1- Optimization problems and the search space

-Search space S is a set of possible values for X S= R^2 in the 2d median example.
-can be continuous,descrete,finite or infinite
-S is very BIG (no exhaustive search)

-F is a function wich quantify the quality of an x

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

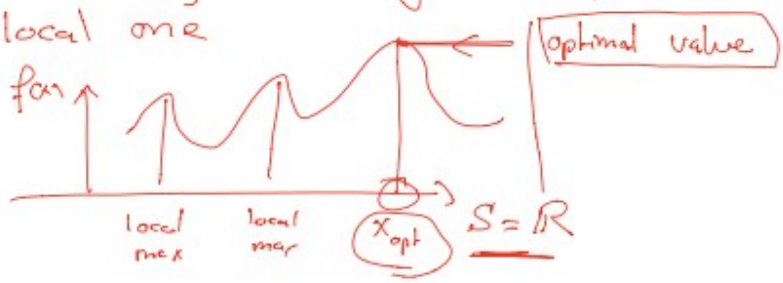
-called objective function , cost function,energy function, fitness(we use fitness)

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

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

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

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



-Optimal value unique
Exemple de problems :

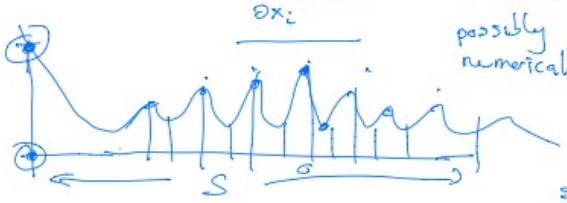
1) $S = \mathbb{R}^n$ $\quad f: \mathbb{R}^n \to \mathbb{R}$, $\quad f$ is continuous and differentiable

$\Rightarrow$ 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.

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

$$x_i = \begin{cases} 0 & \text{I do not take the object} \\ 1 & \text{I take the object} \end{cases} \qquad \boxed{S = \{0, 1\}^n}$$

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

$$\text{subject to} \quad \sum x_i w_i \le C$$

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

4)tsp

5)NK-landscape

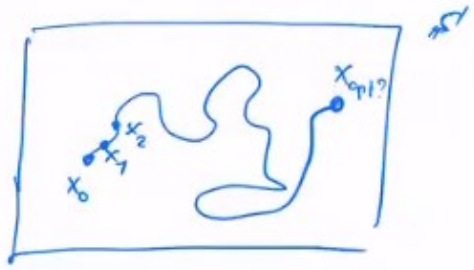
2- Main principles of metaheuristics, neighborhood, movements, exploration operator, population metaheuristics

Goal : explore the search space in a clever way, needed for larges problems for witch exponential algorithm a known
-compromise between cpu time and the quality of the solution
-find a solution,but valid for many different problems
-no guarantee on the quality

Characteristiques:
-no hypothesis on the mathematical properties of the fitness function.
-Require guiding parameters that depend on the problem and specifies how to search in the space.
-need a starting point
-need a stoping condition(iteration,no more improvement,time,value etc)
-inspired by natural processes
-can be parralelisable
-most of them are stochastic,use random numbers to guide search
-based on 2 mechanismes : intersification or diversification and exploration or exploitation
One is about trying various regions of the search space and the other is about improving locally the quality of the solution.

Main principle:

A metaheuristics is an explonation trajectory in the search space S



To move across S we use the concept of neibohood all the point that can be reached from a proint x.

$x_0 \longrightarrow x_1 \in V(x_0) \longrightarrow x_2 \in V(x_1) \longrightarrow \cdots$ until the stopping condition

initial solution

-Voisins trouvées via une transformation elementaire(changement de bit ,echange de valeur ..)

Population metaheuristics

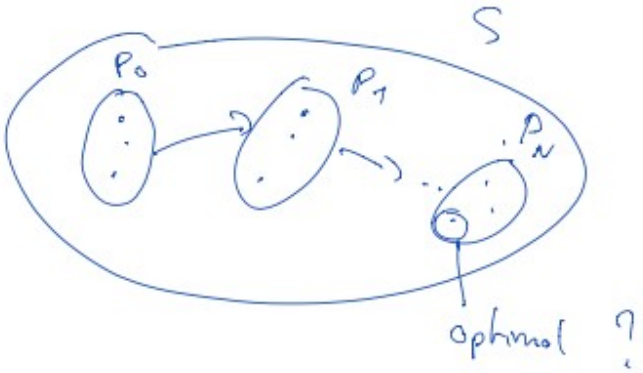
So far we assumed that at each iteration we consider only one possible solution:

$$x_0 \longrightarrow x_1 \in V(x_0) \longrightarrow x_2 \in V(x_1), \ldots$$

But we could also consider a population of solution:

$$P_0 = \{ x_1^{(0)}, x_2^{(0)}, \ldots x_M^{(0)} \}$$

M possible solution at iteration 0



3- The space of permutations

4- Complexity and the need for metaheuristics, exploration versus Exploitation

- 2 mechanismes : diversification and exploration

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

-To solve an optimisation problème, we need an efficient algorithm in term of exécution time and memory usage

Complexity classes :

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

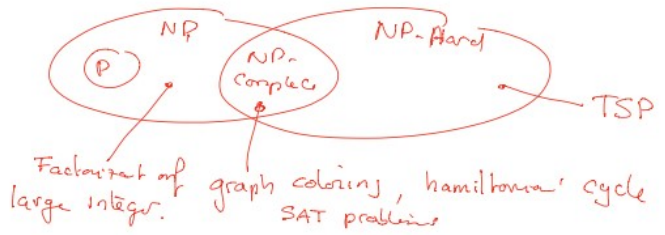
Class NP : Problems for which a solution can be checked in a polynomial time.

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-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.

-Np-hard and np-completes problems can be solved by exponential algorithms

-example: Hamilton cycle
Find a graph that go through all nodes once and only once
Can use the tsp(traveling salesman) algo to a graph to solve it.
Algo is np hard because one cannot check in a polynomial time whether a tour is really the shortest.
We want to find ways to solve np-hard optimization problems.

5- Random search, random walk, hill climbing

- 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.

6- NK-problems: motivation, definition, goal.

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 strategies : $x_i = 0$, $x_i = 1$ $i = 1, \dots N$

The success of these strategie depend of the choice of strategy of the people one is interacting with. (That is wheather people compete or collaborate)

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

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

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 profit of the population.

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

Example $\boxed{\text{MaxOne problem.}}$

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

Obvious solution: $x = (x_1 x_2 \dots x_N) = \underbrace{111 \dots 1}_{N \text{ times}}$

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

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

$f = N$

Another example

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

f_i is max for 1 1 1 then it is a trivial solut

But if f_i is large for 101 and small for 010

then the chain 1 0 1 0 1 0 1 0 1

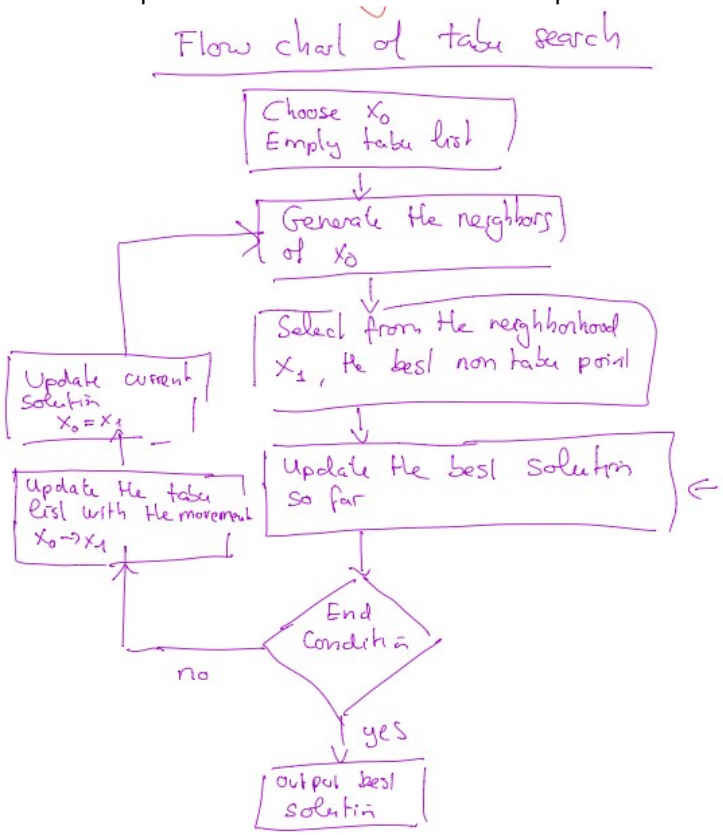
	f
0 0 0	x
0 0 1	x
0 1 0	x
0 1 1	x
	⋮
1 1 1	

