Graphs

Undirected graph edge: $\{a, b\}$ directed edge: (a, b)

A graph (N, E) is **connected** if u, v are connected, for any $u, v \in N$.

A graph (N, E) is **strongly connected** if u, v are connected by a directed path, for any $u, v \in N$.

Incoming cut into a set of nodes S is the set of edges whose arrow terminates in a node of S coming from a node outside S

outgoing cut from a set of nodes S is the set of edges whose arrow starts from a node of S and terminates into a node outside of S

Graph reachability problem

Given a directed graph $G = (N, A)$ and a node S , determine all the nodes that are reachable from S .

Reachability algorithm

1. Start from a set of nodes (initally the starting node)
2. For each node of the set, see the adjacent nodes and put them in the set
3. If in step 2 I have added no nodes, stop. Otherwise go to step 1

$O(n + m)$ where $n = |N|$ and $m = |A|$

For dense graphs ($m \approx n^2$): $O(n^2)$

A **tree** of a graph is a connected and acyclic sub-graph of that graph.

A **spanning tree** of a graph is a tree of that graph that contains all the nodes of it.

- Every tree with n nodes has $n - 1$ edges.
- Any pair of nodes in a tree is connected via a unique path.
- By adding a new edge to a tree, we create a unique cycle.
- Considering a spanning tree and adding an edge, thus creating a cycle, and then removing another edge of that cycle preserves the properties of the spanning tree (I still have a spanning tree)

Optimal cost spanning tree

Given an undirected graph $G = (N, E)$ and acost function, find a spanning tree $G_T = (N, E)$ of minimum total cost.

Theorem 1 (Cayley): a complete graph with $n \geq 1$ nodes has n^{n-2} spanning trees

Prim's algorithm (greedy)

1. Start from a set of nodes composing the spanning tree (initially the start node) and look at all adjacent nodes
2. Add to the set of nodes the adjacent node connected to the set with minimum cost
3. Repeat from step 1 until all nodes are reached

If all edges are scanned at each iteration, the complexity order is: $O(nm)$ where n is the number of nodes and m is the number of edges.

It is possible to improve Prim's algorithm to $O(n^2)$ by keeping at each iteration a an array of indexes of the closest nodes to each node. For sparse graphs it is possible to reach a complexity of $O(m \log n)$.

Given a spanning tree, an edge which is not part of the tree is said to be **cost-decreasing** if, added to the tree and removed another edge, creates a spanning tree of less cost.

A tree T is of minimum total cost if and only if no cost-decreasing edge exists.

Optimality test: to verify if a spanning tree is optimal, it suffices to verify that no cost-decreasing edges exists. Ths complexity of this test is $O(n^2)$ In the worst case.

Shortest path problem

Given a directed graph $G = (N, A)$ with acost $c_{ij} \in R$ for each arc $(i, j) \in A$, and two nodes s and t , determine a minimum cost (shortest) path from s to t .

A path is **simple** if no node is visited more than once.

Dijkstra's algorithm

Output: shortest paths from s to all other nodes of G

Two labels are associated with each node:

- L_j cost of the shortest path from s to node j
- $pred_j$ predecessor of node j in the shortest path from s to node

$$j$$

1. Start from a set, initially containing only node s
2. Consider all the nodes adjacent to the set and compute and assign them L_j and a predecessor
    a. To compute L_j, use the L_j of the nearest node in the set and sum the cost between the two nodes. If the new computed L_j is smaller than the previous one, keep the smallest.
    b. Predecessor is the node used to compute the current L_j
3. Add the adjacent nodes to the set and repeat from step 2
4. When all nodes are in the spanning tree, stop

The complexity is $O(mn)$, hence for dense graphs of $O(n^3)$. Depending on how nodes are scanned, we can reach $O(n^2)$. n is the number of nodes and m is the number of edges.

Dijkstra's algorithm does not work when there are arcs with negative cost.

Floyd-Warshall's algorithm

To detect the presence of circuits with negative costs. It provides a set of shortest paths between all pairs of nodes, even when there are arcs with negative cost.

