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 $n = 1$

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

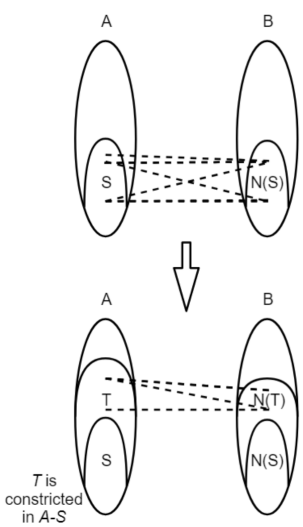
Induction Case
Assume that the theorem is true for all values above $|A| < n$

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.

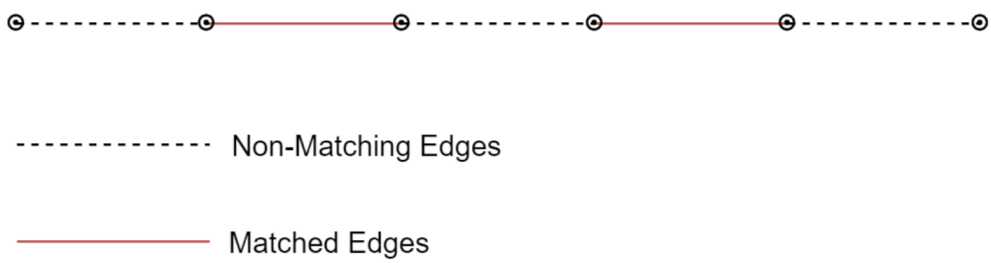


Figure 1.2: Alternating path.

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.

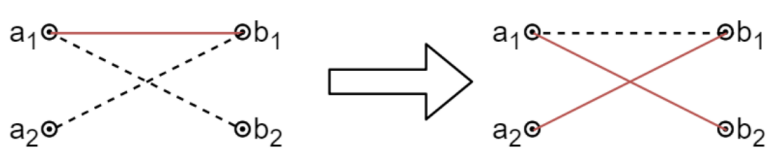


Figure 1.3: Augmenting path edge exchange.

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)

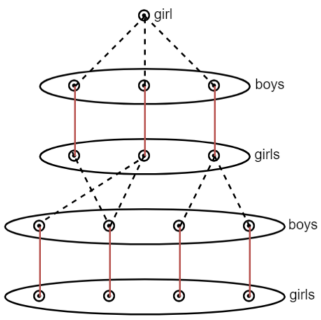


Figure 1.4: Example of a B.F.S.-Tree

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.

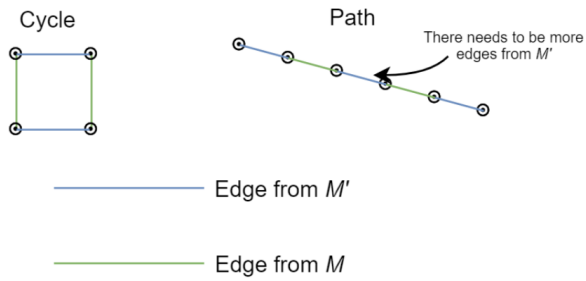


Figure 1.5: Cycle vs path construct

■

1.3 Weak Duality

Maximum matching problem with bipartite graphs:

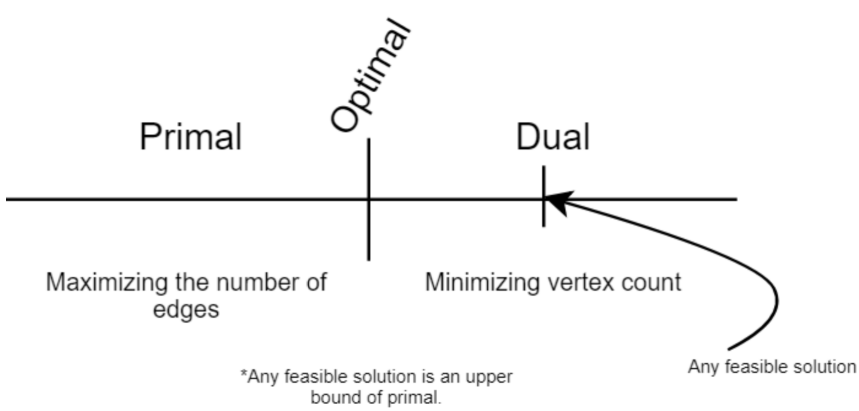


Figure 1.6: Weak duality in matching.

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.

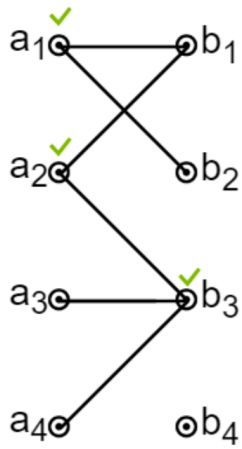


Figure 1.7: Example of a vertex cover. Analyzing what vertices combined are attached to all edges, we can establish $X = (a_1, a_2, b_3)$ for a cover of size 3.

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.*

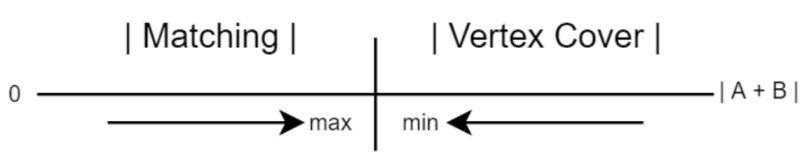


Figure 1.8: Min-Max theorem applied to weak duality in matching.

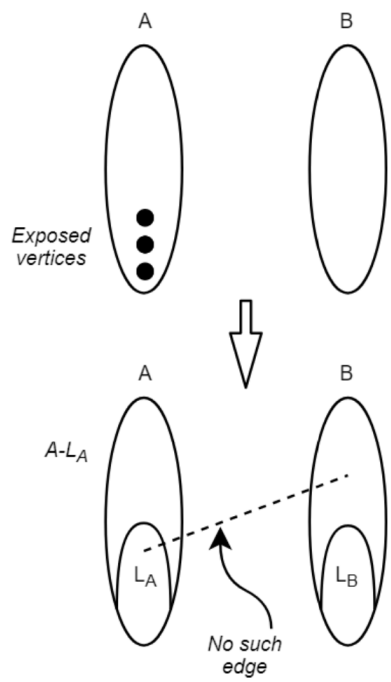


Figure 1.9: Abstraction of sets into L. There may not be any edges between L_A and $B - L$ as all the vertices in L_A are either unmatched or have categorized edges they are connected to in B into L_B, per L's definition.

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.

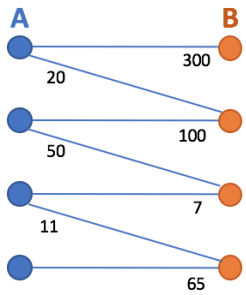
- $|C'| = |M'|$
  Every vertex in C' is an endpoint of a matched edge: every vertex in $A - L_A$ is matched because L includes all unmatched A vertices, and every vertex in $B \cap L$ must also be matched, for if there existed an alternating path to an unmatched vertex then changing the matching by removing 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:

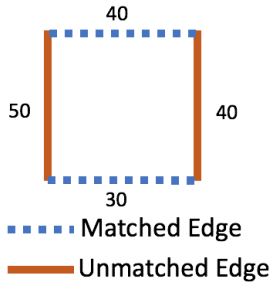
$$max(\sum_{e \in M} w(e)) = -min(\sum_{e \in M} -w(e))$$

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.

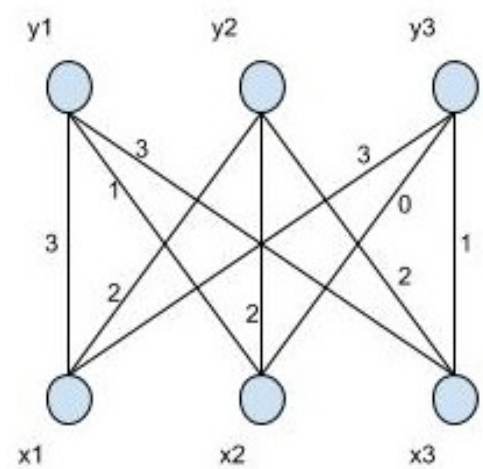


Figure 2.1: For the Assignment Problem we will consider this of bipartite graph.

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.

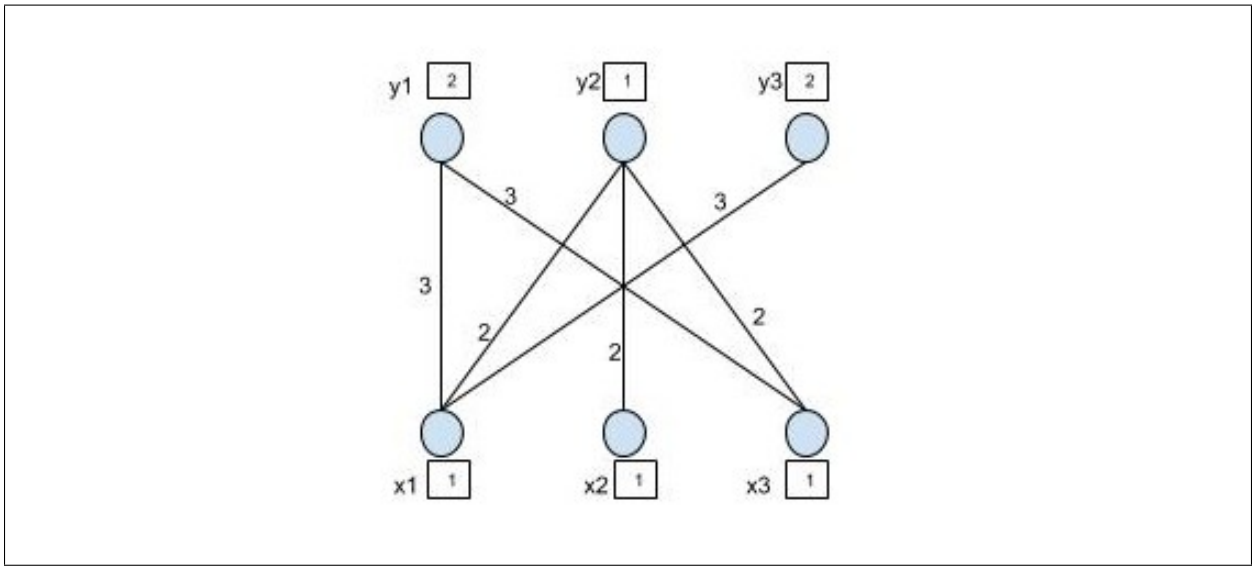


Figure 2.2: An equality graph. This particular equality graph is a subgraph of Figure 2.1.