7- Tabu Search: main principles and convergence

-adapted to quadratic assignment problems(QAP)
-explore the space neighbor to neighbor:

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

-we use the best non tabu x
-if more than one has the best taboo we choose at random
-the actual solution can beworst than the best one
-we use a taboo list to orevent the exploration to return to an already visited point
-taboo list not permanent because we could end up without succesors



Convergence :
-is a property to find the global optimum.
-tabbu will converge if :
-S is finite
-The neigborhood is symmetric x apantenant a V(y) ⇔ y apatenant a V(X)
-Any y apartenant au search space peux étre attain par un nombre finis d'étapes
-Une recherche taboo qui memorize tout les point dans la tabu list mais qui peux recomencer depui le point le plus ancien de la liste convergerra
-en conclusion tabuu search va chercher tout l'espace mais de maniére plus efficace.(temps exponentiel atendue)

8- The different ways to implement the tabu list.
-management a short term memory and a long term memory

-short term memory
    -use a finite size tabu list and kept the most recent and kick the old ones
    -associate a duration to the tabu items(baned time or tenure time)
    No known optimal value for the tenure time,determined empiricaly

-long term mermory used to prevent that some movement are never used
Record statistical information on the called movement
-again no known optimal on the way to do it

-short list = exploration
-long list = exploitation

-we can use aspiration(if a neighboor has the best fitness we use it even if it is in the taboo list)

9- Quadratic Assignment Problems

-Definiton: Combinatorial optmizaton problem

-Example: Find the best way to assign a set of n facilities (factories) to a set of n locations (cities) accordingly to distances and flows (amount of things that needs to be moved)

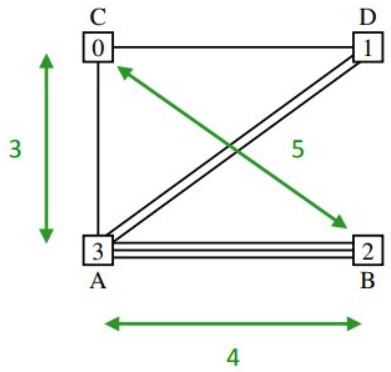
- minimize the sum of products<<dustance flow>>

Research space: Permutations —> Size $n! = n \times (n-1) \times \ldots \times 1$

- Example: Find the best location (A, B, C, D) for each facility (0, 1, 2, 3) in order to minimize

$$I(\psi) = \sum_{i,j=0}^{n-1} w_{ij} \times d_{\psi_i,\psi_j}$$

distances	flows
$d_{AB} = d_{CD} = 4$	$w_{13} = 2$
$d_{AC} = d_{BD} = 3$	$w_{01} = w_{03} = 1$
$d_{AD} = d_{BC} = 5$	$w_{23} = 3$



Fitness $I(\psi) = w_{01} \times d_{\psi_0\psi_1} + w_{03} \times d_{\psi_0\psi_3} + w_{13} \times d_{\psi_1\psi_3} + w_{23} \times d_{\psi_2\psi_3}$

Here $\psi = (C, D, B, A)$

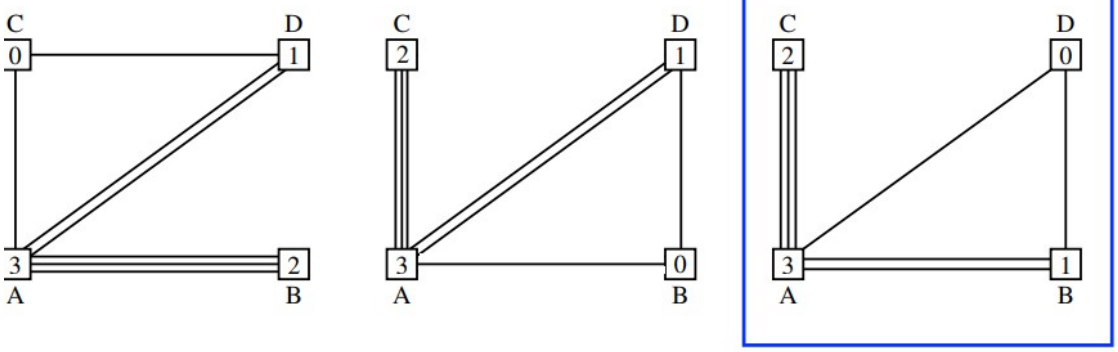
Hence

$$\boxed{I(\psi) = d_{CD} + d_{AC} + 2d_{AD} + 3d_{AB} = 29}$$

- Neighborhood: Permutations of two elements (2-swap)

$\longrightarrow$ $n(n-1)/2$ neighbors

$\psi = (C, D, B, A)$ $\psi' = (B, D, C, A)$ $\psi'' = (D, B, C, A)$

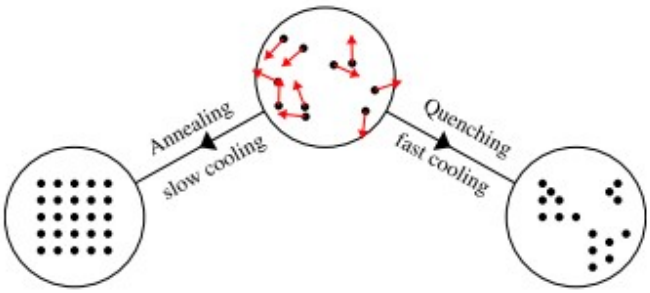


10- Simulated annealing, main principles and the Metropolis rule

-metaheuristique inspired by nature

-annealing is a process by wich a sample is colled down slowly to reach a minimum energy

-Quenching is a quick cooling down-> reaches a local minimum of energy



-We want to find the global minimum of the firness function

-we have to specify the search space and the neiborhood

-we choose a random initial configuration

-At each iteration :

3. **Elementary Configuration Update:** Randomly transform the solution.

4. **Acceptance/Rejection rule:** Keep the update with the probability given by the metropolis rule (figure 2):

$$P = \begin{cases} 1 & \text{if } \Delta E < 0 \\ e^{\frac{-\Delta E}{T}} & \text{otherwise} \end{cases} = \min(1, e^{\frac{-\Delta E}{T}}) \tag{2}$$

5. **Equilibrium conditions:** Consider that equilibrium is achieved if either $12n$ perturbations were accepted or $100n$ perturbations were attempted.

6. **Temperature Reduction:** Use the following simple rule:

$$T_{k+1} = 0.9T_k \tag{3}$$

7. **Freezing conditions:** The system is considered frozen if there were no fitness improvement during the last 3 temperature steps.

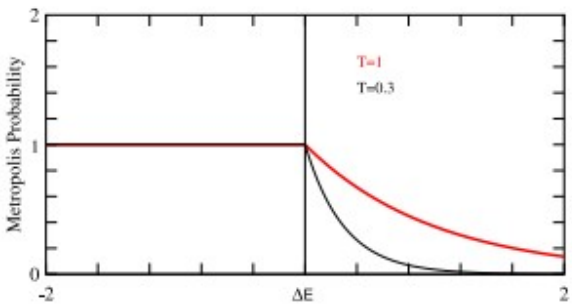


Figure 2: Metropolis rule of equation (2), taken from [1].

