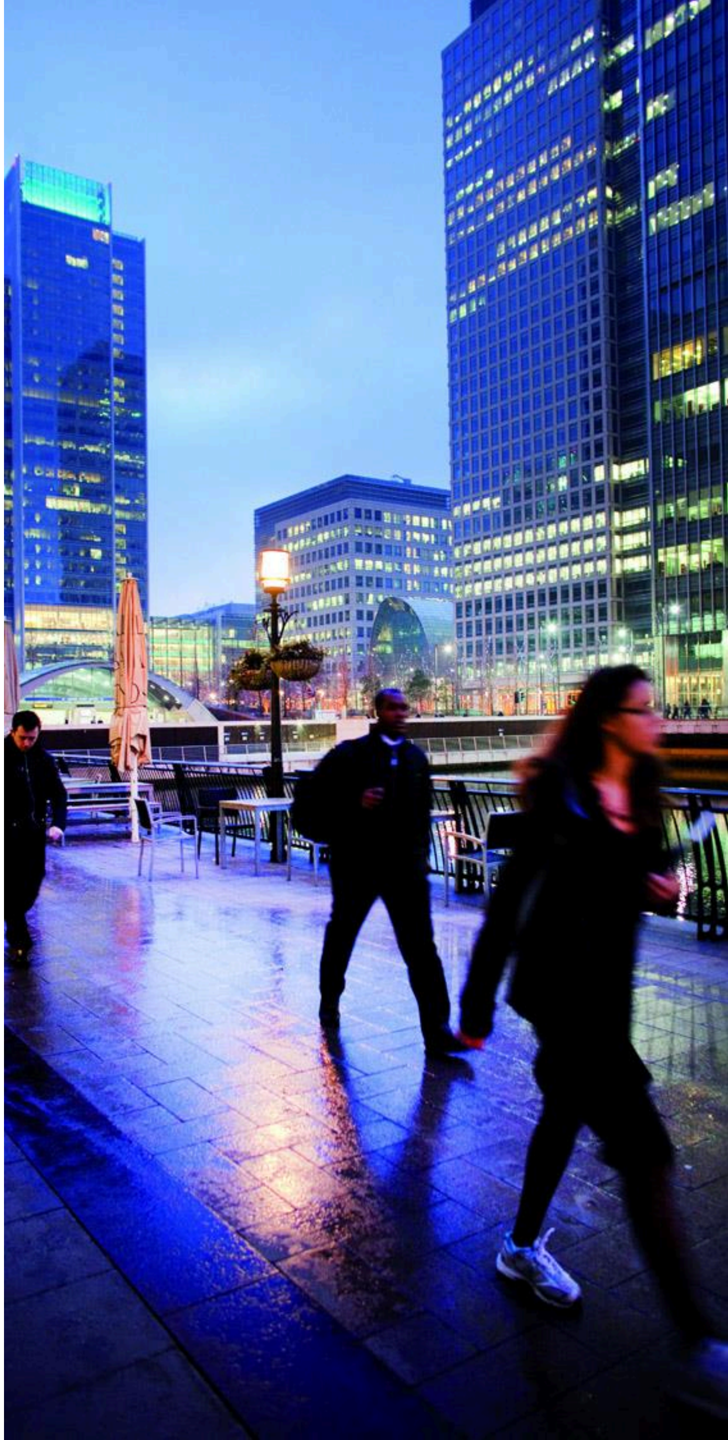
BASICS OF OPTIMISATION METHODS

Jean-Yves LUCAS
Jean-yves.lucas@edf.fr

CEA-EDF-INRIA Summer school
Design and optimization under uncertainty of large-scale numerical models

July 3rd, 2017



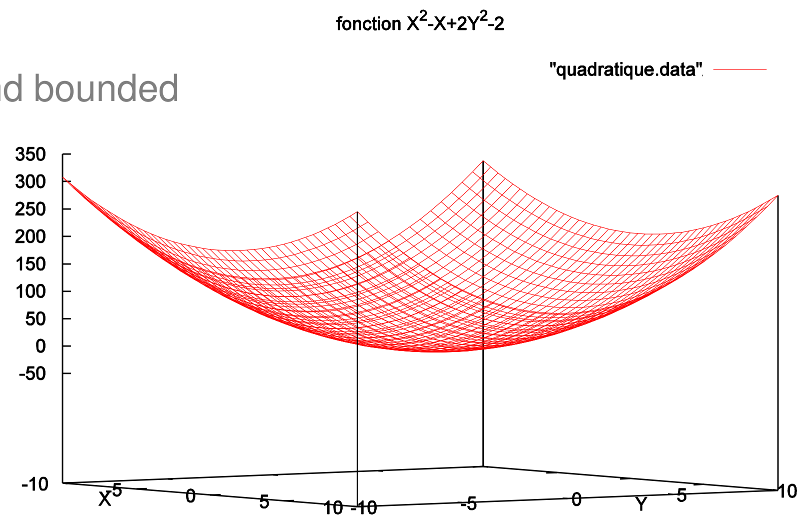
SUMMARY

INTRODUCTION

- **Combinatorial problem**
  - Involves only discrete variables
- **Constraint Satisfaction Problems**
  - Typically defined by
    - A set of discrete variables $V_1, \ldots, V_n$
    - A domain Di of possible values for each variable V_i
    - A set of constraints $K_1, \ldots, K_p$
  - Example :
    - Let the variables a, b, c, d
    - let $a \in \{0,1,2,3,4,5,6,7,8,9\}$
    - let $b \in \{0,1,2,3,4,5,6,7,8,9\}$
    - let $c \in \{0,1,2,3,4,5,6,7,8,9\}$
    - let $d \in \{0,1,2,3,4,5,6,7,8,9\}$
    - Subject to: $a + b + c + d = 2017 - (1000\, a + 100\, b + 10\, c + d)$
- **But in practice some (or all) variables may take real values**
  - Mixed-integer Problems
  - Continuous Problems

