Chapter 2

Modelling in Graph Theory

Many practical problems concerning relationships, networks, scheduling, or assignments can be modelled mathematically in the language of graph theory. In this chapter, we present some basic concepts of graph theory and one example problem with an elegant algorithmic solution.

2.1 Introductory Example

In GoogleMaps as well as in many modern cars you have the possibility to enter two geographical locations A and B and get "the shortest route" from A to B. Usually there are different choices for how to interpret "shortest". Of course, it will in most cases not mean "shortest euclidean distance", because in that case the solution would be trivial, hence uninteresting, namely the straight line through A and B. What makes the problem interesting and non-trivial is the underlying "network of streets", where a "route" from A to B is only allowed to follow these streets. Moreover, "shortest" can then mean "minimal w.r.t. length" or "minimal w.r.t. duration", i.e. the "quickest" route. Last but not least, various criteria can be applied for choosing appropriate streets, e.g. avoiding toll-road, avoiding highways, prefer scenic roads, calculate bicycle routes, etc. As an example see Figure 2.1, which shows the quickest bicycle route from JKU Linz to the RISC Institute in Hagenberg[1].

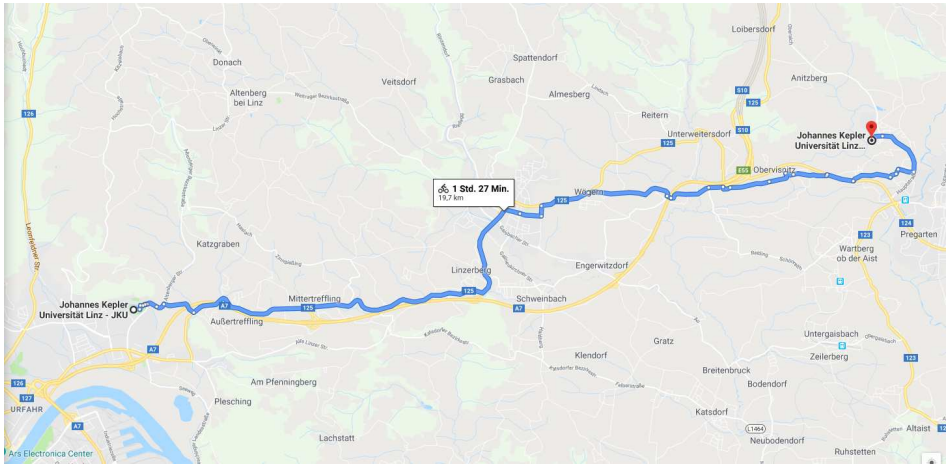

Figure 2.1: Google Maps Route Planner

[1] Note, however, that $1h27$ for not even 20km is a bit exaggerated.

2.2 Graph Theory

Graphs will turn out to be an appropriate mathematical model for a "network of streets" as needed in the routing example in Section 2.1. Informally speaking, in the heart of the concept of a graph is a collection of entities, called *vertices*, with connections between them, called *edges* or *arcs*, see Figure 2.2. There are different types of graphs, depending on whether for instance

- the connections are oriented or not,
- there can be more than one edge between to vertices,
- vertices connected by an edge must be distinct,
- edges have values associated,
- vertices have values associated,
- etc. etc.

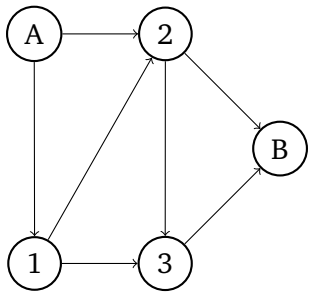

Figure 2.2: A typical graph

Definition 2.1: Undirected Simple Graph, Directed Simple Graph

Let V be a set. The pair $G = (V, E)$ is called an *undirected simple graph* if and only if

$$E \subseteq P(V) \text{ and } \underset{a \in E}{\forall} |a| = 2.$$

The pair (V, E) is called a *directed simple graph* if and only if

$$E \subseteq V^2 \text{ and } \underset{a \in E}{\forall} a_1 \neq a_2.$$

We call G a *simple graph* if and only if it is an undirected or a directed graph. If G is a graph, then $V(G) := G_1$ and $E(G) := G_2$ are called the *vertices* and *edges* of G, respectively.