11- Simulated annealing, flow chart and choice of parameters

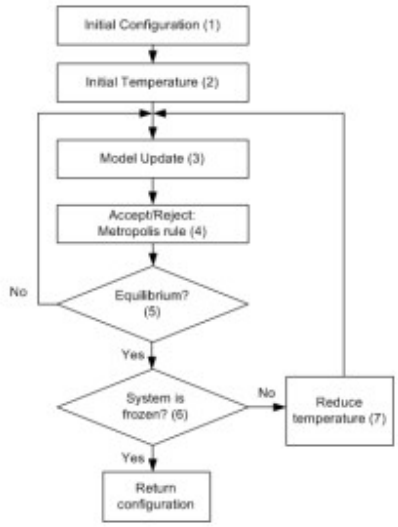


Figure 3: Simulated Annealing (SA) diagram.

. **Initial temperature:** Starting from the initial configuration, compute 100 transformations (permuting two cities) and compute the average energy change $\langle|\Delta E|\rangle$, where $\Delta E = E(x_{\text{neighbor}}) - E(x)$. The initial temperature then can be computed using the equation:

$$e^{\frac{-\langle\Delta E\rangle}{T_0}} = 0.5. \quad (1)$$

Choose an initial acceptation rate T0 of bad solution(depending of fitness)
Ex :t0 = 0.2 if initial condition is good. Choose 0.5 if poor

-Temperature reduction
We decrase the temperatur acording to a geometric law

$$T_{k+1} = 0.9T_k$$

-Termination condition
No improvement dureing the last 3 iterations
-validation
Lancer quelque fois et verifier si la valeur est consistante

12- Convergence of simulated annealing: how to formulate it, how to compute it.

-metaheuristique that can be analised mathematicaly
-result converge in probability,it gives a solution arbitrary class to the global optimum with a probability arbritrary close to 1

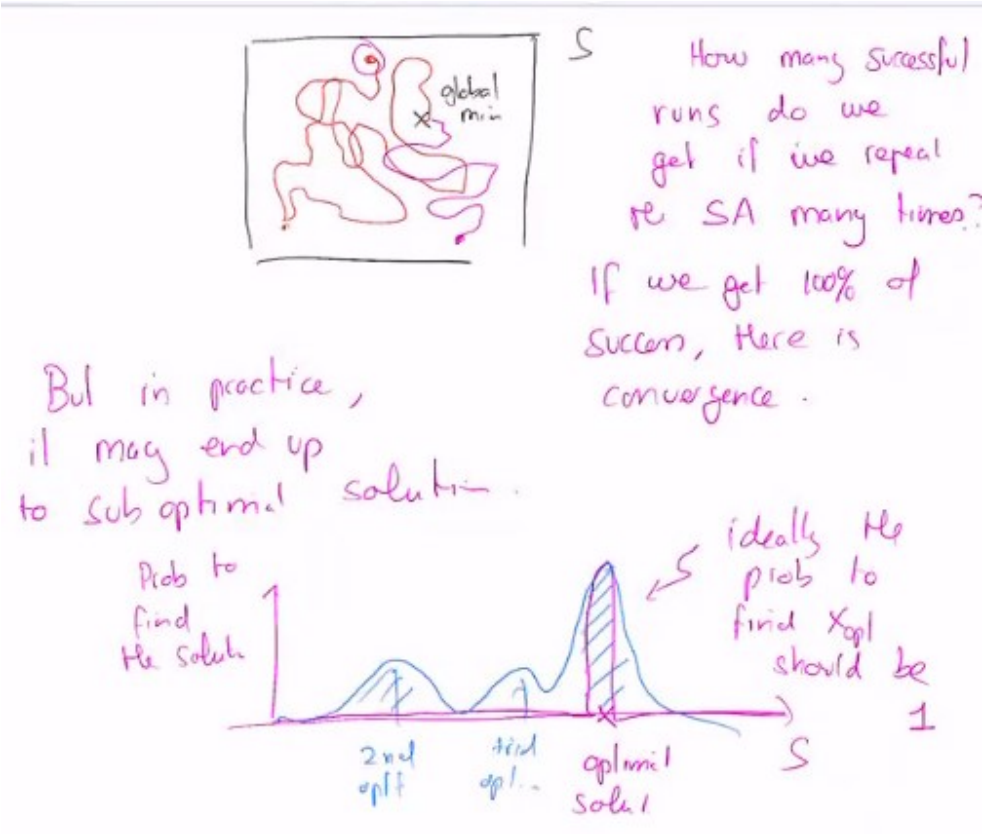
Conditions:
-movements are reversible
-any point in S can be reached in a limited number of movements
-temperatur should not decrease too fast

$$T(t) \sim \frac{C}{\log t} \quad \text{for } t \gg 1$$

C is an unknown constant that reflect the variation of E in S

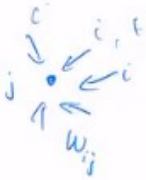
-This schedule of temperature is to slow in practical problem (log(t)
-in practice change faster with the risk of no converging

Ilustaration with simple example:



We want to compute the probability $P(t, i)$ that the SA is at point $i \in S$ at iteration t

$$P(t+1, j) = \sum_i P(t, i)\, \widetilde{W_{ij}}(t)$$



W_{ij} is the transition probability from i to j. Since we have 100 possible values of i an j, W_{ij} is a

$\underset{|S|}{100} \times \underset{|S|}{100}$ matrix.

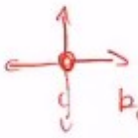
?

For SA, we have:

$$W_{ij} = \begin{cases} 0 & \text{if } j \text{ is not a neighbor of } i \\ \dfrac{1}{k_{out}(i)} \times P_{metropolis}(E_i, E_j, T(t)) & \end{cases}$$

number of neighbors of i

$\min\left(e^{-(E_j - E_i)/T(t)}\right)$



$k_{out}(i) = 4$ in our example.

We also add the following

$$W_{ii} = 1 - \sum_{j \text{ neighbour of } i} W_{ij}$$

?

We want to compute $P(t, k)$ for $k \in S$ as a function of $P(0, \ell)$ the initial probability to start the search from $\ell \in S$

$$P(t, k) = \sum_j P(t-1, j) W_{jk}(t-1)$$

$$= \sum_j \sum_i P(t-2, i) \underbrace{W_{ij}(t-2) W_{jk}(t-1)}_{[W(t-2) \times W(t-1)]_{ik}}$$

← matrix product

$$= \sum_i P(t-2, i) [W(t-2) \times W(t-1)]_{ik}$$

$$= \cdots \quad P(t-3$$

In the end we get:

we can compute it with our example

$$P(t, k) = \sum_\ell P(0, \ell) \underbrace{\left[\prod_{t'=0}^{t-1} W(t')\right]_{\ell k}}_{\equiv W(0, t-1)} = \sum_\ell P(0, \ell) W(0, t-1)_{\ell k}$$

If $T(t=0)$ is large enough, we observe that all the rows of $W(0, t-1)$ are equal.

$$W(0, t-1)_{\ell k} = W(0, t-1)_{1k} \quad \forall \ell$$

Thus $$P(t, k) = \sum_{\ell} P(0, \ell) W(0, t-1)_{\ell k}$$

$$= W(0, t-1)_{1k} \underbrace{\sum_{\ell} P(0, \ell)}_{1}$$

1 because we have to start somewhere.

$$= W(0, t-1)_{1k}$$

Thus $P(t, k)$ does not depend where the search was started: independence of the initial condition.

