Artificial Intelligence

Alexandre CAMINADA
Polytech Nice Sophia

Contents

Let's talk about Artificial Intelligence

1. Insight about two intelligence inspired solvers, Tabu Search and Genetic Algorithms

2. Have some theoretical exercises about modelling and solving a decision problem during class

3. Do a practical work by applying both intelligent solvers on Travelling Salesman Problem and write a report on that

4. Some students will do a short presentation on TSP results during class

Schedule and evaluation

Date	Time	Who	Where	Content
Jan. 28	9-12:15	All	O+228 and remote on ZOOM	Lecture on decision making
Feb. 11	9-10 10:15 - 12:15	All 3 groups	E+142 E+142 / zoom / zoom	Tutorial on decision making
Feb. 18	9-10 10:15 - 12:15	All 3 groups	O+308 E+131 / zoom / zoom	Lecture on multicriteria Tutorial on multicriteria
Feb. 25	9-12:15	3 groups	O+308 / zoom / zoom	Practice on Tabu Search
Mar. 11	9-12:15	3 groups	O+308 / zoom / zoom	Practice on Evolutionary Search
Mar. 18	9-12:15	3 groups	O+303 / zoom / zoom	Delivery of reports on practice
May 20	9-10:30	All	Presential only	Exam

Evaluation :

- 30% on the report about Tabu Search practice
- 30% on the report about Evolutionary Search practice
- 40% on the exam
- Global note based on 10 points to complete with 10 points on machine learning

Team

Alexandre.caminada@univ-cotedazur.fr
Course responsible
Full professor
Lecture and in charge of one group

Ali.ballout@inria.fr
PhD student on AI
In charge of one group

Santiago.marrot@inria.fr
PhD student on AI
In charge of one group

Students

Different profiles

- Pure geek
- Pure maths
- Mixed

From

- EIT Data Sciences
- EIT Autonomous Systems
- Systèmes Informatiques (Software Engineering)

And

- 75% French native
- 25% Neither French or English native

Knowledge versus intelligence

Data: values ; capacity to show

Information: meaning of the data ; capacity to model something

Knowledge: we know what to do to solve the problem ; we can build correlation to find the good answer we need (what is the answer or the « decision ») ; capacity to learn

Intelligence: we create a new way to solve the problem, or to solve it more efficiently ; we find a new knowledge or a knowledge adaptation (what is the question) ; capacity to prospect and abstract

Conscience: we know our own limit in our intelligence to solve the problem and we know the surrounding of ourselve and of the problem (includes moral, ethic…) ; capacity to think

Where are the computer algorithms ?

Decision problem

Wikipedia « Decision aims at solving analytically or numerically problems which consist in finding the best element from a set, for a given criteria ».

In latina « optimisation » comes from « optimum » which means « the best ».

Optimisation problem (P)

Maximise or minimise the value of a given function (criteria or objective or fitness) which most of time is a time, money, effort, ressources… measurement.

Example

Function $f: X \rightarrow \mathbb{R}$ defined from X a set of values to $\mathbb{R}$ the set of real numbers.

Find one element $x*$ of X s.t. $f(x^*) \geq f(x), \forall x \in X$.

Formal writing is often

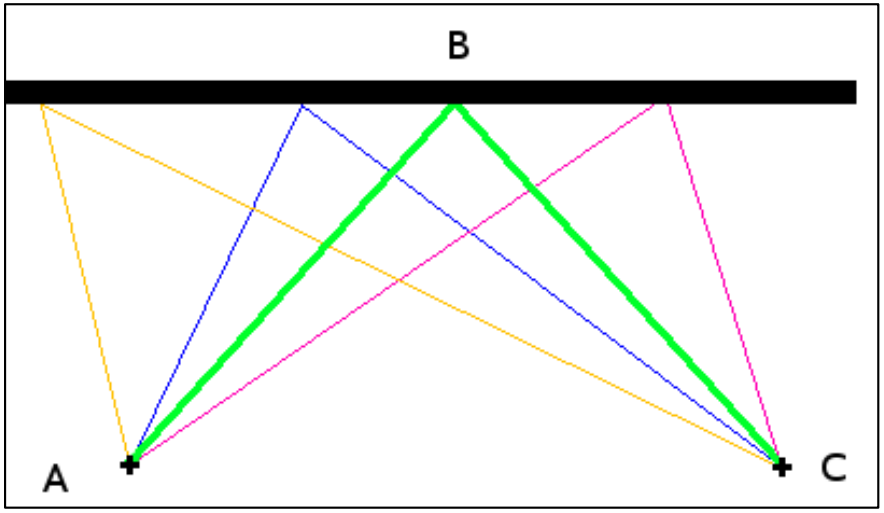
$$\inf_{x \in X} f(x) \quad or \quad \inf\{f(x): x \in X\} \quad or \quad \inf(X) \quad or \quad \begin{cases} \inf f(x) \\ x \in X \end{cases}$$

Decision problem

In the 1st century, Héron, mechanical engineer and maths scientist from Greece, exposes the principle of *finding the shortest path in the optic context for building construction in order to efficiently propagate the natural light.*

That's a decision problem.

Wikipedia « The shortest path to go from A to C passing through a point B on the bold line is obtained when the angle of incidence is equal to the reflected angle (in the figure, this is the green path).»



Decision problem

Two families

Continuous problems

The set X is a subset of $\mathbb{R}^n$ and f is continuous i.e. topological property of f saying that infinitesimal variations of the variable x give infinitesimal variations of $f(x)$.

The solvers come essentially from Analysis (differential calculus and convexity) and Linear Algebra (matrix computation). Important skills are gradient and surface of functions f and the linear applications which model the moves in geometry spaces.

A country wants to buy weapons and is talking to an international arms dealer. This one offers 2 types of packs. The first type of pack contains 100 guns, 200 bulletproof vests, 5 pistols and 50 bazookas. The second type of pack contains 50 guns, 100 bulletproof vests, 10 pistols and 100 bazookas. The 1st pack costs *p1* euros and the 2nd *p2* euros. The country wants to buy at least 1000 guns, 2500 bulletproof vests, 30 pistols and 250 bazookas.

x_1 is the number of pack 1 and x_2 is the number of pack 2.

The problem is a linear program: *Minimise* $p_1 x_1 + p_2 x_2$

Under the constraints: $100x_1 + 50x_2 \geq 1000$ $200x_1 + 100 x_2 \geq 2500$

$5x_1 + 10x_2 \geq 30$ $50x_1 + 100 x_2 \geq 250$

$x_1 \geq 0$ *and* $x_2 \geq 0$ *and* x_1 *and* x_2 *are real numbers (or integer if we discretize the decision)*

It is a linear program because the constraints and the fitness are.

Decision problem

Two families

Discrete problems (or combinatorial problems)

The set X is discrete, $X \subset \mathbb{Z}^n$, finite or countable. There is no notion of continuity.

The solvers come mainly from graph theory.

Easy in theory: try all the solutions, and compare their qualities to find the best. But in practice, the enumeration of all the solutions may take too much time.

In a traveling salesman problem, a salesman must visit a number of cities on the shortest route. In the figure the salesman must visit 18 cities from Paris. This problem is modeled by a valued graph whose vertices are the cities and the edges the connections between these cities. These edges are valued by the kilometric distances.

We look forward an Hamiltonian cycle (a cycle which only travels once in any node) of minimal score. The solving is factorial, $n!$, $17! = 3{,}6 \times 10^{14}$ knowing that 1 year is $3{,}16 \times 10^{13}$ microsecondes. With only 7 more cities: $24! = 6{,}2 \times 10^{23}$.

For n cities, there are $(n-1)!$ circuits.

Decision problem

It is unrealistic to use an enumeration method in the combinatorial domain when the resulting complexity is exponential.

The following table reports enumeration times with a computer calculating each solution in a microsecond (10^-6 s).

	10	20	40	60
n	10^{-5}s	2.10^{-5}s	4.10^{-5}s	6.10^{-5}s
n^2	10^{-4}s	4.10^{-4}s	16.10^{-4}s	36.10^{-4}s
n^3	10^{-3}s	8.10^{-3}s	64.10^{-3}s	216.10^{-3}s
n^5	0,1s	3,2s	1,7 mn	13 mn
2^n	10^{-3}s	1s	12,7 days	366 cent.
3^n	0,059s	58 mn	3855 cent.	$1,3.10^{13}$ cent.
$n!$	3,63s	771 cent.	...	...

Decision problem

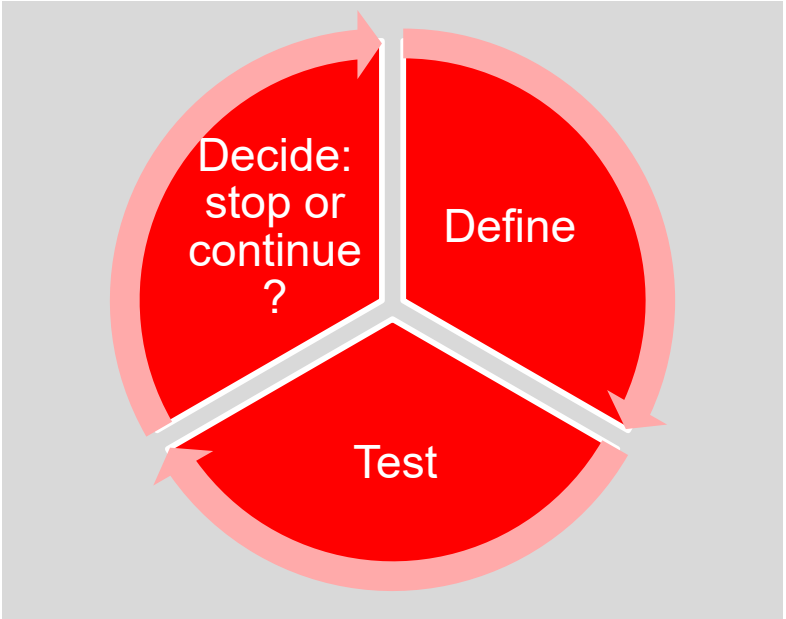
How does a team deal with this kind of problem?

There are three stages.

1/ The 1st step is to model the problem and determine the criterion (s).

2/ The 2nd step is the resolution, that is to say the determination of a solution that seems good (it will say abusively optimal).

3/ The 3rd step is a discussion with the executive decision maker and, if necessary, a return to the first step.



Do we always know how to solve those problems?

Two categories:

1/ Easy problems: small sizes (enumerate), very constrained (few solutions) and polynomial (there is an algorithm) => math's solvers

2/ Difficult issues: the opposite ; use a method which often replicate or include human expertise => artificial intelligence solvers

Decision problem

Class objective

No pretention on testbed: there are thousands of problems and man generates new ones every days.

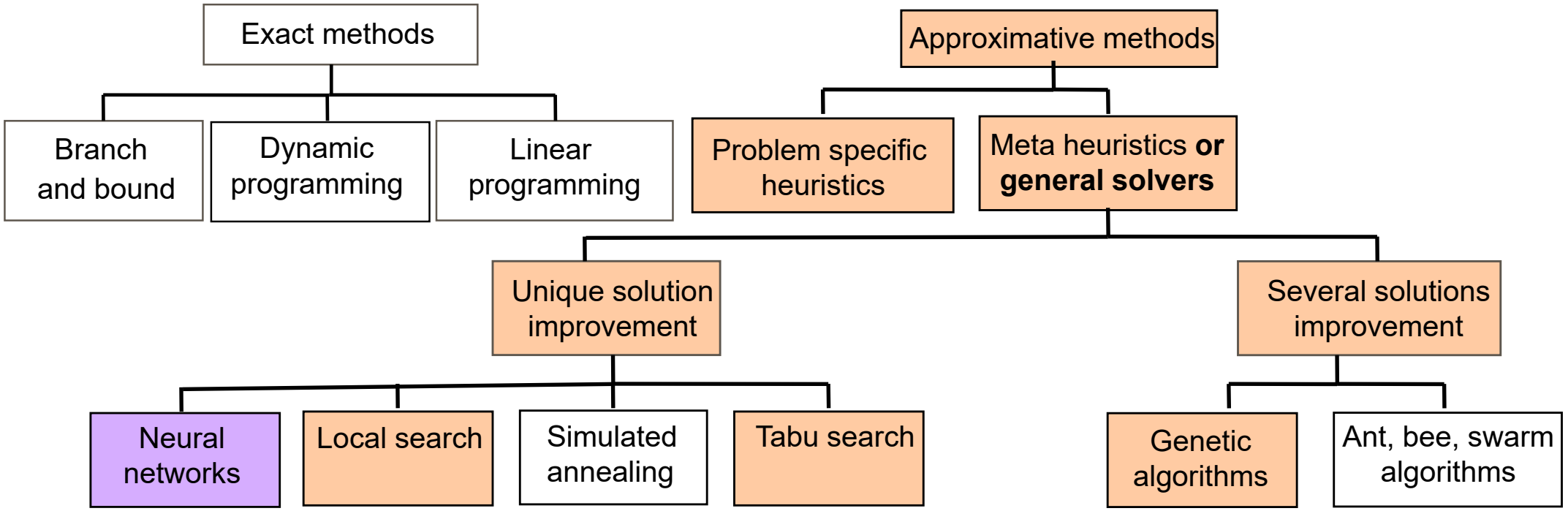
No claim on methods: some work well for some problems but are very bad for others.

Raise awareness of the complexity of the problems and the danger of combinatorics: we can design beautiful models perfect for “school examples” but unusable on real cases.

Raise awareness of a very large number of situations that can be addressed in companies and solutions that an engineer can provide even if the experts have a method that "works".

Decision methods

- Exact algorithms come from maths ; they find the optimal solution and they prove it (we say that the resolution is complete)
- Approximative algorithms come from nature's mimetic ; they find solutions, we hope good, but they do not prove they are optimal (we say that the resolution is uncomplete).



These algorithms start from an initial solution and try to improve it iteratively looking for a best one around the current one

These algorithms start from many initial solutions and try to combine them to create new possibilities

Content

1. Heuristics and meta solvers
2. Tabu Algorithms
3. Evolutionary Algorithms
4. Multicriteria decision

Heuristics – Définitions

Heuristics

- Empirical rules
- Intuitive judgments
- Rules of common sense... (Intelligence? Knowledge?)

Judea Pearl

Strategies, vaguely applicable, to control the problem solving by humans or machines.

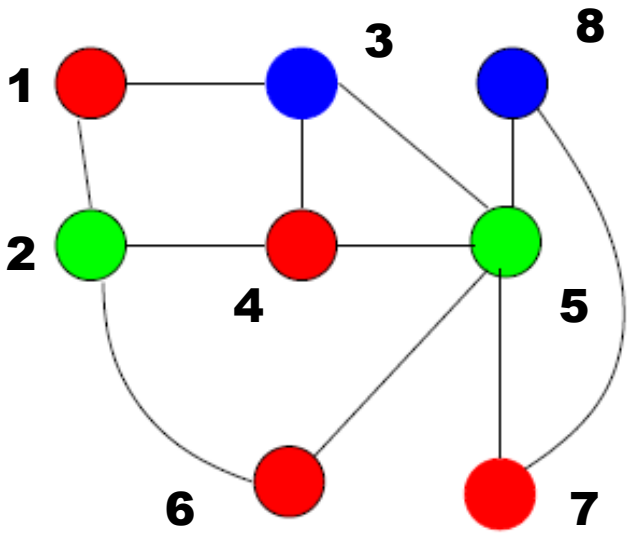
Used permanently in society; car driving, discussion..., complex tasks where detailed and precise knowledge of the environment is impossible.

For decision problem

- Heuristics: experiment-based resolution algorithm not necessarily providing an optimal solution

Heuristics – Example on Graph Coloring problem

Constraint satisfaction: assigning different colors to adjacent nodes



$G(V,E) a\ graph$
$f, coloring\ function\ s.t.$
$(v1, v2) \in E, f(v1) \neq f(v2)$

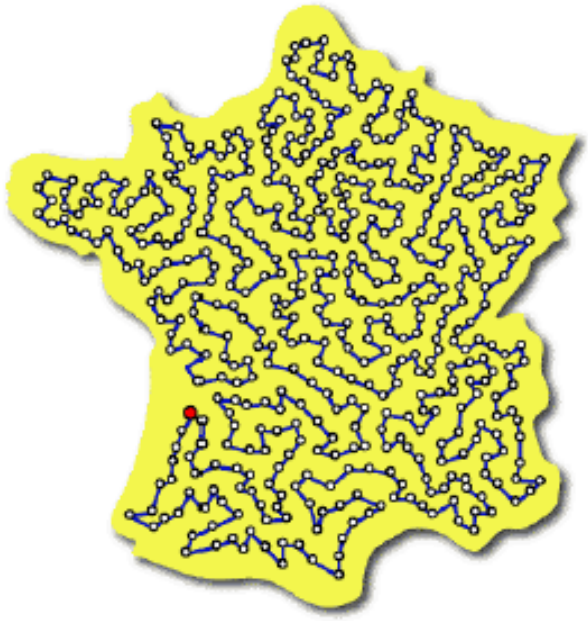
Heuristic: order the colors (R, G, B) and to color the next node, choose the most used valid color

Objective: keep colors less used to solve the following constraints

Heuristics – Example on Travelling Salesman problem

Minimization function: find a Hamiltonian circuit (closed path passing through all the summits once and only one) of minimum length

Space size: $(n-1)!$
with 50 cities, the number of options is 49!
that is $6{,}08.10^{62}$ cases

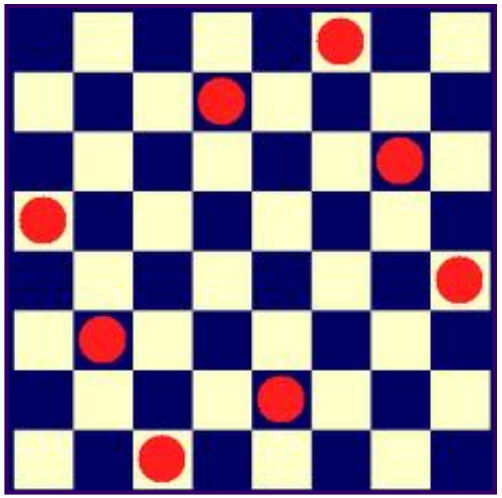


Heuristic: take the next nearest town

Objective: to avoid long distances between two cities

Heuristics – Example N-Queens problem

Combinatorial function: put *n* queens on a *nxn* chessboard but none should be taken



$$A_{64}^{8} = \frac{64!}{56!} = 178\ 462\ 987\ 637\ 760$$

possible arrangements with one queen per cell

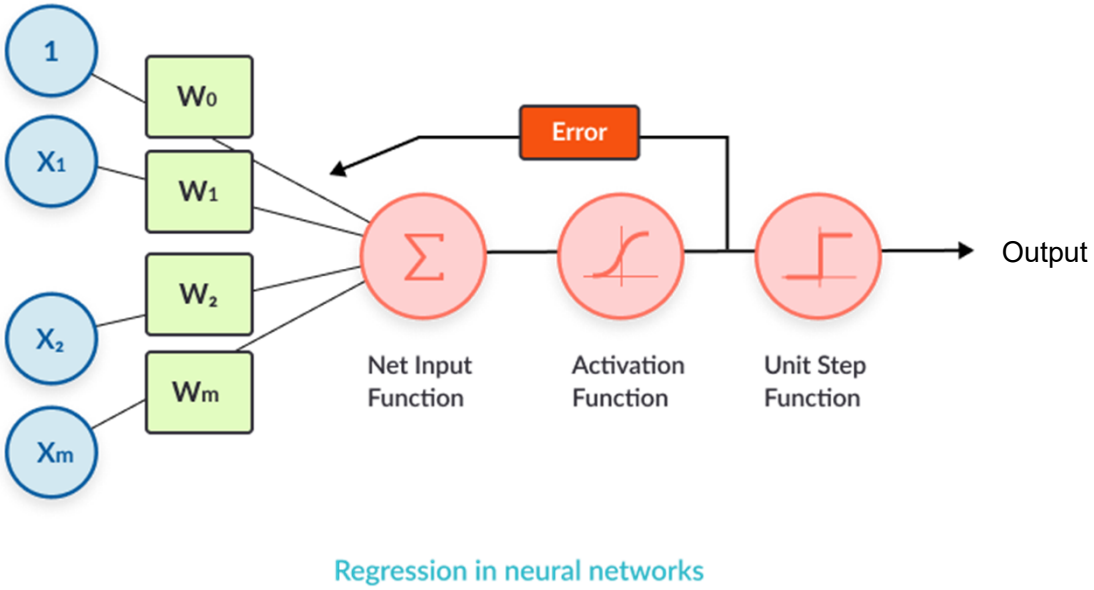
With one queen per line
$8^8 = 2^{24} = 16\ 777\ 216$ possibilities

Heuristic: choose the column that leaves the greatest number of free cells for the lines below

Objective: to leave as much freedom as possible on the chessboard

Heuristic – Example on Neural Network problem

Correlation function: for a given set of data described by a vector of *m* input values and one output, find the weights minimizing the error of the output computation



Regression in neural networks

w_i are between 0 and 1 with a precision of 10^-10

There are 10^10 combinations for each w_i

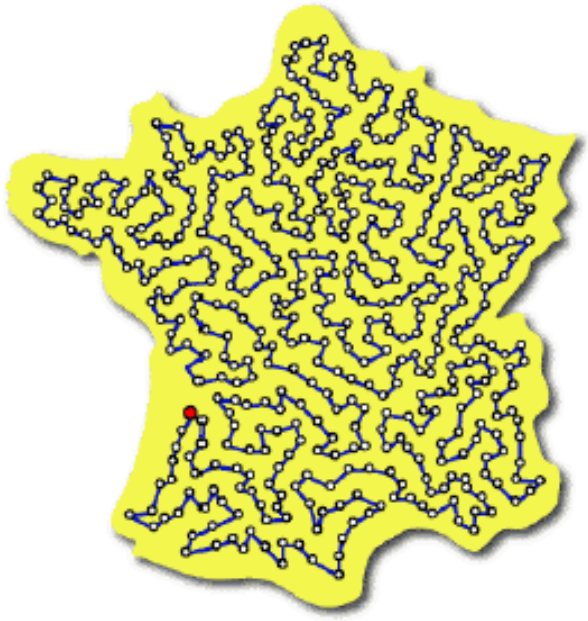
Then 10^10^(m+1) combinations of weight

Heuristic: change the value of w_k which influences the most the output error

Objective: reduce the error for the next presentation of the data

Heuristic - Example of a multicriteria problem

Two opposite decision: find a Hamiltonian circuit (closed path passing through all the summits once and only one) of minimum time and minimum fuel



Both objectives are opposite:
Minimum time requires high speed roads
Minimum fuel requires low speed roads

Heuristic: take the next nearest town if the time or the fuel is reduced

Objective: to get a circuit which is a compromise between both objectives

Heuristics – To construct or to improve

Solving a problem involves finding the assignment of all the variables (give a value to each variable) that maximizes / minimizes the fitness function.

