Modeling, Analysis and Optimization of Networks (part 2 : Design)

Alberto Ceselli
alberto.ceselli@unimi.it

Università degli Studi di Milano
Dipartimento di Informatica

Doctoral School in Computer Science

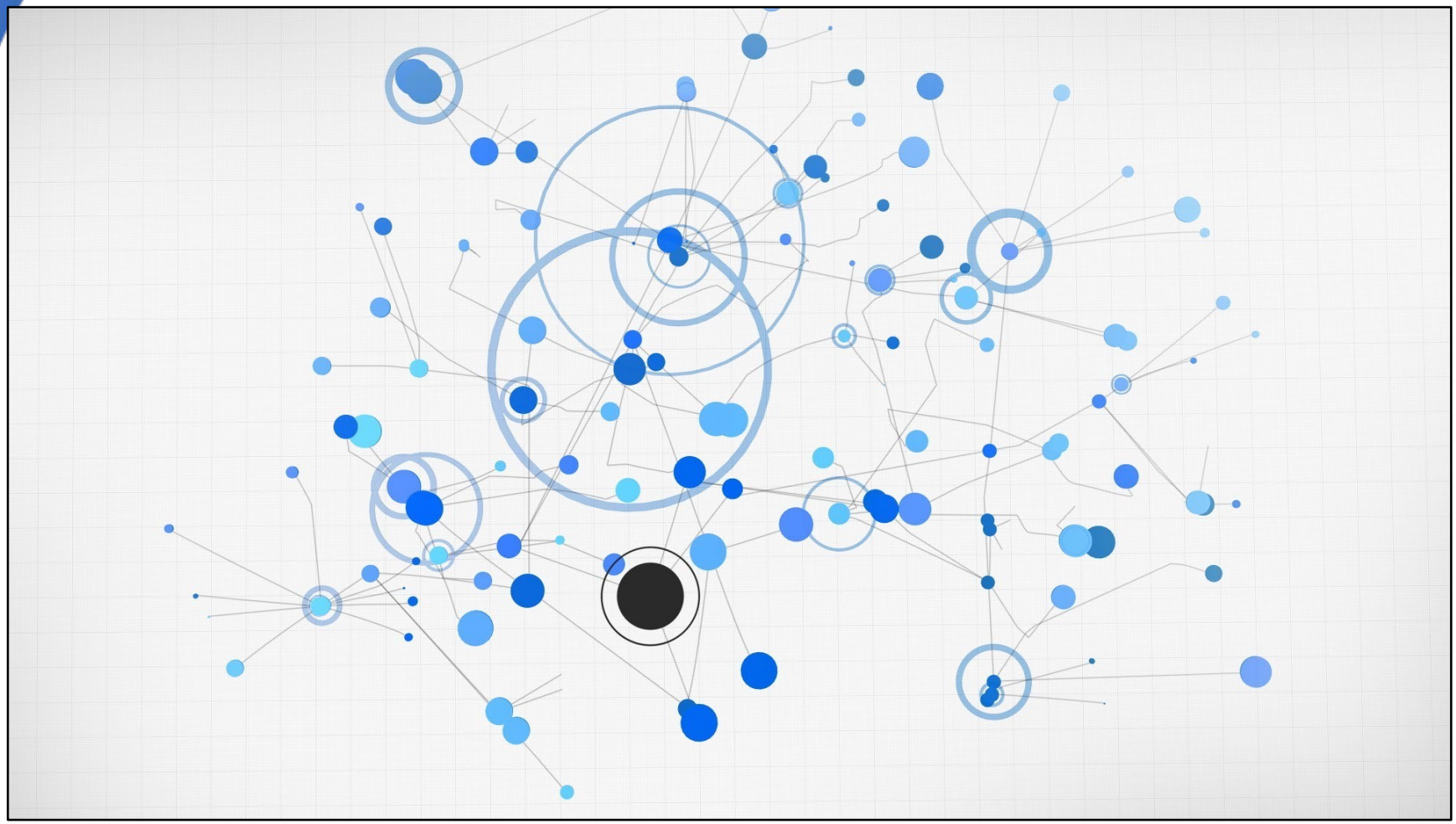
A.A. 2016/2017

What's a Network ?

- Dictionary.com:
  any netlike combination of filaments, lines, veins, passages, or the like
- Wikipedia:
  disambiguation page with 30 entries in 4 categories
- Let's try images !

