Metaheuristics for optimization

Bastien.Chopard@unige.ch.

Rule for the exam:

$\frac{1}{3}$ TP + $\frac{2}{3}$ Oral

↑

Jan/Feb or Aug/Sept

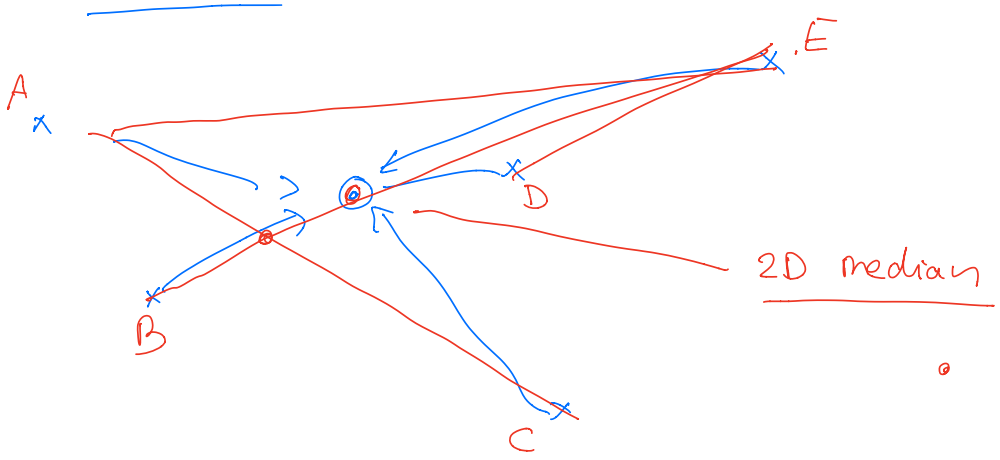
Oral exam: presence (or zoom if needed)

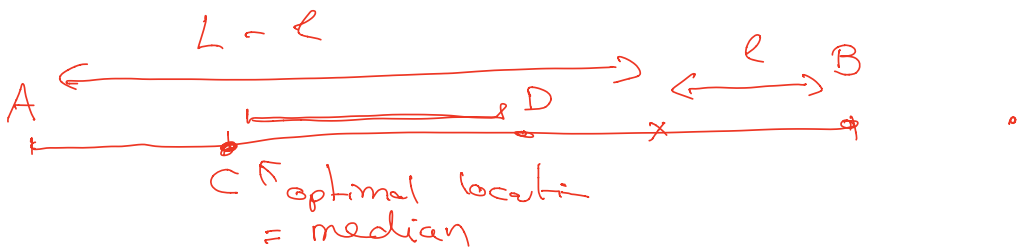
- No preparation, no document
- List of questions

Course material: notes, books

Check Moodle and email

Example:





Chapter 1 Introduction

1.1. Search Space

We want to solve an optimization problem:

Find x such that a given function $f(x)$ is

maximal or minimal

↑ maximization problem ↑ minimization problem

Definition: The **search space** S is the set of possible values for x $S = \mathbb{R}^2$ in the 2D median example.

S can be continuous, or discrete, finite, infinite

But in any case, we will consider situations where S is **very big** (no exhaustive search is tractable)

$x \in S$ is often called a **possible solution** or also a **configuration**

What is f?

f is a function which quantifies the quality of an x

$f : S \longrightarrow \mathbb{R}$ or a subset of $\mathbb{R}$ on which there is an order relation

Terminology: f is called the
- objective function
- cost function
- energy function

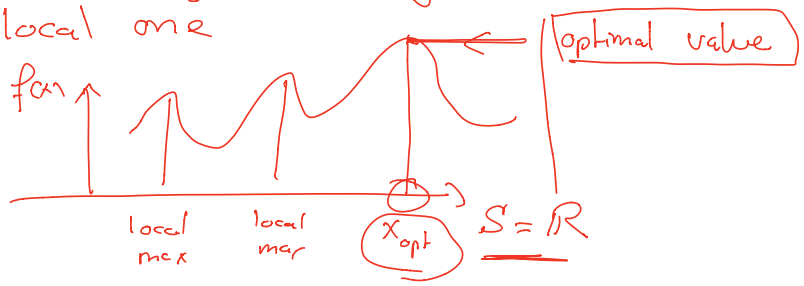
- fitness : what we will use in this course.