-we need a big temperature to be independent to the initial condition

-The temper should decrease slowly enough to converge only toward the global minimum

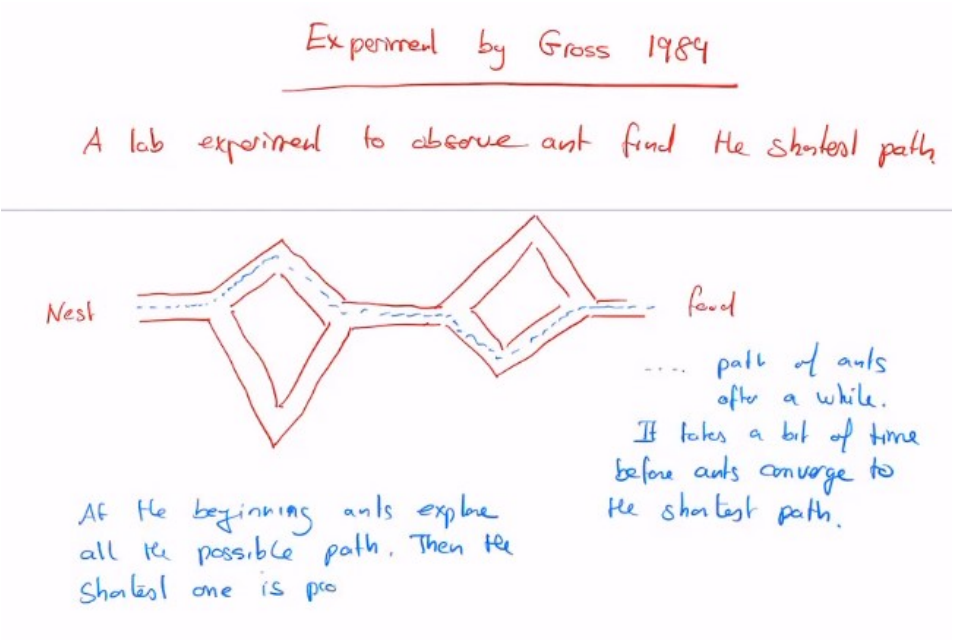
13- Ant-like algorithms: swarm intelligence, the pheromone trail, and observations about real ants.

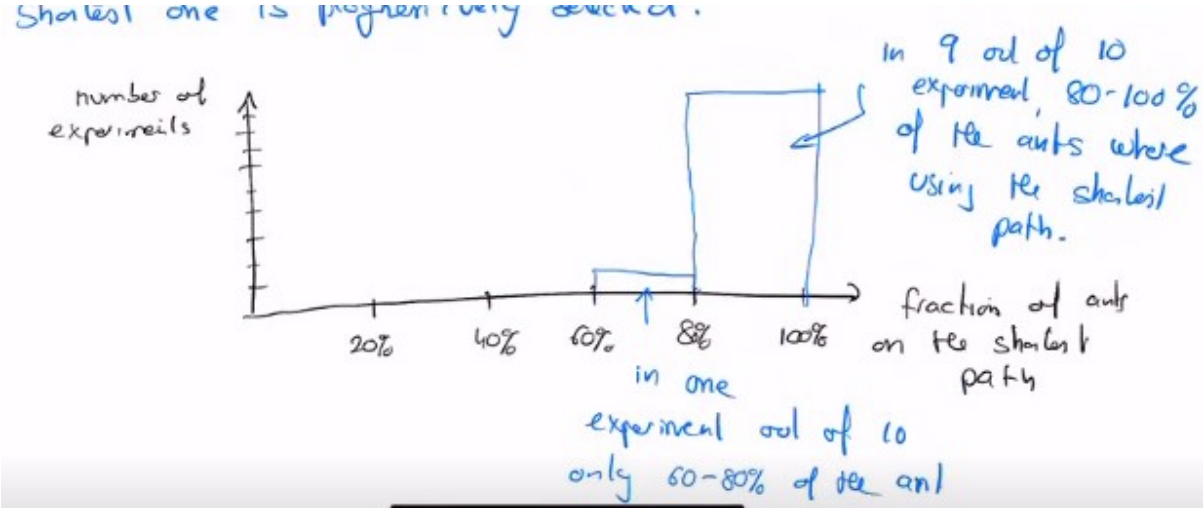
-ants are able to solve optimization problems such as to find the shortest path between their nest and a food suply

Swarm intelligence : despite a limited individual capability, the collection of all individuals can produce behaviours that are more than the sum of individual capability
-colaboration key of sucess
-signature of complex systhems
-collextion of simple entities that interact
-Give the global emergent behavior wich cannot be understand from the behavior of a single entity
-large scale spatial and temporal organisation
-no central control

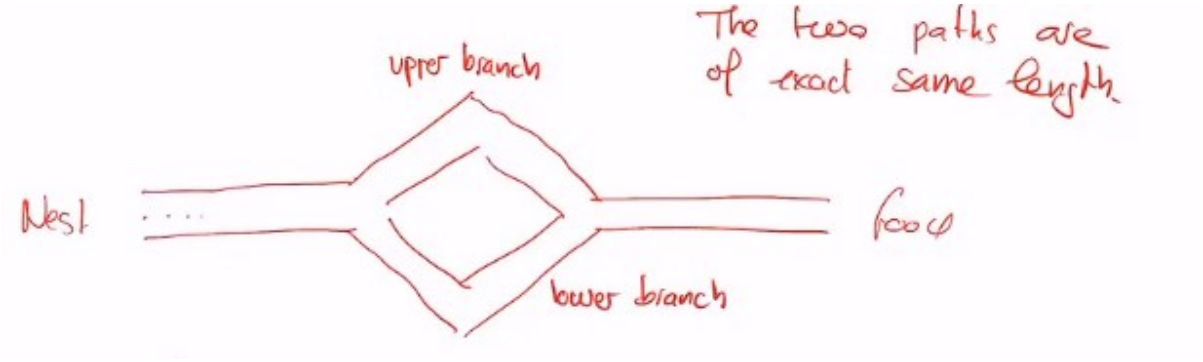
Advantages
-robustes to fault tolerance :
process continue even if an ant disapper or disfunction
adaptation to new situations
natural paralelism

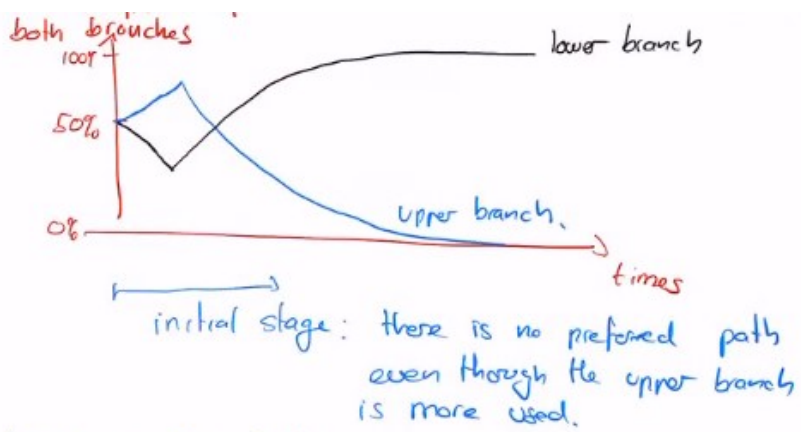
pheromon trail
-how they find the shortest path ? pheromons
-attract other ants
-guide ants allong paths
-move toward place the smell is stronger
-evaporate after some time. Must be alway used to stay





-we start with a random path
-the lucki one find the food first and comme back making the path stronger
-more ants pass and make it stronger
-The other path are gradually weaker





At some time, a fluctuation may reverse the situation and the other path get preferred.

