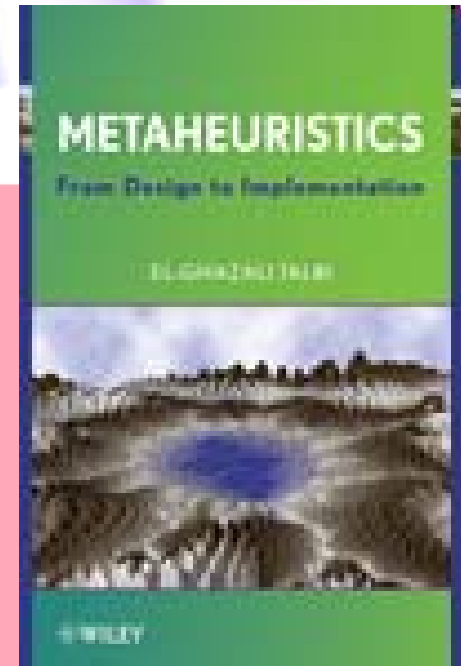


Metaheuristics : from Design to Implementation

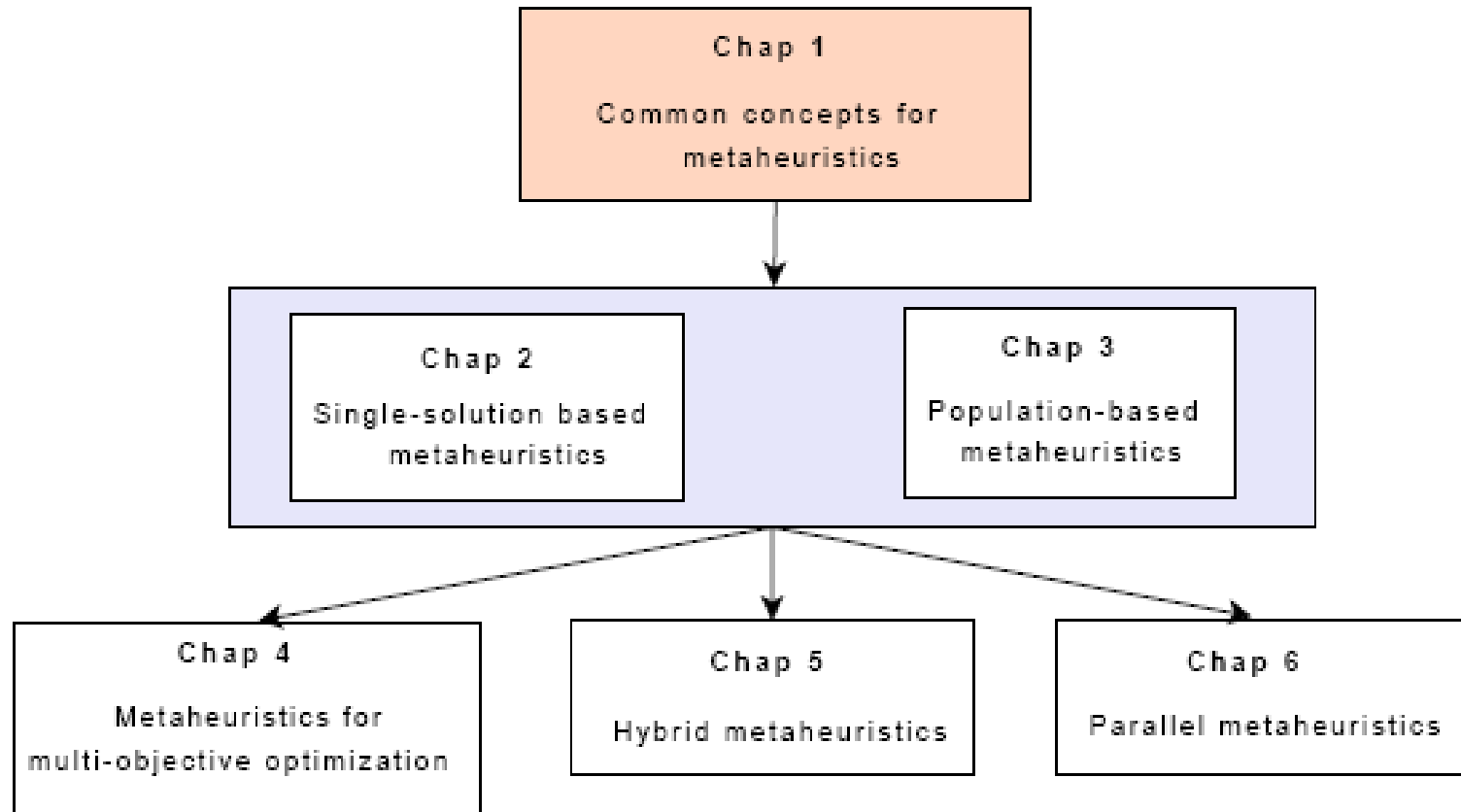
E-G. Talbi

Professor – Polytech'Lille -
Team manager –INRIA – University of
Lille - CNRS –
France



Wiley, 2009 (596pp)
ISBN: 978-0-470-27858-1

Outline



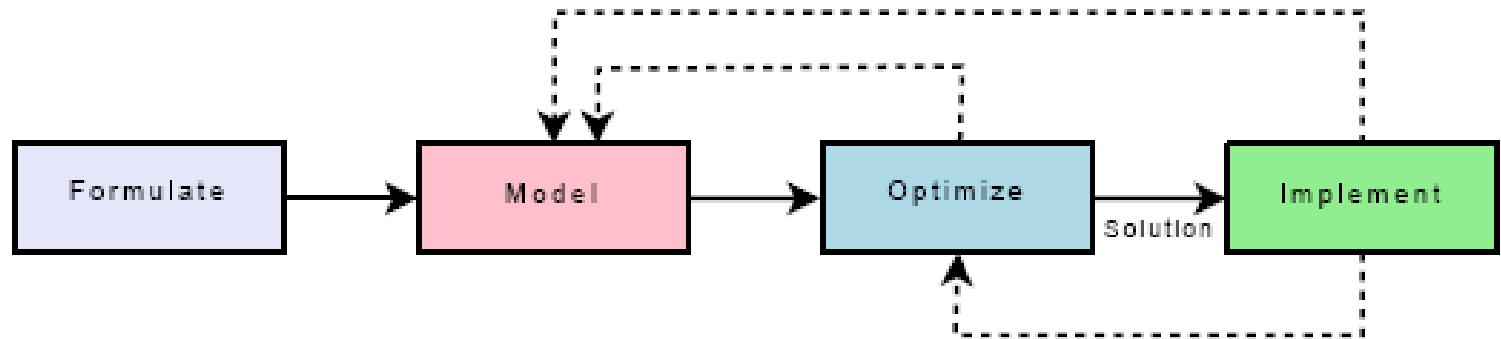
Outline of the book

- Common concepts for Metaheuristics
- Single-solution based metaheuristics
 - Common concepts for S-metaheuristics
 - Local search
 - Landscape analysis
 - Advanced local search (Simulated annealing, Tabu search, VNS, ILS, GLS, ...)
- Population-based metaheuristics
 - Common concepts for P-metaheuristics
 - Evolutionary algorithms (Genetic algorithms, GP, ES, EDA, ...)
 - Swarm intelligence: Ant colonies, Particle swarm, ...
 - Bess, Immune systems, ...
- Metaheuristics for Multi-objective Optimization
- Hybrid Metaheuristics
- Parallel metaheuristics

Outline of the chapter 1

- Optimization models
- Optimization methods
 - Exact algorithms / Approximate algorithms
 - When using metaheuristics?
 - Greedy heuristics
- Main common concept of metaheuristics
 - Representation
 - Objective function
- Constraint handling
- Parameter tuning
- Performance analysis of metaheuristics
- Software frameworks for metaheuristics
 - ParadisEO framework

From modeling to decision making



- In practice, we find solutions for models representing problems
- Usually models are simplifications of the reality

Importance of models

Optimization problem

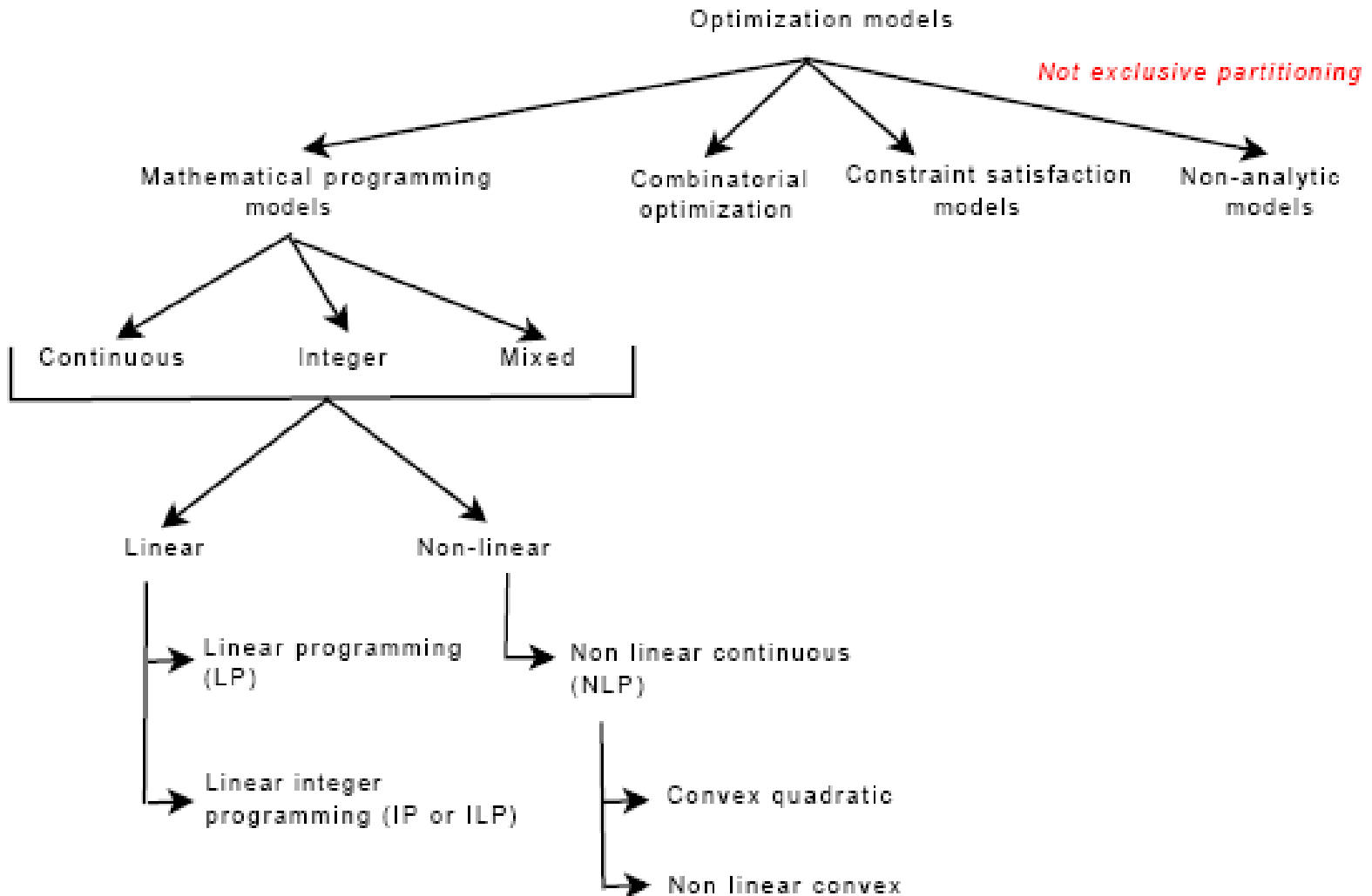
- **Definition** (minimisation problem) : couple (S, f)
 - Given a search space S which represents feasible solutions
 - Given an objective function $f: S \rightarrow R$
 - Find $s^* \in S$ such as :

$$f(s^*) \leq f(s) \quad \forall s \in S$$

s^* : **global optimum**

- Large scale and complex optimization problems in many areas of science and industry (**Telecommunications, Biology, Transportation-Logistics, Environment, Finance, Design, ...**).
- Problem size more and more important (combinatorial explosion) **and/or** Delays more and more reduced.

Optimization models



Complexity theory: Algorithms

Definition 1.2 Big- O notation: *an algorithm has a complexity $f(n) = O(g(n))$ if there exist positive constants n_0 and c such that $\forall n > n_0, f(n) \leq c \cdot g(n)$.*

- **Polynomial-time algorithms:** shortest paths, spanning tree, network flow, ...
- **Exponential-time algorithms**

Complexity	Size=10	20	30	40	50
$O(x)$	0.00001 s	0.00002 s	0.00003 s	0.00004 s	0.00005 s
$O(x^2)$	0.0001 s	0.0004 s	0.0009 s	0.0016 s	0.0025 s
$O(x^5)$	0.1 s	0.32 s	24.3 s	1.7 mn	5.2 mn
$O(2^x)$	0.001 s	1.0 s	17.9 mn	12.7 days	35.7 years
$O(3^x)$	0.059 s	58.0 mn	6.5 years	3855 centuries	2×10^8 centuries

Complexity theory: Algorithms

- Two other notations to analyze algorithms

Definition 1.5 Big- Ω notation: *an algorithm has a complexity $f(n) = \Omega(g(n))$ if there exist positive constants n_0 and c such that $\forall n > n_0, f(n) \geq c.g(n)$. The complexity of the algorithm $f(n)$ is lower bounded by the function $g(n)$.*

Definition 1.6 Big- Θ notation: *an algorithm has a complexity $f(n) = \Theta(g(n))$ if there exist positive constants n_0, c_1 and c_2 such that $\forall n > n_0, c_1.g(n) \leq f(n) \leq c_2.g(n)$. The complexity of the algorithm $f(n)$ is lower bounded by the function $g(n)$.*

Complexity theory: Problems

- **Complexity of a problem** = complexity of the best algorithm solving that problem
- **Tractable (easy)** = there exists a polynomial-time algorithm to solve it
- **Intractable** = if no polynomial-time algorithm exists
- **Most** of real-life problems are intractable

Decision / Optimization problems

- **Decision problem** has always « yes » or « no » answer

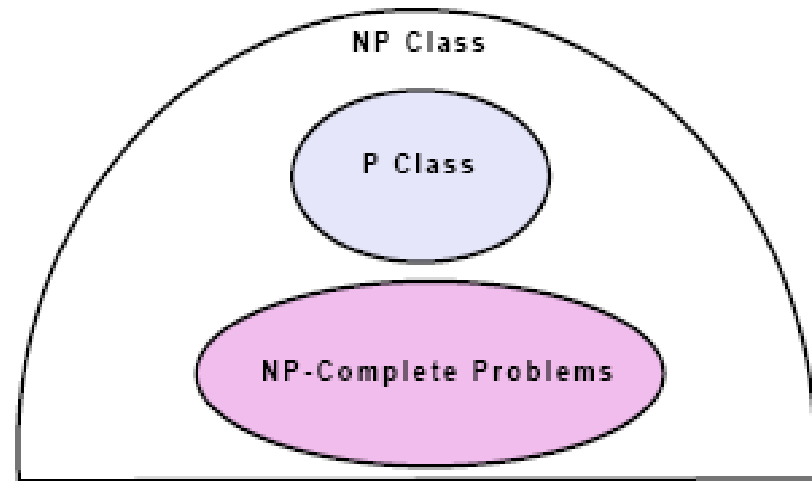
Example 1.5 Prime number decision problem: a popular decision problem consists in answering the following question: is a given number Q a prime number? It will return yes if the number Q is a prime one, otherwise the no answer is returned.

- **Optimization problem** can always be reduced to a decision problem

Example 1.6 Optimization versus decision problem: the optimization problem associated to the traveling salesman problem is: find the optimal Hamiltonian tour which optimizes the total distance. Whereas the decision problem is: given an integer D , is there a Hamiltonian tour with a distance less or equal to D ?

Complexity classes

- **P class**: decision problems solved by a deterministic machine in polynomial time
- **NP class**: decision problems solved by a non-deterministic algorithm in polynomial time
- **NP-complete**: if all problems of the class NP are reduced polynomially to the problem
- **P=NP?** Important open question



Complexity classes: Examples



Other models for optimization

- Optimization under **uncertainty**
 - Input data subject to noise
 - e.g. VRP – stochastic demands, travel times, ...
- **Dynamic** optimization
 - Deterministic objective function, varies over time
 - e.g. VRP – new demand → Track the optimal solution
 - **Multi-periodic** optimization: periodic change (change known a priori)
 - e.g. planning problem (traffic, incoming technology)
- **Robust** optimization: engineering design problems
- **Multi-objective** optimization problems

“Generic” problems

- Ex: **SAT**, **TSP**, and **NLP** are 3 canonical forms of models that can be applied to solve different problems.
- Quadratic Assignment Problem
- Bin Packing and Generalised Assignment Problems
- Hub allocation problems
- Graph Colouring & Partitioning
- Vehicle Routing
- Single & Multiple Knapsack
- Set Partitioning & Set Covering Problems
- Processor Allocation Problem
- Various Staff Scheduling Problems
- Job Shop Scheduling

Size of the search space

- **SAT Problem** : Boolean Satisfiability Problem (First NP-hard problem).