The optimal solution is $x_{opt} \in S$ such that

$$\begin{cases} f(x_{opt}) \geq f(y) & \forall y \in S \quad \text{(max probl)} \\ f(x_{opt}) \leq f(y) & \text{(min probl)} \end{cases}$$

$$\begin{cases} x_{opt} = \underset{y \in S}{\operatorname{argmax}} f(y) \\ x_{opt} = \underset{y \in S}{\operatorname{argmin}} f(y) \end{cases}$$

Note we are looking for a global optimum, not a local one

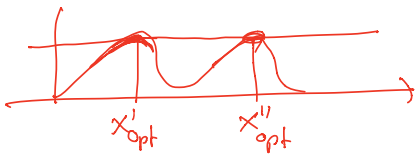


The optimal value is $f(x_{opt})$

Question: is the optimal value unique?
is the optimal solution unique?

Optimal value is unique otherwise there would be a smaller and a bigger one. Choose the best.

The optimal solution may not be unique



Usually S is multidimensional:

$$x \in S \quad x = (x_1, x_2, x_3, \dots, x_n)$$

n-component vector.

n is called the problem size or the number of degrees of freedom

It means that the cardinality of S grows exponentially with n:

$$S = S_1 \times S_2 \times \dots \times S_n \qquad x_i \in S_i$$

$$|S| = |S_1| \cdot |S_2| \dots |S_n| = |S_i|^n$$

S is usually too large to be explored systematically.

→ we need algorithms and in particular metaheuristics

1.2 Examples of optimization problems.

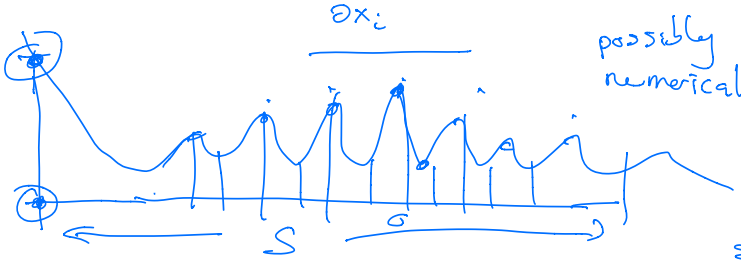
1) $S = \mathbb{R}^n \qquad f: \mathbb{R}^n \to \mathbb{R}$, f is continuous and differentiable

⇒ continuous optimization problem

Solution is obtained by solving

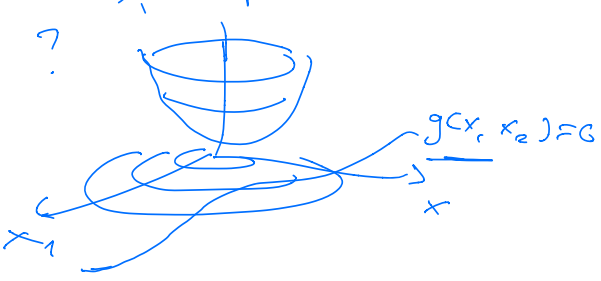
$$\frac{\partial f(x_{opt})}{\partial x_i} = 0 \qquad i = 1, \dots n$$

possibly to be solved numerically.



then we have to enumerate the solution to find the best.

The optimum solution can also be at the boundary (and the derivative is not zero).



What about constraints?

Use Lagrange multiplyers to impose the constraint:

minimize $f - \lambda g$

2) Linear Programming (standard probl in economics)

$$\max \quad z = f(x) = \sum_{i=1}^{n} c_i x_i \qquad x = (x_1\ x_2 \dots x_n)$$

$c_i \in \mathbb{R}$ given.

subject to $\sum_{i=1}^{n} a_{ji} x_i \le b_j$

for m given constraints
$j = 1$ to m
a_{ij}, b_j are given.

The solution are at the edge of a convex polygon.