14- Ant System: description of the algorithm for the TSP problem

- Adapt Pheromone Trail to solve TSP
- Visibility: $\eta_{ij} = \frac{1}{d_{ij}}$
- Trail Intensity: $\tau_{ij}(t)$
- m (# of ants)

Algorithm 1

```
for all t = 1, ..., t_max do
    for all ant k = 1, ..., m do
        choose a city at random
        while there exists a city not visited do
            choose a city j according to (1)
        end while
        mark a path according to (3)
    end for
    update all paths according to (2)
    Keep the best of solutions obtained at last iteration
end for
```

$$p_{ij}^k(t) = \begin{cases} \frac{(\tau_{ij}(t))^\alpha (\eta_{ij})^\beta}{\sum_{l \in J} (\tau_{il}(t))^\alpha (\eta_{il})^\beta} & \text{if } j \in J \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^k(t) \quad (2)$$

$$\Delta\tau_{ij}^k(t) = \begin{cases} \frac{Q}{L^k(t)} & \text{if ant } k \text{ used edge } (i,j) \text{ in its tour} \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

Choice of Parameters

- $\alpha = 1$, $\beta = 5$, $\rho = 0.1$
- $Q = L_{nn}$, $\tau_0 = \frac{1}{L_{nn}}$
- m , t_{max}

15- Ant algorithms: the simple version in {0,1}^N, and discussion of Performance

(aller directement aux performances)
-Ant colony systhem : variation of AS
-AS failed to give better aswer for larges problems(not bigger than oliver30)
-ACS is a different way to choose the next town and update pheromone trail

A new parameter q_0, whose value is between 0 and 1, is introduced such that in iteration $t+1$ ant k, having reached city i, chooses the next city j according to the rule

$$j = \begin{cases} \text{argmax}_{\ell \in J}[\tau_{i\ell}(t)\eta_{i\ell}^{\beta}] & \text{with probability } q_0 \\ u & \text{with probability } 1 - q_0 \end{cases} \tag{5.8}$$

where

$$\text{argmax}_x f(x)$$

The quantity $u \in J$ is a randomly chosen city in the set J of allowed cities which is drawn with probability p_{iu}

$$p_{iu} = \frac{\tau_{iu}(t)\eta_{iu}^{\beta}}{\sum_{\ell \in J} \tau_{i\ell}(t)\eta_{i\ell}^{\beta}} \tag{5.9}$$

Thus, with probability q_0 the algorithm *exploits* the known information as it chooses the best edge available. Otherwise, with probability $1 - q_0$, the *exploration* of new paths is privileged.

Pheromone :
-each ant will deposit an amount phiT0 on each link it used
-at the same time an amount 1-phi of the existing pheromone will evaporate

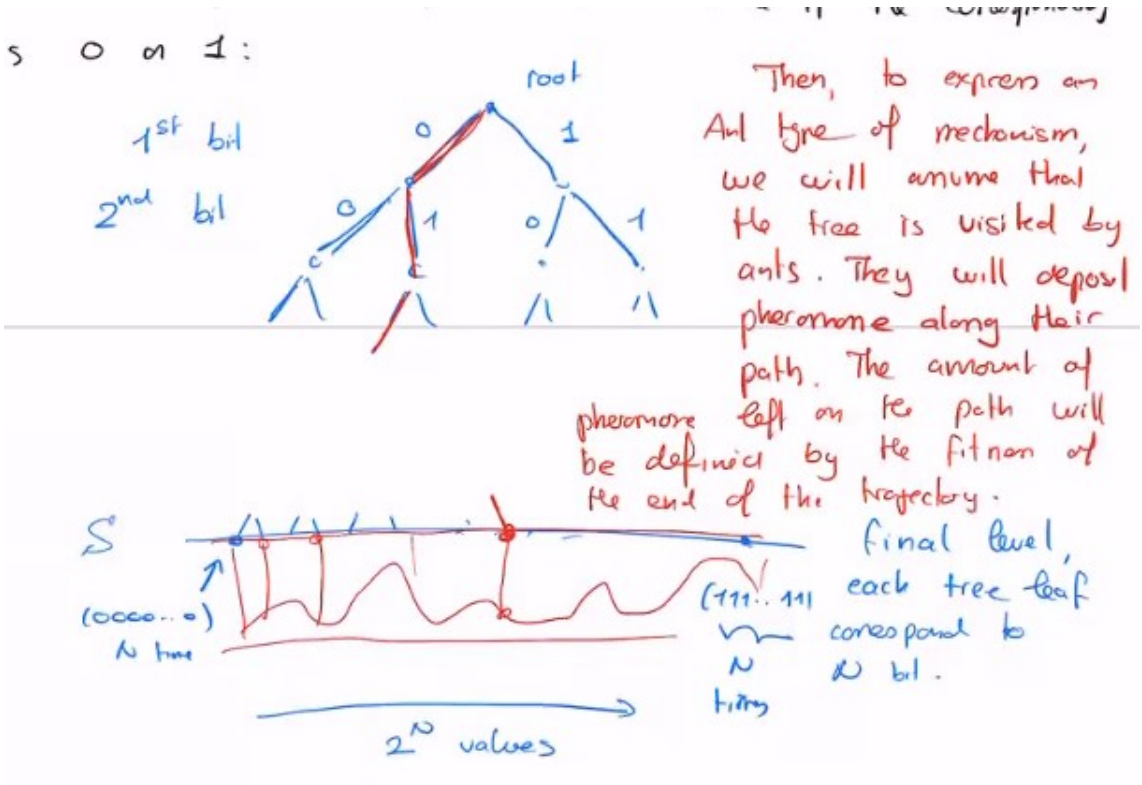
$$\tau_{ij} = (1-\phi)\,\tau_{ij}(t) + \phi\,\tau_0$$

τ_{ij} is here an intermediate value.

Then, one adds an amount $\Delta\tau = \frac{1}{L_{min}}$ only on the links that belong to the shortest path, of length L_{min}.

$$\tau_{ij}(t+1) = \underbrace{(1-\rho)}_{\text{extra evaporation, probably for empirical reason.}}\tau_{ij} + \Delta\tau_{ij}$$

Verssion simple
-we consider a simple benchmark
-we propose anthother version of the algorithm.
-we want to find a bit string wich maximaze a bit function S = {0,1}^N
-we explore a search space in the form of a tree structure i wich each level we decide if it corespond to a bit 0 or 1



-comment on decide quelle chemin prendre ? le nombre de féromone a chaque branche
-avec une probabilité q0 on choisie la meilleur des 2 branches.
-Avec une probabilité 1-q0 le choix serra au hasard pae rapport a :

$$P_{left} = \frac{\tau_{left}}{(\tau_{left} + \tau_{right})} \qquad P_{right} = 1 - P_{left}.$$

Performance :

Statistiques sur 10 répétitions, chacune avec 6 fourmis, après 6 itérations.

effort computationnel	$6 \times 6 = 36$
taux de succès :	7/10
taux de succès à 3% de l'optimum	9/10

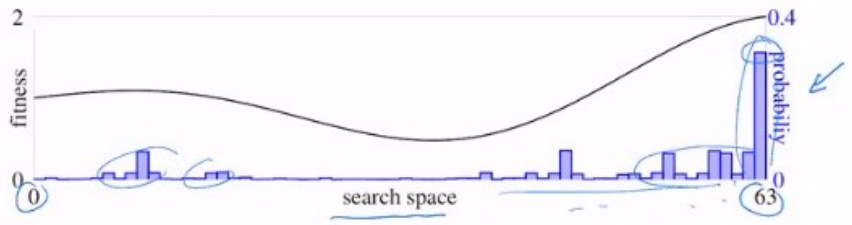
we explore 36 configs. rah-

← success rate

What about a random search ?
Let us take 36 points of S, at random. What is the probability to find the optimal value within those 36 solutions?
$p = 1/64$ is the prob to find the optimal value at random.
With 36 attempt, the prob to find it is
$\boxed{0.43} = 1 - (1-p)^{36}$ is the prob to find it at least once.

Performance de la recheche «Fourmi»

Probabilité d'atteindre chacun des 64 points de S, après 100 itérations.



Ici on a pris $m = 6400$ fourmis tout au long des 100 itérations. Les valeurs indiquées de probabilités représentent la fraction des 6400 fourmis qui a atteint chaque point x à l'itération 100.

