

## CS8451 DESIGN AND ANALYSIS OF ALGORITHMS

### UNIT IV ITERATIVE IMPROVEMENT

The Simplex Method-The Maximum-Flow Problem –Maximum Matching in Bipartite Graphs-The Stable marriage Problem

#### ITERATIVE IMPROVEMENT

The iterative-improvement technique involves finding a solution to an optimization problem by **generating a sequence of feasible solutions** with improving values of the problem's objective function. Each subsequent solution in such a sequence typically involves a small, **localized change** in the previous feasible solution. When no such change improves the value of the objective function, the algorithm returns the last feasible solution as optimal and stops.

Important problems that can be solved exactly by iterative-improvement algorithms include

- Linear programming,
- Maximizing the flow in a network, and
- Matching the maximum possible number of vertices in a graph.

#### The Simplex Method

Linear programming, the general problem of optimizing a linear function of several variables subject to a set of linear constraints:

$$\begin{aligned} &\text{maximize (or minimize)} && c_1x_1 + \cdots + c_nx_n \\ &\text{subject to} && a_{i1}x_1 + \cdots + a_{in}x_n \leq (\text{or } \geq \text{ or } =) b_i \quad \text{for } i = 1, \dots, m \\ &&& x_1 \geq 0, \dots, x_n \geq 0. \end{aligned}$$

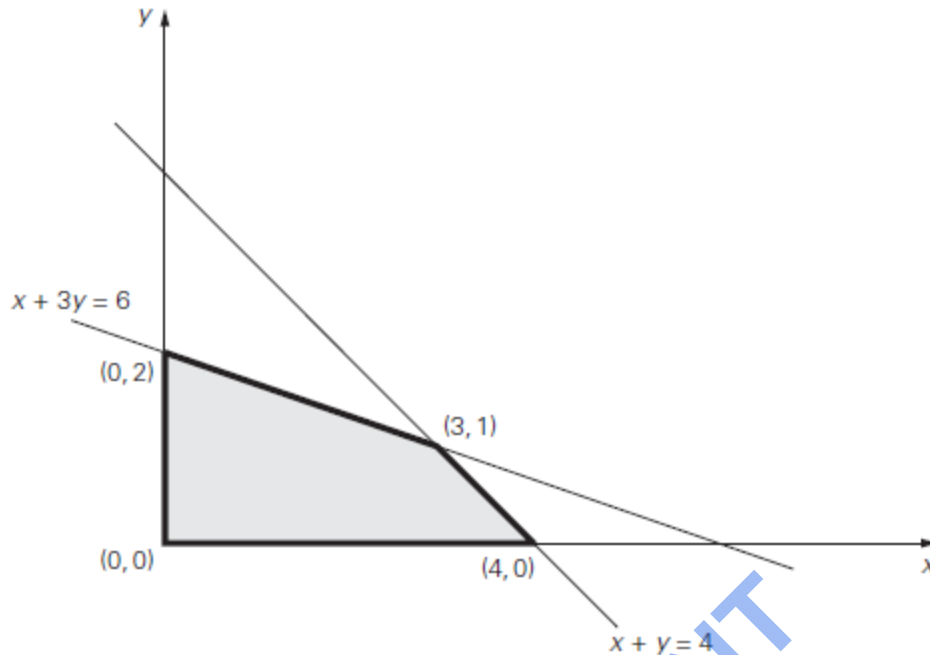
#### Geometric Interpretation of Linear Programming

**Example 1** Consider the following linear programming problem in two variables:

$$\begin{aligned} &\text{Maximize} && 3x + 5y \\ &\text{Subject to} && x + y \leq 4 \\ &&& x + 3y \leq 6 \\ &&& x \geq 0, y \geq 0. \end{aligned}$$

- A **feasible solution** to this problem is any point  $(x, y)$  that satisfies all the constraints of the problem;
- The problem's **feasible region** is the set of all its feasible points.

