Complex-Network Modelling and Inference

Lecture 2: Graph notation and representation

Matthew Roughan
<matthew.roughan@adelaide.edu.au>
https://roughan.info/notes/Network_Modelling/

School of Mathematical Sciences,
University of Adelaide

March 7, 2024

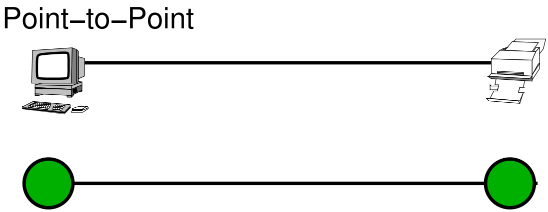
Section 1

Graph Notation

Example graph topologies

- point-to-point
- linear or bus
- ring
- hub and spoke or star
- double star
- fully connected (mesh) or complete topology or clique
- mesh
- (spanning) tree
- hybrid

Point-to-point



description: back-to-back connection of two nodes

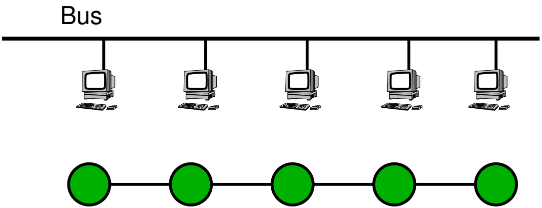
examples:

- (old fashioned) printer connection
- serial link
- PPP (Point-to-Point Protocol)

comments:

- used as a component of a larger network

Bus



description: a single line (the bus) to which all nodes are connected, and the nodes connect only to this bus.

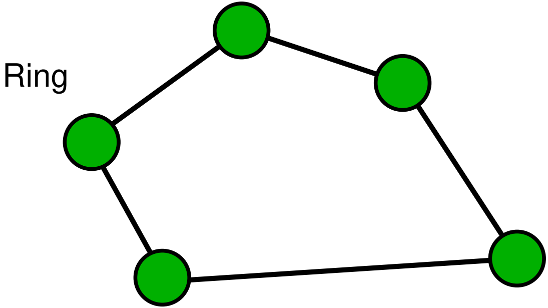
examples:

- physical structure of 10Base2 Ethernet
- logical structure of 10BaseT Ethernet with a hub

comments:

- design often matches a building (corridors)
- no redundancy (failures effect whole network)

Ring



description: Every node has exactly two branches connected to it, so that they form a (logical) ring.

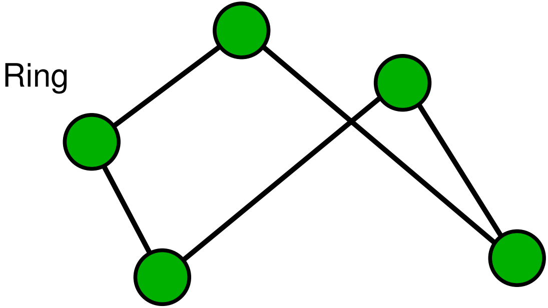
example:

- SONET, FDDI, Token Ring

comments:

- two paths provide some redundancy (a dual ring)

Ring



description: Every node has exactly two branches connected to it, so that they form a (logical) ring.

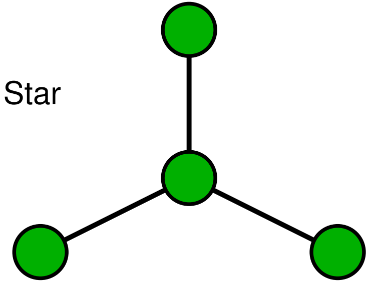
example:

- SONET, FDDI, Token Ring

comments:

- two paths provide some redundancy (a dual ring)

Star



description: peripheral (spoke) nodes are connected to a central (hub) node. All communications is via the hub.

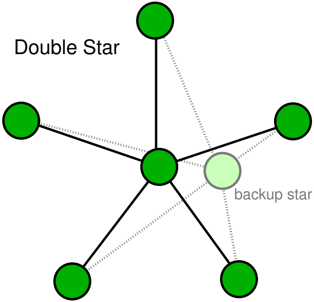
examples:

- physical topology of 10BaseT Ethernet with a hub
- logical topology of 10BaseT Ethernet with a switch

comments:

- hub node failures are critical

Double star



description: two stars, with two hubs, effectively, one is a redundant backup for failures.