X is the set of solutions to the problem ; $x \in X$, one of the solutions of the problem ; x consists of n elements, n being the number of unknown variables.

1/ We want to construct a solution from scratch

- We search an initial (or first) solution to a problem, even not the best
- All variables must be assigned, randomly, or by one heuristic s.t.:

- Coloring: choose the more used valid color
- Salesman: choose the nearest city
- Queen: choose the cell leaving the largest number of free cells

2/ We want to improve a current solution

- We search a new solution derived from a current one (it is called a neighbor)
- One or several variables must be changed, randomly, or by one heuristic s.t.:

- Coloring: move the color of the more conflicted node
- Salesman: move a city from one place to another
- Queen: move the column of the worst queen

In all cases, the decision is: choose a variable and choose a value to assign to the variable

Heuristics – To construct or to improve

1/ Constructive methods

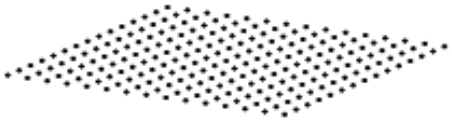
- *A building method works step by step to generate* $x = (<x_1,v_1> <x_2,v_2>...<x_n,v_n>)$ *with* x_i *one unknown variable and* v_i *its value.*
- *Starting from scratch,* $x = ()$*, at each step the current partial solution* $x = (<x_1,v_1>...<x_{i-1},v_{i-1}>)$ *is extended , with* $i \leq n$.
- *One extension consists in choosing the next variable to assign and to choose its value in its domain definition, then to add* $<x_i,v_i>$ *to x.*
- This process is repeated until a complete solution is obtained, optimal or not.
- The previous heuristics are all constructive methods.

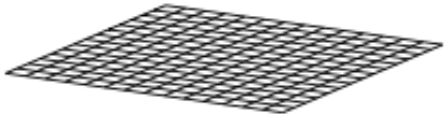
2/ Neighborhood methods

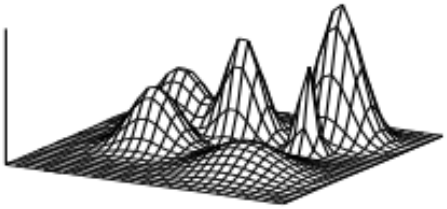
- *A neighborhood method defines how we move from one complete solution* $x \in X$ *to another one x'. Neighborhood means that we define a topology.*
- *A function N associates a subset of X, notice N(x), to any solution* $x \in X$*. A solution* $x' \in N(x)$ *is then called a neighbor of x.*
- *The set of solutions X is then an oriented graph defined by N: G(X,A) s.t.* $\forall (x, x') \in X^2$, $<x, x'> \in A$ *if and only if* $x' \in N(x)$.
- This process is repeated until the time is over or the optimal solution is found.
- The previous heuristics are all neighborhood methods.

Heuristics – Solutions, neighborhood, fitness

In mathematics, the set of parts of a set refers to the set of subsets of that set. If X is a set of solutions to a problem to solve, a neighborhood of X is then a function $N: X \rightarrow P(X)$ and $P(X)$ is of size $2^{|X|}$.







We call a landscape, the triple (X, N, f) with:

- *X, the set of all solutions*
- *$N: X \rightarrow 2^{|X|}$, the neighborhood relation s.t. :*
  $N(x)=\{x' : Hamming(x,x')=d\}$
  saying that:
  only one operation is suffisant to move from x to x'
  NB: very often d=1
- *$f: X \rightarrow \mathbb{R}$, the fitness to optimise*

Neighborhood example [I. Devarenne 2007]

- A neighborhood structure defines all the solutions that are close to a given solution
- It is a subset of directly attainable solutions by performing a given transformation on the solution
- Example : one solution is a triple of binary variables ; from one solution *x*, we define two sets of neighborhood solutions, $N_1(x)$ and $N_2(x)$

Set of solutions got from N_2(x): (0,1,1) (1,1,0) (0,0,0)

↑ N_2: neighbors with 2 moves

Initial solution x=(1,0,1)

↓ N_1: neighbors with 1 move

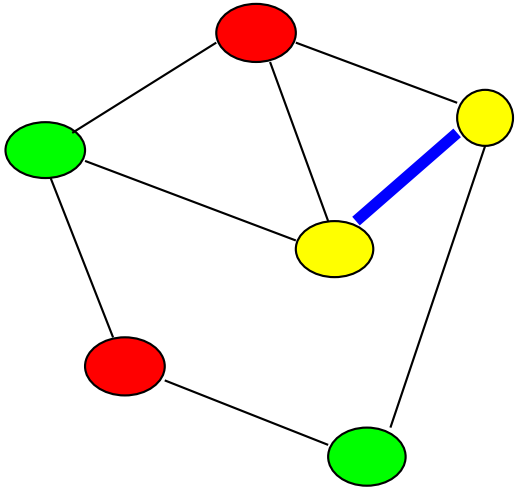
Set of solutions got from N_1(x): (0,0,1) (1,1,1) (1,0,0)

Advantage of N_2: the search is faster as we change two variables in one move

But none move with N2 allows the algorithm to get (1,0,0) with less than 4 moves while N_1 gets it faster

If (1,0,0) is the optimum to our problem, N_1 is better than N_2 while N_2 seems to be more powerful

Neighborhood example [I. Devarenne 2007]



Graph coloring

- Assign one color per node

Conflict

- When two adjacent nodes use the same color
- Plus conflict are the node with the largest number of conflit

Valid coloration

- Coloration without conflict

Chromatic number

- Minimum number of colors to assign a valid coloration

Neighborhood example [I. Devarenne 2007]

Two simples neighborhood functions (NB: the search move to a neighbor solution even if it is worse than the current one)

Move Conflict Node

- Node chosen among all the nodes in **C**onflict
- The new color is the **L**east used by neighboring nodes

Move Plus conflict Node

- Node chosen among the **P**lus conflicting nodes
- The new color is the **L**east used by neighboring nodes

Easy cases to deal with (i.e. 1 radio channel is assigned to 1 transmitter)

Neighborhood example [I. Devarenne 2007]

Problem x.y.z where

x, the number of colors,

y the number of nodes,

z the graph density

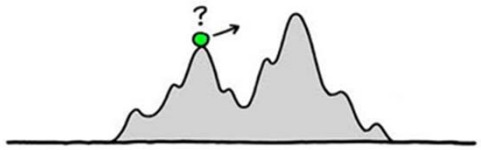
Columns:

s : number of success over 10 runs

it : mean number of iterations in case of success run

c : mean number of conflicts in case of failed run

The best performance are in grey.

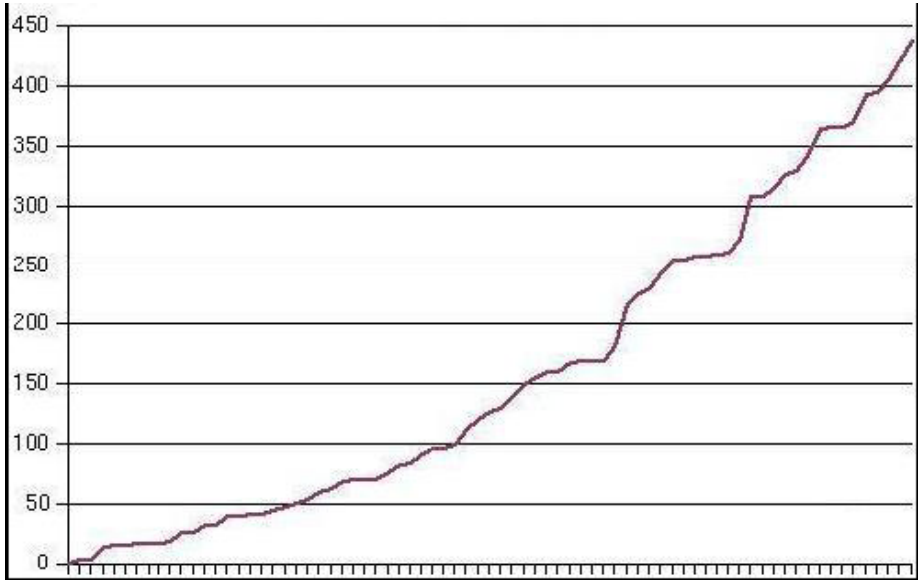


	Conflict Node			Plus Conflict Node		
Problems	s	it	c	s	it	c
4.75.10	10	661	-	10	413	-
4.75.20	0	-	40	4	295	34
4.75.30	10	583	-	0	-	114
8.75.10	10	29	-	10	20	-
8.75.20	10	90	-	10	70	-
8.75.30	10	1110	-	8	618	11
8.150.10	10	148	-	10	114	-
8.150.20	0	-	47	0	-	46
8.150.30	0	-	159	0	-	120
15.150.10	10	110	-	10	47	-
15.150.20	10	118	-	10	88	-
15.150.30	9	743	1	8	433	10
15.300.10	10	220	-	10	162	-
15.300.20	0	-	38	0	-	47
15.300.30	0	-	316	0	-	197

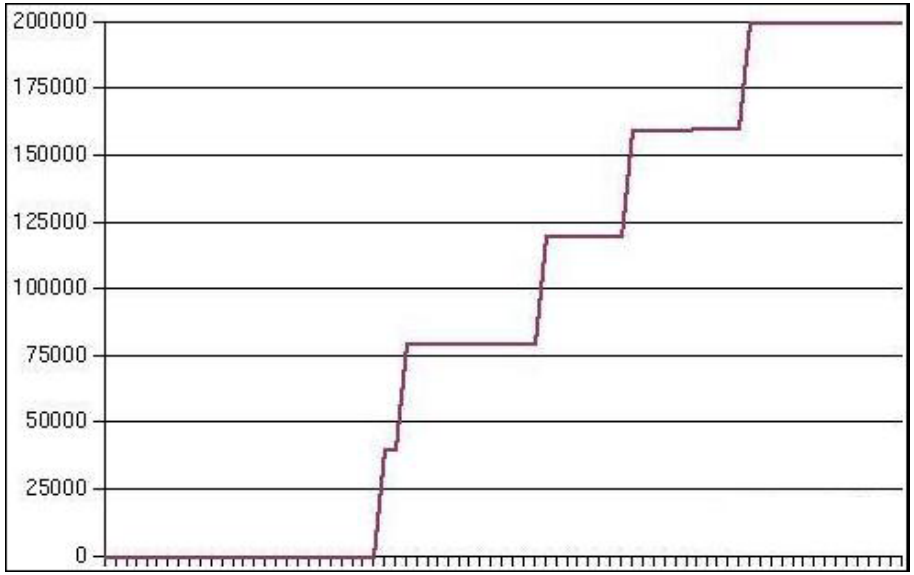
Data analysis

Plus conflict Node neighborhood data.
On the abscissa, the nodes by increasing degree.
On the ordinate, the number of visits per node.

Success run: optimum found



Failed run: optimum not found



Appearance of loops: set of nodes are used repeatedly during the search => loose of efficiency

Introduction of adaptive process

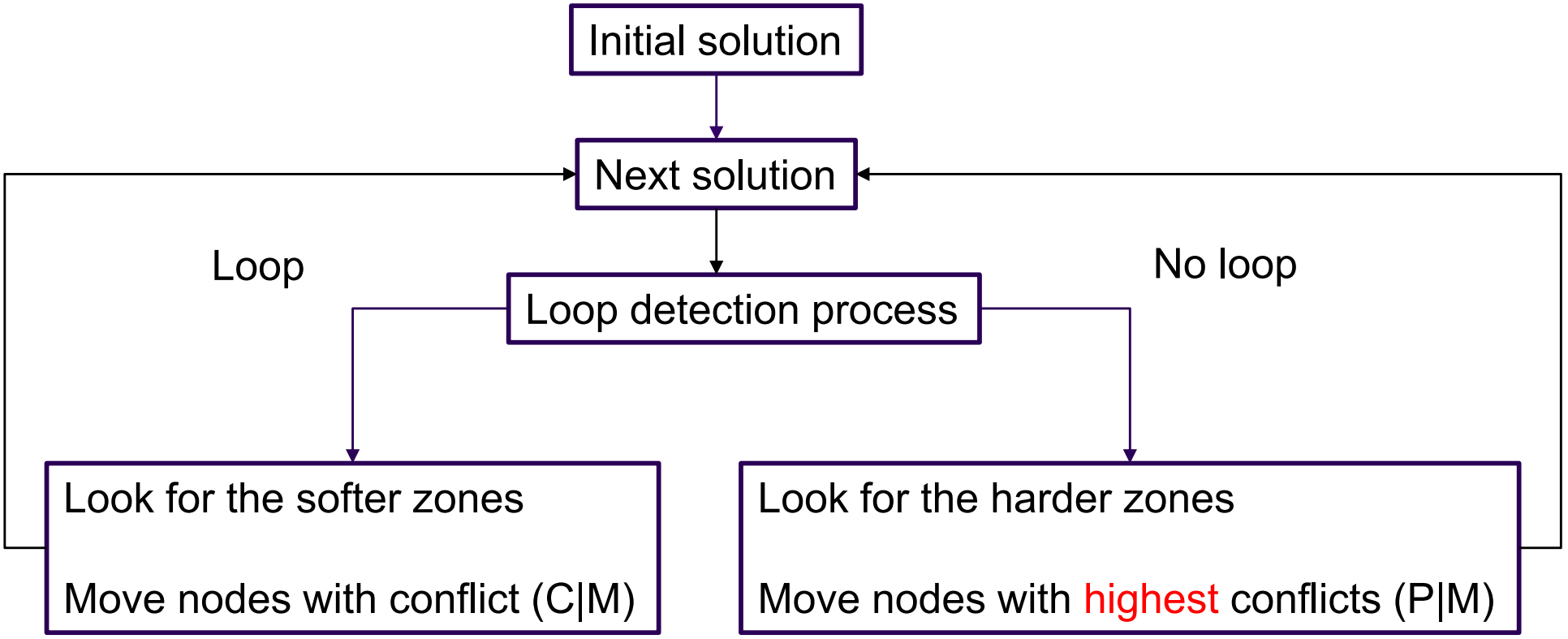
When to adapt? Appearance of loops

- Excessive repetition of a choice during the search
- Symptom of difficulty getting out of a search area

How to adapt? Loop detection and neighborhood change

- Redefining the area of neighboring solutions
- Combining methods of choosing variables

Introduction of adaptive process



Parameters to fix for the loop management:
-how many iterations may be considered as a loop?
-what is the time windows?
-how many common nodes in the time window?

Adaptive neighborhood

Problem x.y.z where

x, the number of colors,

y the number of nodes,

z the graph density

Columns:

s : number of success over 10 runs

it : mean number of iterations in case of success run

c : mean number of conflicts in case of failed run

The best performance are in grey.

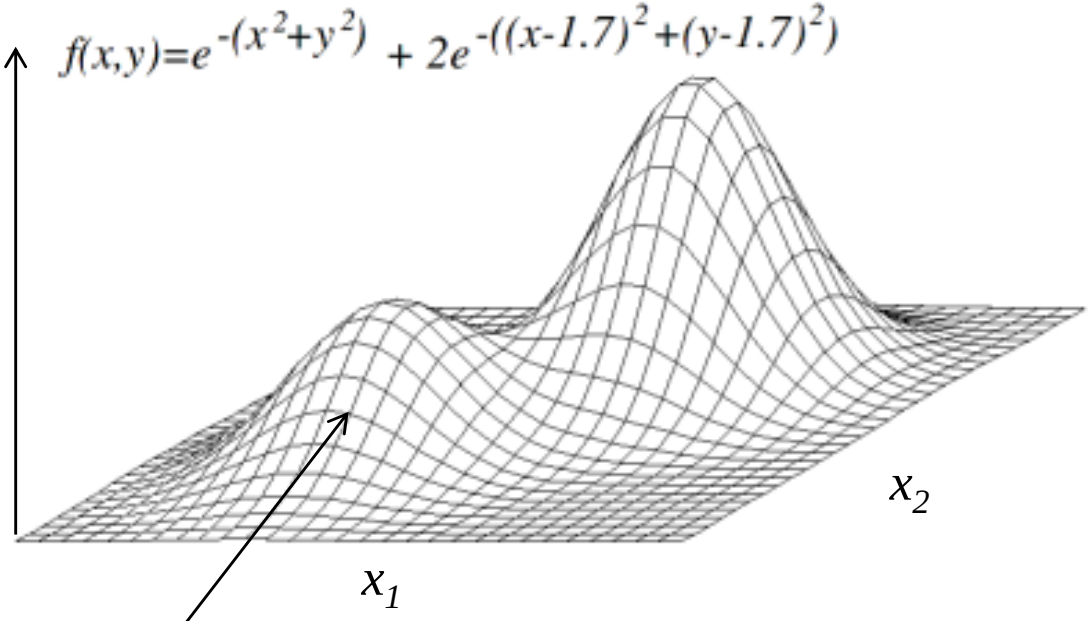
	Conflict Node			Plus Conflict Node			Adaptive method		
Problems	s	it	c	s	it	c	s	it	c
4.75.10	10	661	-	10	413	-	10	499	-
4.75.20	0	-	40	4	295	34	10	394	-
4.75.30	10	583	-	0	-	114	10	297	-
8.75.10	10	29	-	10	20	-	10	23	-
8.75.20	10	90	-	10	70	-	10	67	-
8.75.30	10	1110	-	8	618	11	10	904	-
8.150.10	10	148	-	10	114	-	10	108	-
8.150.20	0	-	47	0	-	46	10	98625	-
8.150.30	0	-	159	0	-	120	10	1319	-
15.150.10	10	110	-	10	47	-	10	44	-
15.150.20	10	118	-	10	88	-	10	101	-
15.150.30	9	743	1	8	433	10	9	463	1
15.300.10	10	220	-	10	162	-	10	173	-
15.300.20	0	-	38	0	-	47	10	53780	-
15.300.30	0	-	316	0	-	197	10	57283	-

Heuristics – Optimum and neighborhood

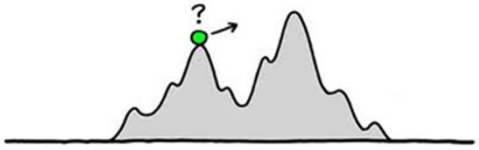
For a given neighborhood N

A solution $x \in X$
is a local maximum
if $f(x') \leq f(x)$ for all $x' \in N(x)$.