Algorithm: SIMPLEX

3) Knapsack: we have n objects of size w_i el value p_i

We want to take as much value in the sack whose capacity is C

$S = \{0,1\}^n$

$$x_i = \begin{cases} 0 & \text{I do not take the object} \\ 1 & \text{I take the object} \end{cases}$$

maximize $f(x) = \sum_{i=1}^{n} p_i x_i$

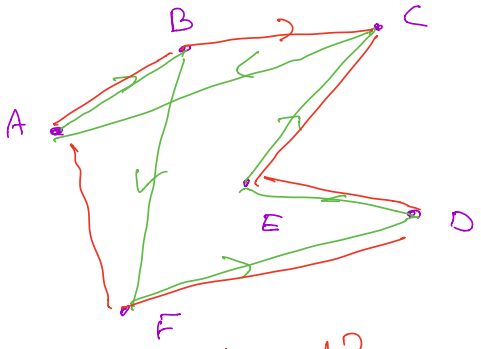
subject to $\sum x_i w_i \le C$

If n is large this becomes a difficult problem for which metaheuristics may be needed.

4) Traveler Salesman Problem TSP

We have n towns and salesman who has to tour through these n towns without going twice to the same town and return to the initial town. The tour should be the shortest possible

towns = {A, B, .. F}



— possible tour ABFDEC

— another possible tour ABCEDF

A tour is specified by a permutation of the symbols A, B, C, D E and F

S = { set of permutation of n towns }

ABCEDF ∈ S

|S| = n! factorial

|S| grows very fast with n

$n! = n(n-1)(n-2)\dots 2.1$

$4! = 24$

$5! = 120$

:

5) NK - Landscape

Inspired by a biological problem ; genetic regulatory networks. S Kaufmann formulate this problem in a mathematical way.

Consider N persons with two possible commercial shategies : $x_i = 0$, $x_i = 1$ $i = 1, \dots N$

The succes of these shategie depend of the choice of shategy of the people one is interacting with. (That is wheather people compete or collaborate)

The profil of agent i is some function f_i which depends on x_i and that of its neighbors

$$f_i(\underbrace{x_i, x_{i_1}, x_{i_2} \dots}_{K \text{ arguments}})$$

($x_{i_1}, x_{i_2}, \dots$: neighbors)

The problem is specified once the f_i are given, as well as K, N

We want to maximize

$$f = \sum_{i=1}^{N} f_i$$

the profil of the population.

Our main interest with NK-problems is to be able to generate synthetic problems of increasing difficulty. (as N and K increase, it become more and more difficult to solve)

Example MaxOne problem.

Find a chain of N bits that maximize the number of ones.

Obvious solution: $X = (x_1 x_2 \dots x_N) = \underbrace{111\dots1}_{N \text{ times}}$

$f_i(x_i, \dots) = x_i \quad K = 1$

$$f = \sum_{i=1}^{n} f_i = \sum_{i=1}^{n} x_i$$

$f = N$

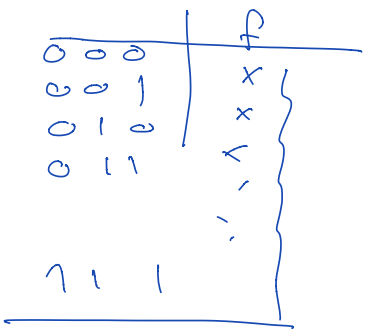
Another example

$X = x_1 x_2 \dots x_N \qquad f_i(x_{i-1}, x_i, x_{i+1}) \quad K = 3$

f_i is max for 111 then it is a trivial solved

But if f_i is large for 101 and small for 010

then the chain 101101101



1.3 Reminder on computational complexity

To solve an optimization problem, we need an efficient algorithm, in terms of execution time T_{cpu} and memory usage

↑ space complexity. ↑ time complexity

We are interested in what is called asymptotic complexity, that is T_{cpu} as a function of n, the problem size, for large enough n

More precisely we want to know how $T(n)$ grows with n.

We use the "big O" notation

If one says that

$$T(n) = O(g(n))$$

it means that $\exists n_0$ and a constant k

such that

$$T(n) \leq k g(n) \quad \text{if } n \geq n_0$$

$T(n) = 0.5 n^2 + 2n = O(n^2)$

$T(n) = n + \log n < n + n = O(n)$

For many problems (sorting, matrix multiplication, ...) there exist algorithms that are polynomial in the problem size.

$$T(n) = O(n^m)$$ where typically $m = 2, 3, \ldots$

Such problems are said to belong to the P-class

But there are other problems for which no polynomial algorithm is known. There are only exponential algorithms

$$T(n) = O(2^n)$$

Those problems are computationally difficult if n is large.