Any equation  $ax + by = c$ , where coefficients  $a$  and  $b$  are not both equal to zero, defines a straight line. Such a line divides the plane into two half-planes: for all the points in one of them,  $ax + by < c$ , while for all the points in the other,  $ax + by > c$ . In particular, the set of points defined by inequality  $x + y \leq 4$  comprises the points on and below the line  $x + y = 4$ , and the set of points defined by inequality  $x + 3y \leq 6$  comprises the points on and below the line  $x + 3y = 6$ . Since the points of the feasible region must satisfy all the constraints of the problem, the feasible region is obtained by the intersection of these two half-planes and the first quadrant of the Cartesian plane defined by the nonnegativity constraints  $x \geq 0, y \geq 0$  (see Figure ).



Thus, the feasible region for problem is the convex polygon with the vertices  $(0, 0)$ ,  $(4, 0)$ ,  $(0, 2)$ , and  $(3, 1)$ . (The last point, which is the point of intersection of the lines  $x + y = 4$  and  $x + 3y = 6$ , is obtained by solving the system of these two linear equations.)

Our task is to find an **optimal solution**, a point in the feasible region with the largest value of the **objective function**  $z = 3x + 5y$ . The optimal solution to the linear programming problem in question is  $x = 3$ ,  $y = 1$ , with the maximal value of the objective function equal to 14.

### Special Cases in Graphical Method: Linear Programming

The **linear programming problems (LPP)** discussed in the previous section possessed unique solutions. This was because the optimal value occurred at one of the extreme points (corner points). But situations may arise, when the optimal solution obtained is not unique.

#### Case 1: Multiple Optimal Solutions: Graphical Method of Linear Programming

Maximize  $z = x_1 + 2x_2$

subject to

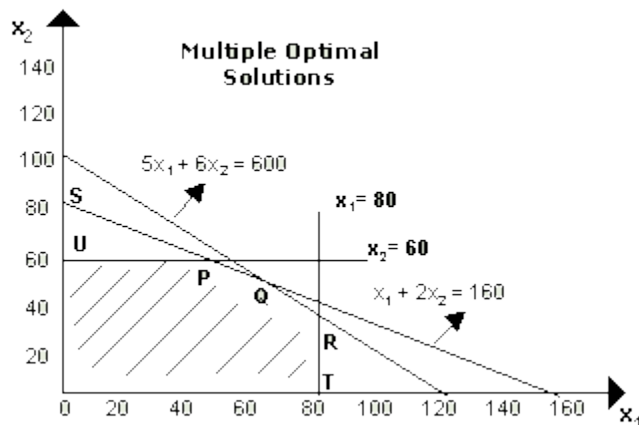
$$x_1 \leq 80$$

$$x_2 \leq 60$$

$$5x_1 + 6x_2 \leq 600$$

$$x_1 + 2x_2 \leq 160$$

$$x_1, x_2 \geq 0.$$



In the above figure, there is no unique outer most corner cut by the objective function line. All points from P to Q lying on line PQ represent optimal solutions and all these will give the same optimal value (maximum profit) of Rs. 160. This is indicated by the fact that both the points P with co-ordinates (40, 60) and Q with co-ordinates (60, 50) are on the line  $x_1 + 2x_2 = 160$ . Thus, every point on the line PQ maximizes the value of the objective function and the problem has multiple solutions.

## 2. Infeasible Problem Linear Programming (LP)

Linear programming problems with the empty feasible region are called *infeasible*. Obviously, infeasible problems do not have optimal solutions. For example, if the constraints include two contradictory requirements, such as  $x + y \leq 1$  and  $x + y \geq 2$ , there can be no points in the problem's feasible region.