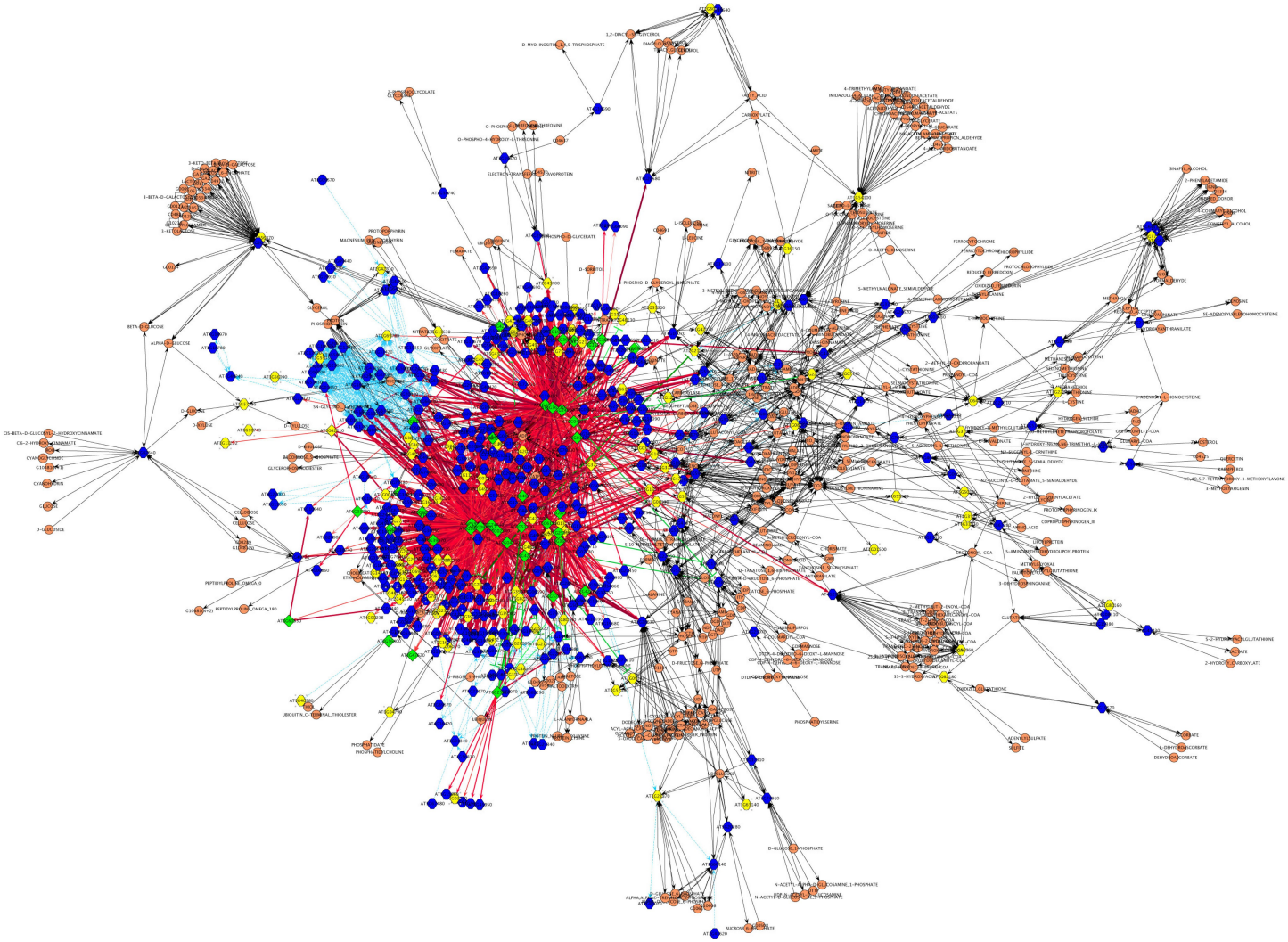
First image in Google :