Complexity Classes

In complexity theory one has problems belonging to classes

P, NP, NP-complete, NP-hard

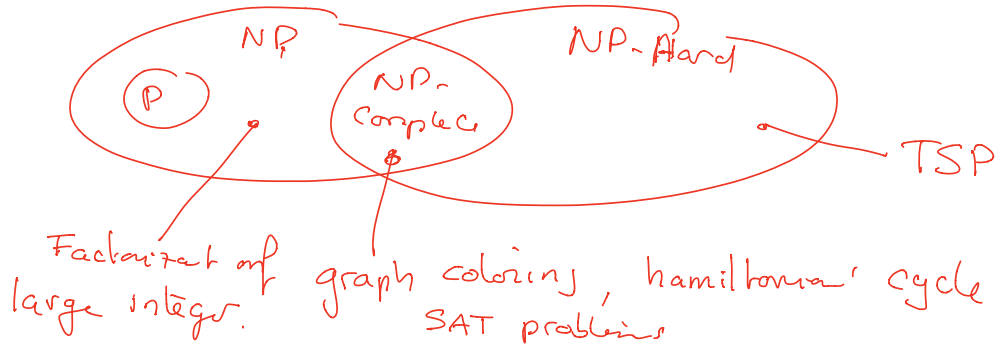
Class P: There is an algorithms that solves the problem in a polynomial time $T(n) = O(n^m)$

Class NP: Problems for which a solution can be checked in a polynomial time.

Thus $P \subseteq NP$

Class NP-hard: Those are problems whose solution can be used to solve any NP problem up to a polynomial additional time.

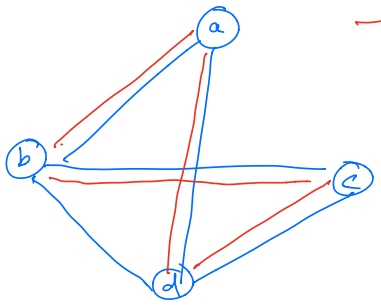
Class NP-Complete Are problems that are both in NP and in NP-hard.



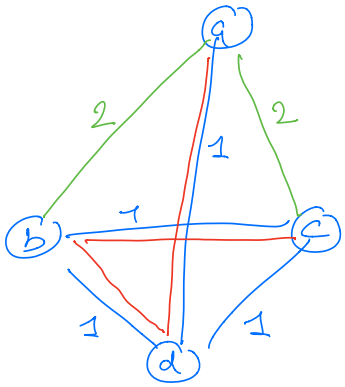
Typically NP-hard and NP-complete problems can be solved by exponential algorithms.

Example Hamiltonian cycle.

Find a path in a graph that goes (close) through all nodes once and only once, using the edges.



— hamiltonian cycle.



no way back to a : no hamiltonian cycle

We can use a TSP algorithm to decide if a graph G has or not an hamiltonian cycle.

Add a weight 1 to all existing edges of G

Add missing edges with weight 2 (to make a fully connected graph).

Now solve the TSP problem of this fully connected graph. If the shortest path is

of length n an hamiltonian cycle exists.
If it is larger it does not exist.

Conclusion a solution of TSP gives a solution of hamiltonian cycle (which actually is NP-complete), but TSP is not NP because one cannot check in a polynomial time whether a tour is really the shortest.

So in this course we want to find ways to solve NP-hard optimization problems

1.4 Metaheuristics

Some optimization problems only have exponential algorithms, and they cannot be solved exactly
Too much CPU time when the search space S become even moderately large.

Metaheuristics are a way to explore S in hopefully a smart way. We want to find optimal solutions with an acceptable computational time and enough quality.

Metaheuristics was introduced by Glover

in 1986

Heuristics : can be defined as a strategy to find solution to a problem

Example : SIMPLEX : choice of the pivot

Specific to a given problem.