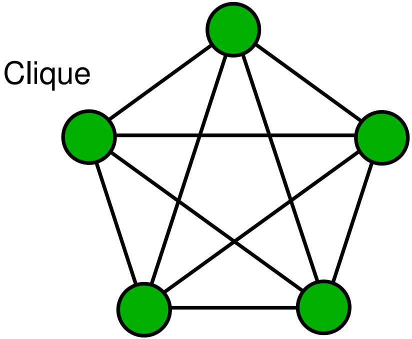
example:

- used for many networks

comments:

- stars are sensitive to failures of hub, or links
- robust to a failure of hub, or single link

Fully connected or clique



description: every node directly connected to every other node (also called a clique).

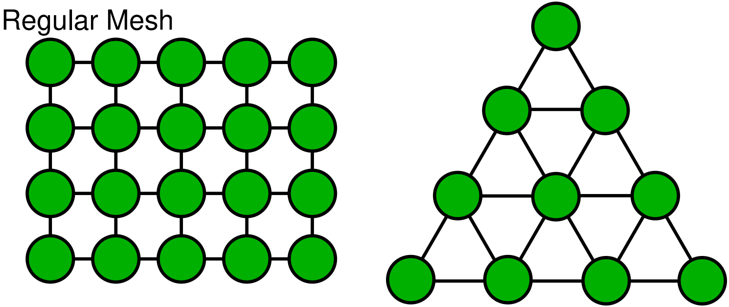
example:

- frame relay network (at a logical level)

comments:

- very robust to failures, but expensive

Mesh



description:

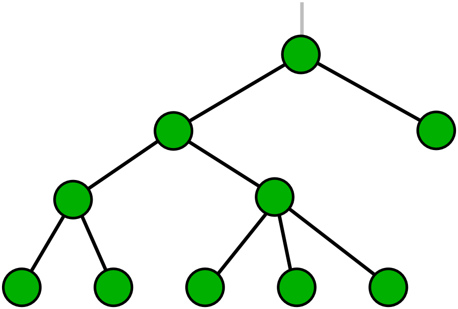
example:

- many real networks are somewhat meshy

comments:

- somewhere between clique, and star
- robust to failures

Tree



description: nodes are arranged as a tree (no loops)

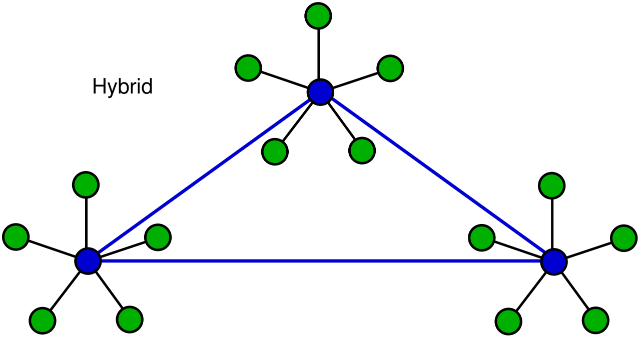
examples:

- shortest path trees in routing
- spanning tree protocol (for switched Ethernets)

comments:

- sensitive to failures

Hybrid



description: A combination of any two or more network topologies in such a way that the resulting network does not have one of the standard forms.
comments:

- a tree connected to a tree is still a tree network
- example is a hierarchical network (as above)

Question

How many possible graphs are there with N nodes?

Simple Set Notation

$$
\begin{array}{rrcl}
\text{membership} & \omega \in U & := & \omega \text{ is in } U \\
\text{subset} & L \subseteq U & := & \text{if } \omega \in L, \text{ then } \omega \in U \\
\text{intersection} & L \cap U & := & \{\omega \,|\, \omega \in L \text{ and } \omega \in U\} \\
\text{union} & L \cup U & := & \{\omega \,|\, \omega \in L \text{ or } \omega \in U\} \\
\text{set difference} & L \backslash U & := & \{\omega \,|\, \omega \in L \text{ and } \omega \notin U\} \\
\text{empty set} & \phi & := & \{\} \\
\text{for all} & \forall \omega & := & \text{do something for all } \omega \\
\text{count} & |U| & := & \text{the number of elements of } U \\
 & & := & \text{cardinality}
\end{array}
$$

Other Notation

I usually use

- lower case for scalars, *e.g.*, x
- lower-case boldface for (column) vectors, *e.g.*, $\mathbf{x}$
- upper-case for matrices or sets, *e.g.*, A

When I write $\mathbf{x} < \mathbf{b}$ I mean every element of $\mathbf{x}$ is less than its corresponding element in $\mathbf{b}$, so

$$x_i < b_i, \quad \forall i$$

and similarly for relational operators, *e.g.*, $\leq$, $\geq$, ...