A few images after: artistic ones



A few images after: artistic ones

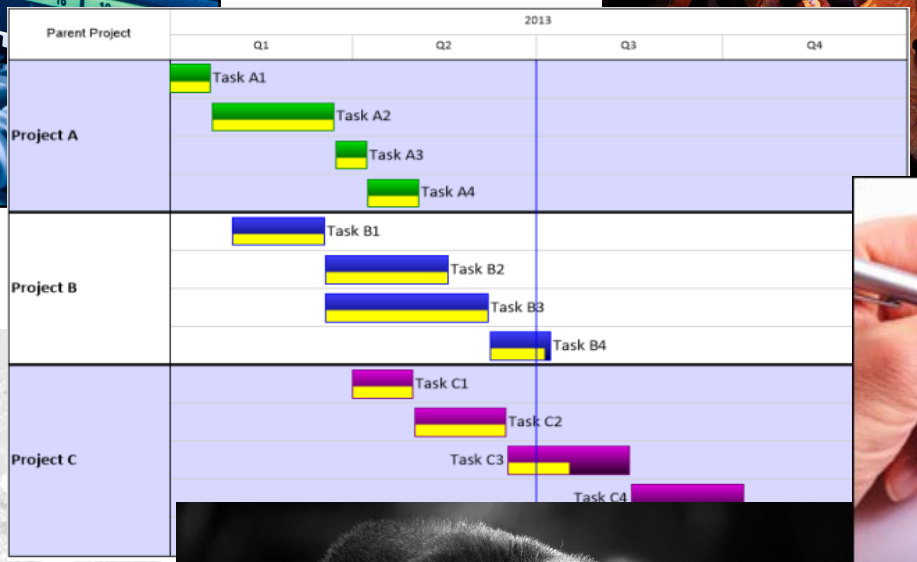
… and very different domains

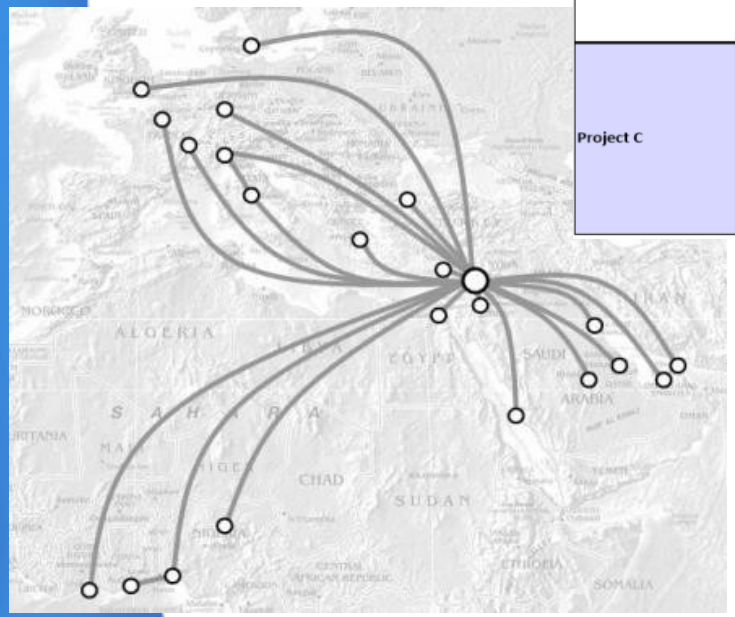


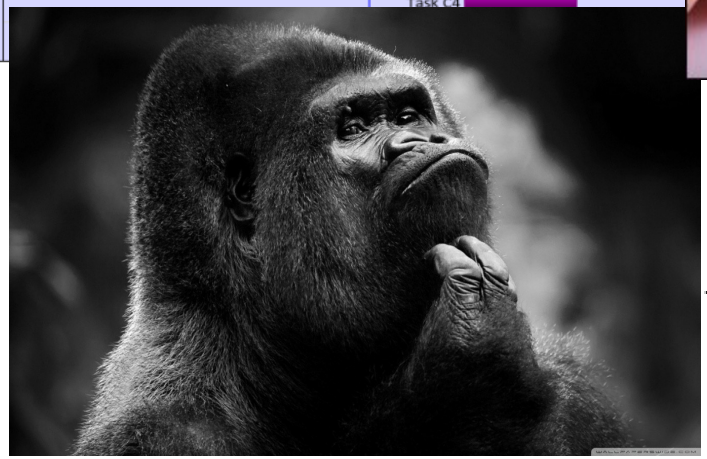
… only rank 26!


9 10 11 12 13 14 15 16 17 18 19
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
11X
13X

Trivia: how many networks do you see?







$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix}.$$

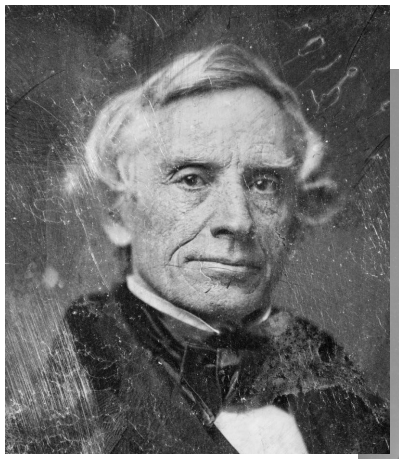
To remember:

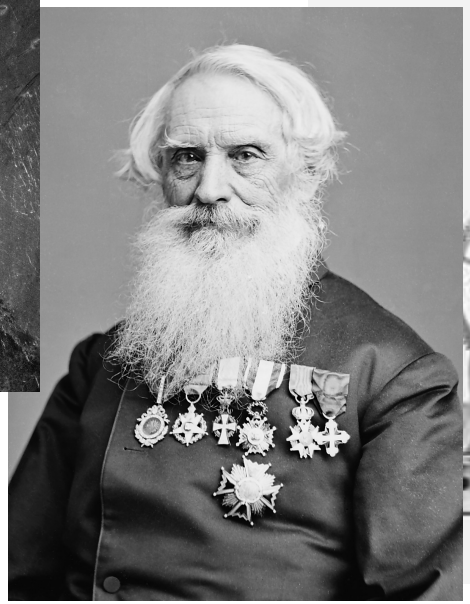
1 - Networks are pervasive !

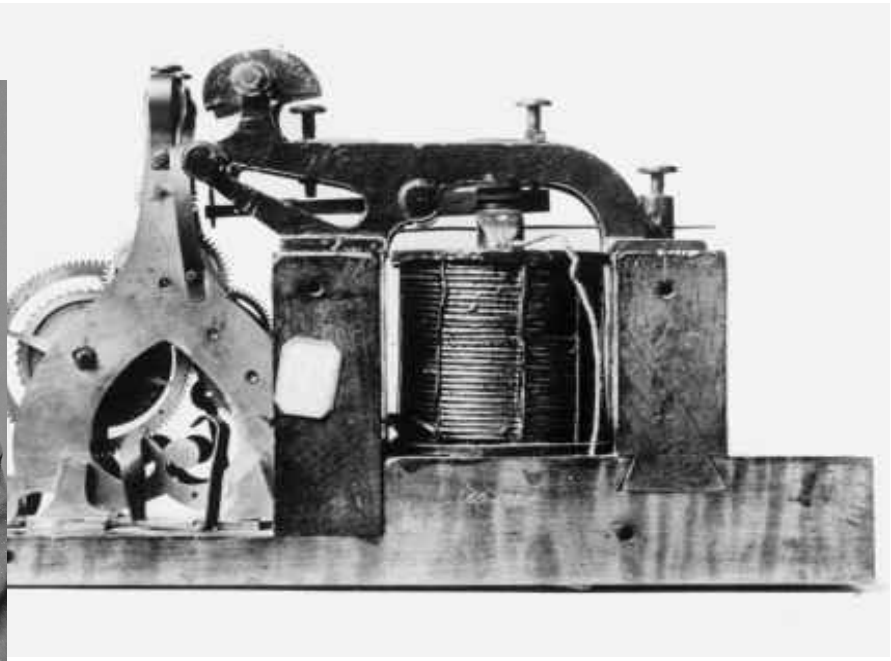
2 - by « network » we mean far more than computers connected by cables;

And what about computer networks?