Metaheuristics : also a strategy to find a solution, but valid for many different problems

Characterization of metaheuristics

- No hypothesis on the mathematical properties of the fitness function. (No need for continuity, differentiability, connexity)
- Require "guiding" parameters that specify how the search space is explored. Their optimal value depends on the problem.

- Need an initial starting point to explore S. Usually chosen at random, but can be better if additional info/ is known.
- Stopping condition is typically given by: (1) limit of CPU: used all CPU time allocated to the search
  (2) a stagnation condition: no more improvement for many iteration.

  Such end conditions are needed because a Metaheuristics cannot know whether it found the optimal solution.

  Note a limit on the number of iteration of the search process is a CPU limit: CPU means here a work done by the algorithm.

- Most metaheuristics are stochastic they use random numbers to guide the search.

- They are often CPU-consuming but easy to code and even to parallelize.

- Metaheuristics are based on two mechanisms called intensification and diversification, or alternatively exploitation and exploration.

One is about trying various regions of the search space, the other one is about improving locally the quality of the solution.

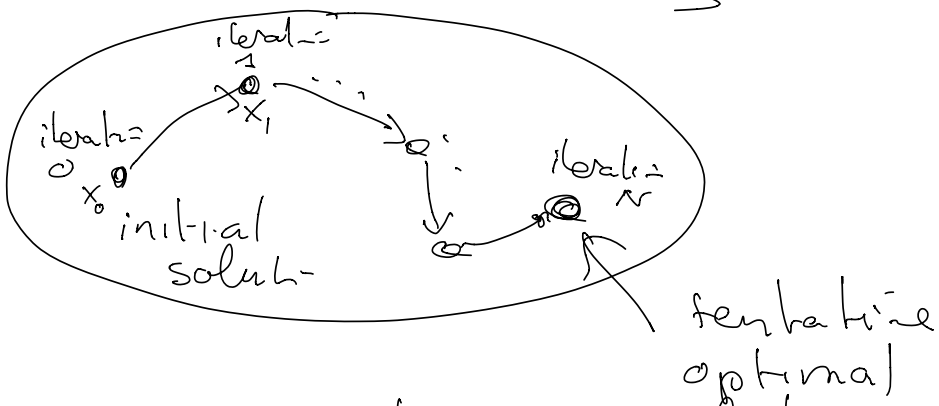
Note: use metaheuristics only when no other method can be used efficiently.

1.5 Main principle

First, the search space S should be coded in mathematical way

Crepresent all possible solutions in way that the computer can use)

. A metaheuristics is a trajectory in S



So it is an iterative process where one moves from the current solution to the next according to the following pattern:

$$x_0 \xrightarrow{u} x_1 \in V(x_0) \xrightarrow{u} x_2 \in V(x_1) \rightarrow \dots$$

$V(x_0)$: neighborhood of x_0

Here $V(x)$, $x \in S$ is the set of solutions that can be reached from

x. This has to be specified for each problem and metaheuristics.
U is called the search operator. or the exploration operator. It picks from $V(x)$, the most promising successor of x
Each meta_heuristics has its own U.

How to define the neighborhood

$V(x)$? for any $x \in S$
A way would be to make an explicit list of all point in $V(x)$, for each x

$$V(x) = \{ y_1^{(x)}, y_2^{(x)} \ldots y_n^{(x)} \}$$

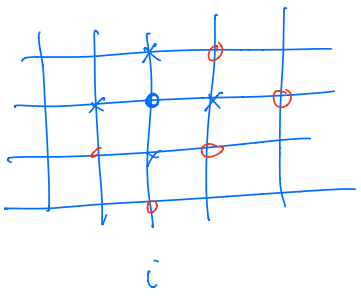
This not practical in general.
Better to define $V(x)$ through transformations T_i (or movement)

$$\left[V(x) = \{ x' \in S \mid x' = T_i(x), \; i = 1, 2, \ldots \ell \} \right]$$

Here we have ℓ movements that produce

a neighborood with l candidate,

Example $S = \mathbb{Z}^2$ Transformation



North (i, j)
$\rightarrow (i, j+1)$

South (i, j)
$\rightarrow (i, j-1)$

Same for East, and West.

Example S is the space of permutations

S contains all the ways to place n objects on n positions.