In some cases, there is no feasible solution area, i.e., there are no points that satisfy all constraints of the problem. An infeasible LP problem with two decision variables can be identified through its graph. For example, let us consider the following [linear programming](#) problem (LPP).

$$\text{Minimize } z = 200x_1 + 300x_2$$

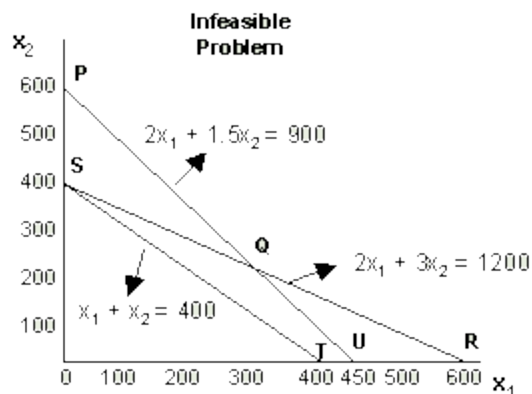
subject to

$$2x_1 + 3x_2 \geq 1200$$

$$x_1 + x_2 \leq 400$$

$$2x_1 + 1.5x_2 \geq 900$$

$$x_1, x_2 \geq 0$$



The region located on the right of PQR includes all solutions, which satisfy the first and the third constraints. The region located on the left of ST includes all solutions, which satisfy the second constraint. Thus, the problem is infeasible because there is no set of points that satisfy all the three constraints.

### 3. Unbounded Solution: Graphical Method in LPP

It is a solution whose objective function is infinite. If the feasible region is unbounded then one or more decision variables will increase indefinitely without violating feasibility, and the value of the objective function can be made arbitrarily large. Consider the following model:

$$\text{Minimize } z = 40x_1 + 60x_2$$

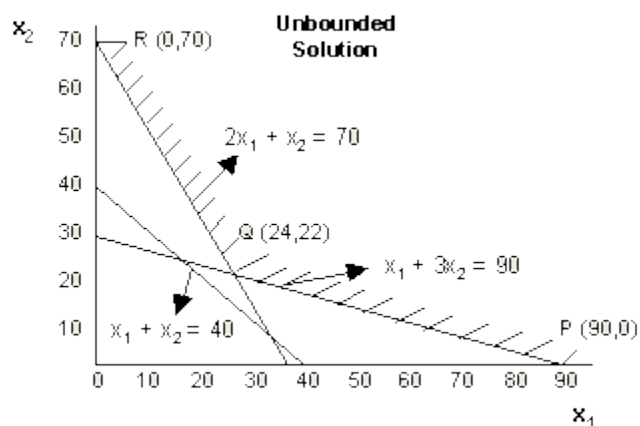
subject to

$$2x_1 + x_2 \geq 70$$

$$x_1 + x_2 \geq 40$$

$$x_1 + 3x_2 \geq 90$$

$$x_1, x_2 \geq 0$$



The point  $(x_1, x_2)$  must be somewhere in the solution space as shown in the figure by shaded portion.

The three extreme points (corner points) in the finite plane are:

$P = (90, 0)$ ;  $Q = (24, 22)$  and  $R = (0, 70)$

The values of the objective function at these extreme points are:

$Z(P) = 3600$ ,  $Z(Q) = 2280$  and  $Z(R) = 4200$

### Limitations of Linear Programming

- **Linearity of relations:** A primary requirement of linear programming is that the objective function and every constraint must be linear. However, in real life situations, several business and industrial problems are nonlinear in nature.
- **Single objective:** Linear programming takes into account a single objective only, i.e., profit maximization or cost minimization. However, in today's dynamic business environment, there is no single universal objective for all organizations.
- **Certainty:** Linear Programming assumes that the values of co-efficient of decision variables are known with certainty.
- Due to this restrictive assumption, linear programming cannot be applied to a wide variety of problems where values of the coefficients are probabilistic.
- **Constant parameters:** Parameters appearing in LP are assumed to be constant, but in practical situations it is not so.
- **Divisibility:** In linear programming, the decision variables are allowed to take non-negative integer as well as fractional values. However, we quite often face situations where the planning models contain integer valued variables. For instance, trucks in a fleet, generators in a powerhouse, pieces of equipment, investment alternatives and there are a myriad of other examples. Rounding off the solution to the nearest integer will not yield an optimal solution. In such cases, linear programming techniques cannot be used.