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.

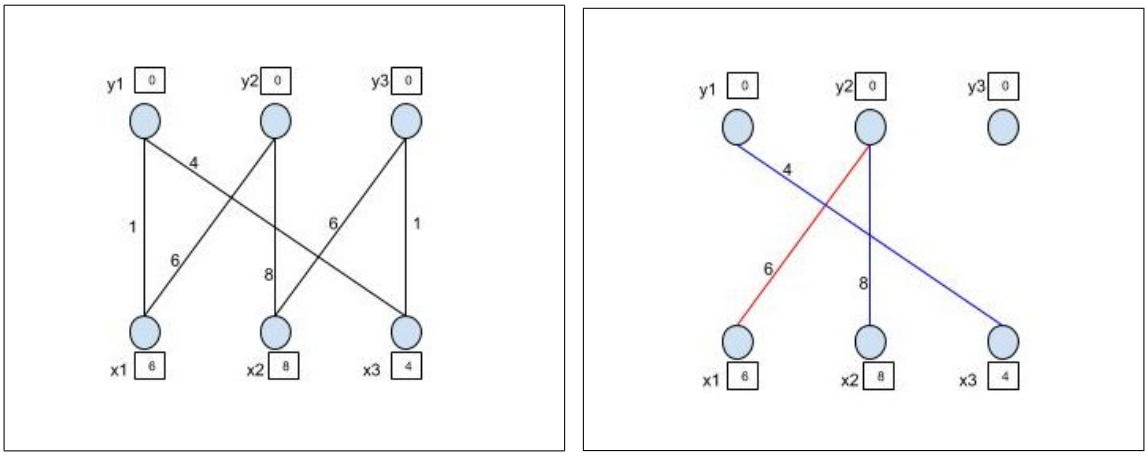
```
while(|M| < n) {
  - if there is an augmenting path in E_l
    - use the augmenting path to augment and increase the size of M
  - else
    - improve l to l' such that E_l exists in E_l'
}
```

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.

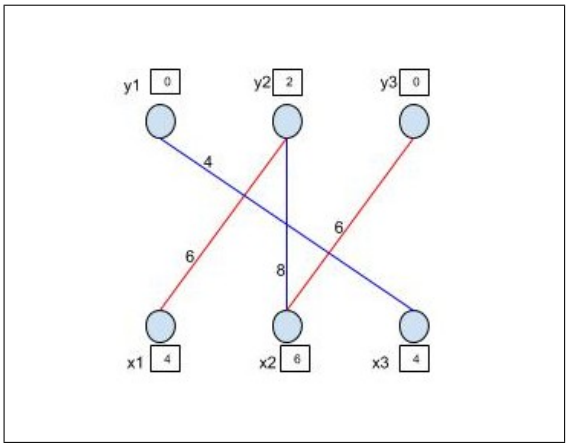
- Assign $y \in (N_l(S) - T)$. We choose y_2, which is matched, and increase the graph by assigning $S = \{x_1\} \cup \{x_2\}, T = \emptyset \cup \{y_2\}$. Go to 3.

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

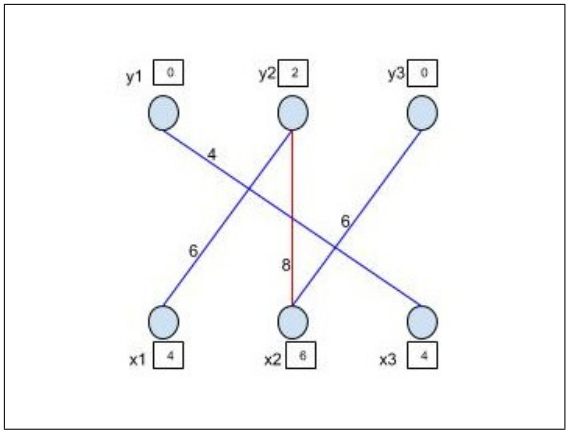
$$\alpha_l = min \begin{cases} (x_1, y_1) & 6 + 0 - 1 \\ (x_1, y_3) & 6 + 0 - 0 \\ (x_2, y_1) & 8 + 0 - 0 \\ (x_2, y_3) & 8 + 0 - 6 \end{cases} = 2$$

- 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.

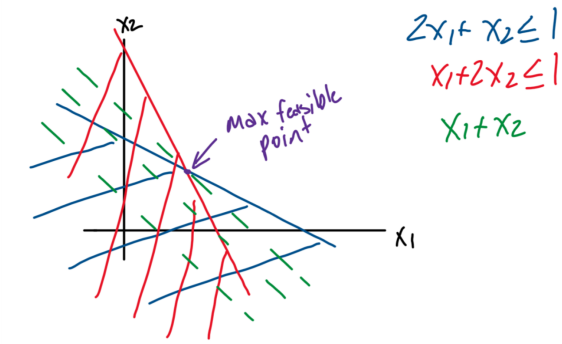


Figure 2.1: Finding optimal solution in the feasible space

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.

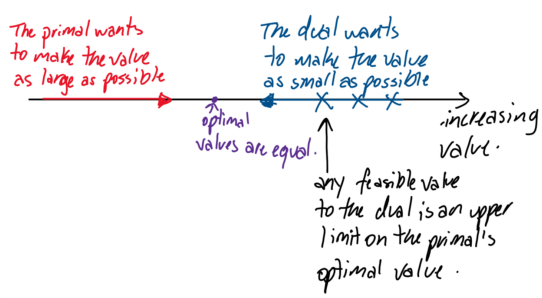


Figure 2.2: The dual places an upper limit on the primal's optimal value

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.

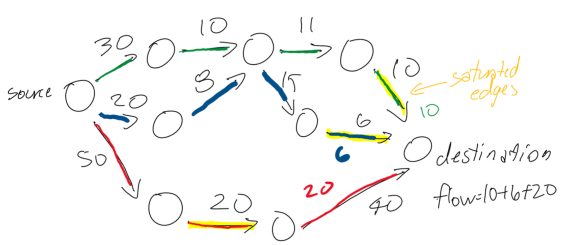


Figure 2.3: Network flow from a source to a destination along edges with capacity

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.

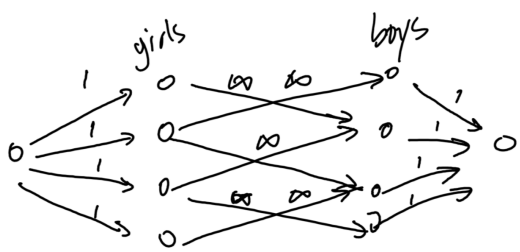


Figure 2.4: Matching Problem as a network flow

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.

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

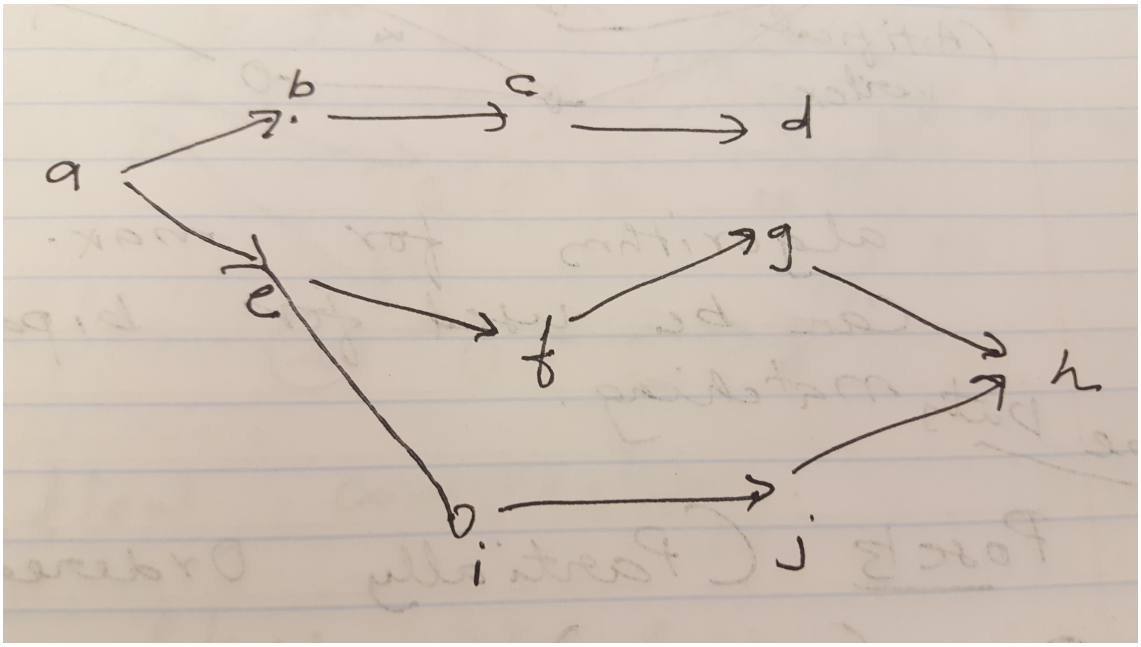
X: Any set.

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 $\alpha(G)/2$.

Session 1

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

Lecture 3 Session 1: September 28

Lecturer: Vijay Garg *Scribe: Bhargavi Karthik*

3.1 LP-Duality

In Kuhn-Munkres (KM) algorithm, we find matching using the concept of labeling the vertices such that it satisfies the feasibility constraints. Using LP-Duality, we can come up with similar ideas for solving problems in a more intuitive way.