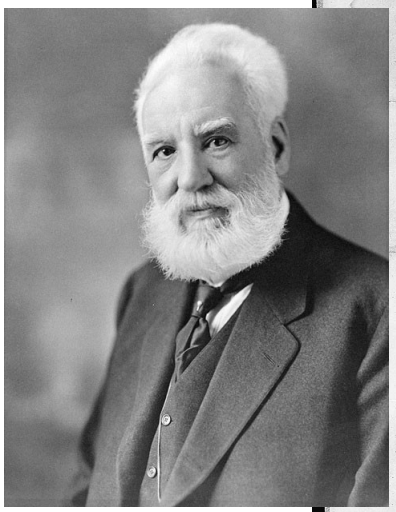
- Samuel Morse (1791 – 1872)
  - professor of arts and design at New York University
  - in 1835, proved that signals could be transmitted by wire …

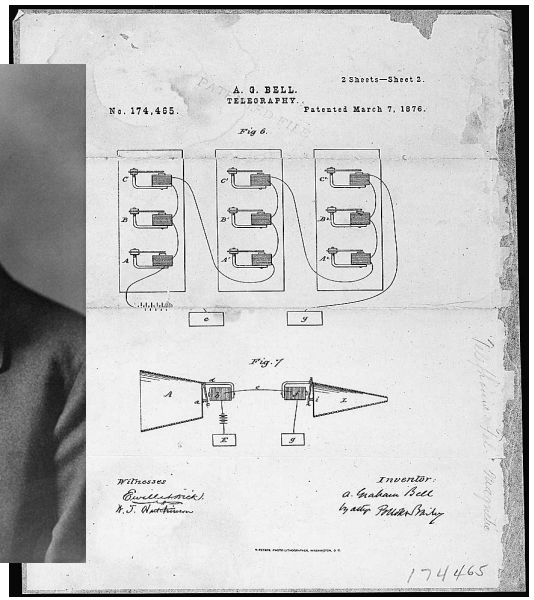






- Alexander Graham Bell (1847 - 1922), and Thomas Watson
  - initially working on multi-tone telegraphy (many signals on the same line at the same time)
  - multi-tone then became the telephone!
  - patented during 1876 ...

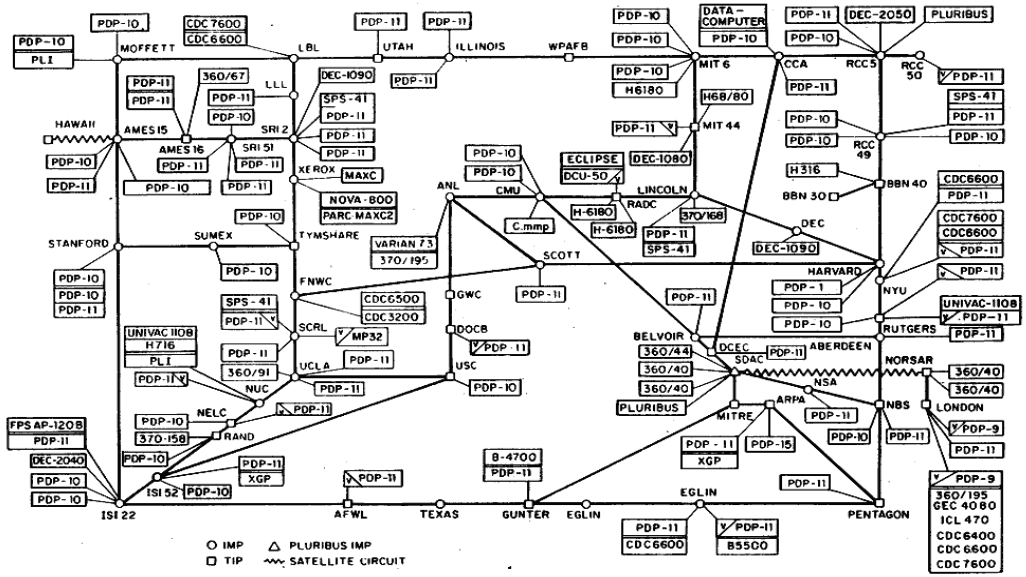




- The Advanced Research Projects Agency Network (ARPANet),
  - J. C. R. Licklider, articulated the ideas in his January 1960 paper, Man-Computer Symbiosis,
  - First operational packet switching network between computers
  - actually deployed in 1969
  - the first message « LO(G) » yielded a system crash!

29OCT69	2100	LOADED OP. PROGRAM EOI2 BBN BARKER BBN	CSK
	22:30	talked to SRI Host to Host	CSK
		Left imp. program running after sending a host dead message to imp.	CSK

ARPANET LOGICAL MAP, MARCH 1977



(PLEASE NOTE THAT WHILE THIS MAP SHOWS THE HOST POPULATION OF THE NETWORK ACCORDING TO THE BEST INFORMATION OBTAINABLE, NO CLAIM CAN BE MADE FOR ITS ACCURACY)

NAMES SHOWN ARE IMP NAMES, NOT (NECESSARILY) HOST NAMES

- WWW: a system of interlinked hypertext documents
  - Tim Berners-Lee wrote a proposal in March 1989
  - Robert Cailliau proposed in 1990 to use hypertexts …
- Google (Larry Page and Sergey Brin in 1998):
  - over 1 million servers, at least 12 data centers located only in the U.S.A. (in 2012);
  - internet search: about 24 PB of user data daily (in 2012);
  - cloud computing: managing and balancing distributed
  - resources.
- Facebook (Mark Zuckerberg et al. in 2004):
  - handling social networks of billion users (1.6×10^9?)

To remember:

1 - Networks are pervasive !

2 - by « network » we mean far more than computers connected by cables;

3 - network problems moved from technologies to applications, and now to services;

4 - networks are in general too complex to be managed by humans without decision support systems.

Modeling, Analysis and Optimization of Networks

- Either « transverse » or « specialization » course offered by D.I.
- (Far) More on modeling and structural properties than on specific techniques and technologies
- Different editions cover different sub-topics

- This year: **network design**

Course objectives

- finding network (design) structures in applications
- modeling as design problems on networks
- main theoretical results on trees and location algorithms
- overview of tools for solving network design problems

PART I: Finding components and designing components

Example 1 : finding micells

- We are performing molecular dynamics simulations
- We are given a set of proteins, each composed by several atoms
- These proteins interact, and during simulation, tend to form clusters
- How to find, at a certain point in time, which clusters are there ?