### Summary of the simplex method

**Step 0 Initialization** Present a given linear programming problem in standard form and set up an initial tableau with nonnegative entries in the rightmost column and  $m$  other columns composing the  $m \times m$  identity matrix. (Entries in the objective row are to be disregarded in verifying these requirements.) These  $m$  columns define the basic variables of the initial basic feasible solution, used as the labels of the tableau's rows.

**Step 1 Optimality test** If all the entries in the objective row (except, possibly, the one in the rightmost column, which represents the value of the objective function) are nonnegative—stop: the tableau represents an optimal solution whose basic variables' values are in the rightmost column and the remaining, nonbasic variables' values are zeros.

**Step 2 Finding the entering variable** Select a negative entry from among the first  $n$  elements of the objective row. (A commonly used rule is to select the negative entry with the largest absolute value, with ties broken arbitrarily.) Mark its column to indicate the entering variable and the pivot column.

**Step 3 Finding the departing variable** For each positive entry in the pivot column, calculate the  $\theta$ -ratio by dividing that row's entry in the rightmost column by its entry in the pivot column. (If all the entries in the pivot column are negative or zero, the problem is unbounded—stop.) Find the row with the smallest  $\theta$ -ratio (ties may be broken arbitrarily), and mark this row to indicate the departing variable and the pivot row.

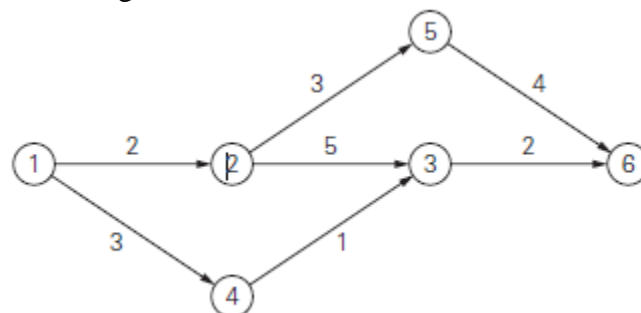
**Step 4 Forming the next tableau** Divide all the entries in the pivot row by its entry in the pivot column. Subtract from each of the other rows, including the objective row, the new pivot row multiplied by the entry in the pivot column of the row in question. (This will make all the entries in the pivot column 0's except for 1 in the pivot row.) Replace the label of the pivot row by the variable's name of the pivot column and go back to Step 1.

### 4.3 THE MAXIMUM-FLOW PROBLEM

The transportation network can be represented by a connected weighted digraph with  $n$  vertices numbered from 1 to  $n$  and a set of edges  $E$ , with the following properties:

- It contains exactly one vertex with no entering edges; this vertex is called the **source** and assumed to be numbered 1.
- It contains exactly one vertex with no leaving edges; this vertex is called the **sink** and assumed to be numbered  $n$ .
- The weight  $u_{ij}$  of each directed edge  $(i, j)$  is a positive integer, called the edge **capacity**. (This number represents the upper bound on the amount of the material that can be sent from  $i$  to  $j$  through a link represented by this edge.)

A digraph satisfying these properties is called a **flow network** or simply a **network**. A small instance of a network is given in Figure.



The total amount of the material entering an intermediate vertex must be equal to the total amount of the material leaving the vertex. This condition is called the **flow-conservation requirement**. If we denote the amount sent through edge  $(i, j)$  by  $x_{ij}$ , then for any intermediate vertex  $i$ , the flow-conservation requirement can be expressed by the following equality constraint:

$$\sum_{j: (j,i) \in E} x_{ji} = \sum_{j: (i,j) \in E} x_{ij} \quad \text{for } i = 2, 3, \dots, n-1,$$

where the sums in the left- and right-hand sides express the total inflow and outflow entering and leaving vertex  $i$ , respectively.