3.1.1 Example for LP Duality

There is an objective function which we are trying to minimize,

$7x_1 + x_2 + 5x_3$

subject to constraints,

$x_1 - x_2 + 3x_3 \geq 10$

$5x_1 + 2x_2 - x_3 \geq 6$

where x_1, x_2, x_3 are the decision variables and assumption is that,

$x_1, x_2, x_3 \geq 0$

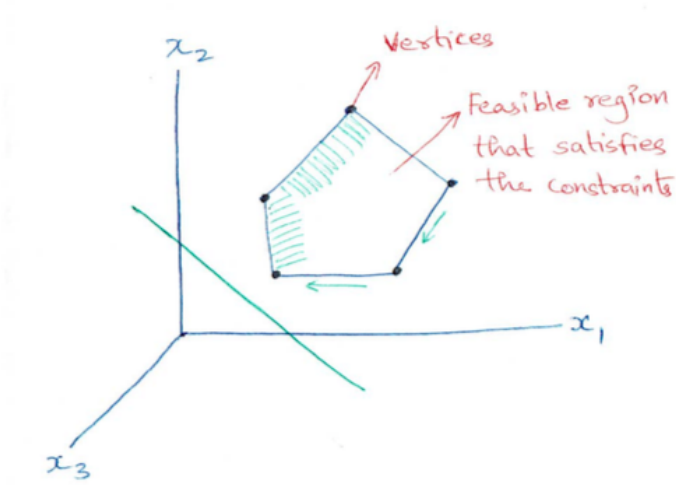


Figure 3.1: Finding optimal solution in the feasible space

Every constraint is in some sense dividing up the feasible region into two half spaces. The goal here is to find the optimal solution. If we can draw level sets for the given objective function, which means we are trying to optimize along some dimension. In this case we are trying to find the minimum. While trying to optimize along some direction, the point one always hits will be a corner. One of the important properties as per the Simplex algorithm is - "The extremal point is always a vertex".

The approach is to figure out one feasible point and start from there. If it is not a corner, keep moving until we find a corner and keep optimizing. However, if we are at a corner and if there are n other corners to go to, then go in the direction that has the most advantage.

Suppose we have solution, x = (2, 1, 3). Lets substitute in the objective function,

$7(2) + 1 + 5(3) = 14 + 1 + 15 = 30$

this solution is also feasible as it satisfies the constraints,

$2 - 1 + 3(3) = 10$

$5(2) + 2(1) - 3 = 9$

Trying to find a bound on primal:

$7x_1 + x_2 + 5x_3 \geq x_1 - x_2 + 3x_3 \geq 10$ by comparing coefficients point-wise and $x_i \geq 0$

solution has to be ≥ 10 but we are trying to get a better or greater lower bound

Add up the constraints equations,

$6x_1 + x_2 + 2x_3 \geq 16$

The objective function is still greater than the above equation. Hence new lower bound is 16.

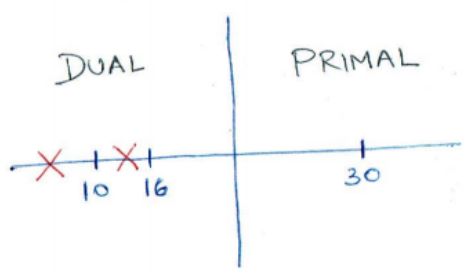


Figure 3.2: Representation of the example as duality

The goal of primal was to minimize and so now the goal of dual will be to maximize. We have to figure out what combination of each constraint should we add such that it gives the tightest bound on the objective function.

We have n decision variables and m constraints in our primal. So dual will have m variables (one variable for each constraint in primal) and n constraints (since n decision variables in primal).

Objective function is $10y_1 + 6y_2$ subject to constraints,

$y_1 + 5y_2 \leq 7$

$-y_1 + 2y_2 \leq 1$

$3y_1$ - $y_2 \leq 5$

In a generalized representation of the primal, objective function can be written as

$$\sum_{i=1}^{n} c_i x_i$$

constraints can be written as

$$\sum_{j} A_{ij} x_j \geq b_j$$

where A is a matrix of dimension m*n, c and b are vectors of size n and m respectively.

For dual, b is translated to c, c is translated to b and matrix A becomes A^T.

3.1.2 Strong Duality

Theorem 3.1 *The primal program has finite optimum if and only if its dual has finite optimum. Moreover, if $x*$ and $y*$ are optimal solutions for (P) and (D) then,*

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*$$

Note: This translates into all the min-max theorems such as max flow = min cut, max matching = min vertex cover and min number of chains to cover a poset = max sized anti-chain

3.1.3 Weak Duality

Theorem 3.2 *If x and y are feasible solutions for (P) and (D) then,*

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} b_i y_i$$

Proof:

Claim 3.3

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{j=1}^{n} (\sum_{i=1}^{m} a_{ij} y_i) x_j$$

This claim is true because y is feasible and $x \geq 0$

Claim 3.4

$$\sum_{i=1}^{m} (\sum_{j=1}^{n} a_{ij} x_j) y_i \geq \sum_{i=1}^{m} b_i y_i$$

This claim is true because x is feasible and $y \geq 0$

Equating claim 3.3 to claim 3.4

Hence proved,

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} b_i y_i$$

■

3.1.4 Complementary Slackness Conditions

Theorem 3.5 *Given that x,y are (P) and (D) feasible solutions. Then x and y are both optimal if and only if the following complementary slackness conditions are satisfied:*

Claim 3.6

$$\forall j \quad either(x_j = 0) \quad or \quad \sum_{i=1}^{m} a_{ij} y_i = c_j$$

Claim 3.7

$$\forall i \quad either(y_i = 0) \quad or \quad \sum_{j=1}^{n} a_{ij} x_j = b_i$$

3.2 LP Duality and Maximum Matching

For the Kuhn-Munkres Algorithm, we started with x variables and came up with labels through initial feasible labeling. These labels are the dual variables. And we tried to get complementary slackness.

3.2.1 Maximum Matching Problem As Linear Program

Primal:

$$max \sum_{i,j} W_{ij} x_{ij}$$

subject to:

$$\sum_{j} x_{ij} \leq 1$$

and

$$\sum_{i} x_{ij} \leq 1$$

where x_{ij} is an edge between A and B, i is an index from A and j is an index from B. For each edge, there is a weight W_{ij}. We need to pick the edges such that the sum of their weights is maximum.

Replacing integral constraint by linear constraint,

$$x_{ij} \in \{0, 1\} \Rightarrow x_{ij} \geq 0$$

The above transformation is called LP-Relaxation. This can be used only when we can prove that the optimal solution is always going to be integral.

When $|A| = |B| = n$

Number of decision variables = n^2

Number of constraints = 2n

Dual:

Number of decision variables = 2n

Number of constraints = n^2

Objective function (minimizing vertex labels):

$$min \sum u_i + \sum v_j$$

subject to: $u_i + v_j \geq w_{ij}$

$u_i \geq 0 \quad and \quad v_j \geq 0$

where u_i is constraints for side A and v_j is constraints for side B

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

Lecture 3 Session 2 : September 28

Lecturer: Vijay Garg *Scribe: Carmina Francia*

3.1 Stable Matching/Marriage

Matching has a lot of real world applications. We see it in markets, buyers/sellers, computer processes, residencies, and so on. So far we've used matching in the context of girls likes boys and boys like girls. These matchings can sometimes have *preferences*.

This is called **Stable matching/marriage**. It is finding a matching between two equal sets each following an ordered preference list.

For example, let's create two preference lists:

Table 3.1: Men Preference List

M1	W4	W1	W2	W3
M2	W2	W3	W1	W4
M3	W2	W4	W3	W1
M4	W3	W1	W4	W2

Table 3.2: Women Preference List

W1	M4	M1	M3	M2
W2	M1	M3	M2	M4
W3	M1	M2	M3	M4
W4	M4	M1	M3	M2

This is still a bipartite and a good matching would be when we don't have one woman assigned to two men and vice versa.

Blocking Pair

What kind of marriage would not last long? Let's say we have the following matchings: *(m,w)* and *(m', w')*. If *m* and *w'* are not engaged, but m prefers *w'* to w and *w'* prefers *m* to *m'* , then there's an incentive for them to break. This is called a **blocking pair**.

The goal of stable matching is to find a matching that has *no* blocking pair.

3.2 Gale and Shapley Algorithm

The idea is similar to DGS algorithm: people bidding and going for the most valuable item. We will have men proposing (bidders). The difference is the other set, in this case women, has a say. Women can reject men.

Data: men preference list and women preference list
Result: (m,w) matchings
Assuming number of men is equal to number of women;
Put all men in a list (initially everyone if free);
while *free list of men is not empty* **do**
  pick any free man;
  man proposes to most preferred woman that he has not proposed to;
  if *woman is free* **then**
    men is engaged with that woman;
  else
    woman rejects the less desirable man of the two men based on her preference list;
  end
  rejected man go back to free list (does not matter where in the list)
end

Key notes:

- *engaged* means man and woman are not married. Matchings can change as algorithm progresses.
- Women rejects based on their preference list
- if you're a *free* woman, you cannot say no. Only way to reject is if a better man comes.
- Once a woman is engaged, she cannot be free.
- Algorithm **ALWAYS** terminates and everybody is married.

This example is *man-optimal*. It is seen as worse from women's perspective. However, as the algorithm progresses, men decrease on their preference while women are safe or their preference increases.

3.3 Stable Matching Example

Let's walk through a man-optimal example...

Table 3.3: Men Preference List

M1	**W4**	W1	W2	W3
M2	*W2*	**W3**	W1	W4
M3	**W2**	W4	W3	W1
M4	*W3*	**W1**	W4	W2

Table 3.4: Women Preference List

W1	**M4**	M1	M3	M2
W2	*M1*	**M3**	M2	M4
W3	*M1*	**M2**	M3	M4
W4	*M4*	**M1**	M3	M2

1. Let's begin with *M1*. *W4* is free so *M1* and *W4* becomes engaged.
2. We move on to *M2*. *W2* is free and becomes engaged with *M2*.
3. We go to *M3*. *W2* is already engaged, so we check the woman's preference list. *W2* prefers *M3* over *M2*. So now, *M3* and *W2* are engaged. *M2* goes back to the free list.
4. We go to *M4*. *W3* is free and becomes engaged with *M4*.

5. We pick *M3*. The next woman on his preference list that he has not proposed to is *W3*. *W3* is engaged to *M4*. We check the *W3*'s preference list. *W3* prefers *M3* over *M4*. So *M3* and *W3* becomes engaged and *M4* goes back to the list.
6. We pick *M4*. *W1* is free so *M4* and *W1* becomes engaged.