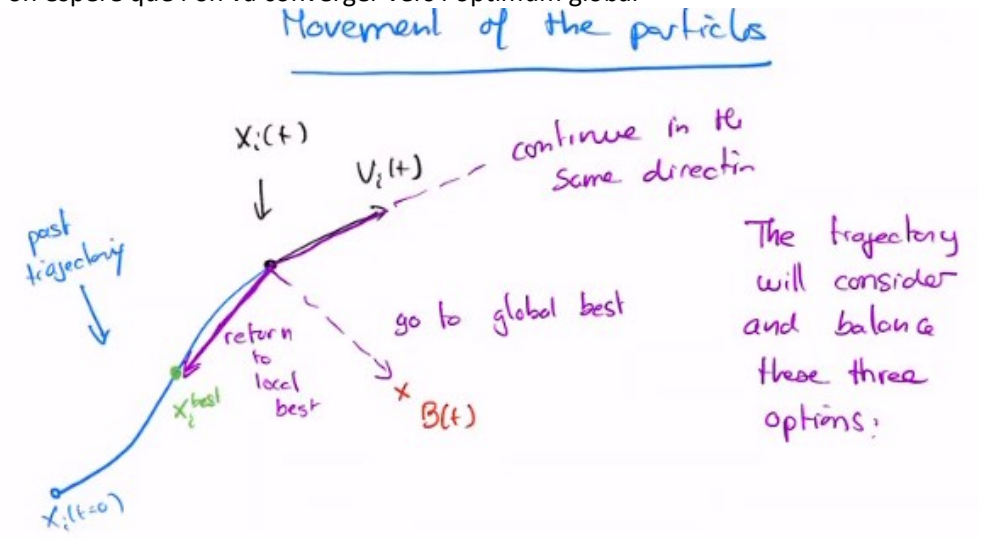
30% des fourmis ont trouvé l'optimun global

16- Particle Swarm Optimization: algorithm and example

-explore colectivelly the ssearch space in order to find an optimal solution

-the individula will follow the one who found the best spot and olso trus t their personal knowlege and explore zone that they think promisin

-on espere que l'on va converger vers l'optimum global



$$V_i(t+1) = \omega V_i(t) + c_1 r_1(t+1)\left[X_i^{best}(t) - X_i(t)\right]$$

$$\begin{cases} V_i(t+1) = \omega V_i(t) + c_1 r_1(t+1)\left[X_i^{best}(t) - X_i(t)\right] \\ \qquad\qquad + c_2 r_2(t+1)\left[B(t) - X_i(t)\right] \\ X_i(t+1) = X_i(t) + V_i(t+1) \end{cases}$$

ω is a parameter $0 < \omega < 1$ is an inertia parameter.
How much of the previous velocity we will keep

- C_1 is called the cognitif coefficient as it takes into account the knowledge of the individual.
- C_2 is called the social coefficient as it considers the knowledge of the group. usually: $C_1 = C_2 = 2$
- r_1 and r_2 are random numbers uniformly distributed in $[0, 1[$

In PSO it is common to use a different random number for each component of the equation.

Algorithm 1 PSO algorithm

1: Initialize
  t = 0;
  Initialize randomly the positions $\boldsymbol{s}_0^i$ in the domain to be explored and the velocities $\boldsymbol{v}_0^i = 0$ for all particles $i = 1..N$.
2: Do
  For each particle :
    Calculate its fitness J_t^i;
    If $J_t^i \leq J_{best}^i$ then $J_{best}^i = J_t^i, \boldsymbol{b}_t^i = \boldsymbol{s}_t^i$
  End for
  Calculate $J_{best}^G = min_i J_{best}^i$, update $\boldsymbol{b}_t^G = \boldsymbol{b}_t^{arg(min_i J_{best}^i)}$
  For each particle :
    Randomly generate r_1, r_2
    Update particle velocity by formula (1)
    Update particle position by formula (2)
  End for
  t = t + 1;
  Until (end criteria are met)

-on a des boudarie(on peux rebondir ou se telporter)
-on a une vmax
-on doit pourvoir definir le quantitée : velocité,addition,multiplication
-pas bon dans un espace de permutation-> continuous optimization instead of combinatorial oprtimzation.
-

17- The Firefly algorithm

-inspired by PSO
-Behaviour of the firefly that emit ligth
-we play with the intensity of the ligth
-for continuous optimization but exist for discrete

-chaque luciole occupe une position Xi(t) dans S a l'itération T
-chaque luciole a une solution possible possible
-plus sa fitness est forte plus elle brille
-l'intensité est :

$$I_i(t) = f(x_i(t))$$

intensity produced by firefly i.

fitness associated with its position

-on attire les luciales dans les zones a haute fitness
- a chaque ittération on considére toute les paire de lucioles (i,j) avec 1<i<n et 1<j<n , ou n est le nombre de lucioles
-Si Ii<Ij alors la luciole i bouge vers j.

Si $I_i < I_j$

L'amplitude du mouvement est donnée par l'attrachivité perque, qui dépend essentiellement de la distance entre deux lucioles.

$$\beta_0 \exp\left(-\left(\frac{r_{ij}}{\gamma}\right)^2\right)$$

où β_0 est un coefficient, r_{ij} la distance euclidienne entre les lucioles i et j, et γ un facteur qui dépend du choix de l'unité.

$r_{ij} \nearrow \Rightarrow$ attrachivité $\searrow$ $\qquad r_{ij} = 0 \Rightarrow$ attrachivité $= \beta_0$ et on choisit souvent $\beta_0 = 1$.

Quand $I_j > I_i$ alors le mouvement est

$$x_i(t+1) = x_i + \underbrace{\beta_0 e^{-\left(\frac{r_{ij}}{\gamma}\right)^2} (x_j - x_i)}_{\text{fraction of distance between i and j which is travelled}} + \alpha \underbrace{(rand_n() - 0.5)}_{\text{n random movement } rand_n() \in \mathbb{R}^n} .$$

mesure l'amplitude du mouvement aléatoire $\alpha \in [0,1]$

Pseudo - code

```
iteration = 0
initialize the position xi at t=0
intensity of light Ii = f(xi)
while iteration < maxIter
   for i = 1 to n
     for j = 1 to n
       if Ii < Ij then
            firefly i moves towards j according to formula above xi(t+1)
            update distances rij and Ii
     end for
   end for
end while
iteration + +
```

18- Genetic algorithms: inspriration and algorithm.

-inspiré de l'évolution de darwin
-on représente les individue dans s par des chromosomes ou son patrimoine génétique
-le degré d'adaptation est donné par la fitness
-la population evolue de génération en génération
-elitisme : on remplace les moin bon individues par le meilleur
-la taille de la population est constante

Pseudo-code

```
generation = 0
initialize population (at random)
while (not end-condition)
    generation ++
    compute fitness of all individuals
    selection of best individuals
    crossover
    mutation
    insert best      * keep the best so far
end while
output best
```

-condition d'arret itération ou plus d'amélioration

Soit $P(t)$ la population à génération t. On visualise l'évolution ainsi :

$$P(t) \xrightarrow{\text{selection}} P'(t) \xrightarrow{\text{crossover}} P''(t) \xrightarrow{\text{mutation}} P(t+1)$$

P, P', P'' et P''' de taille n constante.

-selection : on choisie les meilleur elements
-mutation : on aplique une mutation aléatoire(exploration
-crossover :is considered as fovoring diversity althrougth it act as intesification

19- The different selection operators

Selection proportionel a la fitness :

-On tire n individue et on en tire avec la probabilité pi :

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_i}$$

proportionel a la fitness

-implique que l'on veux maximiser f

-ne sais pas si certain f sont négatifs(corrigé si on ajoute a toute les valeur la fitness f*>0 tq fi + f*>0

-tend a prende les fitness semblable car pi très peux selctive et donc tous les individues ont les meme cchance d'étre choisi

-rajote de la selection avec :

$$\bar{f}_i = f_i - \min_{j \in P(t)} f_j$$

Sélection par rang (stochastique)

On trie les individus selon la qualité de leur fitness, du meilleur au pire.

La sélection va alors choisir plus volontiers les individus du rang élevé, indépendamment de la valeur de fitness. Les avantages de cette méthode sont :

- Cela fonctionne pour tout les problèmes de minimisation et maximisation.
- Cela fonctionne si $f_i < 0$ et/ou $f_i > 0$
- Cela donne toujours une différence entre le meilleur individu et pire individu

Sélection par rang linéaire : (stochastique)

La probabilité de choisir un individu dépend linéairement de son rang.

