5. Graph Algorithms

Plan :

- 5.1 Introduction and definitions.
- 5.2 Shortest-Path Algorithms.
- 5.3 Minimum Spanning Trees.
- 5.4 Backtracking and Branch-and-bound.
- 9.5 Network Flow Problems.

Introduction and definitions

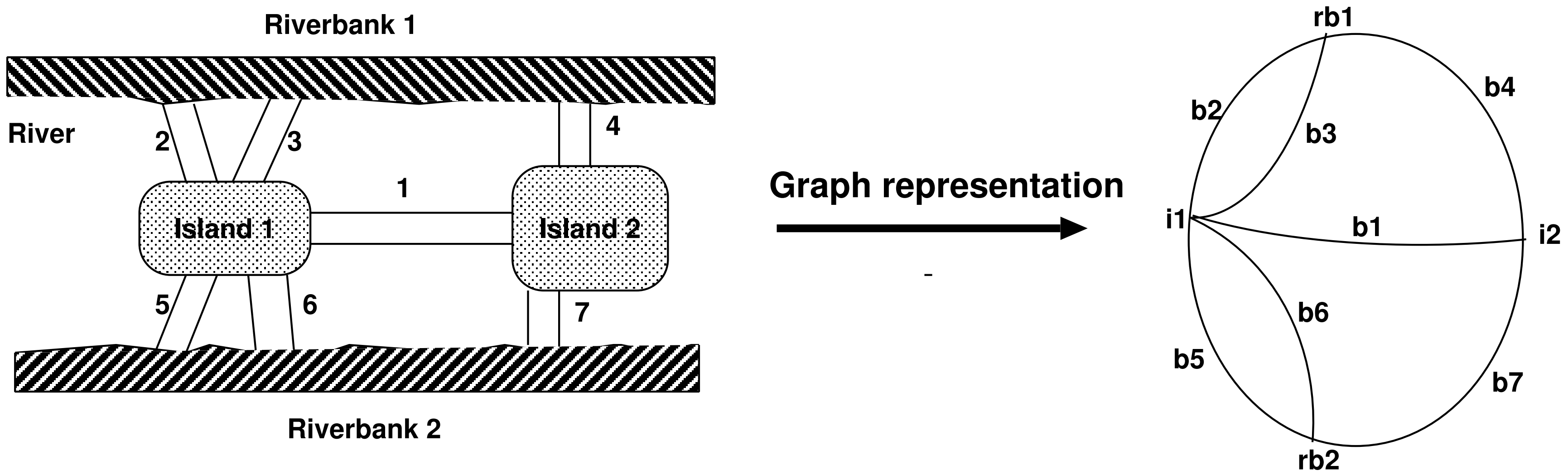


Figure 1: The bridges of Koenigsberg problem

Introduction and definitions

- $Graph = (X, U)$
- Direct graph.
- Rooted graph.
- *tip of leaf* node.
- Path, ancestor, descendant, cycle.
- Tree.

9.2 Shortest-Path Algorithms

- **Weighted graph** : A cost $C_{i,j}$ is associated with each edge (v_i, v_j).
- **Weighted path length :** The cost of a path $v_1 v_2 \ldots v_N$ is equal to $\sum_{i=1}^{N-1} c_{i,i+1}$.
- **Unweighted path length :** the number of edges on the path.
- **Single source shortest-path problem :** Given as input a weighted graph, $G = (V, E)$, and a distinguished vertex, s, find the shortest (weighted or unweighted) path from S to every other vertex in G.
- **All-Pairs Shortest-Path problem :** Find the shortest paths between all pairs of vertices in the graph.

