\nu(G) + \rho(G) = n$$

The result is true for non-bipartite graphs too.

Exercise 1-3

Show that in any graph G= (V,E) (not necessarily bipartite), the size of any maximal matching is at least half the size of a maximum matching M*

Answer:

Consider a maximum matching M* of size $\alpha(G)$. For every edge in M*, at least one endpoint of the edge must be present in any maximal matching(or the edge can be added to the matching, and hence it won't be maximal). Hence, the number of vertices covered by any maximal matching must at least be $|M*| = \alpha(G)$. Since an edge in the matching can cover at most two vertices, the number of edges in the maximal matching must be $\alpha(G)/2$.