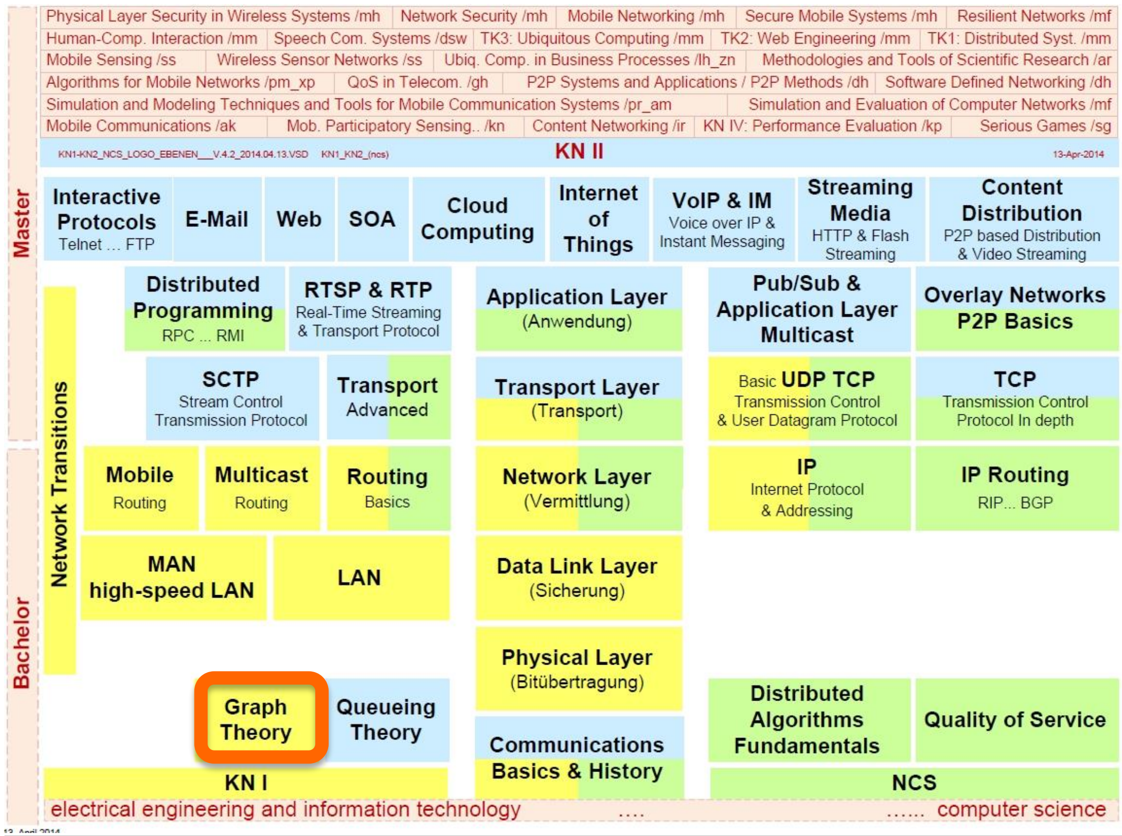
Communication Networks I

Graph Theory for Communication Networks



Prof. Dr.-Ing. Ralf Steinmetz
KOM - Multimedia Communications Lab

Overview

Preliminary Remarks

Graph Theory may be known (to some extend)

Goal:

- To focus on communication network issues of graphs
- To show some “relationships”

Contents

- From Tanenbaum
  - Computer Networks
- From other sections at
  - former KN1
- From NCS
  - Prof. Max Mühlhäuser et.al.

Graphs are a widely used abstraction in many fields

- Electrical engineering: networks, electrical design,…
- Civil engineering: Road maps, pipeline networks
- Chemistry: Molecular structures
- Economics: Organizational structures (organograms)
- And of course computer networks :-)

Graph theory used both in design and optimization

- Many important problems can be modeled as graphs

Popularity of graph theory has increased recently

- Big reason: Computers become more powerful and can solve large optimization problems (modeled and solved as graphs)

Graph theory increasingly important in networking

Graphs often used to model computer networks

- Either networks of single computers or networks of networks

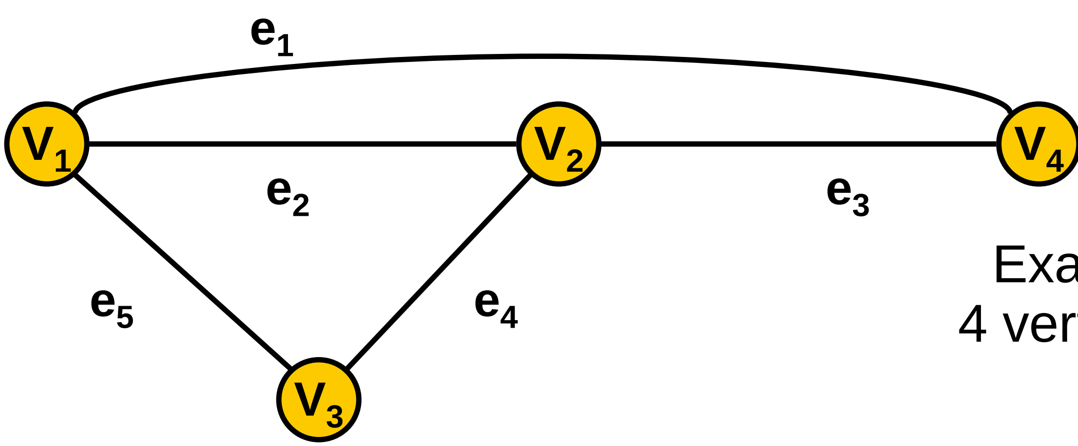
Many basic algorithms from graph theory well-known

- For example, Dijkstra’s Shortest Path Algorithm

Examples of uses of graphs in networking:

- Model routing of messages in network
- Model capacity of network for quality of service
- Analyze network topology
  - How network could be optimized?
  - Insight into possible vulnerabilities
- …

What is a Graph?



Example graph with 4 vertices and 5 edges

Definition of a graph:

Graph *G = (V, E)* consists of two finite sets,

set *V* of vertices (nodes) and

set *E* of edges (arcs)

$E = \{ \{u,v\} \mid u,v \in V\}$

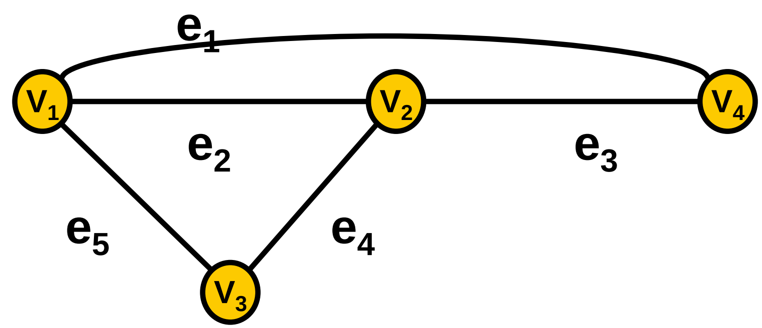
for which the following conditions apply:

1) If $e \in E$, then exists a tuple $(v,u) \in V \times V$, such that $v \in e$ and $u \in e$

2) If $e \in E$ and above (v,u) exists, and further for $(x,y) \in V \times V$ applies $x \in e$ and $y \in e$, then $\{v,u\} = \{x,y\}$

Properties of Graphs

An edge $e \in E$ is undirected
if e = (x,y) is identical to e = (y,x)



An edge $e \in E$ is directed if
e = (x,y) is NOT identical to e = (y,x):
(x,y) ≠ (y,x)

A graph *G* is directed (undirected)
if the above property holds for all edges

A loop is an edge with identical endpoints (e = (x,y) | x = y)

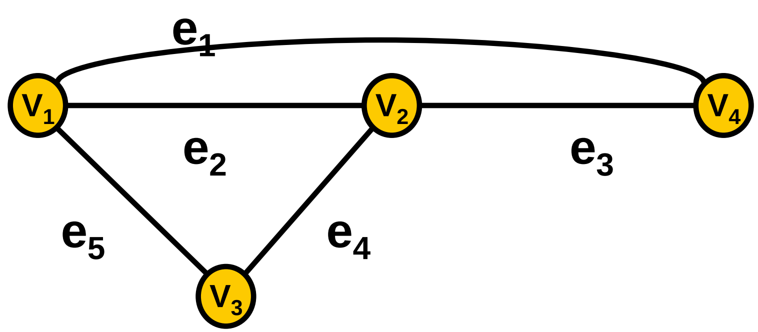
Graph *G* is linear **(simple)**
if for each $(v, u) \in V \times V$ exists at most one $e \in E$ and *G* has no loops

- To linearize a graph,
  replace all multiple edges with a single edge and remove all loops
- *All subsequent graphs are assumed linear, unless otherwise said*

Graph $G_1 = (V_1, E_1)$ is a subgraph of $G = (V, E)$,
if $V_1 \subseteq V$ and $E_1 \subseteq E$ (such that conditions 1 and 2 are met)

TECHNISCHE UNIVERSITÄT DARMSTADT

We want to go from vertex v_1 to vertex v_k



No restrictions: This is called a walk

$(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k) \in E$

A walk is a trail if we go along each *edge* at most once

A trail is a path if we visit each *vertex* at most once

Walk, path, or trail is *closed* if $v_1 = v_k$

A closed path is also called cycle

- In a linear, undirected graph, a cycle has at least 3 edges, why?

If a graph has no cycles, it is called acyclic

Important Types of Graphs

Vertices $v, u \in V$ are connected if there is a path from v to u: $(v, v_2), (v_2, v_3), \ldots, (v_{k-1}, u) \in E$

Graph G is connected if all $v, u \in V$ are connected

Undirected, connected, acyclic graph is called a tree

- Sidenote: Undirected, acyclic graph which is not connected is called a forest

Directed, connected, acyclic graph is also called DAG

- DAG = directed, acyclic graph (connected is “assumed”)

For us, most graphs are connected and undirected

- For example, in computer networks all nodes can talk to each other and traffic flows in both directions

TECHNISCHE UNIVERSITÄT DARMSTADT

Drawing a graph is easy way to visualize it

But this works only for very small graphs

- Real problems are far too big to visualize
- Also, a computer is not very good at reading drawings…

Several different representations appropriate for networks

1. Incidence matrix
2. Adjacency matrix
3. Incidence lists (for sparse graphs)
   - Incidence lists similar to how sparse matrices are represented

Most appropriate form depends on

- the graph and on
- the application/problem

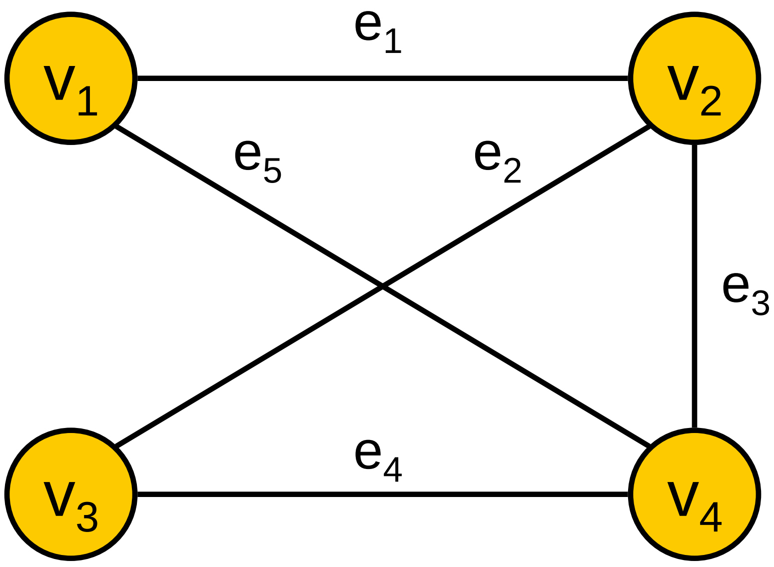
Sample Graph

Consider the following graph $G = (V, E)$

$V = \{v_1, v_2, v_3, v_4\}, |V| = 4$

$E = \{e_1, e_2, e_3, e_4, e_5\}, |E| = 5$

Graph *G* looks like this:



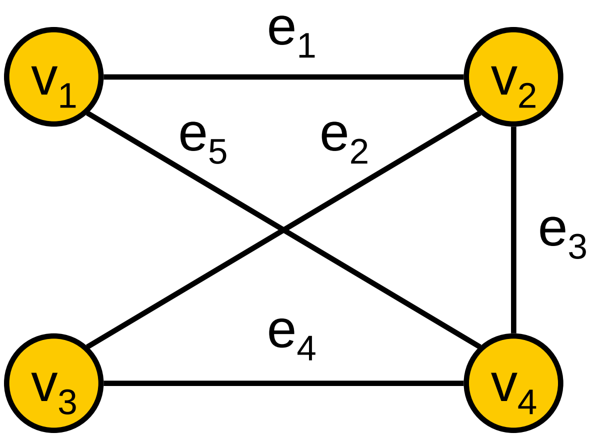
Incidence Matrix

Incidence matrix is a |*V*| *x* |*E*| matrix

It tells which edges are incident to a vertex (= touch it)

- Edge *e* = *(v, u)* is **incident** to both *v* and *u*

$$\text{Vertices}\left\{ I = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}}_{\text{Edges}} \right.$$



Note: Each column has two 1's (no loops allowed)

useful for determining degree of vertex (see below)