OVERVIEW

- **Optimization problem**
  - We are looking for a point in the domain of a function so that the function value is minimal (resp. maximal) at that point.
  - Typically :
    - variables $V_1, \dots, V_n$ are continuous and bounded
  - Example :

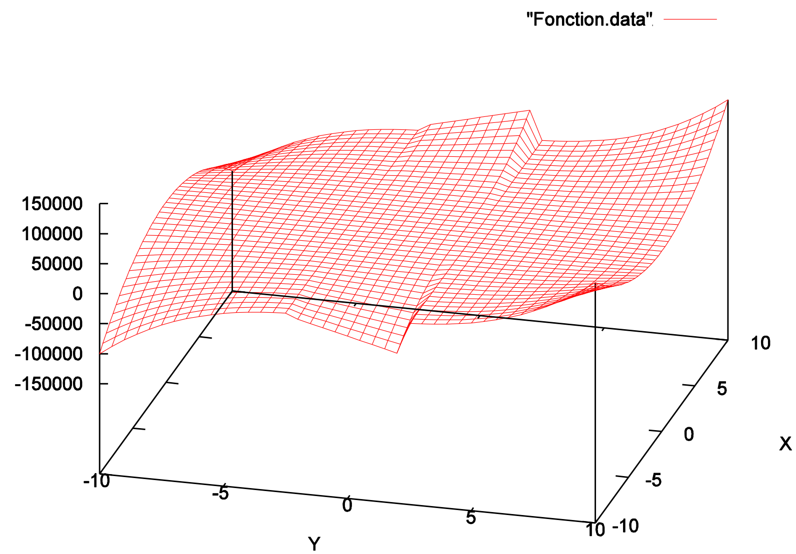


    - let the variables x, y
    - let $x \in [-10, 10]$
    - let $y \in [-10, 10]$

- **If possible we take advantage of continuity and derivability properties of the function**
  - In simple cases (convex), we look for a point at which the 1st order partial derivatives equal 0
  - Use of Newton method or variations (quasi-Newton, truncated-Newton…)

OVERVIEW

- **Optimization problem**
  - Harder cases…
  - Example :
    - Let variables x, y
    - let $x \in [-10, 10]$
    - let $y \in [-10, 10]$



- **We may reach local minima**
- **If possible we break the domain down into sub-domains on which the function is convex (resp. concave)**
  - We also pay attention to 2nd order partial derivatives…

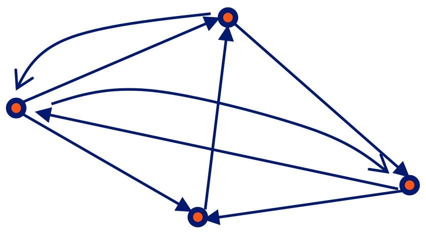
OVERVIEW

- **Constrained optimization problem**
  - Both an optimization problem…
  - And Constraint Satisfaction Problem (CSP)
  - Moreover if some variables are discrete and the others are continuous :
    - Mixed-Integer Constained Satisfaction Problem
  - Typically :
    - variables $V_1, \ldots, V_n$ are either continuous or discrete
    - Feasible solutions are defined through a set of constraints (including bound constraints)
    - Some variables shall take integer values
  - Example :
    - A power plant whose production is computed on a set of successive time-steps
    - At each time-step, the plant is operating or not
    - If the plant produces electricity, then the produced power ranges between Pmin and Pmax
    - When producing, the plant must be in operations for at least 3 consecutive time-steps

COMPLEXITY

- **Complexity (in time)**
  - Number of elementary instructions that are necessary to perform in order to carry the algorithm out, as a function of data size
  - Optimization : two notions :
    - Existence problems: Is there a solution that fulfills all constraints ?
    - Decision problems (optimization problems) : what is the problem's best solution ?
    - These problems are equivalent : Is there a solution so that any other solution has a greater cost (minimization case) ?

- **Polynomial problems**
  - The number of elementary instructions performed is bounded by a multivariate polynomial (data size)
  - Example: shortest path (Dijktra) : O((*a*+*n*)*Log(*n*)) for a graph with *a* edges and *n* vertices



COMPLEXITY

- **Exponential problems**
  - The number of elementary instructions performed is bounded by a power function of the data size
  - The proved exponential problems are scarce…
    - We have to build an exponential number of solutions
      - For instance : the subsets of a set of cardinality n
    - Others…

- **What lies between polynomial problems and exponential problems ?**
  - Problems that have not been proven to be exponential…
  - But for which we do not know any polynomial algorithm.

