Lecture I - Welcome to Combinatorics

[1] Department of Mathematics and Systems Analysis,
Systems Analysis Laboratory, Aalto University, Finland

January 8, 2024

Aalto University

Definitions

Definition

Combinatorial:

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

Optimization (or Optimisation):

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

Definition

Can be achieved by:

- Analysing/Visualizing properties of functions / extreme points or
- Applying numerical methods

Optimisation has important applications in fields such as

- **operations research (OR)**;
- economics;
- statistics;
- bioinformatics;
- machine learning and artificial intelligence.

What is optimisation?

In this course, optimisation is viewed as the core element of mathematical programming.

Math. programming is a central OR modelling paradigm:

- **variables** → decisions/point of interest: business decisions, parameter definitions, settings, geometries, ...;
- **domain** → constraints/limitations: logic, design, engineering, ...;
- **function** → objective function/profit: measurement of (decision) quality.

However, math. programming has many applications in fields other than OR, which causes some confusion;

We will study math. programming in its most general form: both constraints and objectives can nonlinear or linearfunctions, but the domain is discrete.

Types of programming

Aalto University

Combinatorial Optimization

Rule of Thumb:

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

Some useful notation:

- $x \in \mathbb{R}^n$: vector of (decision) variables x_j, $j = 1, \ldots, n$;
- $f : \mathbb{R}^n \to \mathbb{R} \cup \{\pm\infty\}$ - 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 $i = 1, \ldots, m$: inequality constraints;
- $h_i(x) = 0$ for $i = 1, \ldots, l$: equality constraints.

Types of programming

Aalto University

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

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

- **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 $a_i \in \mathbb{R}^n$, $b \in \mathbb{R})$; $X = \{x \in \mathbb{R}^n : x_j \ge 0, j = 1,\dots,n\}$.
- **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.

Course Structure

Structure

- Classes: Lecture and Exercise Session
  $\rightarrow$ attendance is not mandatory;
  $\rightarrow$ attendance is mandatory in **Guest Lecture**;

- Materials: Lectures Notes and Slides
  $\rightarrow$ available on "MyCourse";

Assessment

- 1 project (group or individual): 60%;
  $\longrightarrow$ based on course problems;
- 3 assignments (individual): 45% (15 each).
  $\longrightarrow$ unique problem.
- Guest Lecture.

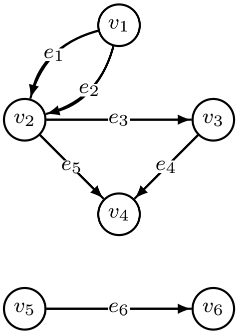
Mixture of **modelling**, **analytical mathematics** and **programming**.

Programming Language: **Python** and **Julia**

Graphs

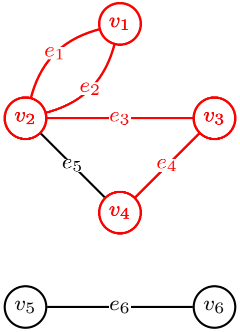
Simple definitions I

- (undirected) graph $G = (V, E, \Psi)$
  - vertices V
  - edges E
  - function $\Psi \colon E \to \{X \subseteq V : |X| = 2\}$
- directed graph $G = (V, E, \Psi)$
  - vertices V
  - edges E
  - function $\Psi \colon E \to \{(v, w) \in V \times V : v \neq w\}$
- Edges e can have a value associated to it:
  $\to w \longrightarrow f_{uv}$
  called **weight** or **flow**;
- in practice: $e = \{u, v\}$, $e = (u, v)$
  respectively, $G = (V, E)$
  ❦ E can contain multiple parallel edges



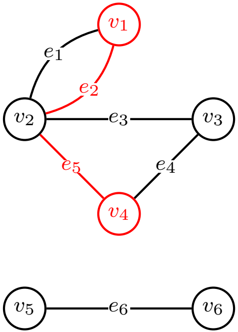
Simple definitions II

- edge progression W in G from u_1 to u_{k+1}:
  - 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. $[v_3, e_3, v_2, e_2, v_1, e_1, v_2, e_3, v_3, e_4, v_4]$
- walk W in G from u_1 to u_{k+1}:
  - edge progression with $a_i \neq a_j$, $1 \leq i < j \leq k$
  - e.g. $[v_2, e_2, v_1, e_1, v_2, e_3, v_3, e_4, v_4]$
- 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. $[v_1, e_1, v_2, e_3, v_3, e_4, v_4]$



Simple definitions III

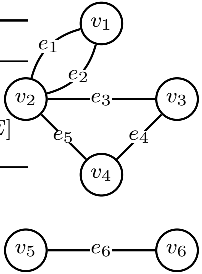
- v reachable from u if there is a $u - v$ path in G
- connected iff there is a $u - v$ path in G for all $u, v \in V(G)$



Testing connectivity

- decide if G is connected
- obvious if we have a graphical representation
- not so obvious if we only have sets $V = V(G)$, $E = E(G)$

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



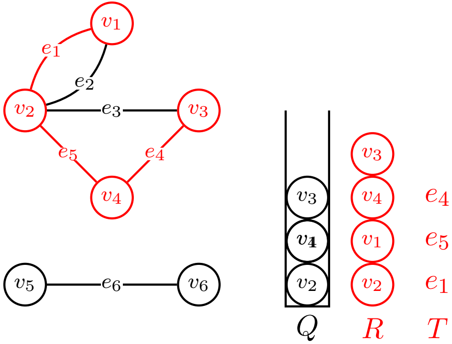
Testing connectivity – An algorithm

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;



Aalto University

Combinatorial Optimization

Correctness

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 = ∅;
2 if Q = ∅ then return R, T;
3 else v := last vertex added to Q;
4 choose w ∈ V(G) \ R with {v, w} ∈ E(G);
5 if there is no such w then
6     set Q := Q \ {v} and go to 2
7 set R := R ∪ {w}, Q := Q ∪ {w},
    T := T ∪ {{v, w}}, go to 2;
```

Idea

- 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 $\{v, w\} \in E(G)$ ↯

Running time

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;

- for each node the incident edges are considered
- runtime depends on the storage of graph
- if *adjacency lists* are used, the runtime is $O(m) = O(|E(G)|)$

Breadth-First Search

Algorithm: Breadth First Search (BFS)

Input: undirected graph G, vertex $s \in V(G)$

Output: tree $T) \subseteq G$

set $Q := \{s\}$ and $T = \{s\}$;
while $Q \neq \emptyset$ **do**
    $v :=$ first vertex in Q
    set $Q := Q \setminus \{v\}$
    while v *has a neighbour not in* T **do**
        $w :=$ first neighbour of v not in T
        set $Q := Q \cup \{w\}$
        set $T := T \cup \{\{v, w\}\}$

Combinatorial Optimization