

ELU 501

Data science, graph theory and social network studies

Yannis Haralambous (IMT Atlantique)

October 27, 2017

ELU 501

Data science, graph theory and social network studies

Yannis Haralambous (IMT Atlantique)

October 27, 2017

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

Part I

Lecture 1 Basic graph notions and algorithms

First definitions

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

- Let V be an arbitrary set and let $V^{(2)}$ be the set of sets of two elements $\{x_1, x_2\}$ with $x_1, x_2 \in V$. An *undirected graph* 🇫🇷 *graphe non orienté* over V is a pair $G = (V, E)$ where $E \subseteq V^{(2)}$.
- V is the set of *vertices* 🇫🇷 *sommet* of G and E the set of *edges* 🇫🇷 *arête*.
- We will denote by x_1x_2 the edge $\{x_1, x_2\} \in E$, x_1 and x_2 are called *extremities* 🇫🇷 *extrémité*.
- An undirected graph is called *simple* 🇫🇷 *graphe simple* if it has neither loop (an edge with twice the same vertex) nor multiple edge (two edges with identical extremities).

First definitions

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

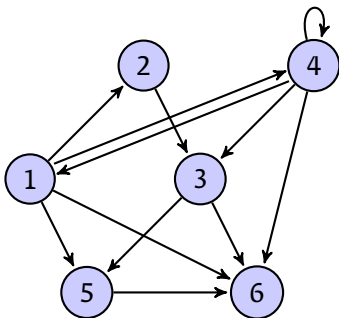
Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

- Let V be an arbitrary set and let V^2 the set of pairs of elements (x_1, x_2) with $x_1, x_2 \in V$. A *directed graph* or *digraph* $\langle \text{graphe orienté} \rangle$ V is a pair $G = (V, E)$ where $E \subseteq V^2$.
- V is the set of *vertices* of G and E the set of its *edges* or *arrows* $\langle \text{arc} \rangle$.
- We will write $x_1 x_2$ the edge $(x_1, x_2) \in E$, x_1 et x_2 are its *extremities*, we will say that x_1 is a *predecessor* $\langle \text{prédécesseur} \rangle$ of x_2 (and x_2 a *successor* $\langle \text{successeur} \rangle$ of x_1).
- A directed graph is called *strict* $\langle \text{graphe strict} \rangle$ if it has neither loop nor multiple edge.
- A graph (weither directed or not) (G, ρ) is called *ordered* $\langle \text{graphe ordonné} \rangle$ if there exists an order of vertices ρ (i.e., an injective application $\rho: G \rightarrow \mathbb{N}$).

Adjacency matrix, weighted graph

- An **adjacency matrix** ($\text{matrice d'adjacence}$) A is a matrix of values $a_{i,j} = n$ when there are n edges between vertices x_i and x_j . (i th line, j th column.)



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

Basic graph notions

Vertices, edges

Adjacency matrix

Weighted graph

Co-citation and bibliographic coupling

Orientation, acyclicity, topological sort

Discretion, completeness

Cliques and stables

Bipartite graphs

Incidence matrix

Trees, forests

Returning to graphs: average degree, density, large networks

Paths

Graph Laplacian

Adjacency matrix, weighted graph

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix

Weighted graph
Co-citation and
bibliographic
coupling

Orientation,
acyclicity,
topological sort

Discretion,
completeness

Cliques and
stables

Bipartite graphs

Incidence matrix

Trees, forests

Returning to
graphs: average
degree, density,
large networks

Paths

Graph Laplacian

- Whenever the graph is simple, the diagonal of the adjacency matrix is zero and $a_{i,j} \in \mathbb{Z}/2\mathbb{Z}$.
- If the graph is undirected, the adjacency matrix is symmetric.
- A *weighted graph* $\langle \text{graphe pondéré} \rangle$ if a graph the vertices and/or edges of which have attached *weights* $\langle \text{poids} \rangle$.
- Any unweighted not simple graph can be replaced by a weighted simple graph.

Co-citation and bibliographic coupling

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix

Weighted graph

Co-citation and
bibliographic
coupling

Orientation,
acyclicity,
topological sort

Discretion,
completeness

Cliques and
stable

Bipartite graphs




Incidence matrix

Trees, forests

Returning to
graphs: average
degree, density,
large networks

Paths

Graph Laplacian

- Let v_i and v_j be vertices of a graph and A its adjacency matrix. If starting from a third vertex v_k you have simultaneously edges to v_i and v_j , you will have $a_{k,i} = 1$ and $a_{k,j} = 1$, so that $a_{k,i}a_{k,j} = 1$.
- We will call $c_{i,j} = \sum_{k=1}^n a_{k,i}a_{k,j}$ the *co-citation index* ⟨ indice de cocitation⟩ (k cites i and j) of v_i and v_j . Co-citation indices are the values of the *co-citation matrix* ⟨ matrice de cocitation⟩ C . (Example: $c_{i,j}$ can be the number of articles citing simultaneously i et j , and $c_{i,i}$ the global number of citations of i including self-citations.)
- By replacing $a_{k,j}$ by ${}^t(A)_{j,k}$ we get $C = A \cdot {}^tA$.
- The graph having C as co-citation matrix (minus the diagonal) is called *co-citation graph* ⟨ graphe de cocitation⟩ of A .
- Co-citation matrices allow us to measure similarity of cited articles.

Co-citation and bibliographic coupling

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix

Weighted graph

Co-citation and
bibliographic
coupling

Orientation,
acyclicity,
topological sort

Discretion,
completeness

Cliques and
stables

Bipartite graphs

Incidence matrix

Trees, forests

Returning to
graphs: average
degree, density,
large networks

Paths

Graph Laplacian

- By re-iterating this operation the other way around (we count the k cited by i and j) we get the **bibliographic coupling graph** $\langle \text{graphe de couplage bibliographique} \rangle$ of matrix $B = {}^t A \cdot A$. ($b_{i,j}$: how many papers are cited simultaneously by i and j ? $b_{i,i}$: how many papers does i cite?)
- Question: what is the best similarity indicator? Cocitation ou bibliographic coupling?
- Answer: to have strong coc. you need to be cited a lot, to have strong b.c., you need big bibliographies. The size of bibliographies is varying less than the number of citations, therefore c.b. is a more uniform indicator.
- On the other hand, b.c. can be calculated at once but for b.c. you need to wait a few years...
- Other applications: the Web (citation \rightarrow hyperlink).

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling





Orientation,
acyclicity,
topological sort

Discretion,
completeness
Cliques and
stables

Bipartite graphs
Incidence matrix

Trees, forests
Returning to
graphs: average
degree, density,
large networks

Paths
Graph Laplacian

- In a directed graph, a *cycle*  *cycle* is a sequence of edges $x_0x_1, x_1x_2, \dots, x_{n-1}x_n, x_nx_0$.
- A directed graph G is called *acyclic*  *graphe acyclique* (or *dag = directed acyclic graph*) if contains no cycles.
- A vertex is a *source*  *source* if it has no predecessor.
- A vertex is a *sink*  *puits* if it has no successor.

Proposition

Any finite dag has always at least one source and one sink.

Topological sort, adjacency matrix of a dag

- We call **topological sort** (*tri topologique*) of a directed graph $G = (V, E)$ a bijection $f: V \rightarrow \{1, \dots, |G|\}$ such that if x is a predecessor of y , then $f(x) < f(y)$.

Proposition

A directed graph is acyclic iff it has a topological sort.

- The adjacency matrix of a dag becomes **strictly triangular** (triangular with zero diagonal) when we order vertices according to a topological sort.

Proposition

Eigenvalues of an adjacency matrix of a finite dag are all zero (= the matrix is **nilpotent**).

The reciprocal of this proposition is also true.

Example of topological sort

ELU 501

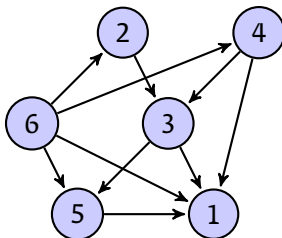
Data science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

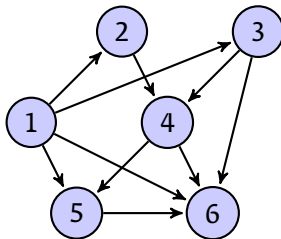
Vertices, edges
 Adjacency matrix
 Weighted graph
 Co-citation and
 bibliographic
 coupling
 Orientation,
 acyclicity,
 topological sort
 Discretion,
 completeness
 Cliques and
 stables
 Bipartite graphs
 Incidence matrix
 Trees, forests
 Returning to
 graphs: average
 degree, density,
 large networks
 Paths
 Graph Laplacian

Before sorting:



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

After sorting:



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

Link between cycles and eigenvalues

ELU 501

Data science,
graph theory
and social
network
studies

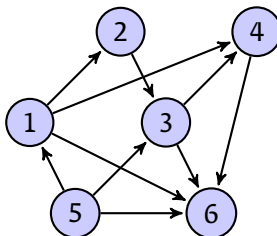
Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

Example:

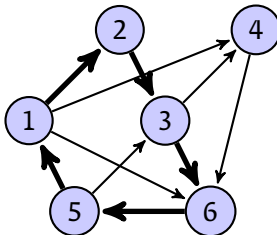
Without cycle:



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

$$vp = 0, 0, 0, 0, 0, 0$$

With cycle:

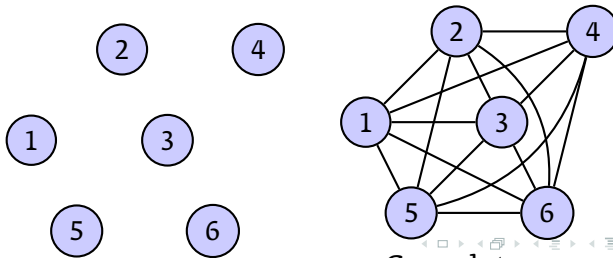


$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$vp = 1.5, \\ -0.59 \pm 1.07i, \\ 0.18 \pm 0.78i, -0.7$$

Discretion, regularity, completeness

- Let $| \cdot |$ denote the *order* $\langle \text{ordre} \rangle$ (nb of vertices) et $\| \cdot \|$ the *size* $\langle \text{taille} \rangle$ (nb of edges).
- A graph G is *discrete* $\langle \text{graphe discret} \rangle$ if $\|G\|=0$.
- The *degree* $\langle \text{degré} \rangle$ of a vertex is the number of adjacent edges.
- If all vertices have the same degree k , we say that G is *k -regular* $\langle \text{graphe } k\text{-régulier} \rangle$.
- G is *complete* $\langle \text{graphe complet} \rangle$ if it contains all possible edges, that is if $\|G\| = \frac{1}{2}|G|(|G| - 1)$.
- We denote by K_n the complete graph of order n , it is $(n - 1)$ -



Cliques and stables

ELU 501

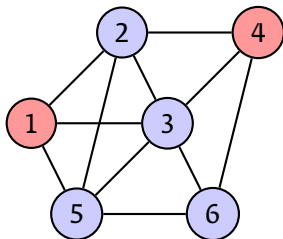
Data science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

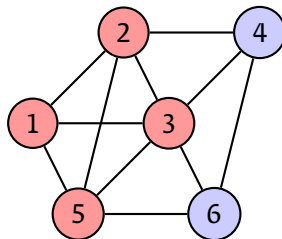
Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

- A *clique* (🇫🇷 *clique*) is a set of vertices inducing a complete subgraph (maximum amount of edges).
- A *stable* (🇫🇷 *stable*) is a set of vertices inducing a discrete subgraph (no edge).



Stable (in red)



Clique (in red)

Bipartite graphs

ELU 501

Data science,
graph theory
and social
network
studies

Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix

Weighted graph
Co-citation and
bibliographic
coupling

Orientation,
acyclicity,
topological sort

Discretion,
completeness
Cliques and
stables

Bipartite graphs
Incidence matrix

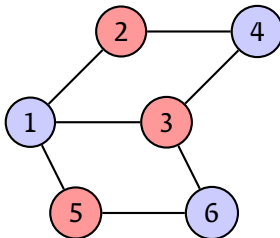
Trees, forests

Returning to
graphs: average
degree, density,
large networks

Paths

Graph Laplacian

- G is *bipartite* (français *graphe biparti*) if $V = V_1 \cup V_2$ with V_1, V_2 being stables.



Proposition

A graph is bipartite if it has no cycle of odd length.


ELU 501

Data science,
graph theory
and social
network
studies

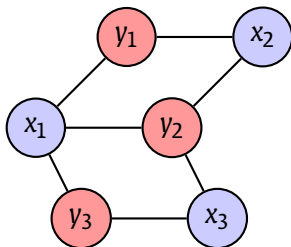
Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

- Bipartite graphs can be described by a more economical data structure called *incidence matrix*  *matrice d'incidence*: if $X = \{x_i\}$ and $Y = \{y_j\}$ are the two stables of G , we define

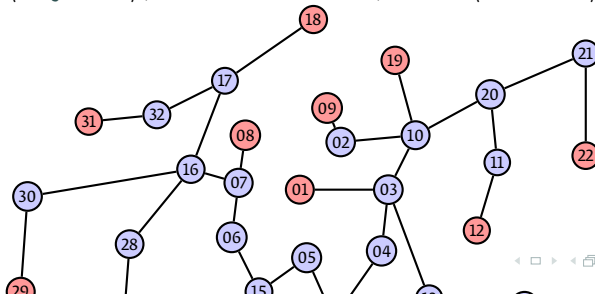
$$B = \{b_{i,j} \mid b_{i,j} = 1 \text{ if } x_i y_j \in G \text{ and } 0 \text{ otherwise}\}.$$



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

Trees: definitions, characterizations

- A graph is **connected** (français *graphe connexe*) if any pair of vertices can be joined by a **path** (français *chemin*) (= a sequence of contiguous edges).
- Connected maximal subgraphs of G are called **connected components** (français *composante connexe*).
- A **trail** (français *chemin simple*) is a path in which no edge appears twice.
- A **tree** (français *arbre*) is a connected acyclic graph.
- External vertices (those of degree 1) are called **leaves** (français *feuille*), and the others, **nodes** (français *nœud*).