Data structure: Matrix P of predecessors of j in the shortest path from i to j , Matrix D of the costs of the shortest path from i to j

Triangluar operation: for a couple of nodes i, j , check whether going from i to j directly is more convenient than passing via another node. $d_{iu} + d_{uj} < d_{ij}$

1. Initialize a matrix D where d_{ij} is the cost of the edge between i and j
2. Initialize a matrix P where p_{ij} is equal to i
3. Use iteratively each node u to make the triangular operation check for all the other nodes, if there is an upgrade in cost somewhere, update the predecessor matrix with node u .
4. If a negative cost circuit is found ($d_{iu} + d_{uj} + d_{ij} < 0$), then stop

The complexity in the worst case is $O(n^3)$, where n is the number of nodes.

Optimal paths in directed acyclic graphs

A **directed acyclic graph (DAG)** is a directed graph containing no circuits.

Given a DAG, find the shortest (longest) path between two nodes s and t .

Topological order: the nodes' indexes are ordered so that, for each arc (i, j) , we have $i < j$

Topological ordering method

At each iteration, take away from the graph a node with no incoming arcs and its outgoing arcs.

The order in which nodes are taken is the topological order.

The complexity is $O(m)$ where $m = |A|$ is the number of edges.

Dynamic Programming for shortest paths in DAGs

For each node, looking at them in topological order: i,…,j,…,n

$L_j = \min_{i=0,\dots,j-1}\{L_i + c_{ij}\}$ then $pred_j = i$ satisfying the minimum

Project planning

A **Project** is a set of m activities with their duration and some precedence constraints.

A project can be represented by a directed graph $G = (N, A)$ where each arc corresponds to an activity and its length is the duration of the activity.

The directed graph G representing any project is acyclic (it is a DAG).

Problem: given a project, schedule the activities to minimize the overall project duration

The minimum overall project duration = length of a longest path from s (start) to t (termination)

Critical Path Method (CPM)

1. Construct the graph G representing the project and find a topological order of the nodes.
2. For each node, going in topological order, determine the T_{min} at which it can be reached (the event associated to it can happen)
$$T_{min,\,j} = \max\{T_{min,\,i} + d_{ij} : (i,j)\epsilon\, \delta^-(j)\}$$
3. For each node, going in inverse topological order, determine theT_{max}as the latest time at which the event associated to that node can occur without delaying the project completion date beyond $T_{min,\,n}$(the T_{min} of the last node)
$$T_{max,\,i} = \min\{T_{max,\,j} - d_{ij} : (i,j)\epsilon\, \delta^+(i)\}$$
4. For each activity (i, j) find the **slack**: $\sigma_{ij} = T_{max,\,j} - T_{min,\,i} - d_{ij}$

An activity with zero slack is **critical**. A path from start to end composed of only critical activities is a **critical path**.

Gantt chart at earlies: each activity (i, j) starts at $T_{min,\,i}$

Gantt chart at latest: each activity (i, j) ends at $T_{max,\,j}$

The complexity of the CPM is $O(m + n)$ where n is the number of nodes and m is the number of edges.

A **network** is a directed and connected graph $G = (V, A)$ with a source $s \in V$ and a sink $t \in V$, with $s \neq t$, and a capacity $k_{ij} \geq 0$, for each arc $(i, j) \in A$

Amount entering a node = amount exiting a node

A **feasible flow** is an assignment of resources on the arcs on the graph traveling from s to t

The **value of flow** is the sum of all the resources exiting from s (or entering in t)

An arc is **saturated** if the load on it is equal to its capacity, **empty** if the load on it is 0.

Maximum flow problem

Given a network $G = (N, A)$ with an integer capacity $k_{ij}$for each arc $(i, j) \in A$, and nodes $s, t \in N$, determine a feasible flow from s to t of maximum value.

A **cut** separating s from t is $\delta(S)$ of G with $s \in S \subset N$ and $t \in N \setminus S$. There are 2^{n-2} cuts.

The **capacity of the cut** is the sum of the capacities of all the arcs outgoing from the cut.