- Many applications : timetabling, routing, ...

- Find a set of boolean variables

$$X = (X_1, \dots, X_n)$$

such as the boolean expression $F = \text{TRUE}$

- Boolean expression $F =$ conjunction of clauses

- Clauses are disjunctions of k variables (k -SAT) :

$$F(X) = (\bar{X}_1 \vee X_3) \wedge (X_1 \vee \bar{X}_2) \wedge (X_4 \vee X_2) \wedge \dots$$

- For $n=100$, size of the search space = $2^{100} \approx 10^{30}$,

- Evaluation of 1000 solutions / sec ; 15 billions of years (Big Bang) to evaluate less than 1% of the search space.

- $k > 2$, NP-hard problem ; $k = 2$, Polynomial problem

Size of the search space

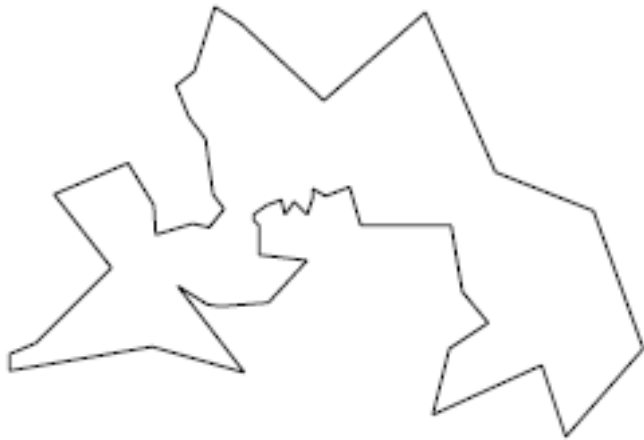
- **TSP Problem** : n cities, find a circuit which minimize the total distance ; all cities must be visited once.
- **Symmetric TSP** \longleftrightarrow $\text{dist}(x,y)=\text{dist}(y,x)$
- $|S| = n!/2n = (n-1)! / 2$; $n > 6$ TSP > SAT
- 10 cities = 181 000 solutions
- 20 cities = 10 000 000 000 000 000 solutions
- 50 cities = 100 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 solutions
- There is 1 000 000 000 000 000 000 000 litres of water in the planet

http://www.ing.unlp.edu.ar/cetad/mos/TSPBIB_home.html

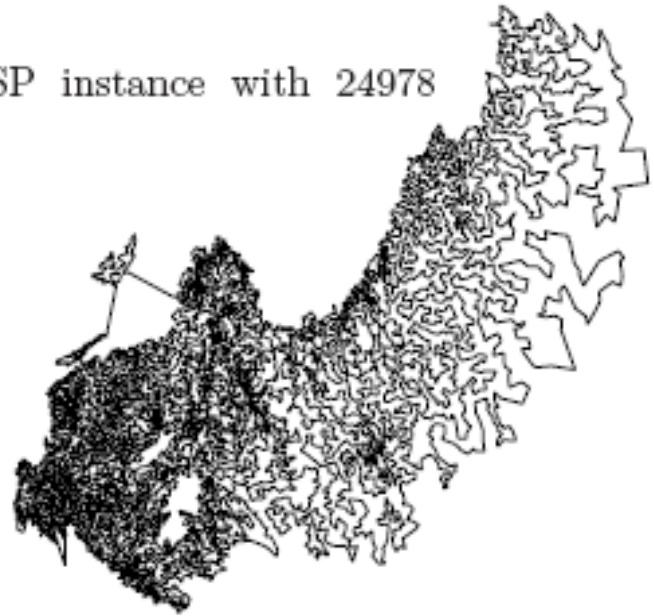
The traveling salesman problem (from Chvatal's page)



TSP instance with 52 cities.



TSP instance with 24978



Metaheuristics

E-G. Talbi

Size of the search space

- NLP problem : Non-Linear Programming Problem
- Many applications : mechanics, aeronautics, ...
- Classical methods don't give "good" results

$$G2(x) = \frac{\sum_{i=1}^n \cos^4(x_i) - 2 \prod_{i=1}^n \cos^2(x_i)}{\sqrt{\sum_{i=1}^n i x_i^2}}$$

$$\prod_{i=1}^n x_i \geq 0.75$$

$$\sum_{i=1}^n x_i \leq 7.5n$$

$$0 \leq x_i \leq 10$$

Non linear Function
Global Optimum unknown

Non-linear Constraint + linear constraint

Size of the search space

- Treated as a **mathematical problem** = infinite number of solutions / dimension.
- **On machine** : suppose a precision of six decimals, each variable may take 10 000 000 different values.
- $|S| = 10\,000\,000^n = 10^{7n}$
- Search space larger than TSP
- **Evaluation function** and **constraints** ?
- Maximize $G2(x)$; Non feasible solutions = 0

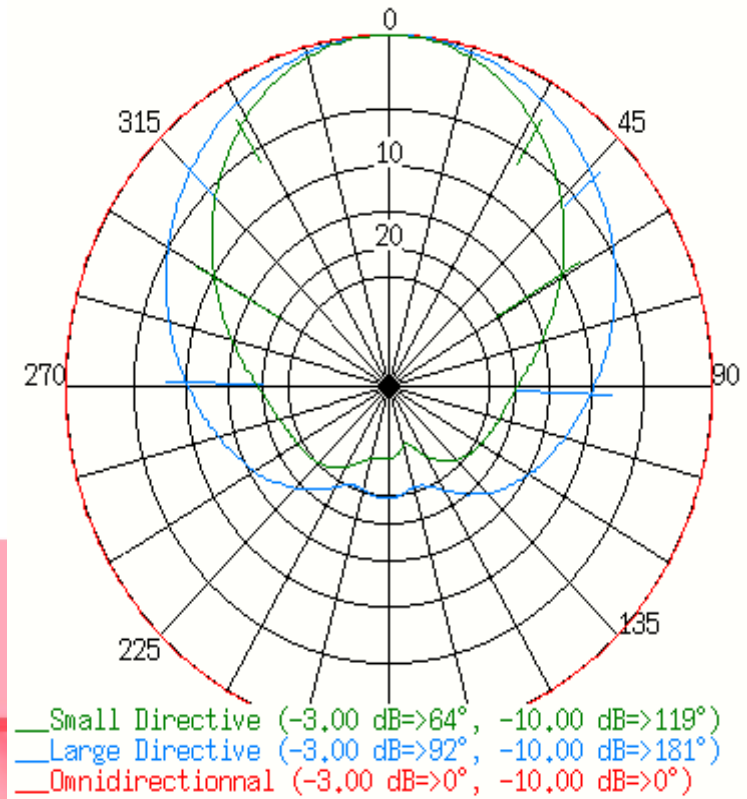
Constraints

- Most of the problems have constraints.
- Ex : Timetabling :
 - List of courses, classes, students / classes, professor / classes, available rooms, capacity of rooms, logistics of rooms (video, computer, microphone, ...).
 - **Hard constraints (must be satisfied) :**
 - Each class must be assigned to a room with a sufficient number of seats and required facilities.
 - Students assigned to different classes must not be scheduled at the same time.
 - Professors must not be scheduled to courses in parallel.

Optimization + Simulation: Cellular network design



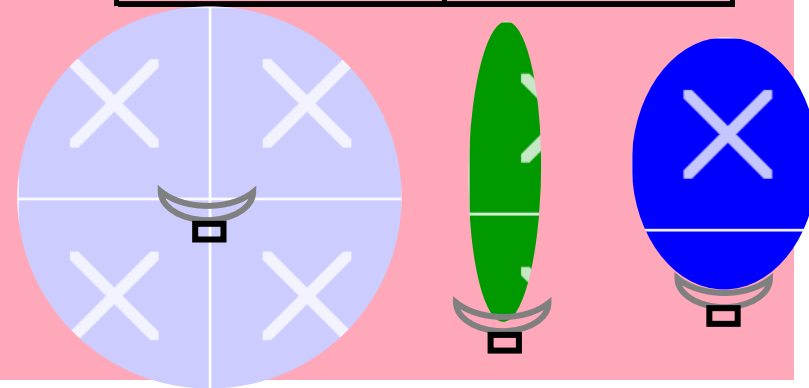
Parameters of antennae



Decision variables

Data	Bounds
Power	[26, 55] dBm
Diagram	3 types
Height	[30, 50] m
Azimut	[0, 359] °
Tilt	[-15, 0] °
Transmitters	[1, 7]

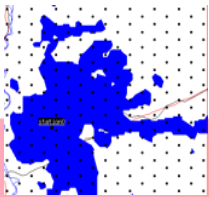
- Search space :
568 sites candidates → $2^{3689160}$ solutions
and ~ $600 \cdot 10^9$ choices for one antenna !
- Cost Evaluation (pop : 100, Gen : 10^5 ,
cost = more than one year !!)



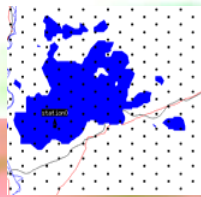
1 omni or 1-3 sectorial

Impact of the parameters via simulation

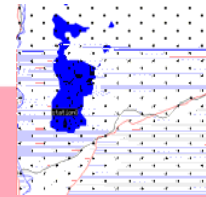
- Cell = zone covered by a BTS
- Limited Traffic for a site (technological constraint)
- Many sites are necessary



Omni-directional
60 dBm

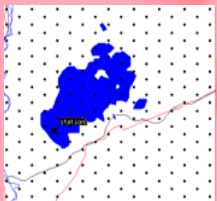


Sectorial type 1

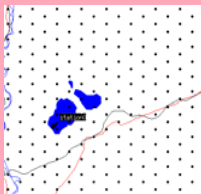


Sectorial type 2

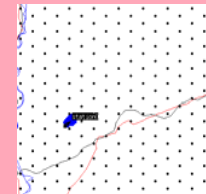
**Propagation Model ?:
Free space**



Azimut 50°



PIRE -10 dBm

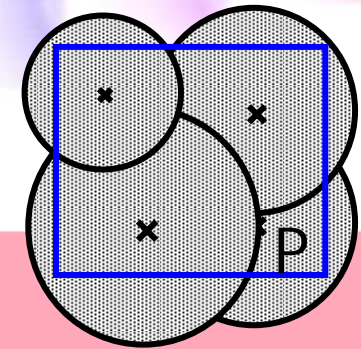


Tilt -15°

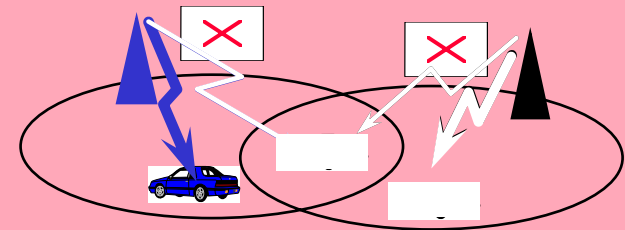
Model : Constraints

- A set of BTS satisfying the constraints :

- Covering

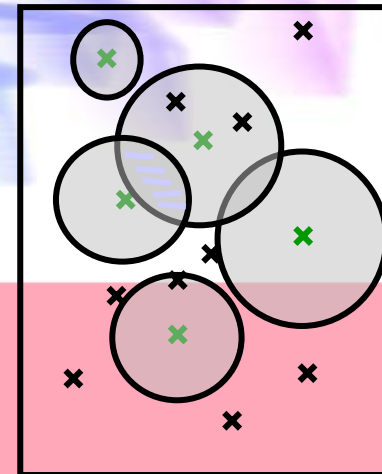


- Handover (mobility)



Model : Objectives

- **Min** Number of sites (cost)
- **Min** Interferences
- **Max** Traffic handled



Zone géographique

- × Used Sites
- × Non-used Sites
- ≡ Covered Zone
- Handover Zone

Other standard problems

- Integer programming problem (**IP**); **Commercially available.**
- (Weighted) constraint satisfaction problem (**CSP**, **WCSP**)
 - Maximum satisfiability problem (**MAX SAT**)
 - Set covering problem (**SCP**)
 - Generalized assignment problem (**GAP**)
- Generalized quadratic assignment problem (**GQAP**)
- Resource constrained project scheduling problem (**RCPSP**)
 - Vehicle routing problem (**VRP**)
 - Cutting stock problem (**CSTP**)
 - 2-Dimensional Packing Problem (**2PP**)

•

...

Ex : Knapsack Problem

- *The classic Knapsack problem is typically put forth as:*
 - A thief breaks into a store and wants to fill his knapsack with as much value in goods as possible before making his escape. Given the following list of items available, what should he take?
 - Item A, weighting w_A pounds and valued at v_A
 - Item B, weighting w_B pounds and valued at v_B
 - Item C, weighting w_C pounds and valued at v_C
 - ...

Ex : Knapsack Problem

■ Input

- Capacity K
- n items with weights w_i and values v_i

■ Goal

- Output a set of items S such that
 - the sum of weights of items in S is at most K
 - and the sum of values of items in S is maximized

The Simplest Versions...

Can items be divided up such that only a portion is taken?

The thief can hold 5 pounds and has to choose from:

3 pounds of gold dust at \$379.22/pound

6 pounds of silver dust at \$188.89/pound

1/9 pound of platinum dust at \$433.25/pound

Are all of the weights or total values identical?

The thief breaks into a ring shop where all of the rings weight 1oz. He can hold 12 ounces; which should he take?

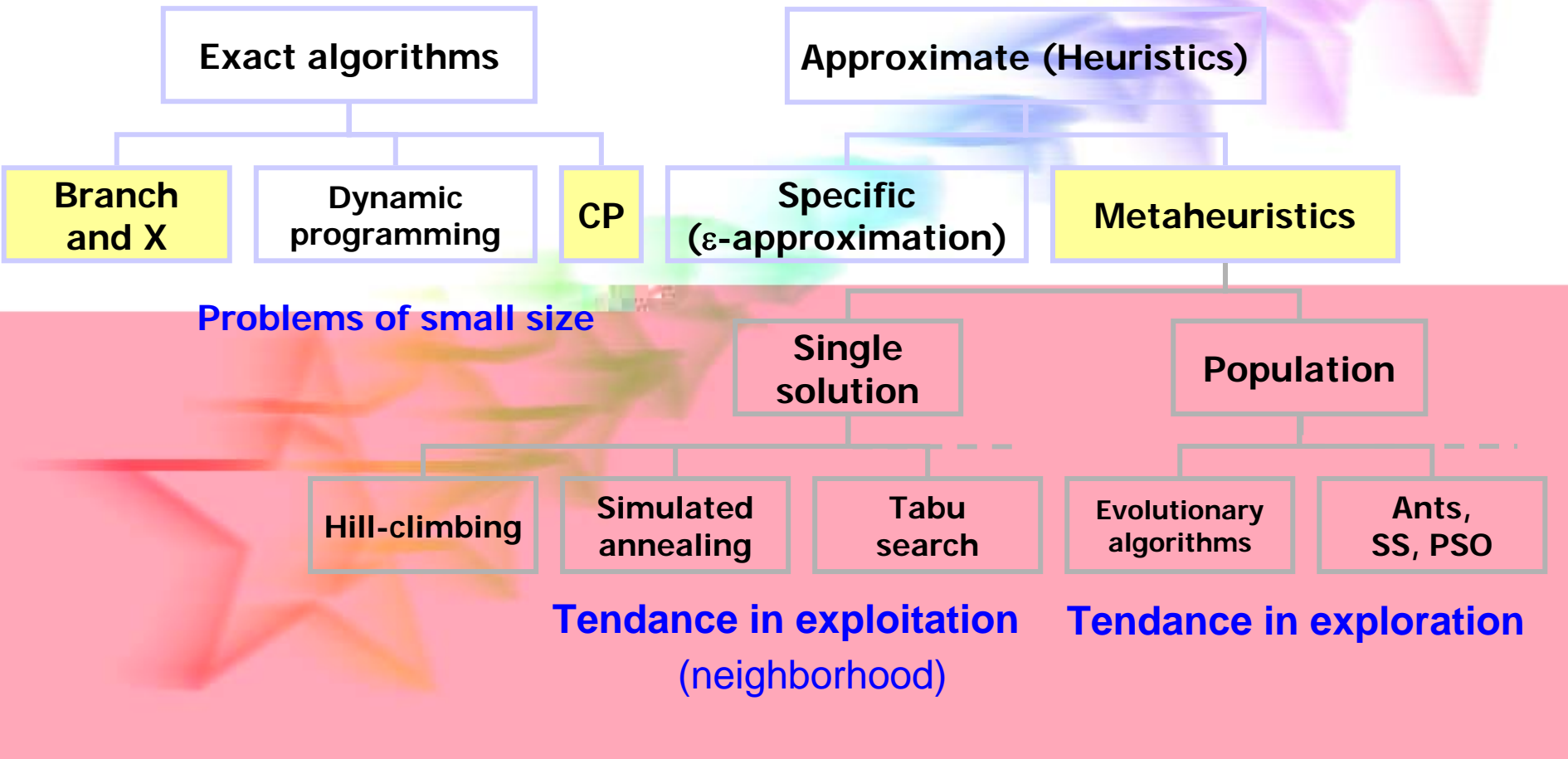
A Deceptively Hard Version...

What if each problem has the same price/pound?

This problem reduces to the **bin-packing problem**: we want to fit as many pounds of material into the knapsack as possible.