ELU 501

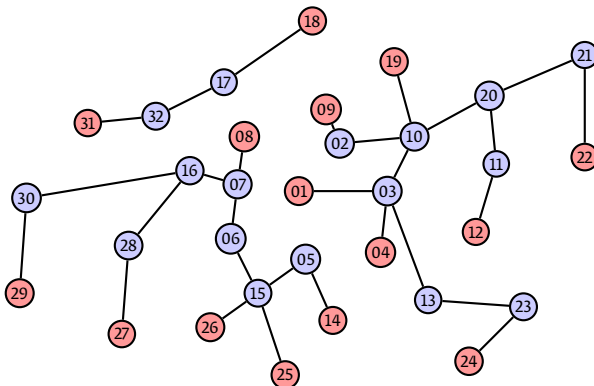
Data science,
graph theory and social
network studies

Yannis Haralambous
(IMT Atlantique)

Basic graph notions

Vertices, edges
 Adjacency matrix
 Weighted graph
 Co-citation and bibliographic coupling
 Orientation, acyclicity, topological sort
 Discretion, completeness
 Cliques and stables
 Bipartite graphs
 Incidence matrix
 Trees, forests
 Returning to graphs: average degree, density, large networks
 Paths
 Graph Laplacian

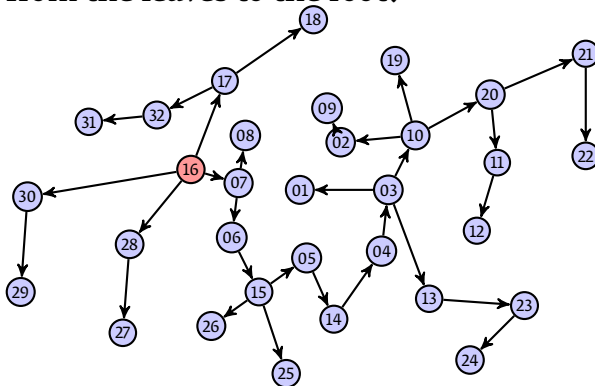
- A *forest* (🇫🇷 *forêt*) is a graph every connected component of which is a tree.



(Leaves in red)

Root, orientation of tree

- A priori, a tree is not directed.
- By choosing an arbitrary vertex (which we call *root of the tree* (🇫🇷 *racine de l'arbre*)), we get a unique orientation with edges going either from the root to the leaves or from the leaves to the root.



(Here node 16 has been chosen as root)

Order and size of a tree

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix

Weighted graph

Co-citation and
bibliographic
coupling

Orientation,
acyclicity,
topological sort

Discretion,
completeness

Cliques and
stables

Bipartite graphs

Incidence matrix

Trees, forests

Returning to
graphs: average
degree, density,
large networks

Paths

Graph Laplacian

- In a tree we always have $\|T\| = |T| - 1$, and this the smallest size maintaining connectivity.
- It is also the largest size maintaining acyclicity (add an edge between existing vertices and you automatically get a cycle).

Proposition

Every finite tree of order ≥ 2 has at least two leaves.

Average degree, density

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

- Let us return to graphs.
- We saw the definition of the degree $d(v_i)$ previously. if $A = (a_{i,j})$ is the adjacency matrix, we have $d(v_i) = \sum_{j=1}^n a_{i,j}$.
- Important result: *the sum of degrees of all vertices of a graph equals twice its size* ($\sum d(v_i) = 2\|G\|$).
- Therefore the average degree c of a graph is equal to $\frac{2\|G\|}{|G|}$.
- If $|G| = n$, the maximum size of the graph is $\|K_n\| = \frac{n(n-1)}{2}$.
- The *density* (densité) ρ of a graph is its size divided by maximum size:

$$\rho = \frac{\|G\|}{\frac{n(n-1)}{2}} = \frac{c}{n-1}.$$

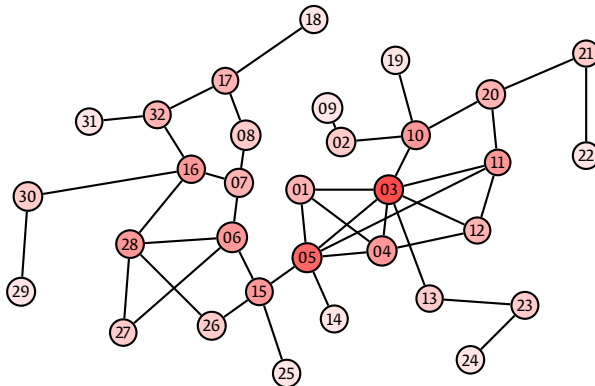
Example

ELU 501
Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)




Basic graph
notions

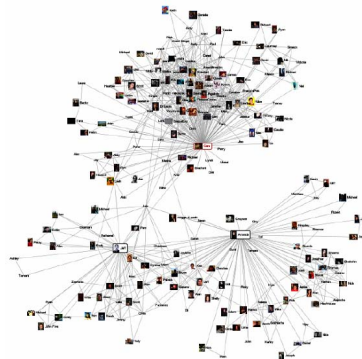
Vertices, edges
 Adjacency matrix
 Weighted graph
 Co-citation and
 bibliographic
 coupling
 Orientation,
 acyclicity,
 topological sort
 Discretion,
 completeness
 Cliques and
 stables
 Bipartite graphs
 Incidence matrix
 Trees, forests
 Returning to
 graphs: average
 degree, density,
 large networks
 Paths
 Graph Laplacian



- Intensity of red color represents degree.
- For this graph G , $|G| = 32$, $\|G\| = 42$, $\min \text{degree} = 1$, $\max \text{degree} = 7$, average degree $c = 2.625$, density $\rho = 0.0847$.

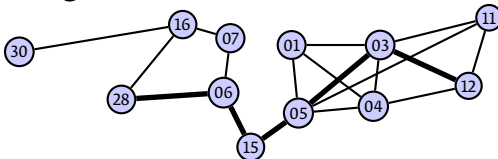
Large networks, density

- We will see later that some graphs (like the Web) can be arbitrarily large, we call them *large networks*  *grand réseau*.
- A large network G is *dense*  *graphe dense* when $\lim_{|G| \rightarrow \infty} \rho > 0$, otherwise it is *sparse*  *graphe épars*.

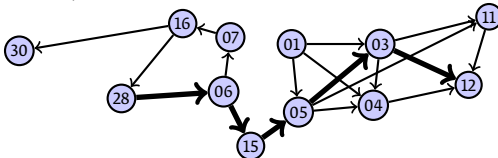


- Example: a social network is sparse when the number of friends does not increase necessarily proportionally

- Recall that a *path* (French: *chemin*) is a sequence of consecutive edges.



- In a directed graph, edges of a path must point in the same direction.

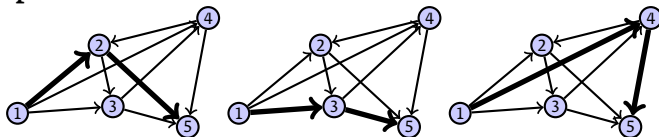


- In the absence of a weight, the length of a path will be counted by the number of edges (counted more than once when we loop through the same edges). In the example, $|x_{28}x_6x_{15}x_5x_3x_{12}| = 5$.

Calculation of the number of paths of length 2

- Les $A = (a_{i,j})$ be the adjacency matrix. Note that we have a path $v_i v_j v_k$ (of length 2) iff $a_{i,j} a_{j,k} = 1$.
- Therefore the number of paths of length 2 between v_i and v_k is $\sum_{j=1}^n a_{i,j} a_{j,k}$.
- I.e., the matrix $N^{(2)}$ of the numbers of paths of length 2 between v_i and v_j is $A \cdot A$.

Example:



$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad A^2 = \begin{pmatrix} 0 & 1 & 1 & 1 & 3 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Calculation of the number of paths of a given length

ELU 501

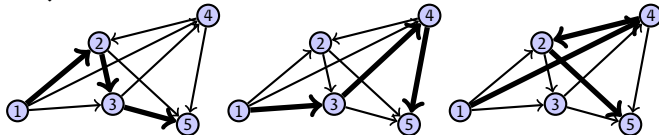
Data science,
graph theory
and social
network
studies

Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

- In the same way we get $N^{(3)} = A^3$ and, more generally, $N^{(r)} = A^r$.



$$A^3 = \begin{pmatrix} 0 & 1 & 1 & 1 & 3 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad A^4 = \begin{pmatrix} 0 & 1 & 1 & 1 & 3 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad \dots$$

Calculation of the number of cycles of a given length

ELU 501

Data science,
graph theory
and social
network
studies

Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges

Adjacency matrix

Weighted graph

Co-citation and
bibliographic
coupling

Orientation,
acyclicity,
topological sort

Discretion,
completeness

Cliques and
stables

Bipartite graphs

Incidence matrix

Trees, forests

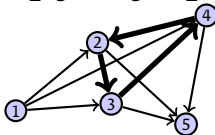
Returning to
graphs: average
degree, density,
large networks

Paths

Graph Laplacian

- In particular, there are $(A^r)_{i,i}$ cycles of length r going through vertex v_i .

Here, $(A^3)_{2,2} = (A^3)_{3,3} = (A^3)_{4,4} = 1$, and hence the only cycles of length 3 are $x_2x_3x_4$, $x_3x_4x_2$, $x_4x_2x_3$:



Calculation of the number of cycles of a given length

ELU 501

Data science,
graph theory
and social
network
studies

Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling

Orientation,
acyclicity,
topological sort

Discretion,
completeness
Cliques and
stables

Bipartite graphs
Incidence matrix

Trees, forests
Returning to
graphs: average
degree, density,
large networks

Paths

Graph Laplacian


- There are $(A^r)_{i,i}$ cycles of length r going through vertex v_i , therefore the total number L_r of cycles of length r is $\sum_{i=1}^n (A^r)_{i,i} = \text{Trace}(A^r)$.
- In the case of the example, we have a periodicity of 3:

$$A^{3k+1} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, A^{3k+2} = \begin{pmatrix} 0 & 1 & 1 & 1 & 3 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, A^{3k} = \begin{pmatrix} 0 & 1 & 1 & 1 & 3 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

for $k \geq 0$.

- Therefore: $\text{Trace}(A^{3k+1}) = \text{Trace}(A^{3k+2}) = 0$,
 $\text{Trace}(A^{3k}) = 3$.

Zero eigenvalues \Rightarrow acyclicity

- Recall from linear algebra ([Schur decomposition](#) ⟨ *décomposition de Schur*⟩): every square matrix A can be written $A = U \cdot T \cdot {}^tU$, where U is a unitary matrix and T a upper triangular matrix the diagonal of which contains the eigenvalues of A . If moreover A is Hermitian ($A = {}^t\overline{A}$), then the eigenvalues are real.
- Hence,

$$A^r = (U \cdot T \cdot {}^tU)^r = U \cdot T^r \cdot {}^tU$$

and

$$\text{Trace}(U \cdot T^r \cdot {}^tU) = \text{Trace}(U \cdot {}^tU \cdot T^r) = \text{Trace}(T^r) = \sum_i \kappa_i^r.$$

where κ_i^r are r th powers of eigenvalues of A .

- We can now show the reciprocal of the proposition linking eigenvalues and acyclicity:

Proposition

A graph with zero adjacency matrix eigenvalues is acyclic.

Geodesic paths, Eulerians, Hamiltonians

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

- A *geodesic path* $\langle \text{chemin géodésique} \rangle$ between v_i and v_j is a path between these vertices having a minimal amount of edges.
- The *geodesic distance* $\langle \text{distance géodésique} \rangle$ between two vertices is the length of an arbitrary geodesic path between them (or ∞ if the vertices belong to distinct connected components).
- A *Eulerian path* $\langle \text{chemin eulerien} \rangle$ is a path going through all vertices of the graph, in which every edge appears once and only once. A graph is called Eulerian if it admits a Eulerian path.

Theorem (Euler (1736))

A connected graph is Eulerian iff all vertices are of even degree.

- A path is called *Hamiltonian* $\langle \text{chemin hamiltonien} \rangle$ if it passes once and only once through each vertex.

Connected components

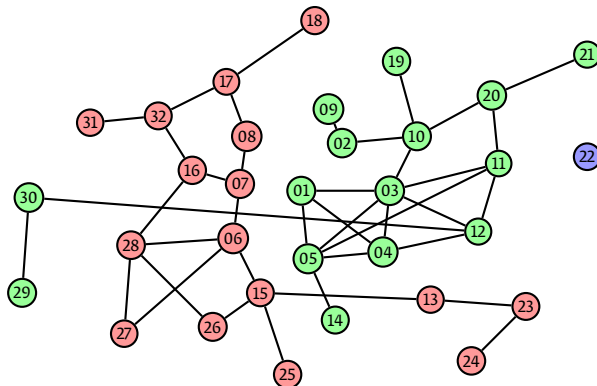
ELU 501

Data science,
graph theory and social
network studies

Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

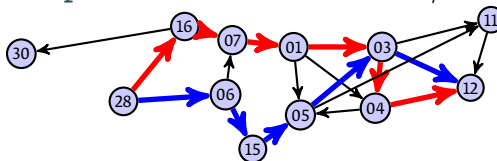
Vertices, edges
 Adjacency matrix
 Weighted graph
 Co-citation and
 bibliographic
 coupling
 Orientation,
 acyclicity,
 topological sort
 Discretion,
 completeness
 Cliques and
 stables
 Bipartite graphs
 Incidence matrix
 Trees, forests
 Returning to
 graphs: average
 degree, density,
 large networks
 Paths
 Graph Laplacian



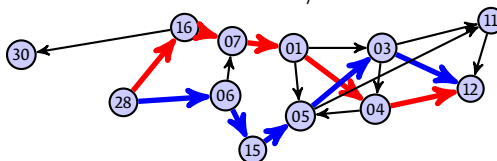
- Recall that a **connected component** (composante connexe) of a graph is a maximal connected subgraph.
- Modulo a column permutation, the adjacency matrix of an unconnected graph is a **block matrix**, having as many blocks as connected components.