Given a feasible flow $\bar{x}$ from s to t and a cut $\delta(S)$ separating s from t, the value of the feasible flow $\bar{x}$ through the cut $\delta(S)$ is:

$$\varphi(S) = \sum_{(i,j)\in\delta^+(S)} x_{ij} - \sum_{(i,j)\in\delta^-(S)} x_{ij}$$

It is always true that: $\varphi(S) = \varphi(\{s\})$ and $\varphi(S) \leq k(S)$

If $\varphi(S) = k(S)$ for a subset $S \subseteq N$ with $s \in S$ and $t \notin S$, then $\bar{x}$ is a flow of maximum value and the cut $\delta(S)$ is of minimum capacity. Duality of the probles:

Given $G = (N, A)$ with integer capacities on the arcs and $s, t \in N$, determine a feasible flow of maximum value.	Given $G = (N, A)$ with integer arc capacities and $s, t \in N$, determine a cut (separating s from t) of minimum capacity.

Ford-Fulkerson's algorithm

A path P from s to t is an **augmenting path** w.r.t. the current feasible flow $\bar{x}$ if $x_{ij} < k_{ij}$ for evey forward arc and $x_{ij} > 0$ for every backward arc.

Given a feasible flow $\bar{x}$, it is possible to build the **residual network** $\bar{G} = (V, \bar{A})$ associated to $\bar{x}$:

- Non-empty arc (i, j) becomes an arc (j, i) with $\overline{k_{ji}} = x_{ij}$ (becomes an arc of inverse direction with capacity equal to the load of the initial arc)
- Non-saturated arc (i, j) becomes an arc (i, j) with $\overline{k_{ij}} = k_{ji} - x_{ij} > 0$ (becomes an arc with capacity equal to the residual capacity of the initial arc)

In poor words, for each arc we create an arc going backwards with its load and an arc going upwards with it's residual capacity.

1. Starting from the current feasible flow, build the residual network
2. Find an augmenting path in the residual network
3. If an augmenting path exists, the flow is not of maximum value: add additional units to send on actual graph. If it does not exist, STOP.

Strong duality: The value of a feasible flow of maximum value = the capacity of a cut of minimum capacity. (the cut composed of all nodes reachable from s in the residual network when no augmenting paths exist)

The complexity of Ford-Fulkerson's algorithm is $O(m^2\, k_{max})$ where m is the number of arcs and k_{max} is the maximum capacity among them.

Minimum cost flow problem

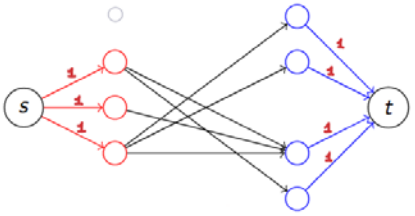
Determine a feasible flow from s to t of value φ and of minimum total cost. (Send at least φ but with minimum cost possible)

Start from a feasible flow $\bar{x}$ of value φ and send, at each iteration, an additional amount of product in the residual network (respecting the residual capacities and the value φ) along cycles of negative cost.

Assignment (matching) problem

Given an undirected bipartite graph $G = (N, E)$, a **matching** $M \subseteq E$ is a subset of non-adjacent edges.

Given a bipartite graph $G = (N, E)$, determine a matching with a maximum number of edges.



$k_{sj} = k_{it} = 1$

Linear Programming

$\max\{f(x): x \in X\} = -\min\{-f(x): x \in X\}$

General form:

$$\begin{aligned} \min \quad & z = c_1x_1 + \cdots + c_nx_n \\ \text{s. t.} \quad & a_{11}x_1 + \cdots + a_{1n}x_n \ (\geq, =, \leq)\ b_1 \\ & \vdots \\ & a_{m1}x_1 + \cdots + a_{mn}x_n \ (\geq, =, \leq)\ b_m \\ & x_1, \ldots, x_n \geq 0 \end{aligned}$$

Matrix notation:

$$\begin{aligned} \min \quad & z = \underline{c}^T\underline{x} \\ \text{s. t.} \quad & A\underline{x} \geq \underline{b} \\ & \underline{x} \geq 0 \end{aligned}$$

$$\min \quad z = [c_1 \ldots c_n] \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

$$\text{s. t.} \quad \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \geq \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \geq \underline{0}$$

Diet problem

Transportation problem

Production planning problem

Assumptions on LP:

- Linearity
- Divisibility (variables can take fractional values)
- Parameters can be estimated with sufficient accuracy

LP **sensitivity analysis** allows to evaluate how sensitive an optimal solution is with respect to small changes in the parameter values.

