UNIT IV ITERATIVE IMPROVEMENT

4.1 THE SIMPLEX METHOD

Linear Programming

Linear programming problem (LPP) is to optimize a linear function of several variables subject to linear constraints:

maximize (or minimize) $c_1 x_1 + ... + c_n x_n$

subject to $a_{i\,1} x_1 + ... + a_{in} x_n \leq (\text{or} \geq \text{or} =)\ b_i,\ i = 1,...,m$ $\qquad x_1 \geq 0, \ ... \ , x_n \geq 0$

The function $z = c_1 x_1 + ... + c_n x_n$ is called the *objective function*;

constraints $x_1 \geq 0, \ ... \ , x_n \geq 0$ are called *nonnegativity constraints*

Example

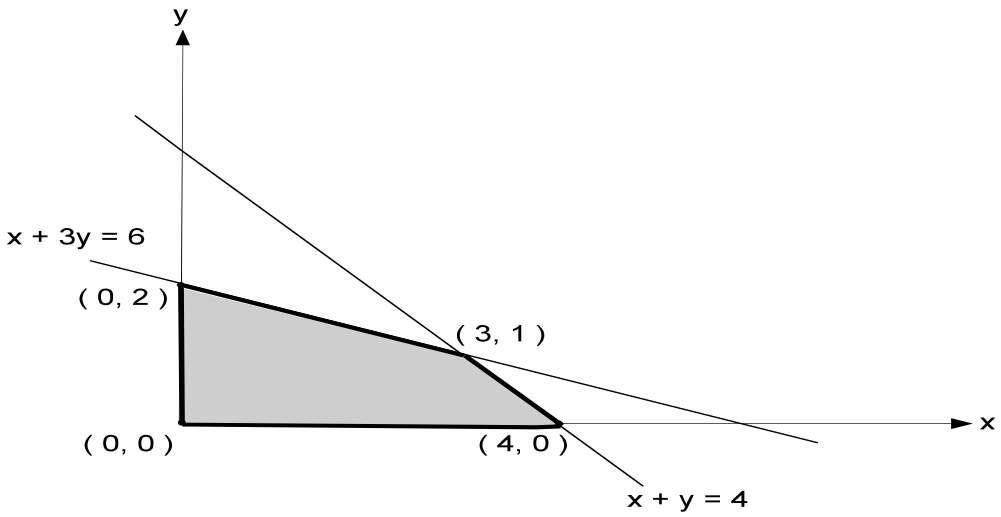
maximize $3x + 5y$

subject to $x + y \leq 4$

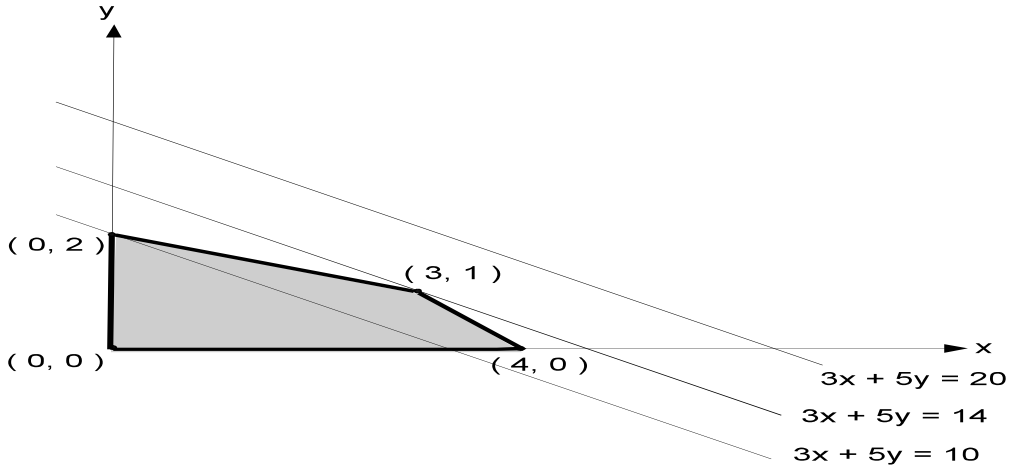
$x + 3y \leq 6$

$x \geq 0,\ y \geq 0$

Feasible region is the set of points defined by the constraints



Geometric solution



Optimal solution: x = 3, y = 1

Extreme Point Theorem Any LP problem with a nonempty bounded feasible region has an optimal solution; moreover, an optimal solution can always be found at an *extreme point* of the problem's feasible region.

Three possible outcomes in solving an LP problem

- has a finite optimal solution, which may not be unique
- *unbounded*: the objective function of maximization (minimization) LP problem is unbounded from above (below) on its feasible region
- *infeasible*: there are no points satisfying all the constraints, i.e. the constraints are contradictory

The Simplex Method

- The classic method for solving LP problems; one of the most important algorithms ever invented.
- Invented by George Dantzig in 1947.
- Based on the iterative improvement idea.
- Generates a sequence of adjacent points of the problem's feasible region with improving values of the objective function until no further improvement is possible.

Standard form of LP problem

- Must be a **maximization** problem
- All constraints (except the nonnegativity constraints) must be in the form of linear equations
- All the variables must be required to be nonnegative
- Thus, the general linear programming problem in standard form with m constraints and n unknowns ($n \geq m$) is
- Maximize $c_1 x_1 + \ldots + c_n x_n$
- Subject to $a_{i\,1} x_1 + \ldots + a_{in} x_n = b_i,\ i = 1,\ldots,m, \qquad x_1 \geq 0, \ldots, x_n \geq 0$

Example

maximize $3x + 5y$

subject to $x + y \leq 4$

$x + 3y \leq 6$

$x \geq 0,\ y \geq 0$

maximize $3x + 5y + 0u + 0v$

subject to $x + y + u = 4$

$x + 3y + v = 6$

$x \geq 0,\ y \geq 0,\ u \geq 0,\ v \geq 0$

Variables u and v, transforming inequality constraints into equality constrains, are called *slack variables*

Basic feasible solutions

A *basic solution* to a system of m linear equations in n unknowns ($n \geq m$) is obtained by setting $n - m$ variables to 0 and solving the resulting system to get the values of the other m variables. The variables set to 0 are called *nonbasic*; the variables obtained by solving the system are called *basic.*

A basic solution is called *feasible* if all its (basic) variables are nonnegative.