COMPLEXITY

- **NP problems : non-deterministic polynomial**
  - There exists a polynomial procedure to build a possible solution and to insure it is actually a solution
  - We will have to run this procedure an undetermined number of times so as to finally get a solution. This number of times is not known in advance, but is bounded by an exponential function of the data size
  - Example :
    - let $a \in \{0,1,2,3,4,5,6,7,8,9\}$
    - let $b \in \{0,1,2,3,4,5,6,7,8,9\}$
    - let $c \in \{0,1,2,3,4,5,6,7,8,9\}$
    - let $d \in \{0,1,2,3,4,5,6,7,8,9\}$
    - Subject to $a + b + c + d = 2017 - (1000\,a + 100\,b + 10\,c + d)$

      – choose a possible value for a, b, c and d,
      – compute a+b+c+d,
      – compute 2017 - (1000 a + 100 b + 10 c + d)
      – Check for equality

⇨ polynomial procedure, but should be run an undetermined number of times (here at most 10^4 i.e. $NbValues^{NbVariables}$) until one (resp. the best resp. no) solution is found

⇨ 10^4 is an upper bound, some (polynomial) methods may be used in order to reduce (with no guarantee) the search space

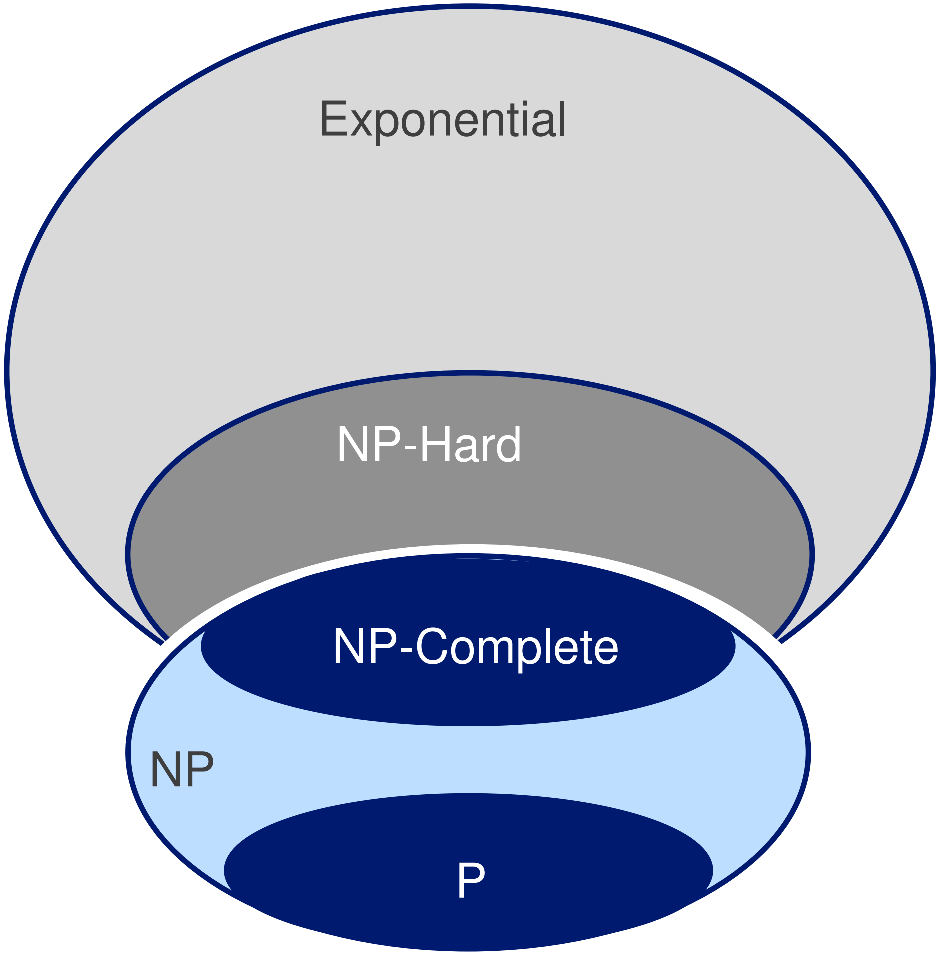
COMPLEXITY

- **NP-complete problems:**
  - NP problems : there exists a polynomial algorithm to check a solution
  - Any NP-problem reduces to a NP-complete problem through a polynomial reduction : a polynomial procedure that transforms the NP problem into an instance of a NP-complete problem,
    - i.e. finding a solution to the NP-problem, comes to finding a given solution (with respect to the reduction) to the NP-complete problem

      ⇒ the NP-complete problem is « at least as hard as » the NP-problem
  - The NP-complete problems are in NP, thus they are all reducible to each other.
  - They are all « equivalent »: if there exists a polynomial algorithm solving one NP-complete problem, then all NP-complete problems are polynomial.