How can we approach this problem?

Optimization methods



Exact methods

- Obtain global optimal solutions
- **Guarantee** their optimality
- Useless for **large** problems (problem, instance).

Table 1.4 Order of magnitude of the maximal size of instances that state-of-the-art exact methods can solve to optimality. For some practical problems, this maximum size may be negligible. For the TSP problem, an instance of size 13509 has been solved to optimality [32].

Optimization Problems	Quadratic Assignment (QAP)	Flow-shop Scheduling (FSP)	Graph Coloring	Capacitated Vehicle routing
Size of the instances	30 objects	100 jobs 20 machines	100 nodes	60 clients

Approximate algorithms

■ Approximate algorithms or Heuristics

- The word *heuristic* has its origin in the old Greek word *heuriskein*: art of discovering new strategies (rules) to solve problems
- Generate « high quality » solutions in a reasonable time for practical use.
- No guarantee to find the global optimal solution

■ Approximation algorithms

- Guarantee on the bound of the obtained solution from the global optimum.
- ε -approximation

Metaheuristics

- The suffix *meta* also a Greek word: upper level methodology.
- Introduced by F. Glover in 1986
- **Metaheuristic**: Upper level general methodology (templates) that can be used as guiding strategies in designing underlying heuristics to solve specific optimization problems

Application of Metaheuristics

- Engineering design, topology optimization and structural optimization in electronics and VLSI, aerodynamics, fluid dynamics, telecommunications, automotive, robotics, ...

- Machine learning and data mining in bioinformatics and computational

biology, finance, ...

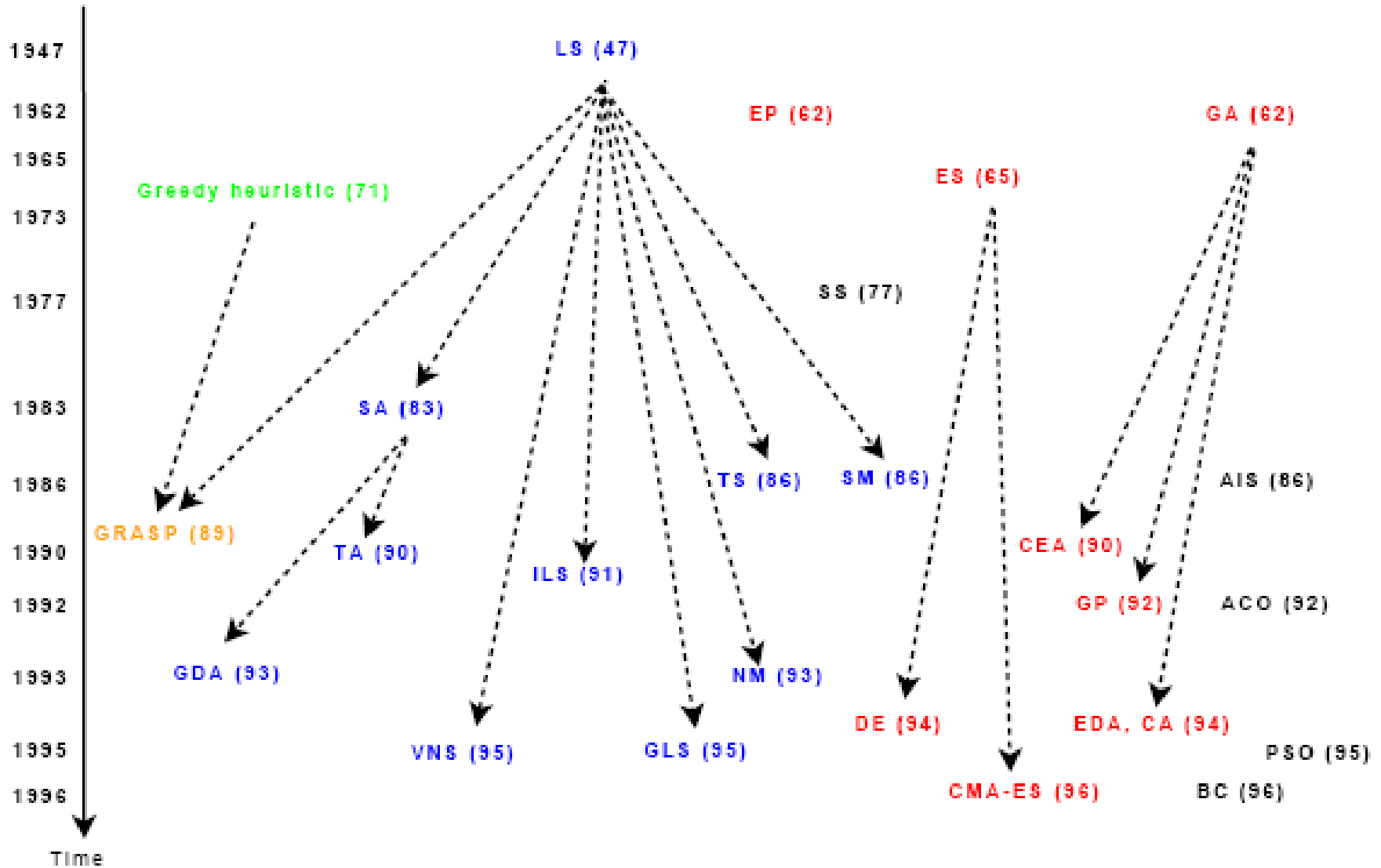
- System modeling, simulation and identification in chemistry, physical

biology, medical, signal and image processing, ...

- Planning in routing problems, robot planning, scheduling and production problems, logistics and transportation, supply chain management, environment, ...

- and so on.

Genealogy of Metaheuristics



Metaneuristics

E-G. I aidi

Greedy heuristics

- Start from scratch (**empty solution**) and constructs a solution by assigning values to one decision variable at a time until a **complete solution** is generated
- **Solution** = presence/absence of a finite set of elements $E = \{e_1, \dots, e_n\}$. Partial solution = subset of E
- Objective function = $f : 2^E \rightarrow \mathbb{R}$

Algorithm 1.2 Template of a greedy algorithm.

$s = \{\}$; /* Initial solution (null) */

Repeat

$e_i = \text{Local-Heuristic}(E \setminus \{e/e \in s\})$;

/* next element selected from the set E minus already selected elements */

If $s \cup e_i \in F$ **Then** /* test the feasibility of the solution */

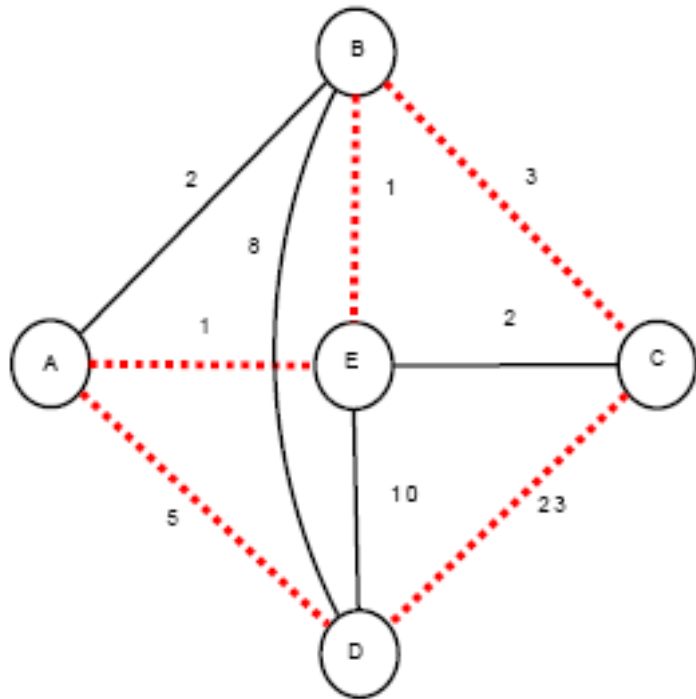
$s = s \cup e_i$;

Until Complete solution found

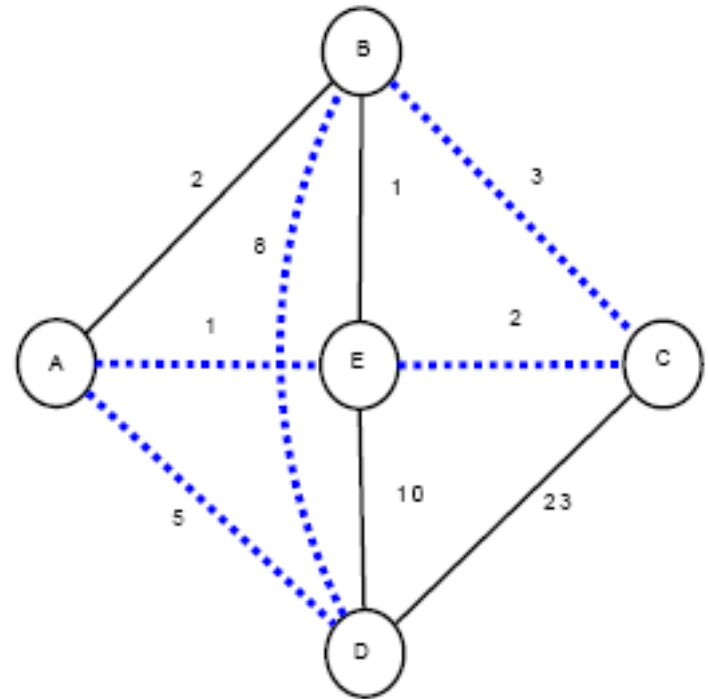
Constructive algorithms (Greedy)

- Two design questions
 - Definition of the set of **elements**
 - Element selection **heuristic**: which gives the best “profit” at each iteration.
- Assignment of decision variables **one by one**
- At each iteration, choose the optimal decision for the current decision variable
- Take an optimal decision at each step does not guarantee **global optimality**.
- **Popular (Simplicity)**
- **Reduced complexity**

Greedy for TSP: nearest neighbor



Greedy final solution : A - E - B - C - D - A with cost = 33



Better solution : A - E - C - B - D - A with cost = 19

Fig. 1.10 Illustrating a greedy algorithm for the TSP using a static heuristic. An element is associated to an edge of the graph, and the local heuristic consists in choosing the nearest neighbor. The obtained solution is $(A - E - B - C - D - A)$ with a total cost of 33, whereas a better solution with a cost of 19 is given (right).

Greedy for knapsack

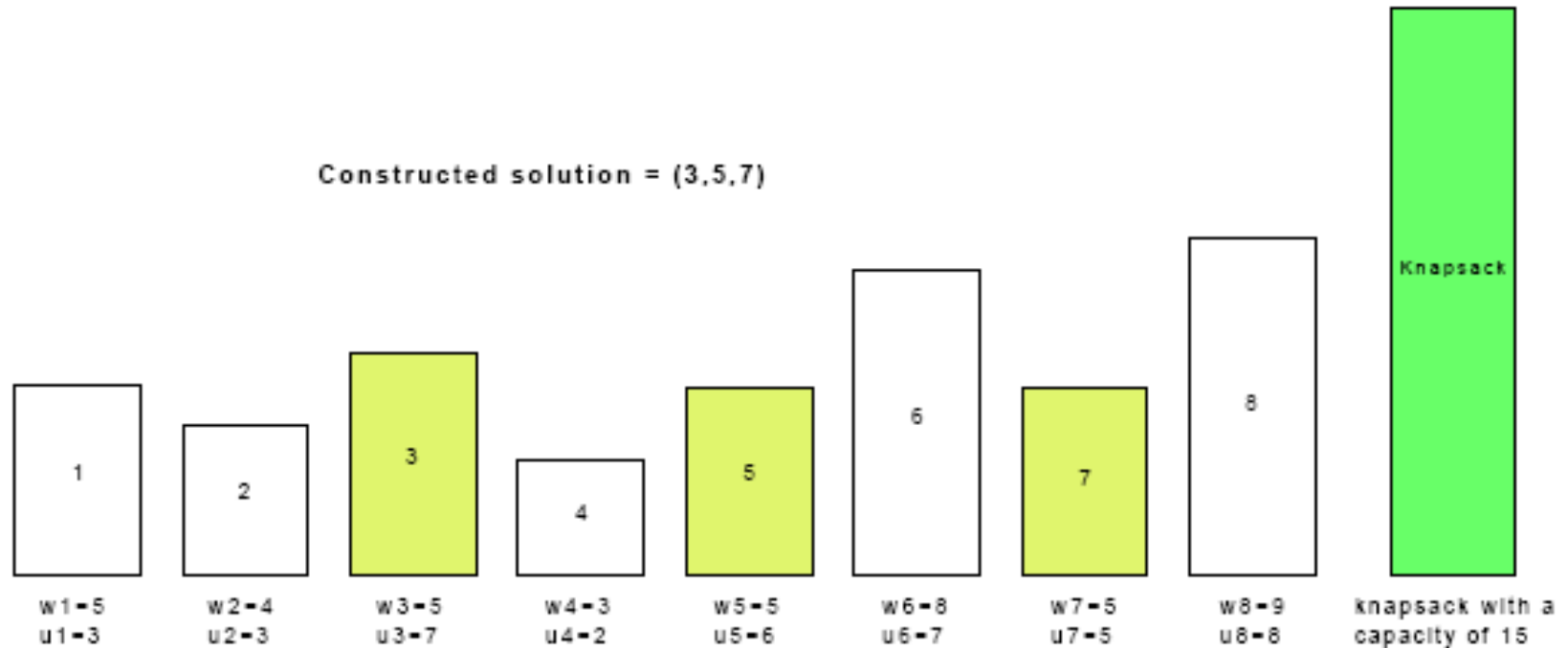
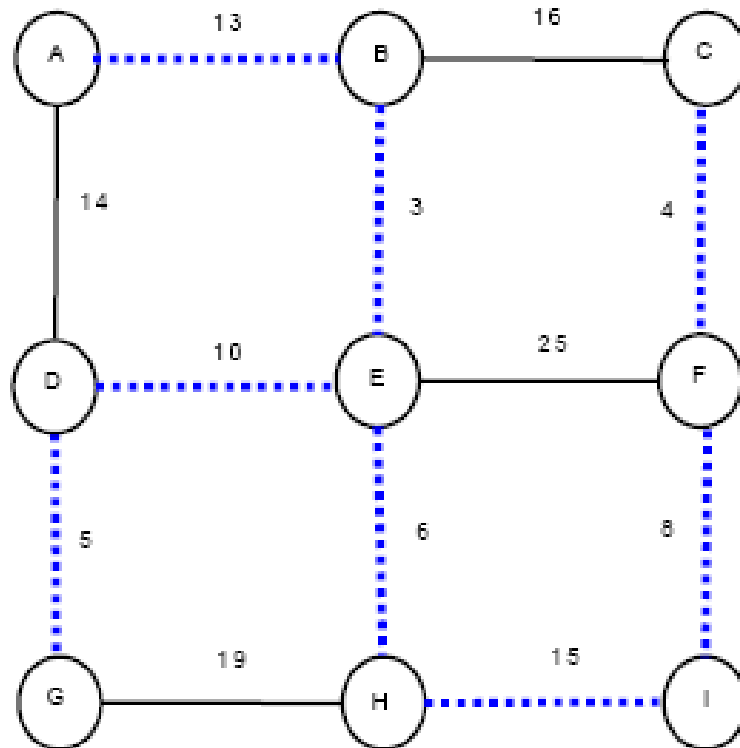


Fig. 1.11 Illustrating a greedy algorithm for the knapsack problem. An element is associated to an object, and the local heuristic consists in choosing an element minimizing the ratio $\frac{w_i}{u_i}$. The final solution may be not optimal.

Greedy for spanning tree



Constructed solution

(B,E), (C,F), (D,G), (E,H), (F,I), (E,D), (A,B), (H,I)

Fig. 1.12 Illustrating a greedy algorithm for the spanning tree problem. The edge (A,D) has not been selected even if it is less costly than the edge (H,I) because it generates a non feasible solution (a cycle).

Constructive algorithms (SAT)

- **Heuristic** : For each variable from 1 to n, in a given order, assign the value which satisfies the maximum number of non satisfied clauses (current).