Example $x + y + u = 4$

$x + 3y + v = 6$

$(0,\ 0,\ 4,\ 6)$ is basic feasible solution

(x, y are nonbasic; u, v are basic)

There is a 1-1 correspondence between extreme points of LP's feasible region and its basic feasible solutions.

Simplex Tableau

maximize $z = 3x + 5y + 0u + 0v$

subject to $x + y + u = 4$

$x + 3y + v = 6$

$x \geq 0,\ y \geq 0,\ u \geq 0,\ v \geq 0$

objective row

	x	y	u	v	
u	1	1	1	0	4
v	1	3	0	1	6
	−3	−5	0	0	0

basic variables = u,v

basic feasible solution = (0, 0, 4, 6)

value of z at (0, 0, 4, 6) =0

Outline of the Simplex Method

Step 0 [Initialization] Present a given LP problem in standard form and set up initial tableau.

Step 1 [Optimality test] If all entries in the objective row are nonnegative then stop: the tableau represents an optimal solution.

Step 2 [Find entering variable] Select the most negative entry in the objective row. Mark its column to indicate the entering variable and the **pivot column**.

Step 3 [Find departing (leaving) variable] For each positive entry in the pivot column, calculate the θ-ratio by dividing that row's entry in the rightmost column (solution) by its entry in the pivot column. (If there are no positive entries in the pivot column then stop: the problem is unbounded.) Find the row with the smallest θ-ratio, mark this row to indicate the departing variable and the **pivot row.**

Step 4 [Form 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. Replace the label of the pivot row by the variable's name of the pivot column and go back to Step 1.

Example of Simplex Method Application

		x	y	u	v	
	u	1	1	1	0	4
←	v	1	3	0	1	6
		−3	−5	0	0	0

↑

basic feasible sol. (0, 0, 4, 6) $z = 0$

		x	y	u	v	
←	u	$\frac{2}{3}$	0	1	$-\frac{1}{3}$	2
	y	$\frac{1}{3}$	1	0	$\frac{1}{3}$	2
		$-\frac{4}{3}$	0	0	$\frac{5}{3}$	10

↑

basic feasible sol. (0, 2, 2, 0) $z = 10$

		x	y	u	v	
	x	1	0	3/2	−1/3	3
	y	0	1	−1/2	1/2	1
		0	0	2	1	14

basic feasible sol. (3, 1, 0, 0) $z = 14$

Notes on the Simplex Method

- Finding an initial basic feasible solution may pose a problem.
- Theoretical possibility of cycling.
- Typical number of iterations is between m and 3m, where m is the number of equality constraints in the standard form.
- Worse-case efficiency is **exponentia**l.
- More recent interior-point algorithms such as Karmarkar's algorithm (1984) have polynomial worst-case efficiency and have performed competitively with the simplex method in empirical tests.

Example 1:

Use Simplex method to solve the formers problem given below.

A farmer has a 320 acre farm on which she plants two crops: corn and soybeans. For each acre of corn planted, her expenses are $50 and for each acre of soybeans planted, her expenses are $100. Each acre of corn requires 100 bushels of storage and yields a profit of $60; each acre of

soybeans requires 40 bushels of storage and yields a profit of $90. If the total amount of storage space available is 19,200 bushels and the farmer has only $20,000 on hand, how many acres of each crop should she plant in order to maximize her profit? What will her profit be if she follows this strategy?

Solution

Linear Programming Problem Formulation

	Corn	Soybean	**Total**
Expenses	$50	$100	$20,000
Storage(bushels)	100	40	19,200
Profit	60	90	Maximize profit

A farmer has a 320 acre farm is unwanted data but c+s<=320.

c = corn planted acres and s = soybean planted acres

$50c + 100s \leq 20{,}000$

$100c + 40s \leq 19{,}200$

Maximize: $60c + 90s = P$

Canonical form of LPP

Maximize: 60c + 90s

Subject to $50c + 100s = 20000$

$100c + 40s = 19200$

$c \geq 0, s \geq 0$

Solving by algebra (Intersection of lines)

Maximize: $60c + 90s$

Subject to $50c + 100s = 20000$ (1)

$100c + 40s = 19200$ (2)

(1)/50 => $c + 2s = 400$

(2)/20 => $5c + 2s = 960$

(2) – (1) => $4c = 560$

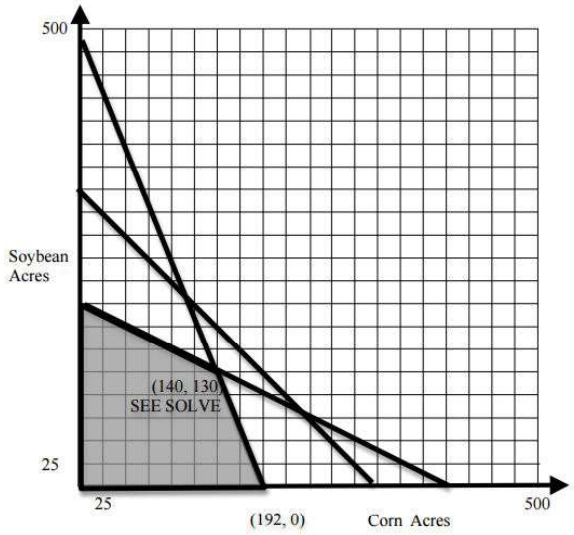
$c = 140$

Substitute c = 140 in (1) then s = 130

Profit: $p = 60c + 90s = 60(140) + 90(130) = \$20{,}100$

She should plant 140 acres corn and 130 acres of soybean for $20,100.

Solving by Graphical method



Profit at (0, 200) = 60c + 90s = 60(0) +90(200) = $18,000

Profit at (192, 0) = 60c + 90s = 60(192) +90(0) = $11,520

Profit at (140, 130) = 60c + 90s = 60(140) +90(130) = $20,100

She should plant 140 acres corn and 130 acres of soybean for $20,100.