La solution générale qui donne p_i est

$$p_i = \frac{1}{n}\left[\beta - 2(\beta-1)\frac{i-1}{n-1}\right], \quad i = 1, \dots, n$$

$$p_1 = \frac{\beta}{n} \leq 1, \beta < 1$$

$$p_n = \frac{2-\beta}{n} \leq 1 \quad \beta \leq 2$$

$$0 \leq p_i \leq 1 \qquad 1 \leq \beta \leq 2$$

β est le nombre de copies expecté du meilleur individu dans P' et est un parametre libre avec contrainte p_1 et p_n.

ex: $\beta = n \times p_1$ $\Leftrightarrow$ après n tirages aléatoires avec remise, le nombre de copies de 1

Il est aussi possible de faire une sélection par rang logarithmique.

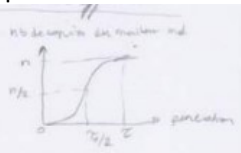
-Selection par tournois

20- The takeover time in genetic algorithms: goal, definition, value for tournament selection.

-on veut savoir encombien de génération le processus de selction aurra trouvé le meilleur individue

-takeover time : temps pour lequel le meilleur individue envahisse toute la population dans le ca sou le crossover et la mutation sont arrétées.mesure l'efficacité de la selection et aussi la perte de densité

de la sélection et aussi de la perte de diversité



On note τ le takeover time. Si τ petit, la sélection est de forte intensité.

τ long, la diversité reste longtemps donc donne plus d'opportunité pour l'exploration.

τ short, convergence prématurée.

-nous somme interessé par la relation entre le takeover time et la taille de la population

On verra que souvent τ est proportionnel à $\log n$. Si on veut décrire l'évolution du nombre de copies du meilleur individu $m(t)$, on peut écrire la fonction logistique : $\tau \sim \log n$

$$n \times \alpha \frac{m}{n}(1-m/n) \rightsquigarrow \frac{dm}{dt} = \alpha \underbrace{\frac{m}{n}}_{\text{probabilité de choisir meilleur individu}} \underbrace{\left(1-\frac{m}{n}\right)}_{\text{probabilité qu'il y ait encore de la place pour augmenter le nombre de copies}}$$

nb repetitions

Cette équation peut étre trouvée avec la sélection proportionel et le tournois

• Sélection par tournois :

La probabilité de ne pas choisir le meilleur individu : $1-m/n$

La probabilité de ne pas avoir le meilleur individu dans un tournois de taille k : $(1-m/n)^k$

=> Probabilité que le meilleur individu soit présent dans le tournois : $1-(1-m/n)^k$

Donc $m(t+1) = n\left(1-(1-m/n)^k\right) \geq n\left(1-(1-m/n)^2\right)$ $k \geq 2$ (slowest selection) $\quad (m/n)^k \leq (m/n)^2$, $k \geq 2$

$$= n\left[1-1+2m/n - m^2/n^2\right]$$

$$= 2m + m^2/n$$

Avec n le nombre de tournois, on a :

$$m(t+1) \geq m(t) + \frac{m^2(t)}{n} = m(t)\left[1-\frac{m}{n}\right] \rightsquigarrow \frac{dm}{dt} = m(1-m/n) \text{ with } \alpha = 1$$

$\Rightarrow \tau = \ln(n-1)/\alpha$, τ max quand $\alpha = 1$ que $\alpha = \Delta/f_1 < 1$

Plus lent que proportionnelle fitness même si les deux sont $O(\log n)$

21- Genetic Programming: goal, fitness evaluation, function and terminal Sets

-on applique le principe de l'évolution génétique avec des programmes informatiques
-on veux trouver un programe qui resolve un probléme donné
-on a une population de programmes que l'on va selectioner ,croisser et muter.
-on a un training sur lequel les input output souhaité sont connu
-language robuste aux languages génétiques 2choix : en arbre ou en stack
-qualité determiné sur un ensemble d'aprentisages

$A = \{(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)\}$ training set

Alors, la fitness f est $f(p) = \sum_{i=1}^{k} (p(x_i) - y_i)^2$ qu'on souhaite minimiser

Si on calcule la fitness comme le nombre de réponses correctes dans l'ensemble A, alors on maximise.

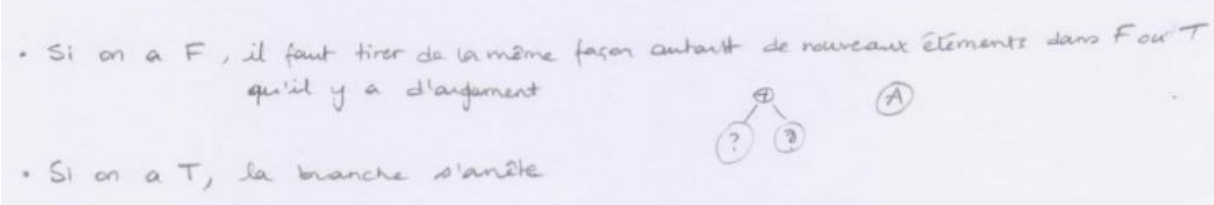
Function set et terminal set : L'espace de programmes qu'on peut construire est spécifié par deux ensembles:

- F : l'ensemble de noeuds fonctions ex: {+, −, ×, /}, {AND, OR, NOT}
- T : l'ensemble de noeuds terminaux (les feuilles : variables et constantes) ex : {A, B, C, D, 3.14, 2.71}

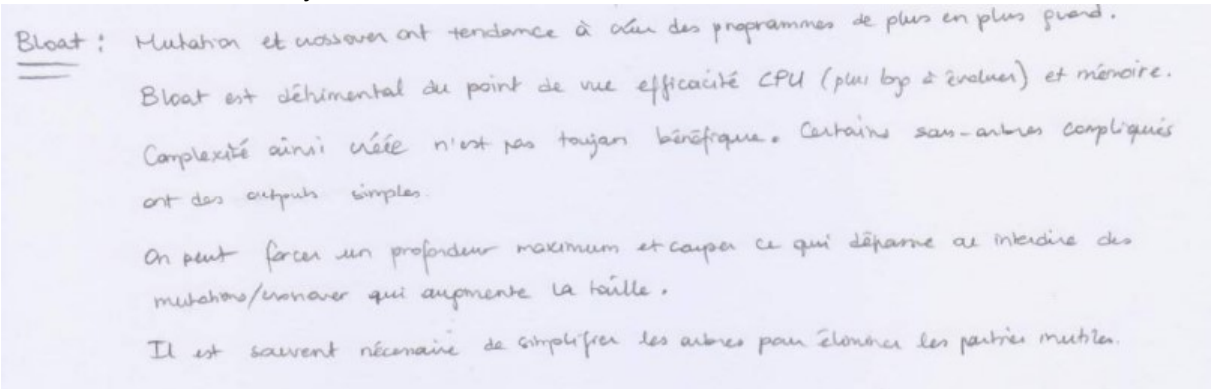
→ $A = \{\langle (A_1, B_1, C_1, D_1), y_1 \rangle, \langle (A_2, B_2, C_2, D_2), y_2 \rangle, \dots\}$

22- Genetic Programming: tree representation, initialization, crossover, mutation, and bloat.

-population initial choisie au hasard
-on choisie au hasar soit dans T soit dans F



-on tire un element F avec probabilité p et on tire un élément t avec une probabilité 1-p
-p affect la profondeur de l'arbre
-On peut faire dépendre p de la valeur dans laquelle on se trouve
-Crossover : on échange les sous arbres choisis au hasard dans chacun des parents
-mutation : on choisie un nœud au hasard et on reconstruit aléatoirement le sous-arbre qui y st attaché
-probabilité de séléction pour mutation peut étre choisie selon la profondeur du nœud pour eviter que les nœuds feille soient majoritairement choisie



23- Genetic Programming: the stack-based, instruction-driven representation.

24- Evolution Strategy: individual and population versions.

25- Performance of metaheuristics: examples, specificities of the performance evaluation, metrics, approach, "No Free Lunch" theorem.