Standard form

$$\begin{aligned} & \min z = \underline{c}^T\underline{x} \\ & s.t.\ A\underline{x} = \underline{b} \\ & \underline{x} \geq 0 \end{aligned}$$

Transformation rules: (To standard form)

$$\max \underline{c}^T\underline{x} = -\min\left(-\underline{c}^T\underline{x}\right)$$

$\underline{a}^T\underline{x} \leq b \rightarrow \begin{cases} \underline{a}^T\underline{x} + s = b \\ s \geq 0 \end{cases}$ where s is a slack variable

$\underline{a}^T\underline{x} \geq b \rightarrow \begin{cases} \underline{a}^T\underline{x} - s = b \\ s \geq 0 \end{cases}$ where s is a surplus variable

x_j unrestricted in sign $\rightarrow \begin{cases} x_j = x_j^+ - x_j^- \\ x_j^+, x_j^- \geq 0 \end{cases}$ (after substituting x_j with $x_j^+ - x_j^-$, we remove x_j from the problem)

Fundamental theorem of Linear Programming

Consider a LP $min\{\underline{c}^T\underline{x}: \underline{x} \in P\}$, where $P \subseteq R^n$ is a non-empty polyhedron of the feasible solutions (in standard or canonical form). Then either there exists (at least) one optimal vertex or the value of the objective function is unbounded below on P.

Vector of reduced costs (with respect to basis B)

$\bar{\underline{c}}^T = \underline{c}^T - \underline{c_B}^T B^{-1} A$

Because $\bar{\underline{c}}^T$ is 0 for basic variables, in suffices to consider:

$\bar{\underline{c}_N}^T = \underline{c_N}^T - \underline{c_B}^T B^{-1} N$

Component cj of the resulting vector represents the change in the objective function value if non basic xj would be increased from 0 to 1 while keeping all other non-basic variables to 0.

A basic feasible solution x is **degenerate** if it contains at least one basic variable $xj = 0$.

LP Duality

Linear combinations with non-negative multipliers of inequality constraints yield valid upper bounds.

Primal problem	**Dual problem**
$\max z = \underline{c}^T \underline{x}$ $s.t. A\underline{x} \leq \underline{b}$ $\underline{x} \geq 0$	$\min w = \underline{b}^T \underline{y}$ $s.t. A^T \underline{y} \geq \underline{c}$ $\underline{y} \geq 0$

Transformation rules

Primal problem	**Dual problem**
Minimization	**Maximization**
m constraints	m variables
n variables	n constraints
Coefficients of the objective function	Right hand side of constraints
Right hand side of constraints	Coefficients of the objective function
A	A^T
Equality constraints	Unrestricted variables
Unrestricted variables	Equality constraints
Inequality constraints $\geq (\leq)$	Variables ≥ 0 (≤ 0)
Variables ≥ 0 (≤ 0)	Inequality constraints $\leq (\geq)$

Weak duality theorem

Given a primal problem P and his dual D:

$$\begin{array}{c|c} \min z = \underline{c}^T \underline{x} \\ s.t. A\underline{x} \geq \underline{b} \\ \underline{x} \geq 0 \end{array} \qquad \begin{array}{c} \max w = \underline{b}^T \underline{y} \\ s.t. A^T \underline{y} \leq \underline{c} \\ \underline{y} \geq 0 \end{array}$$

For every feasible solution $\underline{x}$ of P and every feasible solution $\underline{y}$ of D we have:

$$\underline{b}^T \underline{y} \leq \underline{c}^T \underline{x}$$

Consequence:

If x is a feasible solution of (P) $(x \in X)$, y is a feasible solution of (D) $(y \in Y)$, and the values of the respective objective functions coincide, $\underline{c}^T \underline{x} = \underline{b}^T \underline{y}$, then x is optimal for (P) and y is optimal for (D).