Independent paths

- Between two vertices we can have many paths. Those having no common edge are called *edge-wise independent* (🇫🇷 *chemins indépendants vis-à-vis des arêtes*).



- Those having no common vertex (except for the extremities) are called *vertex-wise independent* (🇫🇷 *chemins indépendants vis-à-vis des sommets*).



- The number of (edge-wise or vertex-wise) independent paths between v_i and v_j is called (edge-wise or vertex-wise) *connectivity* (🇫🇷 *connectivité*).

Minimal cuts

ELU 501

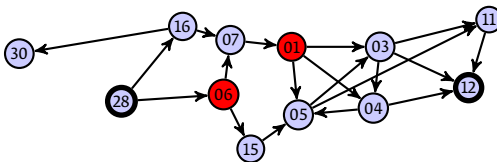
Data science,
graph theory and social
network studies

Yannis Haralambous
(IMT Atlantique)

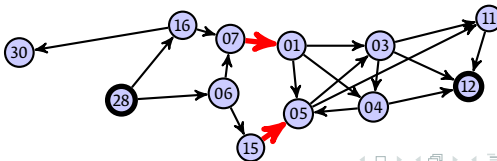
Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

- A **vertex-wise minimal cut** (coupe minimale en sommets) between v_i and v_j is a minimal set of vertices such that when removed, v_i and v_j belong to distinct connected components.



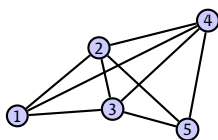
- An **edge-wise minimal cut** (coupe minimale en arêtes) between v_i and v_j is a minimal set of edges such that when removed, v_i and v_j belong to distinct connected components.



Graph Laplacian

- Until now we have studied graphs via their adjacency matrix. There is another useful algebraic structure, called *graph Laplacian* $\langle \text{laplacien de graphe} \rangle$.
- We call $L := D - A$ the *graph Laplacian* $\langle \text{laplacien du graphe} \rangle$ of G where A is its adjacency matrix and D the diagonal matrix containing the degrees of its vertices.

Example:

$G =$  $A = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}, \quad D = \begin{pmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{pmatrix}$

and therefore $L = D - A = \begin{pmatrix} 3 & -1 & -1 & -1 & 0 \\ -1 & 4 & -1 & -1 & -1 \\ -1 & -1 & 4 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ 0 & -1 & -1 & -1 & 3 \end{pmatrix}.$

Basic graph notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and bibliographic coupling
Orientation, acyclicity, topological sort
Discretion, completeness
Cliques and stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to graphs: average degree, density, large networks
Paths
Graph Laplacian

Number of connected components

- Consider a graph with c connected components. Modulo a vertex permutation, the Laplacian can be written as a block matrix, with c blocks.
- Each block is the Laplacian of a component.

Lemma

The Laplacian of a connected graph has at least one zero eigenvalue.

Proof.

Let $d(v_j)$ the degree of v_j , $A = (a_{i,j})$ the adjacency matrix, $L = (\ell_{i,j})$ the Laplacian, and δ_{ij} the Dirac symbol. We have $d(v_j) = \sum_i a_{i,j}$ and $\ell_{i,j} = \delta_{ij}d(v_j) - a_{i,j}$. Let us prove that $\mathbf{1} = {}^t(1, \dots, 1)$ is eigenvector of L : the j th row of $L \cdot \mathbf{1}$ is $\sum_i (\delta_{ij}d(v_j) - a_{i,j}) = 0$. This shows that 0 is always eigenvalue of a Laplacian. □

Number of connected components

- In fact we have a stronger result:

Proposition

A graph has the same number of connected components as its Laplacian has zero eigenvalues.

- In particular, when the second eigenvalue (in increasing order) is nonzero then the graph is connected. We call its value the *algebraic connectivity* (🇫🇷 *connexité algébrique*) of the graph.

Number of connected components

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges

Adjacency matrix

Weighted graph

Co-citation and
bibliographic
coupling

Orientation,
acyclicity,
topological sort

Discretion,
completeness

Cliques and
stables

Bipartite graphs

Incidence matrix

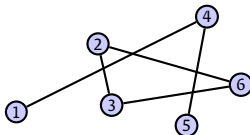
Trees, forests

Returning to
graphs: average
degree, density,
large networks

Paths

Graph Laplacian

Example: $G =$



$$A = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}, \quad D = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

and hence $L = A - D =$

$$\begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 2 & -1 & 0 & 0 & -1 \\ 0 & -1 & 2 & 0 & 0 & -1 \\ -1 & 0 & 0 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & -1 & -1 & 0 & 0 & 2 \end{pmatrix}.$$

the eigenvalues of which are 0, 0, 1, 3, 3, 3. Two zero eigenvalues:
two connected components.

Graph representation

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix

Weighted graph

Co-citation and
bibliographic
coupling

Orientation,
acyclicity,
topological sort

Discretion,
completeness

Cliques and
stables

Bipartite graphs

Incidence matrix

Trees, forests

Returning to
graphs: average
degree, density,
large networks

Paths

Graph Laplacian

- Until now we have represented graphs by their *adjacency matrix* A .
- The corresponding data structure would use n^2 slots (or $n^2/2$ for an undirected graph). That's a lot, especially when the graph is sparse.
- Another possibility: the *adjacency list* $\langle \text{liste d'adjacence} \rangle$. We link each vertex v_i with the linked list of its neighbors $v_{i,j}$: $v_i \rightarrow v_{i,1} \rightarrow v_{i,2} \rightarrow \dots$
- Third possibility: the *adjacency forest* $\langle \text{forêt d'adjacence} \rangle$. With link each vertex v_i with the binary tree of its neighbors (recall that the condition for having a binary tree is that each parent has at most two children and that the label of the left child is smaller than the label of the parent which in turn is smaller than the label of the right child).

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix

Weighted graph
Co-citation and
bibliographic
coupling

Orientation,
acyclicity,
topological sort

Discretion,
completeness

Cliques and
stables

Bipartite graphs

Incidence matrix

Trees, forests

Returning to
graphs: average
degree, density,
large networks

Paths

Graph Laplacian

- Table of comparison:

Operation	Matrix	List	Forest
Insertion	$O(1)$	$O(1)$	$O(\log(m/n))$
Suppression	$O(1)$	$O(m/n)$	$O(\log(m/n))$
Search	$O(1)$	$O(m/n)$	$O(\log(m/n))$
Enumeration	$O(n)$	$O(m/n)$	$O(m/n)$

Example of adjacency list

ELU 501

Data

science,

graph theory

and social

network

studies

Yannis Ha-

ralambous

(IMT

Atlantique)

Basic graph
notions

Vertices, edges

Adjacency matrix

Weighted graph

Co-citation and

bibliographic

coupling

Orientation,

acyclicity,

topological sort

Discretion,

completeness

Cliques and

stables

Bipartite graphs

Incidence matrix

Trees, forests

Returning to

graphs: average

degree, density,

large networks

Paths

Graph Laplacian

- 1: $2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$
- 2: $9 \rightarrow 10 \rightarrow 3 \rightarrow 1$
- 3: $1 \rightarrow 2 \rightarrow 10 \rightarrow 11 \rightarrow 12 \rightarrow 4 \rightarrow 5$
- 4: $5 \rightarrow 1 \rightarrow 3 \rightarrow 12$
- 5: $1 \rightarrow 4 \rightarrow 14 \rightarrow 15 \rightarrow 7$
- 6: $7 \rightarrow 1 \rightarrow 15 \rightarrow 28$
- 7: $8 \rightarrow 5 \rightarrow 6 \rightarrow 16$
- 8: $17 \rightarrow 7 \rightarrow 27 \rightarrow 16$
- 9: 2
- 10: $19 \rightarrow 20 \rightarrow 3 \rightarrow 2$
- 11: $20 \rightarrow 12 \rightarrow 5 \rightarrow 3$
- 12: $11 \rightarrow 3 \rightarrow 4$
- 13: $3 \rightarrow 23$
- 14: 5
- 15: $6 \rightarrow 5 \rightarrow 25 \rightarrow 26$
- 16: $32 \rightarrow 8 \rightarrow 7 \rightarrow 28 \rightarrow 30$
- 17: $18 \rightarrow 8 \rightarrow 32$
- 18: 17
- 19: 10
- 20: $10 \rightarrow 11 \rightarrow 21$
- 21: $20 \rightarrow 22$
- 22: 21
- 23: $13 \rightarrow 24$
- 24: 23
- 25: 15
- 26: $15 \rightarrow 28$
- 27: $28 \rightarrow 8 \rightarrow 6$
- 28: $16 \rightarrow 6 \rightarrow 26 \rightarrow 27$
- 29: 30
- 30: $29 \rightarrow 16$
- 31: 32
- 32: $31 \rightarrow 16 \rightarrow 17$

Example of adjacency forest

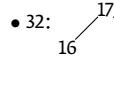
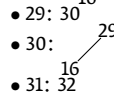
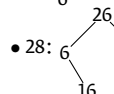
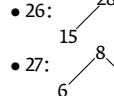
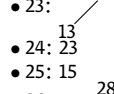
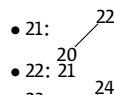
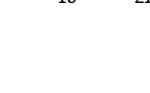
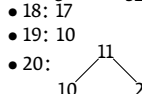
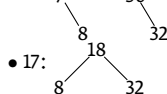
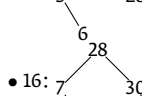
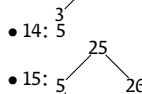
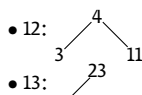
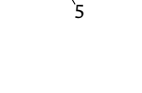
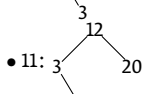
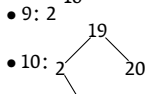
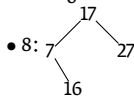
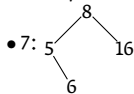
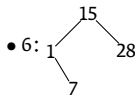
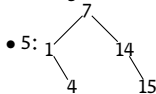
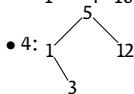
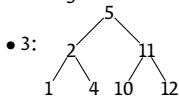
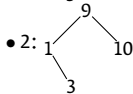
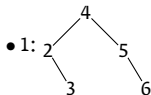
ELU 501

Data science,
graph theory and social
network studies

Yannis Haralambous
(IMT Atlantique)

Basic graph notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and bibliographic coupling
Orientation, acyclicity, topological sort
Discretion, completeness
Cliques and stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to graphs: average degree, density, large networks
Paths
Graph Laplacian



Traversing graph vertices

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

- We sometimes need to traverse all vertices of a graph.
- Here are two symmetric approaches:
- by a *depth-first search* DFS *(parcours en profondeur d'abord)*: we go as far as possible and when we reach an extremity we turn back and investigate all branches we left out by starting by the closest ones;
- or by a *breadth-first search* BFS *(parcours en largeur d'abord)*: we proceed by levels considering first vertices at geodesic distance n and those of geodesic distance $n + 1$ and so on.
- The complexity of the two approaches is the same.
- Peculiarities: DFS uses a recursive procedure. BFS will traverse vertices in order of geodesic distance from the starting point.

Depth-first search (DFS)

Let G be a graph. A *DFS algorithm*  *parcours en profondeur d'abord* is obtained as follows:

DFS(G):

```
for each vertex  $u \in G$ :
    do COLOR( $u$ )  $\leftarrow$  WHITE
for each vertex  $u \in G$ :
    do if COLOR( $u$ ) = WHITE:
        then DFS-VISIT( $u$ )
```

DFS-VISIT(u):

```
COLOR( $u$ )  $\leftarrow$  GREY
for each vertex  $v \in \text{NEIGHBORS}(u)$ :
    do if COLOR( $v$ ) = WHITE:
        then DFS-VISIT( $v$ )
COLOR( $u$ )  $\leftarrow$  BLACK
```

Complexity: $O(V + E)$.

DFS example

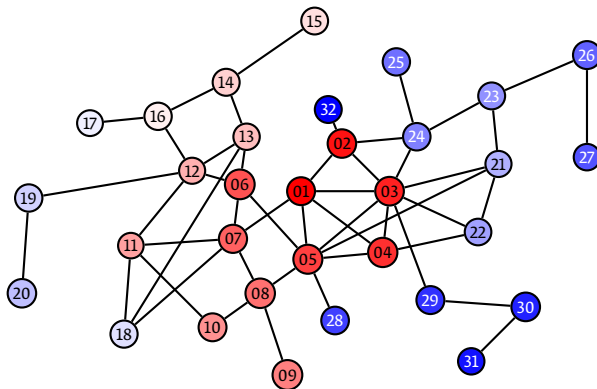
ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)


Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian



We have renumbered the vertices in DFS order starting from vertex 1. Colors go gradually from red to blue. Note how close vertex 32 is from vertex 1.

Breadth-first search (BFS)

Let G be a *connected*. A *breadth-first search* ( *parcours en largeur d'abord*) starting from vertex s is obtained in the following way:

BFS(G, s):

```

for each vertex  $u \in G \setminus \{s\}$ 
  do COLOR( $u$ )  $\leftarrow$  WHITE
       $d[u] \leftarrow \infty$ ,  $\pi[u] \leftarrow \text{NIL}$ 
COLOR[ $s$ ]  $\leftarrow$  GREY,  $d[s] \leftarrow 0$ ,  $\pi[s] \leftarrow \text{NIL}$ ,  $Q \leftarrow \{s\}$ 
while  $Q \neq \emptyset$ :
  do  $u \leftarrow \text{SHIFT}([Q])$ 
      for each  $v \in \text{NEIGHBORS}(u)$ :
        do if COLOR[ $v$ ] = WHITE:
          then COLOR[ $v$ ]  $\leftarrow$  GREY
               $d[v] \leftarrow d[u] + 1$ ,  $\pi[v] \leftarrow u$ , SHIFT( $Q, v$ )
COLOR( $u$ )  $\leftarrow$  BLACK
  
```

Complexity: $O(V + E)$.