Solving by Simplex method

Canonical form of LPP

Maximize: 60x + 90y

Subject to $50x + 100y + s_1 = 20000$

$100x + 40y + s_2 = 19200$

$x \geq 0, y \geq 0$

Iteration I

Basic	z	x	y	s_1	s_2	Solution
s_1	0	50	**100**	1	0	**20000**
s_2	0	100	40	0	1	19200
z	1	-60	**-90**	0	0	0

CPR

Select least ratio θ

Solution/pivot elements

20000/100 =200 √

Select the most negative value in row z.

Pivot element : Intersection of pivot row and pivot column: 100

Basic variables : s_1, s_2, z

Non Basic variables : x, y

Enter variable : y

Leave variable : s_1

Initial solution at $(x, y, s_1, s_2) = (0, 0, 20000, 19200)$

Initial solution z = 0

Pivot row:

Replace the **leaving variable** in basic column with the **entering variable**.

New Pivot Row = Current Pivot Row / Pivot Element

All other rows including z:

New Row = Current Row – (Its Pivot column coefficient)* New Pivot Row

Row y

New Pivot Row = Current Pivot Row / Pivot Element

= (0, 50, 100, 1, 0, 20000) / 100

= $(0, \frac{1}{2}, 1, \frac{1}{100}, 0, 200)$

Row s_2

New Row = Current Row – (Its Pivot column coefficient)* New Pivot Row

= (0, 100, 40, 0, 1, 19200) - (40)*$(0, \frac{1}{2}, 1, \frac{1}{100}, 0, 200)$

= $(0, 80, 0, \frac{-4}{10}, 1, 96)$

Row z

New Row = Current Row – (Its Pivot column coefficient)* New Pivot Row

= (1, -60, -90, 0, 0, 0) - (-90)*$(0, \frac{1}{2}, 1, \frac{1}{100}, 0, 200)$

= (1, -60, -90, 0, 0, 0) + (90)*$(0, \frac{1}{2}, 1, \frac{1}{100}, 0, 200)$

= $(1, -15, 0, \frac{9}{10}, 0, 18000)$

Iteration II

Basic	z	x	y	s_1	s_2	Solution
y	0	1/2	1	1/100	0	200
s_2	0	80	0	-4/10	1	96
z	1	-15	0	9/10	0	18000

NPR

Basic	z	x	y	s_1	s_2	Solution
y	0	1/2	1	1/100	0	200
s_2	0	**80**	0	-4/10	1	11200
z	1	-15	0	9/10	0	18000

CPR

Select least ratio ɵ

Solution/pivot elements

200/(1/2) =400

Select the most negative value in row z.

Pivot element : Intersection of pivot row and pivot column: 80

Basic variables : y, s_2, z

Non Basic variables : x, s_1

Enter variable : x

Leave variable : s_2

Second solution at (x, y, s_1, s_2) = (0, 200, 0, 11200)

Second solution z = 18000 (Improved solution)

Row x

New Pivot Row = Current Pivot Row / Pivot Element

= $(0, 80, 0, \frac{-4}{10}, 1, 11200) / 80$

= $(0, 1, 0, \frac{-1}{200}, \frac{1}{80}, 140)$

Row y

New Row = Current Row – (Its Pivot column coefficient)* New Pivot Row

= (0, 1/2, 1, 1/100, 0, 200) - $(\frac{1}{2})$*(0, 1, 0, $\frac{-1}{200}$, $\frac{1}{80}$, 140)

= (0, 0, 1, $\frac{1}{80}$, $\frac{-1}{160}$, 130)

Row z

New Row = Current Row – (Its Pivot column coefficient)* New Pivot Row

= (1, -15, 0, $\frac{9}{10}$, 0, 18000) - (-15)*(0, 1, 0, $\frac{-1}{200}$, $\frac{1}{80}$, 140)

= (1, -15, 0, $\frac{9}{10}$, 0, 18000) + (15)*(0, 1, 0, $\frac{-1}{200}$, $\frac{1}{80}$, 140)

= (1, 0, 0, $\frac{33}{40}$, $\frac{15}{80}$, 20100)

Iteration III

Basic	z	x	y	s_1	s_2	Solution
y	0	0	1	1/80	-1/160	130
x	0	1	0	-1/200	1/80	140
z	1	0	0	33/40	15/80	20100

The above table has no negative values in row z.

Therefore, the above table is optimum table.

Profit at (140, 130) = 60c + 90s = 60(140) +90(130) = $20,100

Final solution at (x, y, s_1, s_2) = (130, 140, 0, 0)

Final solution z = $20,100 (Optimized solution)

Primal to Dual conversion (Dual to Primal)

[Primal = Dual of Dual]

Primal

Maximize

$$z = \sum_{j=1}^{n} c_j x_j ,$$

subject to:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad (\mathrm{i} = 1, 2, \ldots, \mathrm{m}),$$

$$x_j \geq 0 \quad (\mathrm{j} = 1, 2, \ldots, \mathrm{n}).$$

Dual

Minimize

$$z' = \sum_{i=1}^{m} b_i y_i ,$$

subject to:

$$\sum_{i=1}^{m} a_{ij} y_i \geq c_j \quad (\mathrm{j} = 1, 2, \ldots, \mathrm{n}),$$

$$y_i \geq 0 \quad (\mathrm{i} = 1, 2, \ldots, \mathrm{m}).$$

Example:

The Primal problem

Minimize $4x_1 + 2x_2 - x_3$

subject to $x_1 + x_2 + 2x_3 \geq 3$

$2x_1 - 2x_2 + 4x_3 \leq 5$

$x_1, x_2, x_3 \geq 0.$

The dual problem

Maximize $3y_1 + 5y2$

subject to $y_1 + 2y_2 \leq 4$

$y_1 - 2y_2 \leq 2$

$2y_1 + 4y_2 \leq -1$

$y_1 \geq 0, y_2 \geq 0$

4.2 THE MAXIMUM-FLOW PROBLEM

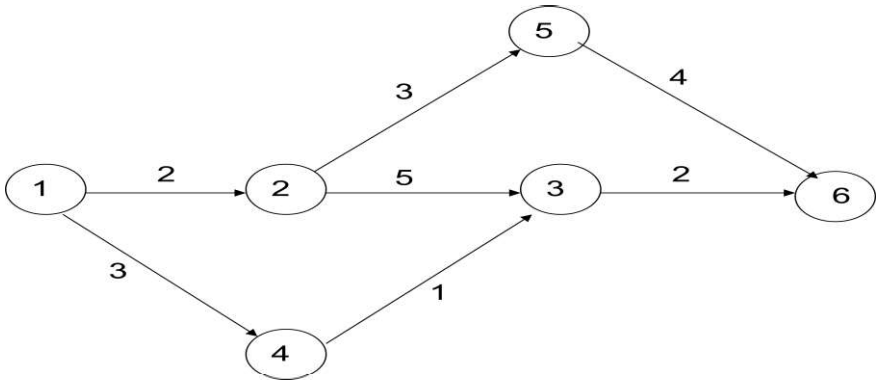
Maximum Flow Problem

Problem of maximizing the flow of a material through a transportation network (e.g., pipeline system, communications or transportation networks)