- **Example** :

$$\bar{x}_1 \wedge (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_1 \vee x_4)$$

- Let us consider the variable x_1 , $x_1 = \text{TRUE}$ (3 clauses)
- ... Clause 1 non satisfied.
- Poor Performances.
- **Heuristic for order selection**: Order all the variables based on the frequencies (from small to large values)

Constructive algorithms (SAT)

- Example :

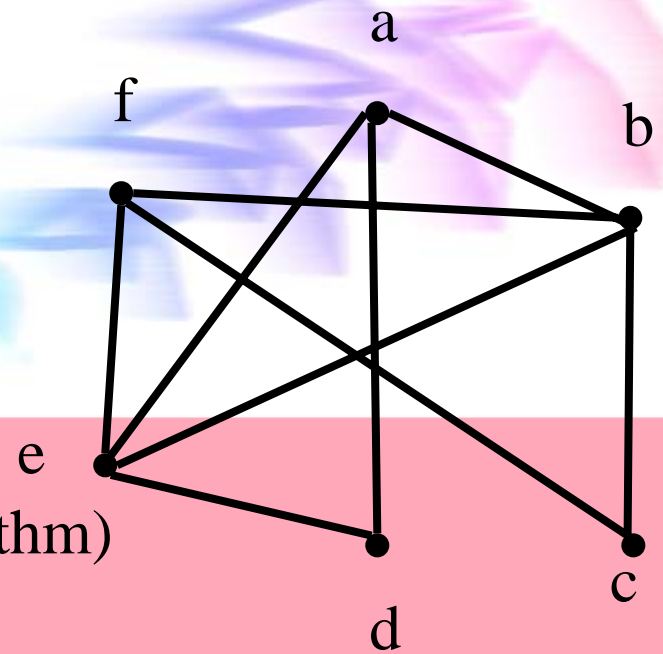
$$(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (\bar{x}_1 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_4) \wedge (x_2 \vee x_5) \wedge (x_2 \vee x_6) \wedge F$$

- F does not contain the variables x_1 and x_2 , but many occurrences of the other variables.
- Other variables = higher frequencies
- x_1 =TRUE (2 clauses), x_2 =TRUE (2 clauses)
- Impossible to satisfy the clauses 3 and 4
- **Other improvements** : Forbid the assignment which initializes a clause to FALSE, frequencies into non treated clauses, size of the clauses in choosing the order, ...

Constructive algorithms (NLP)

- No **efficient** greedy algorithms for NLP problems
- **Example** : function of two variables x_1 and x_2
- Initialize one variable (ex. x_1) to a constant, and vary x_2 until optimum found.
- When optimum found (x_2 =constant), vary x_1 until optimum found
- Efficient when no or small interactions between variables (decomposable function)

Ex : Constructive algorithms (Graph coloring)



Color Largest Degree First (LF Algorithm)

Welsh and Powell [1967]

Order by degree

Node : b, e, a, f, c, d

Color : 1 2 3 3 2 1

Ex : Constructive algorithms (Graph coloring)

Merging Non-adjacent vertices

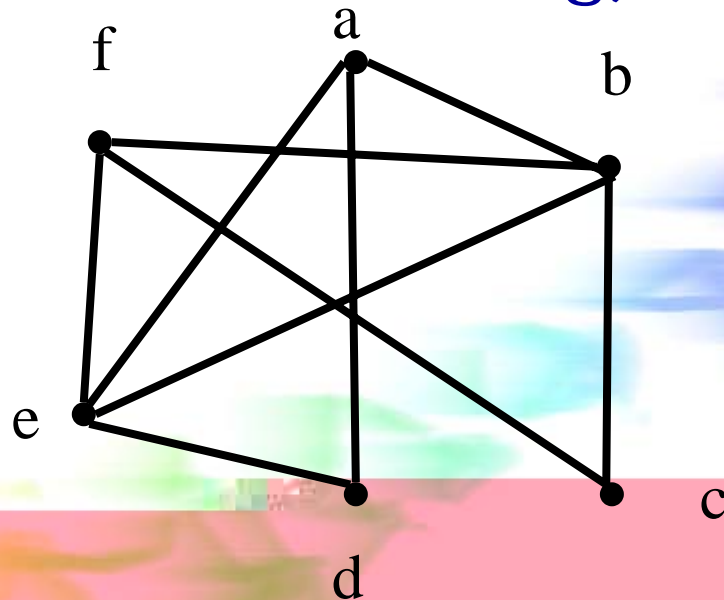
Dutton and Brigham [1981]

DB Algorithm

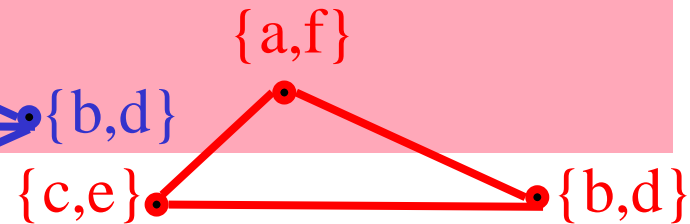
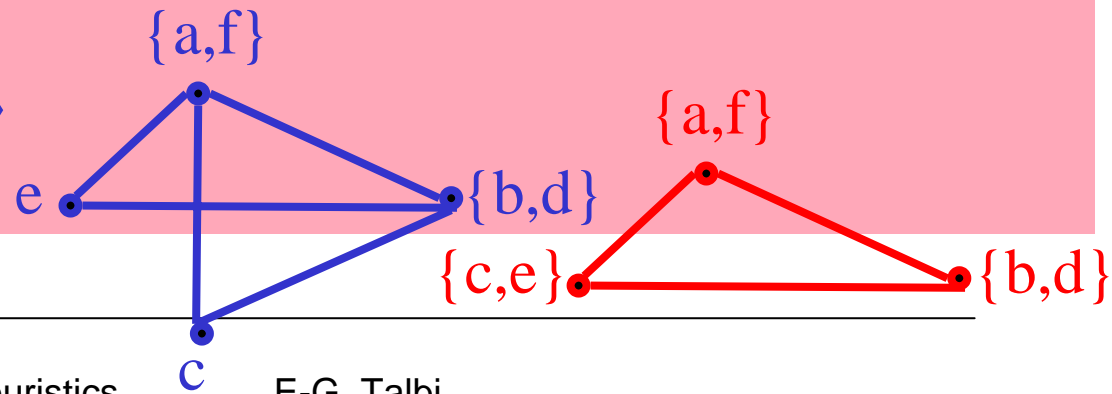
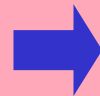
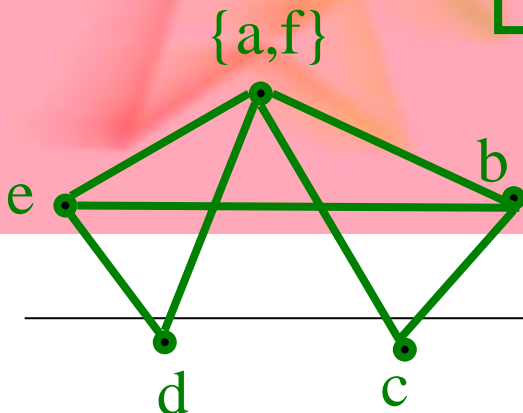
Cluster successively non-adjacent nodes,
Until the existence of non adjacent nodes
i.e. resultant graph is complete.

Heuristic : cluster first adjacent nodes
Which have the maximum number of common adjacent nodes

Ex : Constructive algorithms (Graph coloring)



(a, c)	(a, f)	(b, d)	(c, d)	(c, e)	(d, f)
1	2	2	0	2	1
		2	1	2	



Metaheuristics

E-G. Talbi

When using Metaheuristics?

- An easy problem (e.g. P class) with **VERY large** instances
- An easy problem with **hard real-time constraints**
- **Difficult problem** (e.g. NP-complete) with moderate size and/or difficult input structures
- Optimization problems with **time-consuming objective functions and/or constraints**
- Non-analytical models of optimization problems: **black box scenario** (objective function).
- **Non deterministic complex models**: uncertainty, robust optimization, dynamic, multi-objective, ...

Design / Control problems

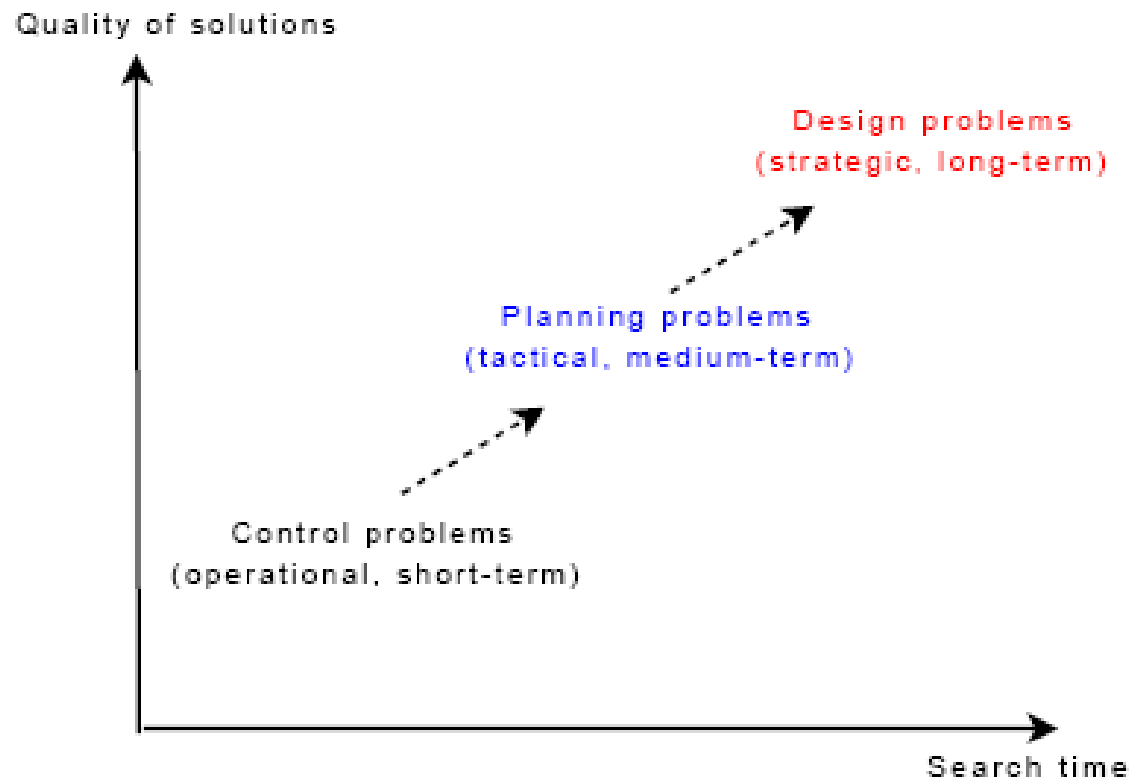
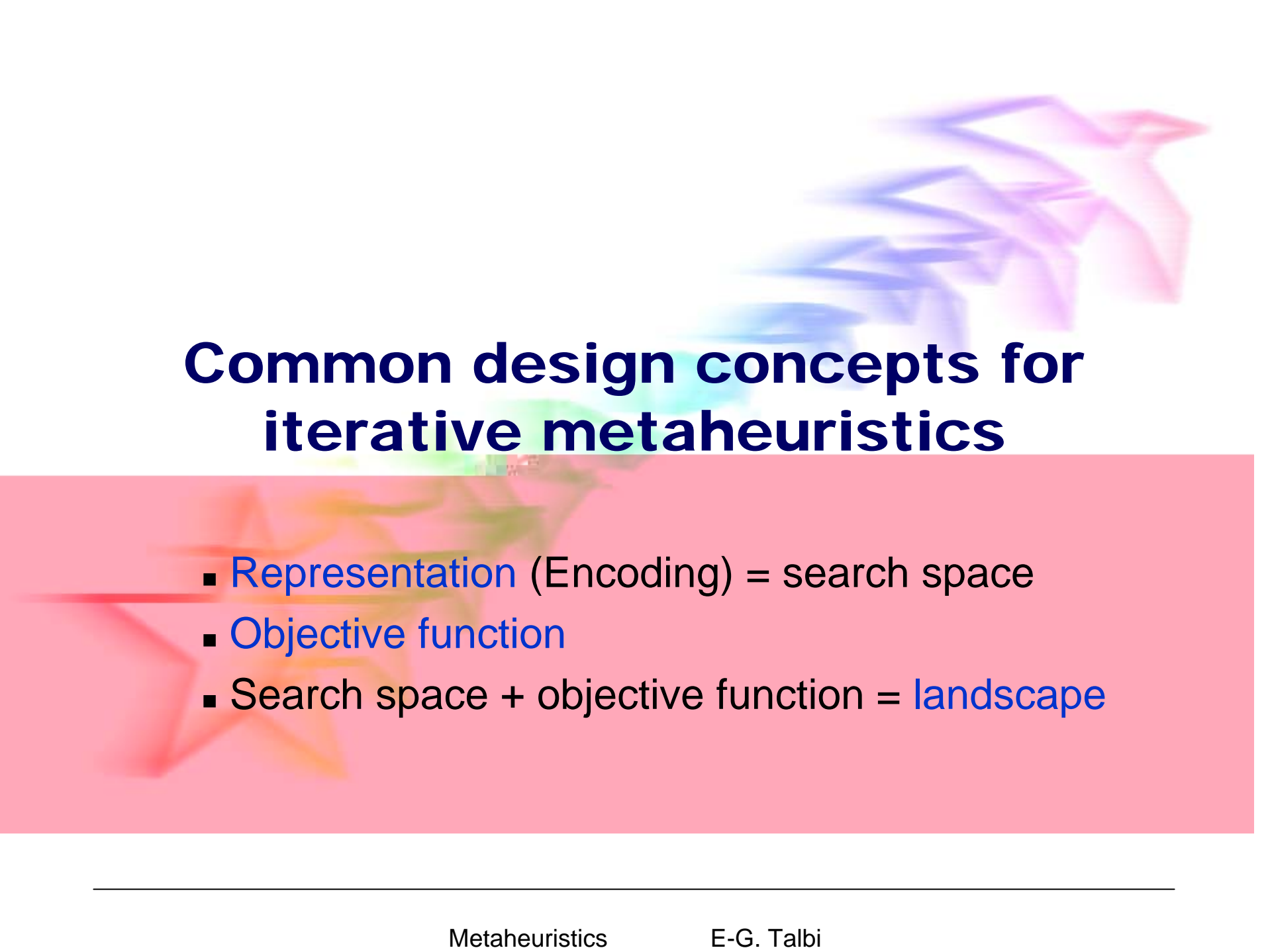


Fig. 1.15 Different classes of problems in terms of the tradeoff between quality of solutions and search time: design (strategic, long-term), planning (tactical, medium-term), control (operational, short-term).



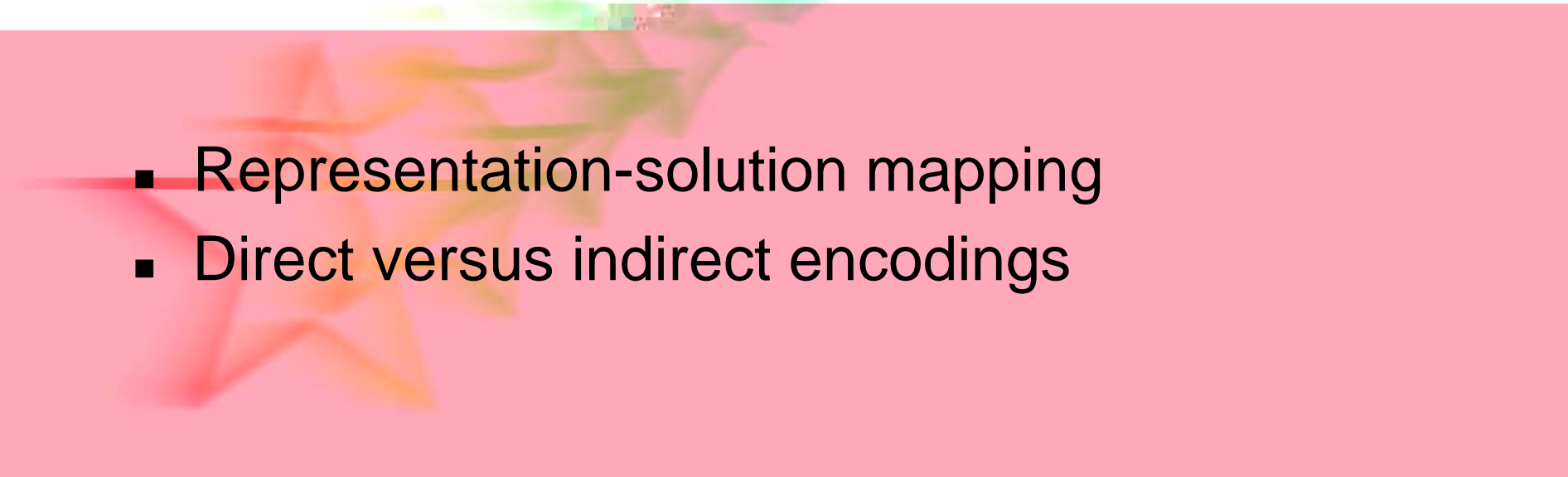
Common design concepts for iterative metaheuristics

- **Representation** (Encoding) = search space
- **Objective function**
- Search space + objective function = **landscape**

Representation: Characteristics

- **Completeness**: All solutions associated to the problem must be represented.
- **Connexity**: A search path may exist between any two solutions (related also to the search operators), especially the global optimal solution.
- **Efficiency**: easy to manipulate by the search operators (time and space complexities).