BFS example

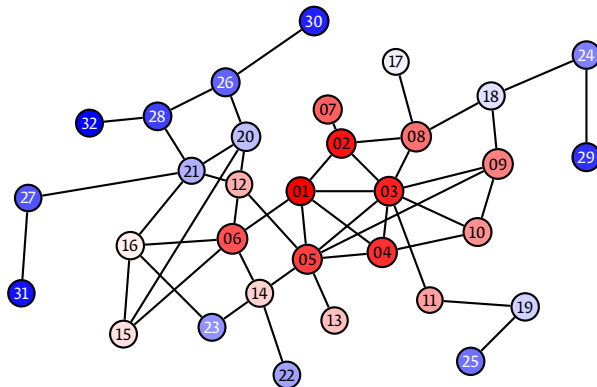
ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian



We have renumbered vertices in the order of BFS starting from vertex 1. Colors go gradually from red to blue. This time 1 is very far from 32.

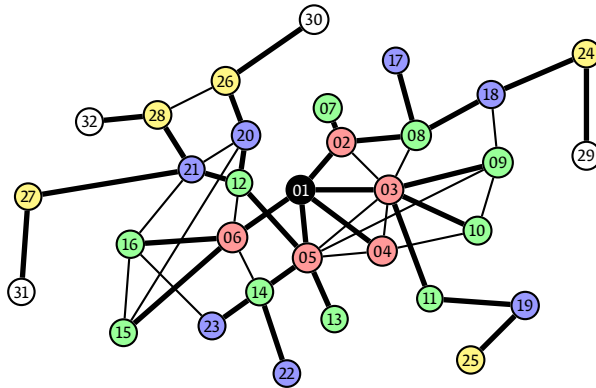
BFS example

ELU 501
Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
 Adjacency matrix
 Weighted graph
 Co-citation and
 bibliographic
 coupling
 Orientation,
 acyclicity,
 topological sort
 Discretion,
 completeness
 Cliques and
 stables
 Bipartite graphs
 Incidence matrix
 Trees, forests
 Returning to
 graphs: average
 degree, density,
 large networks
 Paths
 Graph Laplacian



Colors red, green, blue, yellow, white correspond to five distance levels. In thick stroke: a geodesic spanning tree.

Traversing graph edges

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

- In what follows we will modify BFS to traverse not only vertices but also edges of a connected graph.
- At first we use the predecessor of each vertex to obtain the corresponding edge, this provides us with a *spanning tree* of the graph.
- Afterwards we need to traverse edges which form cycles (called *cross edges*).
- For this we adopt the following strategy: after having traversed the tree edges which are adjacent to a vertex, we add those which are do not belong to the tree but *both extremities of which have already been traversed*.
- The advantage of this approach is that the graph obtained is connected at all times. We call this algorithm BFSE (BFS of edges).

Traversing graph edges

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

BFSE(G, s):

for each vertex $u \in G \setminus \{s\}$

do $\text{COLOR}(u) \leftarrow \text{WHITE}$

$d[u] \leftarrow \infty, \pi[u] \leftarrow \text{NIL}$

$\text{COLOR}[s] \leftarrow \text{GREY}, d[s] \leftarrow 0, \pi[s] \leftarrow \text{NIL}, Q \leftarrow \{s\}, A \leftarrow \{\}$

for each edge $(u, v) \in G$

do $\text{COLOR}((u, v)) \leftarrow \text{WHITE}$

while $Q \neq \emptyset$:

do $u \leftarrow \text{SHIFT}(Q)$

for each $v \in \text{NEIGHBORS}(u)$:

do if $\text{COLOR}[v] = \text{WHITE}$:

then $\text{COLOR}[v] \leftarrow \text{GREY}, d[v] \leftarrow d[u] + 1,$

$\pi[v] \leftarrow u, \text{UNSHIFT}(Q, v), \text{UNSHIFT}(A, (u, v))$

do if $\text{COLOR}[(u, v)] = \text{WHITE}$ **and** $\text{COLOR}[v] = \text{GREY}$:

then $\text{COLOR}[(u, v)] \leftarrow \text{GREY}, \text{UNSHIFT}(A, (u, v))$

$\text{COLOR}(u) \leftarrow \text{BLACK}$

Shortest path search

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

- We will limit ourselves to directed graphs.
- BFS has allowed us to find geodesic paths. But in many situations, edges are provided with a “weight” (for example, a geographic distance), and we search for the shortest path with respect to that weight.
- Weight can be negative, but you should better not have cycles of total negative weight.
- And in any case, a shortest path can not contain a cycle of strictly positive weight.
- We will study three shortest path search algorithms: Bellman-Ford, Dijkstra and A^* . In all three cases we will use a general method, called “relaxation”.

ELU 501

Data

science,

graph theory

and social

network

studies

Yannis Ha-

ralambous

(IMT

Atlantique)

Basic graph
notions

Vertices, edges

Adjacency matrix

Weighted graph

Co-citation and

bibliographic

coupling

Orientation,

acyclicity,

topological sort

Discretion,

completeness

Cliques and

stables

Bipartite graphs

Incidence matrix

Trees, forests

Returning to

graphs: average

degree, density,

large networks

Paths

Graph Laplacian

- We initialize the graph as follows:

INITIALIZE-SINGLE-SOURCE_d(G, s):

for chaque $v \in V$:

$d(v) \leftarrow \infty, \pi(v) \leftarrow \emptyset$

$d(s) \leftarrow 0$

- For each vertex u of the graph, we consider a positive quantity $d(u)$ which will end up being the distance of u from the “origin” s . To store the shortest path information we use the notion of *preceding vertex* $\langle \text{sommet précédent} \rangle \pi(v)$.

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling

Orientation,
acyclicity,
topological sort

Discretion,
completeness
Cliques and
stables


Bipartite graphs
Incidence matrix

Trees, forests

Returning to
graphs: average
degree, density,
large networks

Paths

Graph Laplacian

- **Relaxation** ( *relaxation*) of an edge (u, v) consist in changing $d(v)$ if we can pass through u to arrive at v for less weight cost:

$\text{RELAX}_d(u, v, w)$:

if $d(v) > d(u) + w(u, v)$:

$d(v) \leftarrow d(u) + w(u, v)$

$\pi(v) \leftarrow u$

where $w(u, v)$ is the weight of edge (u, v) .

- All three algorithms *initialize* and *relax*. What changes is the order and the quantity of relaxations.

Bellman-Ford

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling

Orientation,
acyclicity,
topological sort

Discretion,
completeness
Cliques and
stables

Bipartite graphs
Incidence matrix
Trees, forests

Returning to
graphs: average
degree, density,
large networks

Paths

Graph Laplacian

- *Bellman-Ford algorithm* (algorithm de Bellman-Ford) finds shortest paths for both positive and negative weights.

BELLMAN-FORD(G, w, s):

INITIALIZE-SINGLE-SOURCE _{d} (G, s)

for $i \leftarrow 1$ **à** $|V| - 1$:

for each edge (u, v) :

RELAX _{d} (u, v, w)

for each edge (u, v) :

if $d(v) > d(u) + w(u, v)$:

return FALSE

return TRUE

Note that i appears in the loop but not in the rest of the code...

- This algorithm is of complexity $O(VE)$: when we have many edges, i.e., $O(n^2)$, then the algorithm ends up being $O(n^3)$, that's huge!

Min priority queues

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix

Weighted graph
Co-citation and
bibliographic
coupling

Orientation,
acyclicity,
topological sort

Discretion,
completeness

Cliques and
stables

Bipartite graphs

Incidence matrix

Trees, forests

Returning to
graphs: average
degree, density,
large networks

Paths

Graph Laplacian

- To enhance Bellman-Ford, we use a specific data structure: *min priority queues*.
- A *min priority queue* $\langle \text{file de priorité min} \rangle S$ is a queue where elements x are shifted in an order such that the function $d(x)$ takes its minimum value. It uses the following three operations:
 - $\text{INSERT}_d(S, x)$: inserts an element into S .
 - $\text{MINIMUM}_d(S)$: returns the smallest element of S without shifting it.
 - $\text{EXTRACT-MIN}_d(S)$: like MINIMUM but also shifts the element.

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix

Weighted graph
Co-citation and
bibliographic
coupling

Orientation,
acyclicity,
topological sort

Discretion,
completeness

Cliques and
stables

Bipartite graphs
Incidence matrix

Trees, forests

Returning to
graphs: average
degree, density,
large networks

Paths

Graph Laplacian

- Dijkstra is a Dutch name pronounced *Daïk-stra*!
- *Dijkstra's algorithm* $\langle \text{algorithmme de Dijkstra} \rangle$ is much faster than Bellman-Ford, but can be used only for positive weights. It uses a min priority queue Q and a set S .

DIJKSTRA(G, w, s):

INITIALIZE-SINGLE-SOURCE $_d(G, s)$

$S \leftarrow \emptyset, Q \leftarrow \{s\}$

while $Q \neq \emptyset$:

$u \leftarrow \text{EXTRACT-MIN}_d(Q)$

$S \leftarrow S \cup \{u\}$

for each vertex $v \in \text{NEIGHBORS}(u)$:

if $v \notin S$:

$\text{RELAX}_d(u, v, w)$

$Q \leftarrow Q \cup \{v\}$

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

- What is Dijkstra's complexity? Q requires three operations: INSERT (initialization $|V| \cdot O(1)$), EXTRACT-MIN $O(V)$, DECREASE-KEY (in RELAX, $O(1)$). The loop on neighbors is executed exactly once for each edge, it's a $O(E)$. Hence we finally get $O(E + V^2)$.
- By using specific min priority queues called *Fibonacci queues*, we can reach a complexity of $O(E + V \log V)$.

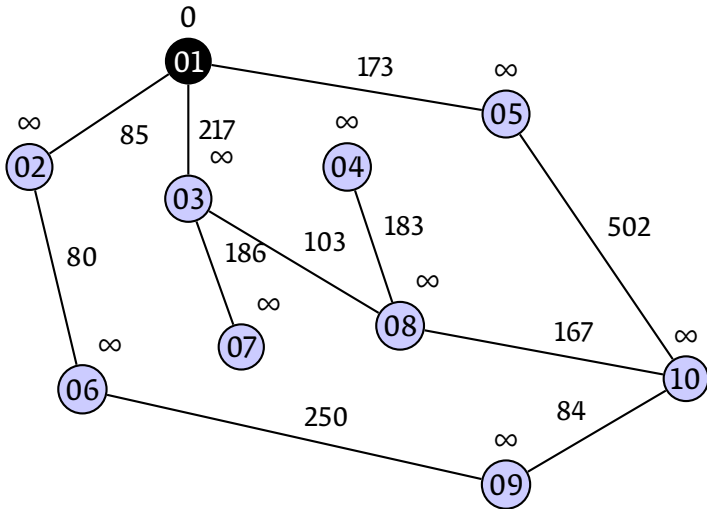
Dijkstra example 1/10

ELU 501
Data science,
graph theory and social
network studies

Yannis Haralambous
(IMT Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian



Dijkstra example 2/10

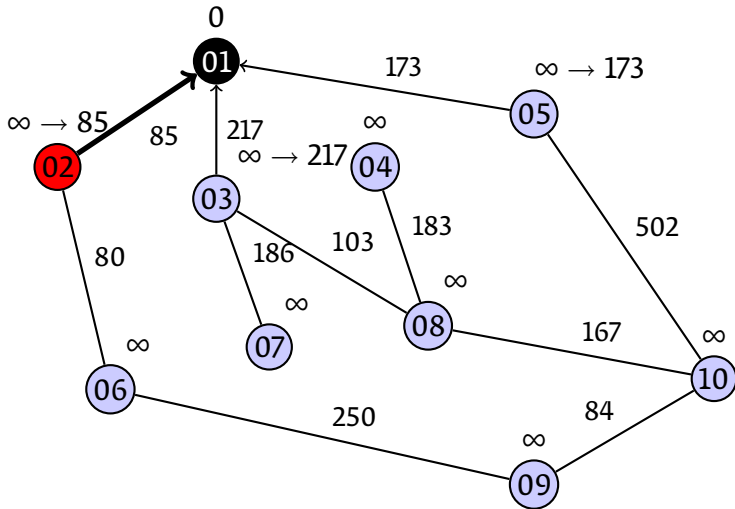
ELU 501

Data science,
graph theory
and social
network
studies

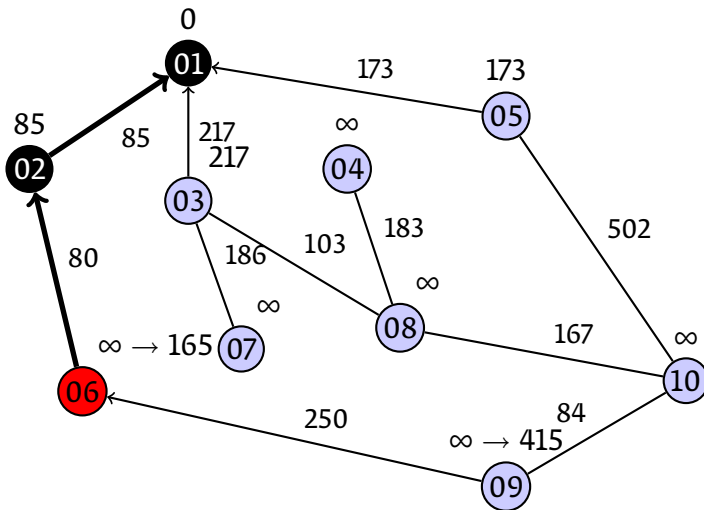
Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian



Dijkstra example 3/10



ELU 501
 Data
 science,
 graph theory
 and social
 network
 studies

Yannis Ha-
 ralambeous
 (IMT
 Atlantique)