The list is now empty and returns the matchings: *(M1, W4)*, *(M2, W3)*, *(M3, W2)*, *(M4, W1)*. Since we don't see any blocking pairs, this is a stable matching.

Now let's say it's a woman-optimal. Same steps apply except we would initialize with all women in the free list and pick a woman to start with. Once the list is empty and everybody is married/matched, you will notice that we come up with the same matchings:*(W1, M4)*, *(W2, M3)*, *(W3, M2)*, *(W4, M1)*.

Note: If the man-optimal and woman-optimal are *equal*, this is the **only stable matching**.

Session 1

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

Lecture 4 Session 1: September 29
Stable Marriage

Lecturer: Vijay Garg *Scribe: Tiffany Tillett*

4.1 Review

Recall the algorithm to find a stable marriage from last class.

4.1.1 Inputs

1. mPref: data structure containing preferences for each man
2. wPref: data structure containing preferences for each woman
3. rank: where rank[i][j] indicates the rank of man j for woman i; higher preference correlates to lower rank

4.1.2 Stable Marriage Algorithm

1. $freelist$ = list of men who are free (initially has all men)
2. while ($freelist$ is not empty):
    3. choose some man m from $freelist$
    4. m proposes to his most preferred woman w that he has not already proposed to
    5. if w is free:
        6. w accepts proposal (assign m to w)
    7. else: // w is already engaged to m'
        8. if $rank[w][m] > rank[w][m']$: // w prefers m'
            9. add m back to $freelist$ // reject m; w still engaged to m'
        10. else:
            11. add m' back to the $freelist$ // reject m'
            12. w gets engaged to m
13. print assignment

4.2 Stable Marriage Algorithm Correctness

Theorem 4.1 *The Algorithm terminates with a matching.*

Proof:

Claim 4.2 *The last woman in a man's preference list must be free*

Invariant: Once a woman is engaged, she stays engaged.

In order for a man to make it to his last preference, the previous $n-1$ women must already be engaged since the only reason they can reject his proposal is if they are engaged to someone that they prefer more. Since the number of men and women is equal, the last woman in the list can never be engaged.

Since the last woman in a man's preference list will always be free, every man is guaranteed that he will match with one woman, so a matching will exist.

■

Theorem 4.3 *The algorithm (if it terminates) ends with a stable matching.*

Proof: Assume that $(m, w')(m', w)$ is a blocking pair where m is married to w' and w is married to m'.

In order for $(m, w')(m', w)$ to be a blocking pair, this means that m must prefer w to w' and that w must prefer m to m'.

Since m prefers w to w', he would have proposed to w first and gotten rejected.

Invariant: A woman's assignment can only improve with execution of the algorithm.

Therefore, w can only have rejected m if she had a proposal from someone she preferred more. If that person was m', we know that she must prefer m' to m. If that person was not m', we know that she prefers that person to m and that she prefers m' to that person, so by extension, she prefers m' to m. Generally, w will always prefer her current fiance to all men she rejected, so she must prefer m' to m.

The requirement for (m, w) to be a blocking pair was that m must prefer w to w' and that w must prefer m to m'. These two requirements cannot co-exist, so by contradiction, we know that $(m, w')(m', w)$ cannot be a blocking pair. ■

4.3 Posets: Partially Ordered Sets

4.3.1 Definition

A poset is defined as $(X, \leq)$ where X is a set and $\leq$ is a binary relation on that set.

A reflexive poset has the following properties:

1. reflexive - for all vertices v in V: $v \leq v$
2. anti-symmetric - for all pairs of vertices u, v in V: if $v \leq u$ and $u \leq v$ then $u == v$

3. transitive - for all groups of vertices u, v, w in V: $u \leq v \leq w$ implies that $u \leq w$

Consider all following posets to be reflexive posets.

4.3.2 Example 1

X: $\mathbb{R}^n$

$\leq$: $U \leq V \equiv$ for all i: $u[i] \leq v[i]$

Claim 4.4 *$(X, \leq)$ is a poset.*

4.3.2.1 Numerical Example 1

$M1 = (2, 5, 3)$; $M2 = (6, 8, 5)$

$2 \leq 6$, $5 \leq 8$, $3 \leq 5$ says that $M1 \leq M2$.

4.3.2.2 Numerical Example 2

$M1 = (1, 3, 1)$; $M2 = (2, 1, 2)$

$1 \leq 2$, $3 > 1$. What does this mean?

Definition 4.5 *$M1$ is incomparable to $M2$ if $(M1 \nleq M2)$ and $(M2 \nleq M1)$.*

4.3.3 Example 2

$z = \{a, b, c\}$

X: { {}, {a}, {b}, {c}, {a,b}, {a,c}, {b,c}, {a,b,c} }

$\leq$: $\subseteq$

$\{a, b\} \leq \{a, b, c\}$

$\{a, b\}$ is incomparable to $\{b, c\}$

4.3.4 Dealing with Incomparable Items

If $M1$ and $M2$ are incomparable, try to find $M3$ which is better than both $M1$ and $M2$.

$M1 = (1, 1, 2, 3)$

$M2 = (2, 2, 1, 2)$

Note: the numbers here indicate preference where 1 indicates first choice, 2 indicates second choice, etc

find $M3 = (1, 1, 1, 2)$ such that $M3 \leq M1$ and $M3 \leq M2$.

4.3.5 Hasse Diagrams: Pictoral Representations of Posets

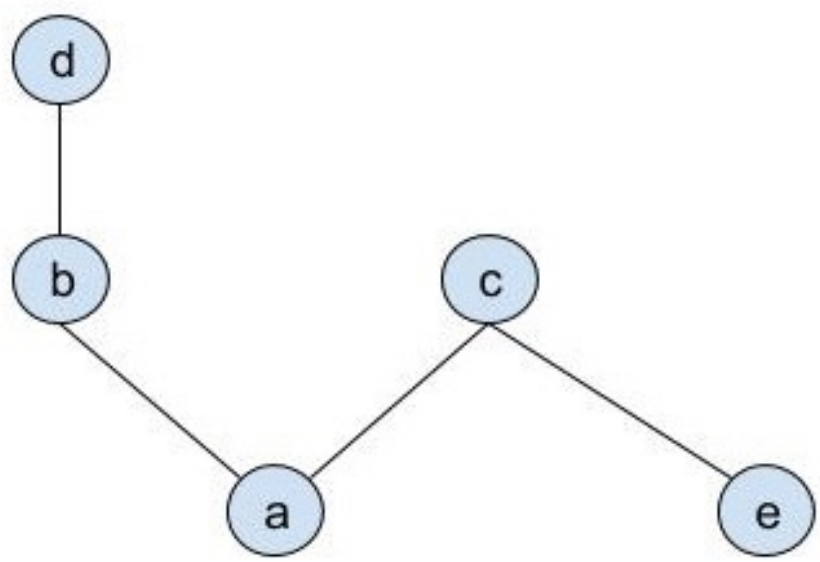


Figure 4.1: Hasse Diagram example

This diagram shows that:

$a \leq b$

$a \leq c$

$b \leq d$

$e \leq c$

c and d are incomparable

a and e are incomparable

4.3.6 Operations on Posets

Given a poset $(X, \leq)$:

Definition 4.6 *Let $x, y \in X$. $z \in X$ is meet of x and y if:*

1. $z \leq x$ *and* $z \leq y$

   AND

2. *for all* z'*:* $z' \leq x$ *and* $z' \leq y \implies z' \leq z$

Where meet is also known as the greatest lower bound. The first condition is the lower bound, and the second condition is that the lower bound must be greater than all other lower bounds.

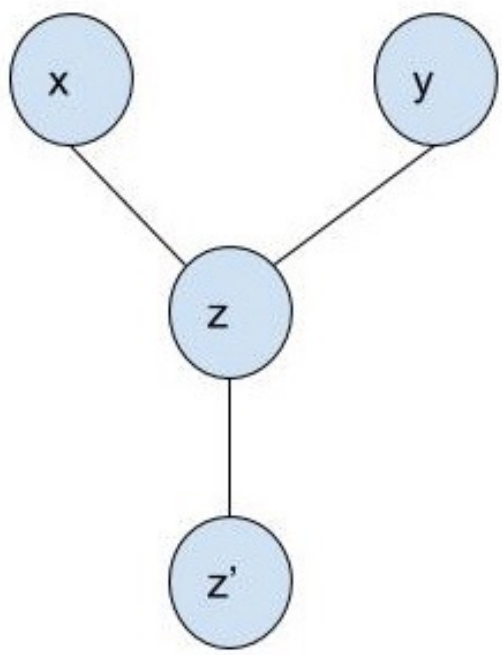


Figure 4.2: Meet Example; z is the meet of x and y

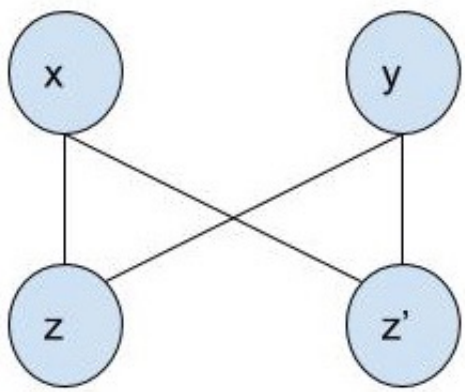


Figure 4.3: z is not a meet; z and z' are incomparable

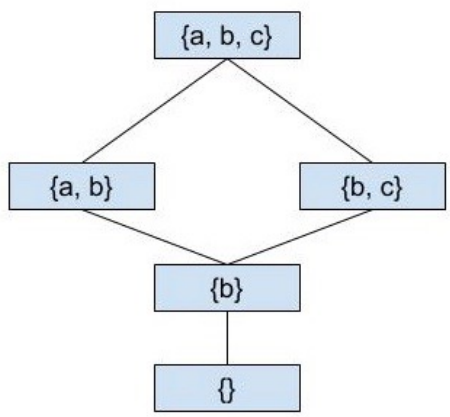


Figure 4.4: meet corresponds to intersection of sets

4.3.7 Lattice Posets

Definition 4.7 *A poset $(X, \leq)$ is lattice (nice) if for all $a, b \in X$, $meet(a, b)$ exists and $join(a, b)$ exists.*