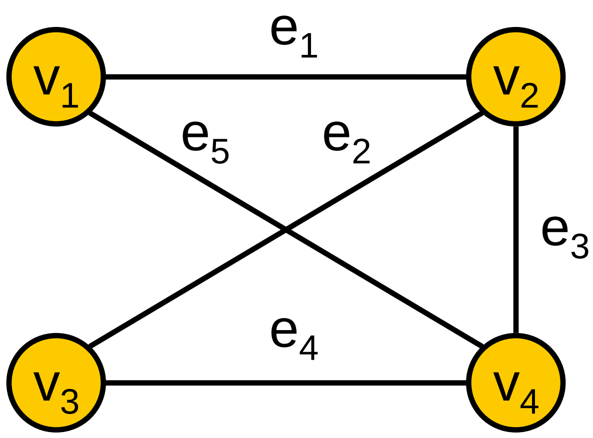
Adjacency Matrix

Adjacency matrix is a |*V*| *x* |*V*| matrix

It tells which vertices are adjacent to each other

- Vertices v and u are adjacent if *(v, u)* $\in E$ or *(u, v)* $\in E$
- Adjacent vertices also called neighbors
- By definition: Vertex is not adjacent to itself

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



Adjacency matrix for undirected graph is symmetric

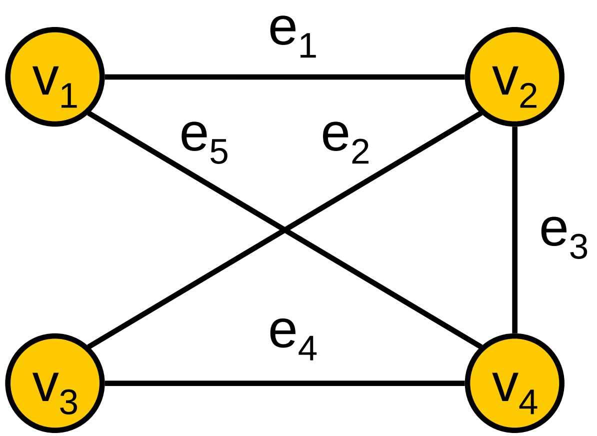
Typically, adjacency matrix is much smaller than incidence matrix and more widely used

Two kinds of incidence lists:

- **Vertex incidence list:**
  Which edges are incident to a vertex
- **Edge incidence list:**
  Which are the endpoints of an edge

Vertex	Edges
v_1	e_1, e_5
v_2	e_1, e_2, e_3
v_3	e_2, e_4
v_4	e_3, e_4, e_5

Edge	Endpoints
e_1	v_1, v_2
e_2	v_2, v_3
e_3	v_2, v_4
e_4	v_3, v_4
e_5	v_1, v_4



Incidence lists especially good for sparse graphs

- Sparse graph has very few edges
- Sparse = far fewer than $n(n - 1)/2$, where $n = |V|$
- List is more efficient in computer than matrix

3 Graph Metrics

Distance d_{ij}	**number of edges along shortest path between node i and node j**
Diameter D **of network**	**max. distance between any 2 nodes**
Average path length L **of network**	**mean distance over all nodes also known as size of an network**
Total amount of nodes N_{ALL}	
Degree k_i **of a node i**	**total number of edges, the node i is attached to**
Av. degree $\langle k \rangle$ **of an network**	**average of all k_i of an network**
Degree Distribution $P(k)$	**probability distribution of k_i**

Vertex Degree

In graph G = (V, E),
the degree of vertex v ∈ V is
the total number of edges (v, u) ∈ E and (u, v) ∈ E

- Degree is the number of edges which touch a vertex

For directed graph, we distinguish between in-degree and out-degree

- In-degree is number of edges coming to a vertex
- Out-degree is number of edges going away from a vertex

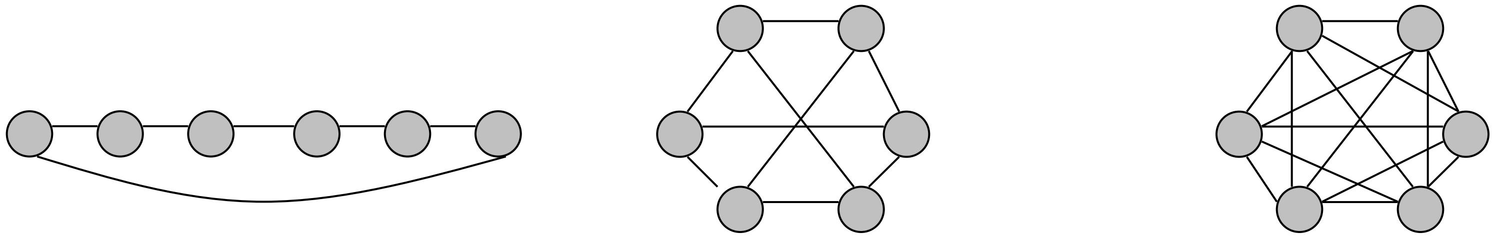
Degree of a vertex can be obtained as:

- Sum of the elements in its row in the incidence matrix
- Length of its vertex incidence list

Motivation for this section

- Given very many nodes in the network
- Issue 1:
  - Where is the requested information/service ?
- Issue 2:
  - Structure of network should be suitable for fast searches or lookup
- Constraints:
  - Only partial view on the network
  - A peer can only contact its “neighbors”
- Need metrics to describe the properties of network topology

Motivation: Which is the difference between the following graphs? Advantages? Disadvantages?



Ideal network characteristics (general)

- Small network diameter (worst-case distance)
- Small average distance / path length
- Limited and small vertex/node degree
- High connectivity (and high fault tolerance)
- Support/allow load balancing of traffic
- Scalability (e.g. O(log n))
- Symmetry

→ but, hard to obtain in reality

3.1 Clustering Metrics

Clustering of one node

- Measured by the Clustering Coefficient CC
  - No. of links among a node's immediate neighbors to each other
- Compared to
  - Max. number of possible links they might have between them

CC has

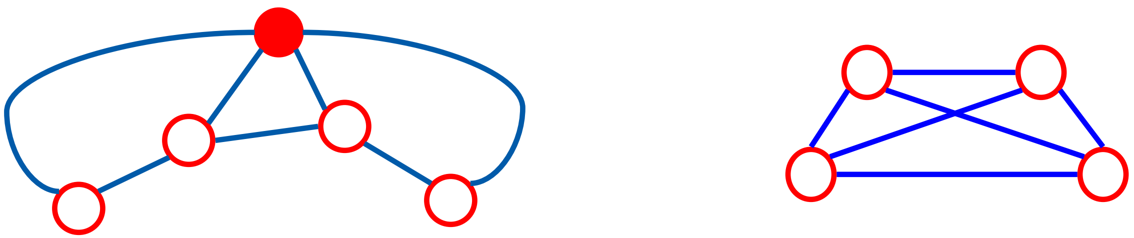
- K - the amount of node‘s neighbors (directly connected)
- N - the actual number of edges among K neighbors of the node

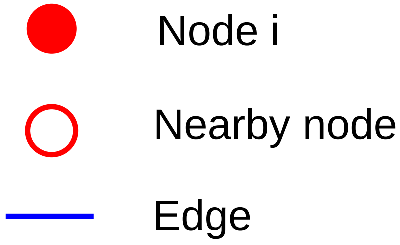
$$CC = \frac{N}{\frac{K(K-1)}{2}}$$

Clustering factor $\mathbf{CC_{network}}$ of a network

- Average clustering factor of all nodes of a network

Clustering Example





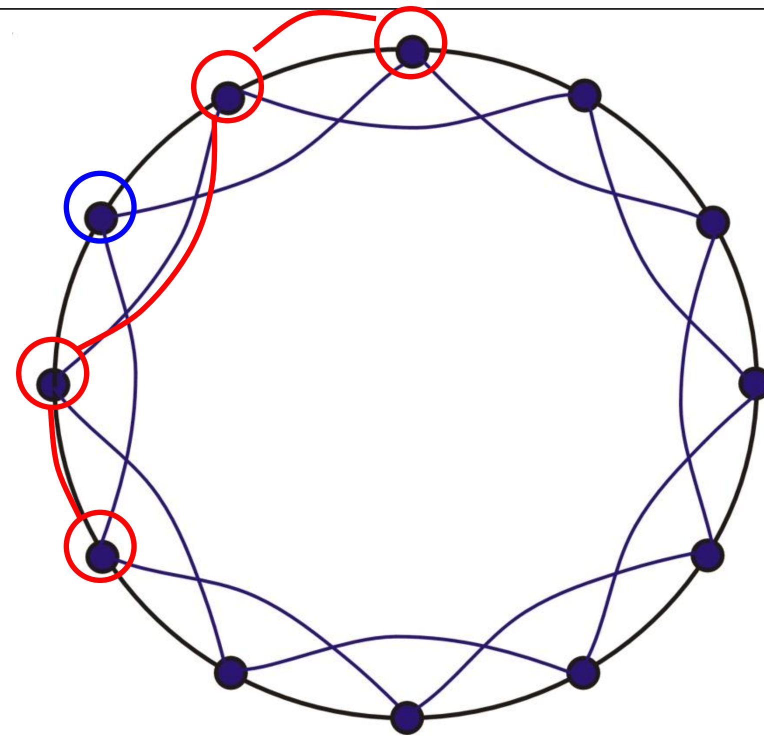
N = 3 edges between the K = 4 adjacent nodes to node I

→ Max. 6 edges: $\frac{K(K-1)}{2} = \frac{4(4-1)}{2} = 6$

$$CC_i = \frac{N}{\frac{K(K-1)}{2}} = \frac{3}{6} = 0.5$$

Clustering Example

Example: regular graph



CC of one node has

- K = 4
- N = 3

$$CC = \frac{3}{\frac{4(4-1)}{2}} = \frac{3}{6} = 0.5$$

CC of whole network

- Average CC of all nodes
- All nodes have same CC
- I.e., $CC_{network} = 0.5$

TECHNISCHE UNIVERSITÄT DARMSTADT

Path length L

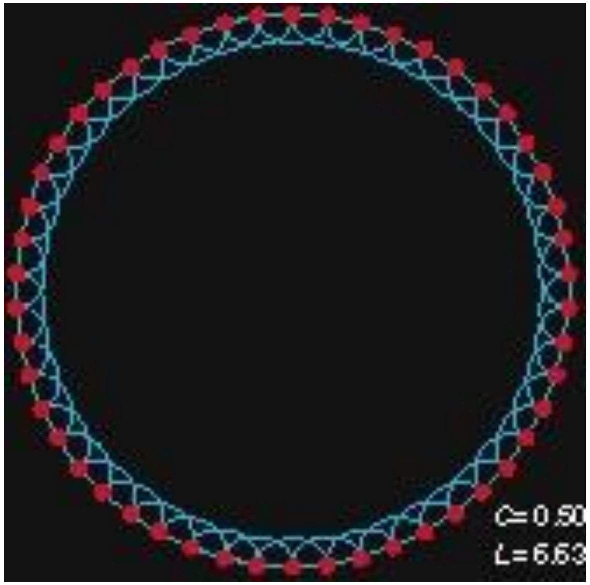
1. choose a node
2. calculate the median distance to the rest of the graph
3. …choose another not yet selected node ..

…

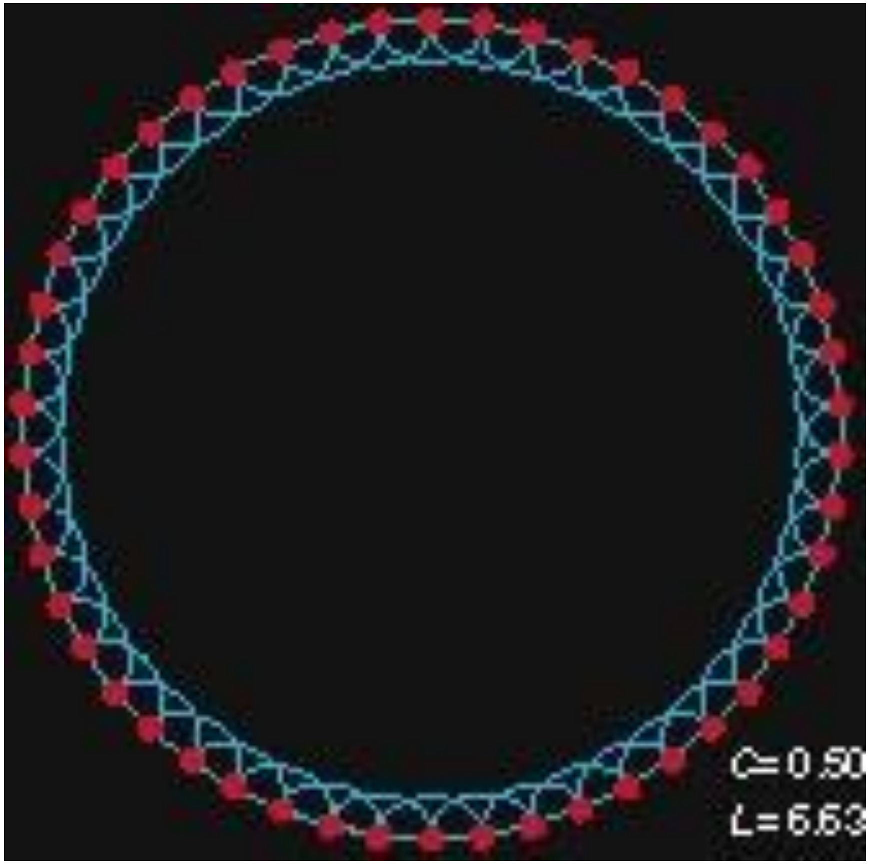
Z. average over all nodes

Issue

- How to find the hops with only a local knowledge of the topology?