Types of Representations

- **Linear representation:** strings of symbols of a given alphabet
 - **Non-linear representation:** based generally on graph structures (e.g. trees)
- 
- Representation-solution mapping
 - Direct versus indirect encodings

Classical linear representations

- Knapsack problem
- SAT problem
- 0/1 IP problems

1 0 0 0 1 1 0 1 1 1 0 1

Binary encoding

- Location problem
- Assignment problem

5.7 6.9 4.2 8.4 2.1

Vector of discrete values

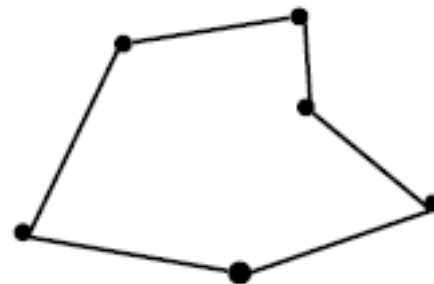
- Continuous optimization
- Parameter identification
- Global optimization

$$f(x) = 2x + 4x.y - 2x.z$$

1.23 5.65 9.45 4.76 8.96

Vector of real values

- Sequencing problems
- Traveling salesman problem
- Scheduling problems



1 4 8 9 3 6 5 2 7

Permutation

Representation

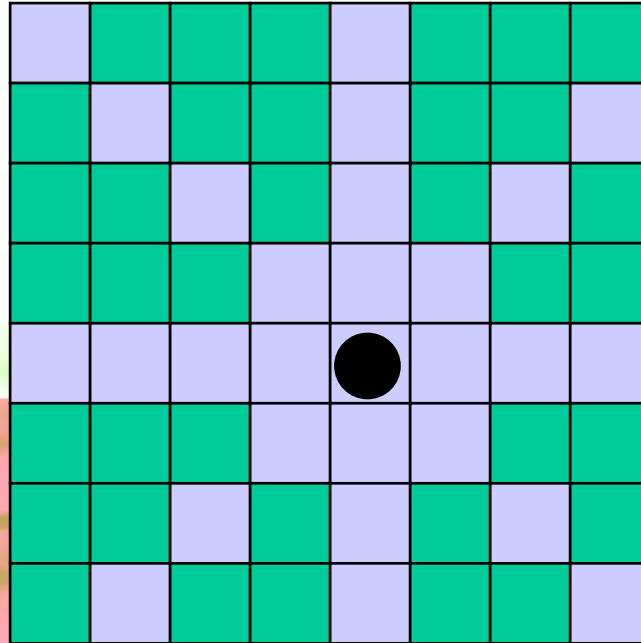
Representation ==> Search Space + Size

- **SAT Problem** with n variables : Binary vector of size n , element = variable
- Size of the search space = 2^n .
- Each solution of the search space is feasible
- **TSP Problem** of n cities : permutation of integer numbers $1, \dots, n$. element = city.
- Size of the search space = $n!$
- Symmetric TSP : $n! / 2$
- Starting city: $(n-1)! / 2$

Representation

- **Problem NLP** : all real numbers for all n dimensions.
- Floating numbers representation to approximate real numbers : 6 digits of precision, 10^{7n} possible different solutions.
- Choice of the representation is **important**.

Example : 8 Queens

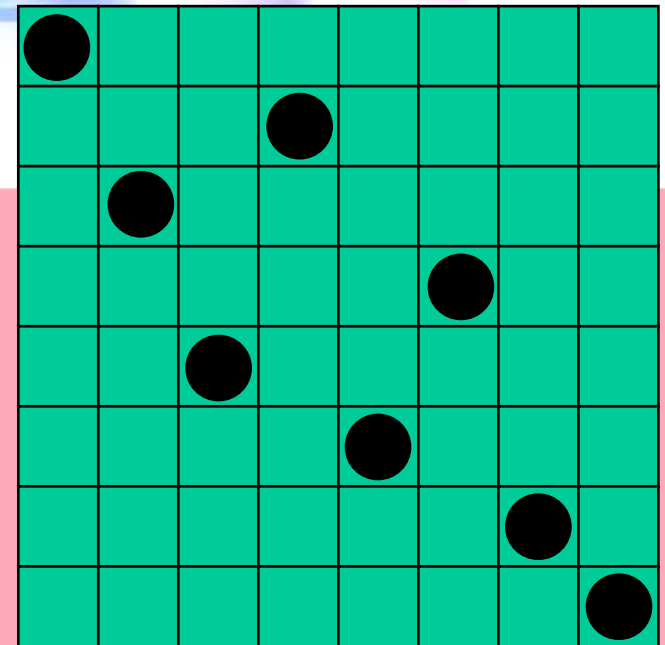


Problem : Place 8 queens on a 8x8 chess such that two queens don't overlap.

Example : 8 Queens

A permutation of the numbers from
1 to 8

1	3	5	2	6	4	7	8
---	---	---	---	---	---	---	---



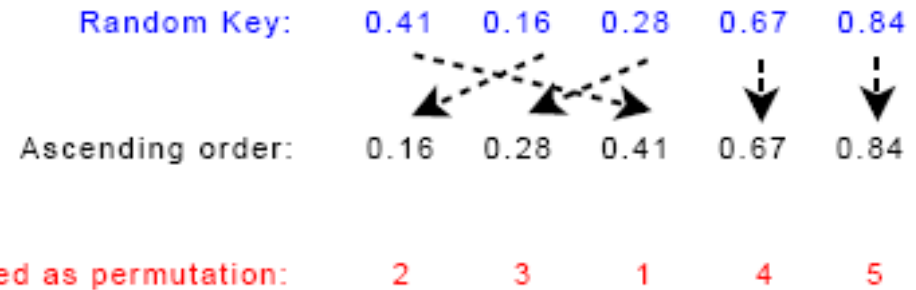
Permutation : a configuration

Problem Representation

- For example, in the 8-Queens problem, when every state is an assignment of the 8 queens on the board:
 - The number of possibilities with all 8 queens on the board is 64 choose 8, which is **over 4 billion**.
 - The solution of the problem prohibits more than one queen per row, so we may assign each queen to a separate row, now we'll have $8^8 > \mathbf{16 \text{ million}}$ possibilities.
 - Same goes for not allowing 2 queens in the same column either, this reduces the space to $8!$, which is only **40,320** possibilities.

Non-traditional linear representations

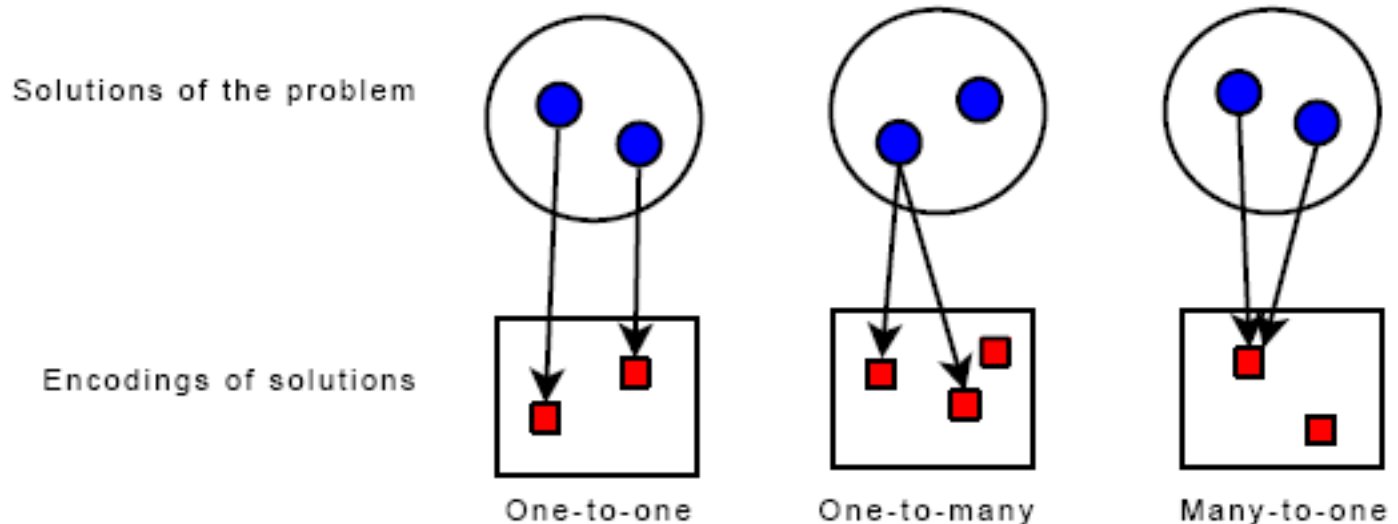
- Random keys encoding: real-values for permutations



- Messy representations
- Non-coding regions
- Diploid representations
- Quantum representations
- Mixed representations

Representation / solution mapping

- Transforms the encoding (genotype) to a problem solution (phenotype).
 - **One-to-one**: single solution = single encoding
 - **One-to-many**: one solution = multiple encodings (redundancy): e.g. symmetry in grouping problems
 - **Many-to-one**: one encoding = multiple solutions



Direct/ Indirect representations

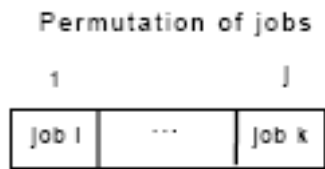
- **Indirect representation:** encoding is not a complete solution
- A **decoder** is required: deterministic or non-deterministic
- Ex: Flow-shop scheduling problem: j jobs. Each job has M operations (M machines).

Matrix of $J \times M$ elements

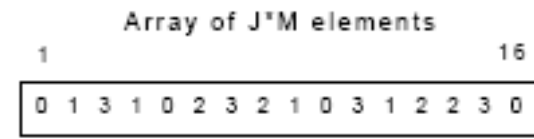
	1	M	
Job i	Op7 m2 1,3	Op3 m3 13,17

Job j	Op6 m2 20,22	Op4 m4 34,36	Op1 m1 51,55

(a) Direct encoding



(b) First Indirect encoding



(c) Second Indirect encoding

Objective function

Formulation of the goal

$$f : S \rightarrow \mathbb{R}$$

- Self-sufficient objective functions
- Guiding objective functions
- Representation decoding
- Interactive optimization
- Relative and competitive objective functions
- Meta-modeling

Self-sufficient objective functions

- Straightforward objective function = original formulation of the problem

- **TSP** : minimize the total distance

$$\min \sum dist(x, y)$$

- **SAT** : boolean formulae satisfied (TRUE).

- **NLP** : minimize the function $G2(x)$.

Guiding objective functions

- Transform the original goal for a better convergence (guide the search)
- **SAT** : Non optimal solution = FALSE →
 - No information on the quality of solutions,
 - No indication on the improvement of solutions to guide the search towards good solutions, ...
 - **New objective function = Number of satisfied clauses**

Graph coloring

$$G = (S, A)$$

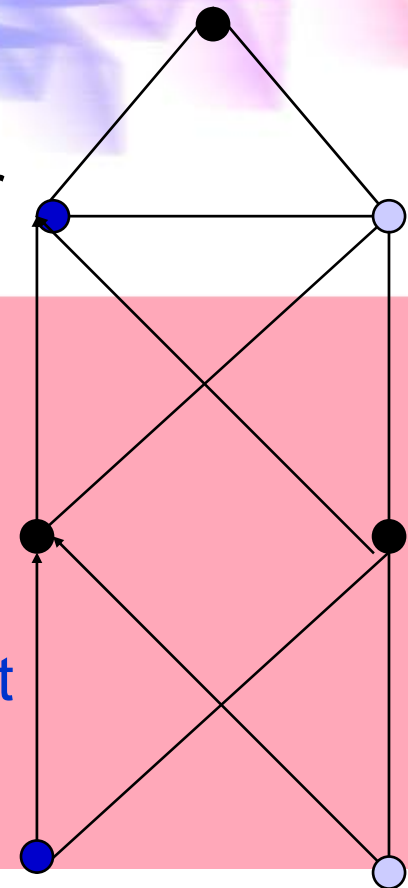
coloring $f: S \rightarrow C$ such as $(s, t) \in A \Rightarrow f(s) \neq f(t)$

$\text{Chr}(G) = \min \text{card } f(S)$, chromatic number of G

f

$$\text{Chr}(G) = 3$$

Many applications : frequency assignment (telecommunications), scheduling, timetabling, register allocation, ...



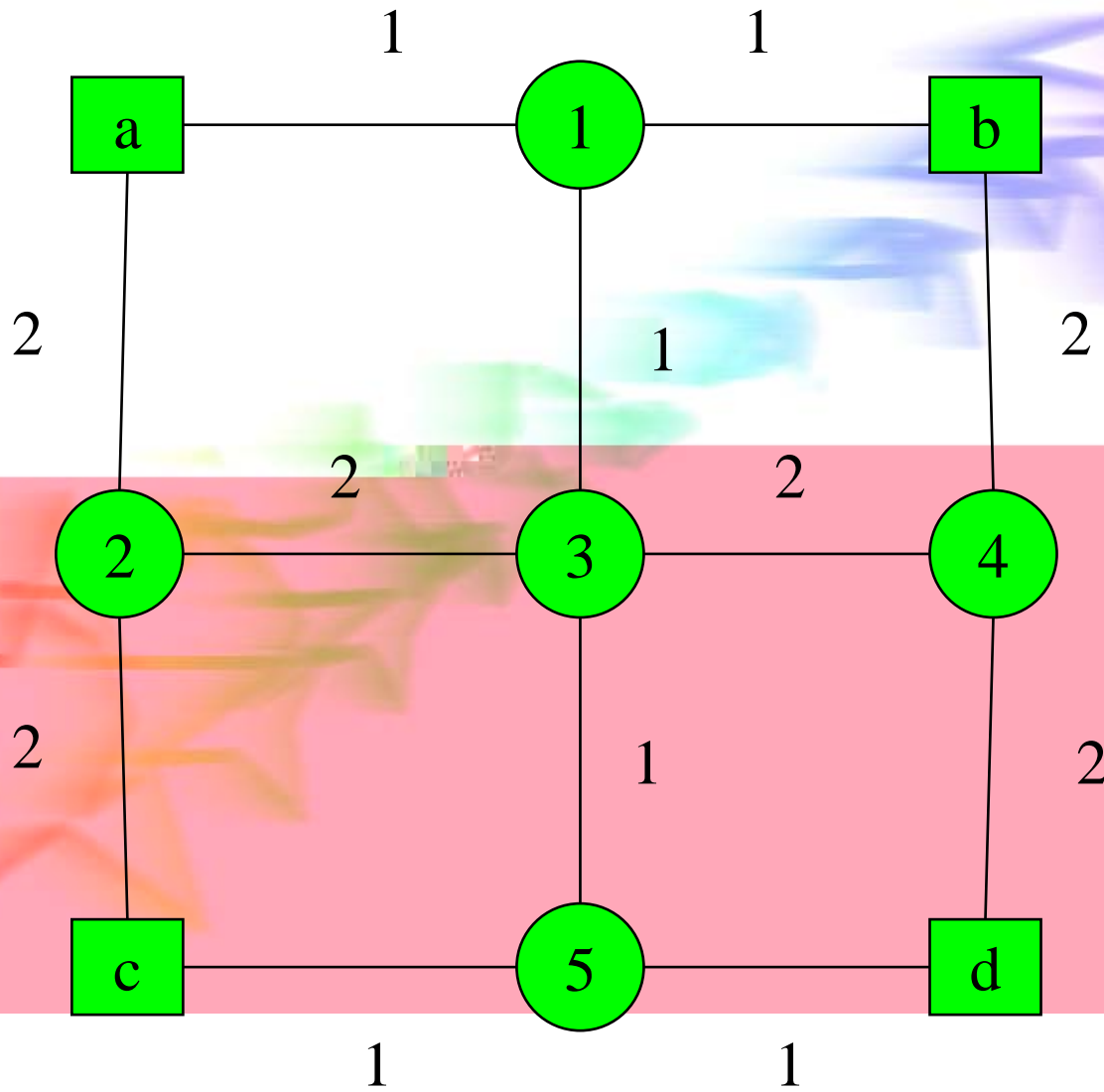
Representation decoding

- Relationship between the representation and the objective function
- Decode the representation to compute the objective function