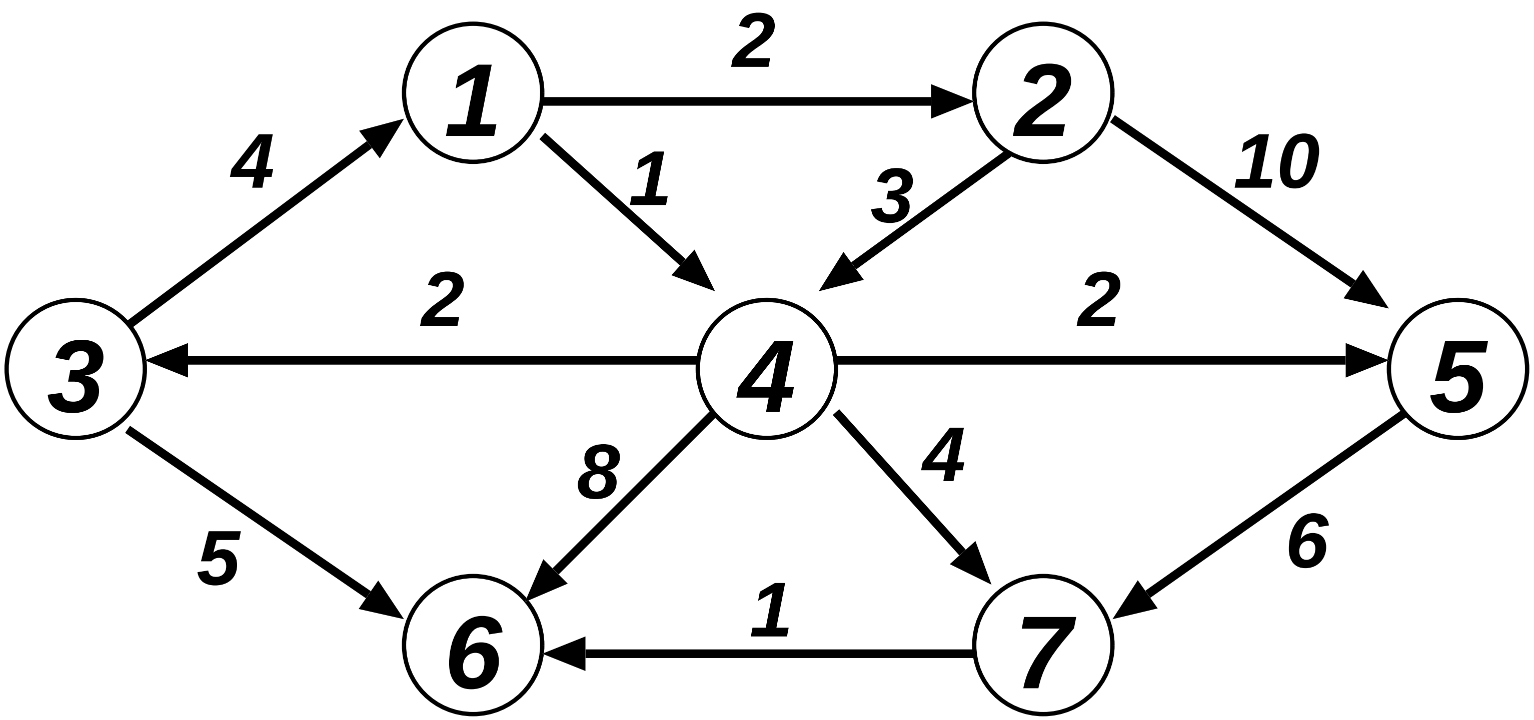


Figure 2: A weighted graph.