Formally represented by a connected weighted digraph with n vertices numbered from 1 to n with the following properties:

- Contains exactly one vertex with no entering edges, called the ***source*** (numbered 1)
- Contains exactly one vertex with no leaving edges, called the ***sink*** (numbered n)
- Has positive integer weight u_{ij} on each directed edge (i,j), called the ***edge capacity***, indicating the upper bound on the amount of the material that can be sent from i to j through this edge.
- A digraph satisfying these properties is called a **flow network** or simply a network.

Example of Flow Network



Node (1) = source

Node(6) = sink

Definition of a Flow

A *flow* is an assignment of real numbers x_{ij} to edges (i,j) of a given network that satisfy the following:

- *flow-conservation requirements*

  The total amount of material entering an intermediate vertex must be equal to the total amount of the material leaving the vertex

- *capacity constraints*

$$0 \le x_{ij} \le u_{ij} \quad \text{for every edge } (i,j) \in E$$

Flow value and Maximum Flow Problem

Since no material can be lost or added to by going through intermediate vertices of the network, the total amount of the material leaving the source must end up at the sink:

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

The *value* of the flow is defined as the total outflow from the source (= the total inflow into the sink). The *maximum flow problem* is to find a flow of the largest value (maximum flow) for a given network.

Maximum-Flow Problem as LP problem

Maximize $v = \sum_{j:(1,j)\in E} x_{1j}$

subject to

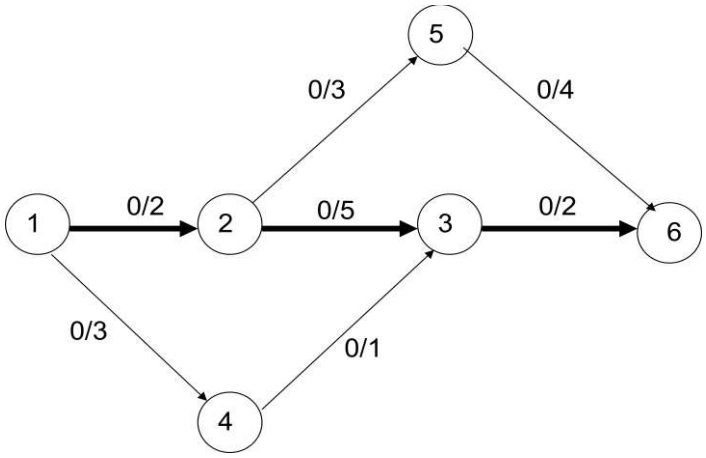
$$\sum_{j:(j,i)\in E} x_{ji} - \sum_{j:(i,j)\in E} x_{ij} = 0 \quad \text{for } i = 2, 3, \ldots, n-1$$

$0 \le x_{ij} \le u_{ij}$ for every edge $(i,j) \in E$

Augmenting Path (Ford-Fulkerson) Method

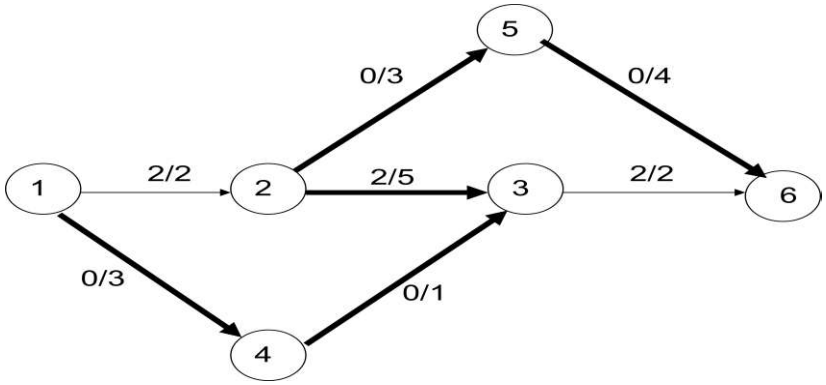
- Start with the zero flow ($x_{ij} = 0$ for every edge).
- On each iteration, try to find a *flow-augmenting path* from source to sink, which a path along which some additional flow can be sent.
- If a flow-augmenting path is found, adjust the flow along the edges of this path to get a flow of increased value and try again.
- If no flow-augmenting path is found, the current flow is maximum.

Example 1



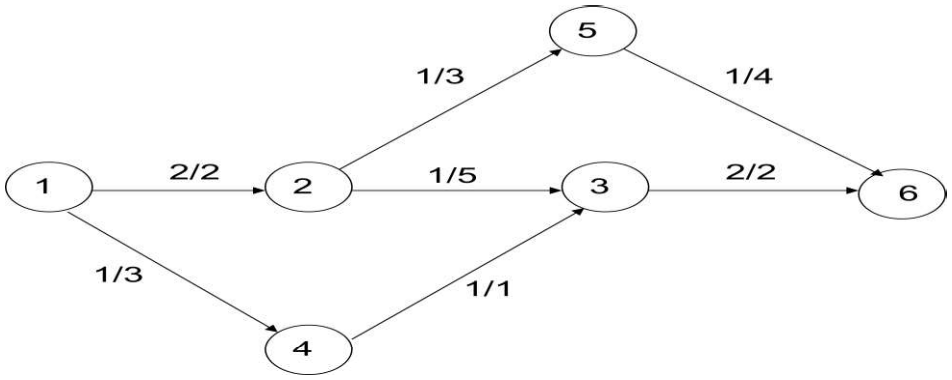
Augmenting path: 1→2 →3 →6

x_{ij}/u_{ij}



Augmenting path: 1 →4 →3←2 →5 →6

Example 1 (maximum flow)