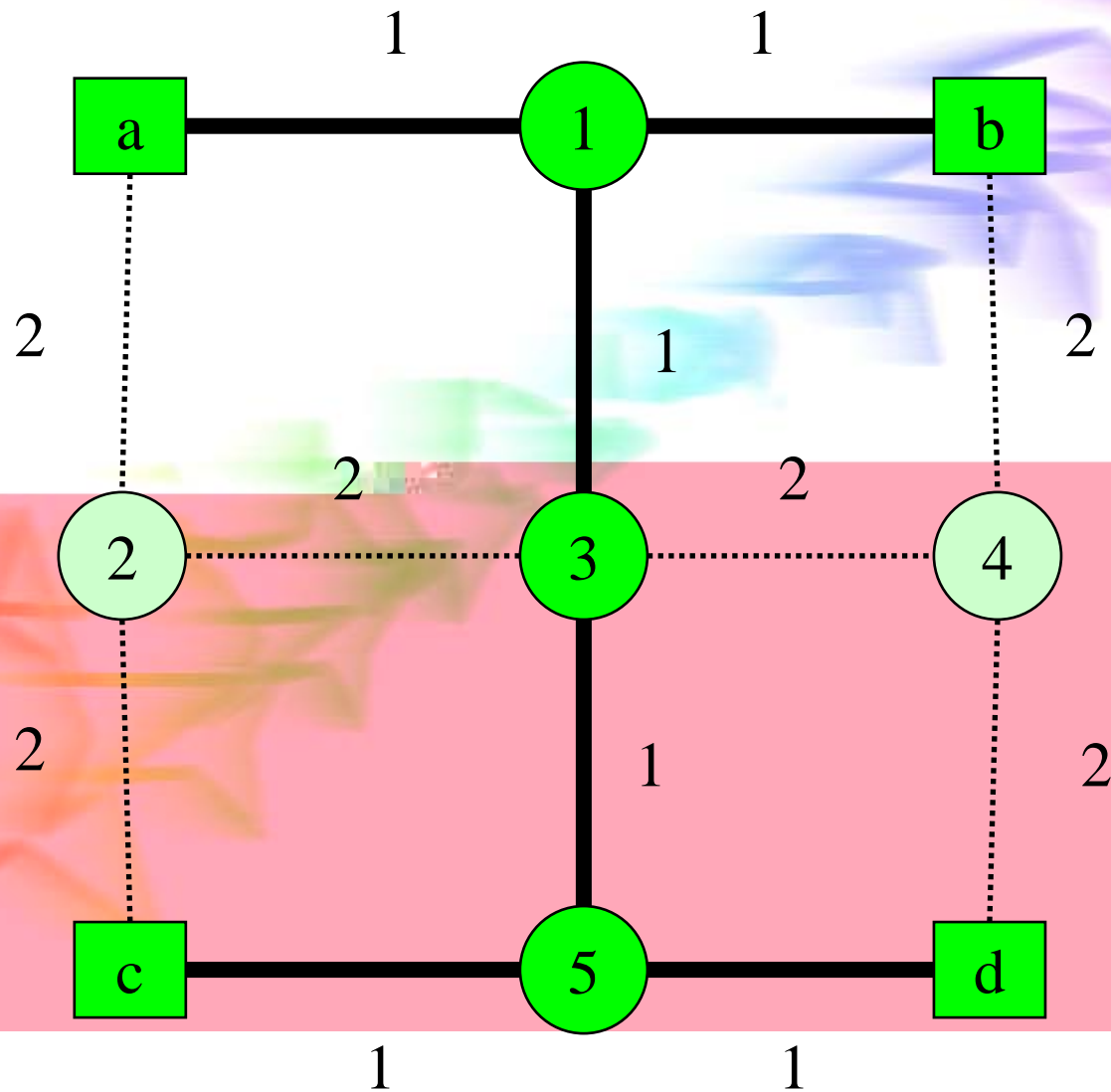
Steiner problem in a graph

- **Steiner problem in a graph:**
non-oriented graph $G=(V,E)$, V : vertices, E : edges
 T : terminals (obligatory)
 c_e : weight of the edge $e \in E$
- **Find a spanning tree of terminal nodes of minimum weight**
(particulier case: if $T = V$, spanning tree problem of minimum weight)
- Steiner vertices: non obligatory vertices which generate the optimal solution.
- **Applications: Telecommunications, Phylogeny in biology, ...**

Steiner problem in a graph



Steiner problem in a graph



Steiner problem in a graph

- Characterization of a solution

$X \subseteq V$: sub-set of non obligatory vertices

- Steiner Tree



Tree which connects the terminal vertices (obligatory using a sub-set X of non-obligatory vertices)

- Each Steiner Tree may be characterized by the sub-set of used non-obligatory vertices

- Optimal Steiner Tree



Spanning Tree with Minimum Weight connecting the terminal vertices and using the optimal subset X^* of non-obligatory vertices

Steiner problem in a graph

- $|V| = p$
- Solution: $s = (s_1, s_2, \dots, s_i, \dots, s_p) \Leftrightarrow X$
- **Representation** by a 0-1 indication of appartenance
- $s_i = 1$, if the i -th vertice non-obligatory is used (i.e., if $v_i \in X$)
 $s_i = 0$, otherwise

Interactive optimization

- The user is involved on-line in the loop of a metaheuristic
- Motivations:
 - User intervention to guide the search process: user knowledge, decision making in multi-criteria problems, ...
 - User intervention to evaluate a solution: human preferences (subjective evaluation), ...
 - Taste of coffee, wine, ...
 - Visual appeal or attractiveness

Relative and competitive objective functions

- Impossible to have an absolute objective function f
 - Ex: Game theory, co-evolution, learning classifier systems
 - Game: Strategy A may be better than B, B better than C, C better than A
- **Relative fitness**: associates a rank to a solution into a population
- **Competitive fitness**: competition between solutions
 - Bipartite
 - Tournament
 - Full

Meta-modeling

- Time consuming part: evaluation of the objective function
- Approximate the objective function
 - Extremely expensive objective functions: structural design optimization (e.g. 3D aerodynamic system design: computational fluid dynamics CFD simulation)
 - Approaches
 - Neural networks
 - Response surface methodology
 - Kriging models, DACE, Gaussian techniques, machine learning approaches (SVM, ...)

Meta-modeling

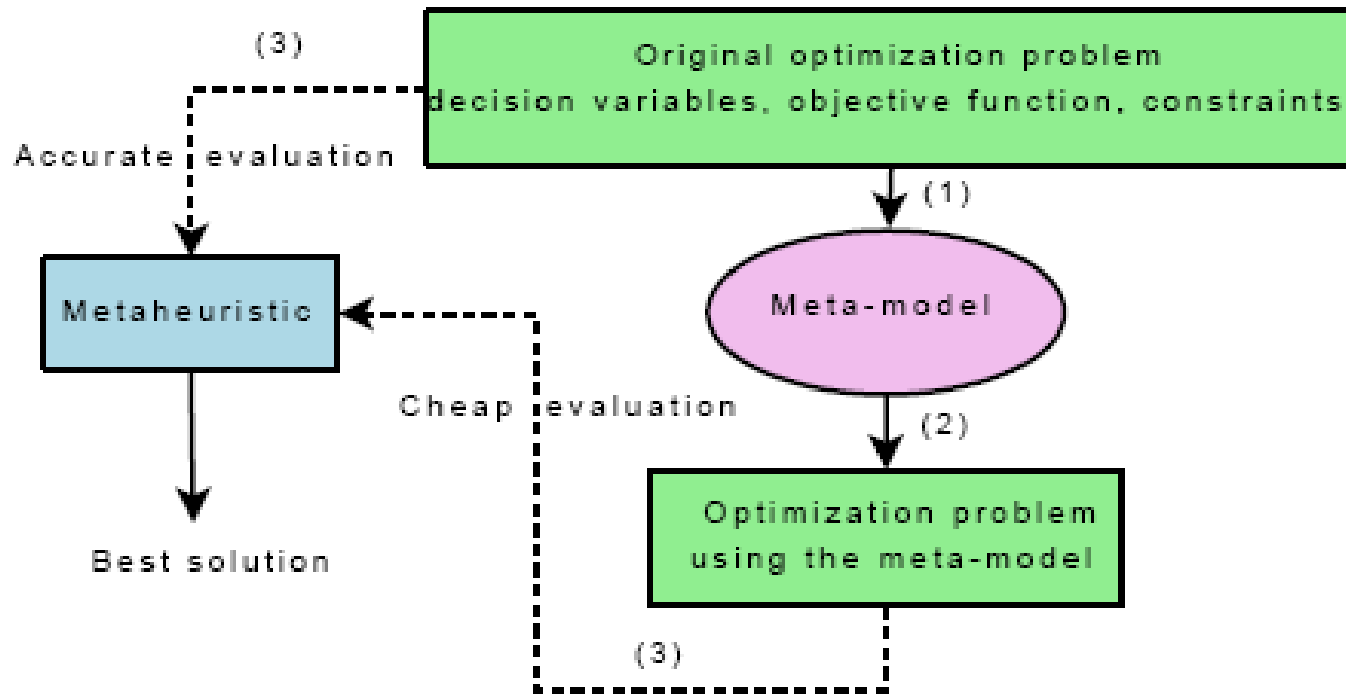


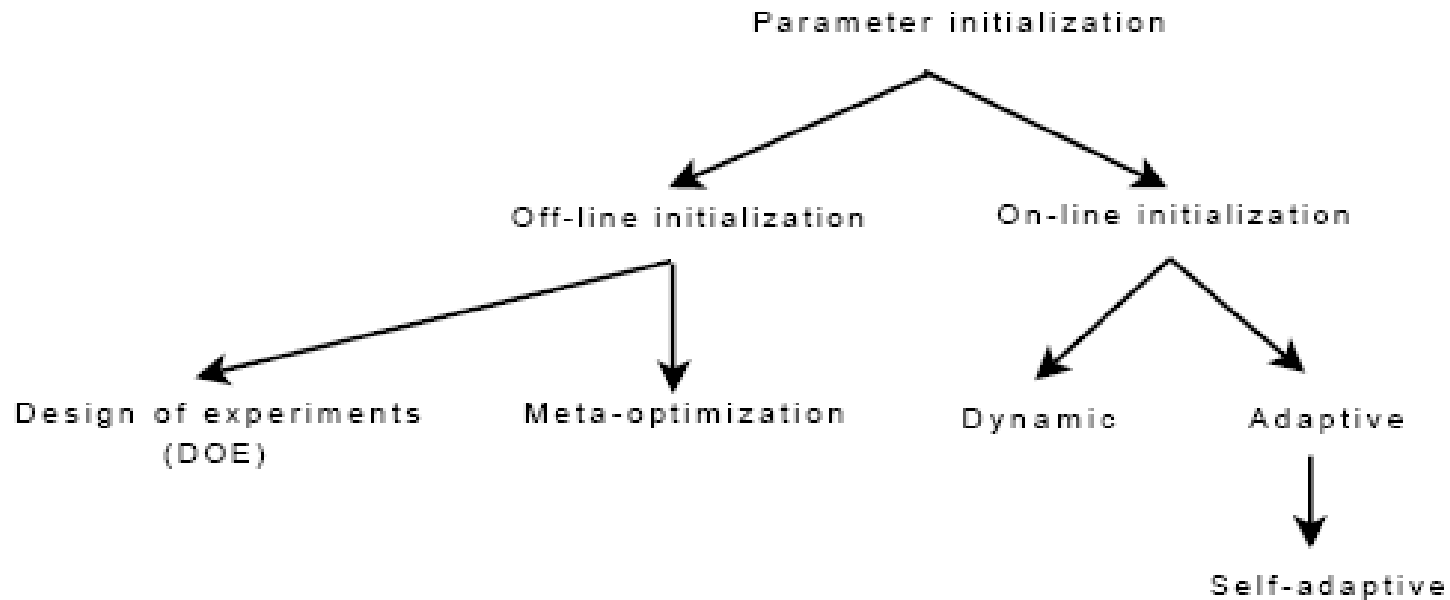
Fig. 1.23 Optimization using a meta-model. Once the model is constructed (1), the metaheuristic can use either the meta-model (2) or alternate between the two models (original and approximate) for a better compromise between accuracy and efficiency (3).

Constraint handling

- Not trivial to deal with constraints
 - **Reject** strategies: only feasible solutions are kept
 - **Penalizing** strategies: penalty functions
 - **Repairing** strategies: repair infeasible solutions
 - **Decoding** strategies: only feasible solutions are generated
 - **Preserving** strategies: specific representation and search operators which preserve the feasibility

Parameter tuning

- **Many** parameters have to be tuned for **any** metaheuristics
- Large flexibility and robustness but careful initialization (efficiency and effectiveness of the search)
- Optimal tuning depends on the problem and instance: **no universally optimal tuning**

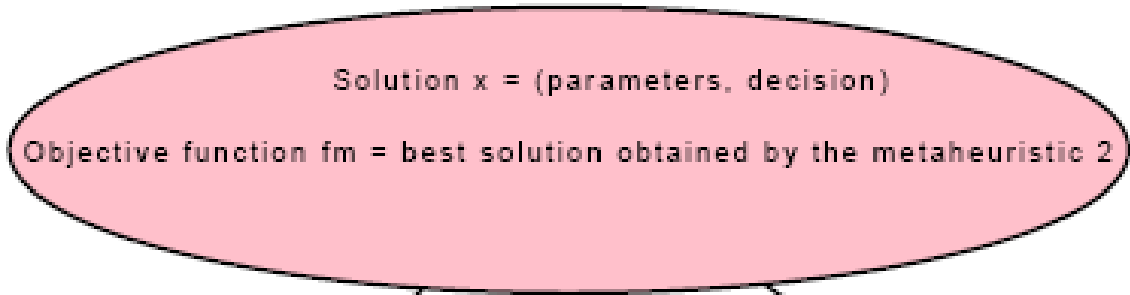


Meta-optimization



Metaheuristic 1 at meta-level

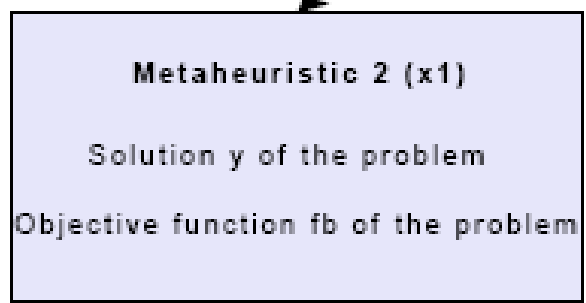
Meta-level



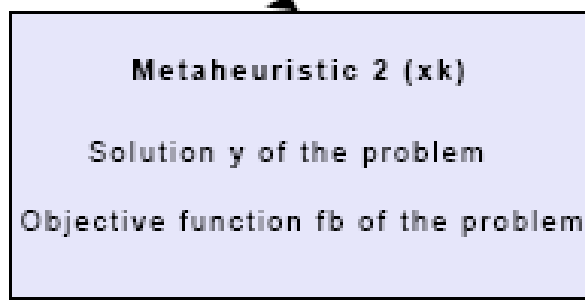
x_1

x_k

Base-level



...

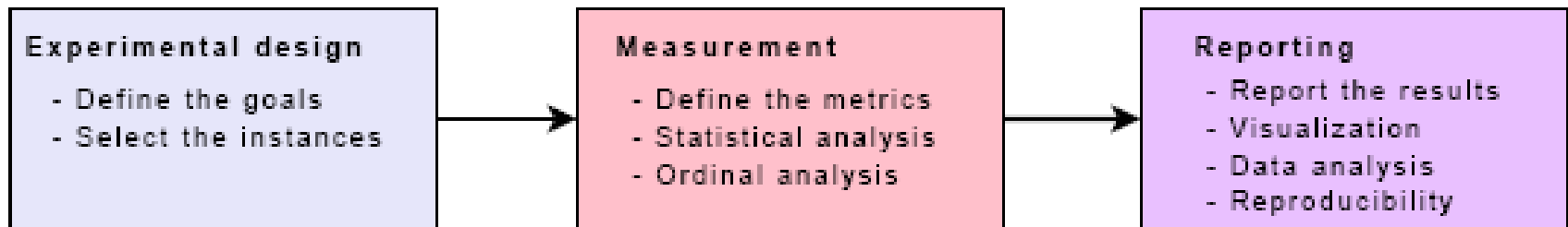


Metaheuristic 2 at base-level

Metaheuristic 2 at base-level

Performance analysis of metaheuristics

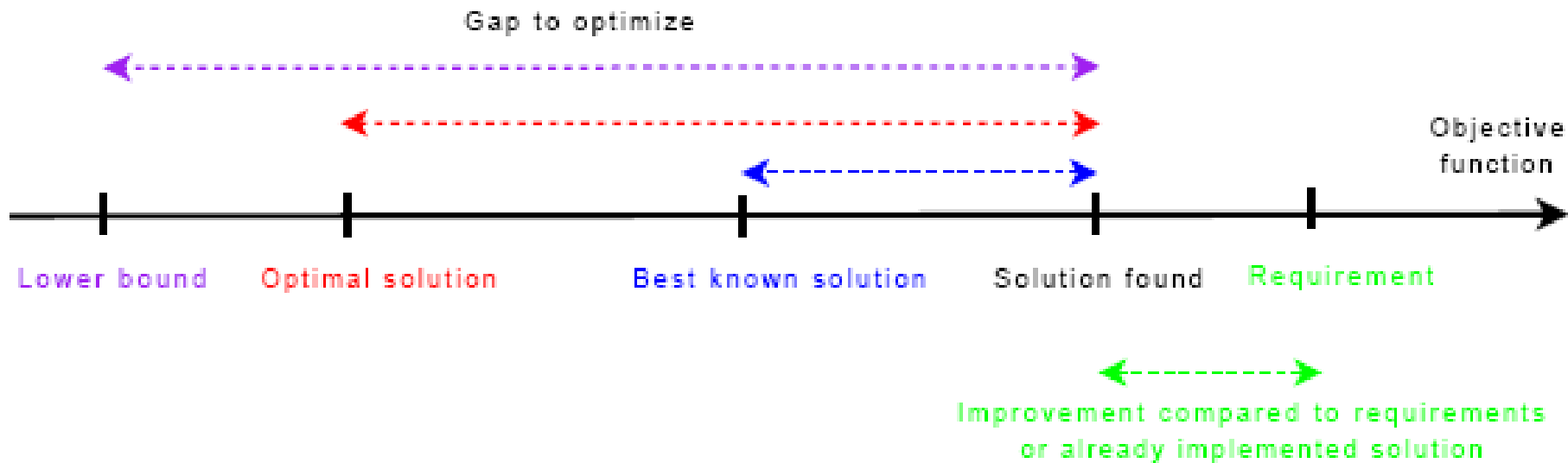
- **Experimental design**: goals of the experiments, selected instances (real-life, constructed) and factors to be defined
 - Random instances may be controversial
- **Measurements**: measures to compute → statistical analysis
- **Reporting** in a comprehensive way



Criteria

- Quality of solutions
- Computational effort: search time, ...
- Robustness (instances, problems, parameters, ...)
- Easy of use, simplicity, flexibility, development cost, ...

