Foundations of Operations Research

Francesco Ostidich

November 9, 2024

Contents

1 Introduction

1.1 Decision making problems

Decision making problems can be arranged in various categories:

- **matematical programming** has a single decision maker and a single objective;
- **multi-objective programming** has instead several objectives;
- **stochastic programming** corresponds to problems where there is a uncertainty level;
- **game theory** fits well when there are multiple decision makers.

1.1.1 Mathematical programming

Mathematical programming opts to minimize a decision variable $x \in \mathbb{R}^n$ inside a feasible region $X \subseteq \mathbb{R}^n$ via a objective function $f : x \to \mathbb{R}$.

MP is categorized in three special cases.

$$X_0 = \left\{ x \in \mathbb{R}^n : g_i(x) \lesseqgtr 0, i = 1, 2, 3, \ldots, m \right\}$$

- **Linear programming** (LP): $X = X_0$, with f and g linear
- **Integer linear programming** (ILP): $X = X_0 \cap \mathbb{Z}^n$, with f and g linear
- **Non linear programming** (NLP): $X = X_0$, with f and g non linear

2 Graph and network optimization

2.1 Graph reachability problem

Given a graph $G = (N, A)$ and a node s, find all nodes that are reachable from s.

Efficient algorithms use queues to keep track of previously expanded nodes; different objectives are pursued based on the queue type, each with its complexities.

2.2 Minimum spanning tree

Given a graph $G = (N, E)$, with each edge having a cost $c_e \in \mathbb{R}$, find a spanning tree $G_T = (N, T)$ with minimum total cost. Being X the set of all the spanning trees of G, the objective is to find

$$\min_{T \in X} \sum_{e \in T} c_e$$

that may not be a easy task being the number of spanning trees 2^{n-2}, with n the number of nodes of the graph.

2.2.1 Kruskal method

Firstly, the arcs list is ordered from the least costly to the most costly. Then, one arc after the other, they are popped and added to the graph. If adding an arc would create a cycle, it is skipped. In the end, what does remain is the minimum spanning tree.

2.2.2 Prim's algorithm

The spanning tree is built iteratively, starting from an initial tree (S, T), where $S = u$ and $T = \emptyset$.

At each iteration, the edge with the lower cost (among the unchosen edges of the cut S) is added to T, and the node it connects to S.

The complexity of the algorithm is $O(nm)$, being n the number of nodes and m the number of edges.

This algorithm is exact, since it always provides an optimal solution. It is also greedy, since it look for the local minima iteratively, without reconsidering previous choices.

A variance of the Prim's algorithm with complexity $O(n^2)$ can be achieved by associating each node with another. Each node i is assigned with another node j, that is either the predecessor (if $i \in S$), or the one in the spanning tree with which it has minimum cost ($\text{argmin}_{i,j} c_{ij}$).

This way, with each iteration, it is faster to choose the node with minimum cost that connects to the newly expanded node.

2.3 Dijkstra's algorithm

Dijkstra's algorithm can be used to find the shortest path from a node to all other nodes of the graph.

Each node is assigned a label storing the cost of the current shortest path found, along with the predecessor in that path. At each iteration nodes in the frontier are expanded, and if the new path for the neighboring is shorter than their previous, the labels of the are updated.

If all m arcs are scanned the complexity is $O(mn)$, but if labels are appropriately updated complexity becomes $O(n^2)$. Dijkstra's algorithm is exact, meaning it always provides an optimal solution. It does not work with negative cost arcs.

The tree made from the set of short paths for a node s is called the arborescence rooted at s.

2.4 Floyd-Warshall algorithm

This algorithm can be used when the graph contains paths with negative costs.

The graph is to be described via a matrix D, in which each cell is the cost between the row and column nodes. Along with D, define a matrix P which, in the end, p_{ij} is the predecessor of j in the shortest path from i to j.

$$D = \begin{bmatrix} 0 & 2 & \infty & 1 \\ \infty & 0 & 3 & 3 \\ \infty & 0 & 2 & \infty \\ \infty & -5 & 5 & \infty \end{bmatrix} \qquad P = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{bmatrix}$$

Then, for each $u = 1, \ldots, n$, if $d_{iu} + d_{uj} < d_{ij}$, then $d_{ij} = d_{iu} + d_{uj}$. If the path cost is updated, then the corresponding element of the P matrix is updated to u.

The Floyd-Warshall algorithm is exact, with complexity $O(n^3)$, being n the number of nodes.

2.5 Topological ordering method

A DAG (directed acyclic graph) can be rearranged so that, for every path (i, j), an order $i < j$ is defined.

Each "layer" of the topological ordered graph is found iteratively, starting from a node which has no incoming edge. With each iteration, fill the following layer with the nodes directly reachable from the current.