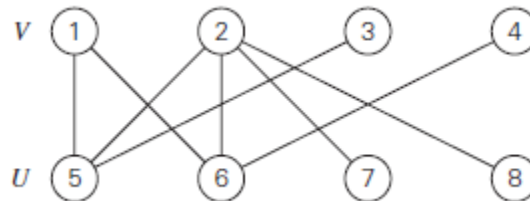
The total amount of the material leaving the source must end up at the sink. Thus, we have the following equality

$$\sum_{j: (1,j) \in E} x_{1j} = \sum_{j: (j,n) \in E} x_{jn}.$$

This quantity, the total outflow from the source—or, equivalently, the total inflow into the sink—is called the **value** of the flow. We denote it by  $v$ . It is this quantity that we will want to maximize over all possible flows in a network.

#### 4.4 MAXIMUM MATCHING IN BIPARTITE GRAPHS

- A **matching** in a graph is a subset of its edges with the property that no two edges share a vertex.
- A **maximum matching (maximum cardinality matching)** is a matching with the largest number of edges.
- The **maximum-matching problem** is the problem of finding a maximum matching in a given graph.
- In a **bipartite graph**, all the vertices can be partitioned into two disjoint sets  $V$  and  $U$ , not necessarily of the same size, so that every edge connects a vertex in one of these sets to a vertex in the other set.
- In other words, a graph is bipartite if its vertices can be colored in two colors so that every edge has its vertices colored in different colors; such graphs are also said to be **2-colorable**. The graph in Figure is bipartite.

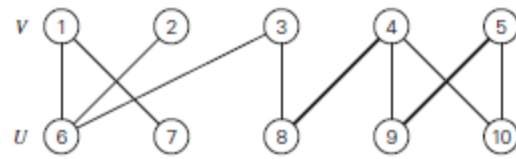


**The iterative-improvement technique to the maximum cardinality- matching problem.**

Let  $M$  be a matching in a bipartite graph  $G = (V, U, E)$ .

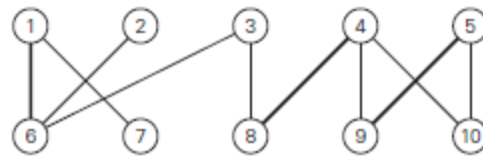
- How can we improve it, i.e., find a new matching with more edges? Obviously, if every vertex in either  $V$  or  $U$  is **matched** (has a **mate**), i.e., serves as an endpoint of an edge in  $M$ , this cannot be done and  $M$  is a maximum matching.
- Therefore, to have a chance at improving the current matching, both  $V$  and  $U$  must contain **unmatched** (also called **free**) **vertices**, i.e., vertices that are not incident to any edge in  $M$ .
- For example, for the matching  $Ma = \{(4, 8), (5, 9)\}$  in the graph in Figure a,
  - Vertices 1, 2, 3, 6, 7, and 10 are free, and
  - Vertices 4, 5, 8, and 9 are matched.
- Another obvious observation is that we can immediately increase a current matching by adding an edge between two free vertices. For example, adding  $(1, 6)$  to the matching  $Ma = \{(4, 8), (5, 9)\}$  in the graph in Figure a yields a larger matching  $Mb = \{(1, 6), (4, 8), (5, 9)\}$  (Figure b).
- Find a matching larger than  $Mb$  by matching vertex 2. The only way to do this would be to include the edge  $(2, 6)$  in a new matching. This inclusion requires removal of  $(1, 6)$ , which can be compensated by inclusion of  $(1, 7)$  in the new matching. This new matching  $Mc = \{(1, 7), (2, 6), (4, 8), (5, 9)\}$  is shown in Figure c.
- Moving further,  $3, 8, 4, 9, 5, 10$  is an augmenting path for the matching  $Mc$  (Figure c). After adding to  $Mc$  the edges  $(3, 8)$ ,  $(4, 9)$ , and  $(5, 10)$  and deleting  $(4, 8)$  and  $(5, 9)$ , we obtain the matching  $Md = \{(1, 7), (2, 6), (3, 8), (4, 9), (5, 10)\}$  shown in Figure d. The matching  $Md$  is not only a maximum matching but also **perfect**, i.e., a matching that matches all the vertices of the graph.

