Aalto University

Lecture Notes - Week I

Welcome to Combinatorics

Fernando Dias, Philine Schiewe and Piyalee Pattanaik

January 29, 2024

CHAPTER 1 Definitions

Here are a few definitions to start the course. First, **combinatorial**:

- *adjective*
- relating to selecting a given number of elements from a larger number without regard to their arrangement.

Now optimization (or optimization, whichever spelling is your favourite):

- *noun*
- the action of making the best or **most effective** use of a **situation** or **resource**.

As any part of optimization, it can be achieved by either analyzing/Visualizing properties of functions / extreme points or by applying numerical methods. Finally, optimization has important applications in fields such as economics, statistics, bioinformatics, machine learning, and artificial intelligence.

CHAPTER 2
Mathematical programming and optimization

In this course, optimization is viewed as the core element of mathematical programming, which is a central OR modelling paradigm. It can be simply defined using three major concepts: **variables**, **domain** and **functions**.

Variables correspond to decisions/points of interest (business decisions, parameter definitions, settings, geometries, among others). In our formulations, it will be the values where changes will be applied, and the goal is to find the best values, according to each particular problem. Limiting which values each variable can assume, the **domain** which represents constraints and limitations (such as logic, design, engineering, etc.). **Objective functions** (which represent performance and quality measurements) are used to evaluate which variable has the best value, considering the limitations present in the constraints.

However, mathematical programming has many applications in fields other than OR, which causes some confusion. In this course, we will study mathematical programming in its most general form: both constraints and objectives are nonlinear functions.

CHAPTER 3 Types of mathematical optimization models

As in any field of optimization, the following rule of thumb is always valid:

The simpler are the assumptions which define a type of problem, the better are the methods to solve such problems.

For this course (and optimization in general), the following notations are useful:

- $x \in \mathbb{R}^n$: vector of (decision) variables x_j, $j = 1, \dots, n$;
- $f : \mathbb{R}^n \to \mathbb{R} \cup \{\pm\infty\}$ - objective function;
- $X \subseteq \mathbb{R}^n$: ground set (physical constraints);
- $g_i, h_i : \mathbb{R}^n \to \mathbb{R}$: constraint functions;
- $g_i(x) \leq 0$ for $i = 1, \dots, m$: inequality constraints;
- $h_i(x) = 0$ for $i = 1, \dots, l$: equality constraints.

Our goal will be to solve variations of the general problem P:

$$\begin{aligned}(P):\ & \min f(x) \\ & \text{s.t. } g_i(x) \leq 0, i = 1, \dots, m \\ & \quad h_i(x) = 0, i = 1, \dots, l \\ & \quad x \in X.\end{aligned}$$

Which applies to any sub-field of optimization:

- **Linear programming (LP):** linear $f(x) = c^\top x$ with $c \in \mathbb{R}^n$; constraint functions $g_i(x)$ and $h_i(x)$ are affine ($a_i^\top x - b_i$, with $a_i \in \mathbb{R}^n$, $b \in \mathbb{R}$); $X = \{x \in \mathbb{R}^n : x_j \geq 0, j = 1, \dots, n\}$.
- **Nonlinear programming (NLP):** some (or all) of the functions f, g_i or h_i are nonlinear;
- **(Mixed-)integer programming ((M)IP):** LP where (some of the) variables are binary (or integer). $X \subseteq \mathbb{R}^k \times \{0, 1\}^{n-k}$
- **Mixed-integer nonlinear programming (MINLP):** MIP+NLP.

In this course, we might face **any** of the previous sub-field, but the major change is that variables can be **discretized** (binary or integer).