A solution $x \in X$
is a global maximum
if $f(x') \leq f(x)$ for all $x' \in X$

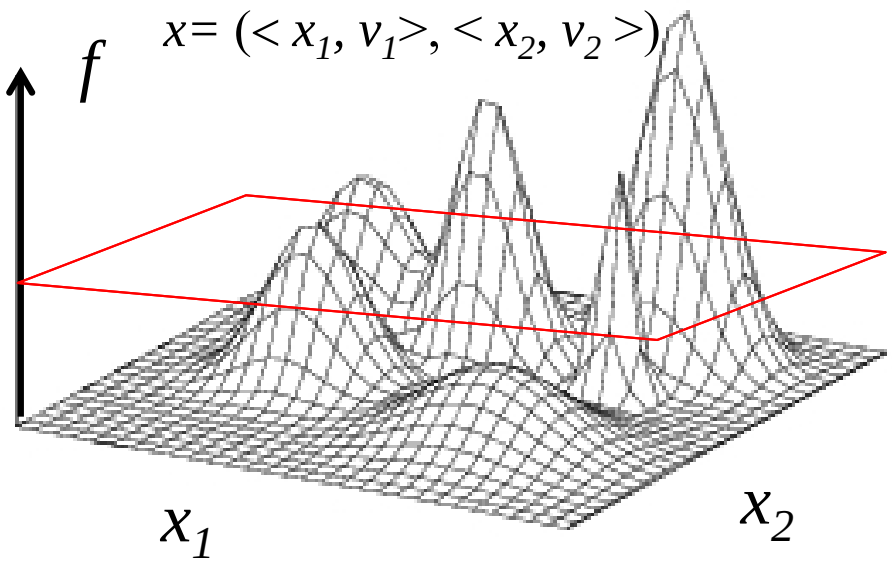


$x = (< x_1, v_1 >, < x_2, v_2 >)$



Heuristics – Data vizualisation, search space and fitness

Maximize *f* with 2 variables

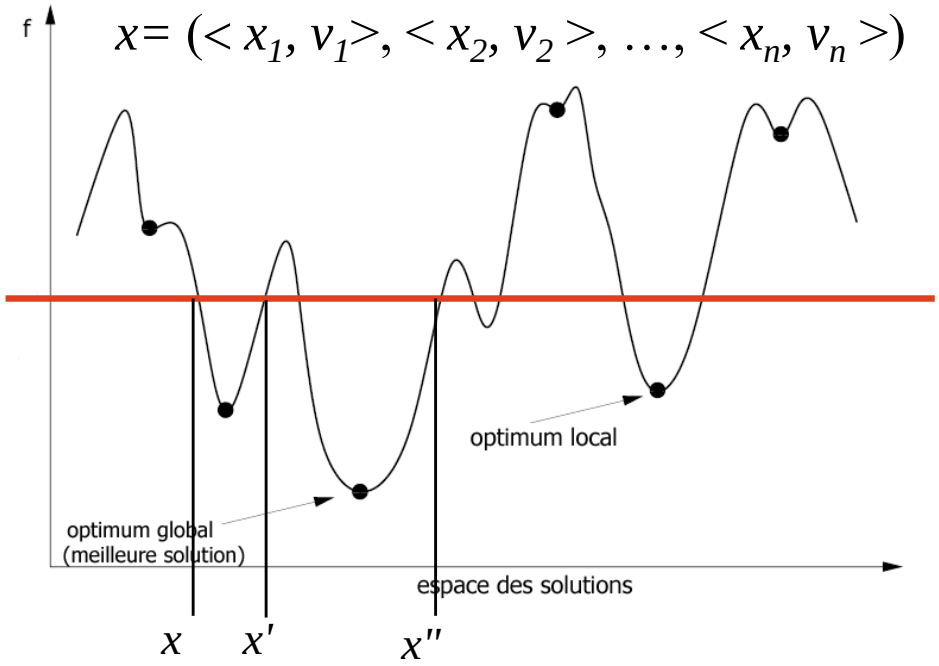


Vizualise a multimodal function with only two variables: **easy**.

The converge to one local optimum depends on the mountain base, not on the mountain height: **difficult.**

Attraction concept: a problem with many small basins is difficult.

Minimize *f* with *n* variables, n> 2



Vizualise a multimodal function with more than two variables: **difficult**.

A lot of solutions have the same fitness function value but are not in the same area of solution space and we cannot figure them.

Heuristics – Local search

From Sébastien VEREL

Let the problem to solve:

- *X, the set of solutions (search space)*
- *N(x), the set of neighborhood solutions of $x \in X$*
- *f: X → $\mathbb{R}$, the function to maximise*

Local search general algorithm

Choose the initial solution $x \in X$
Repeat
    choose $x' \in N(x)$
    if *$f(x) \leq f(x')$* ***store*** *x'*
    $x \leftarrow x'$
Until *(stopping criteria true)*
Return *x' as the best solution find*

This algorithm is independant of the problem to solve. It is a general problem solver.

Heuristics – Local search

From Sébastien VEREL

Let the problem to solve:

- *X, the set of solutions (search space)*
- *N(x), the set of neighborhood solutions of $x \in X$*
- *f: X → $\mathbb{R}$, the function to maximise*

Local search of maximal exploration: the random walk

Choose the initial solution $x \in X$
Repeat
choose randomly $x' \in N(x)$
if *$f(x) \leq f(x')$* ***store*** *x'*
$x \leftarrow x'$
Until *(given number of iterations)*
Return *x' as the best solution find*

Not efficient but used to sample a new problem in a given time.

Heuristics – Local search

From Sébastien VEREL

Let the problem to solve:

- *X, the set of solutions (search space)*
- *N(x), the set of neighborhood solutions of $x \in X$*
- *f: X → $\mathbb{R}$, the function to maximise*

Local search of maximal exploitation: the steepest descent hill climbing

Choose the initial solution $x \in X$
Repeat
choose $x' \in N(x)$ s.t. $f(x')$ is maximal
if *$f(x) \leq f(x')$* ***store*** *x'*
$x \leftarrow x'$
Until *(local optimum reached ⇔ f is not improved)*
Return *x' as the best solution find*

Efficient to sample several local optima but it takes time.

Heuristics – Local search

From Sébastien VEREL

Let the problem to solve:

- *X, the set of solutions (search space)*
- *N(x), the set of neighborhood solutions of $x \in X$*
- *f: X → $\mathbb{R}$, the function to maximise*

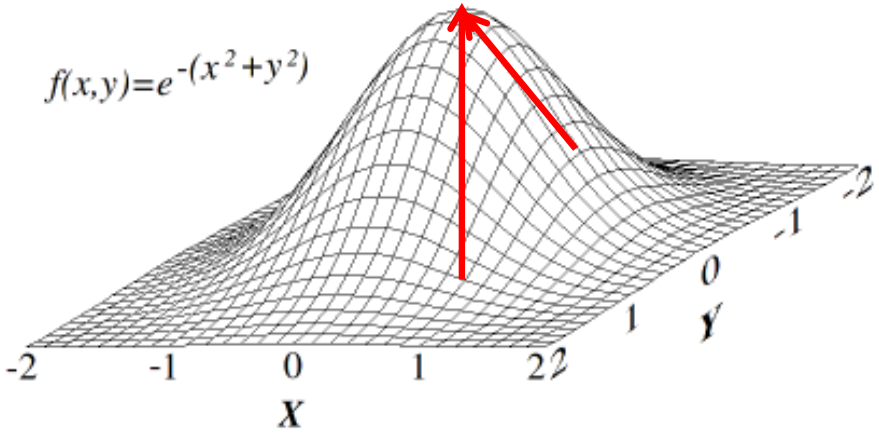
Local search of maximal exploitation: the stochastic hill climbing

Choose the initial solution $x \in X$
Repeat
choose randomly $x' \in N(x)$
if *$f(x) \leq f(x')$* ***then*** *$x \leftarrow x'$*
Until *(local optimum reached or given number of iterations reached)*
Return *x' as the best solution find*

Efficient if we have no time for long run.

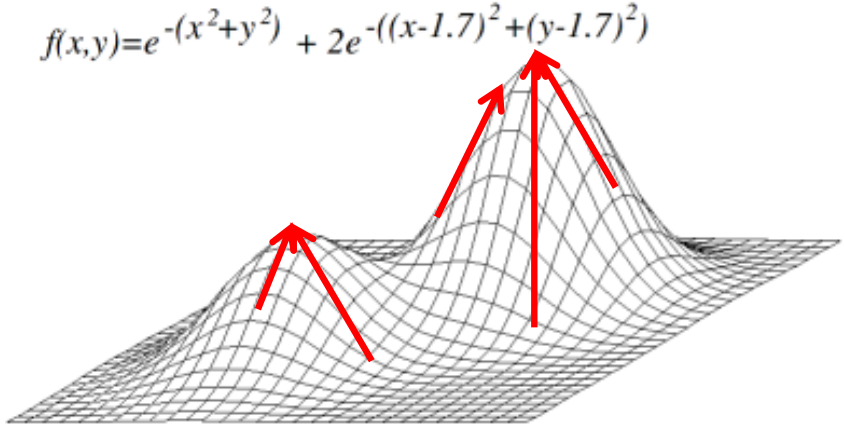
Heuristics – Spaces and functions

http://en.wikipedia.org/wiki/Hill_climbing



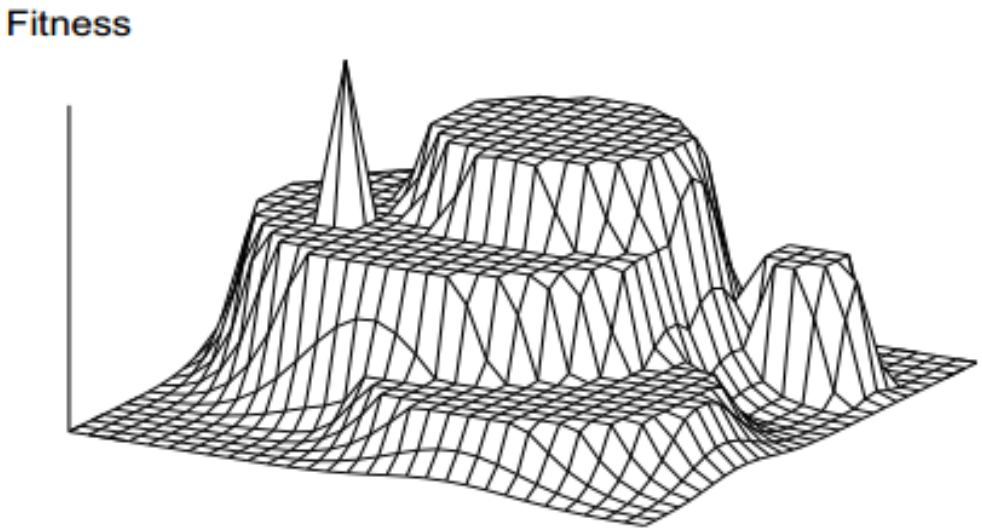
A convex surface. Hill-climbers are well-suited for such surfaces, and will converge to the global maximum.

With local search: use *restart* with different initial solutions to get different local minimal (***random-restart hill climbing***)



A surface with two local maxima. (Only one of them is the global maximum.) If the hill-climber begins in the wrong area, it converge locally.

Heuristics – Spaces and functions



In discrete problems (TSP, graph-coloring…), a lot of functions are locally flat: neutral landscape i.e. subset of solutions which seem equivalent.

Local searches are not efficient for these functions i.e. random drift phenomena.

How to learn from the run – Learning from sampling

1/ One algorithm iteration (and run) is an experience.

2/ Local search is fine to search but it does not learn.

3/ Try to improve it through a meta process called « meta heuristic » dedicated to learn.

The experience is a continuous sampling and assesment of situations (kindof DevOps process)

- « Build, test, deploy, monitor », and do it again with a reinforcement mechanism (« learn »).
- The fitness is better ? Keep the same « move strategy ».
- Else, change it, or start again with a different starting point.

Which reinforcement ? What to change ? this message is both to the Dev and to the program ☺

- Find by yourself: program an ability to solve a problem rather than strictly solving it.

When input/output correlation is not obvious, it is obvious that:

- In the determinism algorithms, the moves are predictable, so we suppose to know how to solve the problem.
- In the stochastic algorithm, the moves are unpredictable, so we suppose to unknow how to solve the problem.

Theoretical knowledge about problem solvers

How to choose the algorithm and the fitness to solve a problem when the problem is NP-complete (exponential execution time in the input data size) ?

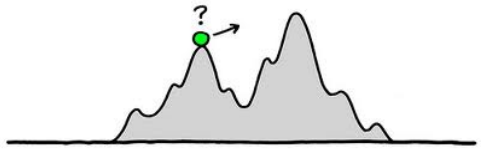
NoFreeLunch principle [Wolpert et Macready1995]. Let:

- F *the set of fitness functions,* $f \in F$, $f : X \rightarrow Y$, *finite sets*
- d^x *a finite series of size* m *of points from* X *and* d^y *the evaluation of them in* Y
- D *the domain definition for* $d=(d^x, d^y)$
- a *one algorithm,* $a : D \rightarrow X$ *s.t.* $a(d)$ *is a solution (i.e. configuration) visited by* a *during the run*
- $p(d^y | f,m,a)$ *the measurement of the performance of* a, *running* m *time on the function* f

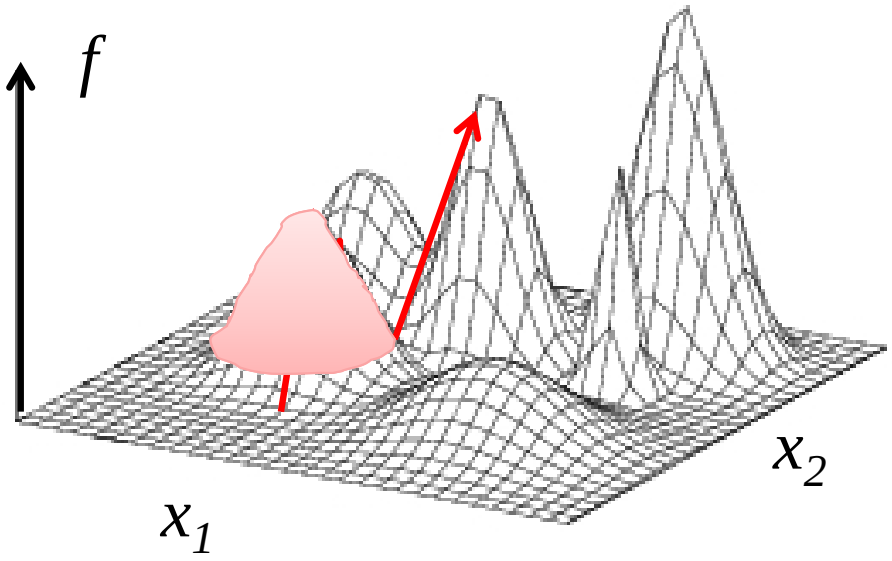
For any couple of algorithms a *and* a', *on* F *they are equivalent in performance:*

$$\sum_{f \in F} p(d_m^y \mid f, m, a) = \sum_{f \in F} p(d_m^y \mid f, m, a')$$

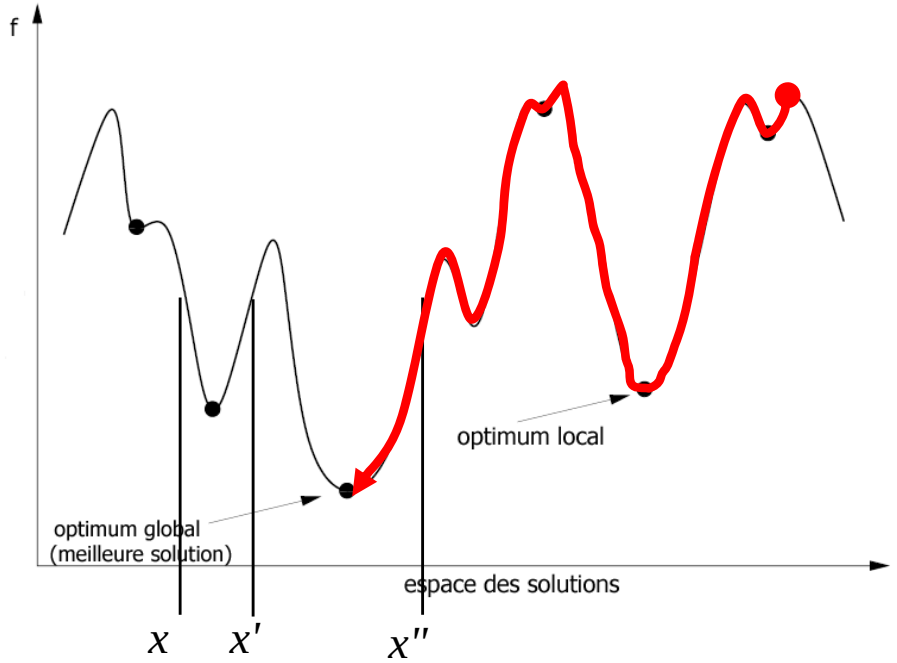
Meta heuristic – Adapt the sampling



Maximize with 2 variables



Minimize with *n* variables

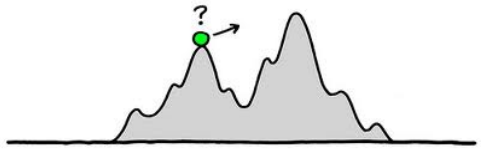


Tabu Search: restrict the neighborhood.
1/ Search the best neighbor up to the local optimum.
2/ Force the algorithm to search other neighbors by memorizing and excluding the previous paths.
3/ Explore from new paths.
Move from one attractive area to another one progressively.

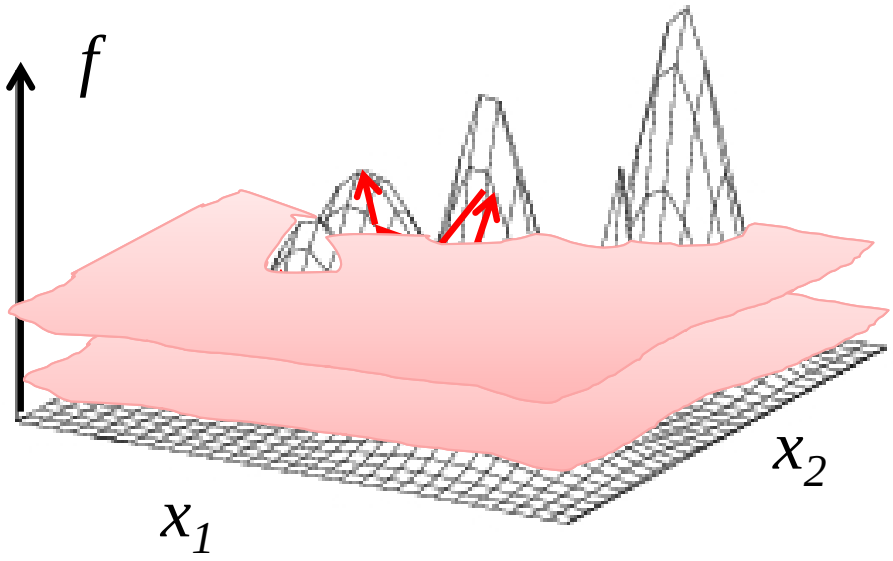
$$x = (< x_1, v_1 >, < x_2, v_2 >, \ldots, < x_n, v_n >)$$

The solution moves from one local optima to another one.