Strong duality theorem

If the feasible region X is non empty, and $\min\{\underline{c}^T \underline{x} : x \in X\}$ is finite, there exist $\underline{x}^* \in X$ and $\underline{y}^* \in Y$ such that

$$\underline{c}^T \underline{x}^* = \underline{b}^T \underline{y}^*$$

That is $\min\{\underline{c}^T \underline{x} : x \in X\} = \max\{\underline{b}^T \underline{y} : y \in Y\}$

For any pair of primal-dual problems P and D, only four cases can arise

P \ D	∃ optimal solution	Unbounded LP	Infeasible LP
∃ optimal solution	Strong duality: $\underline{c}^T\underline{x}^* = \underline{b}^T\underline{y}^*$	Strong duality: $\underline{c}^T\underline{x}^* = \underline{b}^T\underline{y}^*$	Strong duality: $\underline{c}^T\underline{x}^* = \underline{b}^T\underline{y}^*$
Unbounded LP	Strong duality: $\underline{c}^T\underline{x}^* = \underline{b}^T\underline{y}^*$	Weak duality: $\underline{b}^T\underline{y} \leq \underline{c}^T\underline{x}$	Weak duality: $\underline{b}^T\underline{y} \leq \underline{c}^T\underline{x}$
Infeasible LP	Strong duality: $\underline{c}^T\underline{x}^* = \underline{b}^T\underline{y}^*$	Weak duality: $\underline{b}^T\underline{y} \leq \underline{c}^T\underline{x}$	Both P and D are infeasible

Optimality conditions:

$$\begin{cases} \underline{y}^{*T}\left(A\underline{x}^* - \underline{b}\right) = 0 \\ \left(\underline{c}^T - \underline{y}^{*T}A\right)\underline{x}^* = 0 \end{cases}$$

Complementary slackness conditions:

$x* \in X$ and $y* \in Y$ are optimal solutions of, respectively, (P) and (D) if and only if:

$$y_i^*\left(\underline{a_i}^T\underline{x}^* - b_i\right) = 0 \qquad i = 1, \dots, m$$

$$\left(c_j{}^T - \underline{y}^{*T}A_j\right)x_j^* = 0 \qquad j = 1, \dots, n$$

Where $\underline{a_i}$ denotes the i-th row of A and A_j denotes the j-th column of A.

NOTE:

$\left(\underline{a_i}^T\underline{x}^* - b_i\right)$ is the slack of the $i-th$ constraint of P

$\left(c_j{}^T - \underline{y}^{*T}A_j\right)$ is the slack of the $j-th$ constraint of D

At optimality, the product of each variable with the corresponding slack variable of the constraint of the relative dual is $= 0$.

<u>Sensitivity analysis</u>

Conditions must hold with the variations:

Feasibility: $B^{-1}\underline{b} \geq \underline{0}$

Optimality: $\underline{\bar{c}_N}^T = \underline{c_N}^T - \underline{c_B}^T B^{-1}N \geq \underline{0}$

Possible variations:

- Variation of the right-hand side terms: $\underline{b}' \leftarrow \underline{b} + \delta_k\underline{e_k}$ check Feasibility
- Variation of the cost coefficients: $\underline{c}' \leftarrow \underline{c} + \delta_k\underline{e_k}$ check Optimality
  - Non-basic variables: $\delta_k \geq -\bar{c}_k$
  - Basic variables: $\underline{\bar{c}_N}^T \geq \delta_k\underline{e_k}^T B^{-1}N$

Where $\underline{e_k}$ is a vector of 0s with 1 in the k-th position.

Integer Linear Programming (ILP)

The **linear relaxation** of ILP problem is the same problem, but without the integrality constraint.

For any ILP with max, we have $z_{ILP} \leq z_{LP}$,i.e., z_{LP} is an upper bound on the optimal value of (ILP). On the contrary, for any ILP with min, we have $z_{ILP} \geq z_{LP}$,i.e., z_{LP} is a lower bound on the optimal value of (ILP).

If an optimal solution of (LP) is integer, then it is also an optimal solution of (ILP).

Type of methods:

- implicit enumeration: exact methods (global optimum)
  - Branch-and-bound method
  - Dynamic programming
- cutting planes: exact methods (global optimum)
- heuristic algorithms (greedy, local search, …): approximate methods (local optimum)

The **ideal formulation** is that describing the convex hull of X, $conv(X)$, where $conv(X)$ is the smallest convex subset containing X. For any feasible region X of a ILP, there exists an ideal formulation.