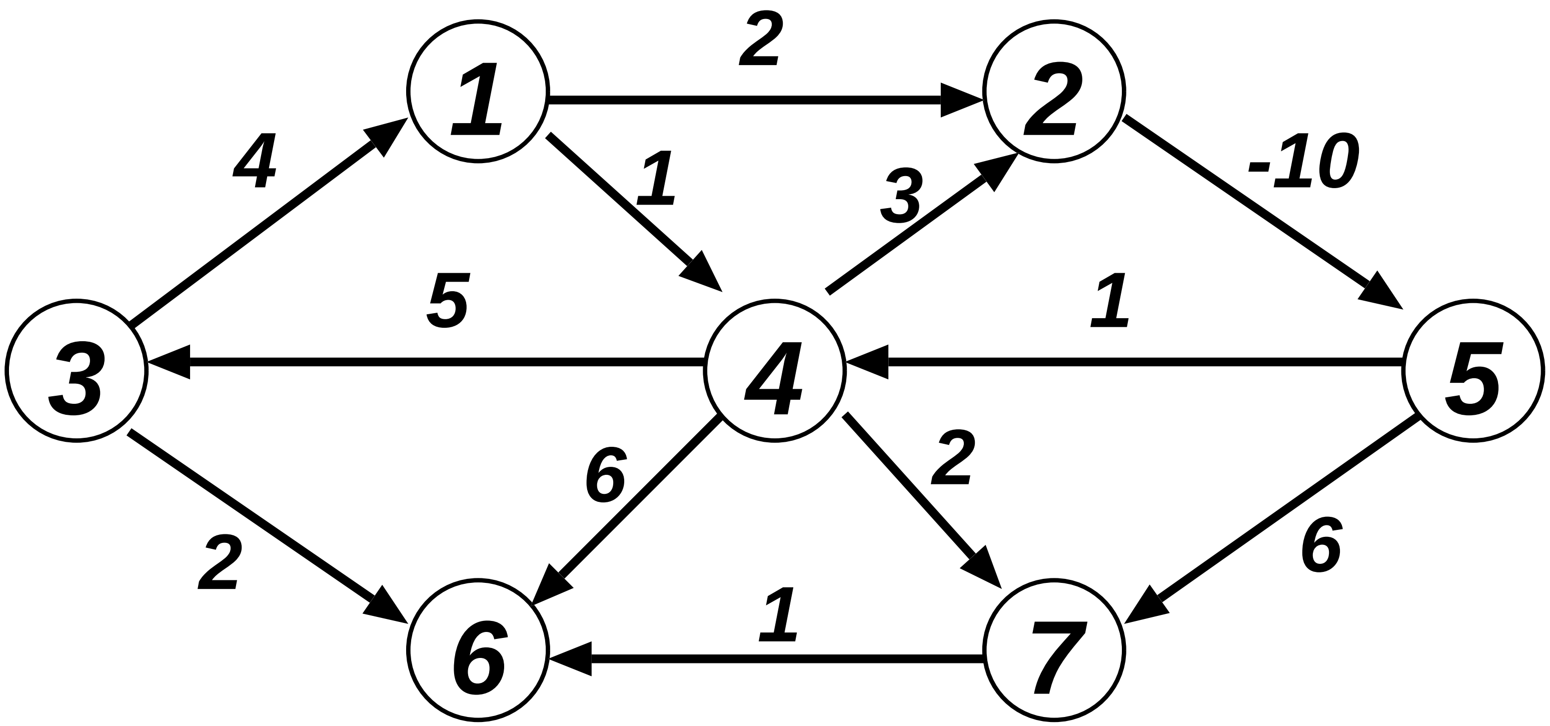


Figure 3: A graph with a negative-cost cycle.

Dijkstra's Algorithm

- Dijkstra's algorithm is based on the greedy approach.
- Each vertex is marked as either *known* or *unknown*. A tentative distance d_v is kept for each vertex. This distance turns out to be the shortest path length from s to v using only *known* vertices as intermediates.
- Dijksta's algorithm proceeds in stages :
  - **Greedy criterion :** At each stage, it selects a vertex v, which has the smallest d_v among the *unknown* vertices, and declares that the shortest path from s to v as *known*.
  - The remainder of a stage consists of updating the values of d_w (w is a successor of v).

Dijkstra's Algorithm

1. S = $\{x_0\}$; d_0 = 0;

   If $\{x_0, x_i\}$ is an arc **Then** d_i = v_{0i}

   else d_i = +$\propto$

2. **While** S $\neq$ X **Do** (X is a set of all vertices)

   choose $x_i \in$ X $-$ S such that d_i = $min_{x_k \in X-S}(d_k)$

   S $\leftarrow$ S $\bigcup$ $\{x_i\}$

   For all $x_j \in$ (X $-$ S) $\bigcap$ $\Gamma(x_i)$**Do**

   d_j = min(d_i + v_{ij}, d_j)

   ($\Gamma(x_i)$: successor of x_i)

   EndFor

   EndWhile (when S = X)

Analysis of Dijkstra's Algorithm

- The first phase of the algorithm requires $O(N)$.
- The second phase is repeated N times. At each time there is $n + 1 + n$ operations.
- Total complexity : $O(N^2)$.
- Can we do better ?
  - *To read the different data we need* $O(N^2)$ *(max number of arcs)*.