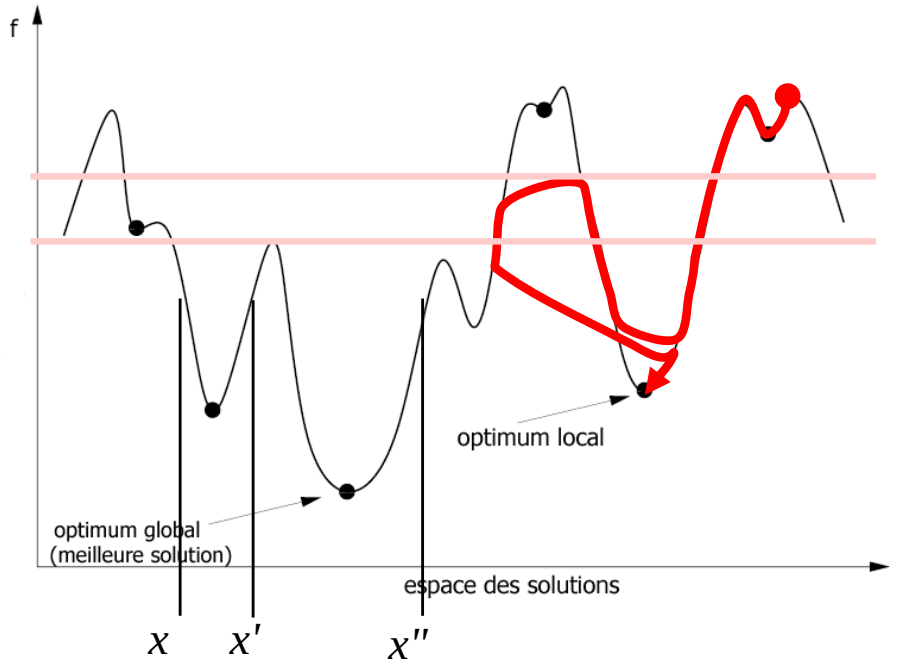
Meta heuristic – Adapt the sampling



Maximize with 2 variables



Minimize with *n* variables

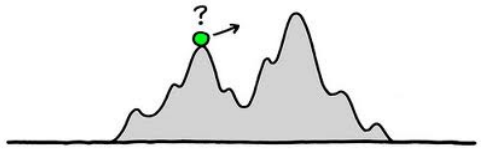


Simulated Annealing: restrict the neighborhood. 1/ Search randomly in all directions but give more chance to neighbors with best fitness than worst, for a given time.
2/ After a while, restrict the chance to use worst neighbors, and repeat from 1/, up to local optima.
Move from worst fitness neighbors to best fitness neighbors progressively.

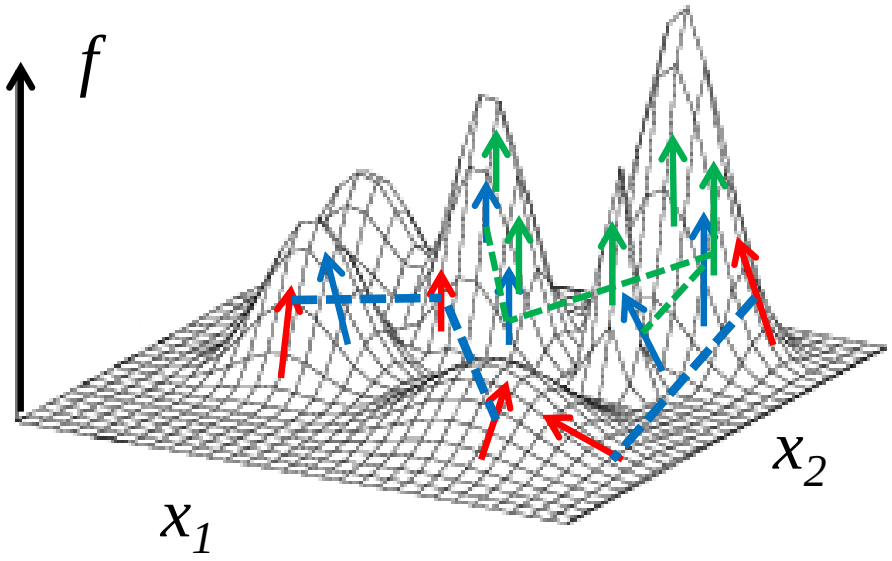
$$x = (< x_1, v_1 >, < x_2, v_2 >, \ldots, < x_n, v_n >)$$

The solution moves from the worst to the best and ends in its best local optima.

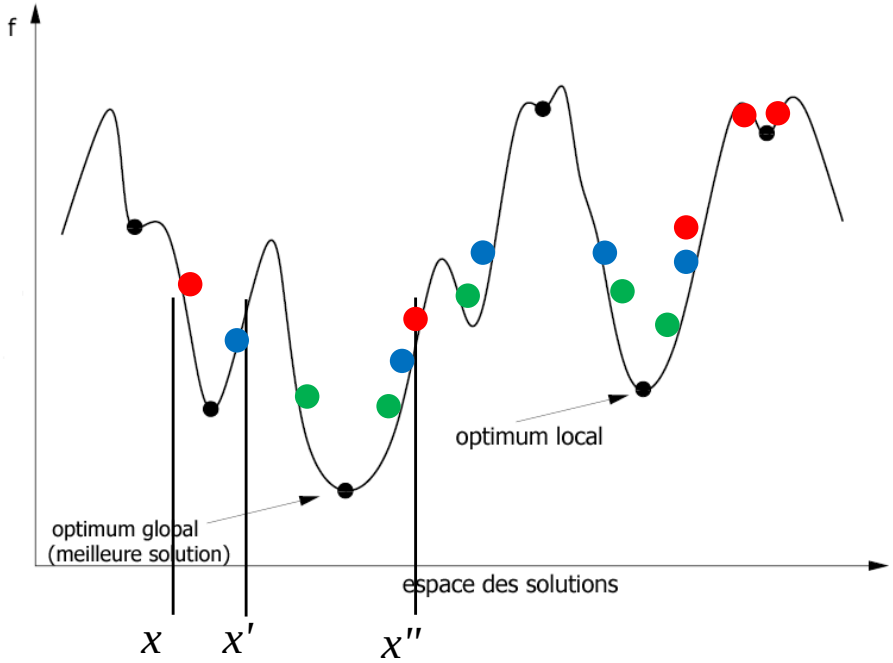
Meta heuristic – Adapt the sampling



Maximize with 2 variables



Minimize with *n* variables



Evolutionary Algorithm: extend the neighborhood.
1/ Search randomly in all directions with several solutions for a given time.
2/ After a while, build new solutions from the current ones, and repeat from 1/, up to time's up.
Explore several attractive areas simultaneously and exploit the best ones.

$$x = (< x_1, v_1 >, < x_2, v_2 >, \ldots, < x_n, v_n >)$$

The solutions move separately and sometime exchange their assignment to converge to several local optima.

Content

1. Heuristics and meta solvers
2. **Tabu Algorithms**
3. Evolutionary Algorithms
4. Multicriteria decision

Tabu Search origin

General problem solver proposed by Fred GLOVER in 1986 (Uty Colorado)

- Glover et alter, *Future paths for integer programming and links to artificial intelligence*, Computer and OR, 13: 533-549, 1986
- Glover and Laguna, *Modern heuristics for CO problems*, Mac Graw Hill Publisher, London, 1995

Tabu search: forbidden to reuse recently visited solutions or movements (from one solution to another) recently used

Introduce a memory in the search strategy with the aim to diversify the search in the solution space

- Diversify means looking for other solutions or movements than those already used

Use a deterministic heuristic as opposed to most mechanisms implemented in meta solvers rather stochastic

- Because determinism (or usefull knowledge) if faster than random for problem solving
- Examine a large sample of solutions, eventually all, in the neighborhood of the current solution

Tabu Search origin

F. Glover worked on very big (lot of combinations) and difficult problems (lot of local optima)

- A deterministic algorithm hangs in local optima
- A random algorithm does not find good solutions

Third way to solve those problems could be: 1/ find a local optimum, 2/ go away from it, and 3/ dot it again ; he called it the “Tabu Search method (TS)”

- The concept is to use a deterministic procedure to get out of the local optimum and manage the loop which causes the return after some movements
- The deterministic procedure manages a memory (called Tabu List) which prohibits the reuse of neighborhoods already visited: the algorithm is forced to use for a given time a sub-neighborhood that is not yet explored
- He defined two ways to do it: strict Tabu List or non-strict Tabu List

TS strict process: the process stores the solutions

1.Define a Tabu List that memorizes the solutions already visited (this list is called a "strict list").

2.Explore the neighborhood solutions, non tabu, of the current solution.

3.Choose the best of these neighbors: it is not necessarily better than the current solution => TS is never blocked when it finds a local optimum.

4.Update the Tabu List: the previous solution becomes tabu

Algorithm 1: Tabu Search with prohibited solutions

1.Initialisation

s_0 starting solution

$s \leftarrow s_0,\ s* \leftarrow s_0, c* \leftarrow f(s_0)$

$TL = \emptyset$ $*$ *The Tabu List*

2.Generate a subset in the neighborhood of s

$s' \in N(s) / \forall s" \in N(s), f(s') \leq f(s")$ and $s' \notin TL$

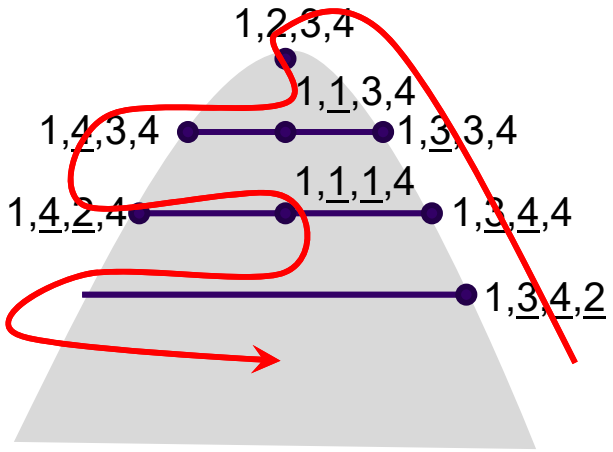
$If\ f(s') < c*\ then\ s* \leftarrow s', c* \leftarrow f(s')$

$Update\ the\ tabu\ list\ TL: TL \leftarrow TL \cup \{s\}$

$Set\ the\ tabu\ duration\ of\ s: TD[s] \leftarrow iteration + offset$

$s \leftarrow s'$

3.Loop in 2 if stopping condition not reached



Strict-list settings

Neighborhood definition: what can we change inside a solution

- It is dedicated for each specific problem as in local search general solver

Tabu Duration (TD) : the list of stored tabu solutions is finite of length TD to fix => the algorithm prevents any cycle of length smaller than TD

- Get out of the Tabu List: FIFO
- Long term => long list => good memory (few risk of loop) but time consuming
- Short term => short list => fast to run but bad memory (high risk of loop) => are you far enough away from the local optimum to avoid going back ?
- Parameter settings: calculate the Hamming distance with the local optima and set a threshold for the distance e.g. “if 10% of the variables are different the algorithm is far enough”
- This threshold must be set for each problem => it needs a **learning mechanism by data analytics**

Tabu Search « intelligence »

1. Strict restriction of possible choices: all movements leading to a tabu solution are forbidden => the neighborhood changes and is more and more restricted with the progress of the search
2. Intensive search: it allows a complete exploration around a local optimum, this is good if it is in the path to the global optima

KPI on Tabu Search performance

If a local optimum is high or isolated, the finite size Tabu List will not prevent the algorithm from looping around this local optimum and returning to it.

This loop is identifiable by measuring the frequency of visits of the variables of the problem. When detected, it is necessary to stop the process and change it: 1/ enlarge the Tabu Duration or 2/ change the neighborhood or 3/ generate a new starting solution.

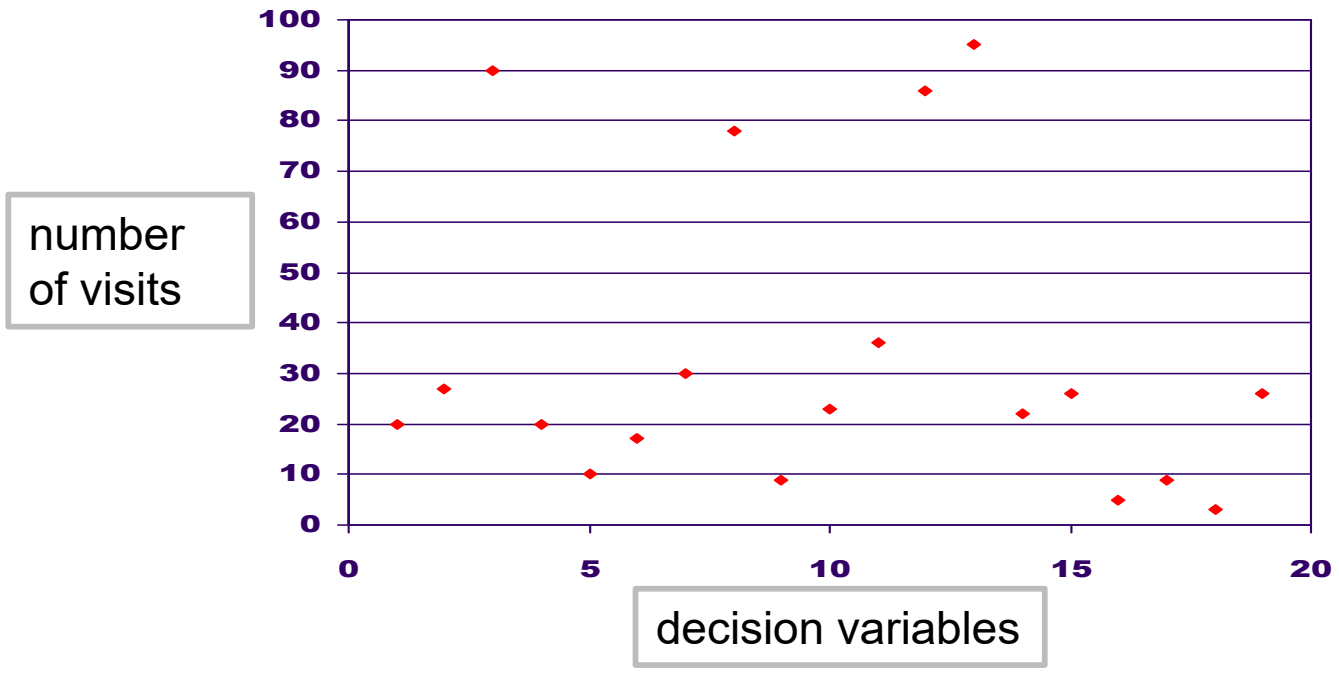
Let, n_j, the number of visit of the variable j

KPI examples:

$$\frac{n_j}{iteration\_number}$$

$$\frac{n_j}{\max\{n_j, \forall j\}}$$

$$\frac{n_j}{\bar{n}}$$



TS non-strict process: the process stores the moves

A move from one state to another is defined by all the variables who change their value from the state s to $s' \in N(s)$; *usually only one variable at a time*.

1. Define a Tabu List that memorizes the moves already applied (this list is called a "non-strict list").
2. Explore the neighboring solutions, with a non Tabu Move from the current solution: either the variable choice is prohibited, or the variable and value choices are prohibited.
3. Choose the best of these neighbors: it is not necessarily better than the current solution => TS is never blocked when it finds a local optimum.
4. Update the Tabu List: the move to go from s to s' becomes tabu i.e. the updated variable or the updated couple (variable, old value).

Algorithm2*: *Tabu Search with prohibited moves

2. Generate a subset in the neighborhood of s

$s' \in N(s)/\forall s" \in N(s), f(s') \leq f(s")$ and $move(s,s') \notin TL$

$If\ f(s') < c*\ then\ s* \leftarrow s', c* \leftarrow f(s')$

$Update\ the\ tabu\ list\ TL: TL \leftarrow TL \cup \{move(s,s')\}$

$Set\ the\ tabu\ duration\ of\ the\ move(s,s'): TD[move(s,s')] \leftarrow iteration + offset$

$s \leftarrow s'$

Non-strict list settings

Algorithm2*: *Tabu Search with prohibited moves

2.Generate a subset in the neighborhood of s

$s' \in N(s)/\forall s" \in N(s), f(s') \leq f(s")$ and $move(s,s') \notin TL$

$If\ f(s') < c*\ then\ s* \leftarrow s', c* \leftarrow f(s')$

$Update\ the\ tabu\ list\ TL: TL \leftarrow TL \cup \{move(s,s')\}$

$Set\ the\ tabu\ duration\ of\ the\ move(s,s'): TD[move(s,s')] \leftarrow iteration + offset$

$s \leftarrow s'$

The Tabu Moves cannot be applied whatever the solutions *s* and *s' i.e. all solutions requiring one Tabu Move become forbidden.*

NB: the non-strict list directly impact the frequency of visits of the variables i.e. the number of variable visits will be reduced if the Tabu Move only memorizes the variable.

NB: the non-strict list avoid to return on a previous visited solution as any subset of that solution becomes prohibited (e.g. all previous solutions with $x_3 = 8$).

Tabu Search « intelligence »

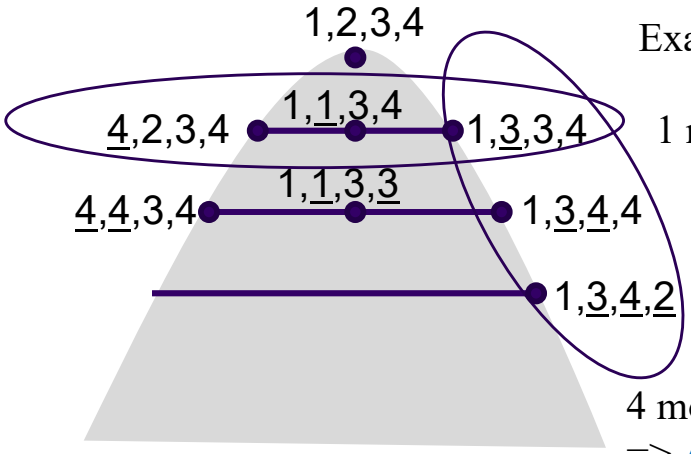
1. Loop avoidance: the algorithm forces to diversify the variable or the variable/value choice (long term memory effect) and then avoid the loops
2. Explorative search: because the local minimum is surrounded by close solutions (subset of variables with the same values), it forces to go away from it ; this is good if it is not in the path to the global optima

Non-strict list settings

Distance between two solutions *s* and *s'* : the Hamming distance is a distance in the mathematical sense which quantifies the difference between two sequences of symbols. To two series of symbols of the same length, it associates the number of positions where the two suites differ.

$$s = (x_i), s' = (x'_i), i \in [\![1, n]\!], \text{Hamming}(s, s') = \sum_{i=1}^{n} ! \, |x_i == x'_i|$$

When a variable is tabu (e.g. x_2), all solutions with a different value for x_2 are tabu => the algorithm moves faster away from the local optimum than if the couple (variable, value) is tabu (e.g. if x_2=2 is tabu, then all solutions with x_2=3 are not tabu).