A cutting plane is an inequality $\underline{a}^T \underline{x} \leq \underline{b}$ that is not satisfied by ${x^*}_{LP}$ but is satisfied by all the feasible solutions of the ILP.

The **Gomory cut** w.r.t. the fractional basic variable $x_{B[r]}$ is: $\sum_{j \in N}(\bar{a}_{rj} - \lfloor \bar{a}_{rj} \rfloor)\, x_j \geq (\bar{b}_r - \lfloor \bar{b}_r \rfloor)$

Integer form	$x_{B[r]} + \sum_{j \in N} \lfloor \bar{a}_{rj} \rfloor\, x_j \geq \lfloor \bar{b}_r \rfloor$
Fractional form	$\sum_{j \in N} (\bar{a}_{rj} - \lfloor \bar{a}_{rj} \rfloor)\, x_j \geq (\bar{b}_r - \lfloor \bar{b}_r \rfloor)$

If the ILP has a finite optimal solution, the cutting plane method finds one after adding a finite number of Gomory cuts.

Laboratory

```
# import the package mip
import mip

#Define sets and variables
I = {'Bread','Milk','Eggs','Meat','Cake'}
J = {'Calories','Proteins','Calcium'}
K = range (4)
c = {'Bread':0.1,'Milk':0.5,'Eggs':0.12,'Meat':0.9,'Cake':1.3}        #DICTIONARY
c = [0.1, 0.5, 0.12, 0.9, 1.3]  #LIST
a = {('Bread','Calories'):30, ('Milk','Calories'):50, ('Eggs','Calories'):150, ('Meat','Calories'):180, ('Cake','Calories'):400,
('Bread','Proteins'):5, ('Milk','Proteins'):15, ('Eggs','Proteins'):30, ('Meat','Proteins'):90, ('Cake','Proteins'):70,
('Bread','Calcium'):0.02, ('Milk','Calcium'):0.15, ('Eggs','Calcium'):0.05, ('Meat','Calcium'):0.08, ('Cake','Calcium'):0.01}
a = numpy.array( [ [1,0,1,0,1,0,1,1,0,0,0,0],
    [0,1,1,0,0,1,0,0,1,0,1,1],
    [1,1,0,0,0,1,1,0,0,1,0,0],
    [0,1,0,1,0,0,1,1,0,1,0,1],
    [0,0,0,1,1,1,0,0,1,1,1,1] ] )
# Define a model
model = mip.Model()

# Define variables
x = [model.add_var(name = i, var_type=INTEGER, lb=0) for i in I]     #lb is the lower bound to the value of the variable
x = {(str(i),j): model.add_var(name = str(i) + ',' + j, lb=0) for i in I for j in J}
y = {j: model.add_var(name = j, var_type=BINARY, lb=0) for j in J}

# Define the objective function
model.objective = mip.minimize ( mip.xsum (x[i]*c[food] for i, food in enumerate(I)) )
model.objective = mip.minimize ( mip.xsum (x[i]*c[i] for I in range(5)) )
model.objective = mip.maximize( mix.xsum(y[j]*r[j] for j in J) - mip.xsum(x[str(i), j]*c[i] for i in I for j in J) )

# Define constraints
for i,food in enumerate(I):
        model.add_constr (x[i]<= q[food])
```

```
for j in J:
        model.add_constr( mip.xsum(x[i]*a[food,j] for i,food in enumerate(I)) >= b[j] )

for j in J:
        model.add_constr( mip.xsum(x[i]*a[I[i],nut] for i in range(5)) >= b[j] )

for i in I:
        for j in J:
                model.add_constr( x[str(i),j] >= q_min[str(i),j]*y[j])
```

 # Optimizing command

```
model.optimize()
```

 # Optimal objective function value

```
model.objective.x
```

 # Printing the variables values

```
for i in model.vars:
  print(i.name)
  print(i.x)
```

#sensitivity analysis

```
print("Dual value:", model.constrs[NUM].pi)
```

where NUM is the number of the constraint, corresponding to the (NUM-1)th time we called “add_constr”