Resume

domenica 10 dicembre 2023 16:14

Mathematical methods for the solution of decision problem

DYNAMIC PROGRAMMING

is based into taking in to account previous decision in such a way to make the new decision faster

SEQUENCE OF DECISION (model an optimisation process)

CANNOT BE APPLIED TO EACH PROBLEM ⟶ IS A SOLUTION FOR PROBLEMS DEFINED OVER STAGES ← ARE THE SUBPROBLEM

SOLUTION OBTAIN RECURSIVELY ↓ ÷ SUBPROBLEMS ⟶ ALL TOGETHER MAKE THE SOLUTION

BASE ON OPTIMALITY PRINCIPLE: PROBLEM DIVIDE IN SUBPROBLEM AND THE SOLUTION IS THE SUM OF THE SOLUTION OF ALL SUBPROBLEM

MAIN IDEA OF DYNAMIC PROGRAMMING

it doesn't matter in which state I am and independently on how much I have invest till then ⟶ the important thing is using in the best way what remain

START THE THING IN THE LAST STATE

THE GRAPH IS TRAVELED BACKWARD starting from the last stage
⟶ Associates to each node a number THAT IS THE LENGTH OF THE LONGHEST PATH (MAXIMUS PATH)

⟹ STAGE $f_i(x) = \max_{y_{i+1}} [f_{i+1}(x_{i+1}) + R(x_i, x_{i+1})]$ ← STAGE; ↳ REWARD: lenght of the arc between x_i and x_{i+1}

Lenght of the maximal path between x_{N-1} / x_N

BACKWARD PHASE ⟶ COMPUTE THE OPTIMAL SOLUTION BETWEEN THE PAIR OF STAGES

GENERAL $T_i^\circ(x_i) = \min_{x_{i+1}} [T_i(x_i, x_{i+1}) + T_{i+1}^\circ(x_{i+1})]$ ↳ COST TO GO

Then we use the FOWARD PHASE and choose the best possibility for the best path ↓ PUT TOGETHER THE SOLUTION OF ALL THE SUBPROBLEM

WITHOUT DYNAMIC PROGRAMMING WE HAVE $N^{\text{modes}-1}$ (N: STAGES) POSSIBILITY

WITH DYNAMIC PROGRAMMING WE HAVE A MAXIMUS OF COMPUTATION ARE $q^2 \cdot N$ (q:)

OPTIMALITY PRINCIPLE (we have to go at the optimal trajectory)

$T^\circ = \min_{x_1} [\min_{x_2} [\cdots [\min_{x_{N-1}} T_N]]]$ ⟹ At each stage we have to make q sums $\forall q$ ⟹ q^2 sums

⟹ So we have to compute $q^2 \cdot N$

BELLMAN

"CURSE OF DIMENSIONALITY": EXPONENTIAL (OR MORE) GROWTH OF THE RESOURCES REQUIRED TO SOLVE A PROBLEM OF "DIMENSION" D WITH RESPECT TO D

If I solve sequential decision problems with brute force (= explicit enumeration) I have the curse of dimensionality with respect to the number of stages

DYNAMIC PROGRAMMING avoides the curse of dimensionality with respect to the number of stages

DYNAMIC PROGRAMMIN WORK'S ONLY TO PROBLEM THAT CAN BE DIVIDED IN SUBPROBLEM

LIMITATION:
- IF THE PREVIOUS STAGE INFLUENCE THE NEXT STAGE WE CANNOT USE D.P.
-
-

SUMMARY:
① BACKWARD: assign, starting from the arrival point, going back to the departure point constructing the partial optimal trajectories, and put togheter the partial trajectories to each arc
associated the optimal cost is the basic idea written on each arc
⟶ minimize problem (we have to search the minimus cost and viceversa)
↳ COST TO GO ↳ MAXIMUM FUNCTION

REAL SUMMARY OF D.P.
- IS REQUIRED THAT THE PROBLEM CAN BE DIVIDED INTO STAGES
- FOR EVERY STAGE, A DECISION POLICY HAS TO BE DETERMINE (BETWEEN THE CURRENT STATE AND THE NEXT ONE)
- A NUMBER OF STATES IS ASSOCIATED WITH EACH STAGE
- THE EFFECT OF THE DECISION AT EACH STAGE CONSISTS IN TRASFORMING THE PRESENT STATE INTO A NEW STATE, ASSOCIATED WITH THE NEXT STAGE
- AT A CURRENT STAGE, THE DECISION TAKEN AT PREVIOUS STAGES DOES NOT INFLUENCE THE DECISION AT NEXT STAGES
- THE BACKWARD SOLUTION PROCESS DETERMINES THE OPTIMAL DECISION POLICY FOR EACH STATE OF THE PREVIOUS STAGE
- THE OPTIMAL DECISION POLICY IS OBTAINED RECURSIVELY
- IT IS DETERMINED BY "TRAVELING" FOWARD THE SEQUECE OF DECISIONS

LINEAR PROGRAMMING

(MP) MATHEMATICAL PROBLEM: $\max g(x)$ → SCALAR OBJECTIVE FUNCTION; $x \in X \subseteq \mathbb{R}^n$ (x: VECTOR OF DECISION VARIABLE; X: SET OF FEASIBLE SOLUTION)

SINGLE OBJECTIVE OPTIMIZATION ⟶ $g(x) = c^T x$ ↳ $[c_1, \dots, c_m]$; $\underline{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}$

(MP) $\max f(x)$ → SCALAR OBJECTIVE FUNCTION ↳ $[c_1, \dots, c_m]$

$\underline{x} \in X \subseteq \mathbb{R}^m$

VECTOR OF DECISION VARIABLE — SET OF FEASIBLE SOLUTION

When every is linear (objective function) this is called LINEAR PROGRAMMING and the set of X is define by LINEAR EQUALITY or inequality CONSTRAINTS ($\leq$)

CONSTRAINS → $A\underline{x} \leq \underline{b}$ (decision vars) $\quad \underline{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}$ (numbers) $\quad A = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mm} \end{bmatrix}$

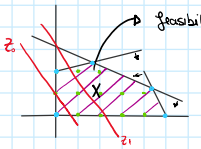
depending on the nature of the variables we have different types of linear programming problems

- CONTINUOUS LINEAR PROGRAMMING $X = \{\underline{x} \in \mathbb{R}^m : A\underline{x} \leq \underline{b}, \underline{x} \geq 0\}$ (Real number)
- INTEGER LINEAR PROGRAMMING $X = \{\underline{x} \in \mathbb{Z}^m : A\underline{x} \leq \underline{b}, \underline{x} \geq 0\}$ (Integer number)
- MIXED INTEGER LINEAR PROGRAMING $X = \{\underline{x} \in \mathbb{R}^m, \underline{y} \in \mathbb{Z}^m : A\underline{x} + D\underline{y} \leq \underline{b}, \underline{x} \geq 0, \underline{y} \geq 0\}$
  (when we have this we have to use the one of integer linear programming)