Example of Calculation of Average Path Length



Example: regular graph with

- 50 nodes and CC = 0.5

Closest node

- L = 1

Most distant node

- At opposite side
- Distance to opposite side,180 degrees
  - ca. half of 25 nodes
  - = 12.5

Average node

- located at 90 degrees
- Ca. half of 12.5 = 6.25
- I.e.,
  - approx. L = 6.25
  - (exactly L = 6.63)

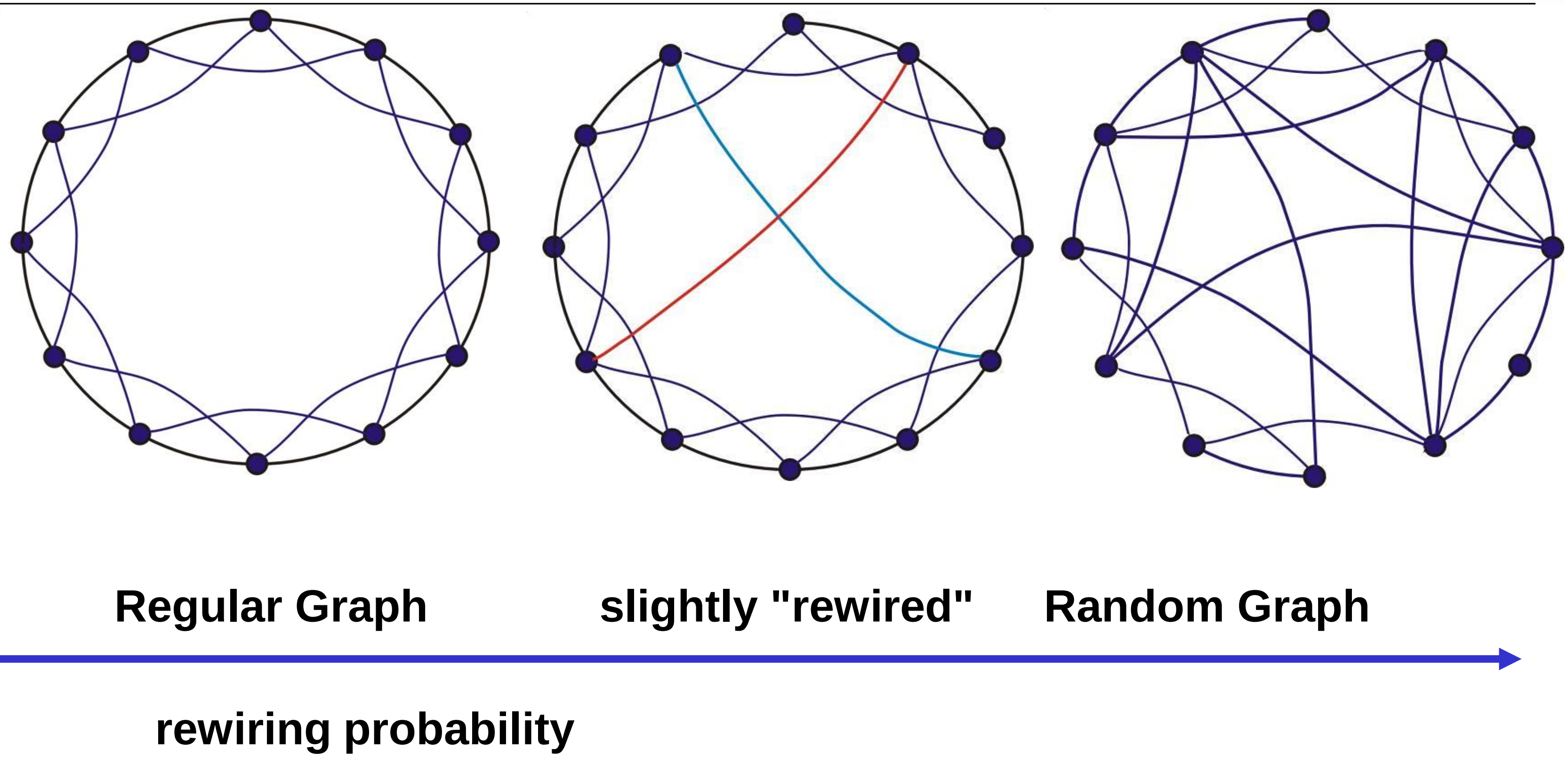
Graphs seem (at a first glance) to be established randomly

- But, characteristics are not “random”

Hence,

- Which are the properties?
- Search for “simple” construction principle

Watts / Strogatz Process

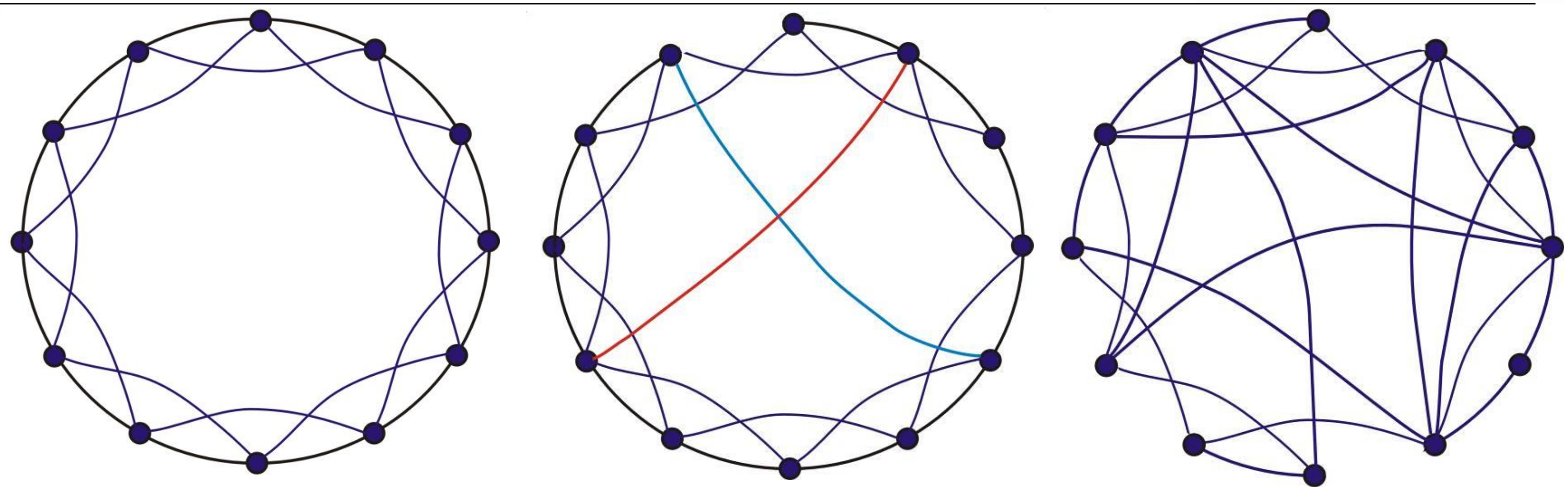


Process (by Watts and Strogatz)

- randomly select edge (by edge) and
- randomly "rewire" it (them)

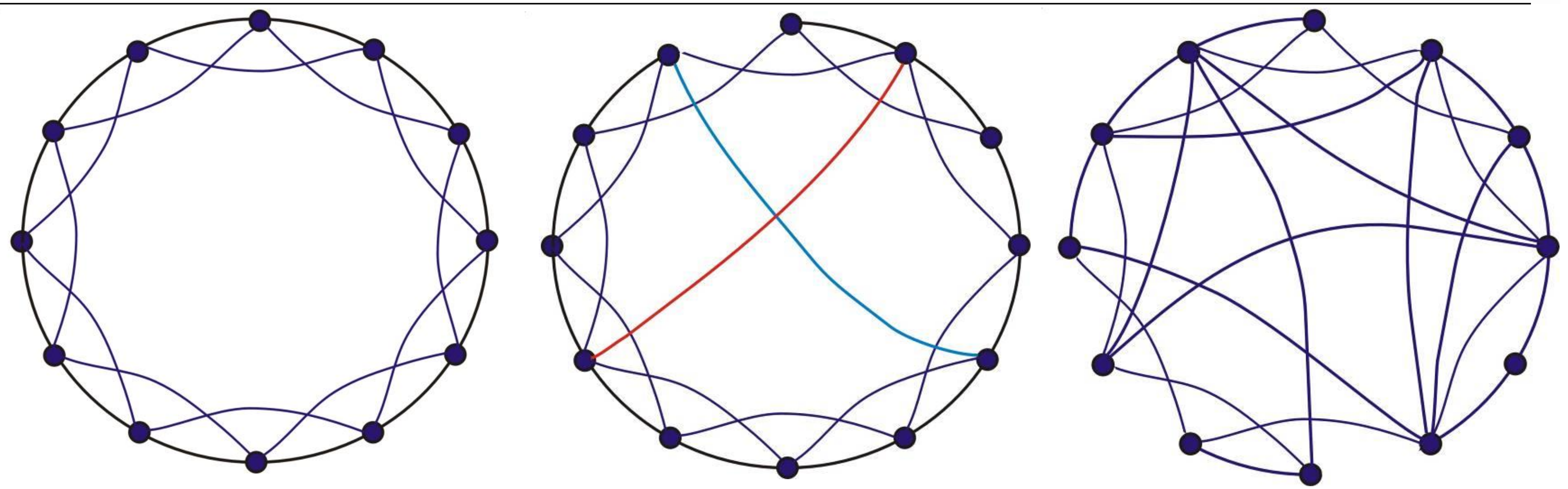
Which is the effect of this process?

Small World Phenomenon



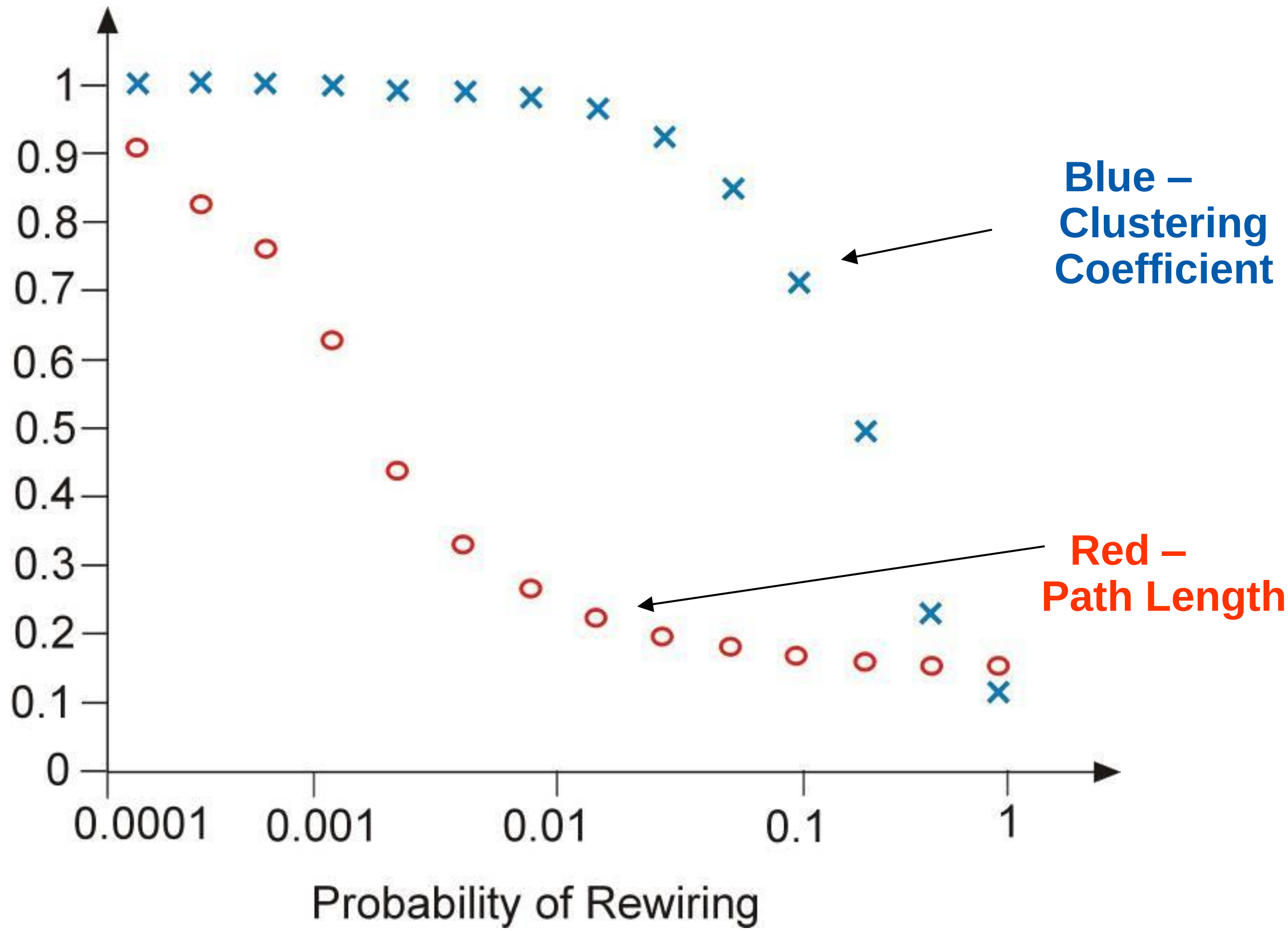
	Regular Graph	Slightly rewired graph	Random graph
Clustering Coefficient	?	?	?
Path Length	?	?	?

