Graphs

Combinatorial optimization

Giovanni Righini

Università degli Studi di Milano

Definitions - 1

A graph, indicated as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, is defined by

- a set of *vertices* $\mathcal{V}$;
- a set of *edges* $\mathcal{E}$.

The vertex set $\mathcal{V}$ is an elementary set, i.e. its elements are just atomic items.
The edge set $\mathcal{E}$ is a complex set, i.e. its elements are sets: each edge is a pair of vertices in $\mathcal{V}$.

$$\mathcal{E} \subseteq \{[i,j] : i \in \mathcal{V}, j \in \mathcal{V}, i \neq j\}.$$

A digraph (directed graph), indicated as $\mathcal{D} = (\mathcal{N}, \mathcal{A})$, is defined by

- a set of *nodes* $\mathcal{N}$;
- a set of *arcs* $\mathcal{A}$.

Each arc is an *ordered* pair of nodes in $\mathcal{N}$.

$$\mathcal{A} \subseteq \{(i,j) : i \in \mathcal{N}, j \in \mathcal{N}, i \neq j\}.$$

We exclude *self-loops*, i.e. edges (arcs) whose *endpoints* coincide.

Definitions - 2

A *subgraph* $\mathcal{G}'$ induced by a subset $\mathcal{V}'$ of vertices is $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ with

$$\mathcal{E}' = \{[i,j] \in \mathcal{E} : i \in \mathcal{V}', j \in \mathcal{V}'.\}$$

An analogous definition applies to di-graphs.

We will not consider *multi-(di-)graphs* and *hyper-graphs*.

A multi-(di-)graph is a (di-)graph whose edge or arc set is a *multi-set*, i.e. it may contain multiple copies of the same element.

An hyper-graph is a graph whose edges are subsets of vertices, not necessarily pairs.

Definitions - 3

If $[i, j] \in \mathcal{E}$, then:

- vertices i and j are *adjacent*,
- edge $[i, j]$ is *incident* to vertex i and to vertex j.

If $(u, v) \in \mathcal{A}$, then:

- arc (u, v) is incident to node u and to node v,
- arc (u, v) *leaves* node u,
- arc (u, v) *enters* node v,
- node u is a *predecessor* of node v,
- node v is a *successor* of node u.

The *degree* of a vertex $i \in \mathcal{V}$ is the n. of edges $e \in \mathcal{E}$ incident to it.
The *in-degree* of a node $i \in \mathcal{N}$ is the n. of arcs $a \in \mathcal{A}$ entering it.
The *out-degree* of a node $i \in \mathcal{N}$ is the n. of arcs $a \in \mathcal{A}$ leaving it.

Definitions - 4

In a graph $G(V, E)$ a *connected component* of G is a subgraph $S = (U, E(U))$, where $U \subset V$ and $E(U) = \{[i,j] \in E : i \in U \wedge j \in U\}$, such that for each pair of nodes i and j in U, there is a path between them in $E(U)$.

In a digraph $D(N, A)$ a *strongly connected component* of D is a subgraph $S = (U, A(U))$, where $U \subset N$ and $A(U) = \{(i,j) \in A : i \in U \wedge j \in U\}$, such that for each ordered pair of nodes i and j in U, there is a directed path from i to j in $A(U)$.

Weights and objectives

A graph is *weighted* when there is a function associating a weight with each edge, i.e. $c : \mathcal{E} \mapsto \Re$, or vertex, i.e. $c : \mathcal{V} \mapsto \Re$.

The same definition applies to digraphs as well.

Weights often represent costs and the objective to be optimized (minimized) is the overall cost of a subset of edges or arcs, representing the solution: so we search for minimum cost paths, minimum cost trees, minimum cost flows, etc.

Combinatorial structures

When we solve *combinatorial optimization problems* on graphs, it is usually because we want to find the best among *solutions* with a particular structure:

- *paths*, representing origin to destination routes on street graphs;
- *trees*, representing links between geographically dispersed sites in telecommunication networks;
- *flows*, representing amounts of freight, passengers, goods, money... moving from one site to another;
- *matchings*, representing pairings in graphs of relations between people, activities, attributes,...;
- and many others...

The *solutions* correspond to *subsets of edges or arcs*.

Complexity - 1

The problem of finding the optimal edge (arc) subset with a given structure is *combinatorial* when the number of solutions is combinatorial in the number of edges or arcs, i.e. there as many solutions as the possible *combinations* of edges and arcs.