Whenever we have to model a problem we have to introduce three object: variables, constrain and objectives

- VARIABLES: change the value of the objective function → define the solution of the problem
- OBJECTIVE: quantifies the quality of a solution
- CONSTRAINTS: identify the set allowed values for the variables, defining the feasible solution set

The area of intersection of the constrains is called ADMISSIBLE REGION (is a polyedra)



feasibility set X { inner points + the sides } intersection of semi-spaces defined by hyperplanes (lines)

DIFFERENCE BETWEEN LP AND IP POLYEDRA?
→ A POLYEDRON FOR AN LP PROBLEM INCLUDE ∞ FEASIBLE SOLUTIONS
FOR AN INTEGER PROG. PROBLEM HAS A FINITE NUMBER OF FEASIBLE SOLUTIONS

GRAPHIC SOLUTION: Feasibly region X and objective function is draw in the plane they are a parallel lines and we have to find where the function is max or min

- this are the VERTICES OF THE POLYEDRO → we can move until the last vertices of the polytron → The last VERTIX OF THE POLYDRON IS THE OPTIMAL SOLUTION

LINEARITY IMPLIES THAT OPTIMAL SOLUTIONS ARE IN THE POLYEDRON VERTICES
MORE THAN A SINGLE EQUIVALENT SOLUTIONS IF THE OBJECTIVE FUNCTION IS PARALLEL TO A CONSTRAINT

A CONSTRAINT IS SATURATED WHEN IS SATISFIED WITH EQUALITY AND THE CORRESPONDING RESOURCE IS CALLED SCARSE RESOURCE
WHEN THE CONSTRAINT IS NOT SATURATED THE CORRESPONDING RESORCE IS CALLED ABUNDANT

When we increased (or decrease) a resource we move the constraint in a parallel line, we can increased the resource until we reach another constraint

PLANNING PROBLEM (EXAMPLE)

- TRASPORTATION PROBLEM: Considering certain time period determine the most convinient way to trasport some available items for satisfying a given demand → determine the quantities of product trasported between each pair (i, j) supplier - customer such that the demand are satisfied

$\sum_{i=1}^{m} s_i \geq \sum_{j=1}^{n} r_j$ m: producer n: customer s_i: produce product r_j: required product

- Objective function: total transportation cost $\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} x_{ij}$ $i: 1 \dots m$ $j: 1 \dots n$

FORMULATION $\min \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} x_{ij}$

- Constraint:
  - total quantity supplied $\sum_{j=1}^{n} x_{ij} \leq s_i$ cannot exceed availability
  - total quantity received $\sum_{i=1}^{m} x_{ij} = r_j$ must be equal to the demand
  - Only positive quantities $x_{ij} \geq 0$

s.t.
$\sum_{j=1}^{n} x_{ij} \leq s_i \quad i = 1, \dots, m$
$\sum x_{ij} = r_i \quad j = 1, \dots, n$
$x_{ij} \geq 0$

- PRODUCT MIX PROBLEM: determine the optimal level of production in a reference time period for a set of products taking into account the limited availability of a set of resources in the same period → determine the quantities to produce in order to maximize the total profit without exceeding the resources availability

- Objective function: total profit from production $\sum_{i=1}^{n} c_i x_i$ n: different product m: different production resources
- Constraint
  - For each resource, the total used quantity cannot exceed the maximum $\sum a_{ij} x_i \leq b_j$ $j: 1 \dots m$
  - Only positive quantities $x_{ij} \geq 0$ $i: 1 \dots n$

FORMULATION
$\max \sum_{i=1}^{n} c_i x_i$
s.t. $\sum_{i=1}^{n} a_{ij} x_i \leq b_j$
$x_i \geq 0$

- PRODUCTION PLANNING PROBLEM: plan the level of production for a single product type over a time horizon determining the production for each time period in which the horizon has been divided → determine the product quantity to be produced in the considered periods to satisfy the demand, minimizing the overall production and inventory cost

- Objective function: total cost for production and inventory in N periods

FORMULATION

cost

- Objective function:
  total cost for production and inventory in N periods
  $\sum_{i=1}^{N} (c_i x_i + r_i s_i)$ x_i: production planned cost s_i: product store cost

FORMULATION
$\min \sum_{i=1}^{N} (c_i x_i + r_i s_i)$
s.t.
$x_i + s_{i-1} - s_i = d_i \quad i = 1 \dots n$
$x_i \leq m_i \quad i = 1 \dots m$
$s_0 = I_0$
$x_i \geq 0 \quad s_i \geq 0$

- Constraints
  - product flow conservation $x_i + s_{i-1} = d_i + s_i$
  - production in each period cannot exceed production capacity $x_i \leq m_i$
  - starting inventory level $s_0 = I_0$
  - production and inventory cannot be negative

STANDARD AND CANONICAL FORMS

CANONICAL FORM OF MAXIMIZATION
$\max x_0 = c^T x$
$Ax \leq b$
$x \geq 0$

CANONICAL FORM OF MINIMIZATION
$\min x_0 = c^T x$
$Ax \geq b$
$x \geq 0$

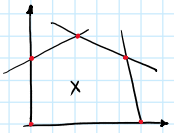
STANDARD FORM
$\max x_0 = c^T x$
$Ax = b \Rightarrow b \geq 0 \Rightarrow b_j \geq 0 \ \forall j = 1 \dots m$
$x \geq 0$

number of constraint $\leq$ number of variables (m)
m → RANK(A)
WE USE STANDARD FORM WHEN WE ARE DEALING WITH GEOMETRY

IN STANDARD FORM THE COEFFICIENT VECTOR IS MADE UP OF NON NEGATIVE QUANTITIES
↳ this are for more feasible solution

BOUNDED: the set of admissible solution are limited and exist ⇒ A NOT EMPTY POLYHEDRON (intersection of halfspaces)
+ IS CLOSED (POLYTOPE)

SOLUTION:
- FEASIBLE WITHOUT BOUNDED → NOT CLOSE
- NOT FEASIBLE → EMPTY POLYHEDRON (cannot be verified simultaneously)



POLYHEDRA
- A point $y \in X$ of a convex set X is an EXTREME POINT (VERTEX) that is a convex combination
  ⟺ do not exist $x_a, x_b \in X$, $x_a \neq x_b$ such that $y = \lambda x_a + (1-\lambda) x_b$ for some $0 < \lambda < 1$

⇒ We can write any point as convex combination of 2 extreme points

* The most significant case is when $m = \text{RANK}(A)$ and $m \leq n$ (GUARDA SLIDE) is the only case we have maximization / minimization problem
⇒ Having this we can split the matrix $A = [B|N]$ (n→ no basic matrix; B → basic matrix (squared matrix))
$A = \left[\underbrace{| | | |}_{B} \ \overbrace{| | |}^{N} \right]$
we can fixed the non-basic variables to obtain the maximization / minimization solution