Small World Phenomenon



	Regular Graph	Slightly rewired graph	Random graph
Clustering Coefficient	**high**	high	**low**
Path Length	**high**	low	**low**

Results

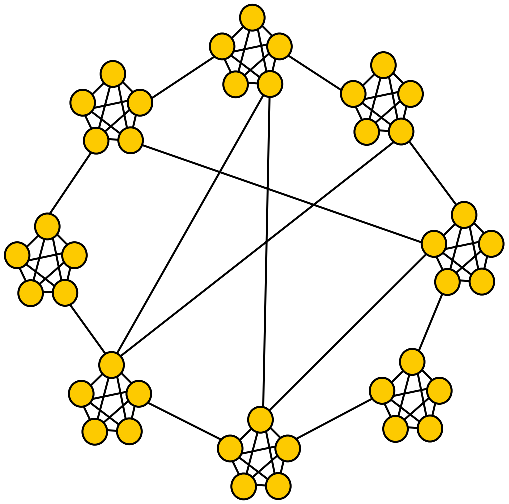


Effect

- "Rewiring" very few edges to be randomly reconnected
- Clustering remains high
- But, path lengths (better look-up times) are dramatically reduced

Small World Graphs/Networks

Slightly rewired graphs = small world graphs



Noticed properties:

1. Clustered sparseness (clustering)

- How “CLIQUISH" a graph is
  - Small World Networks comprise few edges
  - Network set-up by many interconnected clusters

2. Small Diameter (path length)

- Minimal distance between the most apart peers is small
- Average path length is RATHER SMALL
- Path length GROWS LOGARITHMICALLY with size of network

Scale Free Networks

Power laws are scale free

- Because
  - if k is rescaled (multiplied by a constant),
  - then P(k) is still proportional to … $P(k) \propto k^{-\gamma}$
- P(k) probability that a node in the network connects with k other nodes
- γ (gamma) coefficient (may vary ca. 2 to 3)

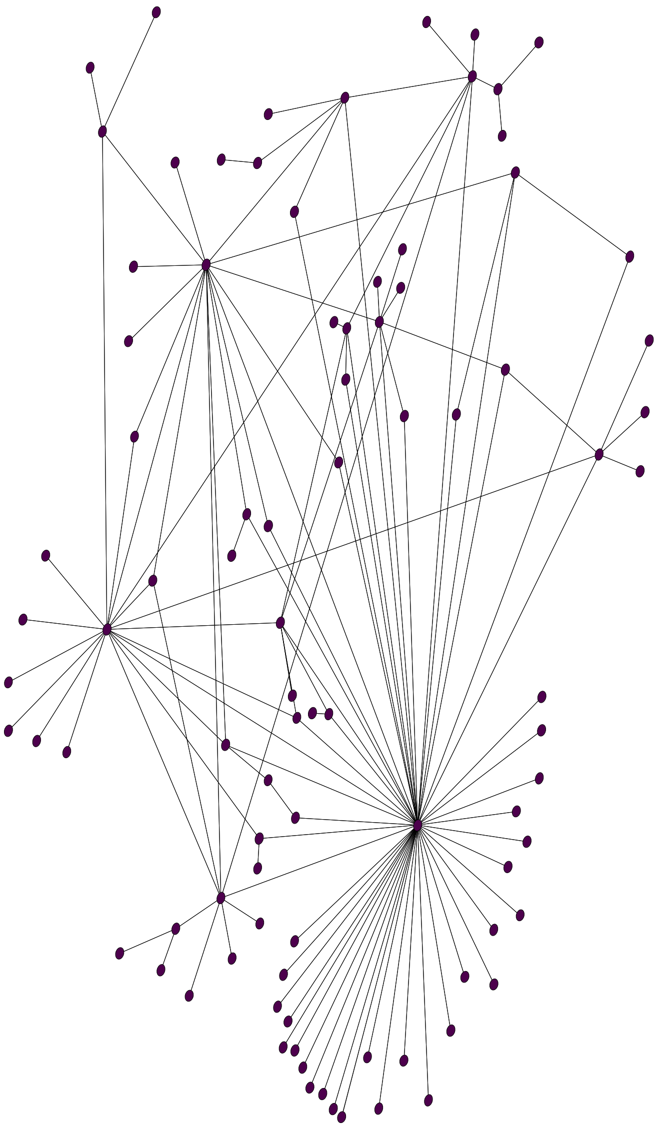
New nodes enter the network

- By attaching to already popular nodes
- (Rich nodes get richer)

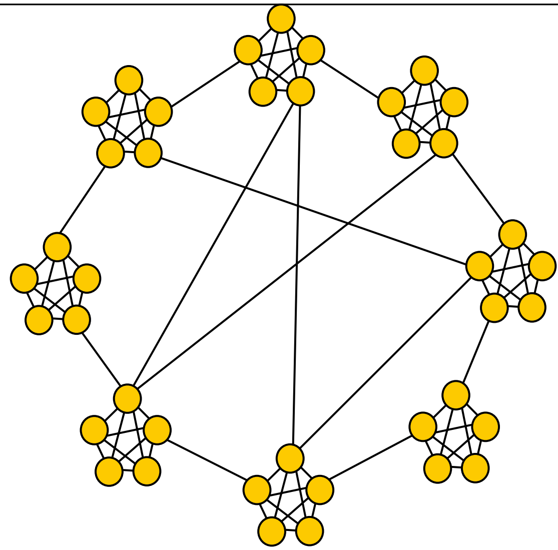
Short diameter

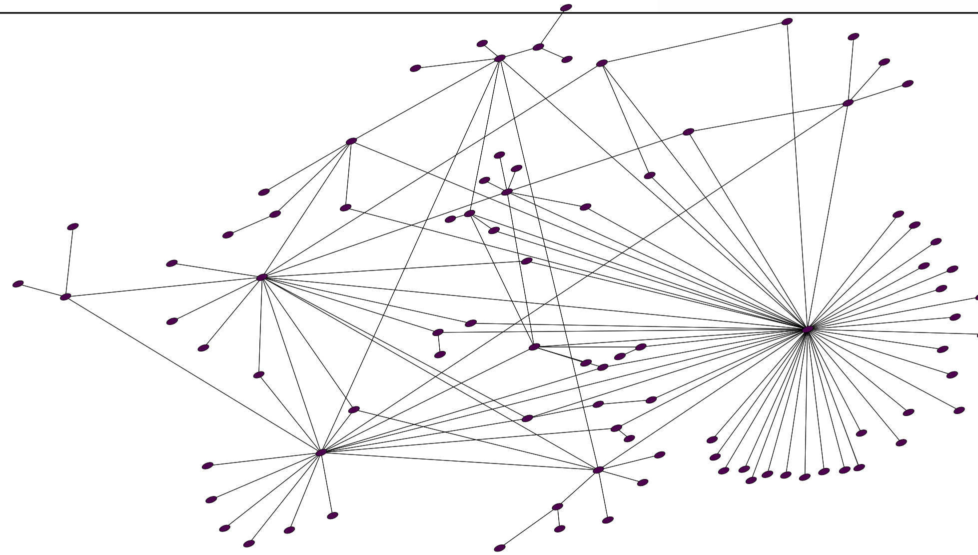
Power-law distribution

- Uneven link / load distribution
- Supports heterogeneity
- Fragile to attacks at high-degree nodes



Small World and Scale Free Networks





Examples

- www-pages 17/19 hops
- Phoning in the US
- Personal relationship US 6 degrees of separation
- Movie relationship between performers 3 hops
- Coauthorship in papers
  - E.g.: http://database.cs.ualberta.ca/coauthorship/
- Example Hubs
  - E.g. airports (vs. Streets)

Characteristics

- Clustering
  - Strong local interaction
  - Cliquish
- Connectivity, degree of nodes
  - Few with high degree
  - Many with low degree
  - NO: Poisson distribution
  - YES: power law distribution $P(k) = k^{-\gamma}$

Statistics resulting from the Watts/Strogatz graphs

- Do not match those of real-world small world graphs like certain network graphs (topologies)
  - e.g. power supply grids, web pages, P2P networks
- Power-law distribution of edges to nodes
  - Watts/Strogatz model does not account for that

Barabási:
2 techniques result in power law distributions

- Dynamic growth
  - constructs small worlds graphs dynamically
  - rather than rewiring a graph in place as with Watts/Strogatz
- Preferential attachment
  - rewiring of nodes preferentially attaching to most connected nodes
  - rather than randomly

Power Law: Distribution of Node Degree

P(k)

- Probability that a randomly selected node has exactly k edges
- I.e., spread of node degrees k_i over the network

E.g. regular lattice

- All nodes have same node degree k_i
- I.e., P(k) is delta function

E.g. random network

- Poisson distribution of node degrees
- I.e., for any degree k >> mean # degree (named <k>)
  - P(k) tends to be 0
- But, does not apply in reality!

But…

Power law distribution of node degrees

- High connectivity is unlikely
- But occurs more often than predicted by random network

Social Networks

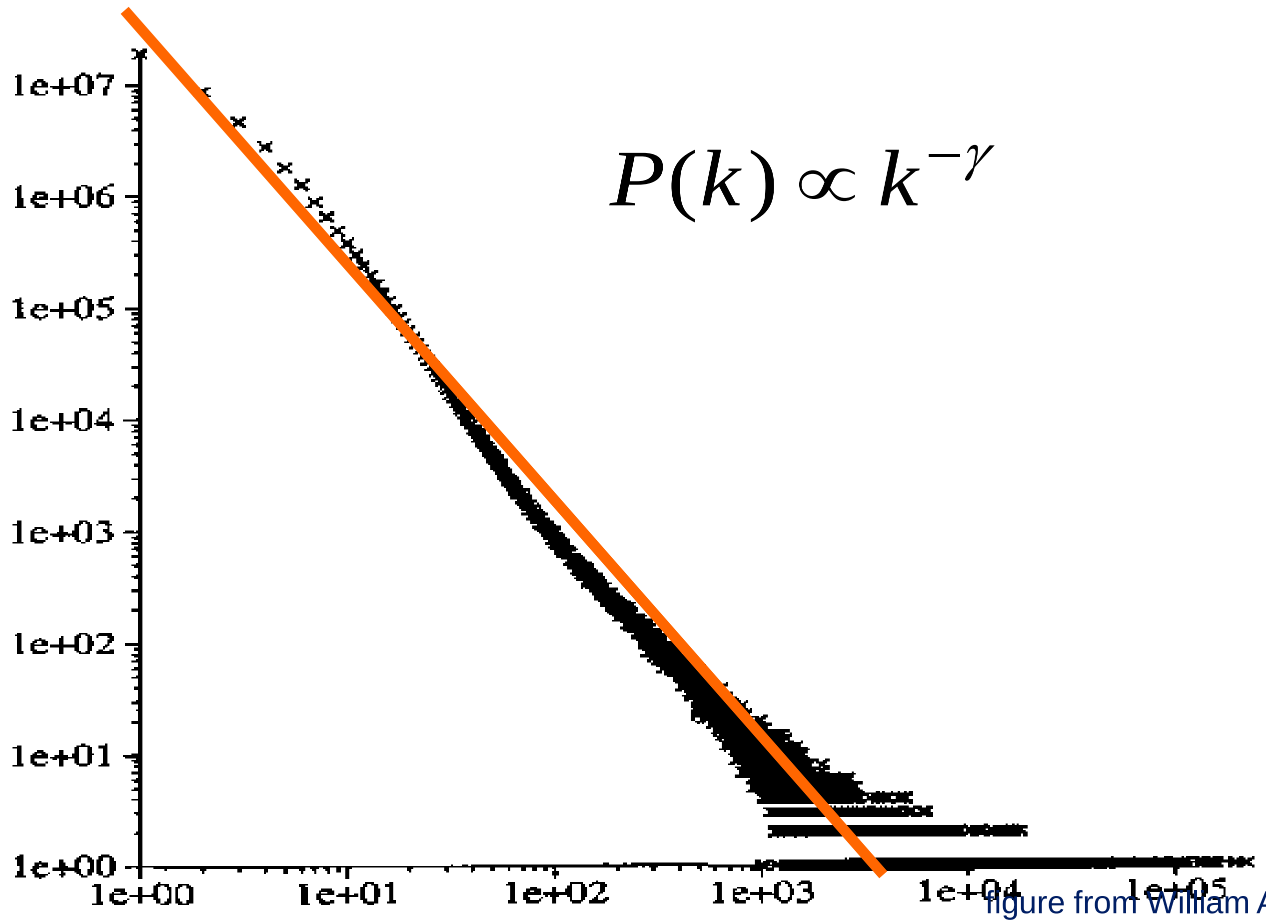


figure from William Ajello

Recall

- Path is a walk from vertex *v* to *u* where we go along each edge and visit each vertex at most once
  - Note: Does not have to visit every edge and vertex of the graph
- Cycle is a path which ends in the vertex where it started

Issues

- Shortest Paths
- Bellman’s Optimality Principle
- Shortest Path Algorithms

Consider graph $G = (V, E)$ where each edge $(v_i, v_j) \in E$ has a length $l_{ij} > 0$

- Length = actual length, cost, weight, … (any suitable metric)

Shortest path problem:

- **For fixed v_1 and v_k, which is the path from v_1 to v_k such that the sum of the lengths of its edges is minimum?**

Longest path is similarly defined

Note: There can be several "shortest" paths

Consider the problem of finding shortest paths from a given node v to all other nodes

- P_j denotes the shortest path from v to j
- L_j denotes the length of the shortest path P_j

4.2 Bellman's Optimality Principle

If P_j is a shortest path from v to j and (i, j) is the last edge of P_j, then P_i (obtained from P_j by dropping edge (i, j)) is a shortest path from v to i

Proof on the next slides

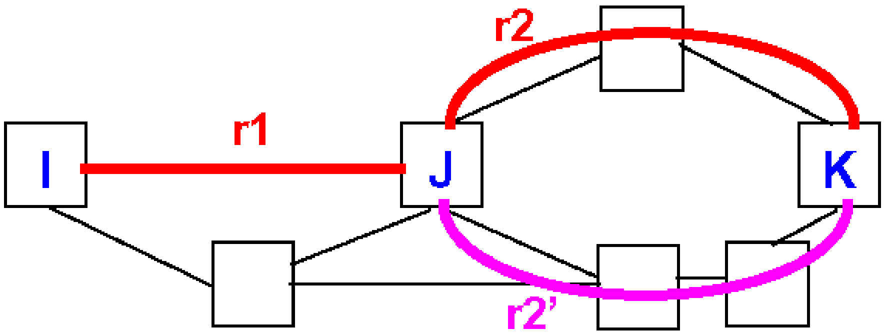
- Idea: For fixed j, try different shortest paths P_i and add (i, j). Lengths of these paths are $L_i + l_{ij}$. Pick i which gives the smallest overall length

Basis for Dijkstra's Shortest Path Algorithm

Optimality Principle

General statement about optimal routes

- If node J is on optimal path from node I to node K
- Then the optimal path from node J to node K uses the same route



Example

- r1: route from I to J
- r2: route from J to K
- If better route r2' from J to K existed
- Then concatenation of r1 and r2' would improve route from I to K

→ Set of optimal routes

- from all sources
- to a given destination

form a tree rooted at the destination: SINK TREE

Proof of Bellman's Principle

TECHNISCHE UNIVERSITÄT DARMSTADT

Simple proof by contradiction

If P_j is a shortest path from v to j and (i, j) is the last edge of P_j, then P_i (obtained from P_j by dropping edge(i, j)) is a shortest path from v to i