Example 2 : detecting block structure in matrices

- We are given a certain matrix
- We would like to understand if it has a block diagonal structure …
- … in order to exploit decomposition in linear algebra algorithms

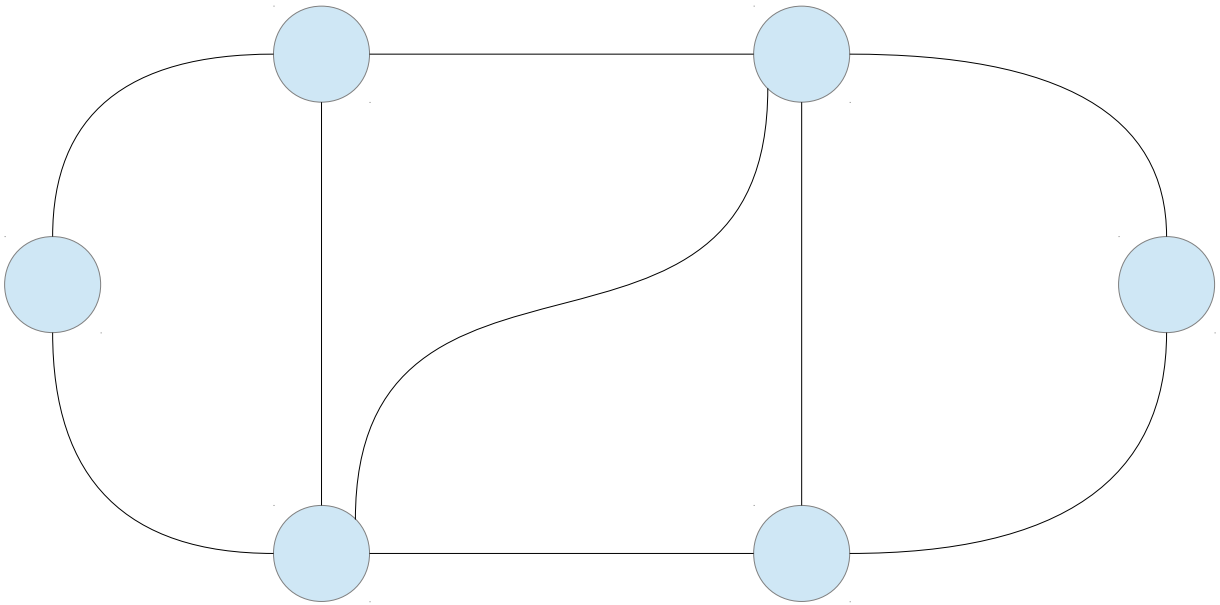
$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & b_{11} & b_{12} & b_{13} & 0 & 0 \\ 0 & 0 & b_{21} & b_{22} & b_{23} & 0 & 0 \\ 0 & 0 & b_{31} & b_{32} & b_{33} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & c_{11} & c_{12} \\ 0 & 0 & 0 & 0 & 0 & c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} A & 0 & 0 \\ 0 & B & 0 \\ 0 & 0 & C \end{bmatrix}$$

What's the common factor among the two problems ?

- Search for connected components in a suitable graph !
- Recall : search in a graph G(V,E) …
- Find connected components :
  - For each i in V, set c[i] = 0
  - Set k = 0, set R = V
  - While R is not empty
    - Pick a vertex i in R
    - Set c[i] = k
    - Perform a search in R starting from i: mark every visited node j by setting c[j] = k
    - Set R = R \ {j | c[j] == k}

Notation

- An undirected graph G(V,E)

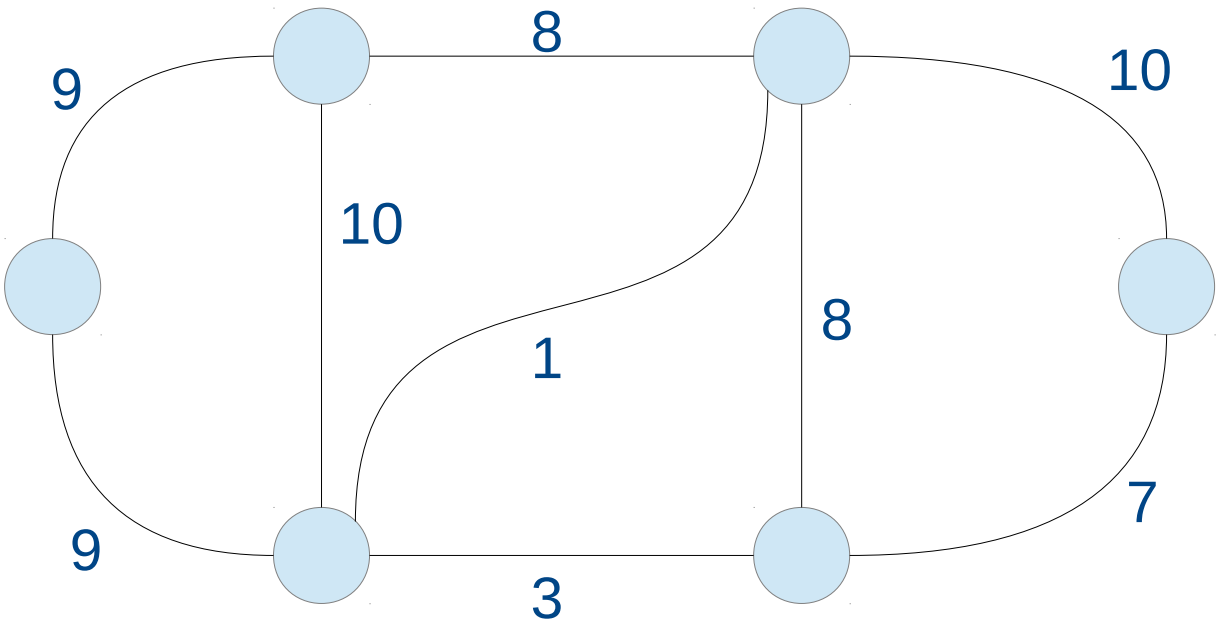


Designing components

- What if components are not just given, but need to be *designed* ?
- Given a graph G(V,E)
- Given a weight function c:E $\rightarrow \mathbb{R}_+$
- Find a subgraph
  - Forming a single connected component
  - Of minimum overall weight
- i.e. a *spanning tree* !