Finding a flow-augmenting path

To find a flow-augmenting path for a flow x, consider paths from source to sink in the underlying undirected graph in which any two consecutive vertices *i*,*j* are either:

- connected by a directed edge (*i* to *j*) with some positive unused capacity $r_{ij} = u_{ij} - x_{ij}$
  - known as *forward edge* (→)

OR

- connected by a directed edge (*j* to *i*) with positive flow x_{ji}
  - known as *backward edge* (←)

If a flow-augmenting path is found, the current flow can be increased by *r* units by increasing x_{ij} by *r* on each forward edge and decreasing x_{ji} by *r* on each backward edge, where

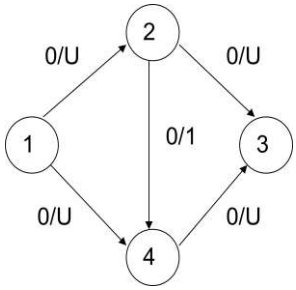
r = min {r_{ij} on all forward edges, x_{ji} on all backward edges}

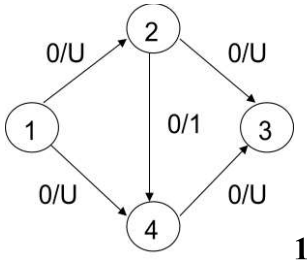
- Assuming the edge capacities are integers, *r* is a positive integer
- On each iteration, the flow value increases by at least 1
- Maximum value is bounded by the sum of the capacities of the edges leaving the source; hence the augmenting-path method has to stop after a finite number of iterations
- The final flow is always maximum, its value doesn't depend on a sequence of augmenting paths used

Performance degeneration of the method

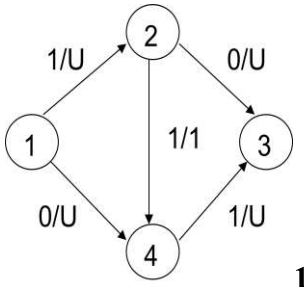
- The augmenting-path method doesn't prescribe a specific way for generating flow-augmenting paths
- Selecting a bad sequence of augmenting paths could impact the method's efficiency

Example 2

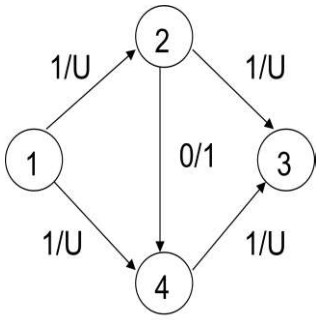




1→2→4→3

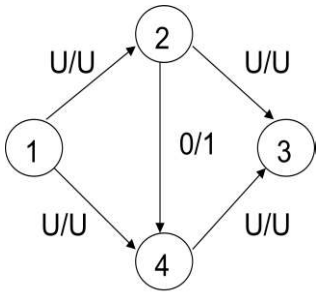


1→4←2→3 **V=1**



V=2

…



V=2U

Requires 2U iterations to reach maximum flow of value 2U

Shortest-Augmenting-Path Algorithm

Generate augmenting path with the least number of edges by BFS as follows.

Starting at the source, perform BFS traversal by marking new (unlabeled) vertices with two labels:

- first label – indicates the amount of additional flow that can be brought from the source to the vertex being labeled
- second label – indicates the vertex from which the vertex being labeled was reached, with "+" or "–" added to the second label to indicate whether the vertex was reached via a forward or backward edge

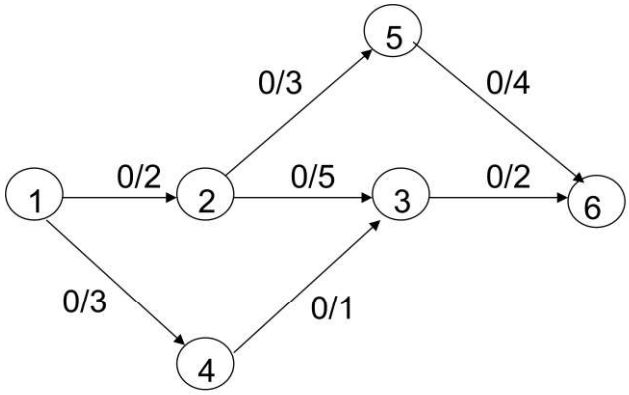
Vertex labeling

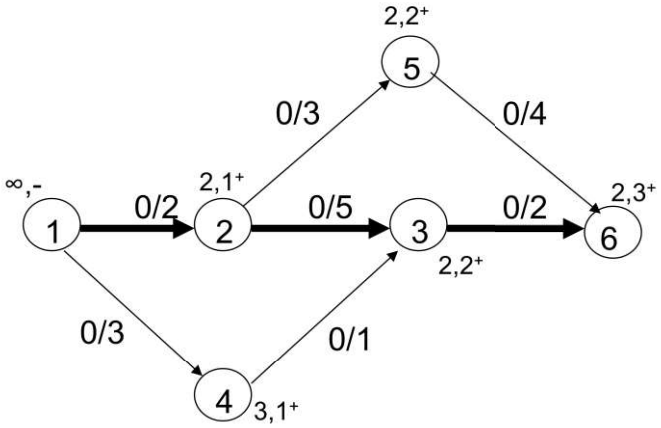
- The source is always labeled with ∞,-
- All other vertices are labeled as follows:
  - If unlabeled vertex j is connected to the front vertex i of the traversal queue by a directed edge from i to j with positive unused capacity $r_{ij} = u_{ij} - x_{ij}$ (forward edge), vertex j is labeled with l_j, i^+, where $l_j = \min\{l_i, r_{ij}\}$

- If unlabeled vertex j is connected to the front vertex i of the traversal queue by a directed edge from j to i with positive flow x_{ji} (backward edge), vertex j is labeled l_j, i^-, where $l_j = \min\{l_i, x_{ji}\}$

- If the sink ends up being labeled, the current flow can be augmented by the amount indicated by the sink's first label.
- The augmentation of the current flow is performed along the augmenting path traced by following the vertex second labels from sink to source; the current flow quantities are increased on the forward edges and decreased on the backward edges of this path.
- If the sink remains unlabeled after the traversal queue becomes empty, the algorithm returns the current flow as maximum and stops.