Suppose that the conclusion is false

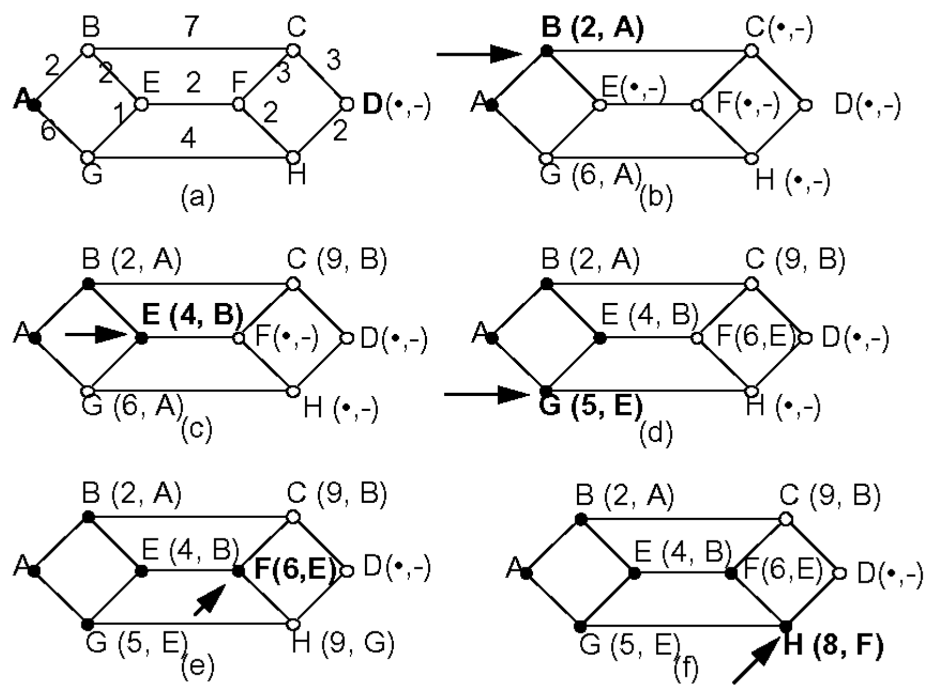
- Then there exists path P_i^* from v to i which is shorter than P_i
- If we now add (i, j) to P_i^*, we get a path from v to j which is shorter than P_j
- This contradicts the assumption that P_j is the shortest path from v to j

Spanning Tree and Optimized Route

- Information about the entire network has to be available

Example

- Link is labeled with distance / weight
- Node is labeled with distance from source node along best known path (in parentheses)



Non-Adaptive Shortest Path Routing

TECHNISCHE UNIVERSITÄT DARMSTADT

Procedure
E.g., according to Dijkstra

Find the shortest path from A to D

- Labels may be permanent or tentative
- Initially, no paths are known
  → all nodes are labeled with infinity (TENTATIVE)
- Discovery that label represents shortest possible path from source to any node:
  → label is made PERMANENT

E.g., ….

1. Node A labeled as permanent

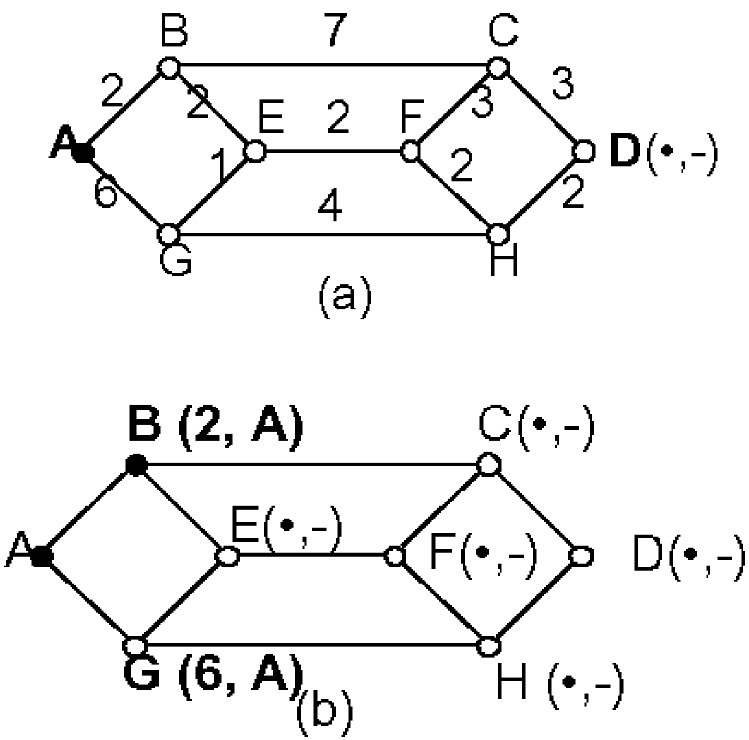
2. relabel all directly adjacent nodes

3. examine

4. this node is the new working node

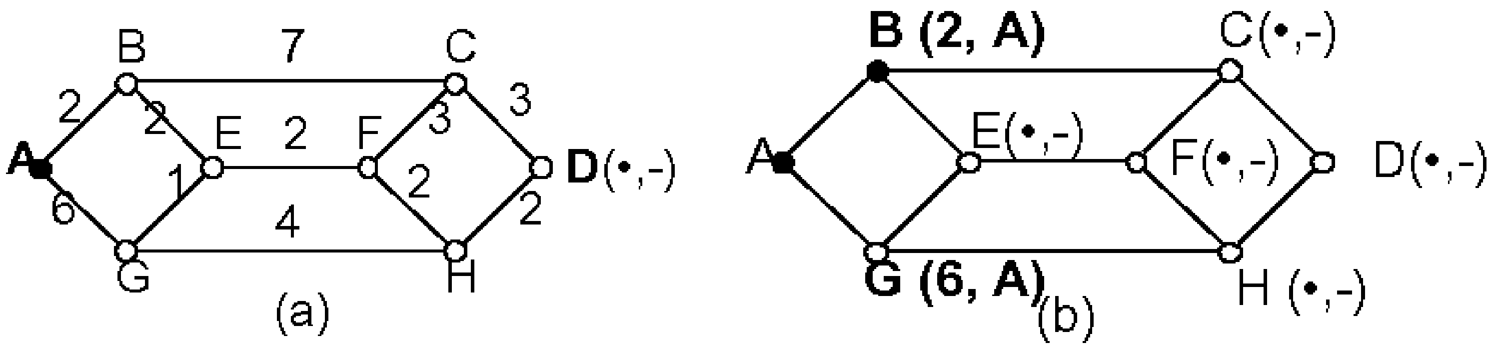
Non-Adaptive Shortest Path Routing

E.g.,



1. **Node A labeled as permanent**
   - Filled-in circle
2. **Relabel all directly adjacent nodes**
   - With the distance to A
     - path length,
     - nodes adjacent to source
   - E.g., B(2,A) and G(6,A)
3. **Examine**
   - All tentatively labeled nodes;
   - Make the node with the smallest label permanent
   - E.g., B(2,A)
4. **This node is the new working node**
   - For the iterative procedure
   - I.e., continue with step 2.

Non-Adaptive Shortest Path Routing (Worksheet 1)



Example (distance indicated by the number on the edge)

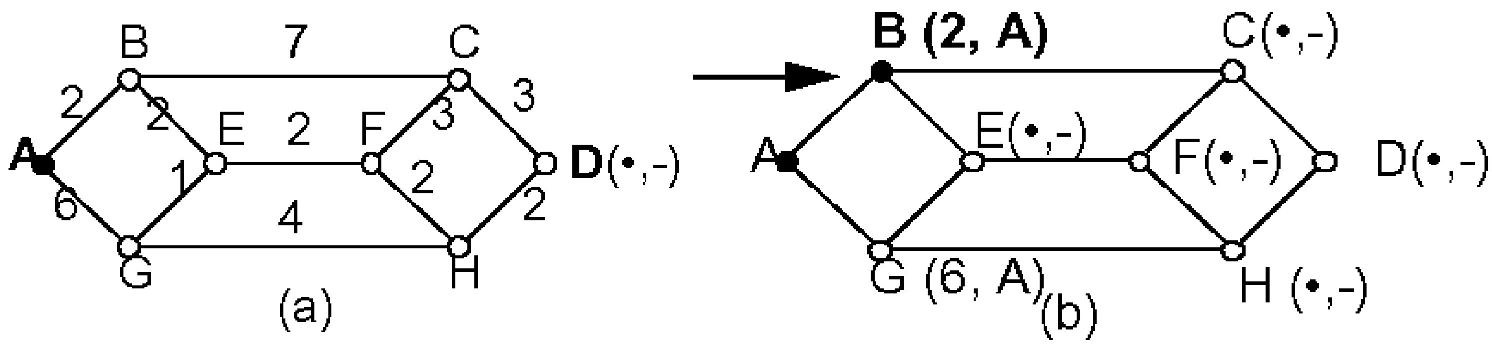
Procedure: e. g. according to Dijkstra

Find: the shortest path from A to D:

1. A flagged as permanent (filled-in circle)
2. Relabel all directly adjacent nodes with the distance to A
   - (path length, IS adjacent to the source):
   - e. g. B(2,A) and G(6,A)

Non-Adaptive Shortest Path Routing (Worksheet 2)

TECHNISCHE UNIVERSITÄT DARMSTADT



Example (distance indicated by the number on the edge)

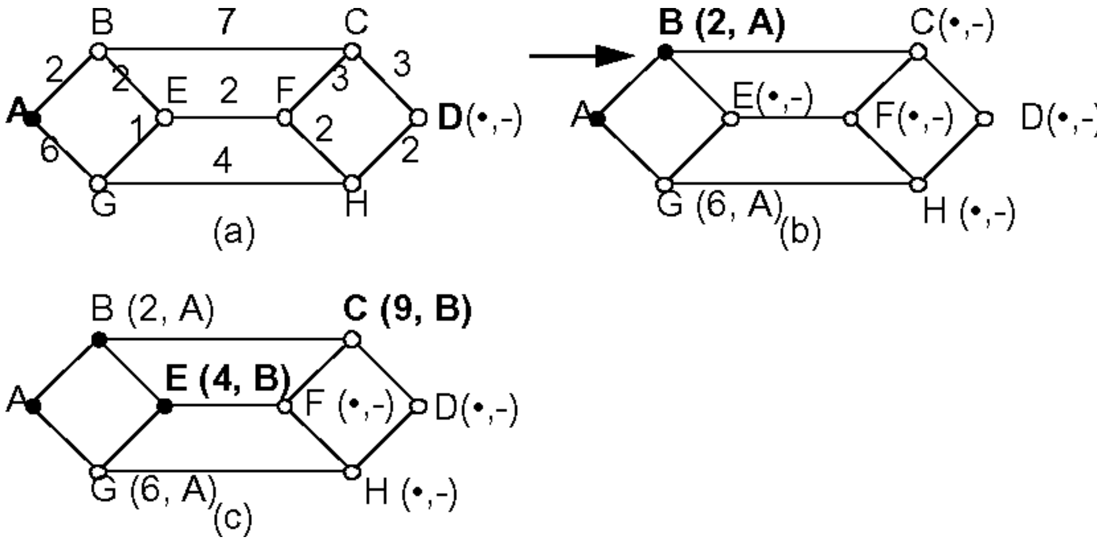
Procedure: e. g. according to Dijkstra

Find: the shortest path from A to D:

...

3. Compare all recent, not firmly flagged IS;
   - Flag the one with the lowest number AS FIXED:
   - B(2,A)
4. This IS is the origin of the iterative procedure
   - (i.e., continue with item 1.)

Non-Adaptive Shortest Path Routing (Worksheet 3)



Example

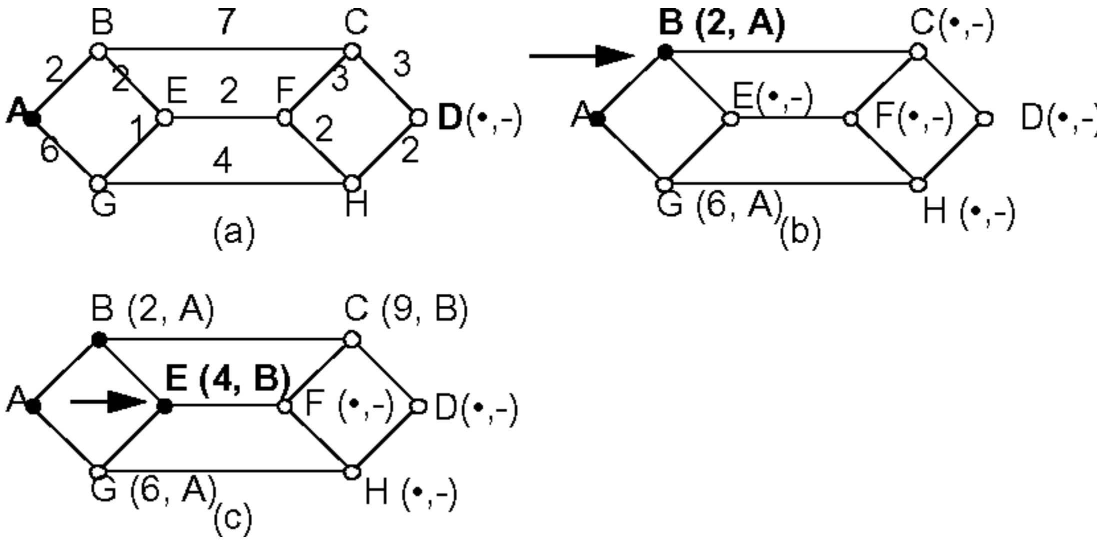
- Link is labeled with distance
- Node is labeled with distance from source along best known path

Procedure: e.g., according to Dijkstra find the shortest path from A to D:

1. **Node B has been labeled as permanent**
   - (filled-in circle)
2. **relabel all directly adjacent nodes with the distance to B**
   - (path length, nodes adjacent to source):
   - A (does not apply, because it is the origin),
   - i.e. E (4,B), C (9,B)

Non-Adaptive Shortest Path Routing (Worksheet 4)

TECHNISCHE UNIVERSITÄT DARMSTADT



Example

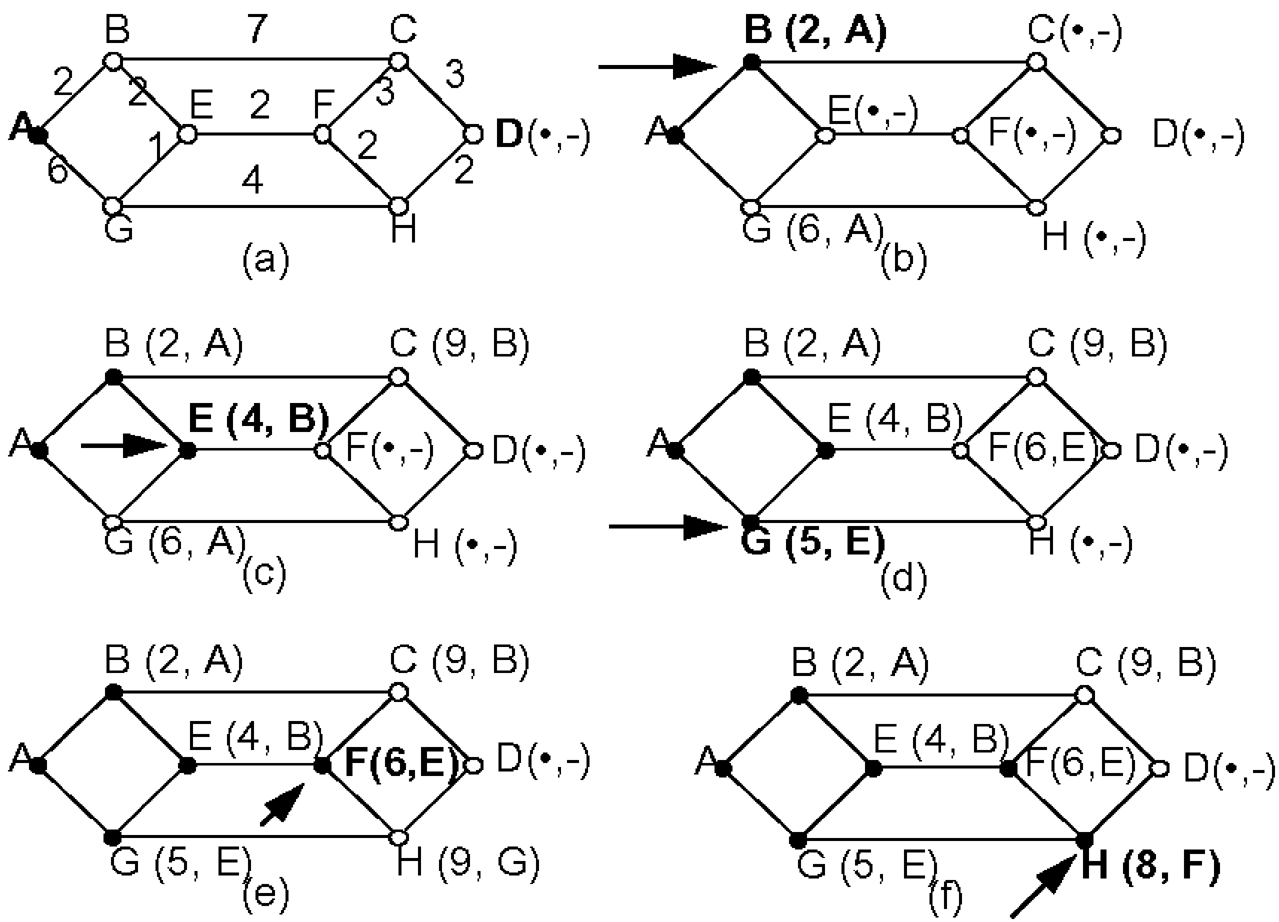
- Link is labeled with distance
- Node is labeled with distance from source along best known path

Procedure: e.g., according to Dijkstra find the shortest path from A to D:

- ....
- ....
- Examine all tentatively labeled nodes;
  - make the node with the smallest label permanent: e.g. E(4,B)
- This node will be the new working node for the iterative procedure ...

Non-Adaptive Shortest Path Routing (Worksheet 5)

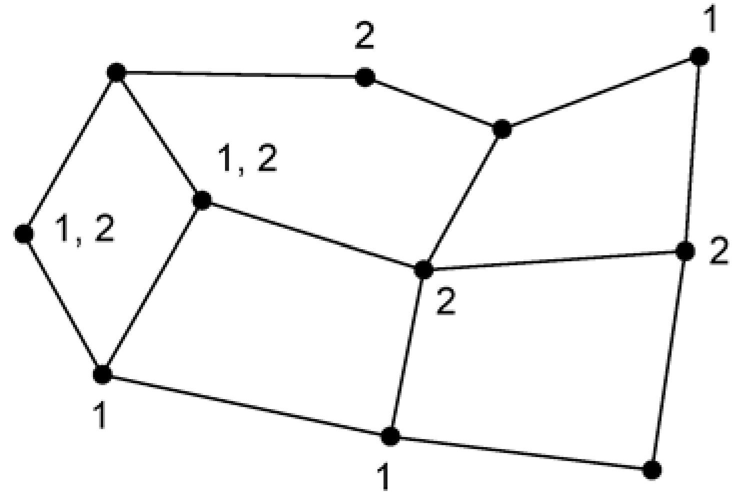
And continue with source E …



Tree is an acyclic, connected graph

- Consider graph *G = (V, E)*

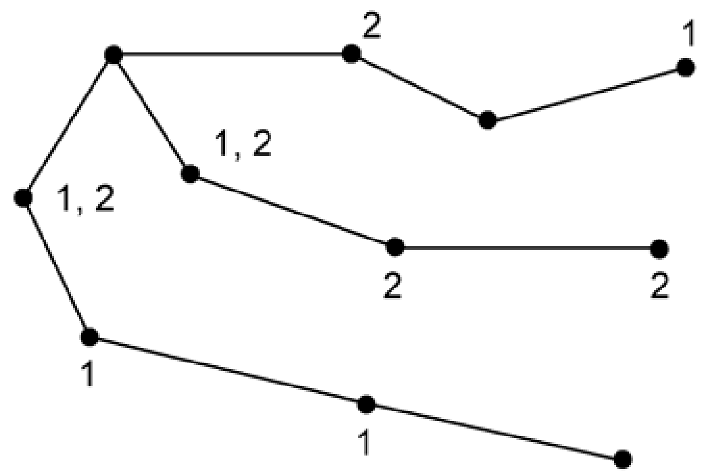
E.g., acyclic connected graph



Spanning tree *T* of graph *G* is a tree containing ALL *v* vertices of *G*

- Such a tree has *v - 1* edges
- Why?

E.g., spanning tree for leftmost node



Shortest Spanning Tree(s)

Given a graph G whose edges *(i, j)* have lengths $l_{ij} > 0$,

the shortest spanning tree *T is a spanning tree for which $\sum l_{ij}$ is the minimum compared to $\sum l_{ij}$ for any other spanning tree *T***

Dijkstra's algorithm gives us a spanning tree

- Not necessarily the shortest spanning tree
- If Dijkstra's algorithm is run for all vertices, then we can find the shortest spanning tree,
- but this is not efficient

Kruskal's algorithm finds the shortest spanning tree

Given a graph $G = (V, E)$, where all edges (i, j) have length $l_{ij} > 0$, the shortest spanning tree T can be obtained as follows:

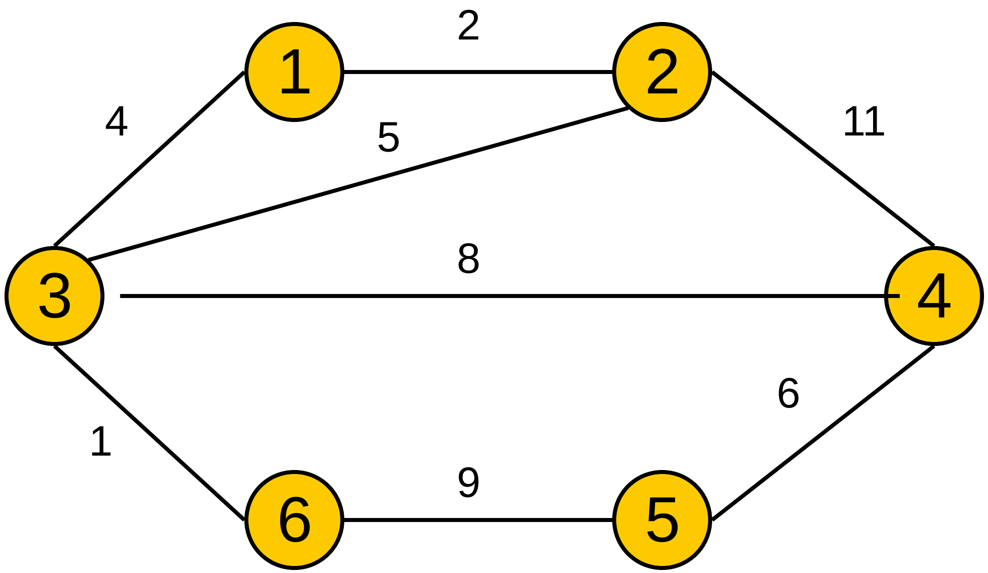
1. Order the edges of G in ascending order of length
2. Choose edges in this order as edges of T
   - Reject an edge if it forms a cycle with the edges already chosen
3. Repeat step 2 until $v - 1$ edges have been chosen

Note: At the intermediate steps, the selected edges may form a disconnected graph

- Eventually we get a tree

Kruskal's Algorithm: Example

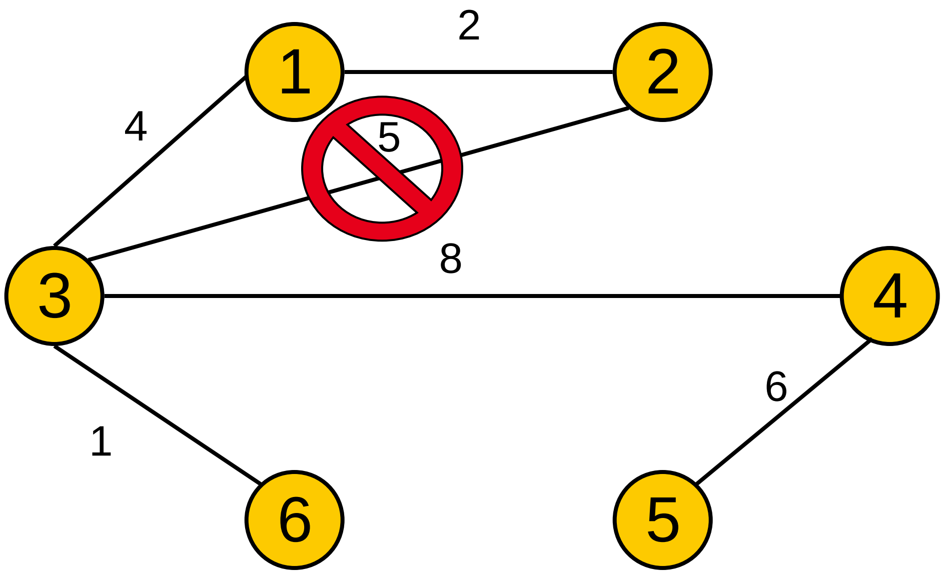
Consider the following graph:



Kruskal's Algorithm: Example

Order edges by length

Edge	Length	Choice
(3, 6)	1	1
(1, 2)	2	2
(1, 3)	4	3
(2, 3)	5	Reject
(4, 5)	6	4
(3, 4)	8	5
(5, 6)	9	
(2, 4)	11	



Stop after 5 edges
Length of spanning tree: 21

TECHNISCHE UNIVERSITÄT DARMSTADT

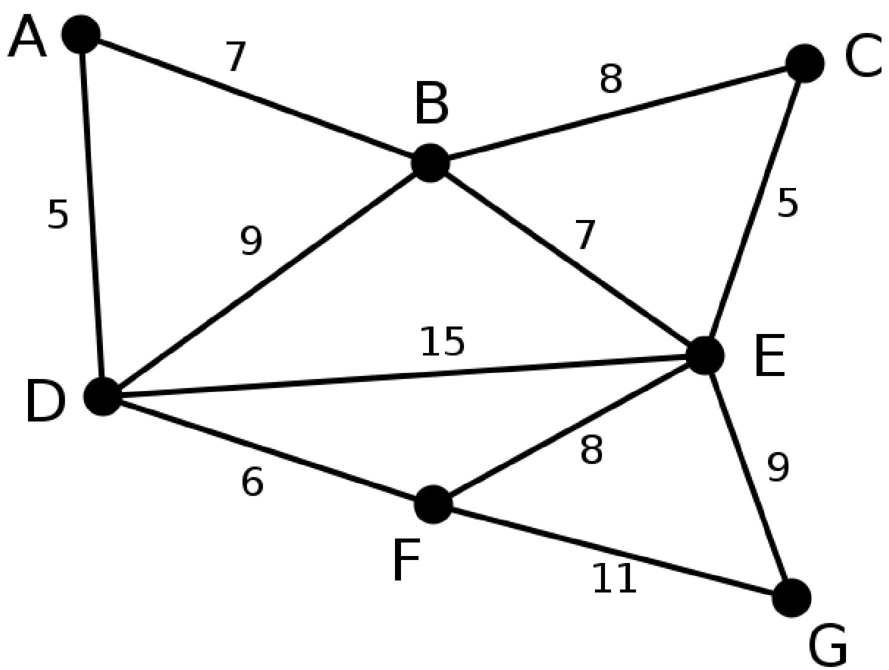
Prim's algorithm also gives shortest spanning tree