In general, we increase the size of a current matching  $M$  by constructing a simple path from a free vertex in  $V$  to a free vertex in  $U$  whose edges are alternately in  $E - M$  and in  $M$ . That is, the first edge of the path does not belong to  $M$ , the second one does, and so on, until the last edge that does not belong to  $M$ . Such a path is called **augmenting** with respect to the matching  $M$ . For example, the path  $2, 6, 1, 7$  is an augmenting path with respect to the matching  $Mb$  in Figure b. Since the length of an augmenting path is always odd, adding to the matching  $M$  the path's edges in the odd-numbered positions and deleting from it the path's edges in the even-numbered positions yields a matching with one more edge than in  $M$ . Such a matching adjustment is called **augmentation**. Thus, in Figure, the matching  $Mb$  was obtained by augmentation of the matching  $Ma$  along the augmenting path  $1, 6$ , and the matching  $Mc$  was obtained by augmentation of the matching  $Mb$  along the augmenting path  $2, 6, 1, 7$ .



(a)

Augmenting path: 1, 6



(b)

Augmenting path: 2, 6, 1, 7



(c)

Augmenting path: 3, 8, 4, 9, 5, 10



(d)

MR STUDENT

**ALGORITHM** *MaximumBipartiteMatching(G)*

```

//Finds a maximum matching in a bipartite graph by a BFS-like traversal
//Input: A bipartite graph  $G = (V, U, E)$ 
//Output: A maximum-cardinality matching  $M$  in the input graph
initialize set  $M$  of edges with some valid matching (e.g., the empty set)
initialize queue  $Q$  with all the free vertices in  $V$  (in any order)
while not Empty( $Q$ ) do
     $w \leftarrow \text{Front}(Q)$ ; Dequeue( $Q$ )
    if  $w \in V$ 
        for every vertex  $u$  adjacent to  $w$  do
            if  $u$  is free
                //augment
                 $M \leftarrow M \cup (w, u)$ 
                 $v \leftarrow w$ 
                while  $v$  is labeled do
                     $u \leftarrow$  vertex indicated by  $v$ 's label;  $M \leftarrow M - (v, u)$ 
                     $v \leftarrow$  vertex indicated by  $u$ 's label;  $M \leftarrow M \cup (v, u)$ 
                remove all vertex labels
                reinitialize  $Q$  with all free vertices in  $V$ 
                break //exit the for loop
            else //  $u$  is matched
                if  $(w, u) \notin M$  and  $u$  is unlabeled
                    label  $u$  with  $w$ 
                    Enqueue( $Q, u$ )
            else //  $w \in U$  (and matched)
                label the mate  $v$  of  $w$  with  $w$ 
                Enqueue( $Q, v$ )
return  $M$  //current matching is maximum

```

**Efficiency of maximum-matching algorithm**

Each iteration except the last one matches two previously free vertices—one from each of the sets  $V$  and  $U$ . Therefore, the total number of iterations cannot exceed  $\lfloor n/2 \rfloor + 1$ , where  $n = |V| + |U|$  is the number of vertices in the graph. The time spent on each iteration is in  $O(n + m)$ , where  $m = |E|$  is the number of edges in the graph. Hence, the time efficiency of the algorithm is in  $O(n(n + m))$ . The efficiency can be improved to  $O(\sqrt{n(n + m)})$  by combining several iterations into a single stage to maximize the number of edges added to the matching with one search.