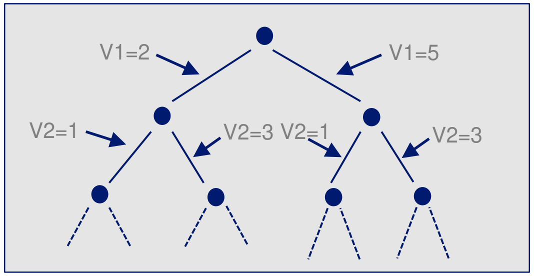
- **NP-hard problems:**
  - A problem Prob is NP-hard if any NP-problem (including NP-complete ones) is reducible to Prob thanks to a polynomial reduction.
    - Thus they are « at least as difficult as » NP- problems

COMPLEXITY

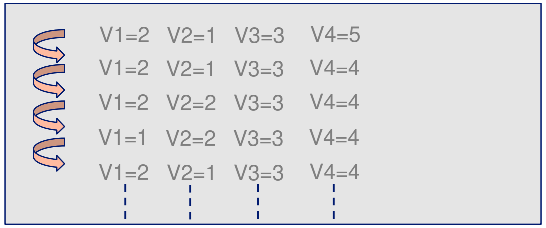


METHOD CLASSIFICATION

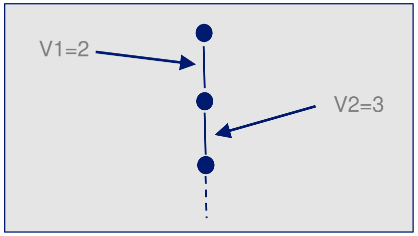
- **A first taxonomy**
  - Enumerative methods
    - Have a look (at least implicitly) at the whole search space
    - Scan a search tree
    - E.g. Branch and Bound, Constraint Programming, Dynamic Programming



  - Iterative methods
    - Build an initial solution
    - Repeat the process of changing some variables assignment so as to get a new solution in the neighborhood of the previous one
    - Until a satisfactory or the best solution is found
    - E.g. hill-climbing methods, simplex, tabu search, genetic algorithm

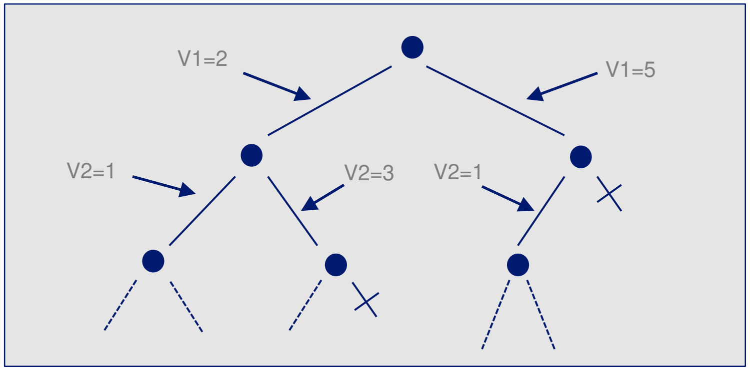


  - Constructive methods
    - Repeat the process of
      - Choose a variable among those not having been yet assigned thanks to a specific criterion Cvar
      - Choose a possible value for this variable according to a specific criterion Cval
    - Until a solution is found or a constraint is violated
    - No backtrack
    - E.g. greedy algorithm



METHOD CLASSIFICATION

- **An other taxonomy**
- **For difficult problems (non-polynomial),**
  - We may distinguish two classes of methods:
    - If we absolutely want to get a solution or prove there's no solution, we need to backtrack
    - Repeat the process of
      - assigning a value to each problem's variable
      - Check the constraints
    - until a solution is discovered or the whole search space has been scanned
    - 1st : exact methods (Branch and Bound, CP) :
      - build a search tree,
      - Avoid to scan irrelevant sub-trees



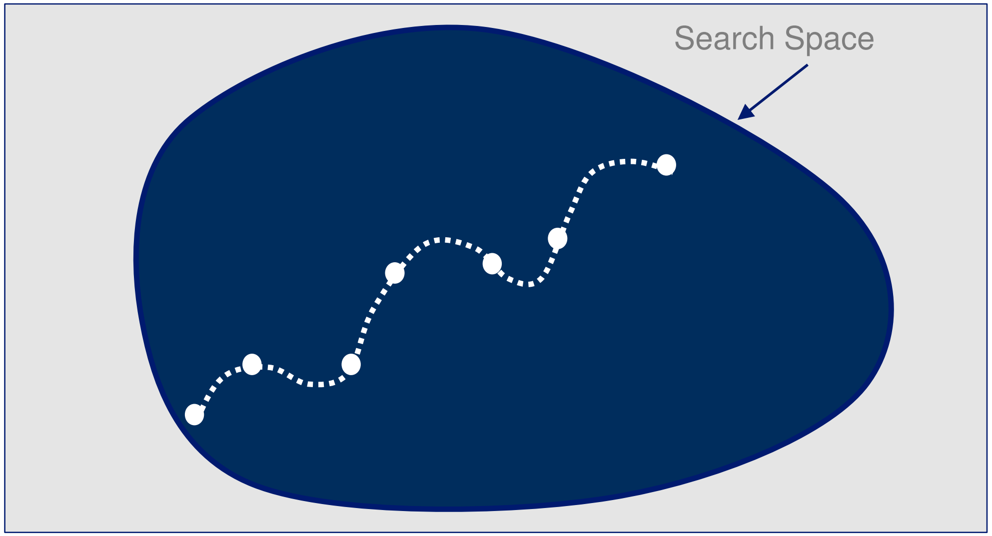
      - Guarantee to find out one (the best) solution
      - No guarantee on the running time