Example with a 4 variables problem:

1 modified variable, we stay close to the local optimum

2 modified variables, we roll away from the local optimum

3 modified variables, we move away from the local optimum

4 modified variables, NO MORE MOVE are possible

=> **Adaptive Tabu Duration is necessary in the non-strict list too**

Non-strict list settings

Tabu Duration for a move: the Tabu Duration prohibits to reuse the move for an offset of iterations. This offset must be set according to the number of variables and possible values for the variables in order not to block the algorithm.

$$Set\ the\ tabu\ duration\ of\ the\ move(s,s') : tabu[move(s,s')] \leftarrow iteration + offset$$

Let n, the number of variables of the problem

Let TD, the Tabu Duration; it could be defined in different way:

$TD(n)$, a constant function of n s.t. $TD(n) = \sqrt{n}$; it does not give good results.

$TD(n)$, a dynamic function of n s.t.

$a)\ 0.5\sqrt{n} \le TD(n) \le 1.5\sqrt{n}$

$b)\ TD(n_i) = \alpha n_i / n$ with n_i the number of visits of the variable i

Effect: the more the variable is used, the longer the TD

c) $TD(current\ iteration) = \alpha LOG(total\ iteration + 1 - current\ iteration)/n$

Effect: the higher the current iteration, the smaller the TD

d) $TD(delta\ fitness) = \alpha / n * delta\ \ fitness$

Effect: the higher the improvement, the smaller the TD

Non strict list settings: testbed

TSP: 50 and 100 cities

Maximal iteration number: respectively 2,000 et 100,000

Four tabu duration protocols:

- TDC constant tabu duration
- TDR random tabu duration in a given interval
- TDF frequency based tabu duration
- TDI iteration based tabu duration

Run parameters:
1/ the algorithm is always greedy
2/ the effect of potential moves is computed incrementally (différenciation of the fitness)

Non strict list settings: testbed

Small problem: 50 cities 10 runs

Best: idem
In mean: 1.TDF 2.TDR 3.TDI 4.TDC

	TDC	TDR	TDF	TDI
	5878	5763	5764	5757
	5824	5645	5649	5649
	5644	5645	5649	5644
	5856	5649	5644	5793
	5645	5771	5651	5814
	5771	5654	5651	5851
	5756	5649	5644	5715
	5988	5649	5818	5768
	5773	5811	5644	5644
	5788	5777	5722	5750
Best found	**5644**	**5645**	**5644**	**5644**
In mean	**5792**	**5701**	**5684**	**5739**

Large problem: 100 cities 2 runs

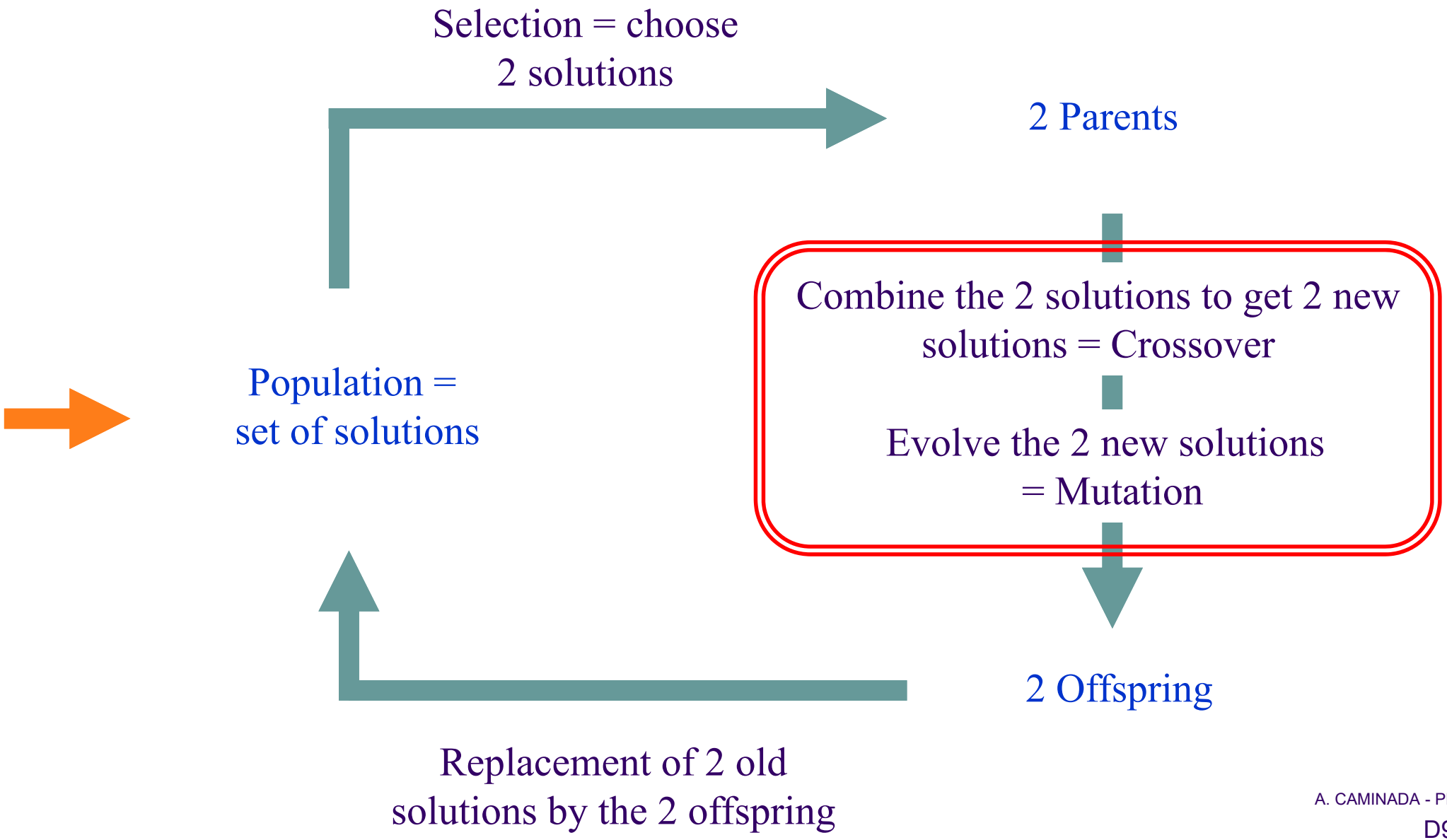
Best: 1.TDI, 2.TDF, 3.TDR, 4.TDC
In mean: 1.TDF, 2.TDR, 3.TDI, 4.TDC

	TDC	TDR	TDF	TDI
	14491	13625	13583	13702
	14242	13584	13562	13546
Best found	**14242**	**13584**	**13562**	**13546**
In mean	**14367**	**13605**	**13573**	**13624**

Content

1. Heuristics and meta solvers
2. Tabu Algorithms
3. **Evolutionary Algorithms**
4. Multicriteria decision

The basic cycle of EA: combining 2 neighborhood functions, crossover and mutation



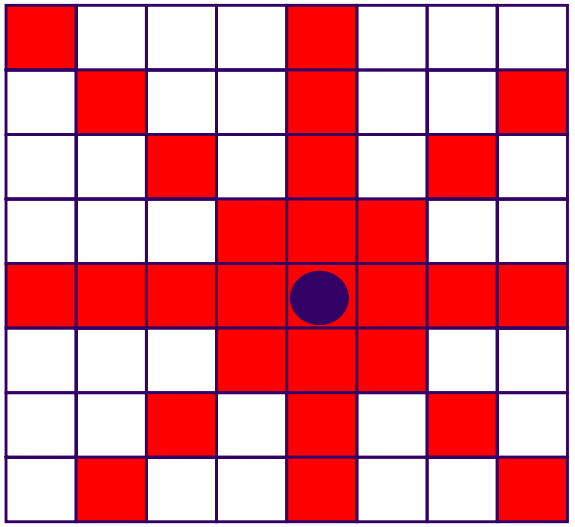
The basic algorithm of EA: replace all the solutions of the population iteratively

Procedure Evolutionary Algorithm

Fix the *Population* size // number of solutions to manage
Initialize the population *P* // randomly
Evaluation of solutions of P
Repeat
    Selection of {s1, s2} in P
    Crossover of s1 and s2 => s3 and s4
    Mutation of s3 and s4 => s5 and s6
    Put s5 and s6 in P'
    When size(P')=size(P) then P=P'
Until Stop criteria

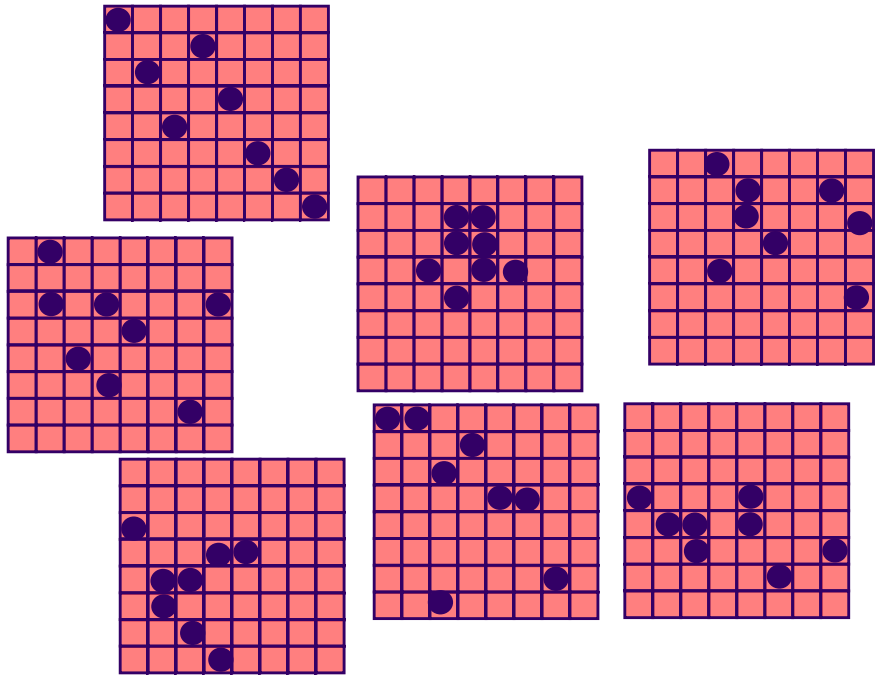
Return The best solution from *P*

See EA on an example
8 Queens problem



Problem: place 8 queens on a 8x8 chessboard such that the queens cannot check each other

Population: a set of solutions (chessboards with queens)



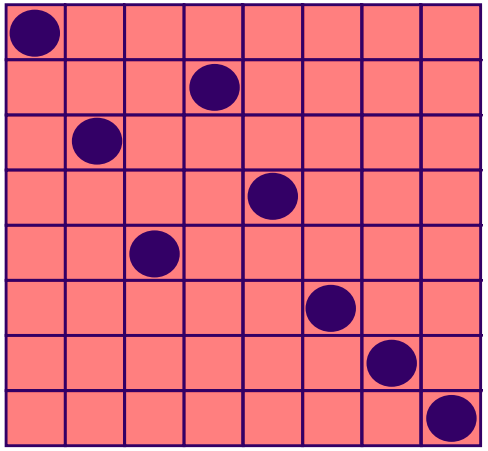
Example: 8 Queens representation

One solution: a permutation of the indexes 1 through 8

1	3	5	2	4	6	7	8

Column 1 ..... Column 8

One solution: a configuration



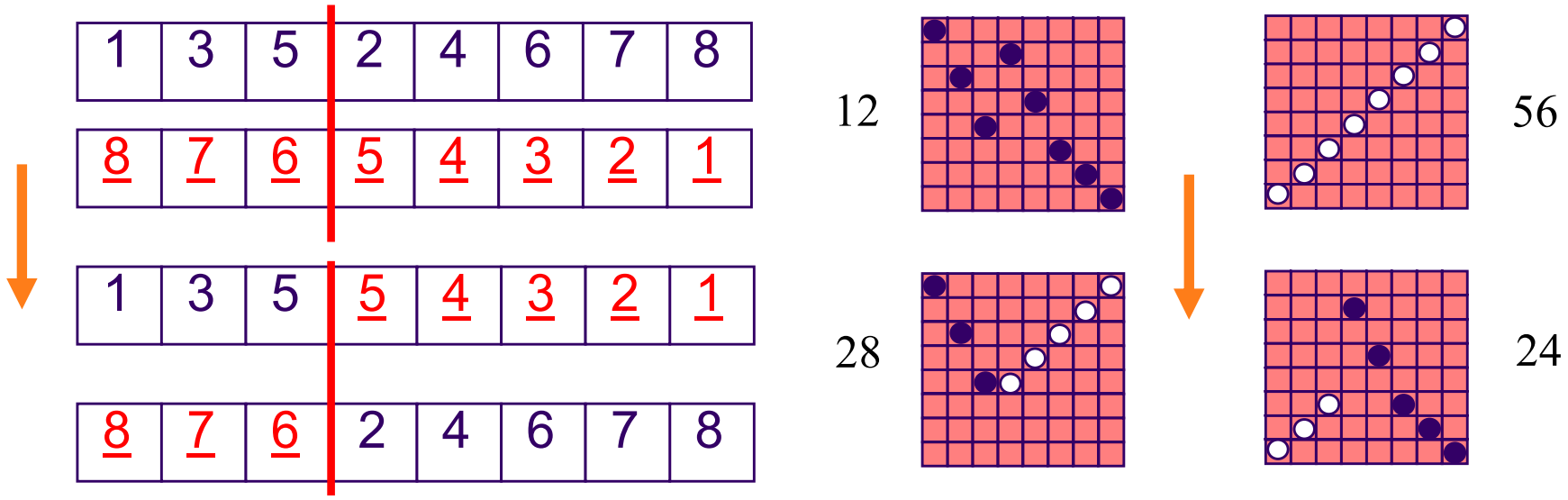
If the solution allows several queens per row, the combination is $\mathbf{A^8_{64} = 64!/56! > 178.10^{12}}$

If the solution allows one queen per row, the combination is $\mathbf{8^8 > 16}$ **million**

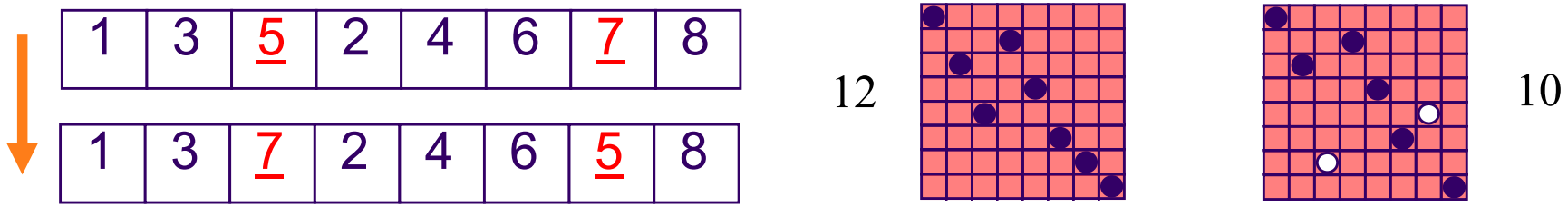
The size of the search space depends of the solution coding and large spaces are more difficult to tackle.

Example: 8 Queens genetic operators

Crossover: combining two solutions (example with **one point crossover**)



Mutation: exchanging two variables inside one solution



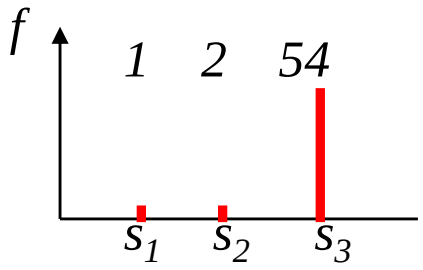
Example: 8 Queens fitness and selection

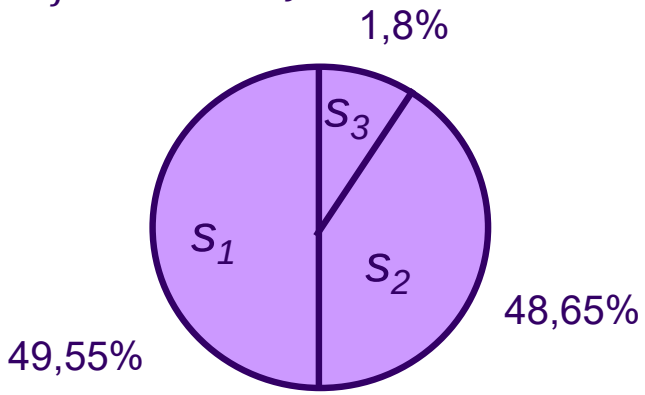
Fitness

- The penalty of one queen is equal to the number of queens it can check
- The fitness of the configuration is equal to the sum of the penalties of all queens
- Optimum when fitness is null

Selection

- Using a roulette wheel based on the fitness evaluation of the configurations
- Worst fitness: $f_{max} = n*(n-1) = 56$
- Selection of s_i with a probability: $p_i = \frac{f_{max} - f_i}{\sum_{j=1}^{n}(f_{max} - f_j)}$