Note that we use *sets* for representing edges, which implies that there cannot be multiple edges between two vertices. For certain applications, however, it is necessary to allow

multiple edges between vertices. In these cases, the edges E can be defined as *multisets* $E = (A, m)$, where

$$A = \{a \in P(V) \mid |a| = 2\} \qquad \text{(for undirected graphs)}$$
$$A = \{a \in V^2 \mid a_1 \neq a_2\} \qquad \text{(for directed graphs)}$$

and

$$m \colon A \to \mathbb{N}_0.$$

The set A is the set of potential edges, and the function m gives the multiplicity of each edge, including the case $m(a) = 0$ meaning that the graph does not contain the edge a. For a multiset $E = (A, m)$ we write $a \in E$ if and only if $a \in A$ and $m(a) \geq 1$.

In the multiset representation, the multiplicity of an edge tells us only, how many connections we have, but these are indistinguishable. If we want to have distinct edges, then we define the edges as $E = (X, e)$, where X is a set and

$$e \colon X \to A \qquad \text{with } A \text{ as above.}$$

For $E = (X, e)$ we write $a \in E$ if and only if $\exists_{x \in X} e(x) = a$.

EXAMPLE 2.2

Figure 2.3 shows different types of graphs. The leftmost is an undirected simple graph

$$G_1 = (\{1, 2, 3\}, \{\{1, 2\}, \{2, 3\}\}),$$

second from left is a directed simple graph

$$G_2 = (\{1, 2, 3\}, \{(1, 2), (2, 1), (1, 3), (2, 3)\}).$$

Second from right is not an undirected simple graph because it contains a double edge between 1 and 2, which can be defined as

$$G_3 = (\{1, 2, 3\}, (\{\{1, 2\}, \{1, 3\}, \{2, 3\}\}, m)) \quad \text{with } m \text{ defined by}$$

a	$m(a)$
$\{1, 2\}$	2
$\{1, 3\}$	0
$\{2, 3\}$	1

Note that one cannot distinguish the two edges between 1 and 2. The rightmost is not a simple directed graph either because it contains two distinct edges a and b from 1 to 2, which can be defined as

$$G_4 = (\{1, 2, 3\}, (\{a, b, c\}, e)) \quad \text{with } e \text{ defined by}$$

x	$e(x)$
a	$(1, 2)$
b	$(1, 2)$
c	$(2, 3)$

By definition, a simple graph cannot contain *loops*, i.e. edges connecting a vertex with itself. In an undirected graph, this would be an edge $\{u, u\}$, but $|\{u, u\}| = 1$, whereas in a

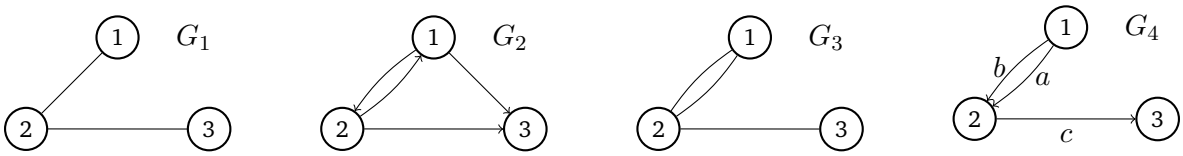

Figure 2.3: Different types of graphs

directed graph it would be an edge $a = (u, u)$, but $a_1 = u = a_2$. If loops should be permitted the restrictions $|a| = 2$ and $a_1 \neq a_2$, respectively, have to be relaxed.

From now on, if we say G is a graph then we mean a graph in one of the representations mentioned above. If we say G is a simple graph then we mean a simple undirected or a simple directed graph. If G is an undirected graph then we use uv as an abbreviation for an edge $\{u, v\} \in E(G)$, if G is directed then uv stands for an edge $(u, v) \in E(G)$.

DEFINITION 2.3: NEIGHBOUR, NEIGHBOURHOOD

Let G be a graph. The vertex v is a *neighbour of* vertex u if and only if $uv \in E(G)$. Furthermore, we call

$$N_G(u) := \{v \in V(G) \mid v \text{ is a neighbour of } u\}$$

the *neighbourhood* of u (in G).