Definition 4.8 *$join(a, b)$ is the greatest lower bound (reverse of meet).*

$(a \vee b) = join(a, b)$

$(a \wedge b) = meet(a, b)$

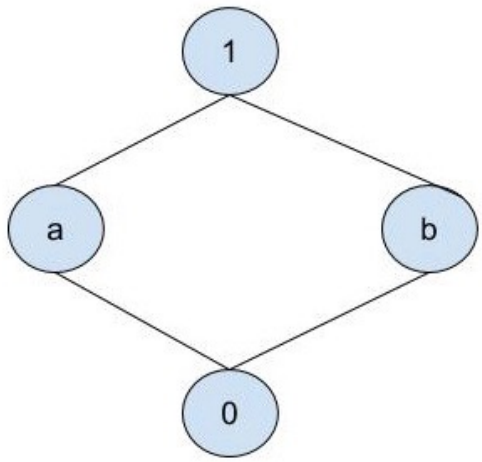


Figure 4.5: meet(a,1) = a

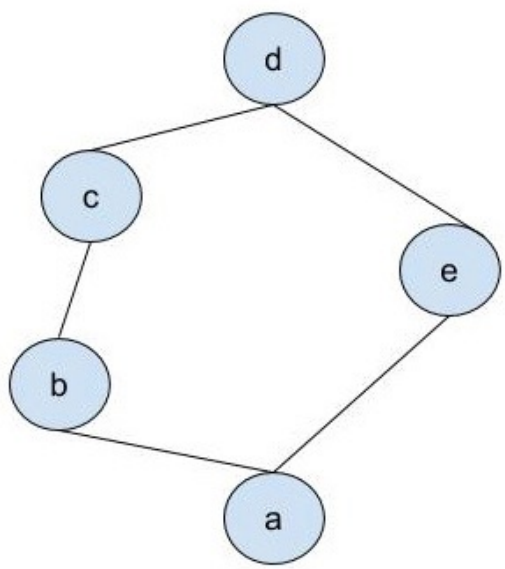


Figure 4.6: meet(c,e) = a; join(c,e) = d; meet(b,e) = a; join(b,e) = d

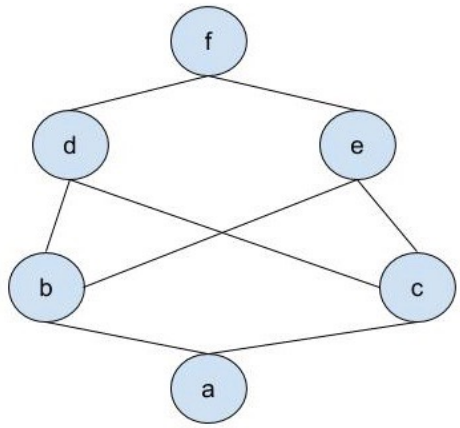


Figure 4.7: meet(b,c) = a; There is no meet of d and e since b and c are incomparable. a cannot be the meet since b and c are bigger than a.

The definition of a lattice can be extended to apply to all finite subsets of X.

Definition 4.9 *A lattice $L = (X, \leq$ is distributive if for all $x, y, z \in X$:*

1. $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
   OR
2. $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$

Referencing Figure 11.6:

$e \wedge (b \vee c) = e \wedge c = a$

$(e \wedge b) \vee (e \wedge c) = a \vee a = a$

This works out. However, we need to show this for all x, y, z.

$c \wedge (b \vee e) = c \wedge d = c$

$(c \wedge b) \vee (c \wedge e) = b \vee a = b.$

This does not work. This lattice is not distributive.

There is a theorem which states that lattices of certain shapes cannot be distributive.

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

Lecture 4, Session 4: September 29

Lecturer: Vijay Garg *Scribe: Zeid Ayssa*

4.1 Stable Marriage Optimality

After we introduced $Gale-Shapley$ stable matching algorithm, we need to prove that the resulting matching is optimal i.e. there is no better matching for men.

Note: throughout this section, we will assume men optimal matching. Furthermore, lower ranking numbers mean higher preference i.e.1 means it is the man's/woman's 1st choice and 4 means that it is the man's/woman's 4th choice.

Assuming men optimal assignment, Below is the Hasse diagram of all possible assignments. Note that this is not necessarily matching assignments, but rather all possible men assignments to women. This is not necessarily a matching due to the fact that if all men were matched to their first woman preference, we could have multiple men preferring the same women and we cannot match a women to more than one man.

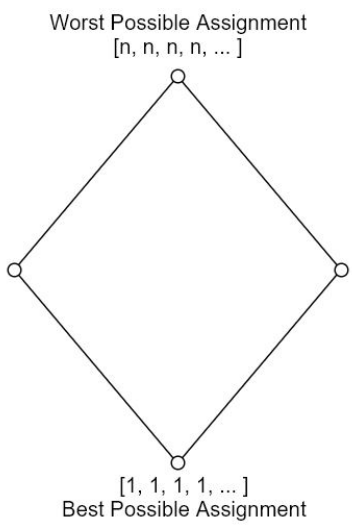


Figure 4.1: Hasse diagram of all possible men assignments

Example: if we have two matchings $M_1 = [1, 2, 1, 3, 1]$ and $M_2 = [2, 3, 2, 3, 1]$, if $M_1 \leq M_2$, where $M_1 \leq M_2$ is defined as $\forall i : M_1[i] \leq M_2[i]$, then we know that every man in M_1 is either happier or as happy compared to M_2 matching.

Let u and v be vectors of size n representing two different men assignments in figure 4.1 where $u \leq v$ is defined as $\forall i : u[i] \leq v[i]$

We know from a previous section that figure 4.1 is a Partially Ordered Set (POSET) because it is Reflexive, Anti-Symmetric, and Transitive.

Claim 4.1 *$(X, \leq)$ is a lattice poset where X is the set of all possible assignments shown in figure 4.1*

Definition 4.2 *A poset is lattice if every two elements in the poset have a unique upper bound or join and a unique lower bound or meet. Minimum upper bound or join is denoted by $\sqcup$ and maximum lower bound or meet is denoted by $\sqcap$.*

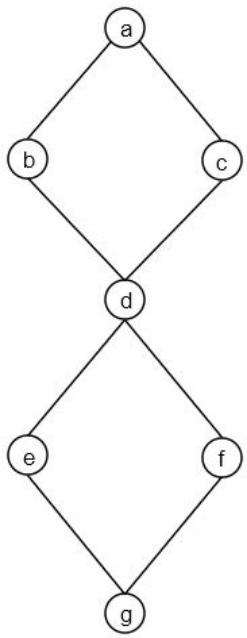


Figure 4.2: Hasse diagram of a lattice poset

Figure 4.2 is an example of a lattice poset i.e. for every two elements there is a unique upper and lower bounds. Let us check:

$b \sqcap c = d$ and $b \sqcup c = a$

$e \sqcap c = g$ and $e \sqcup c = a$

Proof: Let u and v be two different assignment sets with size 4

$u[1, 3, 2, 1]$

$v[2, 1, 3, 1]$

$w = u \sqcap v = [1, 1, 2, 1]$ i.e. pairwise min operation.

We know that $w \leq u$ and $w \leq v$, so w is a more preferable matching to men than u or v. This also means that w will always be greater than or equal to any other assignment set that is less than or equal to u or v. For example, if there is another set $g = [1, 1, 1, 1]$ then $g < u$ and $g < v \implies g \neq w \implies w > g$ since w is already defined as the meet between sets u and v. This proves that the poset is a lattice poset. ■

Claim 4.3 *$(X, \leq)$ is a distributed lattice poset*

We need to verify that poset $(X \leq)$ is a distributive lattice.

Let $x = [a_1, a_2, a_3, \ldots]$, $y = [b_1, b_2, b_3, \ldots]$, and $z = [c_1, c_2, c_3, \ldots]$ be different sets in the poset. For a poset to be a distributed lattice we need to verify that $a \sqcap (b \sqcup c) = (a \sqcap b) \sqcup (a \sqcap c)$ which means we need to verify that:

$min(a_x, max(b_x, c_x) = max(min(a_x, b_x), min(a_x, c_x))$

Proof: Let $x[1, 3, 2, 1]$, $y[2, 2, 3, 1]$, and $z[3, 1, 2, 3]$ be three different assignment sets with size 4. Now, let us see if this is a distributed lattice.

<u>1st *Element*</u>

$min(1, max(2, 3)) = min(1, 3) = 1$

$max(min(1, 2), min(1, 3)) = max(1, 1) = 1$

<u>2nd *Element*</u>

$min(3, max(2, 1)) = min(3, 2) = 2$

$max(min(3, 2), min(3, 1)) = max(2, 1) = 2$

<u>3rd *Element*</u>

$min(2, max(3, 2)) = min(2, 3) = 2$

$max(min(2, 3), min(2, 2)) = max(2, 2) = 2$

<u>4th *Element*</u>

$min(1, max(1, 3)) = min(1, 3) = 1$

$max(min(1, 1), min(1, 3)) = max(1, 1) = 1$

Given that for all elements in the assignment sets x, y, and z the result of $min(a_x, max(b_x, c_x)$ was equal to the result of $max(min(a_x, b_x), min(a_x, c_x))$, then this is a distributive assignment set. Since this should hold true for all possible assignment sets in our poset $(X, \leq)$ then we can say that this is a distributed lattice. ■

Why do we care if it is a distributed lattice or not? If a poset is a distributive lattice then it makes it similar to Boolean algebra since union and intersection distribute over each other. This property will allow us define a Boolean predicate later on to help us prove optimal stable matching.

4.1.1 Hasse Diagram of All Assignments and Stable Matching Region

Figure 4.3 shows the Hasse diagram of all possible assignments. It also shows the men optimal and women optimal stable matching. There is a region that has different sets of stable matching i.e. no blocking pair, but there is only one optimal stable matching for men and women.

A men optimal stable matching is a matching where there is no other stable matching set that leads to happier men i.e. there is no other stable set that is less than the current stable matching. The same applies to women where a women optimal stable matching is a matching where there is no other stable matching set that leads to happier women i.e. there is no other stable set that is less than the current stable matching.

Starting from the initial set $[1, 1, 1, \ldots]$ and taking any path when using $Gale-Shapley$ algorithm will always give us the optimal men or women stable matching.