FOR SOLVING THE SIMPLEX ALGORITHM WE USE $x_B = B^{-1} b - B^{-1} N x_N$ (with this = 0)

$x = \begin{bmatrix} x_B \\ x_N \end{bmatrix} = \begin{matrix} B^{-1} b \\ 0 \end{matrix}$

$x_B = B^{-1} b \geq 0 \Rightarrow$ BASIC FEASIBLE SOLUTION (BFS)

↓
SATISFY THE NO-NEGATIVITY CONSTRAINT → BFS
↓
correspond to extreme points from the geometric point of view

CONVERTION TO STANDARD FORM

SOLVING A LINEAR PROGRAMMING PROBLEM means search among a finite set of solution (BFS) corresponding to polyedron vertices (basic matrix)

we have to write it as equality
Ex: $x_1 + x_2 \leq 5 \Rightarrow x_1 + x_2 + s_1 = 5$
↳ SLACK VARIABLE → THEY HAVE ALSO TO BE POSITIVE

when we have $\leq$ we have to write $+ s_i$
if they are $\geq$ we have a SURPLUS VARIABLE and they are written as $- s_i$

IF BOTH CASES $s_i = 0$ THE CONSTRAIN IS SATISFIED AS EQUALITY

FREE VARIABLE we have to substituted by two non negative variable $x_j = u_j + v_j$

! IF I HAVE MINIMIZATION PROBLEM WE HAVE TO MULTIPLY OBJECTIVE FUNCTION BY -1

* The degenerate case we have more than a single basis matrix

How many basic solutions we have? $\binom{n}{m} = \frac{n!}{m!(n-m)!}$

* FOR CHOOSING THE BASIS WE SELECT THE ONE THAT DON'T HAVE THE EQUALITY IN THE CONSTRAIN EQUATION (SATURATED)

SIMPLEX METHOD

THEOREM (OPTIMALITY CONDITION)
A basic solution of a linear programming problem is optimal if
1) $y_{i0} \geq 0 \quad i = 1 \dots m$ (feasible)
2) $y_{0j} \geq 0 \ \forall j \in \mathbb{R}$ (cannot be improved)

CHANGE THE BASIS MEANS CHANGE THE VERTEX → when we move means that one of the slack variables change the value from 0 to a this can increase until we have the next constraint and the variable that was in the basis that become zero is the one that exit the basis

* DEGENERATE SOLUTION IS A VERTEX IN WHICH IS OBTAIN FROM DIFFERENT MATRICES → THE VALUE OF THE OBJECTIVE DOESN'T CHANGE INDIPENDENTLY ON WHICH MATRIX IN THE CONSIDER IN THE VERTEX → CYCLING
↳ OVERDETERMINED

* DEGENERATE SOLUTION IS A VERTEX IN WHICH IS OBTAIN FROM DIFFERENT MATRICES → THE VALUE OF THE OBJECTIVE DOESN'T CHANGE INDIPENDENTLY ON WHICH MATRIX IN THE CONSIDER IN THE VERTEX → CYCLING
→ OVERDETERMINED

HOW DO WE CHOOSE THE ENTERING VARIABLE

1) If we have more than one negative coefficient we have to choose the most negative → we obtain the largest local increased → the objective function change twice DANTZIG METHOD
   1) directly because x_i enters
   2) indirectly because x_i entering variable change the values of the basic variable
2) (*) we should choose:
   OPTIMAL ENTERING CRITERION $K = \arg\max \left(-y_{0j} \frac{y_{r0}}{y_{rj}} \right)$ actual objective increase

HOW TO CHOOSE THE LEAVING VARIABLE

* for unbounded solution
  1) coefficient negatives $y_{ik} \leq 0$ $x_B = y_{i0} - y_{ik} x_k$

The simplex algorithm includes 5 steps

1. Initialization a starting BFS (Initial vertex) (BASIC FEASIBLE SOLUTION)
2. Optimality test (if the coefficient of the non basic variable in the objective are no-negatives → the solution is optimal)
3. Select the entering variables → choose a non basic variable x_k such that $y_{0k} < 0$
4. Select the leaving variable (such that $\frac{y_{r0}}{y_{rk}} = \min \left\{ \frac{y_{i0}}{y_{ik}} \right\}$
5. Pivoting (solve the equation $x_B = y_{i0} - \sum_{j \in R} y_{ik} x_j \quad \forall x = 0, 1 \dots, m$

FOR THIS OPERATION WE WILL MADE A POLINOMIAL NUMBER OF ITERATION only when we have a exponential number of polinomial we do an exponential number of iteration

* For point 1 we have 3 possibilities to initialize (slack variables, Two phases method and Big-M method)

SLACK VARIABLE — Solving the feasibility problem

$\max -c^T x$ → STANDARD FORM → $\max x_0 = c^T x$

$Ax \leq b$ → $Ax + Is = b$ (we put this for all the inequality constraint; identity matrix)

$x \geq 0$ → $x \geq 0 \quad s \geq 0$

$[A \mid I]$ we are choosing a basis with only slack variables

EXAMPLE

$\max x_0 = 2x_1 + x_2$

$x_1 + x_2 \leq 5$

$-x_1 + x_2 \leq 0$

$6x_1 + 2x_2 \leq 21$

$x_1, x_2 \geq 0$

STANDARD FORM ⟹

$\max x_0 = 2x_1 + x_2$

$x_1 + x_2 + x_3 = 5$

$-x_1 + x_2 + x_4 = 0$

$6x_1 + 2x_2 + x_5 = 0$

$x_1, x_2 \geq 0 \quad x_3, x_4, x_5 \geq 0$

1 INITIALIZATION

$$A = \begin{array}{c} \begin{matrix} x_1 & x_2 & s_3 & s_4 & s_5 \end{matrix} \\ \left[\begin{array}{ccccc} 1 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 6 & 2 & 0 & 0 & 1 \end{array}\right] \end{array} \quad I$$

INITIAL BFS (Basic f. solution)

$$\begin{bmatrix} s_3 \\ s_4 \\ s_5 \end{bmatrix} = b = \begin{bmatrix} 5 \\ 0 \\ 21 \end{bmatrix}$$

We initialized with the origin of the axis

TWO-PHASE METHOD (General method)

$\max x_0 = c^T x$
$Ax = b$
$x \geq 0$

I PHASE

(A) $\min z = 1^T y = \sum_{i=1}^{m} y_i$ → AUXILIARY VARIABLES → In this phase I add the identity and change the objective function

$Ax + Iy = b$

We have to minimize the sum looking for the minimum value of the auxiliary variables (0) to obtain the optimal solution

We will obtain two possibilities

1) $z = 0$ All the auxiliary variables are zero and out of the basis → The basis is a feasible solution for the original problem
2) $z \neq 0$ At least one variable have to remain in the basis → The original problem CANNOT be satisfy without the aux

II PHASE

- If the optimal solution is $z = 0$ → then I can initialize with x_i
- If $z \neq 0$ the problem is not feasible