Example: Shortest-Augmenting-Path Algorithm

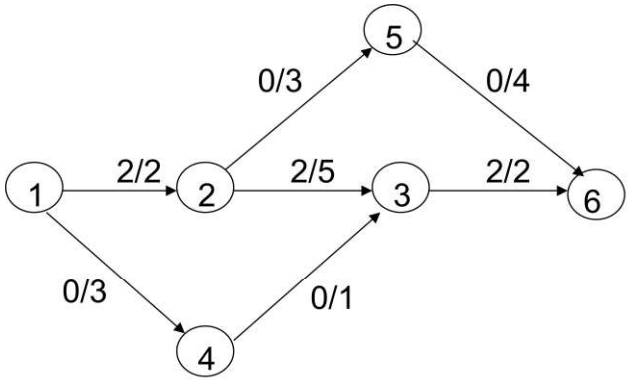


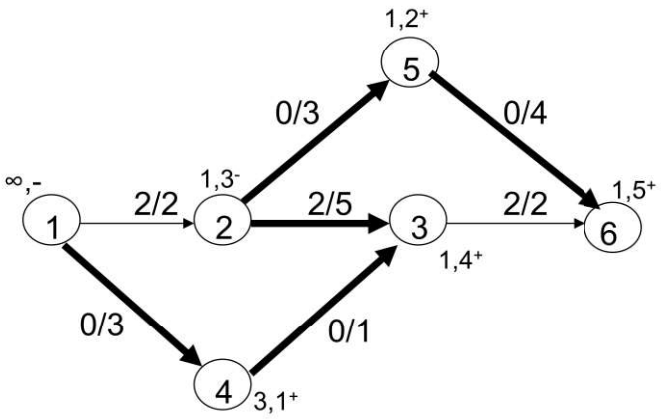


Queue: 1 2 4 3 5 6

↑ ↑ ↑ ↑

Augment the flow by 2 (the sink's first label) along the path 1→2→3→6

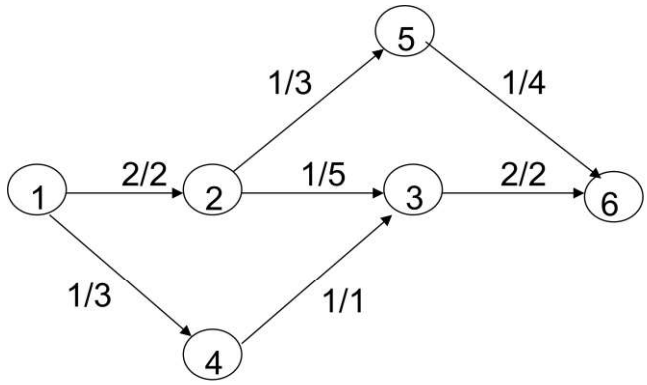


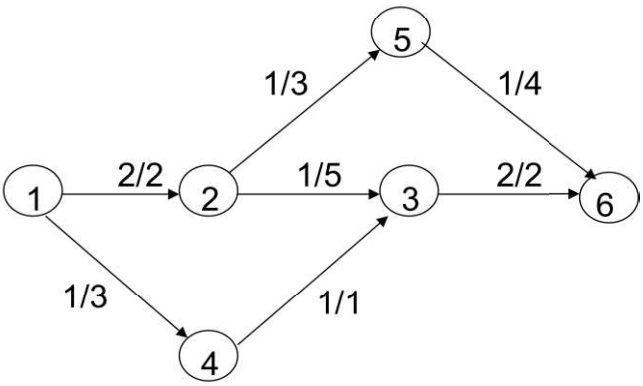


Queue: 1 4 3 2 5 6

↑ ↑ ↑ ↑ ↑

Augment the flow by 1 (the sink's first label) along the path 1→4→3←2→5→6





Queue: 1 4

↑ ↑

No augmenting path (the sink is unlabeled) the current flow is maximum

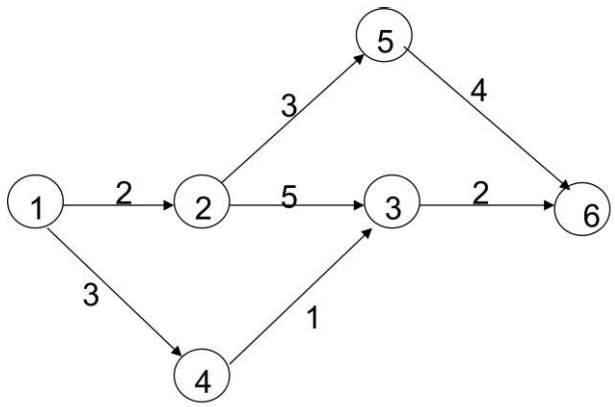
Definition of a Cut

Let X be a set of vertices in a network that includes its source but does not include its sink, and let X, the complement of X, be the rest of the vertices including the sink. The *cut* induced by this partition of the vertices is the set of all the edges with a tail in X and a head in X.

Capacity of a cut is defined as the sum of capacities of the edges that compose the cut.

- We'll denote a cut and its capacity by C(X,X) and c(X,X)
- Note that if all the edges of a cut were deleted from the network, there would be no directed path from source to sink
- *Minimum cut* is a cut of the smallest capacity in a given network

Examples of network cuts



If X = {1} and X = {2,3,4,5,6}, C(X,X) = {(1,2), (1,4)}, c = 5

If X ={1,2,3,4,5} and X = {6}, C(X,X) = {(3,6), (5,6)}, c = 6

If X = {1,2,4} and X ={3,5,6}, C(X,X) = {(2,3), (2,5), (4,3)}, c = 9

Max-Flow Min-Cut Theorem

1. The value of maximum flow in a network is equal to the capacity of its minimum cut
2. The shortest augmenting path algorithm yields both a maximum flow and a minimum cut:
   - Maximum flow is the final flow produced by the algorithm
   - Minimum cut is formed by all the edges from the labeled vertices to unlabeled vertices on the last iteration of the algorithm.
   - All the edges from the labeled to unlabeled vertices are full, i.e., their flow amounts are equal to the edge capacities, while all the edges from the unlabeled to labeled vertices, if any, have zero flow amounts on them.