Basic graph
notions

Vertices, edges
 Adjacency matrix
 Weighted graph
 Co-citation and
 bibliographic
 coupling
 Orientation,
 acyclicity,
 topological sort
 Discretion,
 completeness
 Cliques and
 stables
 Bipartite graphs
 Incidence matrix
 Trees, forests
 Returning to
 graphs: average
 degree, density,
 large networks
 Paths
 Graph Laplacian

Dijkstra example 4/10

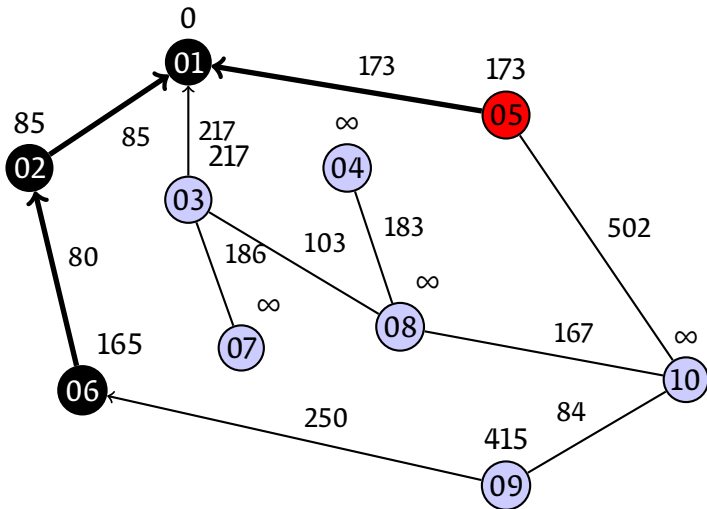
ELU 501

Data science,
graph theory
and social
network
studies

Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian



Dijkstra example 5/10

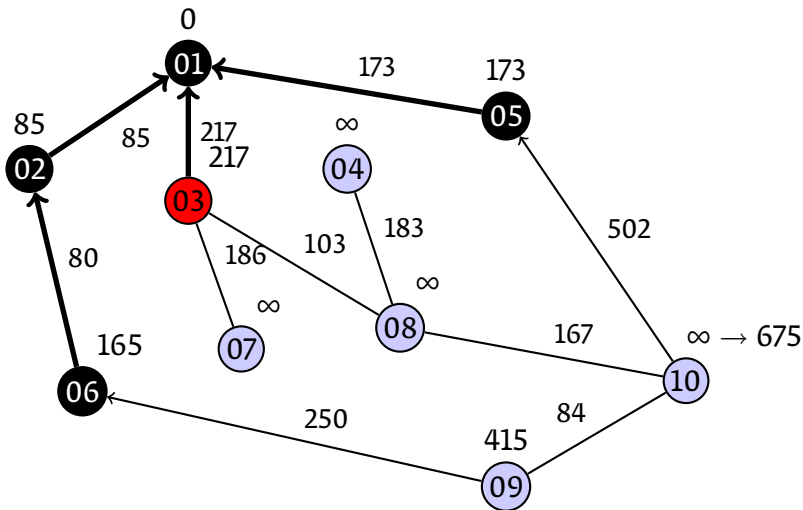
ELU 501

Data science,
graph theory
and social
network
studies

Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian



Dijkstra example 6/10

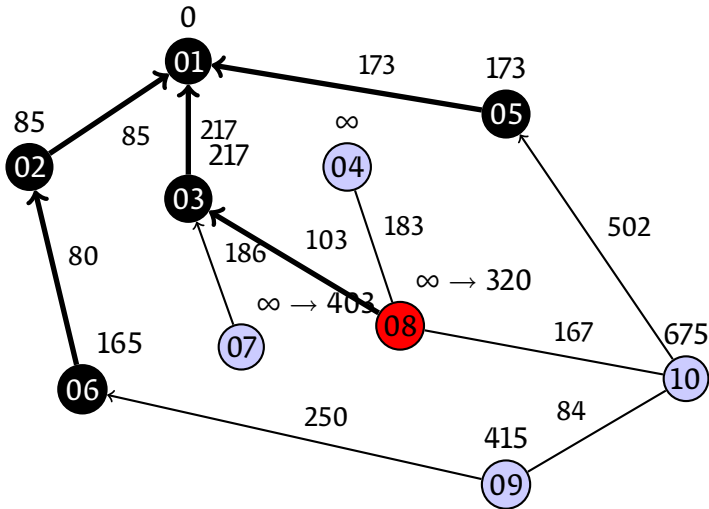
ELU 501

Data science,
graph theory
and social
network
studies

Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian



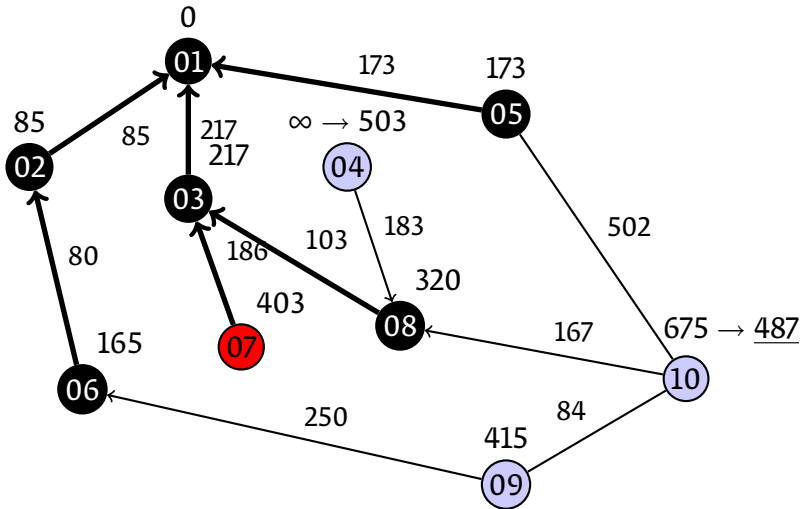
Dijkstra example 7/10

ELU 501
Data science,
graph theory
and social
network
studies

Yannis Haralambous
(IMT Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian



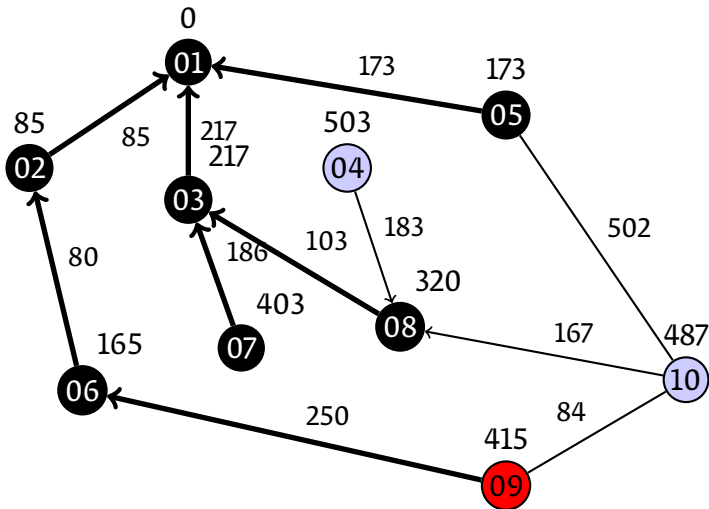
Dijkstra example 8/10

ELU 501
 Data science,
 graph theory
 and social
 network
 studies

Yannis Ha-
 ralambeous
 (IMT
 Atlantique)

Basic graph
notions

Vertices, edges
 Adjacency matrix
 Weighted graph
 Co-citation and
 bibliographic
 coupling
 Orientation,
 acyclicity,
 topological sort
 Discretion,
 completeness
 Cliques and
 stables
 Bipartite graphs
 Incidence matrix
 Trees, forests
 Returning to
 graphs: average
 degree, density,
 large networks
 Paths
 Graph Laplacian

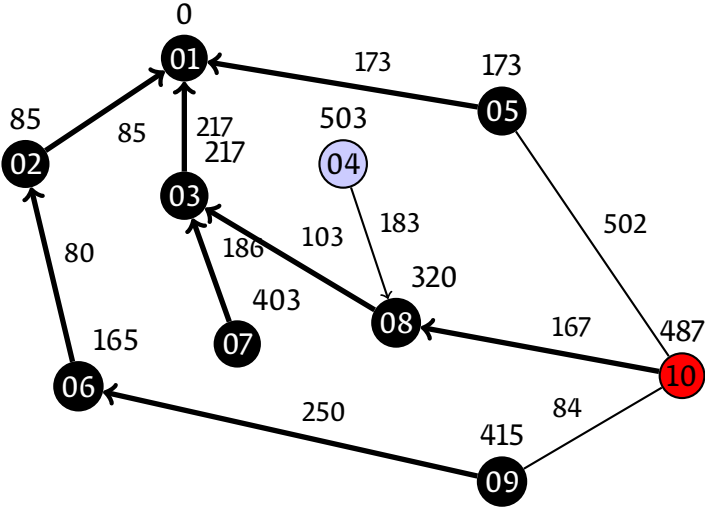


Dijkstra example 9/10

ELU 501
Data science,
graph theory
and social
network
studies

Yannis Haralambous
(IMT Atlantique)

Basic graph
notions
Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

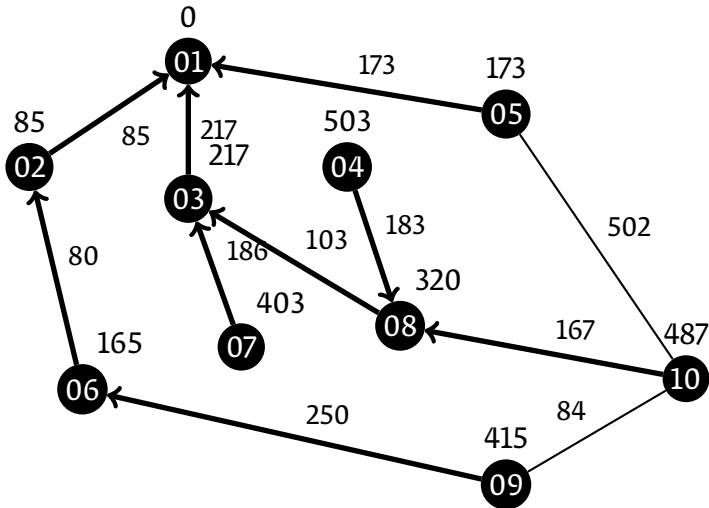


Dijkstra example 10/10

ELU 501
 Data science,
 graph theory
 and social
 network
 studies

Yannis Ha-
 ralambeous
 (IMT
 Atlantique)

Basic graph
 notions
 Vertices, edges
 Adjacency matrix
 Weighted graph
 Co-citation and
 bibliographic
 coupling
 Orientation,
 acyclicity,
 topological sort
 Discretion,
 completeness
 Cliques and
 stables
 Bipartite graphs
 Incidence matrix
 Trees, forests
 Returning to
 graphs: average
 degree, density,
 large networks
 Paths
 Graph Laplacian



Shortest path search with a heuristic


ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

- Bellman-Ford and Dijkstra provide shortest paths between a vertex and *all* other vertices of the graph.
- We may ask: how can we traverse only part of the graph and find relatively rapidly the shortest path between two given vertices?
- Ane *heuristic* ( *heuristique*) (from the Greek verb εὐρίσκειν = to find, out of which we get the famous «(h)eurêka»¹) is any approach to problem solving, learning, or discovery that employs a practical method not guaranteed to be optimal or perfect (WP). In our case will the solution be globally optimal since Dijkstra provides global optimal results, but the heuristic will allow us to find the result faster.

¹Wikipedia: εὐρηκα is the euphoric instant of sudden understanding and certainty after a tense phase...

A^* is born

- The A^* algorithm (algorithm A^*) (pronounced «A-star») calculates the shortest path from vertex s to vertex t .
- It generalizes Dijkstra's algorithm by using a heuristic function h representing the heuristic cost for going from a given vertex to t . This function must be positive and zero only on t .

$A^*(G, w, s, t)$:

INITIALIZE-SINGLE-SOURCE $_d(G, s)$

$S \leftarrow \emptyset, Q \leftarrow \{s\}$

while $Q \neq \emptyset$:

$u \leftarrow \text{EXTRACT-MIN}_{d+h}(Q)$

if $u = t$: **break**

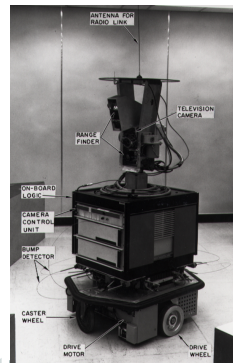
$S \leftarrow S \cup \{u\}$

for each vertex $v \in \text{NEIGHBORS}(u)$:

if $v \notin S$:

$\text{RELAX}_d(u, v, w)$

$Q \leftarrow Q \cup \{v\}$



The SHAKEY robot (1967) (Wikipedia)

Admissible heuristics

ELU 501

Data science,
graph theory
and social
network
studies

Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix

Weighted graph
Co-citation and
bibliographic
coupling

Orientation,
acyclicity,
topological sort

Discretion,
completeness

Cliques and
stables

Bipartite graphs
Incidence matrix

Trees, forests

Returning to
graphs: average
degree, density,
large networks

Paths

Graph Laplacian

- A^* will provide a valid result only if the heuristic h is *admissible* $\langle \text{heuristique admissible} \rangle$ and *monotonic* $\langle \text{heuristique monotone} \rangle$.
- A heuristic h is *admissible* $\langle \text{heuristique admissible} \rangle$ if it doesn't overevaluate distance: i.e., if, by following graph edges, one can reach v from u at a distance $d(u, v)$, then we must necessarily have

$$h(u, v) \leq d(u, v).$$
- A heuristic h is *monotonic* $\langle \text{heuristique monotone} \rangle$ if, for vertices u, u' and v we have always

$$h(u, v) \leq d(u, u') + h(u', v).$$
- Typical example: the distance as the crow flies $\langle \text{distance à vol d'oiseau} \rangle$ (provided there is no wind, raptor, plane, drone, polluting factory, hunter, crow of the opposite sex, etc.).

