Aalto University

Lecture Notes - Part I

Main Takeaways

Fernando Dias, Philine Schiewe and Piyalee Pattanaki

February 4, 2024

CHAPTER 1
Definitions

1.1 INTRODUCTION

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.

1.2 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, $G = (V, E)$

1.3 GRAPH REPRESENTATION

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

CHAPTER 2
Paths and Trees

2.1 SHORTEST PATH

Challenge: finding the **minimum-cost path** from a node to all the other in a **weighted** graph.

Definition 1 (Flow network) *A tuple* $G = (V, E, f)$ *is said to be a* flow network *if* (V, E) *where for every edge* $(u, v) \in E$ *we have an associated positive integer* flow value f_{uv}.

Alternative: Dijkstra's algorithm.

2.2 MINIMAL SPANNING TREES

Instance: An undirected, connected graph G, weights $c\colon E(G) \to \mathbb{R}$.

Task: Find a spanning tree T in G of minimum weight.

This problem has been studied to extension, and **two algorithms** have been proposed from the literature. The first option (in no particular order) is **Kruskal's algorithm** (proposed by Joseph Kruskal in 1956). An alternative is Prim's algorithm (developed in 1930 by Czech mathematician **Vojtěch Jarník** and later re-discovered and republished by computer scientists **Robert C. Prim** in 1957 and **Edsger W. Dijkstra** in 1959).

CHAPTER 3

Flows and Cuts

In **graph theory**, **flow network** is a **directed** graph $G = (V, E)$ where each edge has a **capacity** $u\colon E \to \mathbb{R}_+$ and each edge receives a **flow** $f\colon E \to \mathbb{R}_+$, where the amount of flow allowed in each edge cannot surpass its capacity ($f(e) \leq u(e)$, $e \in E$). Hence, the *excess* of a flow f at $v \in V$:

$$\mathrm{ex}_f(v) := \sum_{e \in \delta^-(v)} f(e) - \sum_{e \in \delta^+(v)} f(e)$$

$\delta^-(v) = \{e \in E\colon e = (u, v)\}$ incoming edges

$\delta^+(v) = \{e \in E\colon e = (v, u)\}$ outgoing edges

The flow in this type of graph also have the satisfy **flow conservation** which state that:

Definition 2 *The total net flow entering a node v is zero for **all nodes** in the network except the source s and sink t.*

CHAPTER 4
Matching

As always, a definition at first:

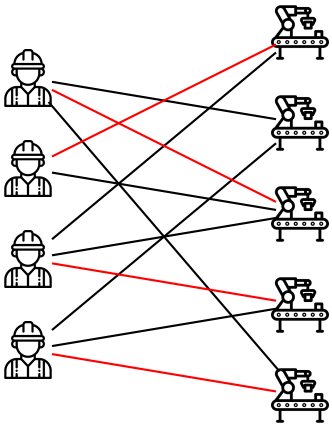
Definition 3 ***Matching*** *in an undirected graph is a set of edges without common vertices.*

Also known as **independent edge set**, this problem goal is to find a subset of the edges as a matching if each node appears in at most one edge of that matching.

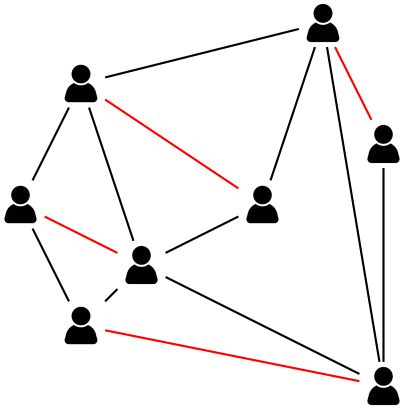
From an undirected graph $G = (V, E)$, $M \subset E$ is called *matching* if all $e \in M$ are pairwise disjoint, i.e., if the endpoints are different. In addition, $M \subset E$ is a *maximum matching* in G if M is a matching with highest cardinality, i.e.,

$$|M'| \leq |M| \quad \text{for all matchings } M'$$

Some illustrations as example:



Assignment different workers to different tasks in order that there is no conflict or overlapping.



Setting pairs for homework assignments.

For this problem, a simple integer linear programming formulation can be calculated:

$$\begin{aligned}
&\text{Maximize} && \sum_{e \in E} x_e && \\
&\text{Subject to:} && && \\
&\sum_{e \in \delta(v)} x_e \leq 1 && && \forall v \in V \\
&x_{ij} \in \{0, 1\} && && \forall e \in E
\end{aligned}$$

where $\delta(v)$ is the set of incident edges of $v \in V$, such that:

$$\delta(v) = \{e \in E : e = \{v, w\}\}$$

CHAPTER 5
Algorithms

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;

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 $T := T \cup \{\{v, w\}\}$

Algorithm: DIJKSTRA'S ALGORITHM

Input: undirected, connected graph G, weights $c\colon E(G) \to \mathbb{R}$, nodes V, source s

```
1  d_v distance to reach node v
2  p_v node predecessor to node v
3  Q ← ∅ set of "unkown distance" nodes.
4  for each node v in V do
5      d_v ← ∞
6      p_v ← FALSE
7      add v in Q
8  d_s ← 0
9  while Q ≠ ∅ do
10     u ← node in Q with min d_u
11     remove u from Q
12     for each neighbor v of u still in Q do
13         d ← d_u + c_uv
14         if alt < d_v then
15             d_v ← alt
16             p_v ← u
17 return d_v,p_v
```

Algorithm: KRUSKAL'S ALGORITHM

Input: undirected, connected graph G, weights $c\colon E(G) \to \mathbb{R}$

Output: spanning tree T of minimum weight

```
1  sort edges such that c(e_1) ≤ c(e_2) ≤ ... ≤ c(e_m)
2  set T := (V(G), ∅)
3  for i := 1 to m do
4      if T + e_i contains no cycle then
5          set T := T + e_i
6  return T
```

Algorithm: PRIM'S ALGORITHM

Input: undirected, connected graph G, weights $c\colon E(G) \to \mathbb{R}$

Output: spanning tree T of minimum weight

```
1  choose v ∈ V(G)
2  set T := ({v}, ∅)
3  while V(T) ≠ V(G) do
4      choose an edge e ∈ δ_G(V(T)) of minimum weight
5      set T := T + e
6  return T
```

Algorithm: FORD-FULKERSON ALGORITHM

Input: digraph $G = (V, E)$, capacities $u\colon E \to \mathbb{Z}_+$, $s, t, \in V$
Output: maximal s-t-flow f

set $f(e) = 0$ for all $e \in E$
while *there exists f-augmenting path in G_f* **do**
    choose f-augmenting path P
    set $\blacksquare_f(P) = \min_{a \in E(P)} u_f(a)$
    augment f along P by $\blacksquare_f(P)$
    update G_f
return f

Algorithm: MAXIMUM MATCHING

Input: undirected graph $G = (V, E)$
Output: maximum matching M

set $M = \emptyset$
while *there exists M-augmenting path in G* **do**
    choose M-augmenting path P
    set $M = (M \setminus E(P)) \cup (E(P) \setminus M)$
return M

Algorithm: MAXIMUM MATCHING BIPARTITE GRAPHS

Input: undirected bipartite graph $G = (V, E)$
Output: maximum matching M

set $M = \emptyset$
construct G'
while *there exists s-t-path in G'* **do**
    choose s-t-path P
    set $M = (M \setminus E(P)) \cup (E(P) \setminus M)$
    update G'
return M