Ford's Algorithm

1. Vertices are labeled : $x_0, x_1 \ldots x_{n-1}$. a_{ij} is the cost of arc (x_i, x_j).
   x_0 is the root.

2. **Initial step**

   Assign to x_0 the "weight" $d_0 = 0$, and label the other vertices in a given order and assign them the weight d_i = +$\propto$ (0 $\leq$ i $\leq$ n $-$ 1)

3. **Explore all arcs**

   Explore all nodes $x_0, x_1 ... x_{n-1}$ in order :

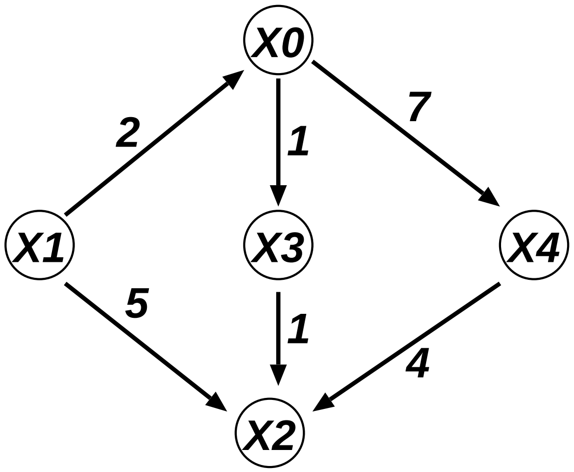
   for each explored node x_i, consider all arcs (x_i, x_j) and set d_j to $d_i + a_{ij}$ if $d_j > d_i + a_{ij}$

4. **Terminating condition**

   While d_j is modified repeat (3).

Ford's Algorithm (cont.)

- The performace of the algorithm depends on the labeling used. In the following figure, if we swap nodes x_2 and x_3, the solution will be obtained without backtrack.



- If there is no path from x_0 to x_i, we will have d_i = $+\propto$.

- How can we modify the algorithm if we want the longest path?

All-Pairs Shortest Path

- Djikstra's algorithm provides the idea for the dynamic programming algorithm.
- We will select the vertices in sequential order. We will define $D_{k,i,j}$ to be the weight of the shortest path from v_i to v_j that uses only $v_1, v_2, \ldots, v_k$ as intermediates.
- By this definition, $D_{0,i,j} = c_{i.j}$, where $c_{i,j}$ is ∞ if (v_i, v_j) is not an edge in the graph.
- By definition, $D_{n,i,j}$ is the shortest path from v_i to v_j in the graph.

$$D_{k,i,j} = min\{D_{k-1,i,j}, D_{k-1,i,k} + D_{k-1,k,j}\}$$

9.3 Minimum Spanning Tree

- A **minimum spanning tree** of an undirect weighted graph G is a tree formed from graph edges that connects all vertices of G at lowest total cost.
- A minimum spanning tree exists if and only if G is connected.
- How to compute the minimum spanning tree ?
  - Use the greedy method. 3 Algorithms :
    - Kruskal's algorithm
    - Prim's algorithm
    - Sollin's algorithm

Kruskal's algorithm

- Kruskal's algorithm selects the $n-1$ edges one at a time using the following greedy criterion :
  - *from the remaining edges, select a least-cost edge that does not result in a cycle when added to the set of already selected edges.*

Kruskal's algorithm

$G = (X.U)$

$G' = (X', U')$ where $X' = X, U' = \emptyset$

Sort all arcs $\in U$ in incresing order.