METHOD CLASSIFICATION

- **For difficult problems (non-polynomial),**
  - We may distinguish two classes of methods:

    - 2nd : approximate methods (heuristics and metaheuristics) :

    - build one solution (possibly repeat that process with randomness)
    - only scan a reduced part of the search space



– These methods are polynomial

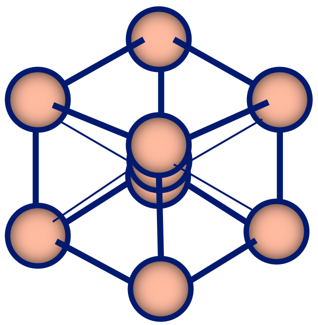
METHOD CLASSIFICATION

- **Heuristics et métaheuristics**
  - Trade-off : swap from a guarantee on finding solutions to a guarantee on bounding the execution time
  - Choosing these methods depend on the problem at hand :
    - Is it easy to find solutions, but hard to find the best ones ?
    - Is finding the best solution mandatory ?
    - Is it enough to get good solutions ?
  - heuristics :
    - Only build one solution according to appropriate criteria,
    - possibly repeat the process with some randomness
  - metaheuristics :
    - Exhibit a set of solutions according a general scheme, which is adapted to the problem to solve
    - This scheme rely on analogy with physics (simulated annealing), biology (genetic algorithms) or social animals behaviour (ant colony algorithms, particle swarm optimization)

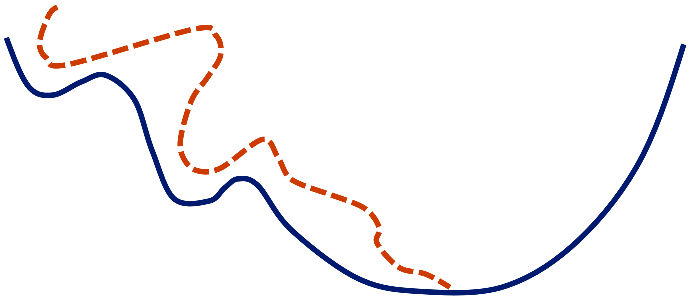
SOME METAHEURISTICS

- **Simulated annealing**
  - Analogy with metallurgy
    - While cooling down, metallic atoms position themselves according to a given structure (e.g. body-centered cubic, face-cubic centered, …)
    - A too fast cooling process lead to non-homogeneous structure

      ⟶ weaknesses
    - The energy level is not minimum
    - Hence successive annealing
      - To lower and lower temperatures
      - bring enough energy for a better atoms positionning



  - same principle :
    - Gradient method
    - Start from temperature T0
    - From time to time we accept a worse solution than the former one (« annealing »)
    - According to a probability depending on the « temperature » : let S^{n+1} a possible successor solution to S^n
      - $P(S^{n+1}) = 1$ if $f(S^{n+1}) < f(S^n)$
      - $P(S^{n+1}) = e^{-(\Delta f/T)}$ otherwise
    - Decrease temperature to 0.



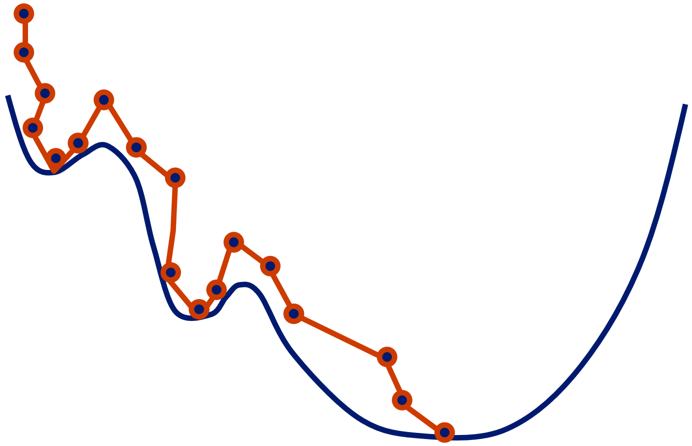
SOME METAHEURISTICS

- **tabu**
  - Gradient method with list management

    - Store the last n solutions we found
    - As long as a neighbour solution improves the objective function, choose it
    - If none, then choose one which deteriorate the cost BUT…
    - …**not among those being in the tabu list**
    - Update tabu list

    - Avoid infinite loops of size less or equal to n
    - But the choice of an appropriate n is crucial

Example with n = 5



SIMPLEX ALGORITHM

- Continuous problems
  - A widely used algorithm: simplex
  - Linear program

    $A \times X \leq B$
    with $X \in \mathbb{R}^n$
    A $m \times n$ matrix
    X $1 \times n$ vector
    B $m \times 1$ right-hand side vector
  - Objective function

    $\max (\delta_1 x_1 + \ldots + \delta_n x_n)$

  - Standard form:

    $\max (\delta_1 x_1 + \ldots + \delta_n x_n)$
    $A' \times X' = B$
    A $m \times (n+m)$ matrix
    X $1 \times (n+m)$ vector
    B $m \times 1$ right-hand side vector
    $\forall i \in [1, n+m] \;\; x_i \geq 0$

    by adding m slack variables

SIMPLEX ALGORITHM

Linear programs properties

- Example :
  - Minimize $7\,x_1 + x_2 + 5\,x_3$
  - Subject to constraints

$$
\begin{aligned}
x1 - x2 + 3\,x3 &\geq 10 \\
5\,x1 + 2\,x2 - x3 &\geq 6 \\
x1, x2, x3 &\geq 0
\end{aligned}
$$