An example: the 8-puzzle

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

- An **8-puzzle** (3 × 3 *taquin*) is a frame with 8 sliding tiles numbered from 1 to 8. You win when you obtain the

1	2	3
4	5	6
7	8	

original configuration out of a given

6	5	3
2	7	1
4	8	

configuration, for example .taquin-QR.pdf

Example: <http://taquin.net/>

- Each configuration of the 8-puzzle corresponds to a permutation of the set $\{0, 1, \dots, 8\}$ (where 0 corresponds to the empty slot). There are $9! = 362,880$ such permutations (although all of them are not obtainable out of the original configuration).

Let us puzzle!

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix

Weighted graph
Co-citation and
bibliographic
coupling

Orientation,
acyclicity,
topological sort

Discretion,
completeness
Cliques and
stable

Bipartite graphs
Incidence matrix

Trees, forests
Returning to
graphs: average
degree, density,
large networks

Paths

Graph Laplacian

- Let us consider puzzle configurations as vertices of a graph, the edges of which correspond to individual slides of a given tile. The weight of each edge is 1 (sliding a tile from one slot to another, horizontally or vertically).
- A possible heuristic is *Manhattan distance* (distance de *Manhattan*) between a given configuration and the original one: we count for each slot (except for the empty slot) the minimum amount of slides of the tile for it to achieve its original position, and we take the

6	5	3
2	7	1
4	8	

sum of amounts for all tiles. (E.g., $h(\text{configuration}) = 12$.)

- Another heuristic: *Hamming distance* (distance de *Hamming*): the number of wrongly placed tiles. We can compare the performances of the two heuristics.

Let us puzzle!

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stable
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

- We calculated the shortest path between the two configurations with and without heuristic (Python code available on Moodle), here are the performances we got:

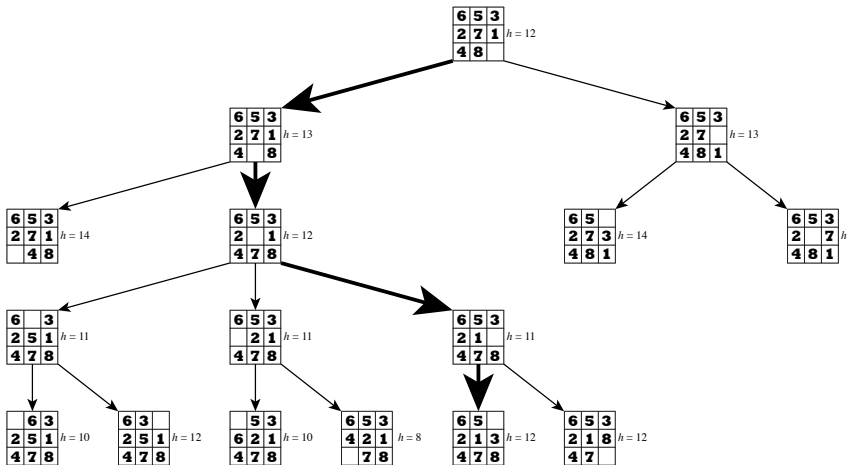
	Size of S	Time
Without heuristic (Dijkstra)	18715	31''
With heuristic h (A^*)	167	2''

- In both cases we found the same sequence of 18 slides:

	(2, 0, 5, 1, 6, 3, 4, 7, 8) $h = 9$
(6, 5, 3, 2, 7, 1, 4, 8, 0) $h = 12$	(2, 5, 0, 1, 6, 3, 4, 7, 8) $h = 8$
(6, 5, 3, 2, 7, 1, 4, 0, 8) $h = 13$	(2, 5, 3, 1, 6, 0, 4, 7, 8) $h = 7$
(6, 5, 3, 2, 0, 1, 4, 7, 8) $h = 12$	(2, 5, 3, 1, 0, 6, 4, 7, 8) $h = 6$
(6, 5, 3, 2, 1, 0, 4, 7, 8) $h = 11$	(2, 0, 3, 1, 5, 6, 4, 7, 8) $h = 5$
(6, 5, 0, 2, 1, 3, 4, 7, 8) $h = 12$	(0, 2, 3, 1, 5, 6, 4, 7, 8) $h = 4$
(6, 0, 5, 2, 1, 3, 4, 7, 8) $h = 13$	(1, 2, 3, 0, 5, 6, 4, 7, 8) $h = 3$
(0, 6, 5, 2, 1, 3, 4, 7, 8) $h = 12$	(1, 2, 3, 4, 5, 6, 0, 7, 8) $h = 2$
(2, 6, 5, 0, 1, 3, 4, 7, 8) $h = 11$	(1, 2, 3, 4, 5, 6, 7, 0, 8) $h = 1$
(2, 6, 5, 1, 0, 3, 4, 7, 8) $h = 10$	(1, 2, 3, 4, 5, 6, 7, 8, 0) $h = 0$

Let us puzzle!

- Here is the beginning of the graph of all possible slides starting from the original configuration (thick arrows are the start of the shortest path):



Minimal spanning trees

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

- In many situations we want to simplify a graph, by limiting the number of edges (without affecting vertices). An elegant way of filtering edges is to keep only those forming a tree. We then say that we have a *spanning tree* $\langle \text{arbre couvrant} \rangle$ of the graph.
- Among the huge amount of spanning trees, and provided our edges carry a weight, we may be interested in those of minimal global weight. Example: a water supply network, where the weight is the cost of tubes.
- We then seek for a *minimal spanning tree* $\langle \text{arbre couvrant minimal} \rangle$.
- Brute force is not the best way to find the minimal spanning tree. We will study two algorithms allowing to do so in an efficient way: Kruskal's and Prim's.

- Let G be an undirected connected graph, the edges of which carry a positive weight w . *Kruskal's algorithm* (algorithm de Kruskal) is as follows:

KRUSKAL-MST(G, w):

$A \leftarrow \emptyset$

sort edges of G in increasing order of w

for each edge uv taken in this order:

if not SAMECONNECTEDCOMPONENT(u, v):

$A \leftarrow A \cup \{uv\}$

return A

- The function SAMECONNECTEDCOMPONENT checks whether vertices u and v belong to the same connected component of the graph A which is under construction.
- At the end of execution of the algorithm, A is a minimal spanning tree of G .

ELU 501

Data

science,

graph theory

and social

network

studies

Yannis Ha-

ralambous

(IMT

Atlantique)

Basic graph
notions

Vertices, edges

Adjacency matrix

Weighted graph

Co-citation and

bibliographic

coupling

Orientation,

acyclicity,

topological sort

Discretion,

completeness

Cliques and

stables

Bipartite graphs

Incidence matrix

Trees, forests

Returning to

graphs: average

degree, density,

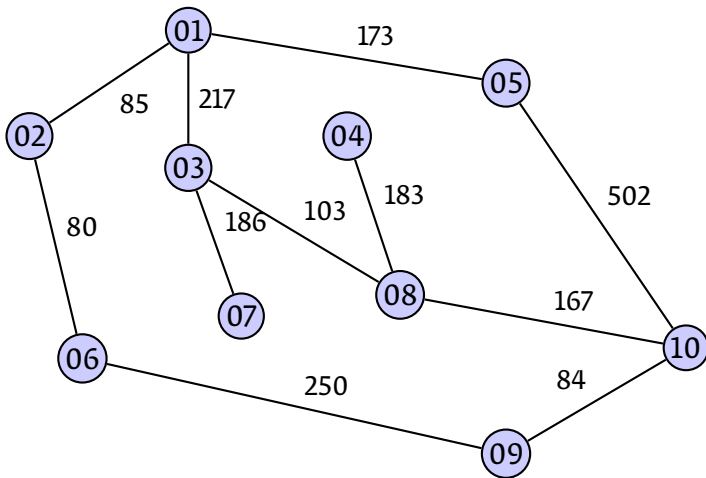
large networks

Paths

Graph Laplacian

- In other words: we add edges by increasing cost but only if they do not create a cycle.
- http://students.ceid.upatras.gr/~papagel/english/java_docs/minKrusk.htm
- Complexity: by using forests of rooted trees we can get a complexity of $O(E \log V)$.
- Advantage: easy to describe and to implement.
- Desadvantage: *no connectivity guaranteed* before the last iteration.

Kruskal's algorithm 1/10



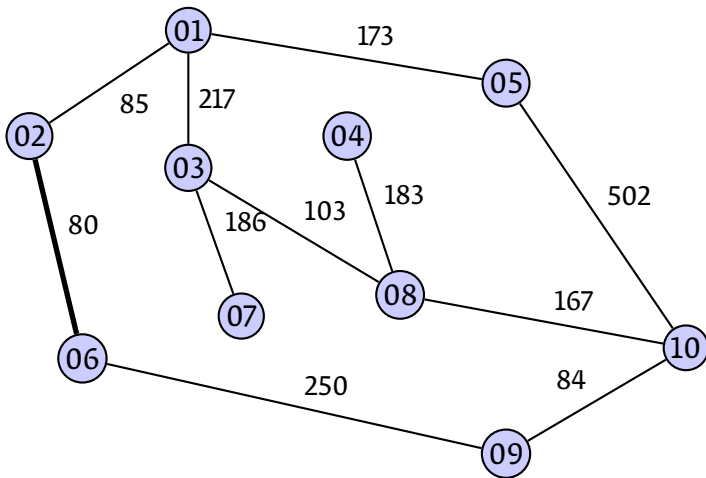
ELU 501
Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
 Adjacency matrix
 Weighted graph
 Co-citation and
 bibliographic
 coupling
 Orientation,
 acyclicity,
 topological sort
 Discretion,
 completeness
 Cliques and
 stables
 Bipartite graphs
 Incidence matrix
 Trees, forests
 Returning to
 graphs: average
 degree, density,
 large networks
 Paths
 Graph Laplacian

Kruskal's algorithm 2/10



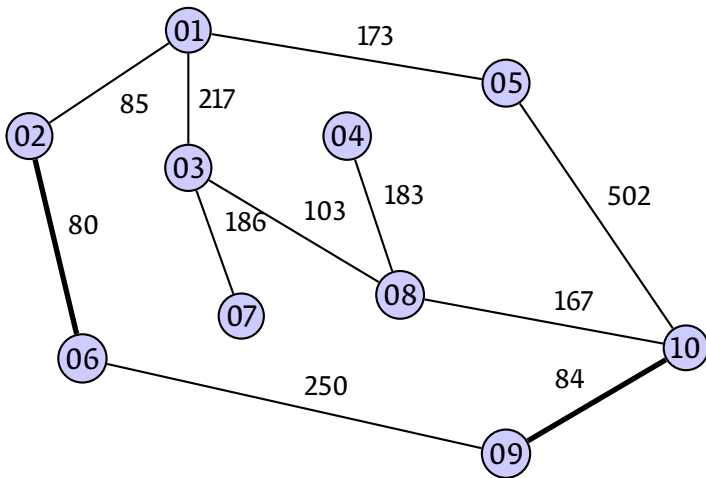
ELU 501
Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
 Adjacency matrix
 Weighted graph
 Co-citation and
 bibliographic
 coupling
 Orientation,
 acyclicity,
 topological sort
 Discretion,
 completeness
 Cliques and
 stables
 Bipartite graphs
 Incidence matrix
 Trees, forests
 Returning to
 graphs: average
 degree, density,
 large networks
 Paths
 Graph Laplacian

Kruskal's algorithm 3/10



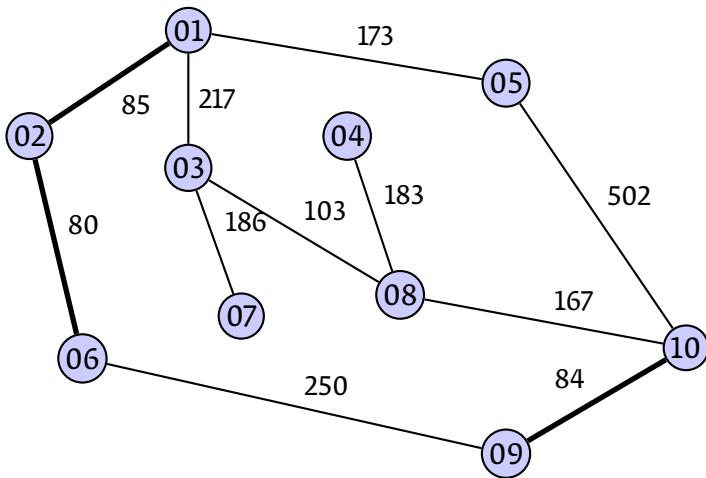
ELU 501
Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
 Adjacency matrix
 Weighted graph
 Co-citation and
 bibliographic
 coupling
 Orientation,
 acyclicity,
 topological sort
 Discretion,
 completeness
 Cliques and
 stables
 Bipartite graphs
 Incidence matrix
 Trees, forests
 Returning to
 graphs: average
 degree, density,
 large networks
 Paths
 Graph Laplacian

Kruskal's algorithm 4/10



ELU 501
Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
 Adjacency matrix
 Weighted graph
 Co-citation and
 bibliographic
 coupling
 Orientation,
 acyclicity,
 topological sort
 Discretion,
 completeness
 Cliques and
 stables
 Bipartite graphs
 Incidence matrix
 Trees, forests
 Returning to
 graphs: average
 degree, density,
 large networks
 Paths
 Graph Laplacian