For each arc $u \in U$ **Do**

If U' $\bigcup \{$u$\}$ does not contain a cycle **Then**

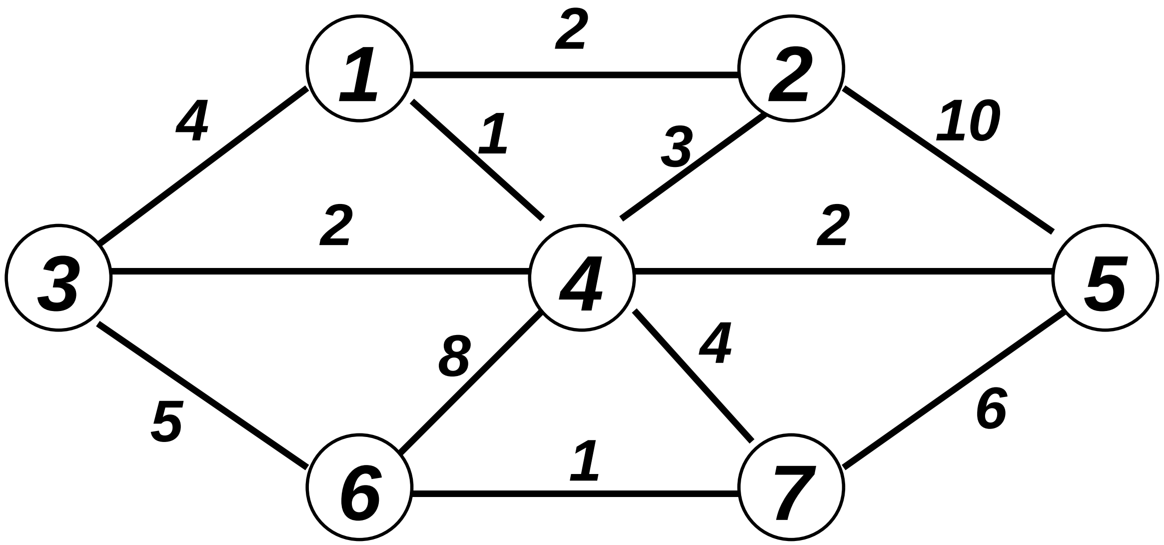
$U' \leftarrow U' \bigcup \{u\}$

EndIf

EndFor

Kruskal's algorithm

Using Kruskal's algorithm, find the minimum spanning tree of the following graph.



Kruskal's algorithm

Choice od data structure and analysis of the algorithm

- To select edges in nondecresing order of cost, we can set up a min heap and extract edges one by one as needed. When there are e edges in the graph, it takes $\Theta(e)$ time to initialize the heap and $O(\log e)$ time to extract each edge.

Prim's algorithm

- Prim's algorithm, like Kruskal's, constructs the minimum spanning tree by selecting edges one at a time.
- The greedy criterion used to determine the next edge to select is :
  - *from the remaining edges, select a least-cost edge whose addition to the selected edges forms a tree.*
- Consequently, at each stage the set of selected edges forms a tree.
- By contrast, the set of selected edges in Kruskal's algorithm forms a forest at each stage.

Prim's algorithm

- This algorithm consists in looking for an acyclic subgraph with a minimal cost.

  Initial step : G' contains one vertice.

  For i=2 to numbre of vertices of G **Do**

  look for an edge of minimal cost that

  connects a vertice of G' to another vertice $\in G - G'$

  and add it to G'

  EndFor

- Using Prim's algorithm, find the minimum spanning tree of the previous example.

- How can we implement Prim's algorithm in order to have a time complexity of $O(N^2)$?

Sollin's Algorithm

- Sollin's algorithm selects several edges at each stage.
- At the start of a stage, the selected edges together with the n vertices of the graph form a spanning forest.
- During a stage we select one edge for each tree in this forest. This edge is a minimum-cost edge that has exactly one vertex in the tree. The selected edges are added to the spanning tree being constructed.
- When several edges have the same cost, two trees can select different edges that connect them, in this case we must discard one of the selected edges.
- At the start of the firxt stage, the set of selected edges is empty.
- The algorithm terminates when only one tree remains at the end of a stage or when no edges remain to be selected.