**4.5 THE STABLE MARRIAGE PROBLEM**

Consider a set  $Y = \{m_1, m_2, \dots, m_n\}$  of  $n$  men and a set  $X = \{w_1, w_2, \dots, w_n\}$  of  $n$  women. Each man has a preference list ordering the women as potential marriage partners with no ties allowed. (Figure a) Similarly, each woman has a preference list of the men, also with no ties. (Figure b) . The same information can also be presented by an  $n \times n$  ranking matrix (Figure c). The rows and columns of the matrix represent the men and women of the two sets,

respectively. A cell in row  $m$  and column  $w$  contains two rankings: the first is the position (ranking) of  $w$  in the  $m$ 's preference list; the second is the position (ranking) of  $m$  in the  $w$ 's preference list. For example, the pair 3, 1 in Jim's row and Ann's column in the matrix in Figure c indicates that Ann is Jim's third choice while Jim is Ann's first. Which of these two ways to represent such information is better depends on the task at hand. For example, it is easier to specify a match of the sets' elements by using the ranking matrix, whereas the preference lists might be a more efficient data structure for implementing a matching algorithm.

| men's preferences |     |     |     | women's preferences |     |     |     | ranking matrix |            |            |            |
|-------------------|-----|-----|-----|---------------------|-----|-----|-----|----------------|------------|------------|------------|
|                   | 1st | 2nd | 3rd |                     | 1st | 2nd | 3rd |                | Ann        | Lea        | Sue        |
| Bob:              | Lea | Ann | Sue | Ann:                | Jim | Tom | Bob | Bob            | <u>2,3</u> | 1,2        | 3,3        |
| Jim:              | Lea | Sue | Ann | Lea:                | Tom | Bob | Jim | Jim            | 3,1        | <u>1,3</u> | 2,1        |
| Tom:              | Sue | Lea | Ann | Sue:                | Jim | Tom | Bob | Tom            | 3,2        | 2,1        | <u>1,2</u> |
| (a)               |     |     |     | (b)                 |     |     |     | (c)            |            |            |            |

- A **marriage matching**  $M$  is a set of  $n$  ( $m, w$ ) pairs whose members are selected from disjoint  $n$ -element sets  $Y$  and  $X$  in a one-one fashion, i.e., each man  $m$  from  $Y$  is paired with exactly one woman  $w$  from  $X$  and vice versa.
- A pair  $(m, w)$ , where  $m \in Y$ ,  $w \in X$ , is said to be a **blocking pair** for a marriage matching  $M$  if man  $m$  and woman  $w$  are not matched in  $M$  but they prefer each other to their mates in  $M$ . For example, (Bob, Lea) is a blocking pair for the marriage matching  $M = \{(Bob, Ann), (Jim, Lea), (Tom, Sue)\}$  (Figure 10.11c) because they are not matched in  $M$  while Bob prefers Lea to Ann and Lea prefers Bob to Jim.
- A marriage matching  $M$  is called **stable** if there is no blocking pair for it; otherwise,  $M$  is called **unstable**. According to this definition, the marriage matching in Figure 10.11c is unstable because Bob and Lea can drop their designated mates to join in a union they both prefer.
- The **stable marriage problem** is to find a stable marriage matching for men's and women's given preferences. Surprisingly, this problem always has a solution.

#### Stable marriage algorithm

- Input: A set of  $n$  men and a set of  $n$  women along with rankings of the women by each man and rankings of the men by each woman with no ties allowed in the rankings
- Output: A stable marriage matching

**Step 0** Start with all the men and women being free.

**Step 1** While there are free men, arbitrarily select one of them and do the following:

- **Proposal** The selected free man  $m$  proposes to  $w$ , the next woman on his preference list (who is the highest-ranked woman who has not rejected him before).
- **Response** If  $w$  is free, she accepts the proposal to be matched with  $m$ . If she is not free, she compares  $m$  with her current mate. If she prefers  $m$  to him, she accepts  $m$ 's proposal, making her former mate free; otherwise, she simply rejects  $m$ 's proposal, leaving  $m$  free.