BIG-M METHOD

In this we also need auxiliary variables

The objective function is modified to penalize the auxiliary variables

$\max c^T x - M 1^T y = \sum_{i=1}^{n} c_i x_i - M \sum_{j=1}^{m} y_j$

To maximize this function
=> We need to put out all the variables out of the basis

$Ax + Iy = b$

$M \gg |c_i|, |b_j|, |a_{ij}|$

$x \geq 0 \quad y \geq 0$

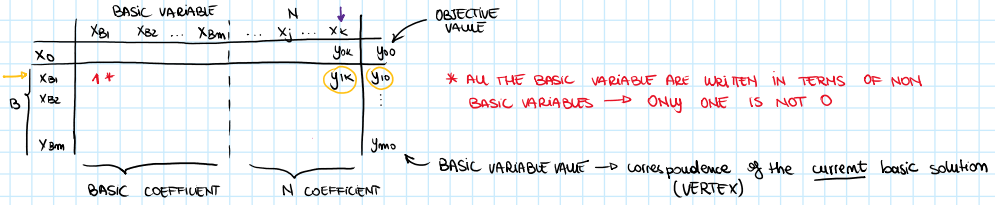
=> If not all the auxiliary variables out of the basis → the original constraint were not feasible → UNFEASIBLE

$x \geq 0 \quad y \geq 0$ $\qquad M \gg |c_i|, |b_j|, |a_{ij}|$

⇒ If not all the auxiliary variables out of the basis → the original constraint were not feasible → UNFEASIBLE

IF M IS NOT LARGE ENOUGH THE SIMPLEX WILL TAKE MANY STEPS (TOO MANY COMPUTATION → OVERFLOW)

SIMPLEX IN TABULAR FORM



* ALL THE BASIC VARIABLE ARE WRITTEN IN TERMS OF NON BASIC VARIABLES → ONLY ONE IS NOT 0

1) WE HAVE TO VERIFY THAT THE COEFFICIENT OF THE NO-BASIC VARIABLE ARE ZERO

2) OPTIMALITY TEST: IF they are ≥ 0 → DONE

3) IF the solution is not optimal → Choose the most negative (no basic variable) → this will be the columb of the entering variable)

4) FOR THE EXITING VARIABLE → I have to compute the ratio $x_k = \frac{y_{i0}}{y_{ik}}$ the minimum ratio

* IF ALL THE COEFFICIENT y_{ik} ARE NEGATIVE → NO BASIC VARIABLE BECOME ZERO → UNBOUNDED CONDITION

5) $x_k = \frac{y_{r0}}{y_{rk}} - \sum_j \frac{y_{rj}}{y_{rk}} x_j - \frac{1}{y_{rk}} x_{Br}$

$x_{Bi} = y_{i0} - y_{ik}\frac{y_{r0}}{y_{rk}} - \sum (y_{ij} - y_{ik}\frac{y_{rj}}{y_{rk}}) x_j + \frac{y_{rj}}{y_{rk}} x_{Br}$

EXAMPLE

$\max z = 3x_E + 2x_1$

$x_E + 2x_1 \leq 6$
$2x_E + x_1 \leq 8$
$-x_E + x_1 \leq 1$
$x_1 \leq 2$
$x_E \geq 0 \quad x_1 \geq 0$

STANDARD FORM →

$\max z = 3x_E + 2x_1$

$x_E + 2x_1 + s_1 = 6$
$2x_E + x_1 + s_2 = 8$
$-x_E + x_1 + s_3 = 1$
$x_1 + s_4 = 2$

$x_E, x_1 \geq 0 \quad s_1, s_2, s_3, s_4 \geq 0$

s_1, s_2, s_3 and zero

$z = 0 - (-3x_E - 2x_1)$

NOT BASIC VARIABLE →

$s_1 = 6 - (x_E + 2x_1)$
$s_2 = 8 - (2x_E + x_1)$
$s_3 = 1 - (-x_E + x_1)$
$s_4 = 2 - (0x_E + x_1)$

IS THE MOST NEGATIVE 3

	x_E	x_1	s_1	s_2	s_3	s_4		
z_0	-3	-2	0	0	0	0	0	
s_1	1	2	1	0	0	0	6	6/1 = 6
s_2	2	1	0	1	0	0	8	8/2 = 4 ← MINIMUM RATIO 4
s_3	-1	1	0	0	1	0	1	— (NEG)
s_4	0	1	0	0	0	1	2	— (0)

1) In this case the pivot is 2 and we divide s_2 by pivot

s_2 will BECOME

s_2 | 1 ½ | 0 ½ 0 0 | 4 NEW ROW → Now I have to update all the other rows

2) x_E becomes basic in place of s_2 and multiply by -3

x_E | 1 ½ | 0 ½ 0 0 | 4

x_E | -3 -3/2 | 0 -3/2 0 0 | -12

Now we update the z subtracting from the new x_E the old z

We will obtain z_{NEW} ⇒ z | 0 -½ | 0 3/2 0 0 | 12

Then we have to update every row always repeating the same EX: $s_1 - (1)x_E$; $s_3 = s_3 - (-1)x_E$; $s_4 = s_4 - (0)x_E$

When we finish → repeat the 1 part and the optimization test

* WHEN WE HAVE INEQUALITY CONSTRAINED THAT ARE ≥ IS BETTER TO USE THE TWO-PHASE METHOD AND IT IS UNNECESSARY TO AD ONE AUXILIARY CONSTRAINT
→ we will have already the $\mathbb{1}$ matrix and max only the − auxiliary constrained and we need to have only non basic variable have to have coefficient different to zero

* we have to sustitude also the new objative function (computed) and then compute the normal tableu without the auxiliare variables putting also the value of them in the originals (DA RIGUARDARE) INITIALIZE I − II − III ...

INTEGER LINEAR PROGRAMMING (One or more variable have to be integer)

We use this kind when fractional values are not acceptable approximation

KNAPSACK PROBLEM

* We have to choose object to be put into the knapsack with a fixed capacity and depending on the utility and then maximize the total
→ Determine the best projects to be funded in order to maximize the total revenue without exceeding the available budget

Objective function: $\max \sum_i^n c_i x_i$ i: project

and then maximize the total
→ Determine the best projects to be funded in order to maximize the total revenue without exceeding the available budget

Objective function: $\max \sum_{j=1}^{m} c_j x_j$ j: project

Constraint
- The available budget is not exceed $\sum a_j x_j \leq b$
- Variable x_j assume binary values $x_i \in \{0,1\}$

* This problem belongs to the family of most difficult problem → have an exponential grow we search for sub-optimal solution

ASSIGNMENT PROBLEM (EASY PROBLEM)

→ Determine the assignment of all activities to the workers such that the total cost is minimized

Objective function $\min \sum_{i}^{m} \sum_{j}^{n} c_{ij} x_{ij}$ m: activities n: workers

Constraints:
- Each activity must be assigned $\sum x_{ij} = 1$
- Each worker can at most perform a single activity $\sum^{m} x_{ij} \leq 1$
- Variables are binary to ensure that the activity is entirely assigned to a single worker $x_{ij} \in \{0,1\}$

* THE NUMBER OF CONSTRAINS IS NOT ASSOCIATED WITH THE COMPLEXITY OF THE PROBLEM!