Complexity is $O(n + m)$, since each node and each arc is considered once.

2.6 Dynamic programming

Dynamic programming is a technique for which an optimal solution is found by resolving a set of recursive equations. It can be applied to decision problems that satisfy the optimality principle, stating that if a path from node 1 to j is the shortest, than the path from 1 to i, where i is the predecessor of j in the path, is also optimal.

2.6.1 Shorter path in DAGs

Let L_n be the cost of the path between node 1 and n. If the objective is to find the minimum-cost path from 1 to i, then the problem becomes to minimize the value of

$$L_j + c_{ji}$$

where j is the predecessor of i and c_{ji} is the cost between j and i.

For example, the set of recursive equations that describe the problem may be

$$\begin{aligned}
&\text{pred}_i = 1, \quad L_1 = 0 \\
&\text{pred}_i = 1, \quad L_2 = L_1 + c_{12} = 4 \\
&\text{pred}_i = 1, \quad L_3 = L_1 + c_{13} = 2 \\
&\text{pred}_i = 3, \quad L_4 = \min_{i=1,2,3}\{L_i + c_{i4}\} = \min\{0+5, 4+2, 2+1\} = 3
\end{aligned}$$

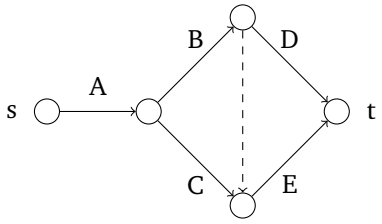
where, with each iteration over the formulae, the cost and predecessor of each node is updated until the formula for the final node has no dependencies left over.

2.7 Project planning

A project is a set of activities each with its duration.

Activities may have a precedence constraint, for example $A_i \propto A_j$. In those cases, the starting node of A_j must appear after the ending node of A_i.

Between activity arches, dummy arcs can be added to symbolize dependencies. Artificial nodes and arcs may also be added in order to obtain a single initial and final node.



2.7.1 Critical path method

The CPM is used to determine a schedule, that is the set of activities that minimizes the total project duration, and the slack of each activity, that is maximum time an activity can be delayed.

After constructing the graph of activities in topological order, find

1. the earliest time T_{min} for each node with increasing indices,
2. the latest time T_{max} for each node with decreasing indices,
3. the slack for each activity $\sigma_{ij} = T_{max,j} - T_{min,i} - d_{ij}$.

A critical path is one that has slack equal to zero.

2.8 Network flow

A network is a directed and connected graph, that contains both a source (a node with only outgoing arcs) and a sink (a node with only ingoing arcs).

Each arc of the graph can be assigned with a value in order to produce a feasible flow. This values must satisfy both the capacity constraint, meaning that the value must be less than the arc original cost, and the flow balance constraint, stating that the sum of the ingoing arc values for a node must be equal to the outgoings.

The value of flow is the sum of outgoing values from the source node. An arc is saturated if the assigned value is the same as the cost, or else it is empty if it is zero.

A cut of the graph has a capacity equal to the sum of the outgoing arcs costs, and given a feasible flow, this has a value through the cut equal to the difference between ingoing and outgoing arc values.

Given a feasible flow, each cut S of the graph has a value ϕ of the flow that is equal to the one of the source node: $\phi(S) = \phi(\{s\})$. Furthermore, the value of the flow is always less than the capacity of the cut: $\phi(S) \leq k(S)$. If $\phi(S) = k(S)$ then the the feasible flow $\underline{x}$ is of maximum value and the cut $\delta(S)$ is of

minimum capacity. From here, two main problem arise: the primal problem consists of finding a feasible flow of maximum value; the dual problem consists of finding a cut of minimum capacity.

2.8.1 Ford-Fulkerson algorithm

The Ford-Fulkerson algorithm is a method for computing the maximum flow in a flow network. It is based on the idea of finding augmenting paths and increasing the flow along these paths until no more augmenting paths can be found.

Firstly, initialize the graph by setting all edge flows to zero. Then, with each iteration, find a path from the source to the sink where the available capacity is greater than the current flow and increase the flow along this path by the smallest available capacity on the path (the bottleneck). After augmenting the flow, update the residual capacities of the edges. This process is to be repeated until no more augmenting path can be found.

This algorithm is non-greedy and exact. The complexity is $O(m^2 k_{max})$, where m is the number of arcs and k_{max} is the biggest capacity among all the arcs. The algorithm can be inefficient in some cases, but when dealing with integer capacities it is possible to reduce complexity to be polynomial using other algorithm implementations.