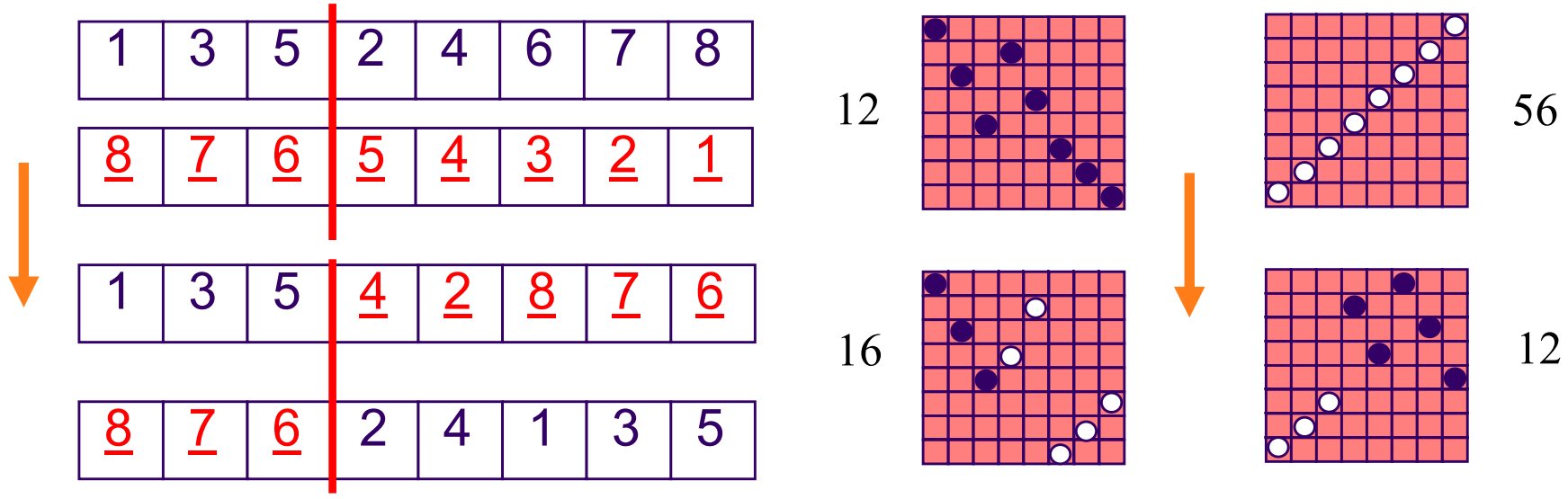




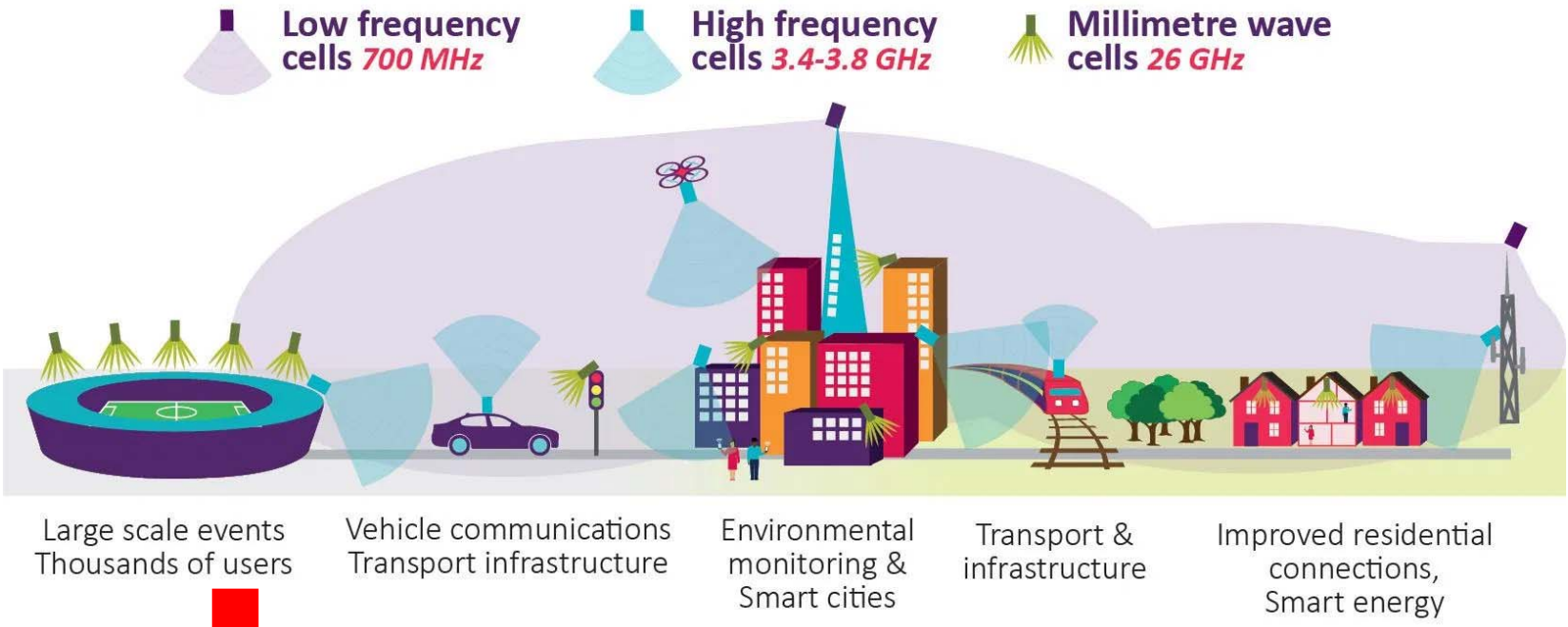
Example: 8 Queens specific crossover

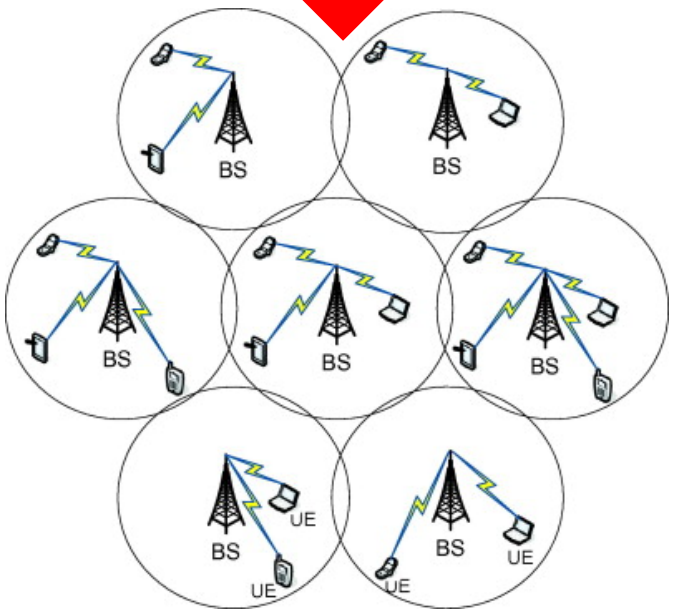
The one-point basic crossover may put 2 queens on the same row, then try to conceive a specific crossover for the problem more attractive.

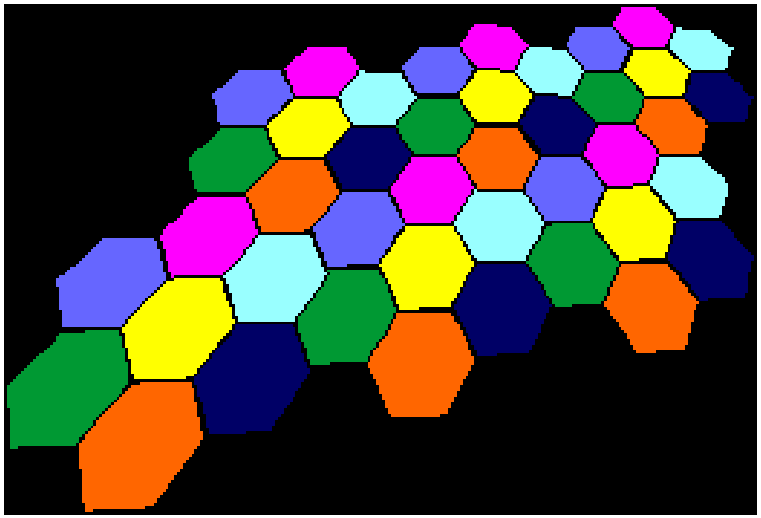
The right part is complete from right to left with the valid position on rows coming from the other solution.



Frequency assignment for cell phone with EA







Frequency assignment for cell phone with EA

Nihat Karaoglu
Computational Modeling Lab
Vrije Universiteit Brussel

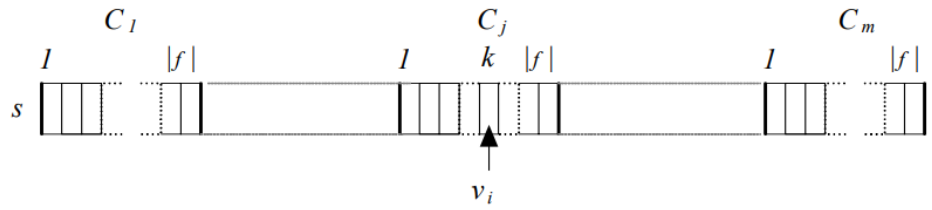


Figure 6 Representation *R. A chromosome contains $m=\lceil n/f \rceil$ genes C_i and each gene represents a cluster with maximum *f* stations as discovered by the cluster discovery algorithm. Each locus *k* in a gene contains the station, which is assigned to frequency *k*.**

```
FAPSTER(I, M)
    t←0
    S←DISCOVER-CLUSTERS( I, M)
    ENCODE(S)
    INIT-POPULATION (Pt )
    while not STOP()
          C ← SELECT-PARENTS (Pt)
       Z←RECOMBINE (C)
          MUTATE (Z)
          t = t + 1 ;
          UPDATE-POPULATION (Pt, Z)
    Z←BEST( Pt)
    return DECODE(Z);
```

Figure 10 FAPSTER (FAP*) Algorithm. Starts with discovering the clusters in the interference graph then encodes this to a primordial chromosome using random assignments.

The recombination consists in exchanging one cluster of stations between two solutions. The effect is that the surrounding stations of the exchanged cluster have a lot of interference with this cluster.

The mutation consists in reparing the interference generated by the recombination of the cluster by changing some frequencies in the bordering stations.

The evolution mechanisms to explore the search space and find the best solution to a problem

Genetic operators: the recombination increase or decrease the diversity of search

- If parents are mixed in equal proportion the new solutions are different from them
- Else the new solutions are close to the parents (the EA moves around)

Selection operators: the selection increase or decrease the diversity of search

- Selection is applied for crossover and for adding solutions in the new population
- If the selection is solution quality based (choosing the best is preferred), the diversity decreases (no exploration of search space)
- Else the selected solutions will be pseudo-random and the diversity increases (more exploration of search space)

To be efficient a population must find the appropriate trade-off on diversity

- Too much diversity: none convergence
- Not enough diversity: local optima

The right strategy depends on the search space configuration i.e. a lot of (few) local optima need to increase (reduce) the diversity

EA testbed – Polynomial optimization

Problem specification:

$$f(x) = \sum_{i=0}^{n} a_i x^i, \quad a_i \in [0,1] \text{ are known}$$

The optimum is

$f(x_{\min}) = 0,0104863$ for $x_{\min} = 0,466661$

The problem is to find $x_{\min}$ and $x \in [0,1]$

- Solution coding of *x* on 16 digits

$$x_{\min} = 0111011101110111$$
$$x_{\min} = 0,466666$$
$$f(x_{\min}) = 0,010486$$

- Population: 50 individuals
- Stopping criteria: 1000 generations
- Initial generation: random
- Selection for reproduction:

$$p_i = q_i \Big/ \sum_i q_i \quad \text{and} \quad q_i(x) = f_{\max} - f(x)$$

- Reproduction
  - Bipoint random crossover
  - Mutation: change the value of one digit with 1% probability
- Renew of the population: step by step

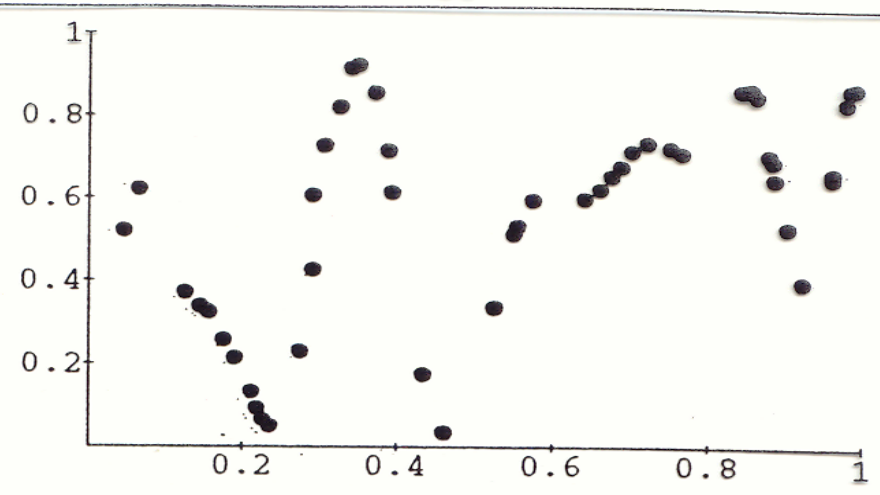
EA testbed – Polynomial optimization

Generation 0

(initial population)

chromosome	solution x
0011011111111000	0.218646
0001110001000101	0.110444
0101100100010110	0.348012
0011011111100110	0.218371
1010101000111011	0.664988
0011011011001000	0.214007
0110010010011111	0.393072
0110111011110111	0.433478
0000101110011111	0.045410
0101000111111010	0.320241
1110111111100100	0.937102
0100001110011010	0.264087
etc ...	etc ...

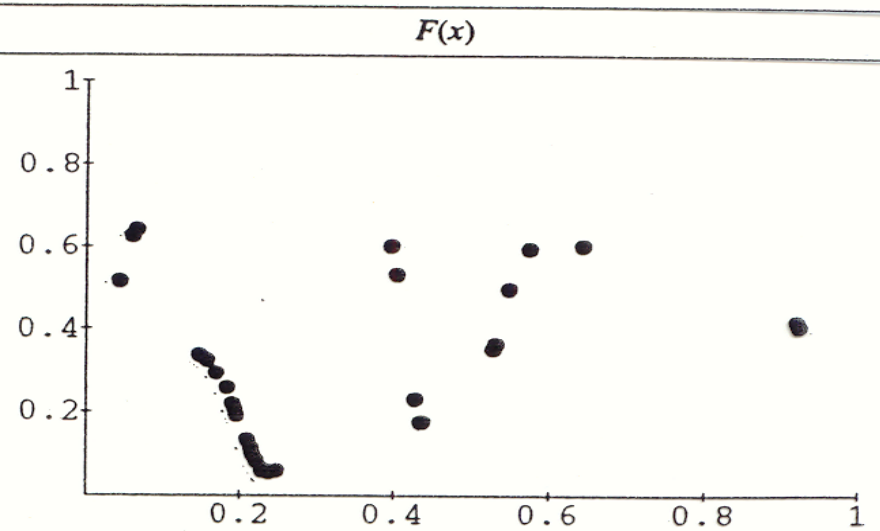
$\bar{F} = 0{,}5242768,\ F_{min} = 0{,}028133$



Generation 5

chromosome	solution x
0011100010000111	0.220828
1000110010010111	0.549202
0110111011110111	0.433478
1010011011100100	0.651941
0010110101110111	0.177615
0110110101010010	0.427054
1000011101111000	0.529198
0010011101111111	0.154299
1000011011110001	0.527138
0010111011110111	0.183474
0011001001001000	0.196429
0011001000001000	0.195452
etc...	etc...

$\bar{F} = 0{,}032507,\ F_{min} = 0{,}013141$

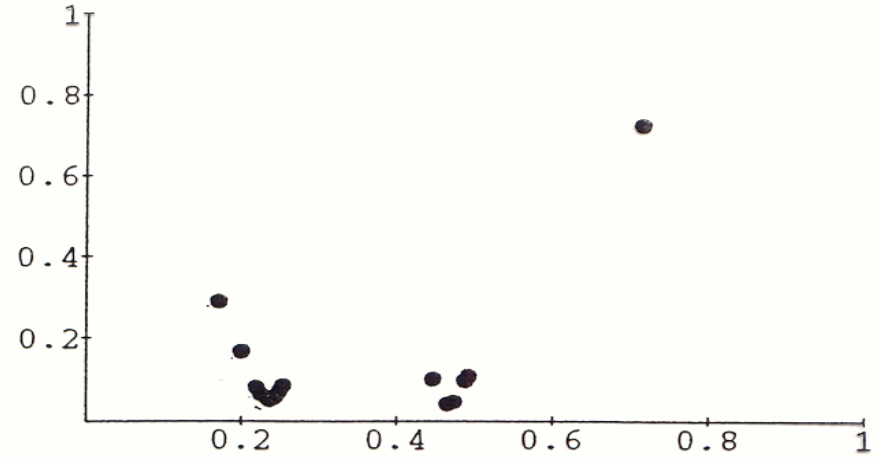


EA testbed – Polynomial optimization

Generation 30

chromosome	solution x
1011100110011100	0.725062
0011100110111101	0.225558
0111110110011110	0.490714
0011110000011101	0.234836
0011110111011101	0.241672
0111000110011100	0.443808
0111100110111101	0.475562
0111100110011101	0.475074
0011101110001101	0.232639
0011100100001101	0.222873
0011100111011101	0.226047
0011100010011101	0.221164
etc...	etc...

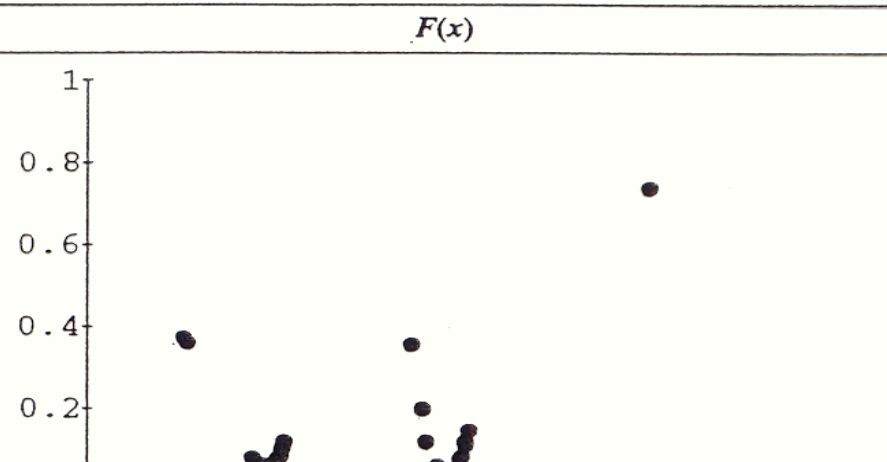
$\overline{F} = 0{,}013140,\ F_{min} = 0{,}010523$



Generation 100

chromosome	solution x
0011100111100000	0.226092
0111100000000000	0.468772
0111110001000000	0.485374
0011100000100000	0.219256
0111110000101001	0.485023
0011110001000000	0.235370
0011101001001011	0.227725
0011111101110000	0.247821
0111111101110000	0.497825
0111101001001000	0.477683
0011110001001000	0.235492
0011100010010011	0.221011
etc...	etc...

$\overline{F} = 0{,}010832,\ F_{min} = 0{,}010523$

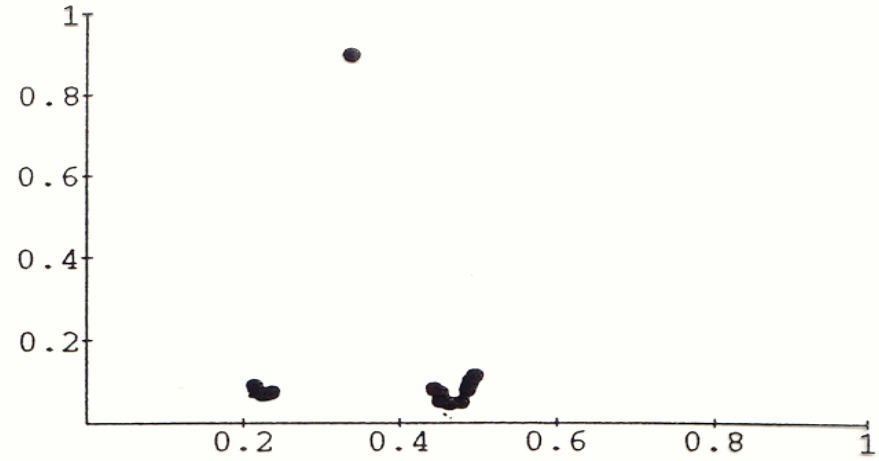


EA testbed – Polynomial optimization

Generation 300

chromosome	solution x
0111110110100101	0.490821
0111010110100101	0.459571
0111010100110010	0.457816
0111010101100101	0.458594
0111010100110010	0.457816
0101010100000101	0.332127
0011010101110011	0.208804
0111110101101001	0.489906
0111110001001101	0.485572
0111010100010001	0.457312
0111010100110001	0.457801
0011010100010101	0.207370
etc...	etc...

$\bar{F} = 0{,}010522$, $F_{min} = 0{,}010487$



No change until the end (1000 generation) => two subsets of solutions

Dynamic equilibrium between the selection and the reproduction

- The selection wants to keep the best and so reduce the diversity
- The reproduction wants to explore new combination and so extend the diversity

The EA algorithm parameters

Procedure Evolutionary Algorithm

Fix the *Population* size // number of solutions to manage
Initialize the population *P* // randomly
Evaluation of solutions of P
Repeat
    Selection of {s1, s2} in P
    Crossover of s1 and s2 => s3 and s4
    Mutation of s3 and s4 => s5 and s6
    Put s5 and s6 in P'
    When size(P')=size(P) then P=P'
Until Stop criteria

Return The best solution from *P*

Selection: roulette wheel (proportional chance) or tournament (the best wins)

Crossover: 1-point or 2-points with specific arrangement rules for the problem

Mutation: local search, hill climbing or Tabu search. To define: apply several mutation steps before to go ahead

Crossover and Mutation: define a probability threshold of applying them

Population evolution: the |P| offspring's replace the |P| parents OR the offspring's and the parents are put together and we select the |P| solution by a selection procedure

Introduction – Domains of application

Domains	*Application Types*
Control	gas pipeline, missile evasion, pursuit, power plant
Design	semiconductor layout, aircraft design, keyboard configuration, communication networks
Scheduling	manufacturing, facility scheduling, resource allocation
Robotics	trajectory planning
Machine Learning	designing neural networks, improving classification algorithms, classifier systems
Signal Processing	filter design
Game Playing	poker, chess
Theoretical problems	travelling salesman, routing, bin packing, graph colouring, partitioning

EA performance

Acceptable performance at acceptable costs on a wide range of problems

- EA tackles a lot of worldwide benchmarks
- EA are used on a lot of real-world applications

Intrinsic parallelism

- Well adapted to huge computation technology (grid computing) and network consideration (distributed computing)

Superior to other methods on complex problems with

- Lots of data or many free parameters
- Complex relationships between parameters
- Many local optima
- Temporal variation of data (dynamic problems)

Content

1. Heuristics and meta solvers
2. Tabu Algorithms
3. Evolutionary Algorithms
4. Multicriteria decision

Multicriteria problem on real example

The car sharing service

- Car sharing enables someone to rent a self-service car on an occasional basis, for one hour, one day, or longer.
- The service, convenient, affordable and environmentally friendly, is an alternative to owning a private car which remains unused 95% of the time.
- A vehicle is picked somewhere and left somewhere else.