TRANSPORTATION PROBLEM WITH FIXED CHARGE

→ Determine which channels must be rented and the flow in them so that the customers demand is satisfied at the minimum total cost

· Objective function: is not linear BUT it can be still linearly modelled
- It include both marginal and fixed cost

· Constraints:
- Supplier capacities $\sum_{j=1}^{n} x_{ij} \leq s_i$
- Customer demands $\sum_{i}^{m} x_{ij} = r_j$

⊕ Variables x_{ij} and y_{ij} must be linked: trasportation is allowed only on active channel $x_{ij} \leq M y_{ij}$
- If a maximum channel capacity q_{ij} is given for channels we can use q_{ij} instead of M $x_{ij} \leq q_{ij} y_{ij}$

SEQUENCING PROBLEM (planning of production activities)

→ Determine the optimal sequence for a set of operation

Objective function: (associated with job completion time)

⊕ THE LINEAR PROGRAMMING PROBLEM IS THE LINEAR RELAXATION OF THE INTEGER LINEAR PROGRAMMING PROBLEM

The solution of the relax problem in general is not the solution of the integer programming

* THE IDEAL FORMULATION OF IP IS THE SMALLEST POLYHEDRON THAT CONTAIN ALL THE FEASIBLE POINTS → IF we use it we will have an LP problem which is the RL of IP
→ THAT IS THE CONVEX COMBINATION OF THE SET OF FEASIBLE POINTS

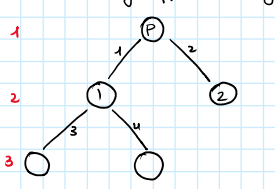
BRANCH AND BOUND METHOD

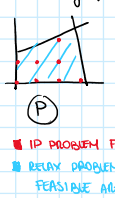
Examines disjoint subsets → divide the feasibility region
1) Relax the problem → Consider the feasibility region into two disjoint subregion to each of them it is associated one optimization problem (BRANCHING)
2) Each subset is evaluated using estimation of the objective function (BOUNDING) and eliminates the subsets that cannot contain the optimal solution

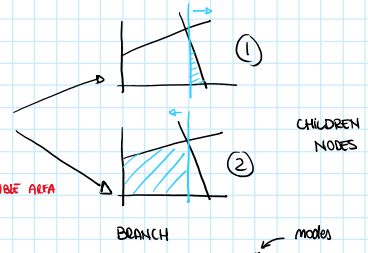
⟹ SUBDIVIDE THE WHOLE PROBLEM INTO SET OF SUBPROBLEMS OF PROGRESSIVELY DECREASING DIMENSIONS (MORE EASY TO SOLVE)

The problems are generated with recursive hierarchical relation
- parent problem → divide into 2 disjoint child (BRANCHING PHASE)
→ THE SOLUTION FOR THE PARENT CORRESPOND TO THE OPTIMAL SOLUTION FOR ONE OF THE CHILD PROBLEMS

the exploration is easily rappresented by a tree graph







* In general an IP problem (0-1) with n variables have n+1 levels and 2^n leaves ⟹ $\sum_{i}^{n} 2^i$ nodes

At each level we add a new constrain

EXPLORE TREE $2^{n+1}-1$ BRUTE FORCES

THE IDEA IS → AS SOON I CAN BOUND SOME PART OF THE TREE PRUNING THE MORE EFFICIENT THE ALGORITHM

BRANCH AND BOUND CAN BE APPLIED TO ANY COMBINATORIAL OPTIMIZATION PROBLEM

METHOD:

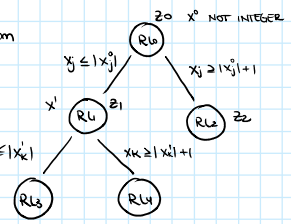
① solve the relax problem associating the solution root $\Rightarrow$ z_0 objective value and x^0 optimal solution

② Choose a not integer solution component x_j^0

③ Partition the feasibility region $P^0 = \{x \in \mathbb{R}^n : Ax = b,\ x \geq 0\}$ into two disjoint region adding P^0 one of the constrain:

$$x_j \leq \lfloor x_j^0 \rfloor \qquad x_j \geq \lfloor x_j^0 \rfloor + 1$$

④ Iterate for the child if the don't have z integer



- ACTIVE NODES: generated nodes that are not solved or solved but with not integer solution
- INTEGER NODES: not explored anymore (NODE FATHOMED FOR INTEGRALITY) $\rightarrow$ when we found an integer solution
- UNFEASIBLE NODES: not explored anymore (NODE FATHOMED FOR UNFEASIBILITY) $\rightarrow$ when $x_k' + 1 \geq \lfloor x_j^0 \rfloor$ that means that the have empty feasibility region

The first integer solution we put in memory as $\rightarrow$ UPPER BOUND for a minimization
(INCUMBENT SOLUTION) $\rightarrow$ LOWER BOUND for a maximization $\rightarrow$ we will never accept an inferior solution

LOWER BOUND is fundamental for BOUNDING

* The parent will be an UPPER BOUND (in maximization) for the discendant of the node

(*) When I find an integer solution and compare it with the LOWERBOUND IF IT IS LESS THAN OR EQUAL TO THE LOWER BOUND IT MEANS THAT IT IS NOT WORTH EXPLORING THAT PART OF TREE $\rightarrow$ FATHOMING FOR BOUNDING

Whenever I found an integer solution we will compare the two if better we replace the last one

If the method not converge the problem was UNFEASIBLE $\rightarrow$ THE WORST CASE SCENARIO IS EXPLORING ALL THE NODES

In case of MINIMIZATION the terms of UB and LB must be exchange and the condition for fathoming for bounding must be reverse

BRANCH AND BOUND ALGORITHM

① INITIALIZATION $\rightarrow$ R_0 the root $z_{LB} = -\infty$ and $z_0 = \infty$ for node R_0

② BRANCHING

③ SEPARATION $\rightarrow$ select a basic fractional variable $x_{Bi} = y_{i0}^j$ and partition P

④ SOLUTION OF R_j
- SOLVED PROBLEM
- IF no feasible solution exit the node is fathomed for unfeasibility $\rightarrow$ ②
- IF optimal solution exists continue ⑤

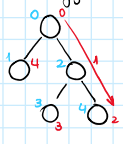
⑤ FATHOMING FOR INTEGRALITY
- check if $z_{LB} = \max\{z_{LB}, z_j\}$

⑥ FATHOMING FOR BOUNDING
- Prune any active node K such that $z_K \leq z_{LB}$

⑦ TERMINATION

The effectiveness strongly depends on the exploration strategy of the tree
There are opposite branching strategy