EXAMPLE 2.4

Using the graphs from Figure 2.3,

$$\begin{array}{lll} N_{G_1}(1) = \{2\} & N_{G_1}(2) = \{1, 3\} & N_{G_1}(3) = \{2\} \\ N_{G_2}(1) = \{2, 3\} & N_{G_2}(2) = \{1, 3\} & N_{G_2}(3) = \{\} \\ N_{G_3}(1) = \{2\} & N_{G_3}(2) = \{1, 3\} & N_{G_3}(3) = \{2\} \\ N_{G_4}(1) = \{2\} & N_{G_4}(2) = \{3\} & N_{G_4}(3) = \{\} \end{array}$$

DEFINITION 2.5: WALK, PATH

Let G be a graph, $n \geq 1$, and $a, b \in V(G)$. A finite sequence[a] $w \colon \mathbb{N}_{1,2n+1} \to V(G) \cup E(G)$ is called a *walk* of length n from a to b in G if and only if

1. $\bigvee\limits_{0 \leq i \leq n} w_{2i+1} \in V(G)$,
2. $\bigvee\limits_{1 \leq i \leq n} w_{2i} \in E(G)$,
3. $w_1 = a$ and $w_{2n+1} = b$, and
4. $\bigvee\limits_{1 \leq i \leq n} w_{2i} = w_{2i-1} w_{2i+1}$.

Let w be a walk of length n from a to b in G, then the subsequences

$$V(w)\colon \mathbb{N}_{1,n+1} \to V(G), i \mapsto w_{2i-1} \qquad E(w)\colon \mathbb{N}_{1,n} \to E(G), i \mapsto w_{2i}$$

are the *sequence of vertices* of w and the *sequence of edges* of w, respectively.

A finite sequence $t\colon \mathbb{N}_{1,2n+1} \to V(G) \cup E(G)$ is called a *trail* of length n from a to b in G if and only if t is a walk of length n from a to b in G and $E(t)$ is injective (from $\mathbb{N}_{1,n}$ to $E(G)$).

A finite sequence $p\colon \mathbb{N}_{1,2n+1} \to V(G) \cup E(G)$ is called a *path* of length n from a to b in G if and only if p is a trail of length n from a to b in G and $V(p)$ is injective (from $\mathbb{N}_{1,n+1}$ to $V(G)$).

[a] Reminder: $\mathbb{N}_{m,n} := \{i \in \mathbb{N}_0 \mid m \leq i \leq n\}$.

A walk is an arbitrary sequence of vertices connected by appropriate edges. A trail does not use any edge twice and a path in addition does not visit any vertex twice. A walk that is not a trail must use at least one edge more than once, which implies that it also visits some vertices more than once, namely those vertices that are connected by that edge.

The length n of a walk/trail/path addresses the number of edges in that walk/trail/path. If we omit the length or the start and end then we assume the respective entities existentially quantified, e.g. if p is a path from a to b in G then we mean a path *of some length* from a to b in G, or if p is a path of length n in G then we mean a path of length n *from somewhere to somewhere* in G, or if p is a path in G then we mean a path *of some length from somewhere to somewhere* in G.

Example 2.6

$w = (2, (2,1), 1, (1,2), 2, (2,1), 1, (1,3), 3)$ is a walk of length 4 from 2 to 3 in G_2.

$$V(w) = (2,1,2,1,3) \qquad E(w) = ((2,1),(1,2),(2,1),(1,3)),$$

w is neither a trail nor a path in G_2.
$t = (2, (2,1), 1, (1,2), 2, (2,3), 3)$ is a trail of length 3 from 2 to 3 in G_2.

$$V(t) = (2,1,2,3) \qquad E(t) = ((2,1),(1,2),(2,3)),$$

t is not a path in G_2.
$p = (2, (2,1), 1, (1,3), 3)$ is a path of length 2 from 2 to 3 in G_2.

$$V(p) = (2,1,3) \qquad E(p) = ((2,1),(1,3)).$$

Definition 2.7: Weighted Graph

The triple $G = (V, E, c)$ is called a *weighted graph* if and only if (V, E) is a graph and $c\colon E \to \mathbb{R}$. We call c the *cost function* of G and $c(e)$ the *costs* of an edge e.

All special properties of the graph (V, E), e.g. being simple, directed, undirected, translate directly to its weighted variant (V, E, c). A sequence is a walk/trail/path in (V, E, c) if and only if it is a walk/trail/path in (V, E).

DEFINITION 2.8: EXTENDED COST FUNCTION, DISTANCE

Let $G = (V, E, c)$ be a weighted graph and $F \subseteq E$. Then we extend the cost function c to

$$c(F) := \sum_{e \in F} c(e).$$

Let w be a walk of length n from a to b in G, then

$$c(w) := \sum_{e \in E(w)} c(e).$$

The *distance* from a to b in G is

$$\text{dist}_G(a, b) := \min(\{c(w) \mid w \text{ is a walk from } a \text{ to } b \text{ in } G\}).$$

2.3 MODELLING THE SHORTEST-CONNECTION-PROBLEM

Our original problem from Section 2.1 can now be nicely formulated in the language of graph theory. Given a network of streets, we define the vertices

$$V := \{C \mid \text{there are streets } s \text{ and } t \text{ crossing at } C\}.$$