CHAPTER 4
Graphs

Some **useful definitions** for this course are:

- **graph** $G = (V, E, \psi)$: a powerful tool used in **discrete mathematics** and **graph theory**, where objects are represented in the form of ”relation”;

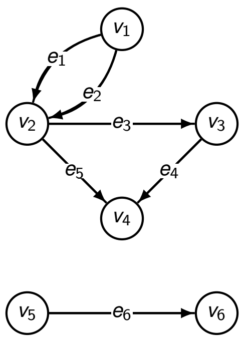
- **undirected** graph are sub-categories of graphs where the direction of interaction does not matter.
  - vertices V (or nodes and points);
  - edges E (or links, arcs and line);
  - function $\psi\colon E \to \{X \subseteq V\colon |X| = 2\}$

- **directed** (where the edge direction is crucial) graph $G = (V, E, \psi)$
  - vertices V
  - edges E
  - function $\psi\colon E \to \{(v, w) \in V \times V\colon v \neq w\}$

- Edges e can have a value associated with it: $\to w \longrightarrow f_{uv}$ called **weight** or **flow**;

- in practice: $e = \{u, v\}$, $e = (u, v)$ respectively, $G = (V, E)$

Remark: E can contain multiple parallel edges.



Several structures can be extracted from graphs, especially based on **edge progression**. Considering W in G from u_1 to u_{k+1}, as an **edge progression** with the following progression:

- sequence $[u_1, a_1, u_2, \dots, u_k, a_k, u_{k+1}]$ with $k \geq 0$
- $a_i = \{u_i, u_{i+1}\} \in E(G)$
- e.g. $[v_3, e_3, v_2, e_2, v_1, e_1, v_2, e_3, v_3, e_4, v_4]$

Generally, **walks** encompasses any edge progression. It can be separated in **closed** and **open** walks. For the former, a walk is considered an open walk if the starting and ending nodes are different, i.e. the starting node and the finishing are different. At the same time, the latter is a closed walk if the starting and ending nodes are identical, i.e. if a walk starts and ends at the same node, then it is said to be a closed walk.

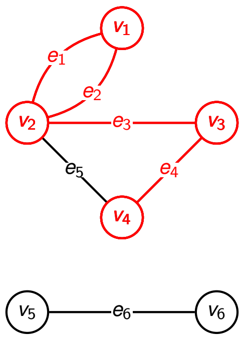
- edge progression with $a_i \neq a_j$, $1 \leq i < j \leq k$
- e.g. $[v_2, e_2, v_1, e_1, v_2, e_3, v_3, e_4, v_4]$

However, it can be decomposed into smaller definitions:

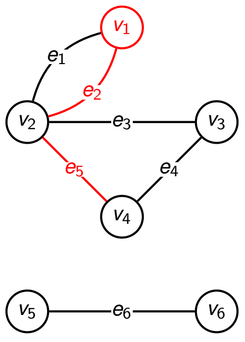
- trail: an open walk in which **no edge is repeated**;
- circuit: **closed** trail;
- cycle: same **starting** and **ending** node.

Another case is a **path**, which is a walk with **no repeating** nodes. For a **path** P in G from u_1 to u_{k+1}, $u_1 - u_{k+1}$ path:

- graph $(\{u_1, \dots, u_{k+1}\}, \{a_1, \dots, a_k\})$ with $[u_1, a_1, u_2, \dots, u_k, a_k, u_{k+1}]$ walk and $u_i \neq u_j$, $1 \leq i < j \leq k+1$
- e.g. $[v_1, e_1, v_2, e_3, v_3, e_4, v_4]$

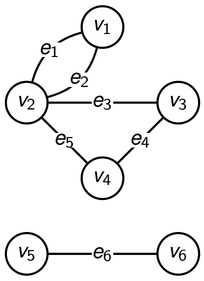


Finally, reachability is a concept in which v is reachable from u if there is a $u - v$ path in G and a graph is **connected** if there is a $u - v$ path in G for all $u, v \in V(G)$.