  - Question : what is the value z*, minimum of $7\,x_1 + x_2 + 5\,x_3$
  - Remark :

$$7\,x_1 + x_2 + 5\,x_3 \geq (x_1 - x_2 + 3\,x_3) + (5\,x_1 + 2\,x_2 - x_3) \geq 16$$

SIMPLEX ALGORITHM

- Linear programs properties
  - The « game » consists in finding positive multiplicators for constraints so that the coeffficient associated with each variable in the sum of constraints left-hand side be less but as close as possible to the coefficient associated to the same variable in the objective function.
  - The sum of right-hand sides is a lower bound of z*

$$x_1 - x_2 + 3x_3 \geq 10$$

$$5x_1 + 2x_2 - x_3 \geq 6$$

$$\text{Minimize } 7x_1 + x_2 + 5x_3$$

SIMPLEX ALGORITHM

- Linear programs properties

$$x_1 - x_2 + 3x_3 \geq 10$$

$$5x_1 + 2x_2 - x_3 \geq 6$$

$$\text{Minimize } 7x_1 + x_2 + 5x_3$$

$$1.5\,(x_1 - x_2 + 3x_3) \geq 15$$

$$5x_1 + 2x_2 - x_3 \geq 6$$

$$7x_1 + x_2 + 5x_3 \geq 6.5x_1 + 0.5x_2 + 3.5x_3 \geq 21$$

SIMPLEX ALGORITHM

- Linear programs properties
  - Let 2 coefficients y_1 et $y_2 \geq 0$

$$\begin{aligned} &\text{Minimize } 7x_1 + x_2 + 5x_3 \\ &y_1(x_1 - x_2 + 3x_3) \geq 10\,y_1 \\ &y_2(5x_1 + 2x_2 - x_3) \geq 6\,y_2 \end{aligned}$$

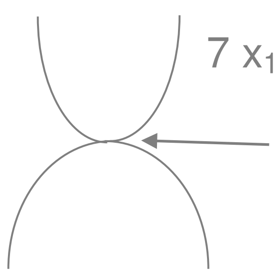
$$\left.\begin{aligned} x_1 - x_2 + 3x_3 &\geq 10 \\ 5x_1 + 2x_2 - x_3 &\geq 6 \\ \text{Minimize } 7x_1 + x_2 + 5x_3 & \end{aligned}\right\} \quad \begin{aligned} &\text{Maximize } 10\,y_1 + 6\,y_2 \\ &y_1 + 5y_2 \leq 7 \\ &-y_1 + 2y_2 \leq 1 \\ &3y_1 - y_2 \leq 5 \end{aligned}$$

SIMPLEX ALGORITHM

- Linear programs properties

$x_1 - x_2 + 3x_3 \geq 10$	Maximize $10y_1 + 6y_2$
$5x_1 + 2x_2 - x_3 \geq 6$	$y_1 + 5y_2 \leq 7$
Minimize $7x_1 + x_2 + 5x_3$	$-y_1 + 2y_2 \leq 1$
	$3y_1 - y_2 \leq 5$
Solution $(7/4, 0, 11/4)$	Solution $(2,1)$