Then all streets split into *street segments* between crossings, in other words, a street segment is characterized by its endpoints, i.e. two crossings c_1 and c_2 on the same street such that no other crossing lies between c_1 and c_2. The street segments will become the edges connecting their endpoint crossings. If a street segment between c_1 and c_2 can be travelled in both directions, then an undirected edge $\{c_1, c_2\}$ is an appropriate model. If it can only be used in one direction, say from c_1 to c_2, then a directed edge (c_1, c_2) is an appropriate model. For typical street networks, directed edges are the better choice because these allow to model one-way streets and in case both directions are available we can use (c_1, c_2) and (c_2, c_1), thus

$$E := \{(c_1, c_2) \in V^2 \mid \text{there is a street segment from } c_1 \text{ to } c_2\}.$$

Every street segment (c_1, c_2) has costs associated, which can be the geographical distance between the c_1 and c_2 or the time required to travel from c_1 to c_2, depending on the application. Note, that the model with directed edges gives also more flexibility in this regard, because it allows to assign different times for the two directions, which is useful for instance when the street segment is a mountain road, where it is usually faster going down than going up. Note that the costs will always be non-negative, i.e.

$$c\colon E \to \mathbb{R}_0^+, (x, y) \mapsto \text{“costs” for going from } x \text{ to } y$$

The question of finding the “shortest connection” from A to B on the underlying network of roads is then just the following problem:

PROBLEM 2.9: SHORTEST PATH PROBLEM

Given: The graph $G = (V, E, c)$ with appropriate cost function c and $A, B \in V$.

Find: $d = \text{dist}_G(A, B)$ and p such that p is a path from A to B in G and $c(p) = d$.

Note that it is sufficient to find a path, we need not search for walks or trails. Assume a walk

$$w = (A, e_1, \ldots, e_k, x, \ldots, x, e_l, \ldots, e_n, B)$$

from A to B in G containing vertex x twice. Then take

$$\bar{w} = (A, e_1, \ldots, e_k, x, e_l, \ldots, e_n, B),$$

which certainly gives also a walk from A to B in G with $c(\bar{w}) \leq c(w)$, such that we can always construct a walk with costs at most $c(w)$ avoiding multiple vertices. Such a walk must be a path.

Note also that we modeled the graph above as a simple graph, i.e. a graph without multiple edges and loops. If the original street network contains multiple edges, which might be the case in real world (think of examples!), then we choose the one with least cost and only put this edge into E, hence avoiding multiple edges. Consider a and b being both edges from x to y and $c(a) \leq c(b)$ and assume a path

$$p = (A, e_1, \ldots, x, b, y, \ldots, e_n, B)$$

from A to B in G. Then take

$$\bar{p} = (A, e_1, \ldots, x, a, y, \ldots, e_n, B),$$

which certainly gives also a path from A to B in G with $c(\bar{p}) \leq c(p)$, such that we can always construct a path with costs at most $c(p)$ avoiding edge b. Similary, we can eliminate loops contained in G. Consider a being a loop from x to x and assume a trail

$$p = (A, e_1, \ldots, e_k, x, a, x, e_l, \ldots, e_n, B)$$

from A to B in G. Then take

$$\bar{p} = (A, e_1, \ldots, e_k, x, e_l, \ldots, e_n, B),$$

which certainly gives also a path from A to B in G with $c(\bar{p}) \leq c(p)$ since $c(a) \geq 0$, such that we can always construct a path with costs at most $c(p)$ avoiding loop a.

2.4 AN ALGORITHM FOR SOLVING THE SHORTEST PATH PROBLEM

We will now briefly describe *Dijkstra's algorithm* for solving the shortest path problem. In fact, the algorithms solves a slightly more general problem, namely it computes $\text{dist}_G(A, v)$ for all $v \in V \setminus \{A\}$ and it allows to reconstruct shortest paths from A to v for all $v \in V \setminus \{A\}$.

The basic idea of the algorithm is to maintain two sets of vertices C and $O = V \setminus C$, where

- C contains the *closed vertices* v, for which $\text{dist}_G(A, v)$ is *already known* and

- O are the remaining *open vertices* v, for which only a *tentative distance* $l(v)$ *from* A *is known*. In fact, $l(v)$ is the shortest distance from A on a path containing only vertices in C except the final vertex v.