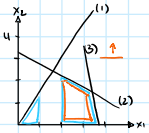
- DEPTH FIRST
  - Fix variable quickly reach an integer solution LB
  - Reduced number of active nodes

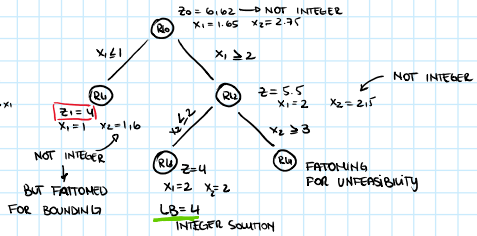


- BREADTH FIRST
  - Compute UB for many nodes hoping to prune many branches by bounding as soon as an integer solution is found
  - Memory usage is larger than depth first strategy

EXAMPLE

$$\max\ -x_1 + 3x_2$$
$$x_1 - \frac{3}{11}x_2 \geq 0$$
$$\frac{3}{2}x_1 + 2x_2 \leq 8$$
$$x_1 + \frac{3}{10}x_2 \leq \frac{7}{10}$$
$$x_1, x_2 \geq 0$$





RELAXATION $\rightarrow$ A solution to an optimization problem cannot worsen if we eliminated a problem constraint
- if $x^0 \in P$ the optimal solution to (RL) is also optimal for IP. However in general $x^0 \notin P$

TOTAL UNIMODULARITY $\rightarrow$ IS AN IDEAL FORMULATION OF A INTEGER PROGRAMMING PROBLEM
- if the determinant is $\{-1, 1\}$
- if all the submatrices are unimodular

$\Rightarrow$ IF A IS TUM then every feasible solution to (RL) is integer
Necesary condition: $A = [a_{ij}]$ to be TUM $a_{ij} = \{0, 1, -1\}$

TU theorem
A is TUM if
1. Every column has at most two non zero elements

TU theorem

A is TUM if

1. Every column has at most two non zero elements
2. The rows can be partitioned into two sets Q_1 and Q_2 such that
   - if a column j contains two no zero elements a_{ij} and a_{rj} with the same sign then $i \in Q_1$ e $h \in Q_2$
   - if a column j contains // // with different sign then both belong to the same matrix

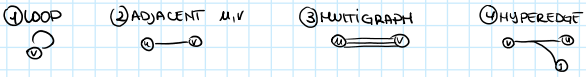
IF RP $x^0 \in \mathbb{Z}_+^m$ $\Rightarrow$ $x^* = x^0$ I.P $\Leftarrow$ IDEAL FORMULATION ($\uparrow$ SUFFICIENT CONDITION) $\Leftarrow$ b INTEGER A T.U.M ($\uparrow$ SUFF. COND.) $\Leftarrow$ A SATISFY THE CRITERION OF THE INDICES ($\downarrow$ SUFFICIENT COND.)

GRAPH THEORY

- Rappresentation relationship among different kinds of entities
- Formal tool to model problems
- Allows modelling many diffrent decision problems

They can be

- UNDIRECTED : set of n nodes and set of m non directed arcs (edge)



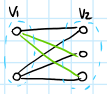
SIMPLE GRAPH → ONLY 2

A set of nodes that are connected to a node is the NEIGHBOURHOOD of the node

Subgraph : is a subset of the graph

Induce subgraph : is a subset all connected

Bipartite graph : graph where exist the partition $V = V_1 \cup V_2$ where → $V_1 \cap V_2 = \emptyset$ $e = (u,v) \in E$ $u \in V_1$ then $v \in V_2$



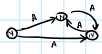
Complete graph : every node is connected → max. number of edges $\binom{n}{2} = \frac{n(n-1)}{2}$

DENSE GRAPH $m = \Theta(n^2)$

SPARSE GRAPH $m = \Theta(n)$

m: arcs

- DIRECTED : set of n nodes and set of m directed arcs (ordered pair)



HEAD TAIL

In this kind of graph we have twice the number of arcs $2\binom{n}{2} = n(n-1)$

Clique → complete graph
- maximum → the larghest
- maximal → cannot be enlarge

- MIXED GRAPH

* GRAPH DENSITY : RATION BETWEEN THE NUMBER OF ARCS AND THE MAXIMUM NUMBER OF ARCS

- UNDIRECTED GRAPH $D = \frac{2m}{n(n-1)}$ SPARSE DENSE $0 \leq D \leq 1$
- DIRECTED GRAPH $D = \frac{m}{n(n-1)}$ SPARSE DENSE $0 \leq D \leq 1$

WALK : A sequence of nodes

PATH : is a simple walk → ALL THE NODES AND ARCS ARE DISTINCT

CONNECTED GRAPH : if there is a walk that connection between the nodes (ALL)

CONNECTED COMPONENTS IF there is a walk between the components
for directed graph → STRONGLY CONNECTED

ACYCLY : graph that not contain any cycle →
- contains at least n-1 arcs
- atmost n-1 arcs
- → Acyclic + CONNECTED + n-1 ARCS

- TREE is an acyclic connected graph
- FOREST is a not connected acyclic graph

Theorem: undirected connected graph is eulerian iff every node has even degree (→ the non zero elements in the row)

if Incidence matrix is unimodular we are happy :) ⇒ T.U.M

SPANNING TREE : All node are connected → contains all nodes

HAMILTONIAN PATH is a path visiting all nodes in V exactly once → HAMILTONIAN CIRCUIT is a close hamiltonian path

MINIMUM SPANNING TREE PROBLEM

→ UNDIRECTED GRAPH

the problem: find a spanning tree with the minimum cost

There are two algorithm that study this kind of problem :

KRUSKAL ALGORITHM

① Given an undirected graph we have to sort the edges → cost are non-decreasing

KRUSKAL ALGORITHM

① Given an undirected graph we have to sort the edges → cost are non-decreasing

② Select the smallest cost unless they make a cycle

③ If the number of arcs is n−1 the algorithm terminates

→ THIS IS A GREEDY ALGORITHM → AT EACH STEP IT TRIES TO INSERT IN THE SOLUTION THE COMPONENT WITH THE MINIMUM COST (LOCAL BEST DECISION) ignore all choices that are already done (and those still to do)

* In general this type of algorithm do not provide the optimal solution

COMPLEXITY OF THIS ALGORITHM:

FOR STEP ① we have to sort m arcs (edges)

WORST CASE:

- I have to include m numbers → m steps
- How many comparison → $\log_2(m)$

⇒ $m \log_2 m$ WORST-CASE COMPLEXITY FOR ORDERING THE ARC

FOR STEP② Select the smallest

WORST CASE:

- I have to consider all the cost → n^2 WORST-CASE COMPLEXITY FOR SELECTING

⇒ WORST CASE $O(m \log_2 m + n^2)$

	GENERAL	SPARSE GRAPH	DENSE GRAPH
1st IMPLEMENTATION	$\Theta(m \log_2 m + n^2)$	$\Theta(n^2)$	$\Theta(n^2 \log_2 m)$
2nd IMPLEMENTATION	$\Theta(m \log m)$	$\Theta(m \log_2 m)$	$\Theta(n^2 \log_2 m)$