CHAPTER 5
Algorithms

For testing connective, the more straightforward approach is a visual representation is available, such as, for example:

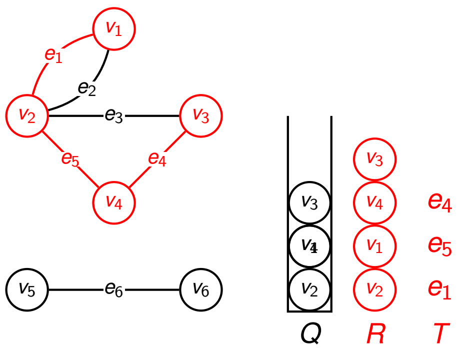


If visual tools are not available, there are a few usual computational representations:

incidence matrix	**adjacency matrix**	**adjacency list**
$A \in \{0,1\}^{\lvert V\rvert \times \lvert E\rvert}$, $a_{v,e} = \begin{cases} 1, & \text{if } v \in e \\ 0, & \text{if } v \notin e \end{cases}$	$A \in \mathbb{Z}^{\lvert V\rvert \times \lvert V\rvert}$, $a_{v,w} = \lvert\{e = \{v,w\} \in E\}\rvert$	$L = [\ell(v) \colon v \in V]$, $\ell(v) = [e \colon e = \{u,v\} \in E]$
$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 2 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$	$\begin{array}{c} \ell(v_1) = [e_1, e_2] \\ \ell(v_2) = [e_1, e_2, e_3, e_5] \\ \ell(v_3) = [e_3, e_4] \\ \ell(v_4) = [e_4, e_5] \\ \ell(v_5) = [e_6] \\ \ell(v_6) = [e_6] \end{array}$
$O(\lvert V\rvert \lvert E\rvert)$	$O(\lvert V\rvert^2)$	$O(\lvert E\rvert \log \lvert V\rvert)$

The easiest algorithm to verify connectivity is via **DFS** (**Depth First Search**):

```
Algorithm: DEPTH FIRST SEARCH (DFS)
Input: undirected graph $G$, vertex $s \in V(G)$
Output: tree $(R, T) \subseteq G$, $R$ reachable from $s$
1 set $R := \{s\}$, $Q := \{s\}$ and $T = \emptyset$;
2 if $Q = \emptyset$ then return $R, T$;
3 else $v :=$ last vertex added to $Q$;
4 choose $w \in V(G) \setminus R$ with $\{v, w\} \in E(G)$;
5 if there is no such $w$ then
6     set $Q := Q \setminus \{v\}$ and go to 2
7 set $R := R \cup \{w\}$, $Q := Q \cup \{w\}$, $T := T \cup \{\{v, w\}\}$, go to 2;
```



The concept of the algorithm is as follows:

- suppose $w \in V(G) \setminus R$ is **reachable** from s

$\Rightarrow$ P is $s - w$ path with $\{x, y\} \in E(P)$, $x \in R$, $y \in V(G) \setminus R$

$\Rightarrow$ x is added to Q in line 7

$\Rightarrow$ Algorithm does not stop before x is removed from Q (line 6)

$\Rightarrow$ there is no $w \in V(G) \setminus R$ with $\{v, w\} \in E(G)$ ↯

In terms of runtime, for each node, the incident edges are considered; therefore, the runtime depends on the storage of graphs. If *adjacency lists* are used, the runtime is $O(m) = O(|E(G)|)$.

Analogous to DFS, there is also **BFS** (**Breadth First Search**) with the following algorithm:

```
Algorithm: BREADTH FIRST SEARCH (BFS)
Input: undirected graph $G$, vertex $s \in V(G)$
Output: tree $T) \subseteq G$
1 set $Q := \{s\}$ and $T = \{s\}$;
2 while $Q \neq \emptyset$ do
3     $v :=$ first vertex in $Q$
4     set $Q := Q \setminus \{v\}$
5     while $v$ has a neighbour not in $T$ do
6         $w :=$ first neightbour of $v$ not in $T$
7         set $Q := Q \cup \{w\}$
8         set $T := T \cup \{\{v, w\}\}$
```