Data: $G = (V, E, c)$ a simple graph, $A \in V$.
Result: $\text{dist}_G(A, v)$ for all $v \in V$,
$\text{pre}_G(v)$ for all $v \in V$ such that[a] $P \asymp ((\text{pre}_G(v), v), v)$ is a shortest path from A to v in G, where P is a shortest path from A to $\text{pre}_G(v)$ in G.

$C = \emptyset$, $O = V$, $l(A) = 0$
for $v \in V \setminus \{A\}$ **do**
  $l(v) = \infty$
end
while $O \neq \emptyset$ **do**
  $v =$ such an $o \in O$ with $l(o) \leq l(x)$ for all $x \in O$
  $C = C \cup \{v\}$, $O = O \setminus \{v\}$
  $\text{dist}_G(A, v) = l(v)$
  for $w \in N_G(v)$ **do**
    if $l(w) > \mathit{dist}_G(A, v) + c((v, w))$ **then**
      $l(w) = \text{dist}_G(A, v) + c((v, w))$
      $\text{pre}_G(w) = v$
    end
  end
end

Algorithm 1: Dijkstra's Algorithm

[a] We write $a \asymp b$ for the concatenation of tuples a and b.

Upon termination of the algorithm $\text{dist}_G(A, v)$ contains the shortest distance from A to v in G for all vertices v. If $\text{dist}_G(A, v) = \infty$ then there is no path from A to v in G.

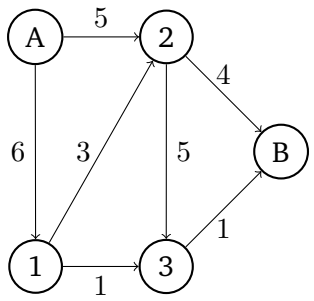

Figure 2.4: A weighted graph

EXAMPLE 2.10

Let us consider the graph depicted in Figure 2.4. After initialization we have $C = \emptyset$, $O = V$ and

v	A	1	2	3	B
$l(v)$	0	∞	∞	∞	∞

In the first run through the while-loop, we choose $v = \mathrm{A}$, hence $C = \{\mathrm{A}\}$ and $O = \{1, 2, 3, \mathrm{B}\}$. We set $\mathrm{dist}_G(\mathrm{A}, \mathrm{A}) = 0$ and compute $N_G(\mathrm{A}) = \{1, 2\}$, hence we update

v	A	1	2	3	B
$l(v)$	0	6	5	∞	∞

and set $\mathrm{pre}_G(1) = \mathrm{A}$ and $\mathrm{pre}_G(2) = \mathrm{A}$. Next we choose $v = 2$, hence $C = \{\mathrm{A}, 2\}$ and $O = \{1, 3, \mathrm{B}\}$. We set $\mathrm{dist}_G(\mathrm{A}, 2) = 5$ and compute $N_G(2) = \{3, \mathrm{B}\}$, hence we update

v	A	1	2	3	B
$l(v)$	0	6	5	10	9

and set $\mathrm{pre}_G(3) = 2$ and $\mathrm{pre}_G(\mathrm{B}) = 2$. Next we choose $v = 1$, hence $C = \{\mathrm{A}, 2, 1\}$ and $O = \{3, \mathrm{B}\}$. We set $\mathrm{dist}_G(\mathrm{A}, 1) = 6$ and compute $N_G(1) = \{2, 3\}$, hence we update

v	A	1	2	3	B
$l(v)$	0	6	5	7	9

and set $\mathrm{pre}_G(3) = 1$. Next we choose $v = 3$, hence $C = \{\mathrm{A}, 2, 1, 3\}$ and $O = \{\mathrm{B}\}$. We set $\mathrm{dist}_G(\mathrm{A}, 3) = 7$ and compute $N_G(3) = \{\mathrm{B}\}$, hence we update

v	A	1	2	3	B
$l(v)$	0	6	5	7	8

and set $\mathrm{pre}_G(\mathrm{B}) = 3$. Finally, we take $v = \mathrm{B}$, hence $C = \{\mathrm{A}, 2, 1, 3, \mathrm{B}\}$ and $O = \emptyset$. We set $\mathrm{dist}_G(\mathrm{A}, \mathrm{B}) = 8$, no more updates necessary and we exit the while-loop.

We have all distances from A to v in $\mathrm{dist}_G(\mathrm{A}, v)$, and we can reconstruct the shortest path using $\mathrm{pre}_G(v)$. The final path is

$$(\mathrm{A}, (\mathrm{A}, 1), 1, (1, 3), 3, (3, \mathrm{B}), \mathrm{B}).$$