⇒ KRUSKAL DEALS WITH MST CONNECTING SMALL TREES (CONNECTING COMPONENTS)

PRIM – DIJKSTRA ALGORITHM

① Start with one node (whichever) and consider all exiting node

② Connect a node with the smallest cost of the exiting node

③ If the numbers of arcs is n−1 the algorithm terminates

⇒ PRIM-DIJKSTRA DEALS WITH THE MSP STARTING WITH A NODES AND INTERATIVELY ENLARGING ONE SINGLE TREE

COMPLEXITY OF THE ALGORITHM (polinomial complexity

FOR STEP② select the node

WORST CASE

- We have to include n−1 arcs but also not consider the ones that they are already included m

$\Theta(m \cdot m)$ → COMPLEXITY FOR SELECTING

	GENERAL	DENSE GRAPH	SPARSE GRAPH
1st IMPLEMENTATION	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
2nd IMPLEMENTATION	$\Theta(m \log m)$	$\Theta(n^2 \log m)$	$\Theta(m \log m)$

SHORTEST PATH PROBLEM

→ DIRECTED GRAPH

A cost is associated to each arc

Objective: find the directed path P* that connects two given nodes with the minimum cost

- If there no exist path to a node then the ⇒ PROBLEM IS UNFEASIBLE
- There exists a DIRECTED CYCLE with NEGATIVE COST ⇒ PROBLEM IS UNBOUNDED

* IF we know A PRIORI that thee is no negative cycle → the problem of the shortest path is POLYNOMIAL PROBLEM (totally unimodular)

SHORTEST PATH
- GENERAL CASE N.P. HARD → Exact algorithm, inefficient for large graph / Approximative algorithm
- When ∄ Negative cycle: POLYNOMIAL → Algorithm for LP (based on I.P. model of the S.P problem)
- When ∄ Negative cycle: POLYNOMIAL → Algorithm based on graph nature of S.P problem: DIJKSTRA, BELLMAN FORD, DYNAMIC PROGRAMMING

DIJKSTRA'S ALGORITHM (NO NEGATIVE COST)

* the theorem suggest that at least one node would be optimize (*)

① Initialization: first node cost 0, the other ones are all ∞ (SOURCE)

② Select one node from the one not selected earlier

* WE CONSIDER ALL ARCS EXITING THE NODES FROM THE SET OF NODES ALREADY CONSIDERED

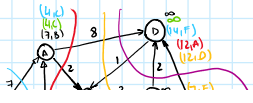
$i \in \arg\min\{h(l) : l \in U\}$ (BASE) (with the smallest label)

③ Update the label ↳ we have to select among all the nodes not yet visited the one with the smallest label

$h(j) = \min\{g(i) + c_{ij}, h(j)\}$ → IMPROVEMENT

(*) At each iteration it checks wether the labels need to be updated

EXAMPLE

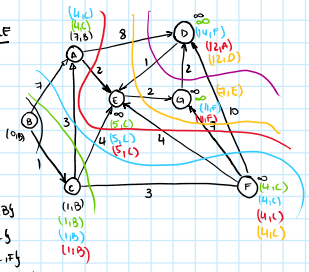


$h(j) = \min\{g(i) + c_{ij}, h(j)\}$ → IMPROVEMENT

(*) At each iteration it checks wether the labels need to be updated

(4) The algorithm finish when there is no update and all the nodes are in the base

EXAMPLE



- INIZ $U=\{B\}$
- $U=\{B,C\}$
- $U=\{B,C,F\}$
- $U=\{B,C,F,A\}$
- $U=\{B,C,F,A,E\}$

THE COMPLEXION OF THE ALGORITHM IN THE WORST CASE SCENARIO IS $\Theta(m^2)$

⇒ DIJKSTRA PROVIDES THE SHORTEST PATH TO ALL THE NODES (ONE-TO-ALL)

Have two phases: (1) Foward phase: construction and updating labels

(2) Backward phase: reconstruction of the shortest paths

* The cost is the minimum (→ Every entering path of the node is on the base and the value will not change anymore)

ARBORESCENCE OF SP: A set of trees with the same root

BELLMAN-FORD ALGORITHM → This algorithm can be use also with NEGATIVE COSTS

(1) Initialization the same as DIJKSTRA BUT ALL OTHER nodes different from the SOURCE are ∞

(2) Compute: $h^k(j) = \min\{h^{k-1}(j), \min\{[c_{ij} + h^{k-1}(i)]\}\}$ (→ at each iteration all the labels of all nodes)

MAIN DIFFERENCE WITH DIJKSTRA:

→ AT EACH STEP I RECONSIDER ALL THE LABELS OF ALL NODES (ALSO WITH NEGATIVE COST)

(ALL ARCS)

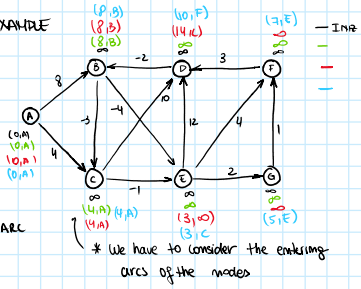
(3) Terminate when number of iteration is $K=n$

If there is an update at $K=n$ there is a negative cycle → PROBLEM IS UNBOUNDED

THE COMPLEXITY IS $\Theta(mn)$ (m: → CHANGE WITH THE TYPE OF GRAPH: SPARSE $\Theta(m^2)$, DENSE $\Theta(m^3)$)

* We have to consider all entering arcs on each node and consider the previous step cost to update the cost

EXAMPLE



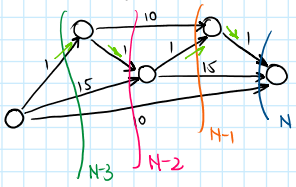
SUMMARY COMPLEXITY

	GENERAL	SPARSE GRAPH	DENSE GRAPH
DIJKSTRA	$\Theta(m^2)$	$\Theta(m^2)$	$\Theta(m^2)$
BELLMAN FORD	$\Theta(mnm)$	$\Theta(m^2)$	$\Theta(m^3)$

DYNAMIC PROGRAMMING (RIVEDERE, FORSE SBAGLIATO)

* WE START FROM THE GOAL AND CONSIDER THE ENTERING ARC AND THE ONE THAT ARRIVE WITH ONLY ONE ARC

EXAMPLE



COMPLEXITY $\Theta(m^2)$

NON LINEAR PROGRAMMING

Is the maximization/minimization problem of non linear function

UNCONSTRAINED $\Omega = \mathbb{R}^m$ (WHOLE)

CONSTRAINED $\Omega \subset \mathbb{R}^m$ (PROPER SUBSET)