In $\mathbb{R}$

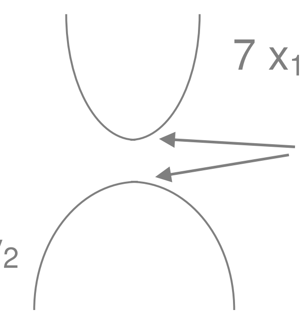


$7x_1 + x_2 + 5x_3$

$Z^* = 26$

$10y_1 + 6y_2$

In $\mathbb{Z}$



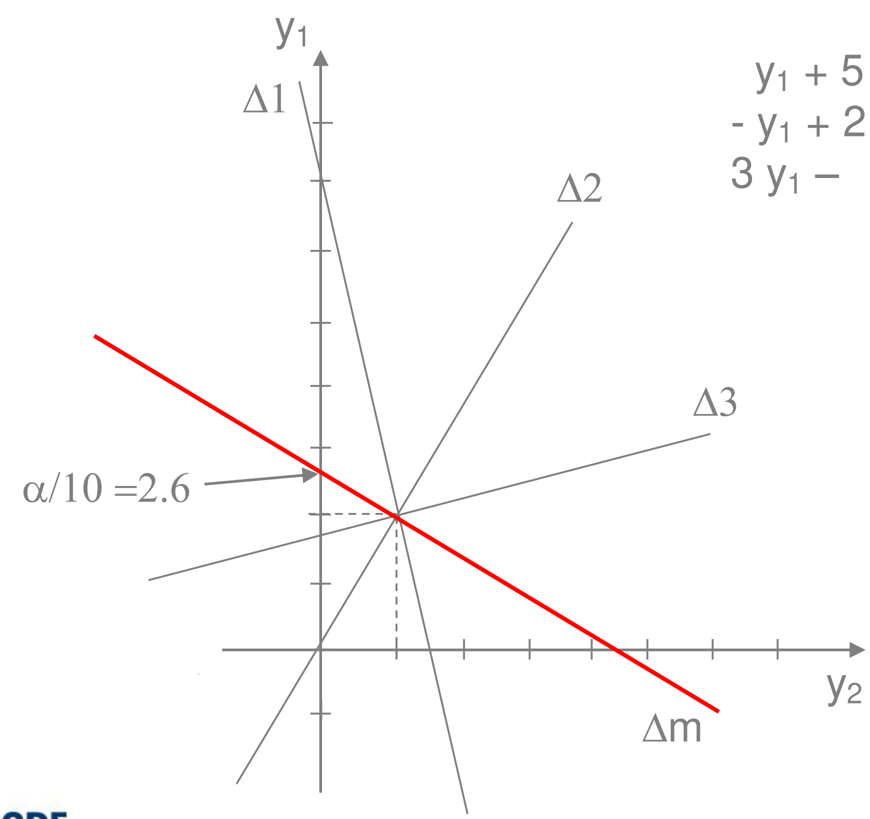
$7x_1 + x_2 + 5x_3$

$Z^* = ?$

$10y_1 + 6y_2$

SIMPLEX ALGORITHM

Geometrical insight



$$\left.\begin{array}{r} y_1 + 5\,y_2 \le 7 \\ -\,y_1 + 2\,y_2 \le 1 \\ 3\,y_1 - \quad y_2 \le 5 \end{array}\right\} \quad \begin{array}{ll} y_1 = -5\,y_2 + 7 & (\Delta 1) \\ y_1 = -2\,y_2 - 1 & (\Delta 2) \\ y_1 = 1/3\,y_2 + 5/3 & (\Delta 3) \end{array}$$

maximize $(10\,y_1 + 6\,y_2)$

let: $10\,y_1 + 6\,y_2 = \alpha$

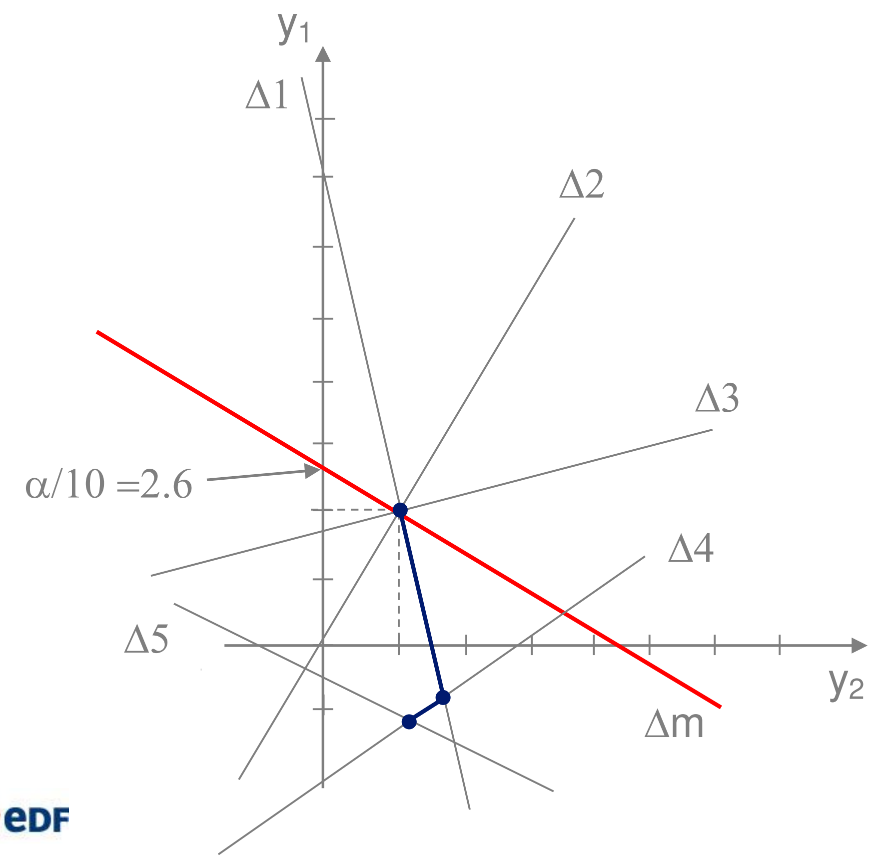
$\Rightarrow$ maximize α

$y_1 = -\,6/10\,y_2 + \alpha/10 \; (\Delta m)$

SIMPLEX ALGORITHM

Simplex algorithm

- start from an initial point on a vertex of the polytope
- Move toward a neighbour vertex following an edge, so that the value of the objective function is improved



maximize $(10\ y_1 + 6\ y_2)$

OPTIMIZATION UNDER UNCERTAINTY

- Optimizing a function in the presence of randomness in the optimization process

  - The uncertainty may lie
    - in the objective function
    - in the constraints

  - The randomness affects the data
    - the coefficients associated to decision variables
      - » In the constraints
      - » In the objective function
      - » both

OPTIMIZATION UNDER UNCERTAINTY

- Optimizing a function in the presence of randomness in the optimization process