Graph Notation

An Undirected Graph $G(N, E)$ is

- a set of *nodes* $N = \{1, 2, \ldots, n\}$ (also called *vertices*)

$$|N| = n$$

- a set of *links* $E \subseteq N \times N$ (also called *edges* or *ties*)

$$E \subseteq \{(i, j) : i, j \in N, i \neq j\}$$

- in an undirected graph: links are symmetric

$$(i, j) \in E \Rightarrow (j, i) \in E$$

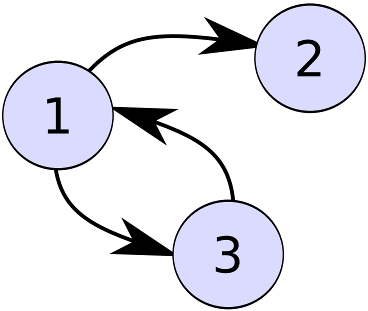
  so we usually only list them in one direction

Graph Notation

A Directed Graph $G(N, E)$ is the same except that

- each link (or *arc*) (i, j) implies a link
  - from *source* i
  - to *destination* j
- The existence or otherwise of (i, j) tells us nothing about (j, i)

Example Directed Graph



$$
\begin{aligned}
N &= \{1, 2, 3\} \\
E &= \{(1,2),\ (1,3),\ (3,1)\}
\end{aligned}
$$

Graph Notation

- The network is defined by the *graph*, $G(N, E)$
- Often we have additional information about links/nodes, *e.g.*,
  - link/node capacities
  - link weights
  - link distances
- Model using functions, *e.g.*, $W(\cdot)$

$$W : E \to \mathbb{R}$$

or, $F(\cdot)$

$$F : N \to \mathbb{R}$$

Some definitions

- *null graph:* a graph with no vertices or edges.
- *empty graph:* graph with no edges.
- *infinite graph:* ∞ edges or vertices (or both)
- *subgraph:* subgraph $G' = G'(N', E')$ of $G(N, E)$ has

$$N' \subseteq N \text{ and } E' \subseteq E$$

- *spanning subgraph:* subgraph G' of $G(N, E)$

$$N' = N$$

- *planar graph:* one that can be drawn in the Euclidean plane without any crossing.

Definitions for digraphs

- *digraph:* a directed graph.
- *arc:* a directed edge with a *source* or *origin* and *destination* or *target*
- *DAG:* directed acyclic graph – a directed graph with no cycles.

Paths (Walks)

- a *path* is an ordered series of links such that the end of one is the beginning of the next, *i.e.*,

$$P = (i_0, i_1), (i_1, i_2), \ldots, (i_{n-2}, i_{n-1})$$

  where $i_0, i_1, \ldots, i_{n-1} \in N$
  and $(i_0, i_1), (i_1, i_2), \ldots, (i_{n-2}, i_{n-1}) \in E$.
- we usually abbreviate it to something like

$$P = i_0 - i_1 - i_2 - \cdots - i_{n-1}$$

  and in fact it can also be defined as a series of nodes, such that the appropriate connections exist.
- a *loop free* or *simple* path contains each node at most once.
  - often, paths are defined to be loop free
  - *walks* allow cycles