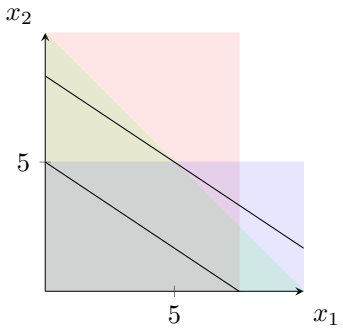
The theorem of strong duality derived from this algorithm tells that the value of a feasible flow of maximum value equals to the capacity of a cut of minimum capacity.

3 Linear programming

3.1 Graphic example

$$\begin{aligned} f(x_1, x_2) &= 0.04x_1 + 0.06x_2 \\ x_1 + x_2 &\leq 10000 \\ x_1 &\leq 7500 \\ x_2 &\leq 5000 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Let $f(x_1, x_2)$ be the function to maximize. The feasible region, in which the variables satisfy the constraints, can be plot graphically.



In the previous chart, the black lines represent the gradient of $f(x_1, x_2) + c$. The maximum value for x_1 and x_2 is found in the vertex of the feasible area where c is at its maximum. Solutions may be unique, if found in a vertex, multiple, if found on a feasible area side, unbounded, if the feasible is such, or infeasible if the polyhedron (feasible area) is empty (infinite number of vertices).

3.1.1 Polyhedron vertices

Solution are always found on a vertex, or at least on a facet, of the polyhedron. The polyhedron is defined by the set of constraint equations: $P = \{\underline{x} \in \mathbb{R}^n : A\underline{x} = \underline{b}, \underline{x} \geq \underline{0}\}$. As an assumption, the matrix $A \in \mathbb{R}^{m \times n}$ is full rank, thus $m \leq n$. The facets, that are the edges found in $\mathbb{R}^2$, are obtained by setting one variable equal to zero. The vertices are obtained by setting $n - m$ variable to zero.

3.2 Simplex method

The objective of the simplex method is to iter over all the vertices until an optimal solution is found. This is done by generating a path along the edges of the polyhedron.

Some initial actions are to be taken.

Constraints that are disequalities are to be transformed into equations, by adding a slack variable.

$$\begin{aligned} x_1 + x_2 &\leq 5 \\ x_1 + x_2 &= 5 - s_1 \\ x_1 + x_2 + s_1 &= 5 \end{aligned}$$

A system $A\underline{x} = \underline{b}$ can be rewritten as $B\underline{x}_B + N\underline{x}_N = \underline{b}$, where B is a feasible basis of A.
Given the problem

$$\begin{cases} z = \underline{c}^T \underline{x} \\ A\underline{x} = \underline{b} \end{cases} \qquad \begin{cases} z = -x_1 - x_2 \\ x_1 - x_2 + s_1 = 1 \\ x_1 + x_2 + s_2 = 3 \end{cases}$$

where all variables are non-negative ($x_1, x_2, s_1, s_2 \geq 0$), the system can be written as

$$A = \begin{bmatrix} 1 & -1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \qquad \underline{b} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \qquad \underline{c} = \begin{bmatrix} -1 \\ -1 \\ 0 \\ 0 \end{bmatrix}$$

where the basis B and the matrix N of the partition $A = \begin{bmatrix} BN \end{bmatrix}$ are chosen to be

$$B = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \qquad N = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix}$$

making the reduced costs $\underline{c}_B$ and $\underline{c}_N$

$$\underline{c}_B = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \qquad \underline{c}_N = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

since each of their row corresponds to the row of $\underline{c}$ equals to the B and N chosen columns of A.
From B^{-1}, a canonical form can be found for the previous components.

$$\overline{\underline{c}}_N^T = \underline{c}_N^T - \underline{c}_B^T B^{-1} N$$

$$\overline{\underline{b}} = B^{-1}\underline{b}$$

$$A\underline{x} = \underline{b} \iff B\underline{x}_B + N\underline{x}_N = \underline{b} \iff x_{B_i} + \sum_{j=1}^{n-m} \overline{a}_{ij} x_{N_j} = \overline{b}_i \quad \forall i = 1, \ldots, m$$

When moving from a vertex to an adjacent one, a column of B is substituted with a column of N.

3.2.1 Pivoting operation

Given the matrix $\begin{bmatrix} \underline{b} & I\underline{s} & A \end{bmatrix}$

1. select the pivot value $\overline{a}_{rs} \neq 0$,
2. divide the r-th row by $\overline{a}_{rs}$,
3. for each row $i \neq r$, subtract the resulting r-th row multiplied by $\overline{a}_{is}$.

3.2.2 Adjacent vertex

The decision of which columns are to be switched in and out of the basis can be done via the Bland's rule.