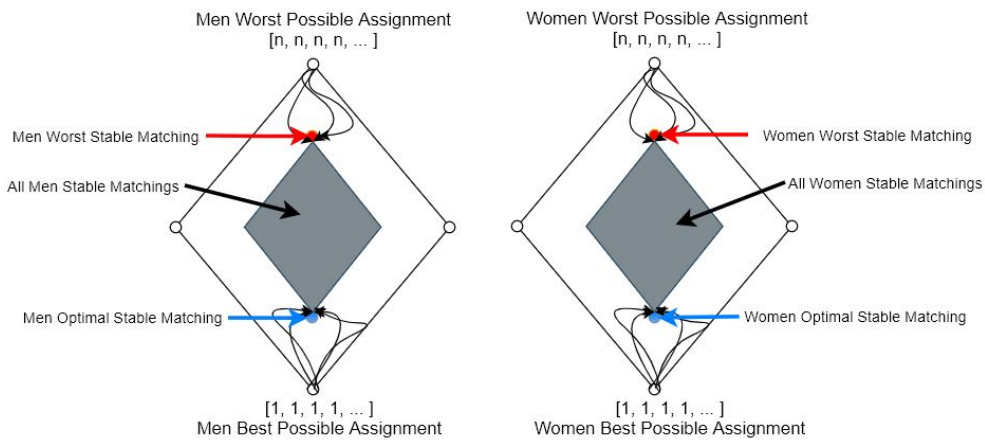


Figure 4.3: Hasse diagram of all possible assignments along with the stable matching region

4.1.2 Finding Stable Matching Assignments

If we want to write an algorithm to find the optimal stable matching, we need to figure out some sort of logic to determine if the current man and woman matching is the best we can do before moving on. Trying to go over all the possible matchings would take a lot of time since the total number of possible assignments is n^n because for every man there are n women i.e. $m = [\ldots n] \implies [n][n]$ array.

Figure 4.4 shows the preference for men and women where we can formally represent them as:

<u>Men Preferences:</u>

$m_1 = [2, 3, 1, 4]$

$m_2 = [2, 1, 3, 4]$

$m_3 = [1, 3, 4, 2]$

$m_4 = [3, 2, 4, 1]$

<u>Women Preferences:</u>

$w_1 = [3, 2, 1, 4]$

$w_2 = [1, 2, 4, 3]$

$w_3 = [4, 1, 2, 3]$

$w_4 = [4, 3, 2, 1]$

Figure 4.4 also shows one possible stable matching $\{\{m_1, w_2\}, \{m_2, w_1\}, \{m_3, w_3\}, \{m_4, w_4\}\}$.

4.1.2.1 Forbidden State

Figure 4.5 shows a matching that has a forbidden state. We can see from figure 4.5 that w prefers m' more than m, we can also see that m' prefers w more than w', therefore this a blocking pair which means that this is not a stable matching.

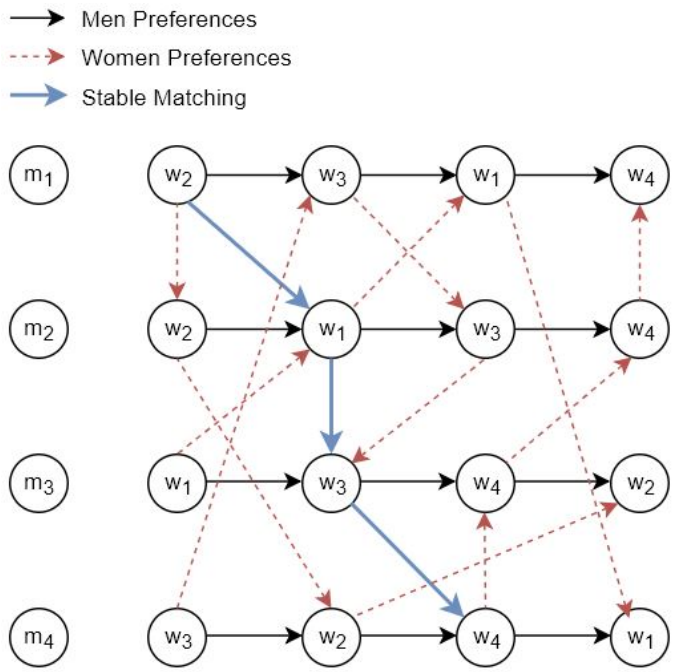


Figure 4.4: Possible men and women matching

From figure 4.5 we can say that a matching is not stable if there is a preference arrow that points to a node before the current matching region since it will represent a blocking pair. Similarly, we can say that a matching is stable if there are no preference arrows pointing to a node that is before the current matching region. If our algorithm makes sure that there are no forbidden states, then we know for sure that if a women rejects a man then there is no way there was going to be a stable marriage that would include this pair. If a matching does not include a forbidden state, then this also proves the claim that this is a distributed lattice.

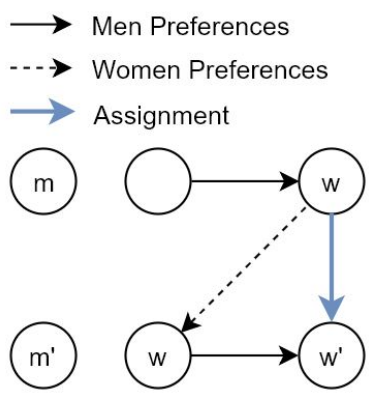


Figure 4.5: Forbidden state

Figure 4.6 shows a matching that has a blocking pair or a forbidden state. A matching with a forbidden state means that the matching is not stable.

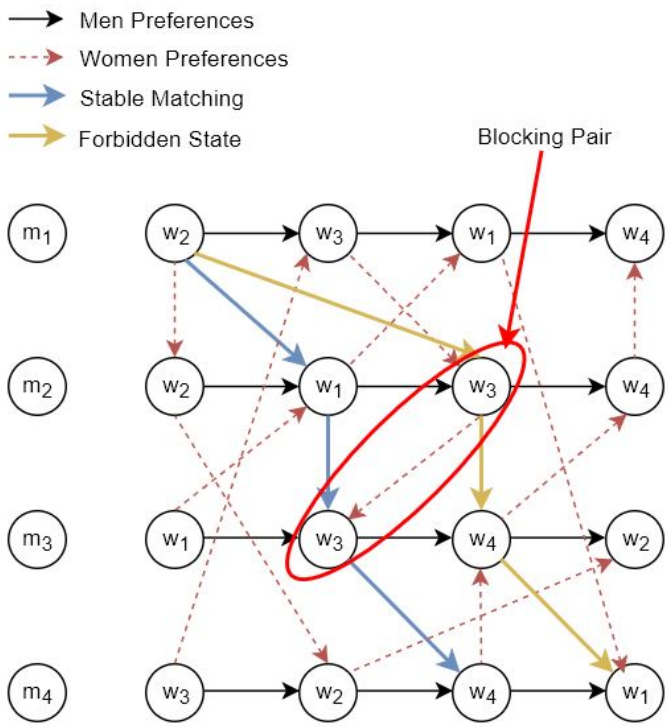


Figure 4.6: Possible men and women matching

In the next class we will go over a formal definition of the forbidden state and how it will help us prove that $Gale - Shapley$ algorithm will give us an optimal stable matching. We will define an algorithm that uses (G, i, B) where:

G : Assignment

i : men

B : Boolean Predicate, where B:

- Is a matching
- Has no blocking pair

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

Scribe Assignment 1: September 29

Lecturer: Vijay Garg *Scribe: Zach Southwell*

1.1 The LP formulation of Max-Flow Min-Cut problem

1.1.1 Problem Domain

We are given a connected graph which is directed from an origin node s to a destination node t, where each edge in the graph has some non-negative capacity which represents the maximum flow across the edge.

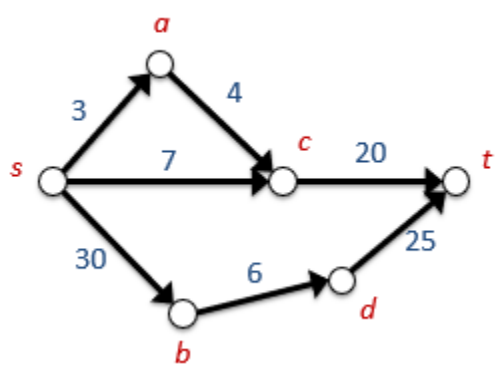


Figure 1.1: sample flow

1.1.2 Primal Problem

We want to calculate the *maximum flow* which can be sent from s to t. Another way to state this is that we want to maximize the flow through each edge from s subject to these constraints:

- Conservation constraints: The size of a the incoming flow for a vertex must be the same as the outgoing flow
- Capacity constraints: No edge may carry more than its maximum capacity. Define $c(u, v)$ as the maximum capacity for an edge $(u, v) \in E$
- The flow through each edge must be non-negative

This can be stated as the following linear program:

Maximize:

$$\sum_{v:(s,v)\in E} f(s,v)$$

Subject to:

$$\sum_{u:(u,v)\in E} f(u,v) = \sum_{w:(v,w)\in E} f(v,w)$$
$$f(u,v) \leq c(u,v) \qquad \forall (u,v) \in E$$
$$f(u,v) \geq 0 \qquad \forall (u,v) \in E$$

1.1.3 Dual Problem

The dual problem is to *cut* the graph into two components with one containing s and the other containing t by removing the edges with the minimum total capacity. Let S be the component containing s and T be the component containing t. Every node in the graph (except for s and t) has a dual variable $y(u)$ which is 1 if $u \in S$ and 0 otherwise, and every edge in the graph has a dual variable $y(u,v)$ which is 1 if $u \in S$ and $v \in T$ and 0 otherwise. The dual problem can be stated as:

Minimize:

$$\sum_{(u,v)\in E} c(u,v) * y(u,v)$$

Subject to:

$$y(v) + y(s,v) \geq 1 \qquad \forall v : (s,v) \in E$$
$$y(v) - y(u) + y(u,v) \geq 0 \qquad \forall (u,v) \in E (s \neq u, t \neq v)$$
$$-y(u) + y(u,t) \geq 0 \qquad \forall u : (u,t) \in E$$

1.1.4 Primal Example

For figure 1.1, the primal problem can be stated as:

Maximize:

$$f(s,a) + f(s,c) + f(s,b)$$

Subject to:

$$\begin{gathered}
f(s,a) - f(a,c) = 0 \\
f(s,b) - f(b,d) = 0 \\
f(a,c) + f(s,c) - f(c,t) = 0 \\
f(b,d) - f(d,t) = 0 \\
f(s,a) \leq 3 \\
f(s,c) \leq 7 \\
f(s,b) \leq 30 \\
f(a,c) \leq 4 \\
f(b,d) \leq 6 \\
f(c,t) \leq 20 \\
f(d,t) \leq 25 \\
f(u,v) \geq 0 \qquad \forall (u,v) \in E
\end{gathered}$$

1.1.5 Dual Example

Minimize:

$$c(s,a)*y(s,a)+c(s,b)*y(s,b)+c(s,c)*y(s,c)+c(a,c)*y(a,c)+c(b,d)*y(b,d)+c(c,t)*y(c,t)+c(d,t)*y(d,t)$$

Subject to:

$$\begin{gathered}
y(a) + y(s,a) \geq 1 \\
y(b) + y(s,b) \geq 1 \\
y(c) + y(s,c) \geq 1 \\
y(c) - y(a) + y(a,c) \geq 0 \\
y(d) - y(b) + y(b,d) \geq 0 \\
-y(c) + y(c,t) \geq 0 \\
-y(d) + y(d,t) \geq 0
\end{gathered}$$

1.1.6 Solution

The following mapping satisfies the primal with a value of 16:

$$f(s,a) = 3; f(s,c) = 7; f(s,b) = 6; f(a,c) = 3; f(c,t) = 10; f(b,d) = 6; f(d,t) = 6$$

And the following 'cut' satisfies the dual with a value of 16:

$$y(s,a) = 1; y(s,c) = 1; y(s,b) = 0; y(a,c) = 0; y(c,t) = 0; y(b,d) = 1; y(d,t) = 0$$

Since the primal and dual solutions have the same value, we know that these solutions are optimal.

A value of 1 for $y(s,a), y(s,c), y(b,c)$ in the dual problem indicates that the edge belongs to the 'cut' set. Finding these constraining edges is the intuitive and natural approach to solving the maximization problem. This is interesting because even without formal knowledge of LP duality in this case, you would have a tendency to construct the dual in order to solve the primal problem.

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

Lecture : LP Formulation of the Stable Marriage Problem

Lecturer: Vijay Garg *Scribe: Eric Addison*

0.1 Introduction

In this note, we will work out a linear programming (LP) formulation of the Stable Marriage problem. The problem will be briefly reviewed, followed by introduction of relevant notation. Characteristics of the problem will be translated into linear algebraic constraints, which will allow us to write down the LP statement of the problem, along with its dual (DLP). Finally, we will work through a small example.

The formulation and notation in this note is based primarily on that found in [1], however several other articles can easily be found that provide similar treatments.

0.2 Review of the Stable Marriage Problem

Consider two sets of agents M and W (e.g. men and women), with $|M| = |W|$ for simplicity. Each agent provides a complete list of preferences of the members from the opposite set, which implicitly defines a ranking operator ($\geq_m$ for each man m and $\geq_w$ for each woman w) such that $(M, \geq_w)$ and $(W, \geq_m)$ are totally ordered sets. We say that a woman w prefers man m to another man m' if $m \geq_w m'$.

The goal of the stable marriage problem is to find a *stable matching* between the two sets M and W. Define a *matching* as a mapping $\mu : (M \cup W) \to (M \cup W)$ where:

- $\mu(w) = m \Leftrightarrow \mu(m) = w$, i.e. m is matched to w
- if $\mu(w) \notin M$, then $\mu(w) = w$ and w is not matched
- if $\mu(m) \notin W$, then $\mu(m) = m$ and m is not matched

A matching is *stable* if $\nexists (m, w)$ such that $m >_w \mu(w)$ and $w >_m \mu(m)$ (where $>_m$ is the irreflexive version of $\geq_m$). That is, there is no pair of matched man and matched woman who both prefer each other to their current match. Such a pair is know as a *blocking pair*.

0.3 LP Formulation

We begin by defining the *incidence vector* of a matching μ as a vector $x \in \{0, 1\}^{|M| \times |W|}$, indexed such that $x_{m,w} = 1 \Rightarrow \mu(m) = w$, else $x_{m,w} = 0$. The following linear constraints are necessary and sufficient to

classify an arbitrary x as a matching:

$$\sum_{j \in W} x_{m,j} \leq 1 \quad \forall m \in M \tag{A}$$

$$\sum_{i \in M} x_{i,w} \leq 1 \quad \forall w \in W \tag{B}$$

$$x_{m,w} \geq 0 \quad \forall (m,w) \in M \times W \tag{C}$$

Constraints (A) and (B) together define the requirement that a man (or woman) can be either unmatched or matched to a single other person: no double matchings. Constraint (C) is satisfied by the definition of x, but explicitly listing it will complete the linear program.

If the nature of the incidence vector is unclear, imagine representing is as a matrix X, where each entry X_{ij} is a boolean value that determines whether man m_i is matched to woman w_j. Clearly to form a valid matching, we can have at most a single 1 in each row and column, or else there would be a man or woman who is matched to more than one partner. A perfect matching is achieved when the matrix X is a permutation matrix.

0.3.1 The Assignment Problem

Before describing the linear program associated with the marriage problem (MP), we can first do so for the simpler assignment problem (AP), which contains only one sided preferences. In this problem, imagine that the set M is a set of buyers, and W is a set of sellers. Define a *fractional matching* as a vector $x \in \mathbb{R}^{|M| \times |W|}$ that satisfies constraints (A-C). The linear program AP can then be written as

$$\text{Maximize} \sum_{(i,j) \in M \times W} \alpha_{ij} x_{ij}$$

$$\text{subject to constraints A-C}$$

where α_{ij} is the total value to both parties of a matching between i and j. We know from linear optimization theory that the optimal solution(s) x^* will lie at corners of the feasible region, which given these constraints will result in a solution in the form of the previously defined incidence vector, which takes values $x^* \in \{0,1\}^{|M| \times |W|}$.

Unfortunately, this linear program is not in the standard form of

$$\begin{aligned} \text{Maximize } & \mathbf{c}^T \mathbf{x} \\ \text{s.t. } & A\mathbf{x} \leq \mathbf{b} \\ & x_i \geq 0 \end{aligned}$$

from which we could easily write down the dual problem as:

$$\begin{aligned} \text{Minimize } & \mathbf{b}^T \mathbf{y} \\ \text{s.t. } & A^T \mathbf{y} \geq \mathbf{c} \\ & y_i \geq 0 \end{aligned}$$

Where we have switched from index notation to vector notation. How can we reform the constraints (A) and (B) to look like standard form? Start by referring to the incidence vector $\mathbf{x}$ by a single index, i.e. $x_{i,j} \to x_k$, where $k = (i-1)N + j$. We can pick out the vector $\mathbf{x}^i$, where $\mathbf{x}^i_j = x_{i,j}$ for a particular man m_i by multiplying $\mathbf{x}$ by the following matrix:

$$A = [\underbrace{\mathbf{e}_1, \mathbf{e}_1, ..., \mathbf{e}_1}_{N}, \underbrace{\mathbf{e}_2, \mathbf{e}_2, ..., \mathbf{e}_2}_{N}, ..., \underbrace{\mathbf{e}_N, \mathbf{e}_N, ..., \mathbf{e}_N}_{N},]$$

Where the basis vector $\mathbf{e}_i \in \mathbb{R}^N$ has all zeros except for the ith row, and $A \in \mathbb{R}^{N \times N^2}$. To be more explicit, the matrix A looks like:

$$A = \begin{bmatrix} 1 & 1 & ... & 1 & 0 & 0 & ... & 0 & 0 & ... \\ 0 & 0 & ... & 0 & 1 & 1 & ... & 1 & 0 & ... \\ 0 & 0 & ... & 0 & 0 & 0 & ... & 0 & 1 & ... \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

where each block of 1's is N elements long.

We can then rewrite constraint (A) as:

$$A\mathbf{x} \leq \mathbf{1}_N$$

where the vector of ones $\mathbf{1}_N = [1, 1, ..., 1] \in \mathbb{R}^N$.

Similarly, we can rewrite constraint (B) using a matrix that picks out the woman vector $x_{i,w}$:

$$B = [\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_N, \mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_N, ...] = [\underbrace{I_N, I_N, ..., I_N}_{N}]$$

Where I_N is the $N \times N$ identity matrix. Now constraint (B) becomes:

$$B\mathbf{x} \leq \mathbf{1}_N$$

We can combine these into a single constraint by stacking the matrices:

$$\begin{bmatrix} A \\ B \end{bmatrix} \mathbf{x} = D\mathbf{x} \leq \mathbf{1}_{2N}$$

Now we can write down the dual of the assignment problem as:

$$\begin{aligned} \text{Minimize } & \mathbf{1}_{2N}^T \mathbf{y} \\ \text{s.t. } & D^T \mathbf{y} \geq \alpha \\ & y_i \geq 0 \end{aligned}$$

We can choose at this point to rewrite the vector $\mathbf{y}$ as

$$\mathbf{y} = \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix}$$

which then changes the dual problem statement to

$$\begin{aligned} \text{Minimize } & \mathbf{1}_N^T \mathbf{u} + \mathbf{1}_N^T \mathbf{v} \\ \text{s.t. } & A^T \mathbf{u} + B^T \mathbf{v} \geq \alpha \\ & u_i, v_j \geq 0 \end{aligned}$$

or, moving back to index notation,

$$\begin{aligned} \text{Minimize } & \sum_{i \in M} u_i + \sum_{j \in W} v_j \\ \text{s.t. } & u_i + v_j \geq \alpha_{ij} \quad \forall (i,j) \in M \times W \\ & u_i, v_j \geq 0 \end{aligned}$$

As discussed previously in class, the dual variables u_i and v_j can be interpreted as node labels in a bipartite graph.