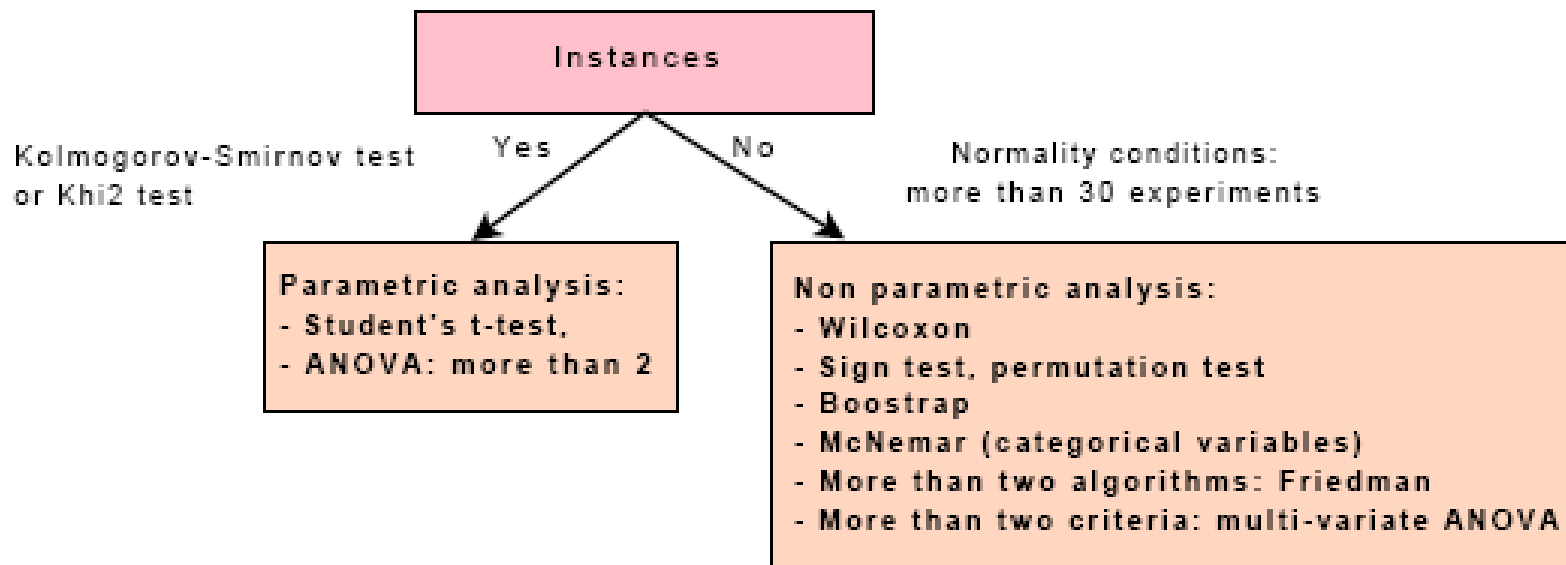
Quality of solutions



Measurements

- **Statistical analysis:** performance assessment and estimate the confidence of the results to be scientifically valid
- **Ordinal analysis:** ranking metaheuristics (e.g. Borda count voting method, Copeland's method)

According to the characteristics of the measurements



Reporting: Interaction plots

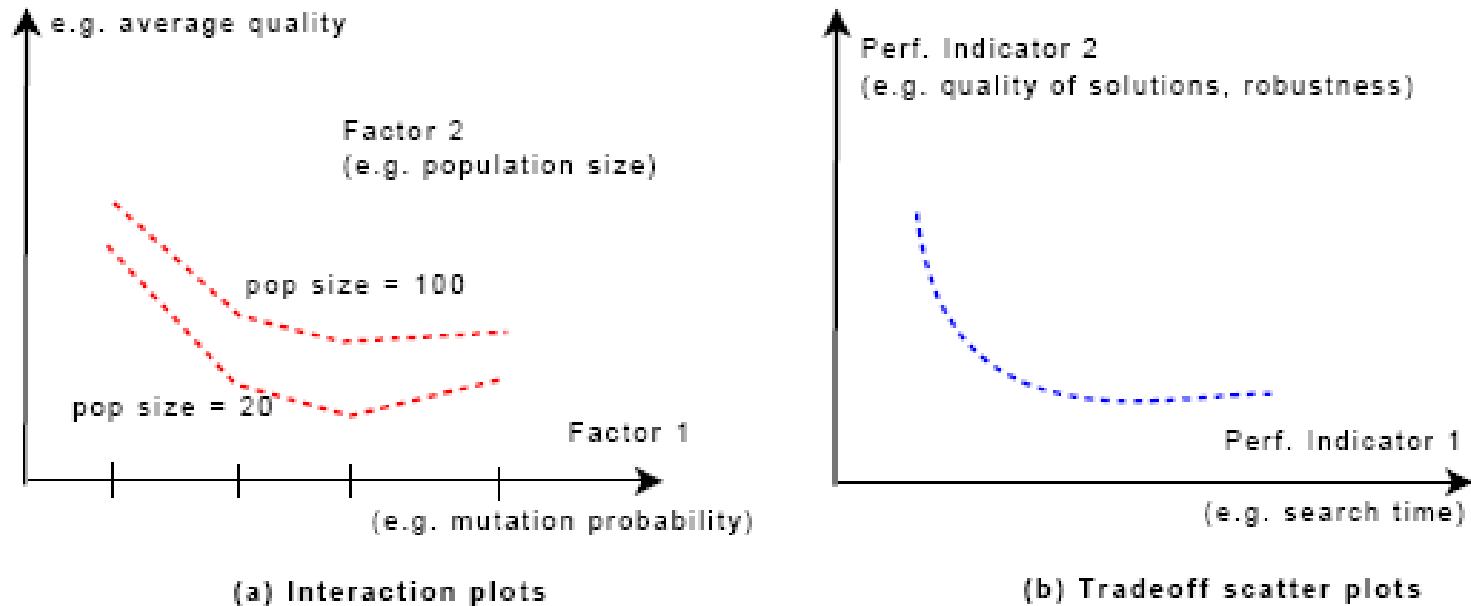
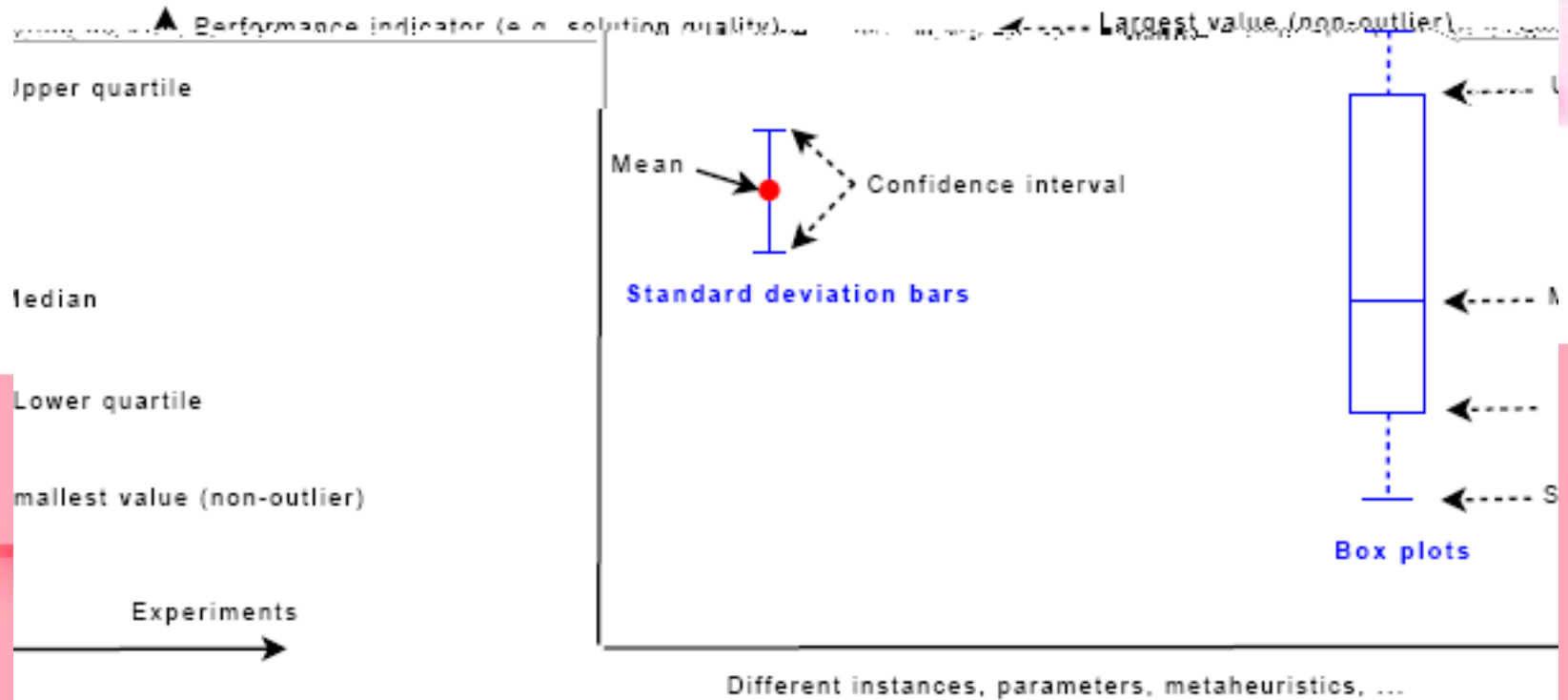


Fig. 1.29 (a) Interaction plots analyse the effect of two factors (parameters, e.g. mutation probability, population size in evolutionary algorithms) on the obtained results (e.g. solution quality, time). (b) Scatter plots analyze the tradeoff between the different performance indicators (e.g. quality of solutions, search time, robustness).

Reporting: deviation bars, confidence intervals



results: deviation bars,

Fig. 1.30 Some well known visualization tools to report confidence intervals.

Why a software framework?

- 3 major approaches are used for the development of metaheuristics:
 - **From scratch or no reuse**: costly (time, manpower), error prone, difficult maintain and evolve, ...
 - **Code reuse**: reuse third party code – application dependent, error prone, time consuming, ...
 - **Code and design reuse (design patterns)**: use of generic templates, inheritance,
 - Problem dependent part
 - Invariant part

Why a software framework?

- Optimization Problems in practice
 - **Diversity**
 - Continual **evolution** of the **modeling** (regards needs, objectives, constraints, ...)
 - **Need to experiment** many solving methods, techniques of hybridization, parameters, ...

Main characteristics of software frameworks

- Maximum design and code reuse
- Flexibility and adaptability
- Utility
- Transparent and easy access to performance and robustness
- Portability
- Ease of use and efficiency

ParadisEO framework

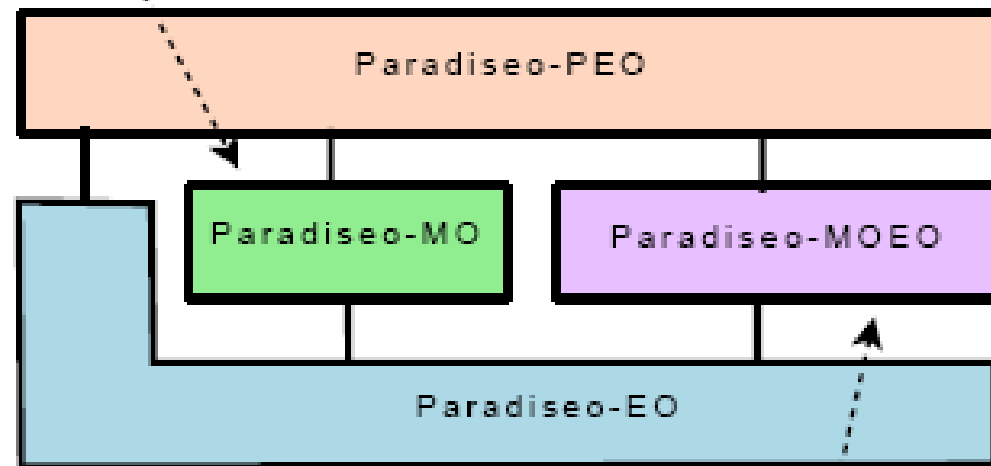


- A templates-based, ANSI-C++ compliant Metaheuristic Computation Framework.
- GForge Project by INRIA Dolphin Team.
- **Paradigm Free** (genetic algorithms, genetic programming, particle swarm optimization, local search, TS, SA, ILS, VNS, EDA, ES, ...).
- Hybrid, distributed and cooperative models.
- **Flexible** / a considered problem.
- **Generic** components (variation operators, selection, replacement, termination, particle behaviors ...).
- Many **services** (visualization, managing command-line parameters, saving/restarting, ...).

ParadisEO framework

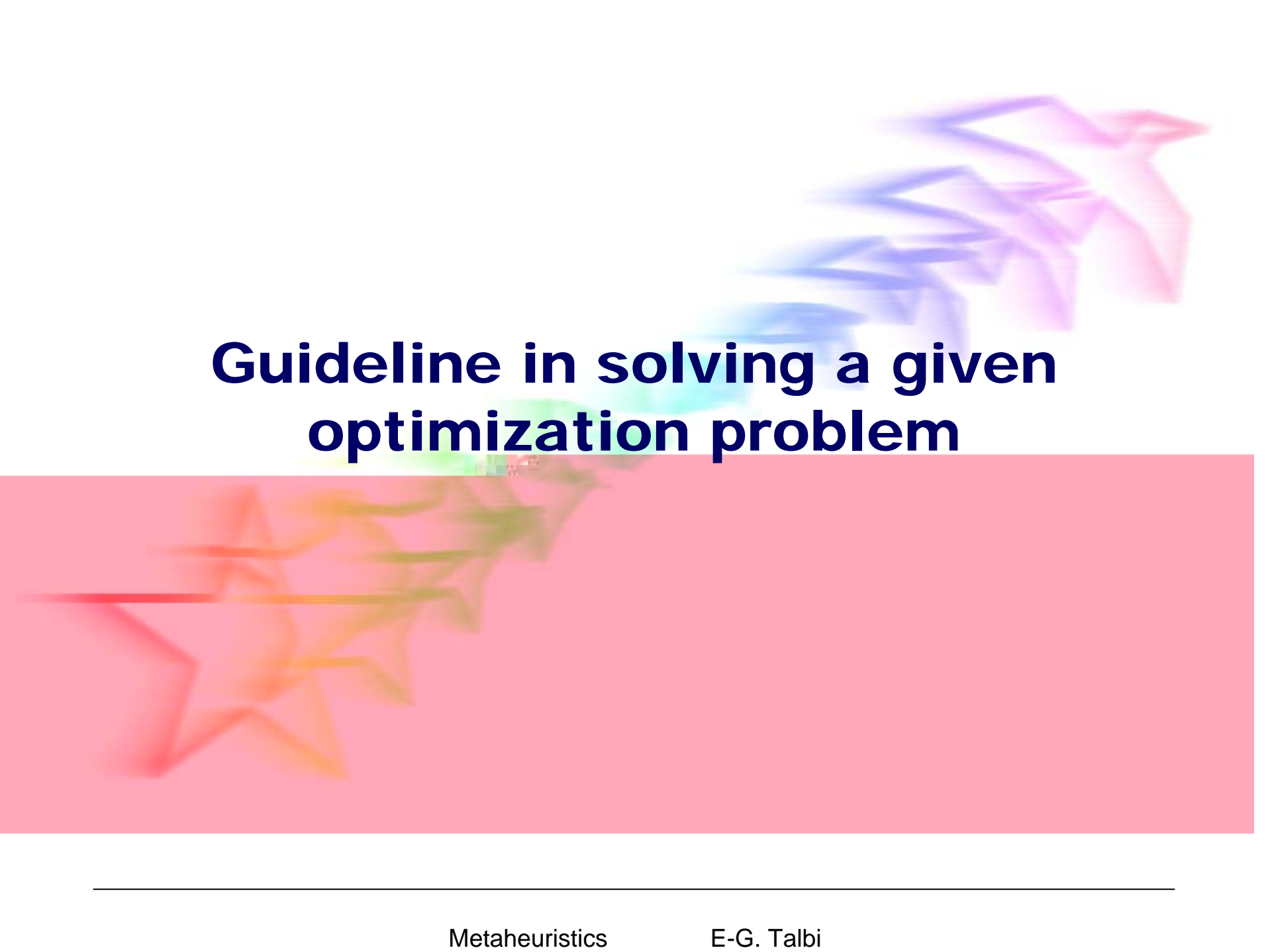
Single-solution-based metaheuristics
(local search, simulated annealing,
tabu search, iterated local search,
variable neighborhood search,
threshold accepting, ...)