Paths example

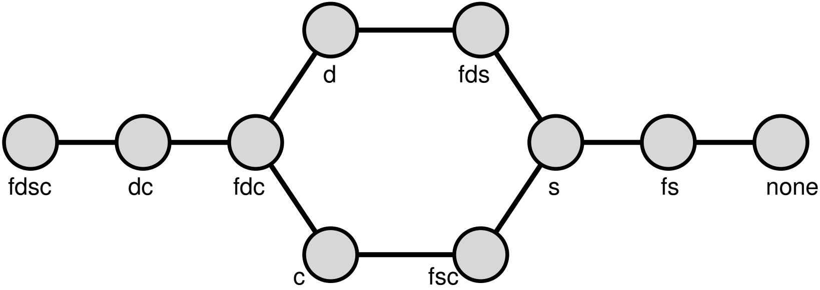
Consider the following well known problem: "A ferryman has been charged with taking a dog, a sheep, and a cabbage across the river. His rowboat can only take one at a time (plus himself). He cannot leave the dog with the sheep, or the sheep with the cabbage. How many ways can he make the transfer without repeating himself."

Denote

- Ferryman f
- Dog d
- Sheep s
- Cabbage c

Paths example

Label each possible state by who is on the left bank of the river. The following graph shows the only possible transitions.



There are two possible loop-free paths the ferryman can take.

Section 2

Graph Representations

Graph Representations

- The way you draw a graph doesn't matter
  - the picture of the graph is not the graph
  - two graphs can be isomorphic

Graph Isomorphism

Isomorphism expresses the fact that often the labels on a graph are arbitrary.

Definition

A *isomorphism* between two graphs G and H is a bijection[1] between the nodes $N(G)$ and $N(H)$ of the two graphs

$$f : N(G) \to N(H),$$

such that any two vertices $u, v \in N(G)$ are adjacent if and only if $f(u), f(v) \in N(H)$ are adjacent. We say two graphs are *isomorphic* is such a function exists.

So an isomorphism is a change of labels, preserving the edges.

[1] Loosely, a bijection is a function between two sets where each element of one set is paired with exactly one element of the other set and visa versa.

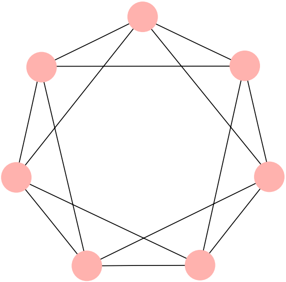
Graph Isomorphism Example 1

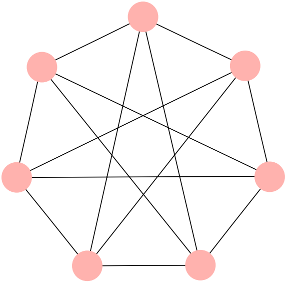
Are these two graphs isomorphic?

Graph Isomorphism Example 2

Are these two graphs isomorphic?

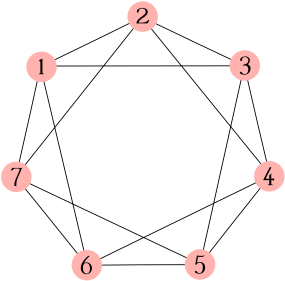
Graph Isomorphism Example 3



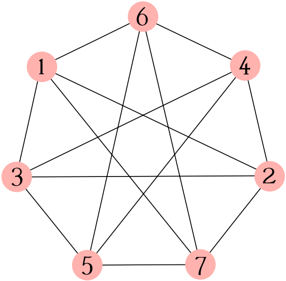


Are these two graphs isomorphic?

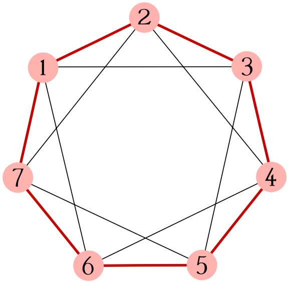
Graph Isomorphism Example 3



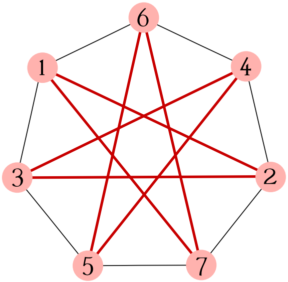
Yes!



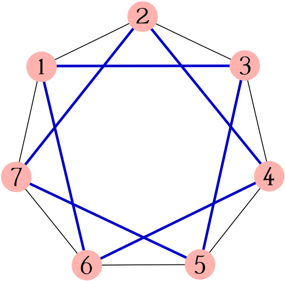
Graph Isomorphism Example 3



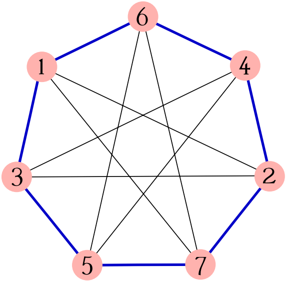
Yes!