ALGORITHM *ShortestAugmentingPath(G)*

//Implements the shortest-augmenting-path algorithm
//Input: A network with single source 1, single sink n, and positive integer capacities u_{ij} on
// its edges (i, j)
//Output: A maximum flow x
assign $x_{ij} = 0$ to every edge (i, j) in the network
label the source with $\infty, -$ and add the source to the empty queue Q
while not *Empty(Q)* **do**
  $i \leftarrow$ *Front(Q)*; *Dequeue(Q)*
  for every edge from i to j **do** //forward edges
    if j is unlabeled
      $r_{ij} \leftarrow u_{ij} - x_{ij}$
      if $r_{ij} > 0$
        $lj \leftarrow \min\{l_i, r_{ij}\}$; label j with $l_j, i+$
        Enqueue(Q, j)
  for every edge from j to i **do** //backward edges
    if j is unlabeled
      if $x_{ji} > 0$
        $l_j \leftarrow \min\{l_i, x_{ji}\}$; label j with $l_j, i-$
        Enqueue(Q, j)
  if the sink has been labeled
    //augment along the augmenting path found
    $j \leftarrow n$ //start at the sink and move backwards using second labels
    while $j \neq 1$ //the source hasn't been reached
      if the second label of vertex j is $i+$
        $x_{ij} \leftarrow x_{ij} + l_n$
      else //the second label of vertex j is $i-$
        $x_{ij} \leftarrow x_{ij} - l_n$
      $j \leftarrow i$; $i \leftarrow$ the vertex indicated by i's second label
    erase all vertex labels except the ones of the source
    reinitialize Q with the source
  return x //the current flow is maximum

Time Efficiency

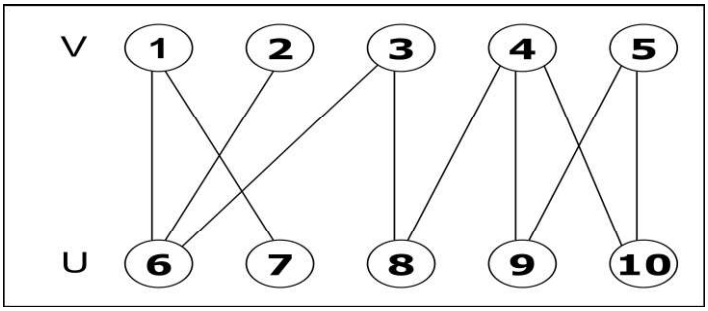
- The number of augmenting paths needed by the shortest-augmenting-path algorithm never exceeds $nm/2$, where n and m are the number of vertices and edges, respectively.
- Since the time required to find shortest augmenting path by breadth-first search is in $O(n+m)=O(m)$ for networks represented by their adjacency lists, the time efficiency of the shortest-augmenting-path algorithm is in $O(nm^2)$ for this representation.
- More efficient algorithms have been found that can run in close to $O(nm)$ time, but these algorithms don't fall into the iterative-improvement paradigm.

4.3 MAXIMUM MATCHING IN BIPARTITE GRAPHS

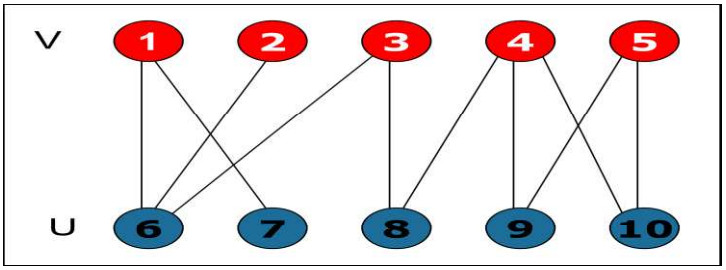
Bipartite Graphs

Bipartite graph: a graph whose 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 V to a vertex in U.

A graph is bipartite if and only if it does not have a cycle of an odd length.

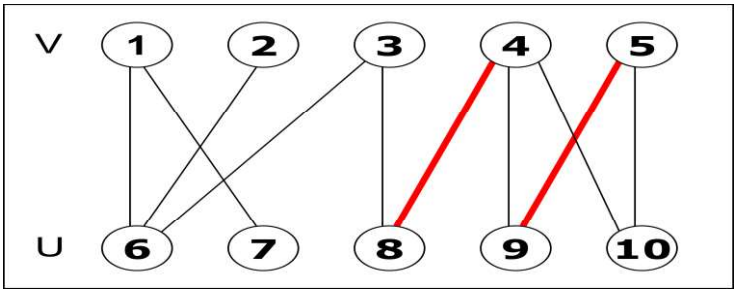


A bipartite graph is *2-colorable*: the vertices can be colored in two colors so that every edge has its vertices colored differently



Matching in a Graph

A *matching* in a graph is a subset of its edges with the property that no two edges share a vertex



a matching in this graph M = {(4,8), (5,9)}

A *maximum* (or *maximum cardinality*) *matching* is a matching with the largest number of edges

- always exists
- not always unique

Free Vertices and Maximum Matching

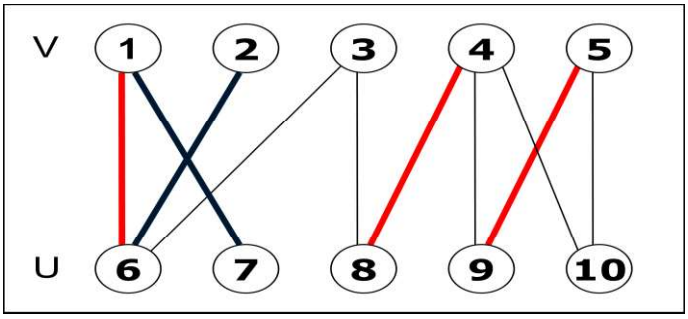
For a given matching M, a vertex is called *free* (or *unmatched*) if it is not an end point of any edge in M; otherwise, a vertex is said to be *matched*

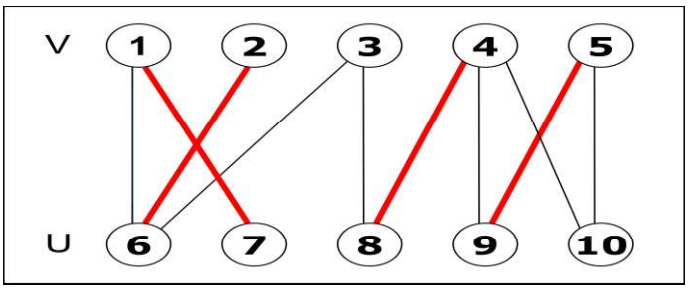
- If every vertex is matched, then M is a maximum matching
- If there are unmatched or free vertices, then M may be able to be improved
- We can immediately increase a matching by adding an edge connecting two free vertices (e.g., (1,6) above)
- Matched vertex = 4, 5, 8, 9. Free vertex = 1, 2, 3, 6, 7, 10.