Kruskal's algorithm 5/10

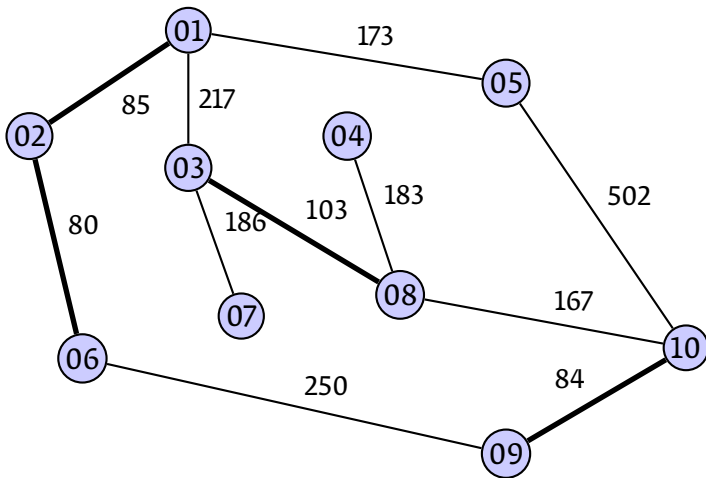
ELU 501

Data science,
graph theory
and social
network
studies

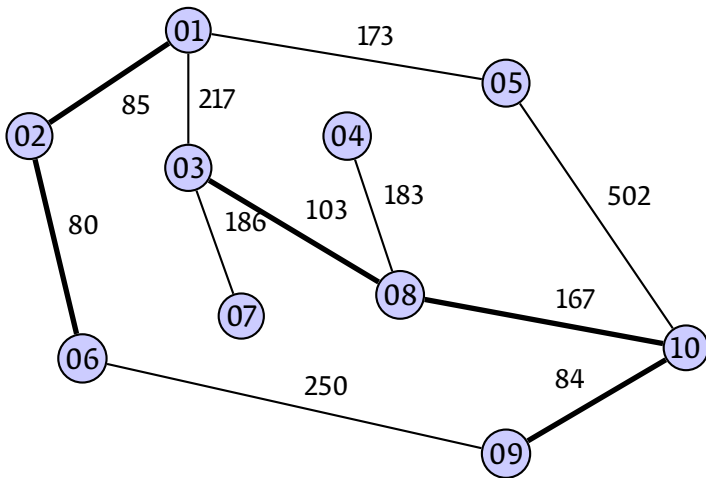
Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
 Adjacency matrix
 Weighted graph
 Co-citation and
 bibliographic
 coupling
 Orientation,
 acyclicity,
 topological sort
 Discretion,
 completeness
 Cliques and
 stables
 Bipartite graphs
 Incidence matrix
 Trees, forests
 Returning to
 graphs: average
 degree, density,
 large networks
 Paths
 Graph Laplacian



Kruskal's algorithm 6/10



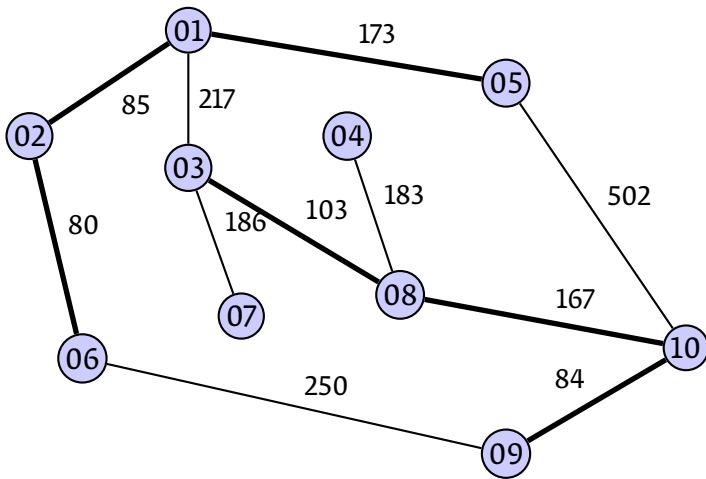
ELU 501
Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
 Adjacency matrix
 Weighted graph
 Co-citation and
 bibliographic
 coupling
 Orientation,
 acyclicity,
 topological sort
 Discretion,
 completeness
 Cliques and
 stables
 Bipartite graphs
 Incidence matrix
 Trees, forests
 Returning to
 graphs: average
 degree, density,
 large networks
 Paths
 Graph Laplacian

Kruskal's algorithm 7/10



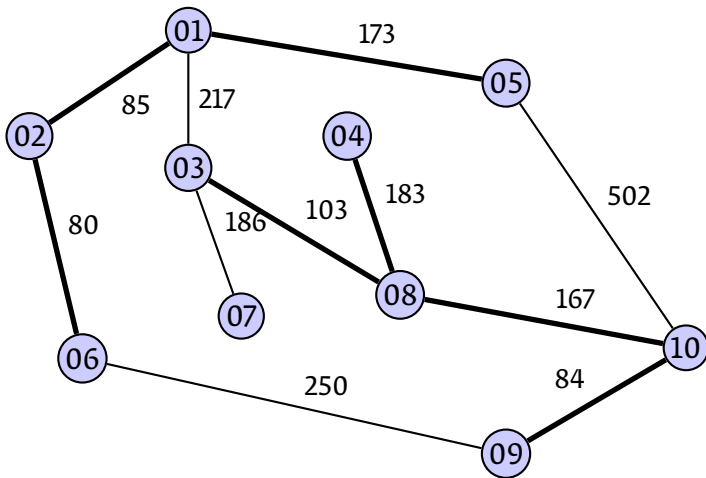
ELU 501
Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
 Adjacency matrix
 Weighted graph
 Co-citation and
 bibliographic
 coupling
 Orientation,
 acyclicity,
 topological sort
 Discretion,
 completeness
 Cliques and
 stables
 Bipartite graphs
 Incidence matrix
 Trees, forests
 Returning to
 graphs: average
 degree, density,
 large networks
 Paths
 Graph Laplacian

Kruskal's algorithm 8/10



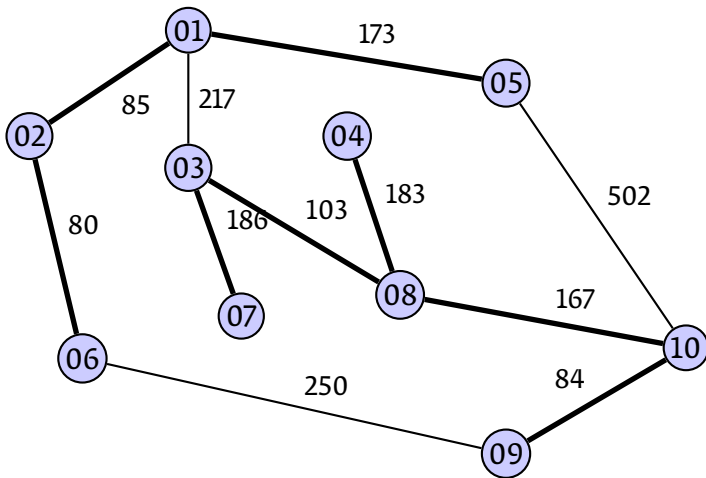
ELU 501
Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

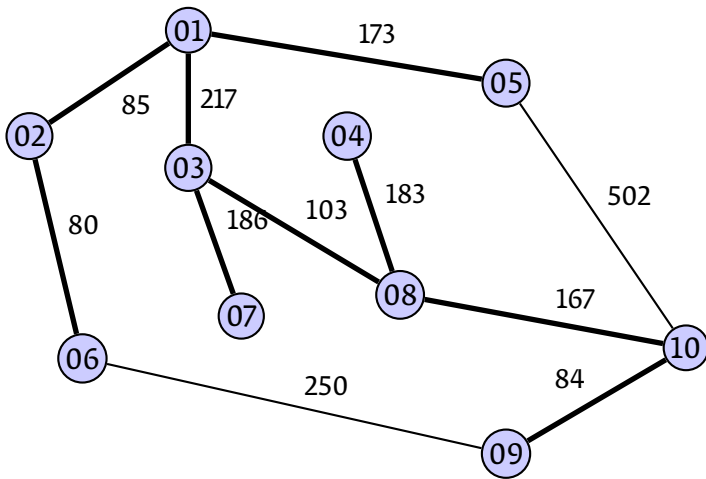
Vertices, edges
 Adjacency matrix
 Weighted graph
 Co-citation and
 bibliographic
 coupling
 Orientation,
 acyclicity,
 topological sort
 Discretion,
 completeness
 Cliques and
 stables
 Bipartite graphs
 Incidence matrix
 Trees, forests
 Returning to
 graphs: average
 degree, density,
 large networks
 Paths
 Graph Laplacian

Kruskal's algorithm 9/10



Data science,
graph theory
and social
network
studies

Yannis Haralambous
(IMT Atlantique)



ELU 501

Data

science,

graph theory

and social

network

studies

Yannis Ha-

ralambous

(IMT

Atlantique)

Basic graph
notions

Vertices, edges

Adjacency matrix

Weighted graph

Co-citation and

bibliographic

coupling

Orientation,

acyclicity,

topological sort

Discretion,

completeness

Cliques and

stables

Bipartite graphs

Incidence matrix

Trees, forests

Returning to

graphs: average

degree, density,

large networks

Paths

Graph Laplacian

- This time we choose an initial vertex r , we attach to all vertices values called *keys* $\langle \text{clé} \rangle$, and we build A by adding edges (without creating cycles) by always choosing the edge of minimum key.
- Whenever a new vertex is added, we do (pseudo-)relaxations of the keys of neighboring vertices.
- We have voluntarily written Prim's pseudo-code so that it resembles with the one of Dijkstra's algorithm:

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling

Orientation,
acyclicity,
topological sort

Discretion,
completeness
Cliques and
stables

Bipartite graphs
Incidence matrix

Trees, forests
Returning to
graphs: average
degree, density,
large networks

Paths
Graph Laplacian

- Let Q be a min priority queue.

PRIM-MST(G, w, r):

INITIALIZE-SINGLE-SOURCE $_{\phi}(G, r)$:

$A \leftarrow \{r\}, Q \leftarrow V$

while $Q \neq \emptyset$:

$u \leftarrow \text{EXTRACT-MIN}_{\phi}(Q)$

$A \leftarrow A \cup \{u\}$

for each vertex $v \in \text{NEIGHBORS}(u) \cap Q$:

PSEUDORELAX $_{\phi}(u, v, w)$

where PSEUDORELAX $_{\phi}$ is defined as follows:

PSEUDORELAX $_{\phi}(u, v, w)$:

if $\phi(v) > w(u, v)$:

$\phi(v) \leftarrow w(u, v)$

$\pi(v) \leftarrow u$

Comparison between Dijkstra and Prim

ELU 501

Data science,
graph theory and social
network studies

Yannis Haralambous
(IMT Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix

Weighted graph
Co-citation and
bibliographic
coupling

Orientation,
acyclicity,
topological sort

Discretion,
completeness

Cliques and
stables

Bipartite graphs
Incidence matrix

Trees, forests

Returning to
graphs: average
degree, density,
large networks

Paths

Graph Laplacian

- The main difference between RELAX and PSEUDORELAX is:

$\text{RELAX}_d(u, v, w):$

if $d(v) > \underline{d(u)} + w(u, v):$

$d(v) \leftarrow \underline{d(u)} + w(u, v)$

$\pi(v) \leftarrow u$

$\text{PSEUDORELAX}_\phi(u, v, w):$

if $\phi(v) > w(u, v):$

$\phi(v) \leftarrow w(u, v)$

$\pi(v) \leftarrow u$

- In the first case, $d(v)$ uses the value of $d(u)$ (together with the one of $w(u, v)$), so that the distance from the origin has *repercussions through the whole graph*.
- In the second case, the value of $\phi(v)$ depends only on edges adjacent to v , the rest of the graph does not interfere.

Complexity of Prim's algorithm

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian

- The body of the while loop is executed $|G|$ times, EXTRACT-MIN has a cost of $O(\log |V|)$ and the loop “for each edge” is executed $O(E)$ times. One can show that the global complexity of Prim's algorithm is $O(E \log V)$.
- (By using Fibonacci queues, we can lower it to $O(E + V \log V)$.)
- <http://students.ceid.upatras.gr/~papagel/project/prim.htm>