Graph Isomorphism Example 3



Yes!



Graph Isomorphism

The graph isomorphism problem is very interesting

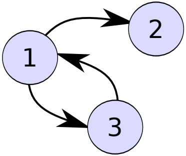
- There is no known polynomial time algorithm
- Babai has proposed (2016) a quasi-polynomial time algorithm, but it isn't verified yet
- It is thought that if $N \neq NP$, then this might be an intermediate problem, which is not NP-complete, but is harder than polynomial time.

Graph Representations

- The way you draw a graph doesn't matter
  - two pictures of graphs can be isomorphic
- We talk about graphs mathematically, but when we have to implement any algorithm, we need to store a graph in a computer
- Graph Representations are *equivalent* ways to represent the information contained in a graph
- Each has advantages and disadvantages
  - cost of memory/storage
  - computational cost of various operations
  - computational cost of complex algorithms

Edge List Representation

As per the mathematical definition $G(N, E)$
Example directed graph



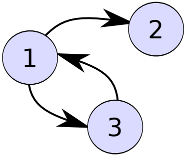
$$
\begin{aligned}
N &= \{1, 2, 3\} \\
E &= \{(1,2),\ (1,3),\ (3,1)\}
\end{aligned}
$$

Adjacency Matrix Representation

Use a $|N| \times |N|$ binary *indicator* matrix

$$A_{ij} = \begin{cases} 1, & \text{if } (i,j) \in E, \\ 0, & \text{otherwise.} \end{cases}$$

Example directed graph



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

We could store a weight for each edge in a *weighted* adjacency matrix, or could even store a pointer to an edge object.

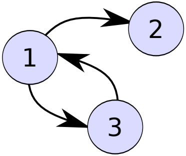
Incidence Matrix Representation

Use a $|N| \times |E|$ matrix to say which edges use which nodes

$$H_{ij} = \begin{cases} 1, & \text{if node } j \in \text{ edge } i, \\ 0, & \text{otherwise.} \end{cases}$$

For a directed graph use -1 to indicate source node

Example directed graph

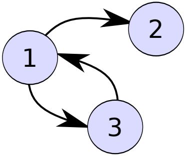


$$H = \begin{array}{c c} & \begin{array}{ccc} node_1 & node_2 & nodes_3 \end{array} \\ \begin{array}{c} edge_{12} \\ edge_{13} \\ edge_{31} \end{array} & \begin{pmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \\ 1 & 0 & -1 \end{pmatrix} \end{array}$$

Neighbourhood/Adjacency List Representation

Create a list of nodes, and for each node, list its neighbours

Example directed graph



$$\begin{aligned} 1 &: \quad 2, 3 \\ 2 &: \\ 3 &: \quad 1 \end{aligned}$$

Basic Graph Operations

```
add_node(G, i)
remove_node(G, i)
add_edge(G, (i,j))
remove_edge(G, (i,j))
adjacent(G, i,j)
...
```

Basic Graph Operation Costs

	Edge List	Adjacency Matrix	Neighbourhood List
Storage	$O(\lvert N\rvert + \lvert E\rvert)$	$O(\lvert N\rvert^2)$	$O(\lvert N\rvert + \lvert E\rvert)$
Add node	$O(1)$	$O(\lvert N\rvert^2)$	$O(1)$
Add edge	$O(1)$	$O(1)$	$O(1)$
Remove node	$O(\lvert E\rvert)$	$O(\lvert N\rvert^2)$	$O(\lvert E\rvert)$
Remove edge	$O(\lvert E\rvert)$	$O(1)$	$O(\lvert N\rvert)$
Adjacent	$O(\lvert E\rvert)$	$O(1)$	$O(\lvert E\rvert)$

- You can make adjacency matrices more attractive using sparse matrices.
- Incidence matrices aren't very efficient because you (effectively) have to change the entire matrix to add or remove an edge of node, but they are useful mathematical representations
- Preferred option depends on what you are doing with the graph, and how sparse/dense the graph is.

Further reading I