Augmenting Paths and Augmentation

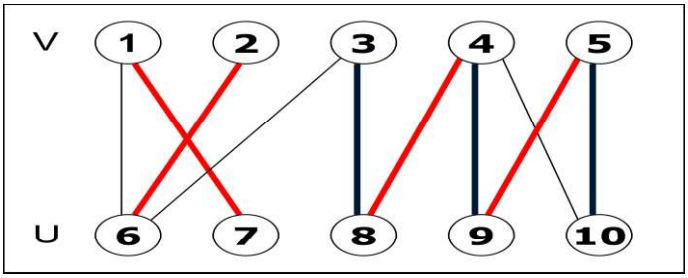
An *augmenting path* for a matching M is a path from a free vertex in V to a free vertex in U whose edges alternate between edges not in M and edges in M

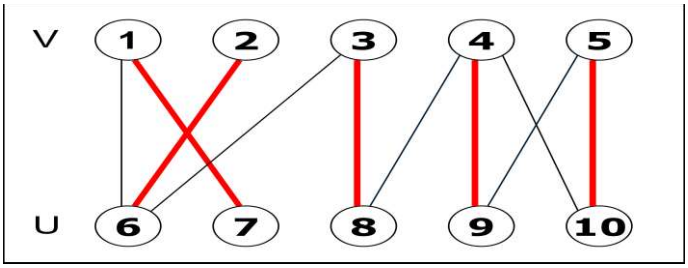
- The length of an augmenting path is always odd
- Adding to M the odd numbered path edges and deleting from it the even numbered path edges increases the matching size by 1 (*augmentation*)
- One-edge path between two free vertices is special case of augmenting path





Augmentation along path 2,6,1,7





Augmentation along 3, 8, 4, 9, 5, 10

Matching on the right is maximum (*perfect* matching).

Theorem: A matching M is maximum if and only if there exists no augmenting path with respect to M.

Augmenting Path Method (template)

- Start with some initial matching . e.g., the empty set
- Find an augmenting path and augment the current matching along that path. e.g., using breadth-first search like method
- When no augmenting path can be found, terminate and return the last matching, which is maximum

4.4 THE STABLE MARRIAGE PROBLEM.

Stable Marriage Problem

- There is a set Y = $\{m_1,\ldots,m_n\}$ of n men and a set X = $\{w_1,\ldots,w_n\}$ of n women. Each man has a ranking list of the women, and each woman has a ranking list of the men (with no ties in these lists).
- A *marriage matching* M is a set of n pairs (m_i, w_j).
- A pair (m, w) is said to be a *blocking pair* for matching M if man m and woman w are not matched in M but prefer each other to their mates in M.
- A marriage matching M is called *stable* if there is no blocking pair for it; otherwise, it's called *unstable*.
- The *stable marriage problem* is to find a stable marriage matching for men's and women's given preferences.

Instance of the Stable Marriage Problem

An instance of the stable marriage problem can be specified either by two sets of preference lists or by a ranking matrix, as in the example below.

men's preferences

	1st	2nd	3rd
Bob:	Lea	Ann	Sue
Jim:	Lea	Sue	Ann
Tom:	Sue	Lea	Ann

women's preferences

	1st	2nd	3rd
Ann:	Jim	Tom	Bob
Lea:	Tom	Bob	Jim
Sue:	Jim	Tom	Bob

ranking matrix

	Ann	Lea	Sue
Bob	2,3	1,2	3,3
Jim	3,1	1,3	2,1
Tom	3,2	2,1	1,2

Data for an instance of the stable marriage problem. (a) Men's preference lists; (b) women's preference lists. (c) Ranking matrix (with the boxed cells composing an unstable matching).

Stable Marriage Algorithm (Gale-Shapley)

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
- *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

Example

Free men: Bob, Jim, Tom

	Ann	Lea	Sue
Bob	2,3	**1,2**	3,3
Jim	3,1	1,3	2,1
Tom	3,2	2,1	1,2

Bob proposed to Lea, Lea accepted Bob

Free men: Jim, Tom

	Ann	Lea	Sue
Bob	2,3	1,2	3,3
Jim	3,1	1,3	2,1
Tom	3,2	2,1	1,2

Jim proposed to Lea, Lea rejected

Free men: Jim, Tom

	Ann	Lea	Sue
Bob	2,3	**1,2**	3,3
Jim	3,1	1,3	**2,1**
Tom	3,2	2,1	1,2

Jim proposed to Sue, Sue accepted

Free men: Tom

	Ann	Lea	Sue
Bob	2,3	**1,2**	3,3
Jim	3,1	1,3	**2,1**
Tom	3,2	2,1	1,2

Tom proposed to Sue, Sue rejected

Free men: Tom

	Ann	Lea	Sue
Bob	2,3	1,2	3,3
Jim	3,1	1,3	**2,1**
Tom	3,2	**2,1**	1,2

Tom proposed to Lea, Lea replaced Bob with Tom

Free men: Bob

	Ann	Lea	Sue
Bob	2,3	1,2	3,3
Jim	3,1	1,3	2,1
Tom	3,2	2,1	1,2

Bob proposed to Ann, Ann accepted

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

Analysis of the Gale-Shapley Algorithm

- The algorithm terminates after no more than n^2 iterations with a stable marriage output.
- The stable matching produced by the algorithm is always *man-optimal*: each man gets the highest rank woman on his list under any stable marriage. One can obtain the *woman-optimal* matching by making women propose to men.
- A man (woman) optimal matching is unique for a given set of participant preferences.
- The stable marriage problem has practical applications such as matching medical-school graduates with hospitals for residency training.