Prim's algorithm 1/10

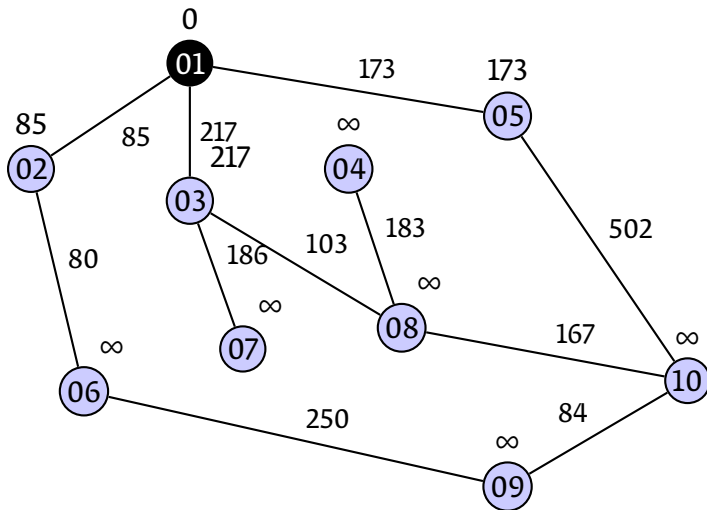
ELU 501

Data science,
graph theory
and social
network
studies

Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian



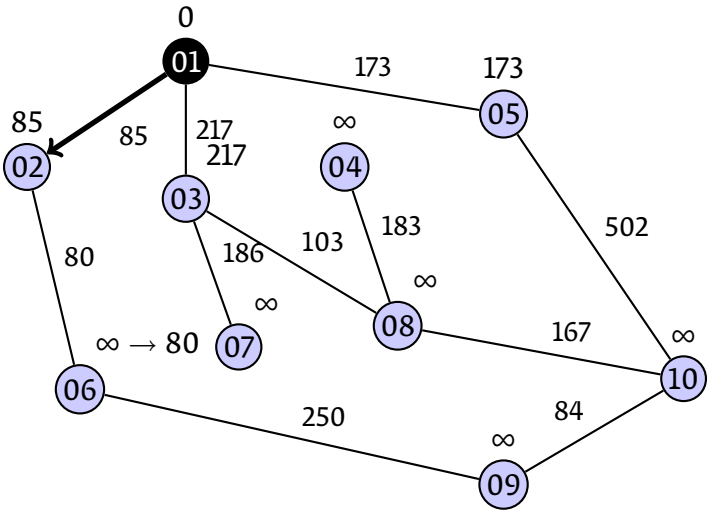
Prim's algorithm 2/10

ELU 501
Data science,
graph theory
and social
network
studies

Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian



Prim's algorithm 3/10

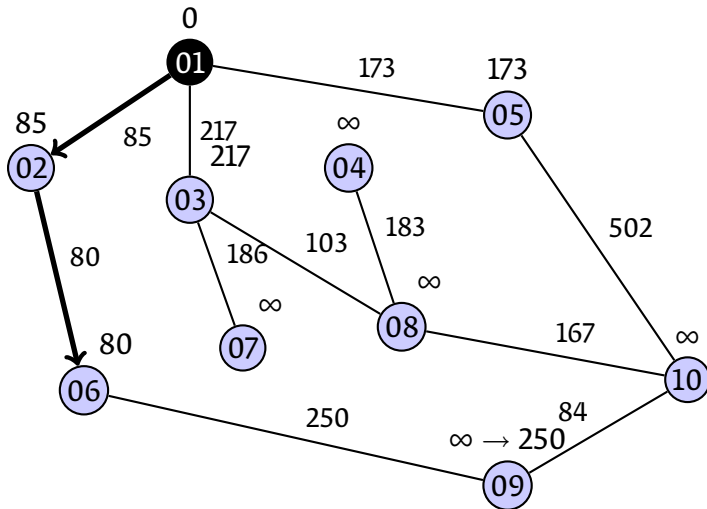
ELU 501

Data science,
graph theory
and social
network
studies

Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian



Prim's algorithm 4/10

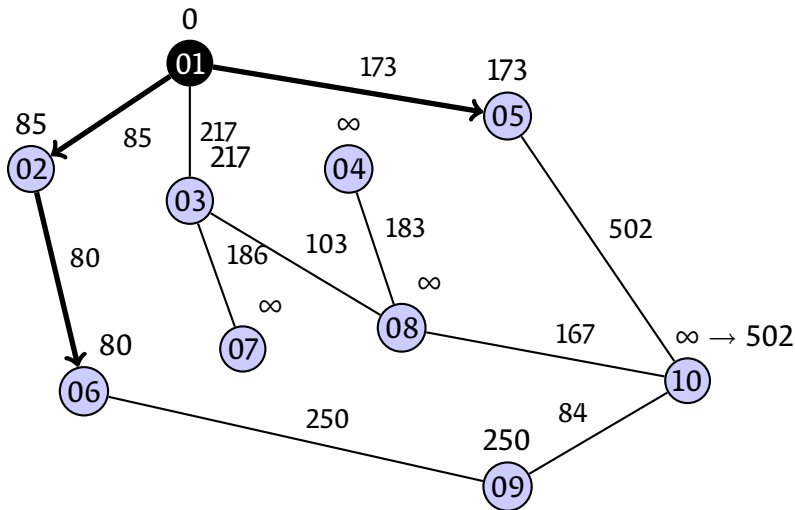
ELU 501

Data science,
graph theory
and social
network
studies

Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian



Prim's algorithm 5/10

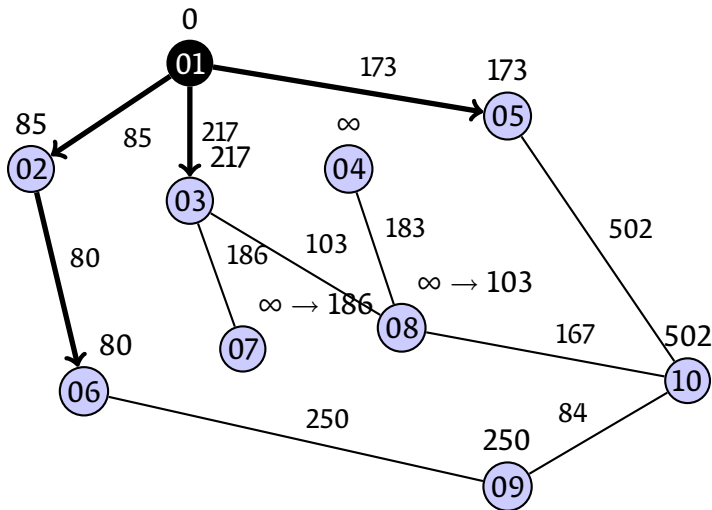
ELU 501

Data science,
graph theory
and social
network
studies

Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian



Prim's algorithm 6/10

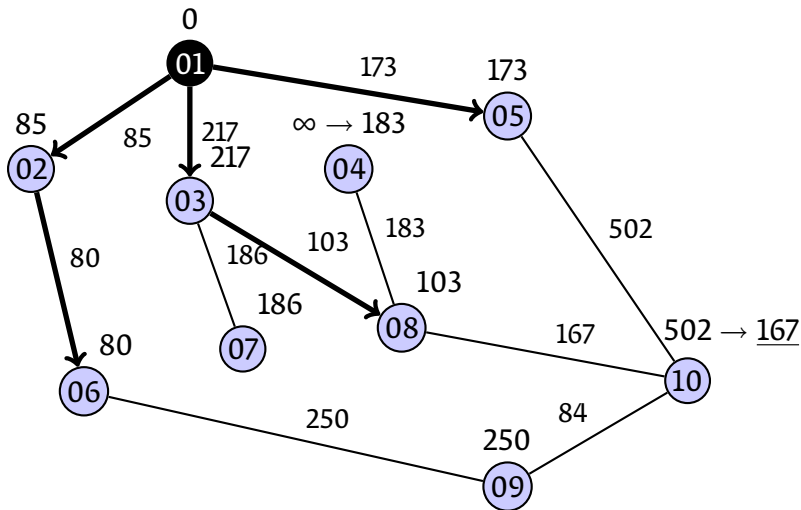
ELU 501

Data science,
graph theory
and social
network
studies

Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian



Prim's algorithm 7/10

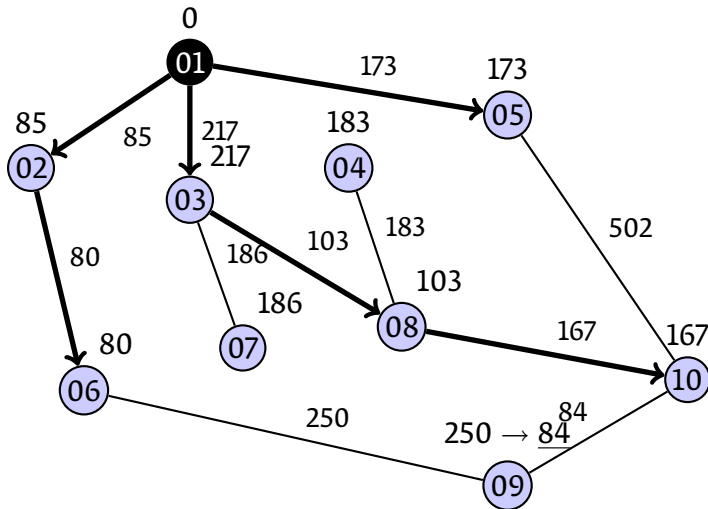
ELU 501

Data science,
graph theory
and social
network
studies

Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian



Prim's algorithm 8/10

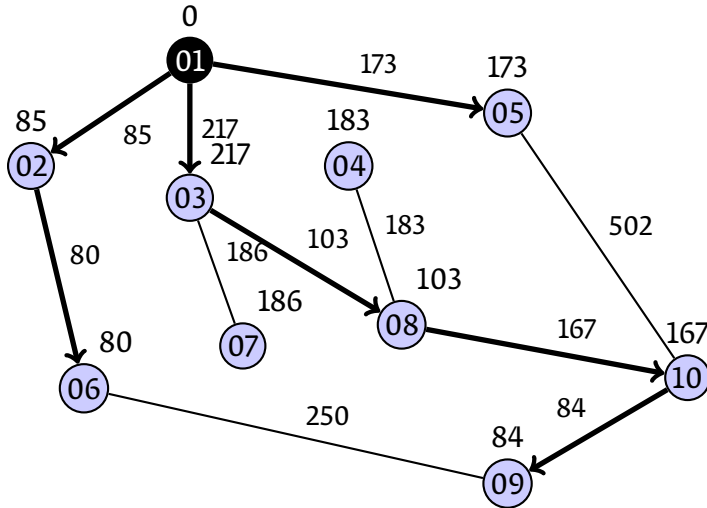
ELU 501

Data science,
graph theory
and social
network
studies

Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian



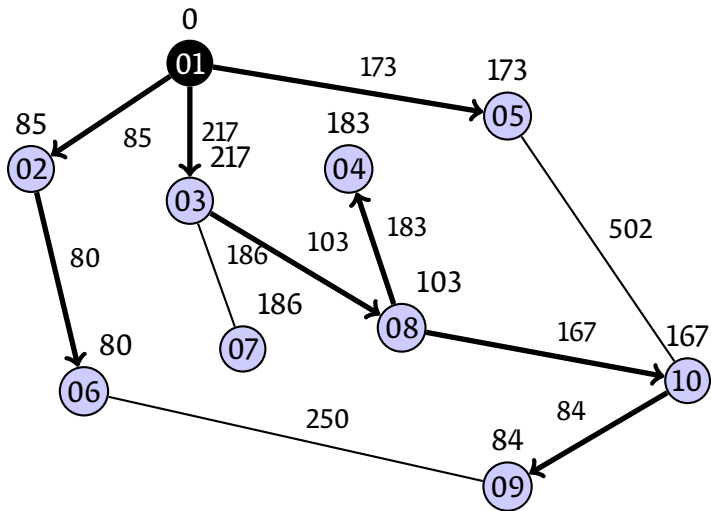
Prim's algorithm 9/10

ELU 501
Data science,
graph theory
and social
network
studies

Yannis Haralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian



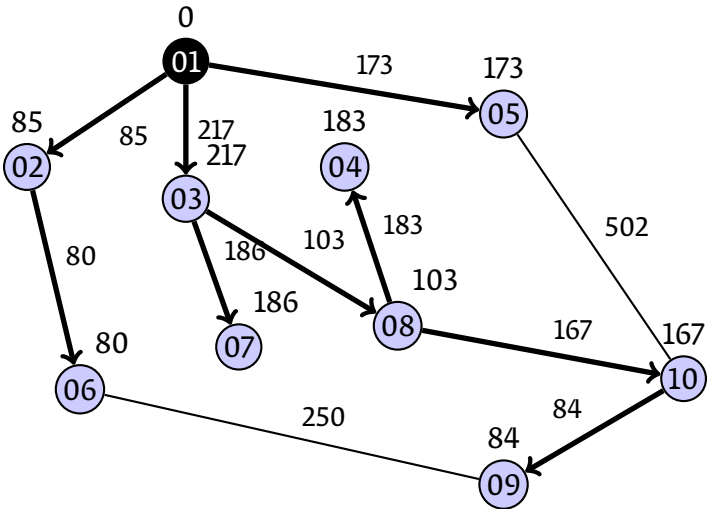
Prim's algorithm 10/10

ELU 501
 Data science,
 graph theory
 and social
 network
 studies

Yannis Ha-
 ralambous
 (IMT
 Atlantique)

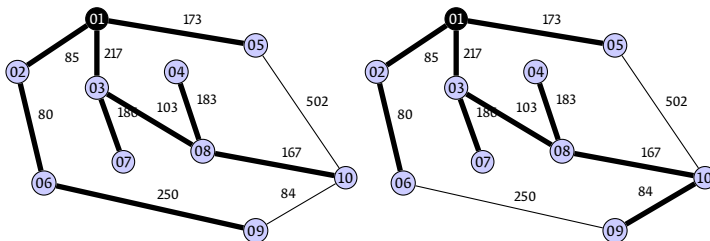
Basic graph
notions

Vertices, edges
 Adjacency matrix
 Weighted graph
 Co-citation and
 bibliographic
 coupling
 Orientation,
 acyclicity,
 topological sort
 Discretion,
 completeness
 Cliques and
 stables
 Bipartite graphs
 Incidence matrix
 Trees, forests
 Returning to
 graphs: average
 degree, density,
 large networks
 Paths
 Graph Laplacian



Do not confuse the two trees

Note that the trees obtained by Prim's algorithm (or by Kruskal's) and by Dijkstra's algorithms are not the same:



To the left, the *shortest paths tree* 🇫🇷 *arbre des plus courts chemins*, obtained by Dijkstra's or Bellman-Ford's algorithm. To the right, the *minimal spanning tree* 🇫🇷 *arbre couvrant minimal*, obtained by Prim's or Kruskal's algorithm.

ELU 501

Data
science,
graph theory
and social
network
studies

Yannis Ha-
ralambous
(IMT
Atlantique)

Basic graph
notions

Vertices, edges
Adjacency matrix
Weighted graph
Co-citation and
bibliographic
coupling
Orientation,
acyclicity,
topological sort
Discretion,
completeness
Cliques and
stables
Bipartite graphs
Incidence matrix
Trees, forests
Returning to
graphs: average
degree, density,
large networks
Paths
Graph Laplacian



CORMEN, LEISERSON, RIVEST & STEIN, *Introduction to algorithms*, 3th edition, MIT Press, 2009. [!\[\]\(2b376d1a92330ab09dad2665d2f89bf5_img.jpg\) CORMEN, LEISERSON, RIVEST & STEIN, *Algorithmique*, 3^e édition, Dunod, 2010](#)



M.E.J. NEWMAN, *Networks: An Introduction*, Oxford University Press, 2010.



And for those who are nostalgic of the French way of doing mathematics, elegant, crystal clear and ruthlessly theoretic you may consult the legendary C. BERGE, *The theory of graphs*, John Wiley, 1966. [!\[\]\(c444627dab9fee9a1550c053ffaaaae2_img.jpg\) C. BERGE, *Théorie des graphes et ses applications*, Dunod, 1963](#)