Difference to Kruskal is that
Prim gives a tree at every stage of the algorithm

Graph *G* with $V = \{1, \ldots, v\}$, edges of length $l_{ij} > 0$

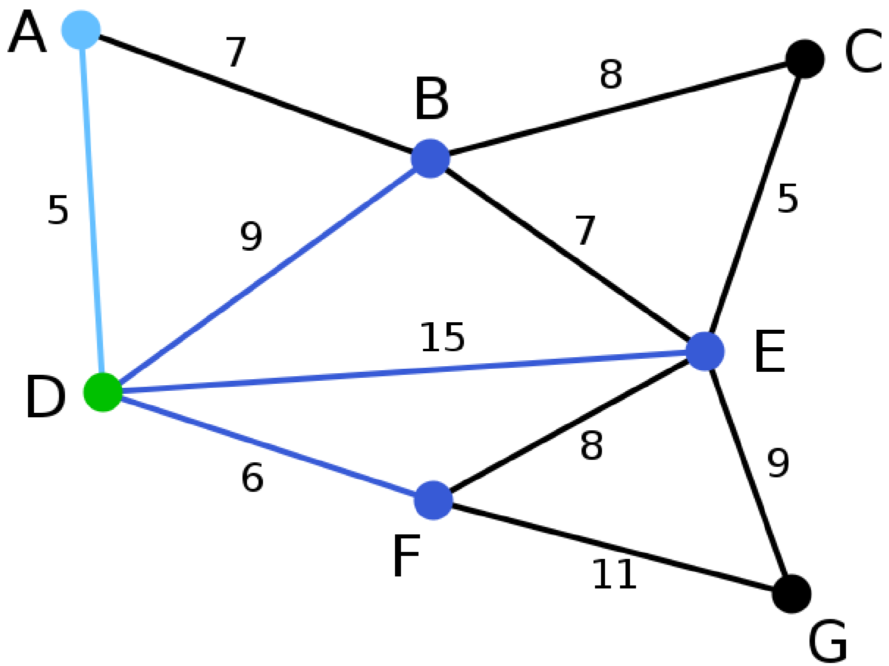
1. Initialize:
   - $i(k) = 1$ $(k = 1, \ldots, v)$, $U = \{1\}$, $S = \emptyset$
   - Label vertex k $(= 2, \ldots, v)$ with $\lambda_k = l_{ik}$ (or ∞, if no edge $(1, k)$)
2. Let λ_j be smallest λ_k for $k \notin U$
   - Add vertex j to U and edge $(i(j), j)$ to S
   - If $U = V$, then stop
3. For every $k \notin U$:
   - If $l_{jk} < \lambda_k$, then set $\lambda_k = l_{jk}$ and $i(k) = j$
   - Go to step 2

Prim’s Algorithm – Example Worksheet 1



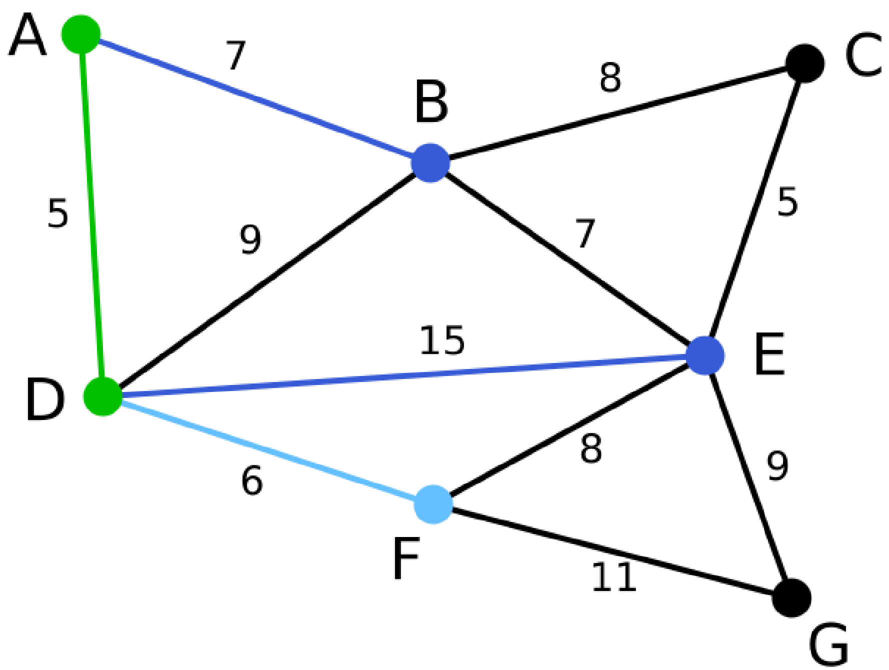
Source: Wikipedia

Prim's Algorithm – Example Worksheet 2



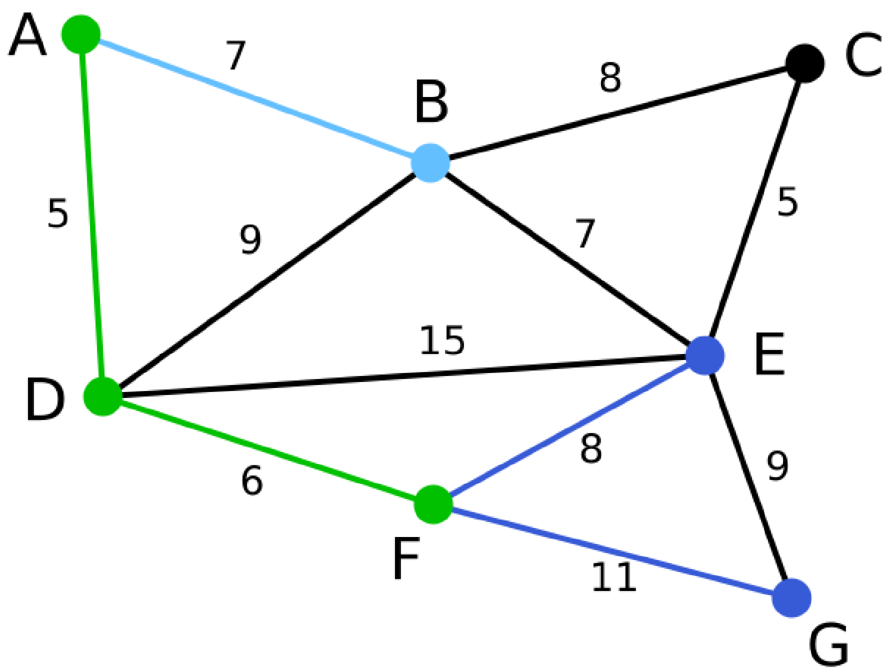
Source: Wikipedia

Prim's Algorithm – Example Worksheet 3



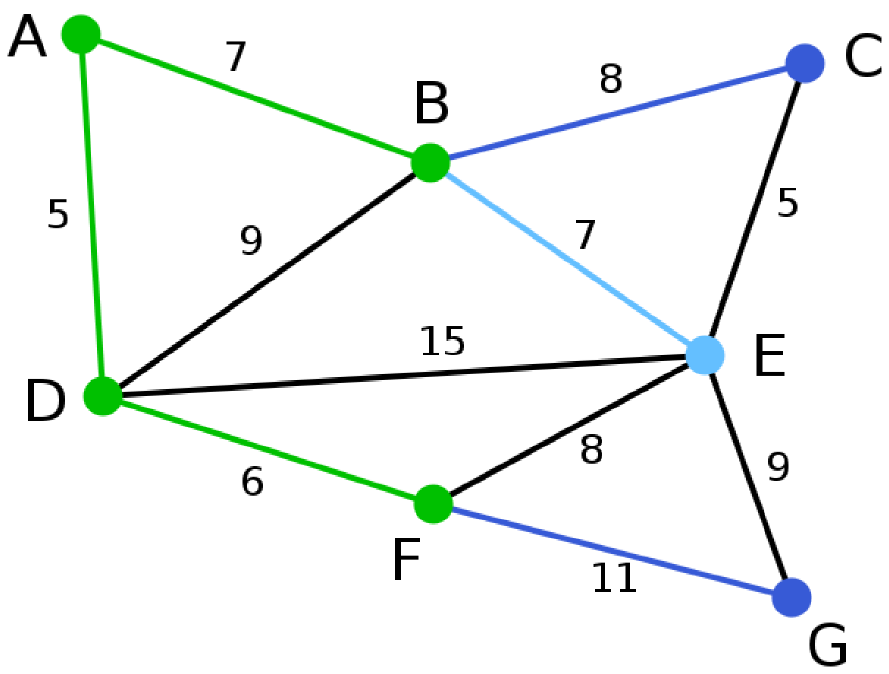
Source: Wikipedia

Prim’s Algorithm – Example Worksheet 4



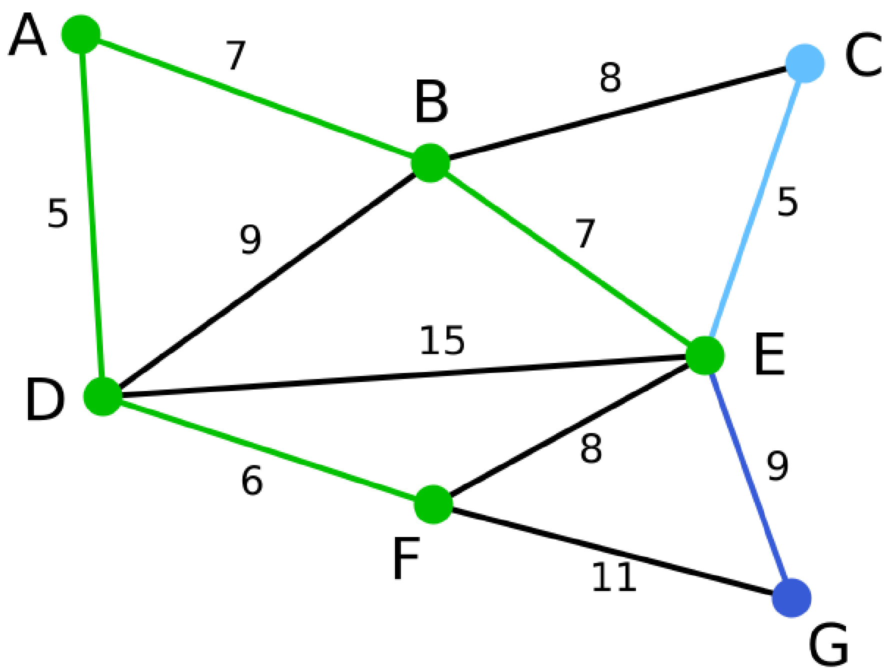
Source: Wikipedia

Prim's Algorithm – Example Worksheet 5



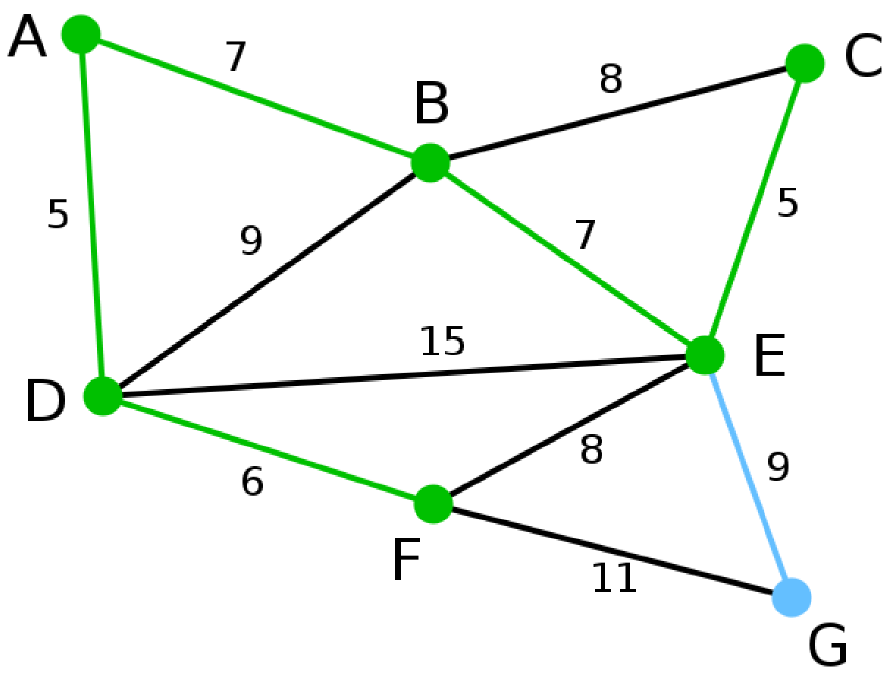
Source: Wikipedia

Prim's Algorithm – Example Worksheet 6



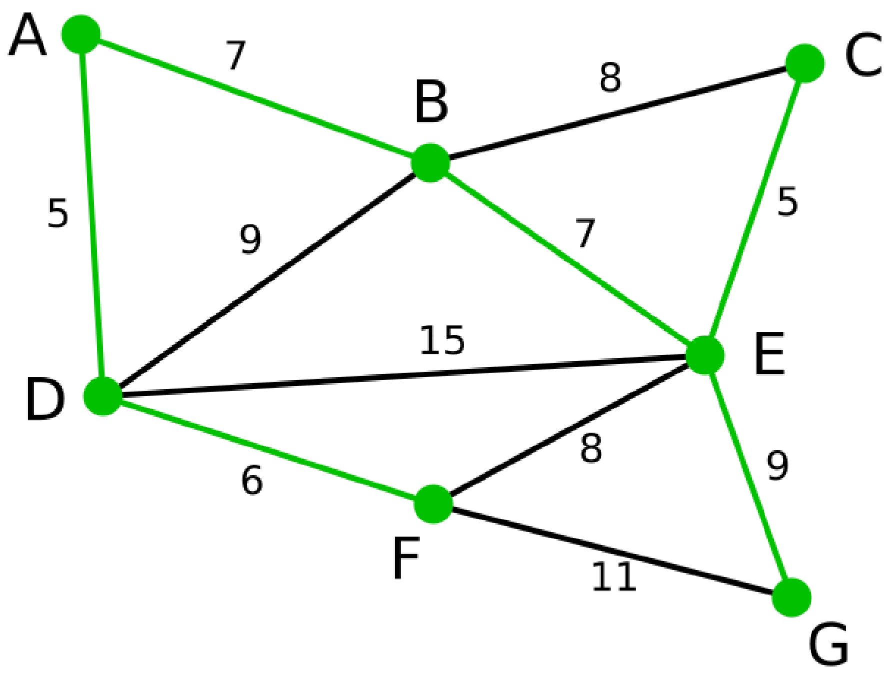
Source: Wikipedia

Prim's Algorithm – Example Worksheet 7



Source: Wikipedia

Prim's Algorithm – Example Worksheet 8



Source: Wikipedia

Kruskal's Algorithm and Prim's Algorithm

Kruskal's Algorithm

Prim's Algorithm

- Keeps always a connected graph
- No need for having always a global view of the whole graph