$n=3$ $S = \{(a\ b\ c), (a, c, b), (b, a\ c), (b\ c\ a), (c, a\ b), (c, b\ a)\}$

How can one define a neighborhood of a permutation?

A standard way is to generate neighbors by exchanging to objects

$$(a_1\ a_2\ a_3 \dots a_i \dots a_j \dots a_n).$$

$$T = \xrightarrow{(i\ j)} (a_1\ a_2\ a_3 \dots a_j \dots a_i \dots a_n).$$

For instance, for $n = 3$ the movements could be

$$T_i \in \{(1,2), (1,3), (2,3)\}$$ 3 elements

Note $(1,1)$ is not interesting and $(1,2)$ is the same as $(2,1)$

$n = 4$

$$T_i \in \{(1,2)\ (1,3)\ (1,4)\ (2,3)\ (2,4)\ (3,4)\}$$ 6 elements.

We see that this produces neighborhood of size $O(n^2)$: n possibilities for the i value el $n-1$ for the j value. And one divides this number by 2.

Note S grows exponentially with n and $V(x)$ grows quadratically.

There are other ways to generate neighbors of a formulation. One can restrict to exchanges like $(i, i+1)$
$\Rightarrow |V(x)| = O(n)$

Yet another way:

$(a_1\ a_2\ a_3 \dots a_i \dots a_j \dots a_n)$

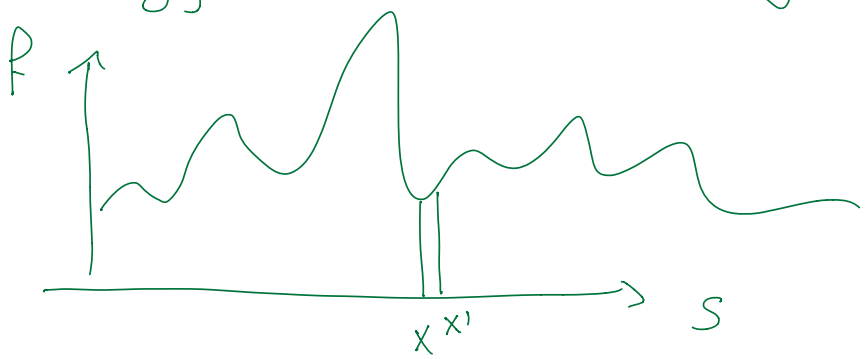
displacement + insertion. $\Rightarrow |V(x)| = O(n^2)$

One define a local search if

$V(x)$ is small in comparison to S

Energy landscape (or fitness landscape)

It is the representation of $f(x)$ according to the topology induced by the neighborhood



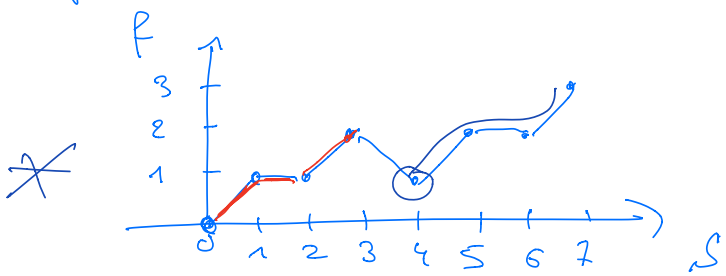
here x' is by definition a neighbor of x
That's why they are close on the x-axis.

For the drawing, the fitness landscape is limited to simple neighbors: $x-1, x, x+1$
For larger $V(x)$ it is hard to draw.
But the fitness landscape tells how different the fitness of the neighbors are in $V(x)$

and this relates to the difficulty to pick the next best candidate.

The neighborhood affects the efficiency of the search.