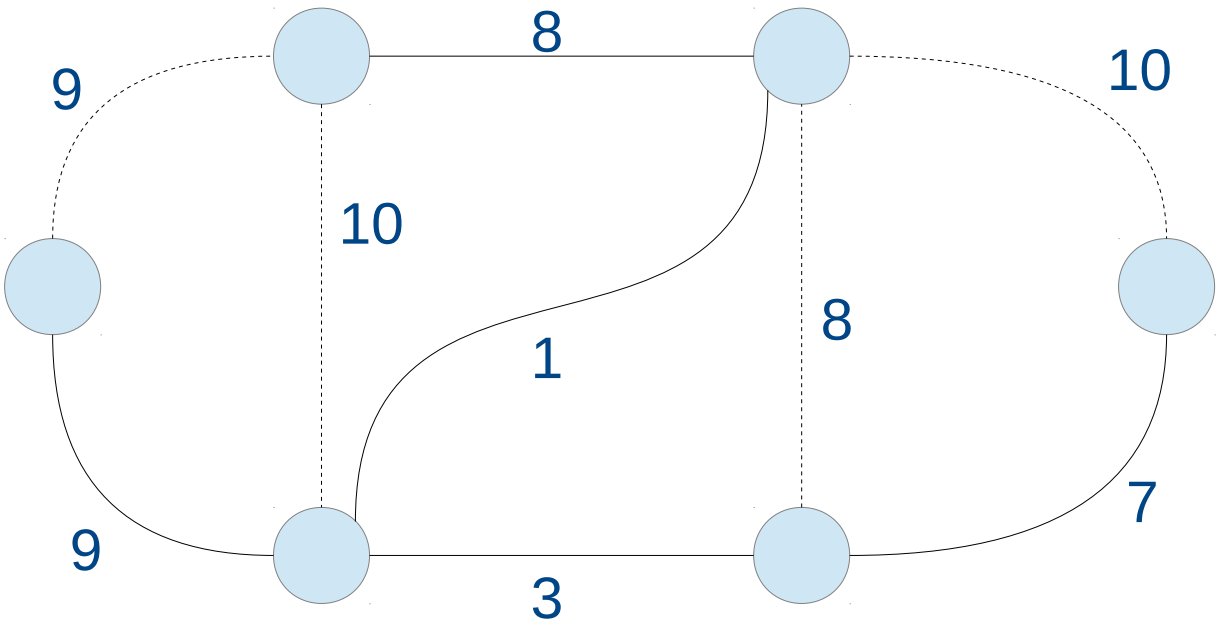
Notation

- An undirected graph G(V,E)
- A function c: E → ℝ (edge costs)



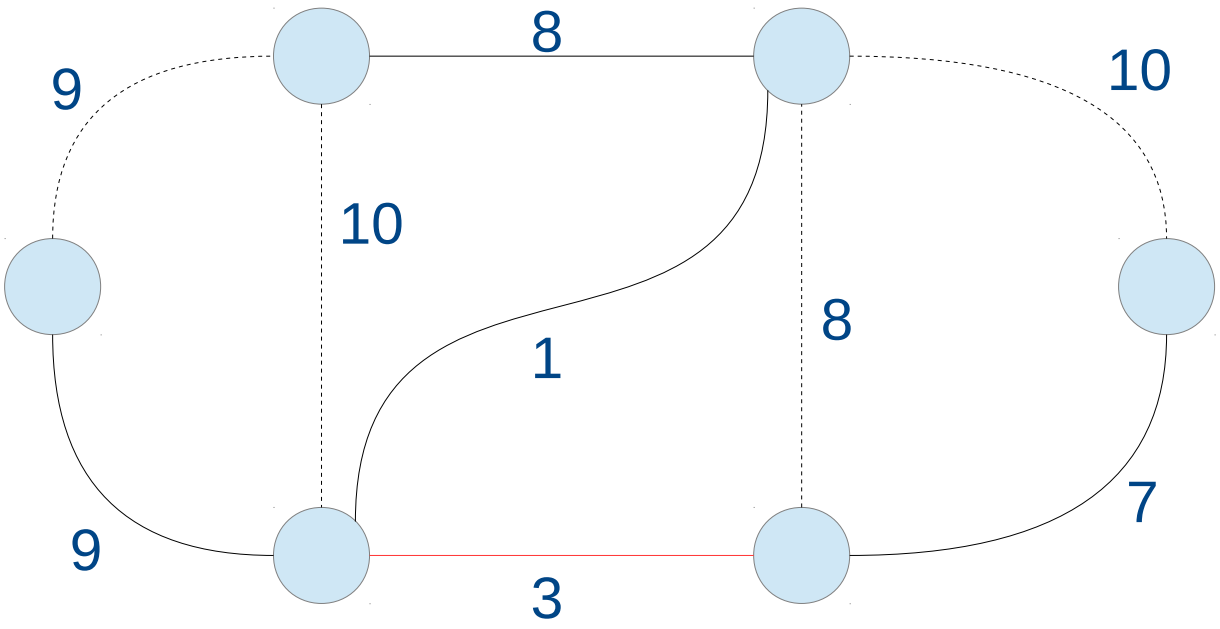
Notation : a spanning tree

- An undirected graph G(V,E)
- A function c: E → ℝ (edge costs)