  - Different approaches

    - chance constrained optimization

    - stochastic optimization

OPTIMIZATION UNDER UNCERTAINTY

- chance constrained optimization
  - in the presence of random data some constraints are not mandatorily fulfilled
    - Let f be a left-hand side of a constraint
    - Let x be the set of decision variables
    - Let ω be set of random data (e.g. scenarios)
    - a chance constraint:
      - » $Prob(f_i(x, \omega) \leq 0) \geq \eta$ (η is the level of confidence)
    - Percentile optimization
      - » minimize γ
      - » $Prob(f_i(x, \omega) \leq \gamma) \geq \eta$
    - able to cope with probability laws
    - may be non-linear, non-convex…
    - usually results in difficult optimization problems

OPTIMIZATION UNDER UNCERTAINTY

- stochastic optimization
  - Usually randomness leads to scenarios
    - From historical data
    - From Monte-Carlo simulations
    - decisions have to be made over time periods
    - a prominent division:
      - single-stage stochastic optimization
      - multi-stage stochastic optimization

OPTIMIZATION UNDER UNCERTAINTY

- stochastic optimization

  - single-stage problems
    - decision is implemented with no subsequent recourse
    - X set of all possible decisions
    - ξ random information only available after decision is made
    - $F(X, \xi)$ cost function
    - we do not directly optimize $F(X, \xi)$
    - instead we minimize $\mathbb{E}[F(X, \xi)]$

    - the general single-stage optimization problem becomes:
    - $\zeta^* = \min_{x \in X} \{f(x) = \mathbb{E}[F(X, \xi)]\}$

    - assume that X is convex and $F(X, \xi)$ is convex in X for any realisation ξ
    - otherwise subdivide the domain into pieces where convexity is met

OPTIMIZATION UNDER UNCERTAINTY

- stochastic optimization

  - multi-stage optimization
    - aims at finding a sequence of decisions at successive steps $t \in [0, T]$
      - may correspond to temporal or decisional chronology
    - ξ random information available after partial decisions are made
    - $F(X, \xi)$ cost function
    - we do not directly optimize $F(X, \xi)$

    - the general multi-stage optimization problem is:

    - $\zeta^* = \min_{x_0 \in X0} \mathbb{E}[\inf_{x_1 \in X1} F(X_1, \xi_1) + \mathbb{E}[\ldots + \mathbb{E}[\inf_{x_T \in X_T} F(X_T, \xi_T)]]] \}$
    - X_i : decisions made at stage i
    - an important application : 2-stage optimization with recourse
      - 1st stage : structural, « here and now » variables
      - 2nd stage : recourse, « wait and see » variables
      - Here and now variables have the same assignment in any scenario
      - Wait and see variables may have different assignments in each scenario

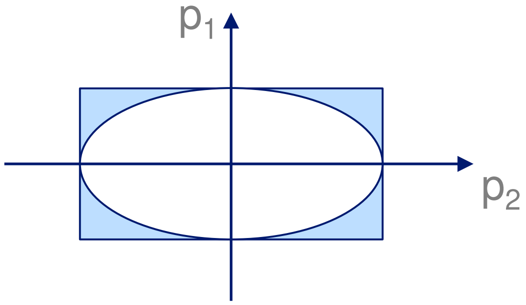
OPTIMIZATION UNDER UNCERTAINTY

- stochastic optimization

  - Robustness
    - optimizing in average or expectation may lead to desastrous situation for some « unlucky » scenarios.
    - hence the notion of robustness

      » either you add terms in the objective function (the variance)
      - Min $c^T x + \sum_{k=1}^{K} p_k z_k + K[\sum_{k=1}^{K} p_k {z_k}^2 - (\sum_{k=1}^{K} p_k z_k)^2]$

      » or you add terms in the constraints
      - « you cannot be so unlucky as all misfortunes occur at the same time »
      - let p_1 and p_2 two random parameters
      - add quadratic contraints such as $\frac{{P_1}^2}{a^2} + \frac{{P_2}^2}{b^2} = r$
      - this yield difficult (non-linear) problems



OPTIMIZATION UNDER UNCERTAINTY

optimization methods

- Stochastic dual dynamic programming
- MIPs
- Decomposition methods
  - decompose the problem according to the scenarios
    - that relax the coupling constraints (non-anticipativity constraints stating that for instance in a 2-stage problem, the 1st-stage variables should take the same value in each scenario)
    - make sure that at the end of the solving process, the non-anticipativity constraints are fulfilled
  - Cutting-plane methods
  - Augmented-lagrangian methods
  - and so on…
- Monte-Carlo simulations
- Metaheuristics
- …

BLACK-BOX OPTIMIZATION

- In some cases the objective function is not known analytically
  - Or the function is rather complex (e.g. physical and chemical conditions inside a boiler in operation)
- Need to run a black-box software to evaluate solutions
- Any evaluation may be cheap (favourable case) or costly (try a surrogate model ?)
- A block-box function is not always a nasty one (continuous, smooth, convex…)
- Need to distinguish the process of generating candidate solutions and assessing their relevance (correctness, evaluation…)
  - Methods :
    - Genetic algorithm
    - Particle swarm
    - Descent/gradient
    - Local search
    - Variable neighbourhood search
    - DIRECT (Dividing RECTangles)
  - Somehow constuct a response surface
  - Balance between intensification and diversification

Thank you