The non basic variable to enter the basis is $s = \min\{j : \overline{c}_j < 0\}$ ($\overline{c}_j > 0$ for maximization problems). The non basic variable to leave the basis is $r = \min\{i : \overline{b}_i/\overline{a}_{is} = \theta^*\}$, where $\theta^* = \min\{b_i/a_{is} : a_{is} \geq 0\}$.

Essentially, the column j of the pivot is chosen as the first where the z-row is negative, and the row of the pivot is the one where b_i/a_r is minimal but strictly positive.

3.2.3 Tableau algorithm

$$\begin{cases} z = -x_1 - x_2 \\ 6x_1 + 4x_2 + x_3 = 24 \\ 3x_1 - 2x_2 + x_4 = 3 \end{cases}$$

$$B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad\qquad N = \begin{bmatrix} 6 & 4 \\ 3 & -2 \end{bmatrix}$$

The problem consists in finding the minimum of the objective function z (maximum of $-z$). x_3 and x_4 are the basic variables.

The first two column of the basis that have been chosen are the one of x_3 and x_4.

Firsty, build the initial tableau.

		x_1	x_2	x_3	x_4
$-z$	0	-1	-1	0	0
x_3	24	6	4	1	0
x_4	6	3	-2	0	1

After choosing a pivot (3), exchange the row identifier (x_4) with the pivot column one (x_1), and apply the pivoting operation.

		x_1	x_2	x_3	x_4
$-z$	2	0	-5/3	0	1/3
x_3	12	0	8	1	-2
x_1	2	1	-2/3	0	1/3

Continue until the z-row contains only non-negative variables.

		x_1	x_2	x_3	x_4
$-z$	9/2	0	0	5/24	-1/12
x_2	3/2	0	1	1/8	-1/4
x_1	3	1	0	1/12	1/6

		x_1	x_2	x_3	x_4
$-z$	6	1/2	0	1/4	0
x_2	6	3/2	1	1/4	0
x_4	18	6	0	1/2	1

Therefore, the feasible basic solution $\underline{x} = (1/2, 0, 1/4, 0)$, with $z = 6$, is optimal.

A solution may be degenerate if it contains a null basic variable.

3.2.4 Two phase simplex method

Since the standard simplex method starts from the vertex found in the origin, if the origin is not actually a vertex of the feasible area the algorithm won't work.

In this scenario, another iteration of the simplex method is run before hand in order to find a feasible vertex from which to start. The first phase requires the addition of other variables $\underline{y}$ to the initial problem system, and a new objective function $v = \sum_{i=1}^{m} y_i$. In particular, the system $A\underline{x} = \underline{b}$ is expanded to be $A\underline{x} + I\underline{y} = \underline{b}$.

In the following (minimization) problem the objective function is $z = x_1 + x_2 + 10x_3$.

$$A = \begin{bmatrix} 0 & 1 & 4 \\ -2 & 1 & -6 \end{bmatrix} \qquad \underline{b} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

The objective function v must be rewritten as a function of $\underline{x}$ variables.

$$\begin{aligned} v &= y_1 + y_2 \\ y_1 &= -x_2 - 4x_3 + 2 \\ y_2 &= 2x_1 - x_2 + 6x_3 + 2 \\ v &= 2x_1 - 2x_2 + 2x_3 + 4 \end{aligned}$$

Therefore, here is the first phase problem to solve.

		x_1	x_2	x_3	y_1	y_2
$-v$	-4	2	-2	2	0	0
x_2	2	0	1	4	1	0
x_4	2	-2	1	-6	0	1

In the end, after a solution is found, if a y_i variable is in the basis (i.e. it is the identifier of a row), it has to be swapped in with on $\underline{x}$ variable. This is done by applying the pivoting operation with the first value of the y_i row as pivot (x_1 column).

This is its solution.

		x_1	x_2	x_3	y_1	y_2
$-v$	0	0	0	0	1	1
x_2	2	0	1	4	1	0
x_1	0	1	0	5	1/2	-1/2

Now the second phase begin. The objective function z is to be made canonical, since it should not contain the basic variable that did remain in the first phase In this case, those basic variables are x_2 and x_1, which functions, dependent on other variables, are found from the solution system of the previous phase.

$$\begin{aligned} z &= x_1 + x_2 + 10x_3 \\ x_2 &= -5x_3 \\ x_1 &= 2 - x_3 \\ z &= 2 + 4x_3 \end{aligned}$$

Finally, the second phase can be started from the same table found in phase one, removing the y partition and swapping in the z objective function.

		x_1	x_2	x_3
$-z$	2	0	0	4
x_2	2	0	1	4
x_1	0	1	0	5

On this last table (which actually already represent the best feasible solution), the simplex algorithm can be applied directly, providing $\underline{x} = (0, 0, 4)$, with $z = 2$, as best feasible solution.

4 Integer linear programming