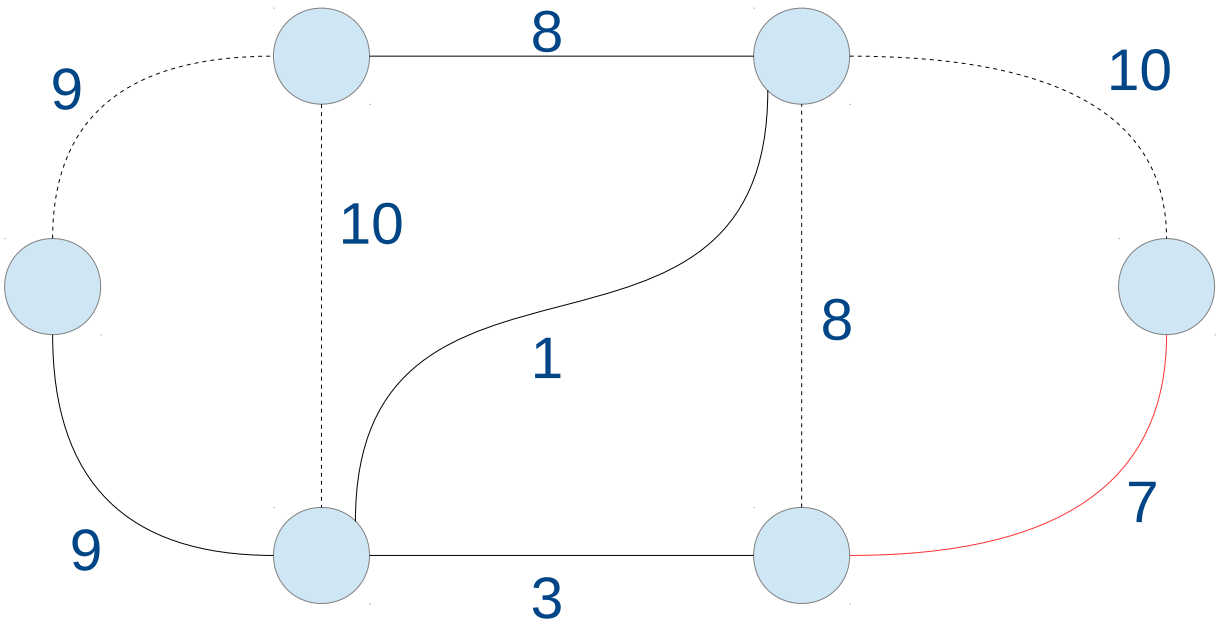
Notation : not a spanning tree (disconnected)

- An undirected graph G(V,E)
- A function c: E → ℝ (edge costs)



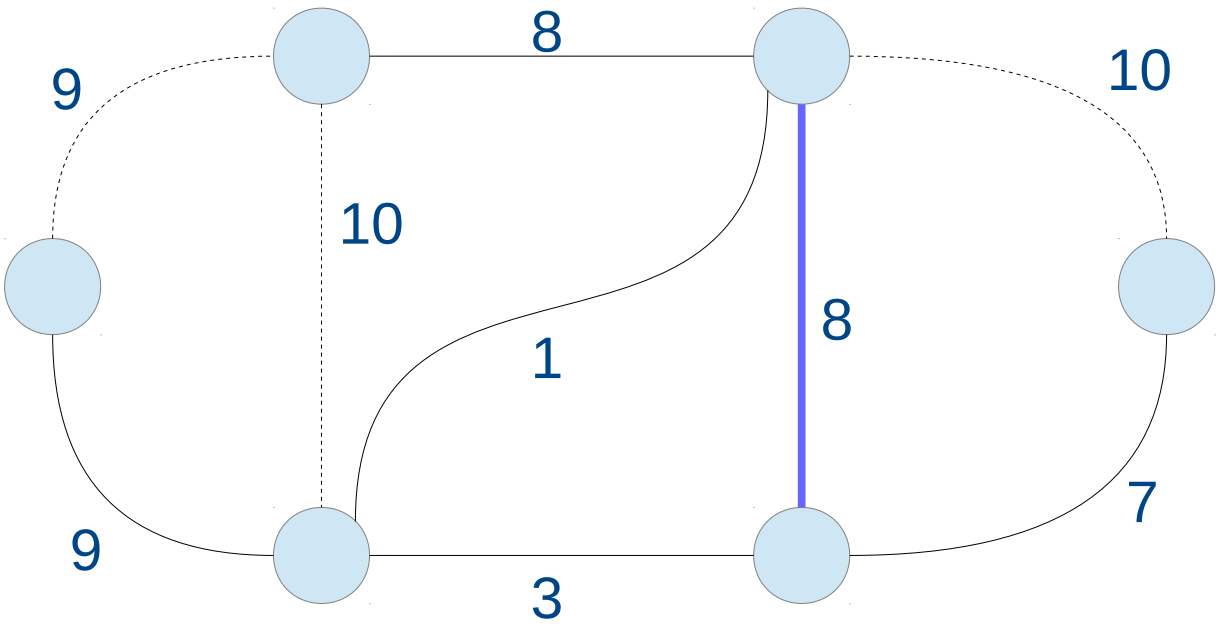
Notation : not a spanning tree (non spanning)

- An undirected graph G(V,E)
- A function c: E → ℝ (edge costs)



Notation : not a spanning tree (loops)

- An undirected graph G(V,E)
- A function c: E → ℝ (edge costs)



Observations

a - How many vertices are included in a spanning tree ?

b - How many edges ?

c - How many loops ?