9.4 Backtracking and Branch and Bounds

Traveling Salesperson

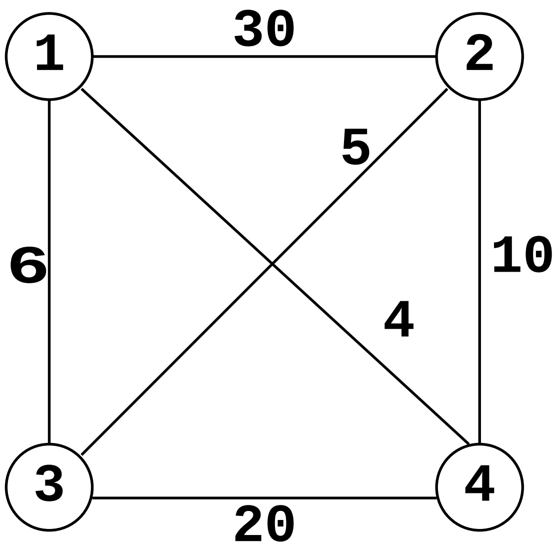


Figure 4: A 4-vertex network

Traveling Salesperson

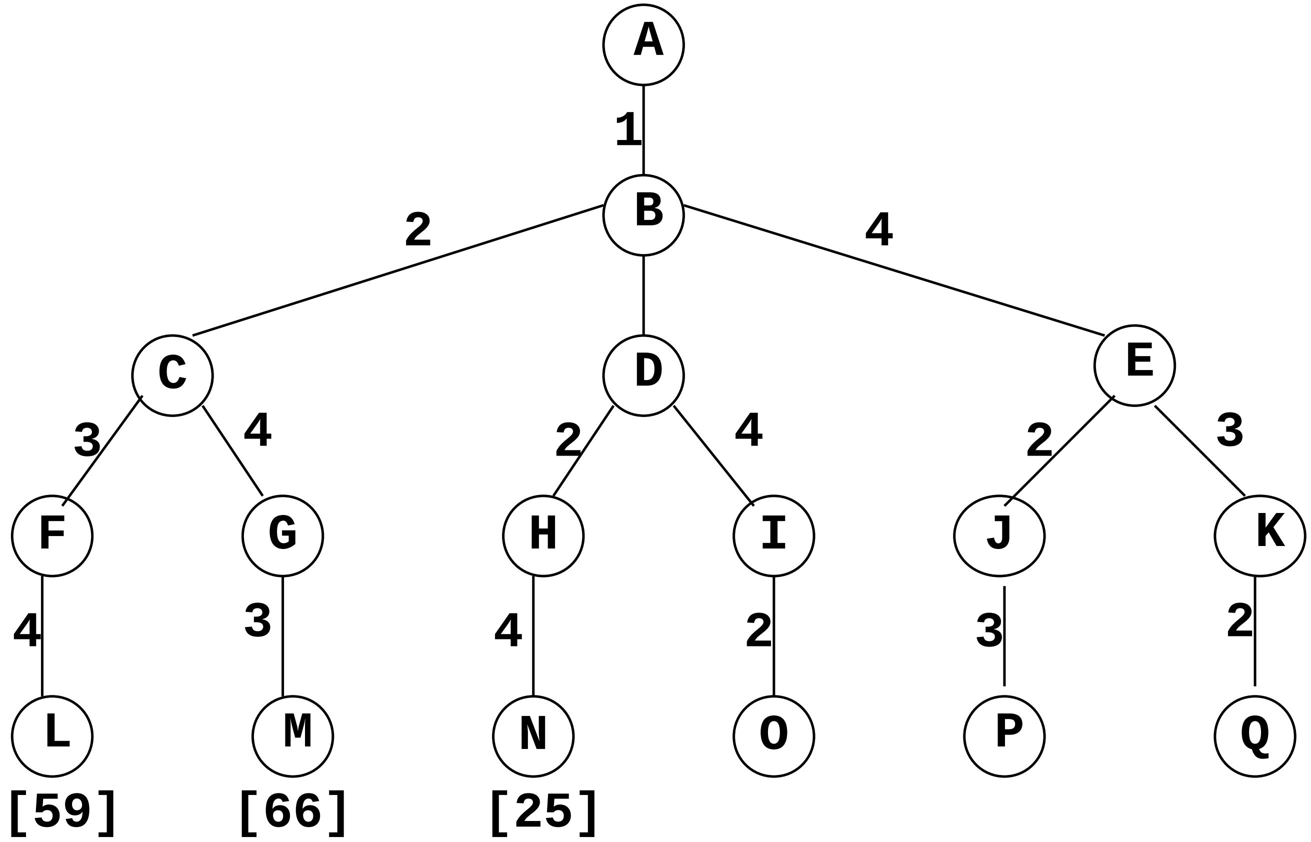


Figure 5: Solution space tree for a 4-vertex network

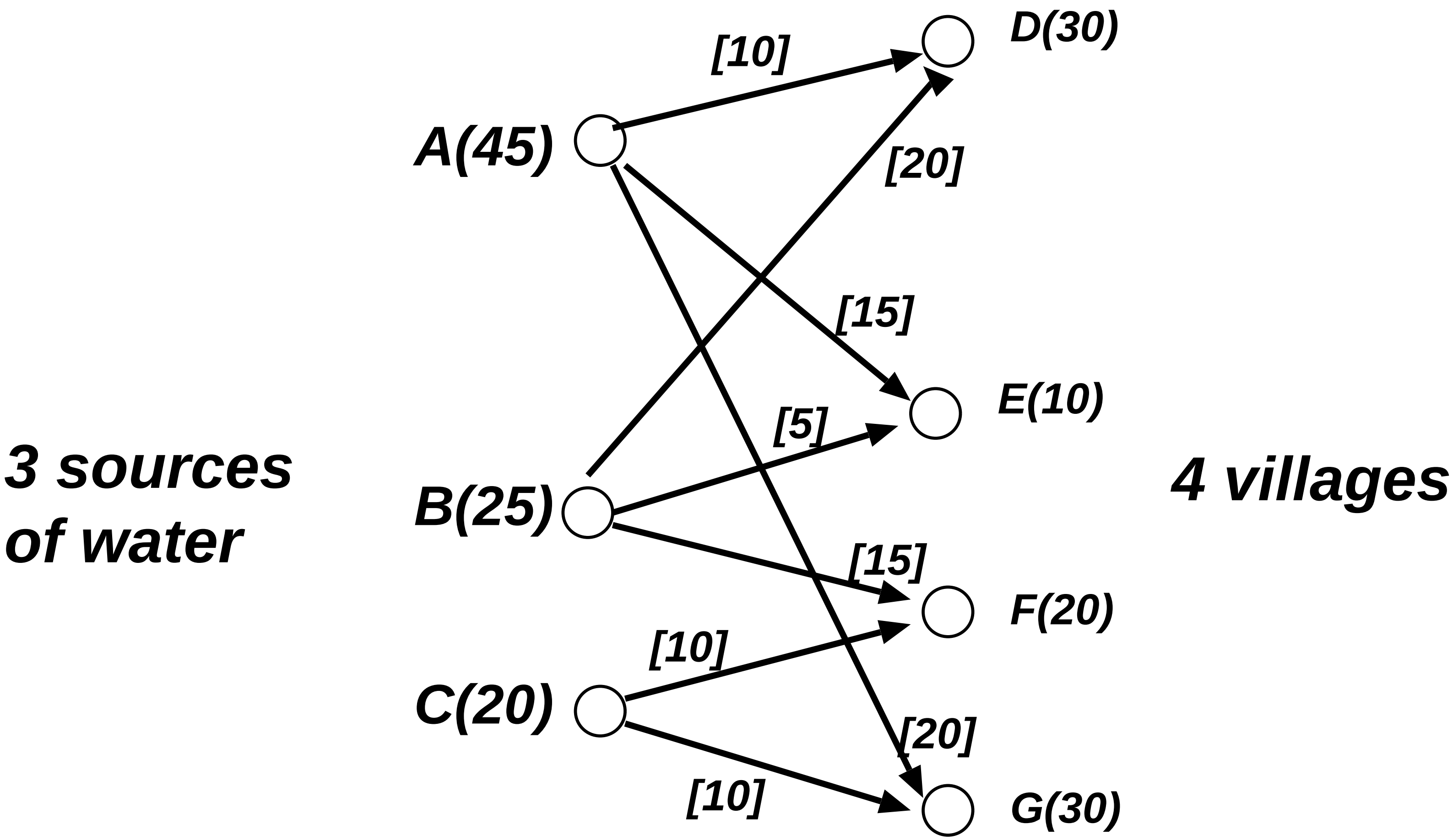
Traveling Salesperson

Formulation of the problem :

- $x_{ij} = 1$ if the edge $i \rightarrow j$ exists and 0 otherwise.
- Objective function to maximize : $\sum_i \sum_j d_{ij} x_{ij}$
- if N is the number of nodes, there are $N!$ possibilities to consider.

9.5 Network Flow Problems

Example



9.5 Network Flow Problems

A network flow is a direct graph without cycle where :

- each edge u has a capacity $c(u) \geq 0$,
- 2 particular nodes **source** and **sink**, the first one has no predecessors while the second one has no successors.

A flow φ of a network is an amount $\varphi(u)$ associated to each edge u such that :

- $\varphi(u) \geq 0, u \in U$.
- $\Sigma_{u \in U_x^-} \varphi(u) - \Sigma_{u \in U_x^+} \varphi(u) = 0$ **if** $x \neq source, x \neq sink$.
- $\varphi(u) \leq c(u)$.

U_x^- : the set of edges (i, x).

U_x^+ : the set of edges (x, j).

9.5 Network Flow Problems : Ford-Fulkerson's Algorithm

The goal of the algorithm is to look for a maximum flow from **source** to **sink**.

1. Look for an initial complete flow (flow such that each path from **source** to **sink** includes at least one saturated (filled) edge).

2. Improve the initial flow :

   (a) Look for an improving chain by using Ford-Fulkerson's labeling.

      i. Mark (by $+i$) the vertice j of any non saturated edge (i, j) such that i is marked.

      ii. Mark (by $-l$) the vertice k of any edge (k, l) having a non empty flow and such that l is not marked.

   (b) If we cannot mark the **sink** then we have obtained a maximum flow.

   (c) Otherwise, complete the marked chain and goto (a).