GLOBAL MINIMUN: The minimum value of the whole domain

LOCAL MINIMUN: The minimum value in the neighbourhood

MATRIX- POSITIVE SEMI-DEFINITE eigenvalues are non-negative

- NEGATIVE SEMI-DEFINITE eigenvalues are non-positive
- POSITIVE DEFINITE eigenvalue are positive
- NEGATIVE DEFINITE eigenvalues are negative

GRADIENT $\nabla f = \left[\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \quad \dots \quad \frac{\partial f}{\partial x_m}\right]$

HESSIAN $\nabla^2 f = \begin{bmatrix} \frac{\partial f^2}{\partial x_1^2} & \frac{\partial f^2}{\partial x_1 \partial x_2} & \dots \\ \vdots & & \\ \frac{\partial f^2}{\partial x_m \partial x_1} & & \frac{\partial f^2}{\partial x_m^2} \end{bmatrix}$

* In a convex set every local minimum is also a global minimum

PROCEDURE

(1) Solve the gradient of the function equal to zero → Obtain candidate points (STATIONARY POINTS)

(2) Obtain the Hessian of the function and evaluate it at each candidate points

* Unless we know the shape of the function being minimized or can determine whether it is convex, WE CANNOT SAY WETHER A POINT IS THE GLOBAL MINIMUM

GENERAL SCHEME OF UNCONSTRAINED NLP ALGORITH

① Choose a search direction d^k (start with a basic solution)

② Minimize along that direction to find a new point $\underline{x}^{k+1} = \underline{x}^k + \alpha^k \underline{d}^k$

where k is the current iteration number
α^k is a positive scalar → STEP SIZE

For calculating the step-size we need:
- minimize the function $f(\underline{x}^{k+1}) = f(\underline{x}^k + \alpha^k \underline{d}^k)$
- Set $\frac{df(\underline{x}^k + \alpha^k \underline{d}^k)}{d\alpha^k} = 0$

For finding the minimum we use the iteration step: $\alpha^{k,h+1} = \alpha^{k,h} - \frac{\varphi'(\alpha^{k,h})}{\varphi''(\alpha^{k,h})}$

⇒ THIS IS KNOWN AS NEWTON METHOD FOR SCALAR FUNCTION (→ the tangent)

$\alpha^c = \alpha^b - \frac{\varphi'(\alpha^b)\cdot(\alpha^b - \alpha^a)}{\varphi'(\alpha^b) - \varphi'(\alpha^a)}$ ⇒ REGULA-FALSI METHOD (tipo bisezione)

TYPE OF NLP ALGORITHMS

COMPUTATIONAL HEAVIER

- TYPE 0: Require only the evaluations of the function
- TYPE 1: Require the evaluation of the function and it's gradient
- TYPE 2: Require the evaluation of the function, it's gradient and it's Hessian

NELDER-MEAD ALGORITHM
→ IS A TYPE 0 algorithm

Ⓐ It's is a heuristic search method (convergence is not guaranty)

Ⓦ It's a generalization of a tetrahedron to arbitrary dimensions (convex hull of k+1 vertices)

Given: n+1 vertices x_i $i: 1 \dots n+1$
Define this coefficient:
- REFLECTION
- CONTRACTION
- EXPANSION
- SHRINKAGE

① Sort by function value: $f_1 < f_2 < \dots < f_{n+1}$

Calculate $x_m :=$ average of the first n points

② Reflection: Compute $x_r := x_m + R(x_m - x_{n+1})$

IF $f_1 < f_R < f_n$ then replace the worst point x_{n+1} with x_R

if the point it much better $f_R < f_1$ → Expantion
if instead is worst than $f_n < f_R$ then I use → Contraction

⑥ Shrinkage: evaluate f at the n points the go to step 1

⇒ DON'T REQUIRE HYPOTHESIS OF THE FUNCTION JUST TO COMPUTE THE VALUES

this algorithm is fairly efficient at each iteration, but take a lot of iterations, sometime is flakey and simple

THE GRADIENT ALGORITHM
→ IT IS A TYPE 1 ALGORITHM
use the gradient as the search direction

$\underline{d}^k = \begin{Bmatrix} + \\ - \end{Bmatrix} \nabla f(\underline{x}^k)$ for $\begin{Bmatrix} \max \\ \min \end{Bmatrix}$

so $\underline{x}^{k+1} = x^k \begin{Bmatrix} + \\ - \end{Bmatrix} \alpha^k \nabla f(\underline{x}^k)$

the gradient at a point is the rate of change of the function at that point

① Choose an initial point x^0

② Compute the gradient $\nabla f(\underline{x}^k)$, where k is the iteration number

③ Compute the search vector $\underline{d}^k = \pm \nabla f(\underline{x}^k)$

④ Compute $\underline{x}^{k+1}$ $\underline{x}^{k+1} = \underline{x}^k + \alpha^k \underline{d}^k$ using mono-dimensional search method to find α^k

⑤ Determine convergence $|f(\underline{x}^{k+1}) - f(\underline{x}^k)| < \varepsilon_1$ or the magnitude $\|\nabla f(\underline{x}^k)\| < \varepsilon_2$

PROPERTIES OF THIS ALGORITHM
- RATE OF CONVERGENCE 1 $\|x_{k+1} - x^*\| \leq a \|x_k - x^*\|$ (linear convergence)
- IT HAVE THE CAPTURE PROPERTY", it means converges to a stationary point

ISSUE → ZIG-ZAGGY

ISSUE → ZIG-ZAGGY

* for quadratic function convergence speed depends on ratio of the highest second derivative ("condition number")

NEWTON-RAPHSON ALGORITHM

→ TYPE 2 ALGORITHM

It's a generalization of the NEWTON METHOD (monodimensional search)

consider φ depending only on one parameter → Minimize the second order Taylor expansion

Instead of having the taylor expansion we have the Hessian

$$\nabla g(\underline{x}) \approx \nabla g(\underline{x}^0) + \nabla^2 g(\underline{x}^0) \cdot (\underline{x} - \underline{x}^0) \Rightarrow \underline{x} = \underline{x}^0 - \left[\nabla^2 g(x^0)\right]^{-1} \nabla g(\underline{x}^0)$$

Generic iteration $\underline{x}^{k+1} = \underline{x}^k - \left[\nabla^2 g(\underline{x}^k)\right]^{-1} \nabla g(\underline{x}^k)$

(only we can applicate this if the Hessian is invertible)

This method:

- Converge much faster than the gradient algorithm → RATE OF CONVERGENCE $\|\underline{x}^{k+1} - \underline{x}^0\| \leq a \|x^k - x^0\|^2$
- Computational heavier
- Cannot be applied when the Hessian is singular and when it has small eigenvalues (numerical instability → overflow)
- Has no "capture properties"

* Typically this 2 method are combined (GRADIENT → N-P when we are near to the point)

① Choose a starting point x^0

② Compute the gradient and then the Hessian

③ Compute x^{k+1}

This algorithm reduce the number of iterations but increase the computation