Source: Wikipedia

Consider a directed graph $G = (V, E)$, where each edge (i, j) has capacity $c_{ij} > 0$

- One vertex s (**source**) produces a flow
- One vertex t (target or **sink**) is where flow disappears

Flow may be network traffic, electricity in wires, water in pipes, cars on the road, …

- Note: Possible to have several sources or sinks

Denote f_{ij} the flow along (directed) edge (i, j)

We have two conditions:

- Edge condition: $0 \le f_{ij} \le c_{ij}$
- Vertex condition:
  - Inflow = Outflow

$$\sum_k f_{ki} - \sum_j f_{ij} = \begin{cases} 0 \\ f \\ -f \end{cases}$$

Flow Augmenting Paths

A path in a directed graph means
a sequence of undirected edges which form a path

- If we travel an edge in its direction: **Forward edge**
- If we travel an edge in opposite direction: **Backward edge**
- Note: This does not change the direction of the directed edge!
  - See below for example

Goal: to maximize flow *f* from *s* to *t*

- *Idea: Increase flow on forward edges or reduce flow on backward edges*

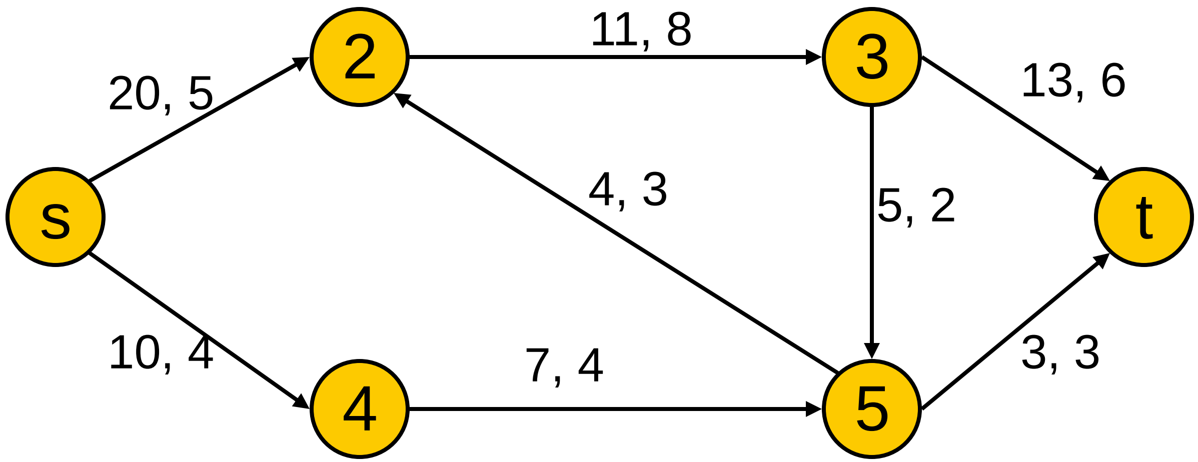
A flow augmenting path is a path for which:

- No forward edge is used to capacity, i.e., $f_{ij} < c_{ij}$ for these
- No backward edge has flow 0, i.e., $f_{ij} > 0$ for these

6.1 Flow Augmenting Paths: Example

First number = capacity, second number = current flow

Flow from s to t is 9



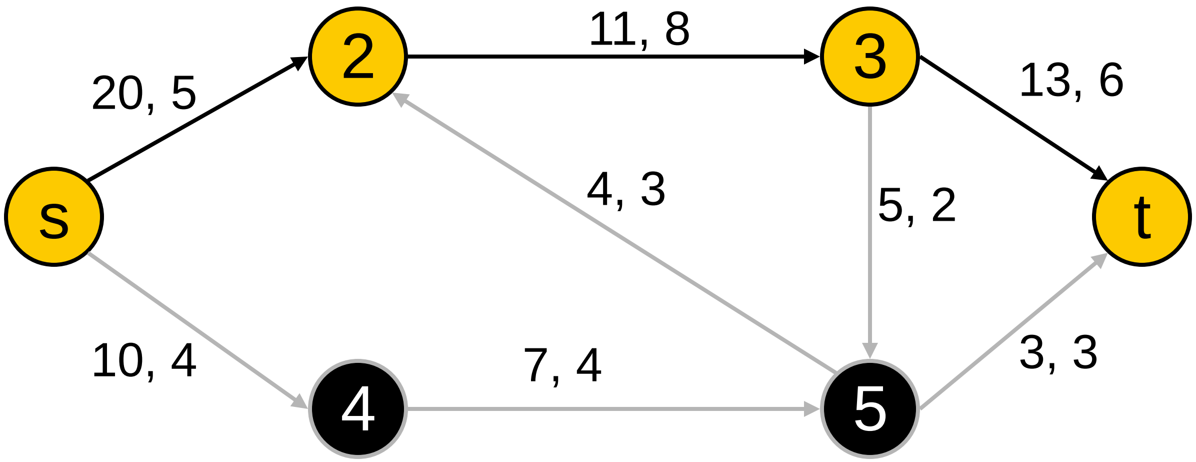
Denote Δ_{ij} as possible increase of flow on edge *(i, j)*

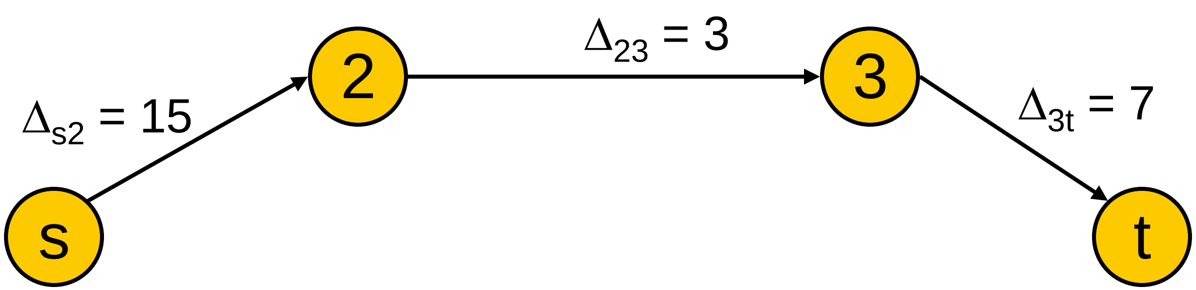
- $\Delta_{ij} = c_{ij} - f_{ij}$ for forward edges
- $\Delta_{ij} = f_{ij}$ for backward edges

Flow Augmenting Paths: Example

TECHNISCHE UNIVERSITÄT DARMSTADT

One possible augmenting path is: *s - 2 - 3 - t*

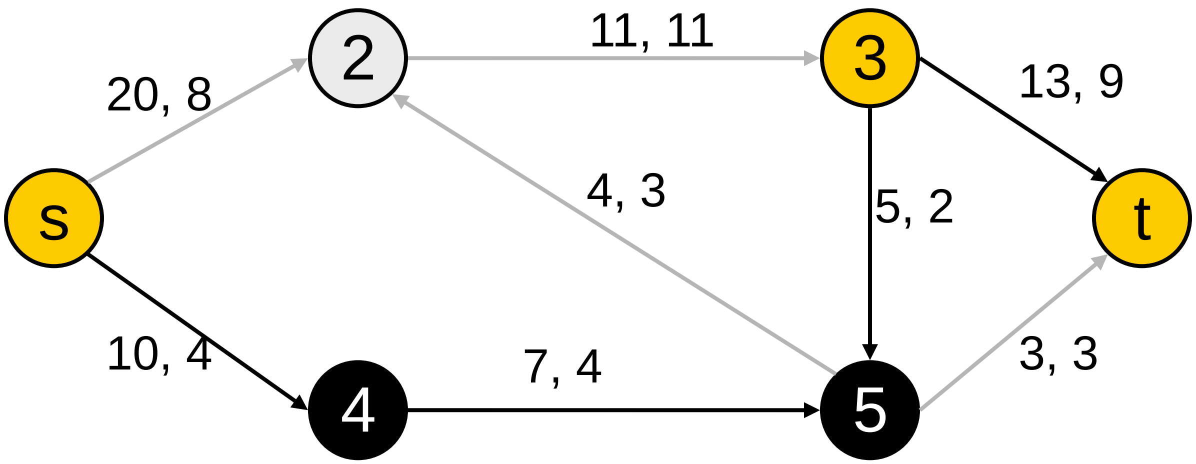




We can increase flow from *s* to *t* by 3 to 12

Flow Augmenting Paths: Example

Other augmenting path is: *s - 4 - 5 - 3 - t*



Edge (3, 5) is a backward edge

- Flow of 2 units that is going from 3 to 5
  could go from 3 to *t* and therefore increase total flow from *s* to *t*

This path allows increase of 2 in total flow which is 14

- 14 is also the maximum flow we can have

6.2 Cuts

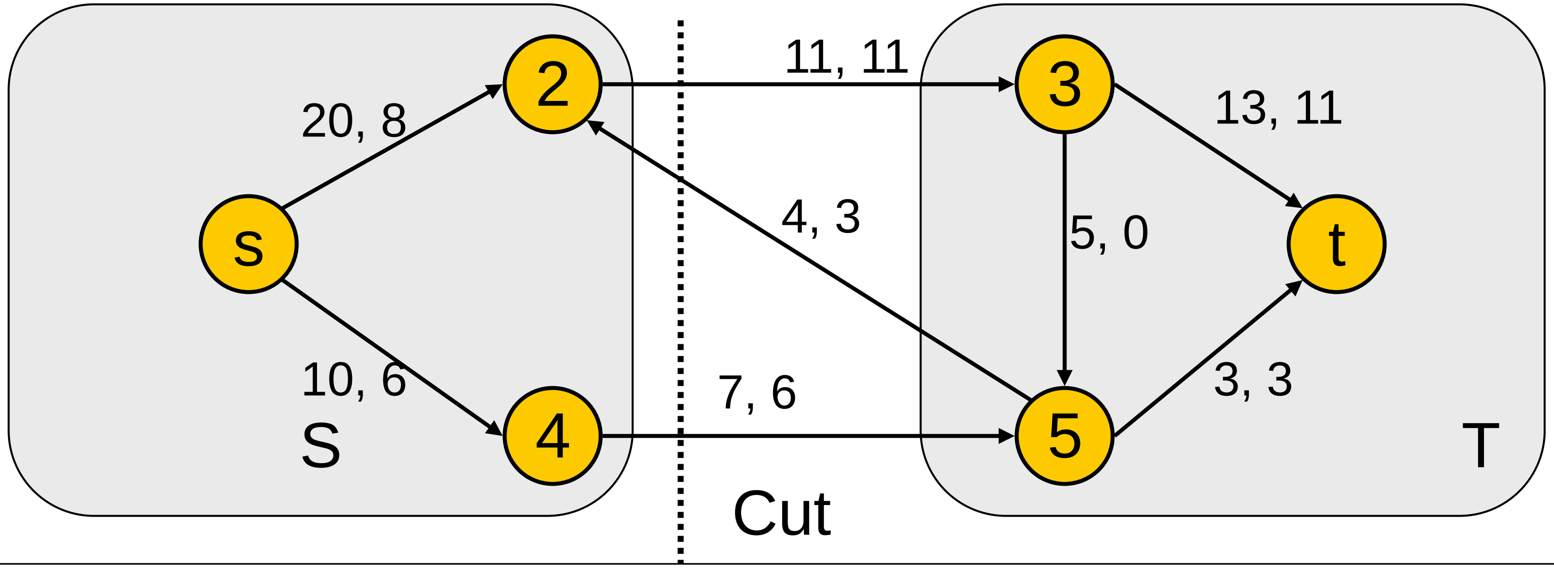
Cut set is a set of edges in the network

Idea: To find flow from *s* to *t*, we cut network somewhere in between & see what flows on those edges

- Any flow from s to t must pass those edges

Cut partitions network into *S* and *T*

- Net flow between *S* and *T* determines flow between s and t
- Sum of "positive" capacities *S* → *T* determines capacity of cut



TECHNISCHE UNIVERSITÄT DARMSTADT

How to find maximum flow in graph?

Maximum flow is equal to the capacity of the minimum cut set (= cut set with smallest capacity)

- Known as **max-flow min-cut theorem**

Can find maximum flow (min cut) with Ford-Fulkerson algorithm

7 Annex: Vocabulary English - German

Acyclic = azyklisch

Adjacency X = Adjazenz X
- X = list or matrix

Connected = zusammenhängend

Degree = der Grad
- Eingangsgrad, Ausgangsgrad

Diameter = der Durchmesser

(Un)Directed = (un)gerichtet

Eccentricity = die Exzentrizität

Edge = die Kante

Flow = der Fluss

Graph = der Graph

Greedy algorithm = gieriger Algoritmus

Incidence X = Inzidenz X
- X = list or matrix

Path = der Pfad / der Weg

Radius = der Radius

Spanning tree = Spannbaum

Sparse = dünnbesetzt

Subgraph = der Teilgraph

Tree = der Baum

Vertex/node = der Knoten