**Step 2** Return the set of  $n$  matched pairs.

**THEOREM** The stable marriage algorithm terminates after no more than  $n^2$  iterations with a stable marriage output.

**PROOF** The algorithm starts with  $n$  men having the total of  $n^2$  women on their ranking lists. On each iteration, one man makes a proposal to a woman. This reduces the total number of women to whom the men can still propose in the future because no man proposes to the same woman more than once. Hence, the algorithm must stop after no more than  $n^2$  iterations.

**Application of the stable marriage algorithm:**

|               |     |             |             |             |  |
|---------------|-----|-------------|-------------|-------------|--|
|               |     | Ann         | Lea         | Sue         |  |
| Free men:     | Bob | 2, 3        | <u>1, 2</u> | 3, 3        | Bob proposed to Lea<br>Lea accepted              |
| Bob, Jim, Tom | Jim | 3, 1        | 1, 3        | 2, 1        |  |
|               | Tom | 3, 2        | 2, 1        | 1, 2        |  |
|               |     | Ann         | Lea         | Sue         |  |
| Free men:     | Bob | 2, 3        | <u>1, 2</u> | 3, 3        | Jim proposed to Lea<br>Lea rejected              |
| Jim, Tom      | Jim | 3, 1        | 1, 3        | 2, 1        |  |
|               | Tom | 3, 2        | <u>2, 1</u> | 1, 2        |  |
|               |     | Ann         | Lea         | Sue         |  |
| Free men:     | Bob | 2, 3        | <u>1, 2</u> | 3, 3        | Jim proposed to Sue<br>Sue accepted              |
| Jim, Tom      | Jim | 3, 1        | 1, 3        | <u>2, 1</u> |  |
|               | Tom | 3, 2        | 2, 1        | 1, 2        |  |
|               |     | Ann         | Lea         | Sue         |  |
| Free men:     | Bob | 2, 3        | <u>1, 2</u> | 3, 3        | Tom proposed to Sue<br>Sue rejected              |
| Tom           | Jim | 3, 1        | 1, 3        | <u>2, 1</u> |  |
|               | Tom | 3, 2        | 2, 1        | <u>1, 2</u> |  |
|               |     | Ann         | Lea         | Sue         |  |
| Free men:     | Bob | 2, 3        | 1, 2        | 3, 3        | Tom proposed to Lea<br>Lea replaced Bob with Tom |
| Tom           | Jim | 3, 1        | 1, 3        | <u>2, 1</u> |  |
|               | Tom | 3, 2        | <u>2, 1</u> | 1, 2        |  |
|               |     | Ann         | Lea         | Sue         |  |
| Free men:     | Bob | <u>2, 3</u> | 1, 2        | 3, 3        | Bob proposed to Ann<br>Ann accepted              |
| Bob           | Jim | 3, 1        | 1, 3        | <u>2, 1</u> |  |
|               | Tom | 3, 2        | <u>2, 1</u> | 1, 2        |  |

An accepted proposal is indicated by a boxed cell; a rejected proposal is shown by an underlined cell.

The stable marriage algorithm has a notable **shortcoming**. It is not “gender neutral.” In the form presented above, it favors men’s preferences over women’s preferences. We can easily see this by tracing the algorithm on the following instance of the problem:

|       |         |         |
|-------|---------|---------|
|       | Woman 1 | Woman 2 |
| Man 1 | 1, 2    | 2, 1    |
| Man 2 | 2, 1    | 1, 2    |

The algorithm obviously yields the stable matching  $M = \{(\text{man 1, woman 1}), (\text{man 2, woman 2})\}$ . In this matching, both men are matched to their first choices, which is not the case for the women. One can prove that the algorithm always yields a stable matching that is **man-optimal**: it assigns to each man the highest-ranked woman possible under any stable marriage. Of course,

this gender bias can be reversed, but not eliminated, by reversing the roles played by men and women in the algorithm, i.e., by making women propose and men accept or reject their proposals.

MR STUDENT