Due to the *combinatorial explosion* in the number of solutions, it is impractical to enumerate all of them explicitly: hence we need suitable (efficient) *graph optimization algorithms*.

According to the classification established by the Computational Complexity Theory, an algorithm is *efficient* if it computes an *optimal solution* taking *polynomial time and space* in the size of the instance.

Complexity - 2

In the case of graph optimization problems the size of the instance is the size of the graph.

We indicate with n the number of vertices or nodes.
We indicate with m the number of edges or arcs.
The computational complexity of graph optimization algorithms is usually given as a function of n and m.

We do not know polynomial complexity algorithms for all combinatorial optimization problems on graphs, but for some of them we do.

It is very important to know these well-solved cases, because they often occur as sub-problems within larger and more complicated optimization problems.

Complexity - 3

The maximum number of edges a graph can have is

$$m^{max} = \frac{n(n-1)}{2}.$$

The maximum number of arcs a digraph can have is

$$m^{max} = n(n-1).$$

A (di)graph is *complete* if and only if it contains m^{max} edges or arcs. A (di)graph is *dense* (*sparse*) when $\frac{m}{m^{max}}$ is large (small).

The density/sparsity of a graph can affect the computing time of graph optimization algorithms, according to the data-structures used to represent the graph.

Data-structures

There are many possibilities to store the information corresponding to a (di)graph in a computer memory, i.e. in a *data-structure*.

The choice of the most suitable data-structure depends:

- on the efficiency of the operations we need to execute on it;
- on the density/sparsity of the graph.

The most used data-structures to represent graphs are:

- adjacency matrix,
- incidence matrix,
- edge (arc) list,
- (in/out-)stars.

Adjacency matrix

An *adjacency matrix* M is a square $n \times n$ matrix, whose rows and columns correspond to vertices/nodes. Each entry $M[i,j]$ contains the piece of information associated with edge $[i,j]$ or arc (i,j).

For instance:
$M[i,j] = 0$ when $[i,j] \notin \mathcal{E}$ and $M[i,j] = 1$ when $[i,j] \in \mathcal{E}$;
$M[i,j] = \infty$ when $[i,j] \notin \mathcal{E}$ and $M[i,j] = c_{ij}$ when $[i,j] \in \mathcal{E}$.

An arc set requires the whole matrix; an edge set requires half of it.

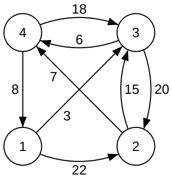

Figure: A digraph.

∞	22	3	∞
∞	∞	15	7
∞	20	∞	6
8	∞	18	∞

Table: Its adjacency matrix.

Incidence matrix

An *incidence matrix* M is an $n \times m$ matrix, whose rows correspond to vertices/nodes and whose columns correspond to edges/arcs.

Each entry $M[i, e]$ contains a significant piece of information if and only if edge e is incident in vertex i.
The same applies to arcs and nodes, with the sign indicating the direction.

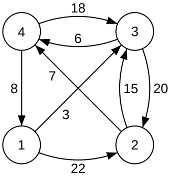

Figure: A digraph.

	(1, 2)	(1, 3)	(2, 3)	(2, 4)	(3, 2)	(3, 4)	(4, 1)	(4, 3)
1	-22	-3					8	
2	22		-15	-7	20			
3		3	15		-20	-6		18
4				7		6	-8	-18

Table: Its incidence matrix.

Edge (arc) list

An *edge (arc) list L* is a list of all the edges (arcs) in the (di-)graph.

Each element in the list is a record with all relevant information about the edge (arc).

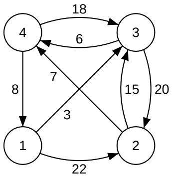

Figure: A digraph.

Node	Node	Cost
1	2	22
1	3	3
2	3	15
3	2	20
2	4	7
3	4	6
4	1	8
4	3	18

Table: Its arc list.

(In/Out-)stars

A *star S* is a list of all edges incident in a vertex.
An *in-star I* is a list of all arcs entering a node.
An *out-star O* is a list of all arcs leaving a node.

Each element in the list is a record with all relevant information about the edge (arc).

The whole (di-)graph is represented by a list of all (in/out-)stars, for all its vertices (or nodes).

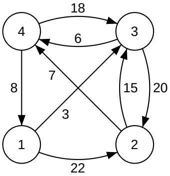

Figure: A digraph.

Node	Successors and weight
1	2,22 3, 3
2	3,15 4, 7
3	2,20 4, 6
4	1, 8 3,18

Table: Its out-stars.