Parallel, distributed
and hybrid metaheuristics

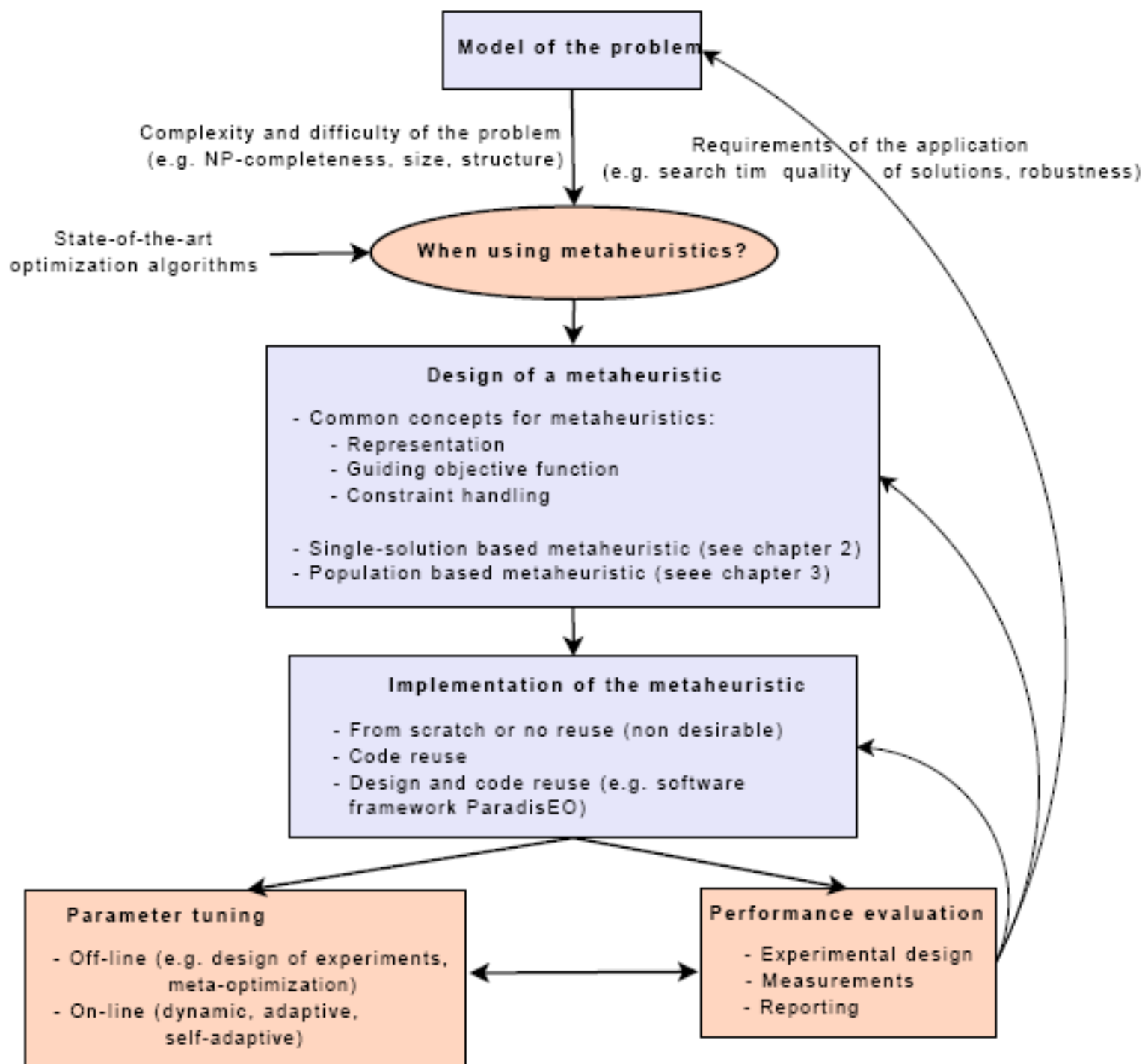


Population-based metaheuristics
(evolutionary algorithms,
particle swarm optimization, ant colony,
estimation of distribution algorithms,
differential evolution, ...)

Multi-objective
metaheuristics
(NSGA-II, SPEA2,
IBEA, ...)



Guideline in solving a given optimization problem





Exercises

- Representation and objective functions for the following problems
- Greedy heuristics

Ex : Magic Square

N=3

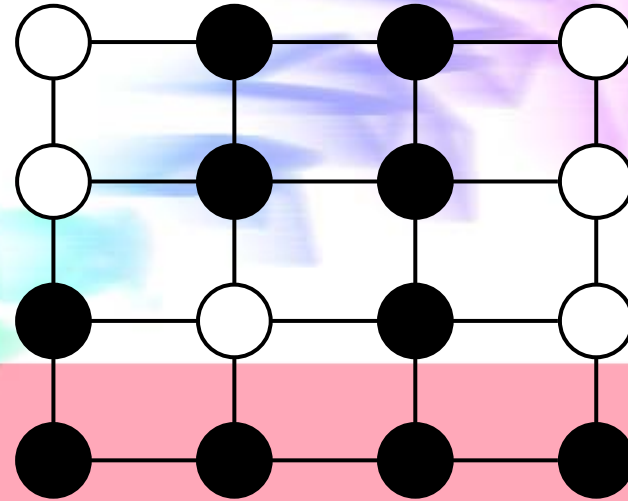
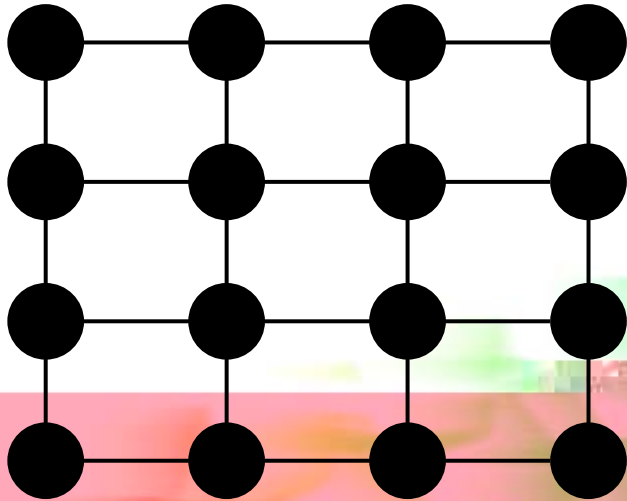
2	7	6	→15
9	5	1	→15
4	3	8	→15

15 ↙ ↓ ↓ ↓ ↘ 15

15 15 15 15 15

- Distinct N^2 integers over a matrix $N*N$
- Magic square of order N : Arrangement such that the N numbers in all rows, all columns, and both diagonals sum to the same constant (magic sum)
- The magic sum has the value $M(N) = (N^3+N)/2$.
- Magic squares of order $n = 3, 4, 5, \dots$, the magic constants are: 15, 34, 65, 111, 175, 260, ...

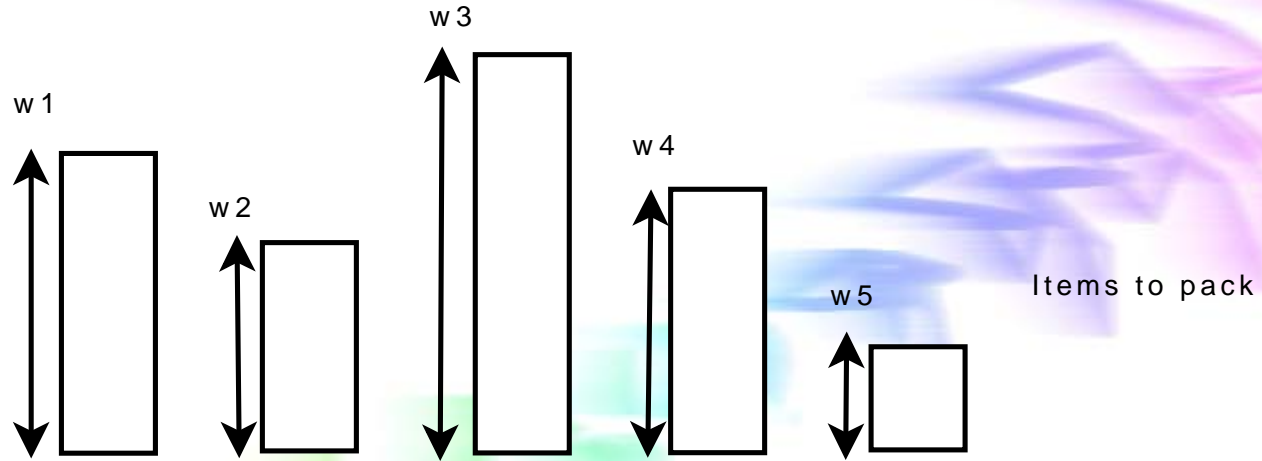
Ex : 'Casse-tete'



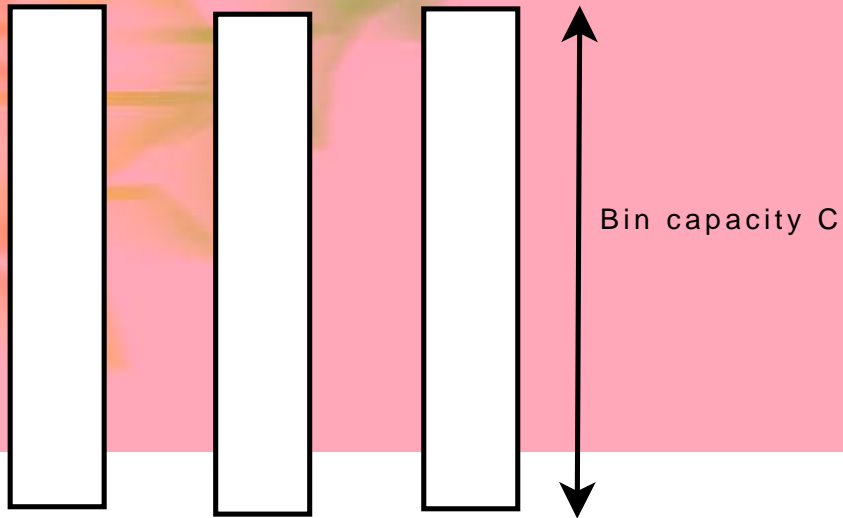
A non feasible solution

- 2D grid of size 4×4 covered by 16 jetons
- Each row and each column of the grid there will be a pair number of jetons.

Bin packing



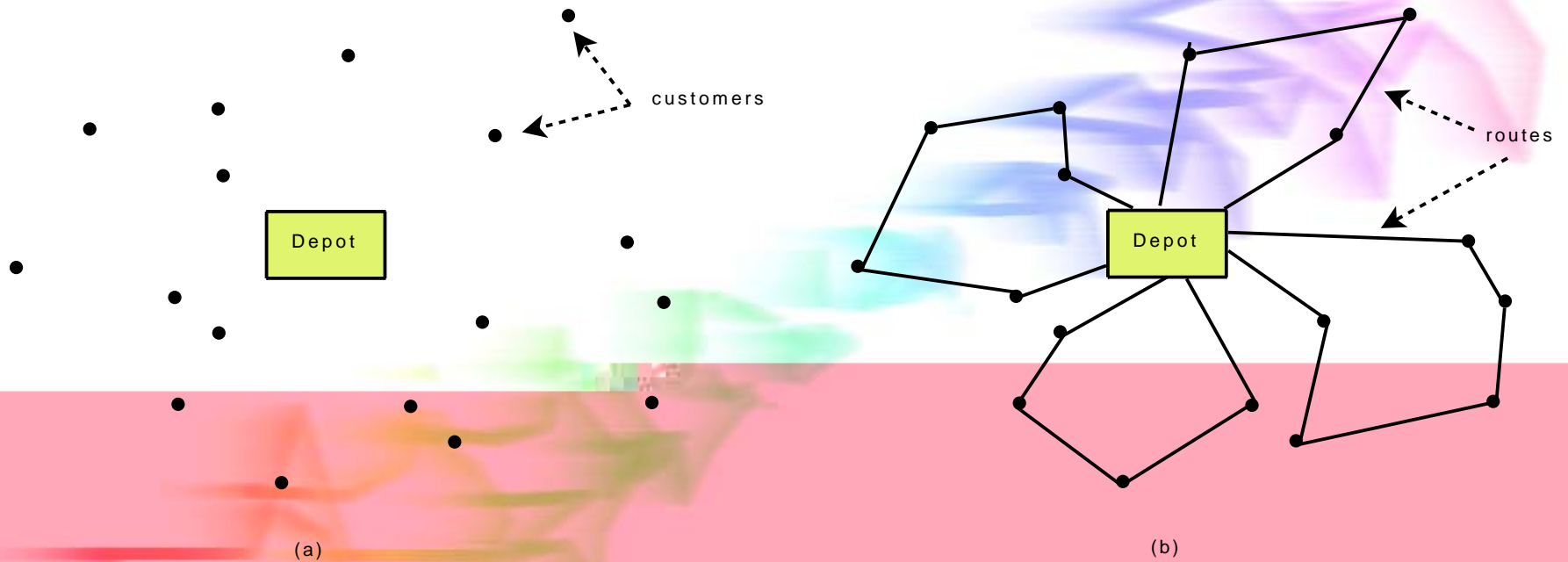
Bins and items have the same width



Metaheuristics

E-G. Talbi

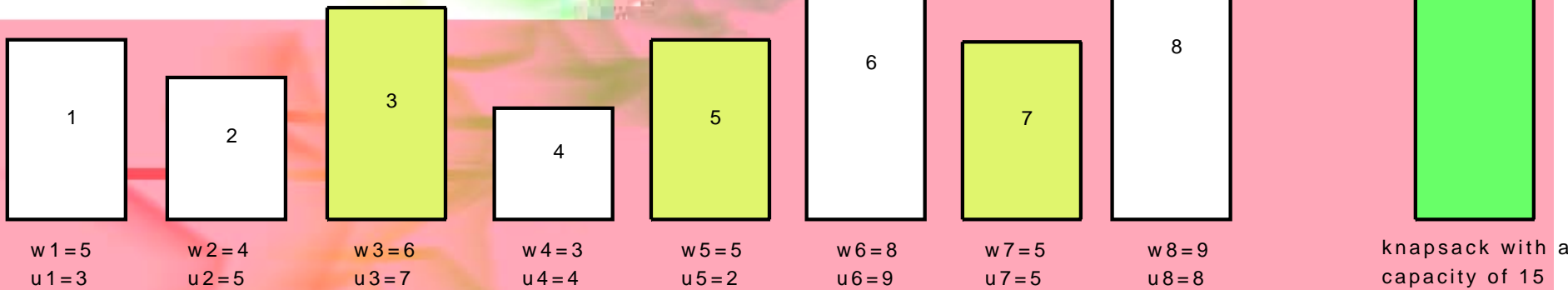
VRP: Vehicle routing problem



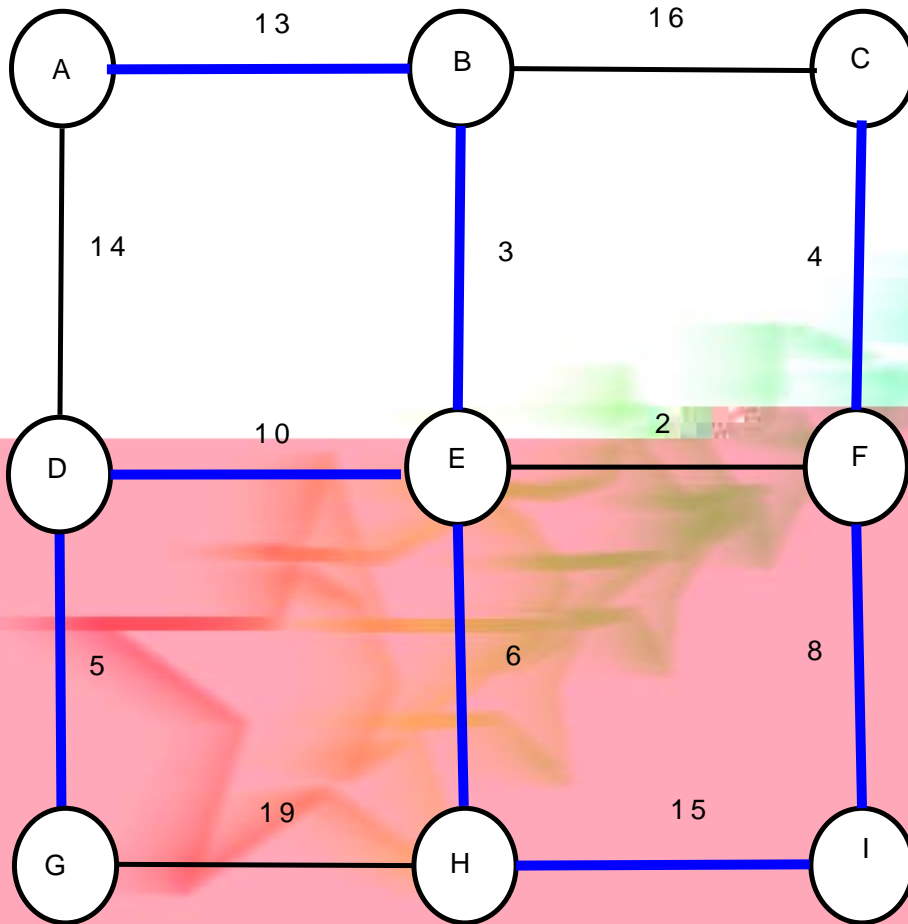
- Constraints :
- Objectives :

Greedy for knapsack

Constructed solution = (3,5,7)



Greedy for spanning tree



Constructed solution
(B,E), (C,F), (D,G), (E,H), (F,I), (E,D), (A,B), (H,I)