→ (a) |V| vertices, (b) |V|-1 edges, (c) 0 loops !

(b) + (c) → (a)

Example 3 : message passing
(Ahuja, Magnanti, Orlin « Network Flows »)

- An intelligence service has n agents in a nonfriendly country. Each agent knows some of the other agents and has in place procedures for arranging a rendezvous with anyone he knows.
- For each such possible rendezvous, say between agent i and agent j, any message passed between these agents will fall into hostile hands with a certain probability p_{ij}.
- The group leader wants to transmit a confidential message among all the agents while minimizing the total probability that the message is intercepted.

Example 4 : reducing data storage

(Ahuja, Magnanti, Orlin « Network Flows »)

- In several different application contexts, we wish to store data specified in the form of a two-dimensional array more efficiently than storing all the elements of the array (to save memory space).
- We assume that the rows of the array have many similar entries and differ only at a few places. One such situation arises in the sequence of amino acids in a protein found in the mitochondria of different animals and higher plants.
- Since the entities in the rows are similar, one approach for saving memory is to store one row, called the reference row, completely, and to store only the differences between some of the rows so that we can derive each row from these differences and the reference row.
- How to choose the reference rows ?

The « roots » of spanning trees

- Cut Optimality Conditions
- Path Optimality Conditions
- Kruskal Algorithm
- Prim Algorithm
- Boruvka Algorithm

Cut optimality conditions

- Remove any edge from a spanning tree, and the tree disconnects in two components [S,T]
- The edges of G connecting S with T form a *cut*
- **Cut Optimality Conditions (COC)** : a spanning tree P* is minimum if and only if
  - For every tree edge (i,j) in P*,

    $c(i,j) \leq c(k,l)$ for every edge (k,l) in the cut [S,T] induced by the removal of (i,j)
  - (proof on the blackboard)

Path optimality conditions

- Take any pair of vertices (i,j) and a spanning tree P : there is a single path in P connecting them (otherwise → loop)
- **Path Optimality Conditions (POC)** : a spanning tree P* is minimum if and only if
  - For every non-tree edge (k,l) in G,

    $c(i,j) \leq c(k,l)$ for every edge (i,j) in the path from from k to l in P*
  - (proof on the blackboard)

A comment on optimality conditions

- Cut optimality conditions yield an « external » representation of a minimum spanning tree (i.e. look to non-tree edges)
- Cut optimality conditions yield an « internal » representation of a minimum spanning tree (i.e. look to tree edges)
- Similar conditions hold for the *maximum* spanning tree problem

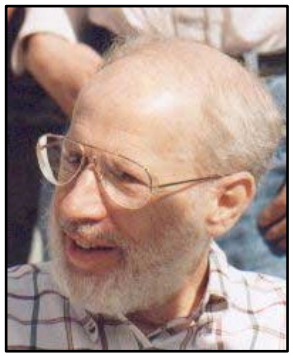
Straightforward path-based algorithm

- Start with a random spanning tree P
- Iteratively,
  - check if POC are respected
  - If they are, then P is optimal → stop
  - else, find a pair of edges violating them and swap, obtaining a new P
- Drawback : not polynomially bounded.

Kruskal's algorithm

- Build an optimal spanning tree P* from scratch, adding one edge at a time
- Sort edges by non-decreasing cost
- Set L = empty set
- For t = 1 .. |V|-1
  - Pick the minimum cost edge (i,j) from E \ L that is not forming loops with edges in L
  - Add (i,j) to L
- P* = L

  (correctness proof at the blackboard)
- Computational complexity : O(|E||V|)
  (proof at the blackboard)



Prim's algorithm

- Build an optimal spanning tree P* from scratch, adding one edge at a time
- Choose a random vertex i
- Set S = {i}, L = empty set
- For t = 2 .. |V|
  - Pick the minimum cost edge (i,j) in the cut [S, V\S]
  - Add j to S
  - Add (i,j) to L
- P* = L

  (correctness proof at the blackboard)
- Computational complexity : O(|E||V|)
  (proof at the blackboard)

Boruvka's algorithm (1/2)

- nearest-neighbor (N, i, j) →
  - Input : a tree spanning the vertices N
  - Output : an edge (i, j) with the minimum cost among all edges from N
    cij = min{clk:(l,k) in E, l in N and k not in N}
  - *Scan all the edges in the adjacency lists of nodes in N, and find a min cost edge among those arcs that have one endpoint not belonging to N.*
- merge(N, M) →
  - Input : two vertex sets N and M
  - Output : merge N and M, appending M to N
  - *Append lists*

Boruvka's algorithm (2/2)

- For each i in V, let N(i) := { i }
- Let P* := empty set ; Let K := V;
- While |P*| < |V|-1
  - Let E := empty set
  - For each N(k) do
    nearest-neighbor(N(k), i(k), j(k))
    Let E:= E U { (i(k), j(k)) }
  - For each (i , j) in E do
    If i and j belong to (different) trees N(i) and N(j)
    - Merge(N(i), N(j))
    - Let P* := P* U {(i , j)}

Comparison of Algorithms

Algorithm	Data Structure	Complexity
Naive path based	NN	No polynomial bound
Kruskal	List	O(|E||V|)
Kruskal	Lists w. eff. union-find	O(|E| Q + edge sorting)
Prim	List	O(|E||V|)
Prim	Fibonacci heap	O(|E| + |V| log |V|)
Prim	Johnson's data structure	O(|E| log log C)
Boruvka	Circular doub. linked lists	O(|E| log |V|)
Boruvka	Chazelle's data structure	O(|E| alpha(|E|,|V|))