0.3.2 The Marriage Problem

The marriage problem, having two-sided preferences, requires an additional constraint to avoid blocking pairs:

$$\sum_{j>_m w} x_{m,j} + \sum_{i>_w m} x_{i,w} + x_{m,w} \geq 1 \quad \forall (m,w) \in M \times W \tag{D}$$

where the notation $j >_m w \equiv \{j \in W : j >_m w\}$ and $i >_w m \equiv \{i \in M : i >_w m\}$, for example, $j >_m w$ is the set of all women that are more preferable to m than is w. Constraint (D) demands that at least one of three conditions are met:

- m is matched to a woman more desirable than w
- w is matched to a man more desirable than m
- m is matched to w

How does constraint (D) ensure there are no blocking pairs? Consider two matches (m, w') and (m', w), where $w >_m w'$ and $m >_w m'$, so that (m, w) is a blocking pair. Then the pair (m, w) violates constraint (D):

$$\sum_{j>_m w} x_{m,j} + \sum_{i>_w m} x_{i,w} + x_{m,w} = 0 + 0 + 0 = 0$$

because m is not matched to a woman more desirable than w, w is not matched to a man more desirable than m, and m is not matched to w. Therefore, $\exists$ a blocking pair $(m, w) \Rightarrow$ constraint (D) is violated, so

$$\text{Constraint (D) is satisfied } \Rightarrow \nexists \text{ any blocking pairs.}$$

The linear program for the marriage problem (MP) can then be written as

$$\text{Maximize} \sum_{(i,j) \in M \times W} x_{ij}$$

$$\text{subject to constraints A-D}$$

But once again, the constraint (D) is not in standard form. We can go through a similar exercise as before, creating a large single matrix that allows us to combine all constraints into a single constraint.

For instance, consider the first sum in constraint (D):

$$\sum_{j>_m w} x_{m,j}$$

For each man m_i, define a relative ranking indicator matrix $R^i \in \mathbb{R}^{N \times N}$ such that $R^i_{k,l} = 1 \Rightarrow w_k \geq_m w_l$, else $R^i_{k,l} = 0$. That is, $R^i_{k,l}$ indicates whether man m_i prefers woman w_k over w_l. Then form a block matrix $R = \text{diag}(R^1, R^2, ..., R^N) \in \mathbb{R}^{N^2 \times N^2}$, which allows us to write:

$$\sum_{j>_m w} x_{m,j} \geq z, \; \forall (m,w) \rightarrow R\mathbf{x} \geq \mathbf{z}$$

For brevity, and because the remainder of the reduction to standard form is rather tedious, we'll cut right to the chase at this point.

Note that whereas constraints (A) and (B) each introduced N decision variables into the dual assignment problem (which we called u_i and v_j), constraint (D) will introduce an *additional* N^2 decision variables! Further, since the inequality on constraint (D) is $\geq$, we can flip this and carry on by multiplying the inequality through by -1, which will have the effect of bringing in the new decision variables as subtractions in the dual marriage problem.

This results in the following dual problem, with a third decision variable γ introduced to represent the constraints from (D):

$$\begin{aligned}
\text{Minimize } & \sum_{i \in M} \alpha_i + \sum_{j \in W} \beta_j - \sum_{(i,j) \in M \times W} \gamma_{i,j} \\
\text{s.t. } & \alpha_m + \beta_w - \sum_{j <_m w} \gamma_{m,j} - \sum_{i \leq_w m} \gamma_{i,w} \geq 1 \quad \forall (m,w) \in M \times W \\
& u_i, v_j, \gamma_{i,j} \geq 0
\end{aligned}$$

Note that the third term in constraint (D) has been absorbed into the second summation in the dual problem constraint by replacing $<_w$ with $\leq_w$.

0.4 Example

We can work through the stable marriage problem in the linear programming setting by following the example, presented in the spirit of [2].

Consider an $N = 3$ instance of the stable marriage problem. Let the men's P_M and women's P_W preference lists be given by:

$$P_M = \begin{bmatrix} 3 & 1 & 2 \\ 1 & 3 & 2 \\ 3 & 2 & 1 \end{bmatrix} \qquad P_W = \begin{bmatrix} 3 & 1 & 2 \\ 2 & 3 & 1 \\ 3 & 2 & 1 \end{bmatrix}$$

The constraints can be written exhaustively as follows:

$$x_{1,1} + x_{1,2} + x_{1,3} \leq 1 \tag{A1}$$
$$x_{2,1} + x_{2,2} + x_{2,3} \leq 1 \tag{A2}$$
$$x_{3,1} + x_{3,2} + x_{3,3} \leq 1 \tag{A3}$$
$$x_{1,1} + x_{2,1} + x_{3,1} \leq 1 \tag{B1}$$
$$x_{1,2} + x_{2,2} + x_{3,2} \leq 1 \tag{B2}$$
$$x_{1,3} + x_{2,3} + x_{3,3} \leq 1 \tag{B3}$$
$$(x_{1,3}) + (x_{3,1}) + x_{1,1} \geq 1 \tag{D[1,1]}$$
$$(x_{1,1} + x_{1,3}) + (x_{2,2} + x_{3,2}) + x_{1,2} \geq 1 \tag{D[1,2]}$$
$$(0) + (x_{2,3} + x_{3,3}) + x_{1,2} \geq 1 \tag{D[1,3]}$$
$$(0) + (x_{1,1} + x_{3,1}) + x_{2,1} \geq 1 \tag{D[2,1]}$$
$$(x_{2,1} + x_{2,3}) + (0) + x_{2,2} \geq 1 \tag{D[2,2]}$$
$$(x_{2,1}) + (x_{3,3}) + x_{2,3} \geq 1 \tag{D[2,3]}$$
$$(x_{3,2} + x_{3,3}) + (0) + x_{3,1} \geq 1 \tag{D[3,1]}$$
$$(x_{3,3}) + (x_{2,2}) + x_{3,2} \geq 1 \tag{D[3,2]}$$
$$(0) + (0) + x_{3,3} \geq 1 \tag{D[3,3]}$$

And subject to those, we wish to maximize the objective function:

$$\text{Maximize: } p = x_{1,1} + x_{1,2} + x_{1,3} + x_{2,1} + x_{2,2} + x_{2,3} + x_{3,1} + x_{3,2} + x_{3,3}$$

We can follow our nose through a brute-force solution to this problem by first noticing the rather convenient fact that, since man m_3 and woman w_3 both prefer each other to any others, they must be matched according to constraint (D[3,3])! That is, we know $x_{3,3} \geq 1$. Given the non-negativity constraint and constraints (A3) and (B3), we can also immediately identify that $x_{3,1} = x_{3,2} = x_{1,3} = x_{2,3} = 0$, and that $x_{3,3} = 1$. At this point our unused constraints now look like:

$$x_{1,1} + x_{1,2} \leq 1 \tag{A'1}$$
$$x_{2,1} + x_{2,2} \leq 1 \tag{A'2}$$
$$x_{1,1} + x_{2,1} \leq 1 \tag{B'1}$$
$$x_{1,2} + x_{2,2} \leq 1 \tag{B'2}$$
$$x_{1,1} \geq 1 \tag{D'[1,1]}$$
$$(x_{1,1}) + (x_{2,2}) + x_{1,2} \geq 1 \tag{D'[1,2]}$$
$$(0) + (1) + x_{1,2} \geq 1 \tag{D'[1,3]}$$
$$(0) + (x_{1,1}) + x_{2,1} \geq 1 \tag{D'[2,1]}$$
$$(x_{2,1}) + (0) + x_{2,2} \geq 1 \tag{D'[2,2]}$$
$$(x_{2,1}) + (1) \geq 1 \tag{D'[2,3]}$$
$$(1) + (0) \geq 1 \tag{D'[3,1]}$$
$$(0) + (x_{2,2}) \geq 1 \tag{D'[3,2]}$$

From here, everything quickly collapses to find $x_{1,1} = 1$, $x_{2,2} = 1$, and all others equal 0.

We are then left with an incidence vector $x =$[1 0 0 0 1 0 0 0 1], or a matching of $(1, 1)$, $(2, 2)$, and $(3, 3)$. It can be easily verified that this matching is stable, and in fact we know that it *must* be stable since it was constructed using the constraints that guarantee a stable matching.

From the other direction, we can examine the dual for this problem. The dual is not so easy to solve by inspection as the primal was for this contrived case, so here we only display the dual objective and constraints for completeness. We wish to minimize the dual objective function

$$\begin{aligned}\text{Minimize: } d = {} & \alpha_1 + \alpha_2 + \alpha_3 + \beta_1 + \beta_2 + \beta_3 \\ & - \gamma_{1,1} - \gamma_{1,2} - \gamma_{1,3} - \gamma_{2,1} - \gamma_{2,2} - \gamma_{2,3} - \gamma_{3,1} - \gamma_{3,2} - \gamma_{3,3}\end{aligned}$$

subject to:

$$\begin{aligned}\alpha_1 + \beta_1 - (\gamma_{1,2}) - (\gamma_{1,1} + \gamma_{2,1}) &\geq 1 \\ \alpha_1 + \beta_2 - (0) - (\gamma_{1,2}) &\geq 1 \\ \alpha_1 + \beta_3 - (\gamma_{1,1} + \gamma_{1,2}) - (\gamma_{1,3}) &\geq 1 \\ \alpha_2 + \beta_1 - (\gamma_{2,2} + \gamma_{2,3}) - (\gamma_{2,1}) &\geq 1 \\ \alpha_2 + \beta_2 - (0) - (\gamma_{1,2} + \gamma_{2,2} + \gamma_{3,2}) &\geq 1 \\ \alpha_2 + \beta_3 - (\gamma_{2,2}) - (\gamma_{1,3} + \gamma_{2,3}) &\geq 1 \\ \alpha_3 + \beta_1 - (0) - (\gamma_{1,1} + \gamma_{2,1} + \gamma_{3,1}) &\geq 1 \\ \alpha_3 + \beta_2 - (\gamma_{3,1}) - (\gamma_{1,2} + \gamma_{3,2}) &\geq 1 \\ \alpha_3 + \beta_3 - (\gamma_{3,1} + \gamma_{3,2}) - (\gamma_{1,3} + \gamma_{2,3} + \gamma_{3,3}) &\geq 1\end{aligned}$$

References

[1] Alvin E. Roth, Uriel G. Rothblum, and John H. Vande Vate. Stable matchings, optimal assignments, and linear programming. *Math. Oper. Res.*, 18(4):803–828, 1993.

[2] V. J. Sethuraman and C.P. Teo. Linear programming brings marital bliss. *Mathematical Medley*, 25(2):89–91, 1998.