The real case study

- « Twizy Way » by RENAULT: Electrical Vehicle (EV) car sharing in St Quentin en Yvelines (230.000 people, 120km2)
- Technical constraint: the EV must always be picked and left in a charging station
- Cost expenditure: no more than 20 charging stations to place and 5 to 10 parking places by station
- Find the best locations for stations: a multi-objective problem

« Twizy Way » by RENAULT: problem analysis

To run efficiently, there must be at any time:

- At minimum one car in each station to pick a car
- At minimum one free place in each station to leave a car

It is not possible to use “jockeys” that is people in charge of car redeployment along day or night

- Strongly expensive for cars, not like bike

The subject is then for 20 stations and *n* cars:

- locating the stations
- and dimensioning the number of parking per station
- knowing that the distance between two stations must be at minimum 600 m (300 m is the maximum walk distance to get a car)
- knowing that the stations must be in dense area (the service must be used)
- knowing that input/output flow of any station must be equilibrated (neither empty or full stations)

Fitness to optimize

Fitness 1: the service must be used => maximize the flow between the stations

$$f_1 = \max_{s \in \Omega} \left[\sum_{st_i \in s} \sum_{st_j \in s \setminus \{st_i\}} flow(st_i, st_j) \right]$$

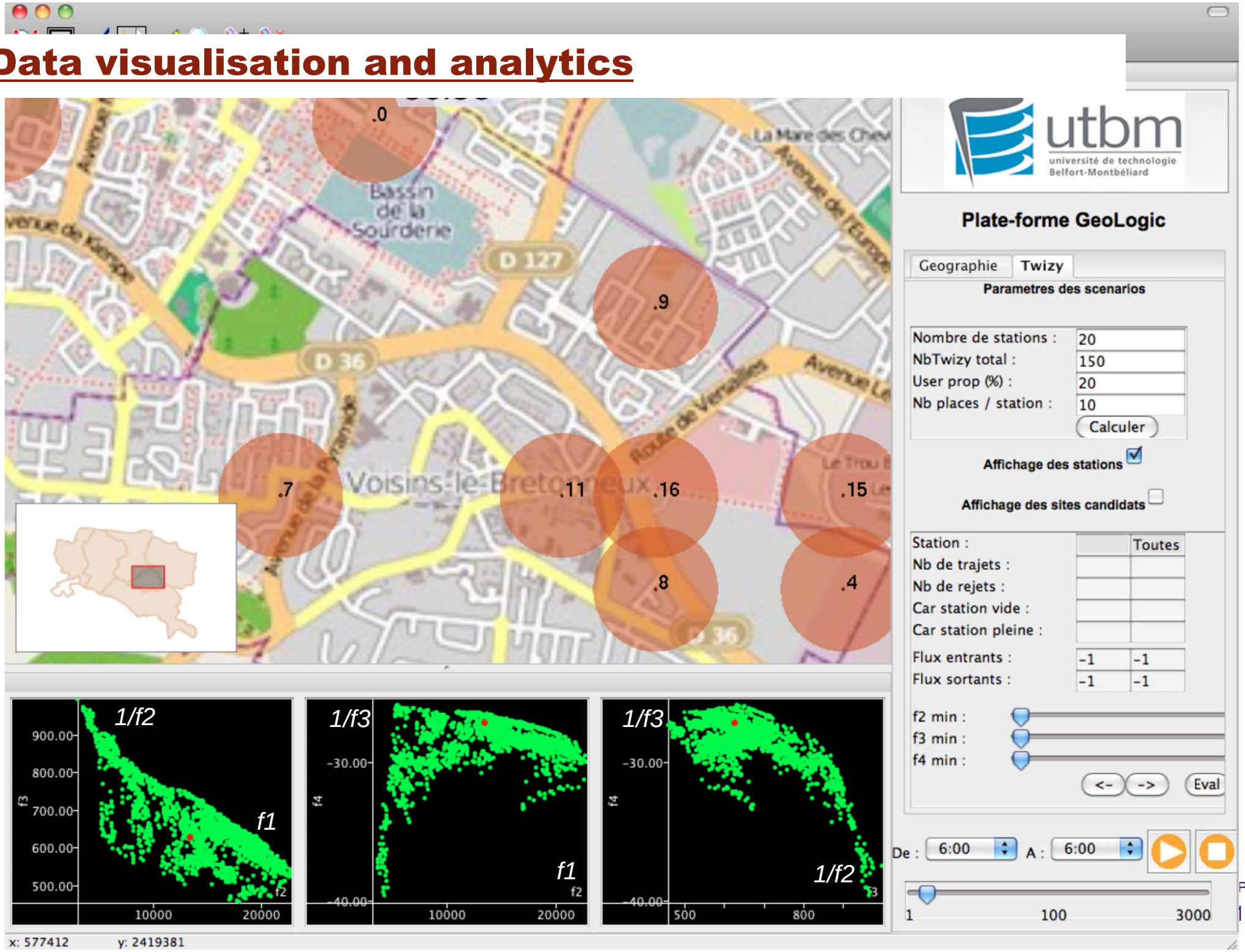
Fitness 2: neither empty or full station => minimize the (output – input) flow per station

$$f_2 = \min_{s \in \Omega} \left[\sum_{st_i \in s} \sum_{t} output(st_i, t) - input(st_i, t) \right]$$

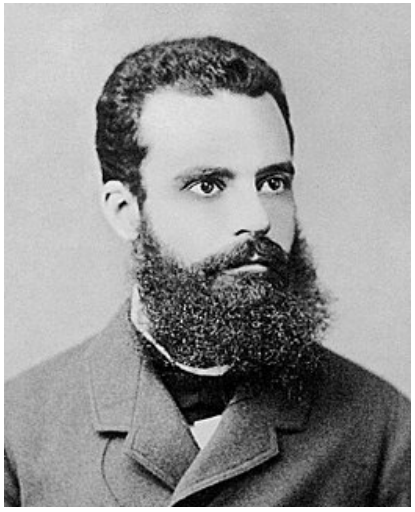
Fitness 3: use of cars all hours not only morning and evening => minimize the difference of hourly flow

$$f_3 = \min_{s \in \Omega} \left[\sum_{st_i \in s} \sum_{t} flow(st_i, t) - \overline{flow}(st_i) \right]$$

Data visualisation and analytics



Multicriteria decision

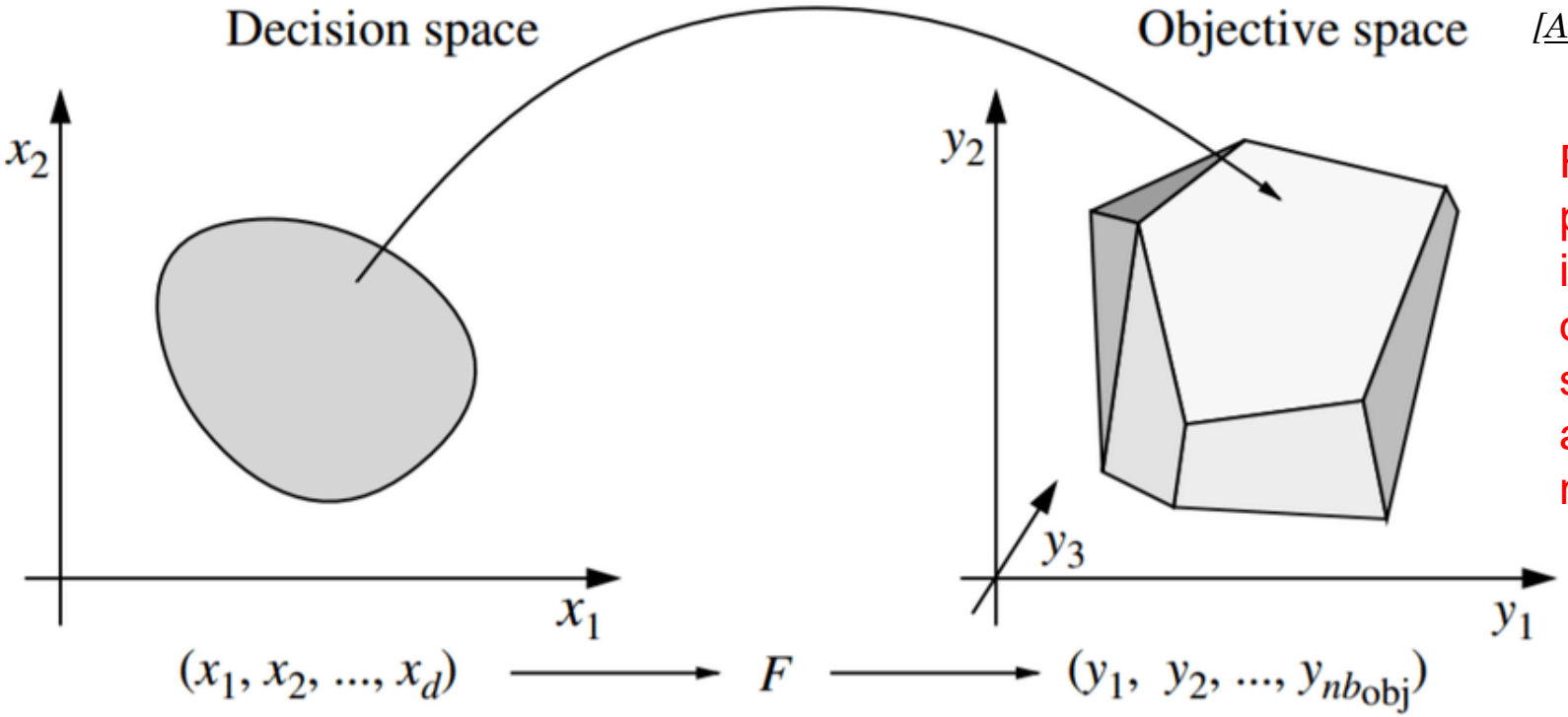


Origin in 1896 by Vilfredo PARETO, economist

$$\min F(x)$$

$$F(x) = (f_1(x), f_2(x), ..., f_{nbobj}(x)) \text{ with } nbobj \geq 2$$

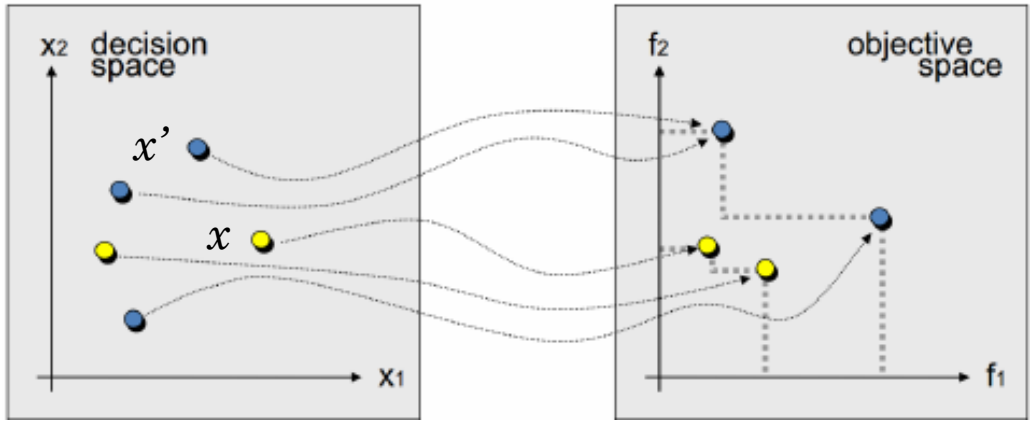
$$\text{such that } x \in \Omega \text{ and } x = (x_1, x_2, ..., x_d)$$



[Anagnostopoulos, 2017]

For these problems, there is no garantee of solution *x* such that all y_i are minimum

Pareto dominance between solutions



[Liebana, 2014]

The Pareto dominance defines a partial order relation between objects

$$\min F(x)$$

$$x \text{ dominates } x' \Leftrightarrow \forall\, i \in [1..n]\ \ f_i(x) \leq f_i(x')$$
$$\text{and } \exists\, \mathrm{k} \in [1..n] \text{ such as } f_k(x) < f_k(x')$$

Pareto optimality

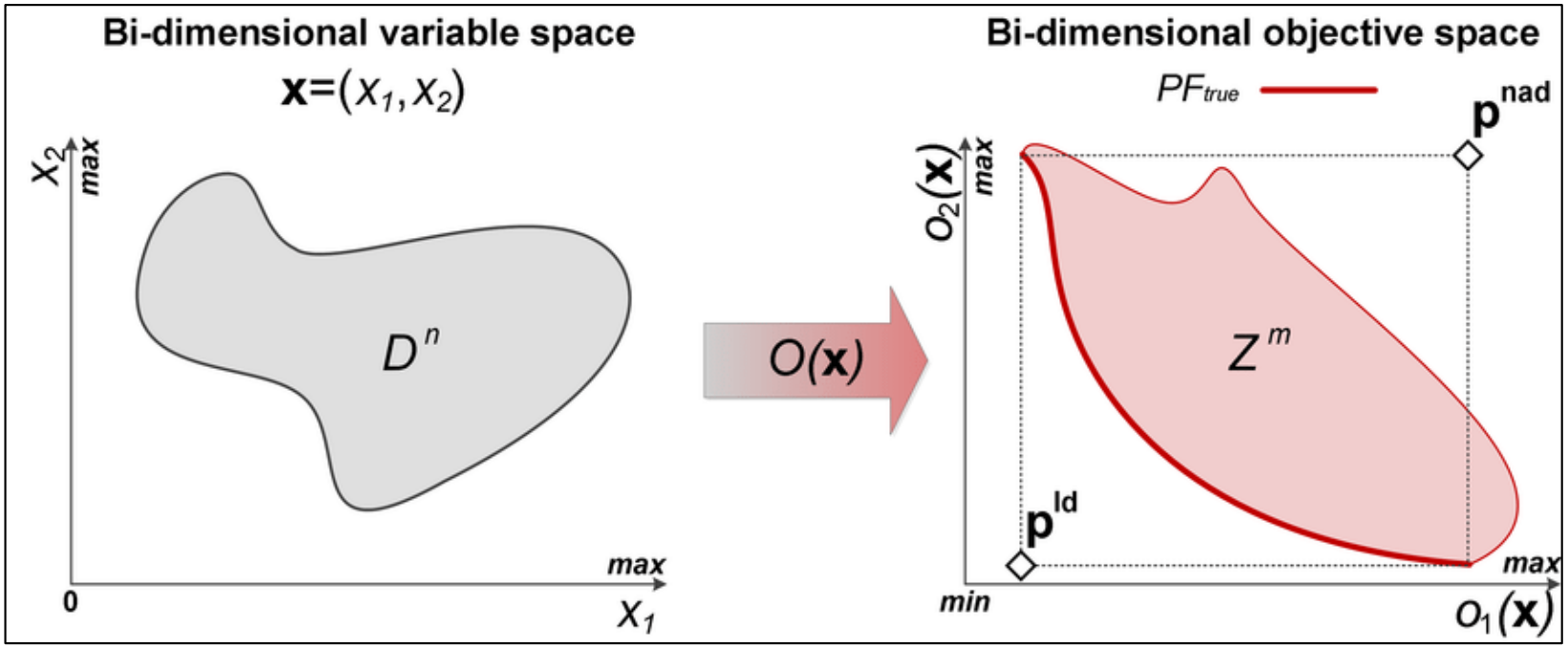
Definition: $x^* \in \Omega$ is Pareto optimal (or Pareto non dominated)

$\Leftrightarrow \neg\exists\, x \in \Omega$ such as x dominates x^*

The Pareto Front (PF) is the set of non dominated solutions.

The *ideal* (*nadir*) point p^{id} (p^{nad}) is: $p^{id} / p^{id}_i = \min(f_i(x))$ and $x \in PF$

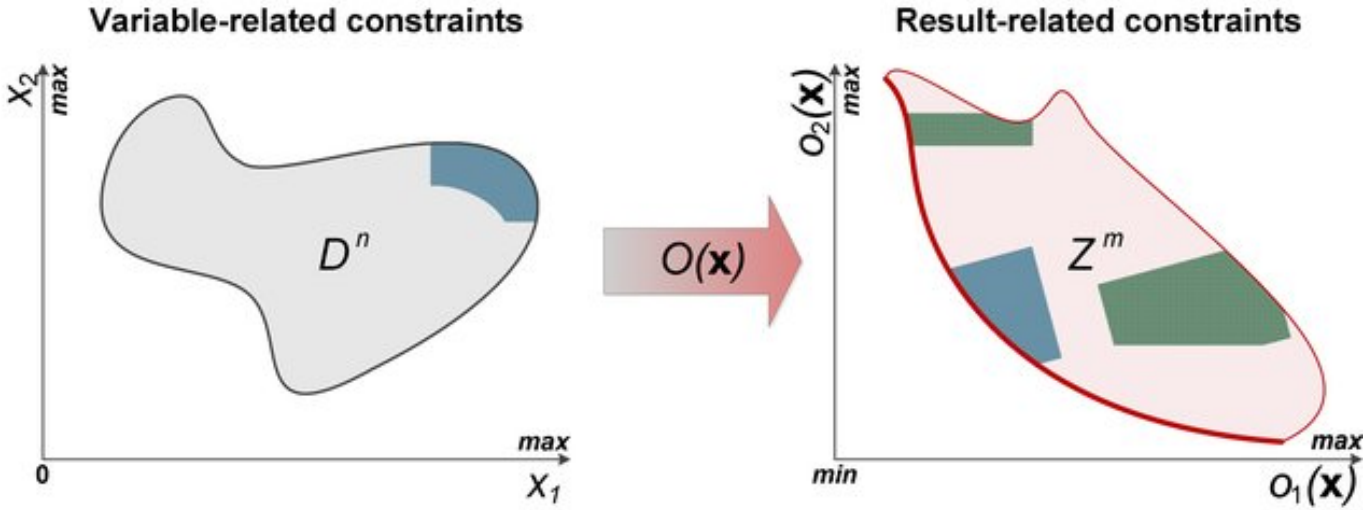
$p^{nad} / p^{nad}_i = \max(f_i(x))$ and $x \in PF$



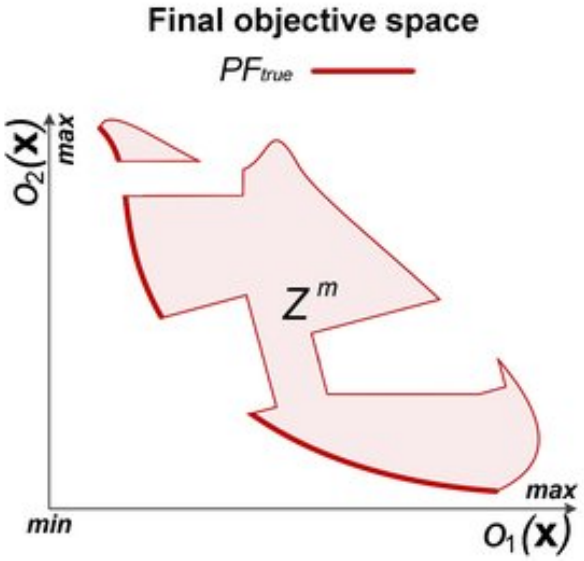
[Zavoianu, 2015]

Pareto optimality under constraints

The hardest problems have non continuous Pareto Front due to constraints on variable assignements



[Zavoianu, 2015]



How to manage these problems

Different options

1. Convert the problem in a single objective one => only one solution is found by algorithm run
   - Agregation method
   - ε-constraint method
   - Goal method

2. Keep the multicriteria without Pareto concept
   - Deal separetely with the objectives in lexicographic order

3. Keep the multicriteria with Pareto concept
   - Deal globally with the objectives with the dominance management

Agregation method

The fitness is a linear agregation of the objectives

How to fix the weights?