We consider here the maxOne problem.
Take the case of 3 bits $(x_1\ x_2\ x_3)$
which we can see as a value between
0 and 7 $\quad 0 = 000 \quad 7 = 111$
The fitnes is the number of 1s in x

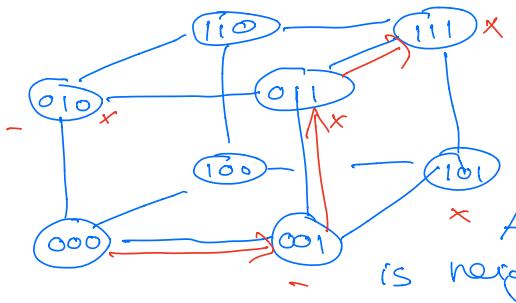


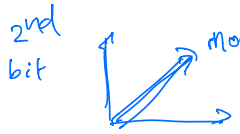
$V(x) = \{x-1,\ x,\ x+1\}$

Consider a hill climbing metaheuristics: go to the highest of the neighbors

If we start from $x_0 = 0$, one typically get stuck at $x_3 = 3$, after 3 iterations

Different neighborhood (hypercube)





A point x' is neighbor of x if x and x'

Here we found the global optimal $x_{opt} = (1\,1\,1)$ in 3 iterations.

differ by only 1 bit out of 3. Now x has 3 neighbors.

Before we were blocked in a local optimal!

Example of simple metaheuristics

- Random walk: pick a neighbor at random, disregarding its fitness.
- Random search: pick at random a successor in S
- Hill climbing: take the best of the neighbors
- Stochastic hill climbing: pick randomly a neighbor with probability according to fitness value.

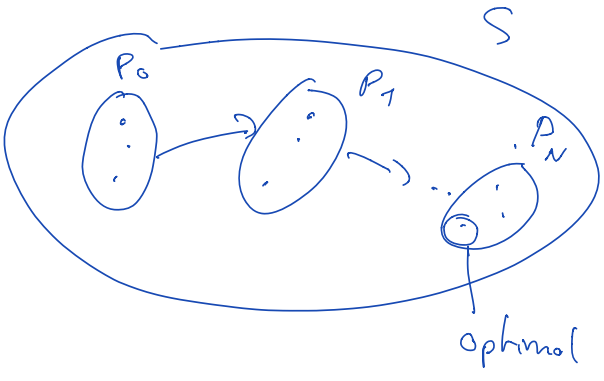
Population metaheuristics

So far we assumed that at each iteration we consider only one possible solution:

$$x_0 \to x_1 \in V(x_0) \to x_2 \in V(x_1) \dots$$

But we could also consider a population of solution:

$P_0 = \{ x_1^{(0)}, x_2^{(0)}, \dots x_M^{(0)} \}$ M possible solutions at iteration 0



Chapter 2 The tabu search

2.1 Basic principles

- Proposed in 1986 by Glover
- Adapted to Quadratic Assignment problems (QAP)
- Explores the space from neighbor to neighbors

$$x_i \to x_{i+1} \in V(x_i)$$

The operator U chooses the best $x' \in V(x)$ / best non-tabu candidate.

In $V(x)$ there might be forbidden solutions.

U takes the non-tabu $x' \in V(x) - \{x\}$ which has the best fitness

IF more than one x' has this best fitness,

one is chosen at random.

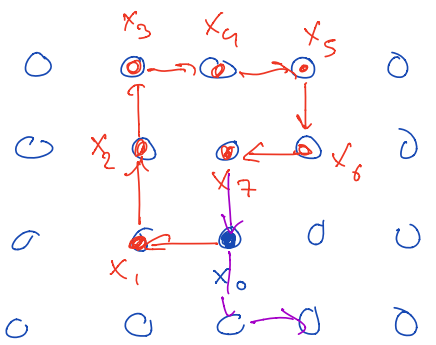
- The specificity of the tabu search is the existence of a tabu list which indicate points in S that are not allowed, or movements that are forbidden.

- The main goal of the tabu list is to prevent the exploration to return to an already visited point

- It is important to note that the attribute "tabu" is no permanent. The tabu list is always updated, iteration after iteration.

Example showing why the tabu list cannot be permanent

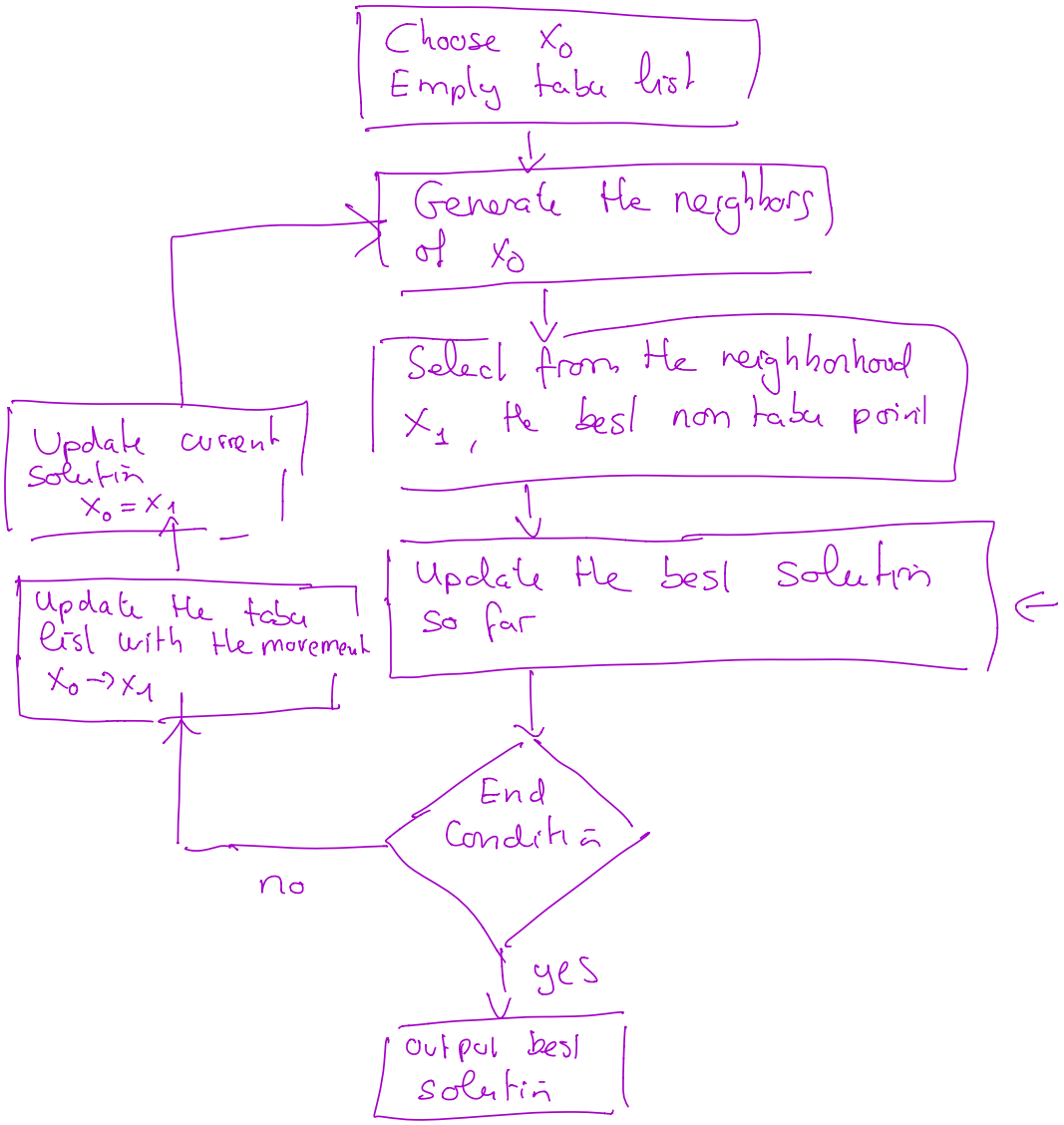
walker in $\mathbb{Z}^2$, with the neighborhood North, South East and West.



If all the visited points are in the tabu list, once in x_7 there is no succesor possible

The algorithm would stop due to a lack of successor. So one has to modify the tabu list dynamically.
For instance, the oldest tabu point could be removed from the tabu list. So x_0 would be accessible again

Flow chart of tabu search



Convergence

The convergence of a metaheuristics is its property to find the global optimum. Taba Search will converge if

- S is finite
- the neighborhood is symmetric $x \in V(y) \Leftrightarrow y \in V(x)$.
- Any $y \in S$ can be reached by a finit number of steps.
- Then, a taba search that memorizes all the visited point in the taba liste but allows the search to restart from the oldest point in the taba list will converge.

In other words taba search will explore all the space. But in a much inefficient way. A exponential time in $|S|$ is expected.