It needs several trials

All solutions are in the hyperplan

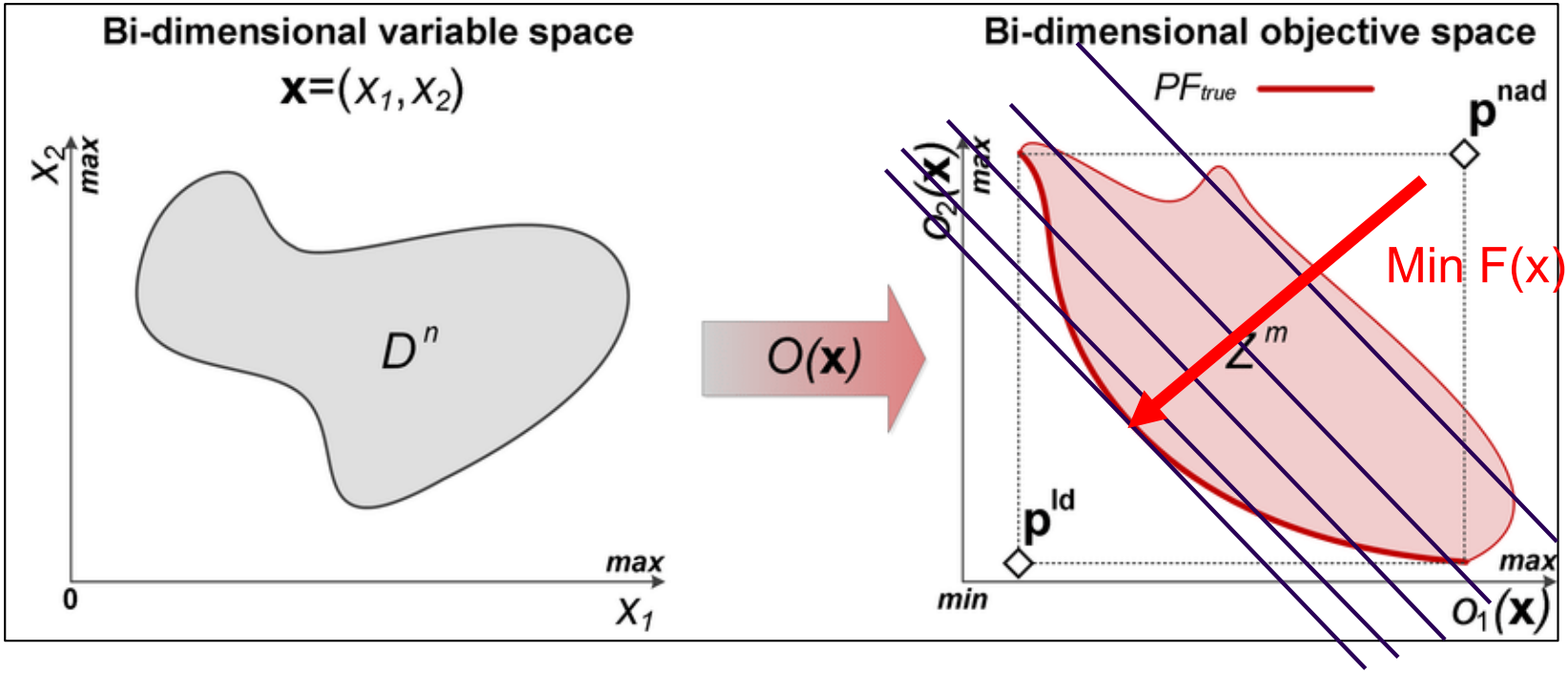
$$\min F(x)$$

$$F(x) = \sum_{i=1}^{nbobj} \lambda_i f_i(x) \text{ with } nbobj \geq 2$$

such that

$$x \in \Omega \text{ and } x = (x_1, x_2, ..., x_d)$$

$$\lambda_i \geq 0 \text{ and } \sum_{i=1}^{nbobj} \lambda_i = 1$$



Ɛ-constraint method

One objective is used as fitness

All other are converted as constraint to respect

How to fix the Ɛ?

It needs several trials

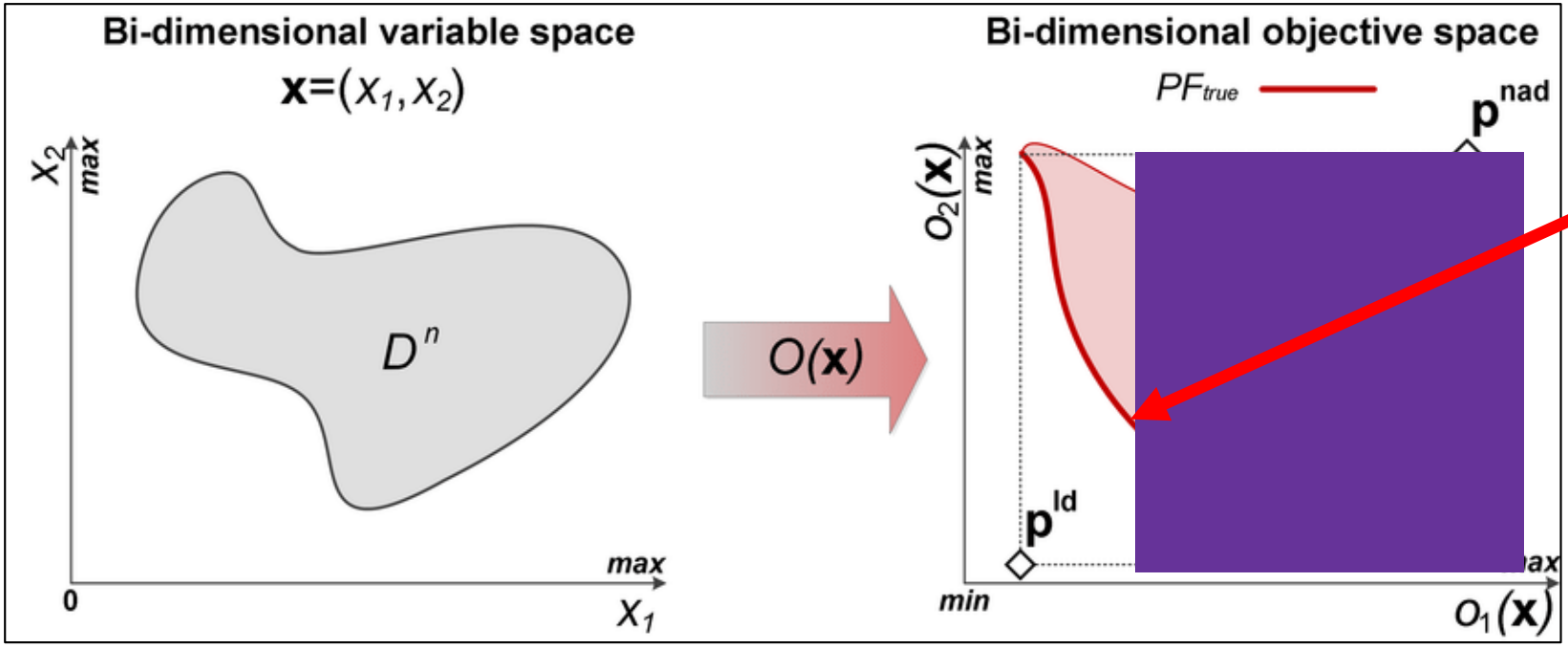
All solutions are in the reduced area

$$\min f_k(x)$$

such that

$$x \in \Omega \text{ and } x = (x_1, x_2, ..., x_d)$$

$$f_i(x) \leq \varepsilon_i \text{ where } i \in [1..nbobj] \setminus k$$



$O_1(x) \leq \varepsilon_1$

Goal method

For all objectives, a reference to reach is set

How to fix the goal value in each dimension?

It needs several trials

No restriction on the solution space

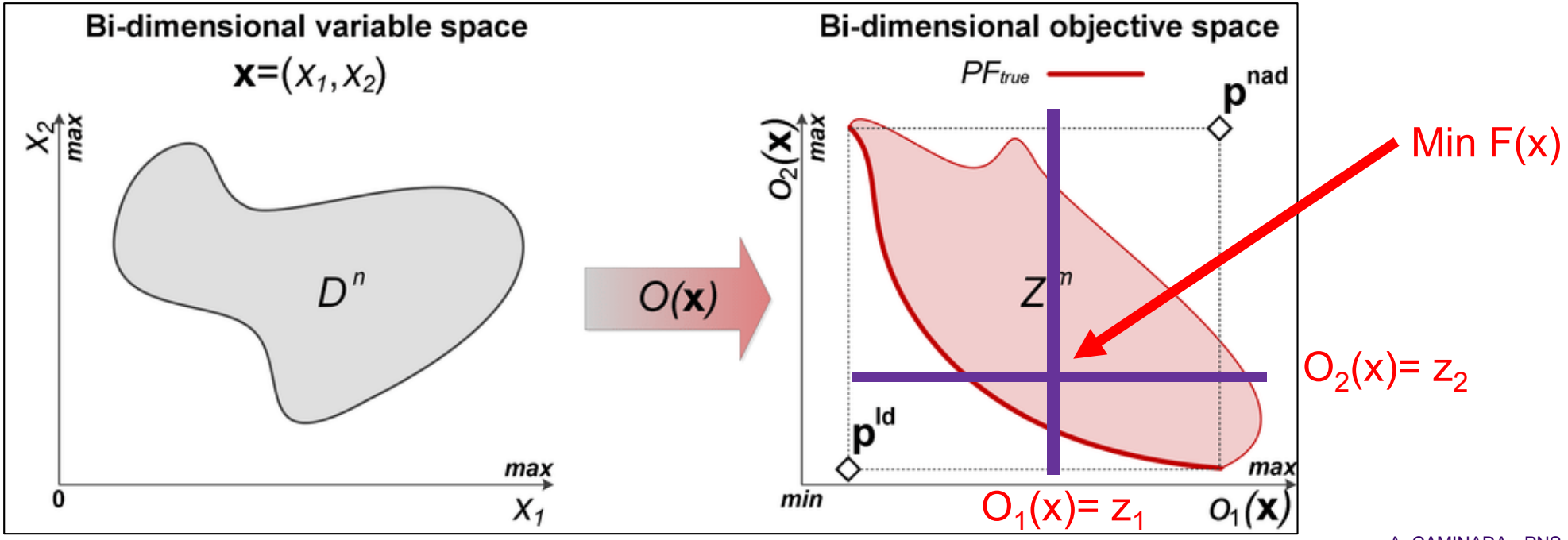
$$\min F(x)$$

$$F(x) = \left(\sum_{i=1}^{nbobj} | f_i(x) - z_i |\right) \text{ with } nbobj \geq 2$$

such that

$$x \in \Omega \text{ and } x = (x_1, x_2, ..., x_d)$$

z is the goal value



Lexicographic method

A preference order is given for all objectives

How to fix the order?

It needs several trials

The last objective is in a reduced area

$x \in \Omega$ and $x = (x_1, x_2, ..., x_d)$

$step1.\min\ f_1(x)$

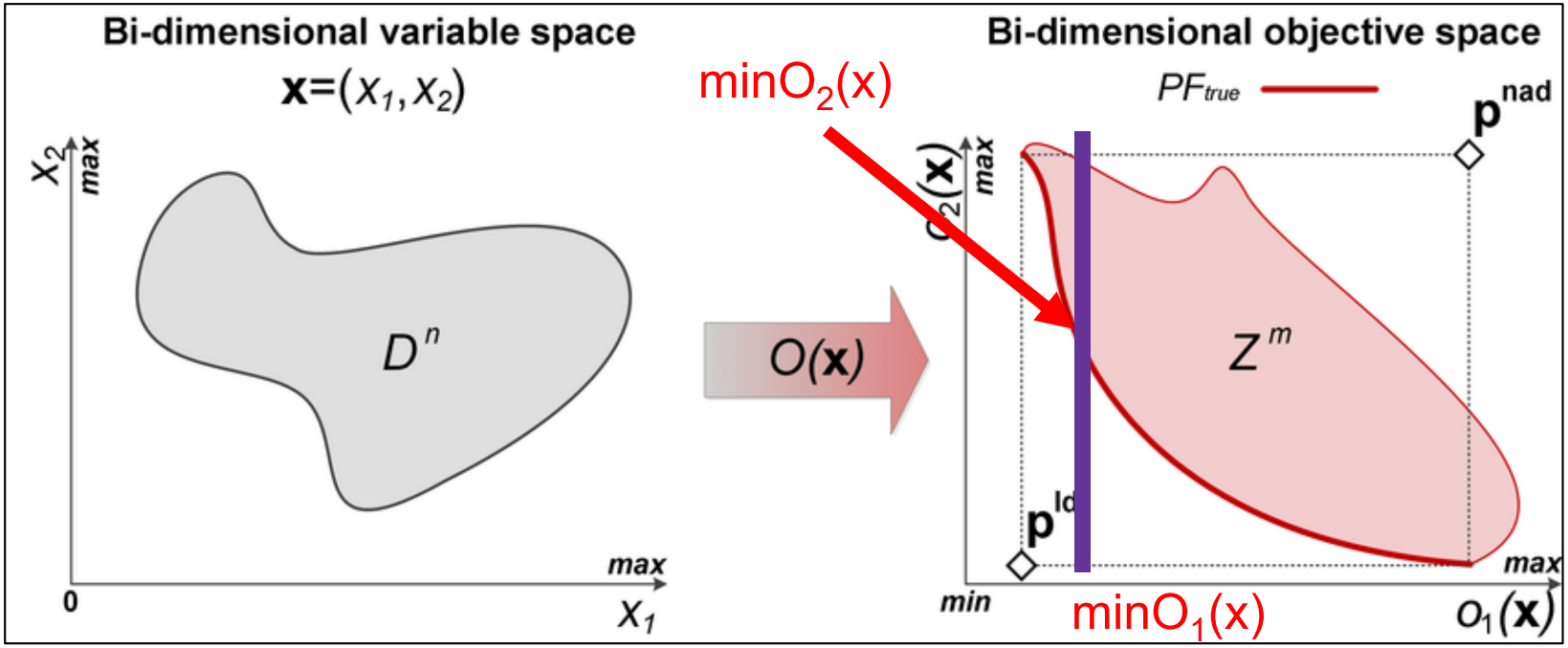
$step2.\min\ f_2(x)$

such that $f_1(x) = \min\ f_1(x)$

$step3.\min\ f_3(x)$

such that $f_1(x) = \min\ f_1(x),\ f_2(x) = \min\ f_2(x)$

etc.



Dominance method

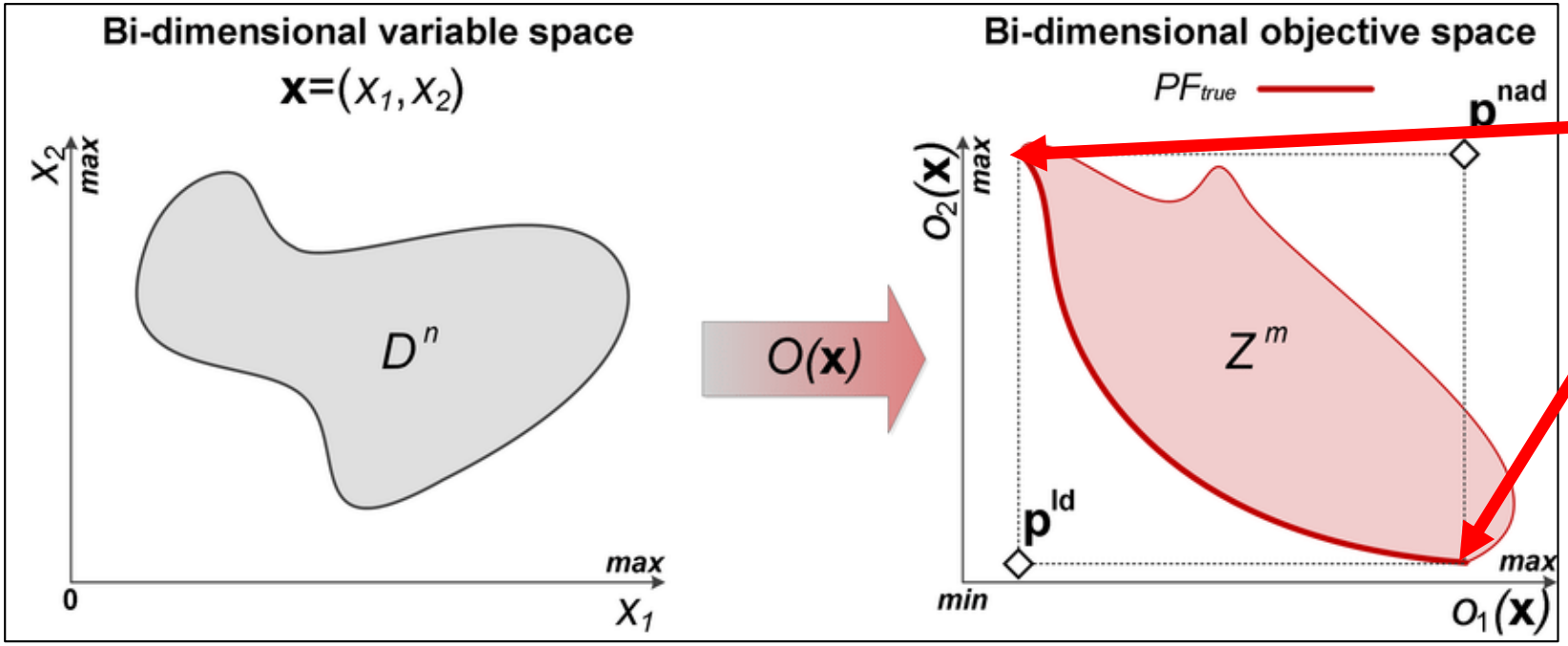
$$\min F(x)$$
$$F(x) = (f_1(x), f_2(x), \ldots, f_{nbobj}(x))$$
$$\text{with } nbobj \geq 2$$
$$\text{such that } x \in \Omega \text{ and } x = (x_1, x_2, \ldots, x_d)$$

The original problem is kept !

The optimum is a set of solutions

The Pareto Front of non dominated solutions is optimum

The algorithm must always rank the solutions to keep the best ones and some other promising solutions using a ranking procedure



PF=optimum

Ranking and filtering of solutions found during search

The algorithm cannot keep all visited solutions to progress, then the current solutions are filtered to keep the most promising one and continue their improvement

1. Ranking based on domination

The rank of a solution is the number of solutions dominating it + 1

NDS (Not Dominated Sorting)

2. Ranking based on domination

Non dominated solutions are ranked 1 (set S1) then removed

Non dominated solutions are ranked 2 (set S2) then removed

Etc.

NSGA (Nondominated Sorting Genetic Algorithm)

3. Mean Ranking on all objectives

The rank of a solution is the average rank over each objective

WAR (Weighted Average Ranking)

Example: NSGA-II genetic algorithm for the « Twizy Way »

Require: population size N, generation number *nbIter* {*nbIter* can be replaced by a time limit}

1: $P \leftarrow$ init(N) {init population P with N random individuals}
2: $Q \leftarrow \emptyset$ {init an empty children population}
3: eval(P) {eval objectives for each individual}
4: **for** $i = 1$ to *nbIter* **do**
5: $P \leftarrow P \cup Q$
6: assignRank(P) {based on Pareto dominance}
7: **for** each non-dominated front $f \in P$ **do**
8: setCrawdingDist(f) {mesure the distance between solutions}
9: **end for**
10: sort(P) {by rank and in each rank by the crawding dist }
11: $P \leftarrow P[0 : N]$ {filter the solutions and keep the N best ones}
12: $Q \leftarrow$ buildChildren(P) {continue the improvement of the best ones}
13: **end for**