-no exact,no guarantee quality of the solution

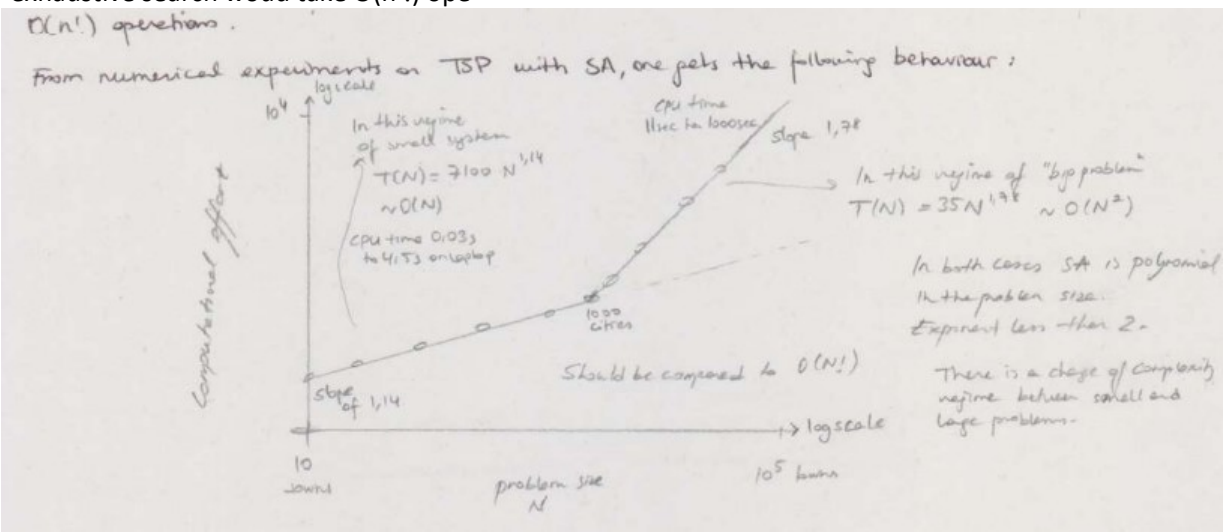
-stockastic and do not have twice the same behaviour

Ex :TSP with SA

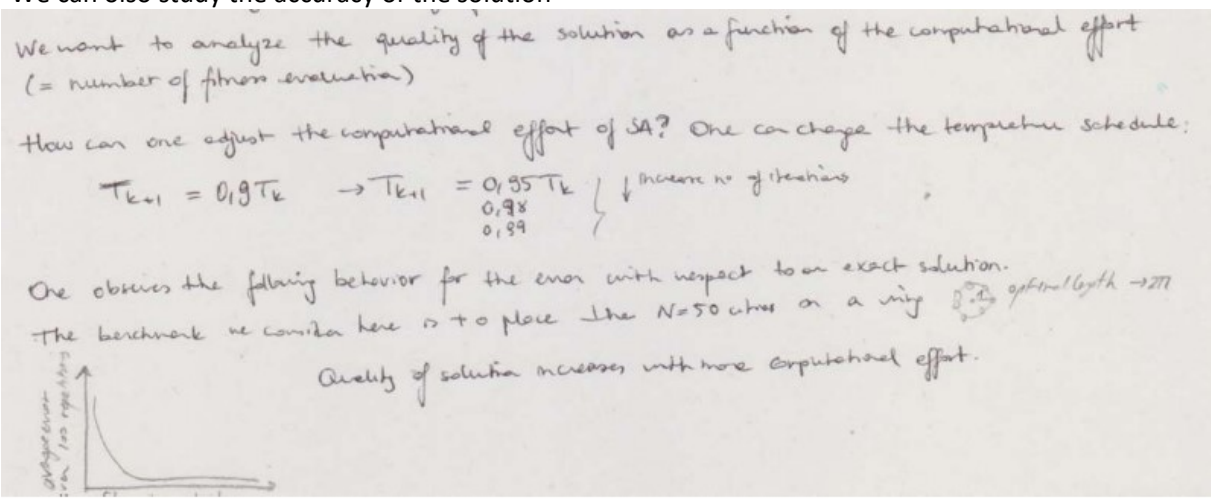
-fist question the computation with the problem size

-Sa give an answer after no improvement after 3 change of temperature

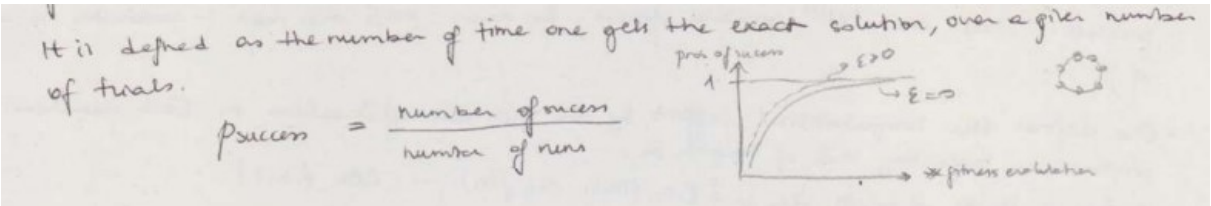
-exhaustive search woud take O(n !) opé



-We can olso study the accuracy of the solution



-the average the error we can olso be use the probability of success as a métric of the capability of SA to find the solution



Principle of evaluation :
-need a bechmark :real problem of synthetic one
-differents metheuristic extracted
-metrics :

Metrics : - Computational effort : CPU time, or preferably the work done : # fitness evaluation
- Quality of the solution : average error or a probability of success at a given accuracy
- Complexity measure : we hope that the metaheuristics is not exponentially slow.

Statistics : evaluation should be done on several repetition : $N \in [100, 1000]$

When comparing two metaheuristics, one needs statistical test to determine the relevance of the difference of performance.

No free lunch théorem :
Quel est la meilleur méthaeuristique ?
Peu on dire que un metharisque A étant meilleur que B dans un probléme veut dire que A est tout simplement meilleur ?

Le theoréme indique :
-Une methaeuristique ne peux pas étre meilleur qu'une autre dans tout les problémes possibles
-si les performance sont composé de toute les fitness function possible alors aucune méthaeuristique ne gagne
-si A est bon dans un probléme y il serra moin bon dans un probléme y

-Hypothéses
-on a une fitnes function f :S->y dans R mais y est finis

- One considers fitness function $f : S \mapsto Y \in \mathbb{R}$ but Y is finite too.
- All possible problems in S can be specified by selecting one fitness function among the possible. There are $|Y|^{|S|}$ possible fitness : for each $x \in S$ one need to associate a value of Y
- One defines the computational effort by the number of iterations m. Each metaheuristics produces a trajectory in S of length m.
It gives a series of points $d_m = \{(x_1, f(x_1)), (x_2, f(x_2)), \dots, (x_m, f(x_m))\}$
- Let $P(d_m \mid f, m, A)$ be the probability that d_m contains the optimal solution of f using metaheuristics A.

Then the theorem states that :

$$\sum_f P(d_m \mid f, m, A) = \sum_f P(d_m \mid f, m, B)$$

globally the performance of A and B are the same.

So in theory no metaheuristics can be better than any other, if all possible problems are equally likely.
But in practice, not all possible problems are equally likely, and we still may think that some metaheuristics is better for a class of problem.

26- Phase transition in optimization problems: problem description and properties.

27- Phase transition in optimization problems: the RWSAT algorithm and its behavior.

0.3. Metaheuristics

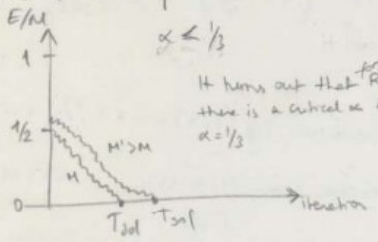
We consider a simple metaheuristics, called RWSAT for Random Walk SAT, to solve a k-XORSAT problem, and we will analyze its behaviour as a function of the difficulty parameter α. We will see that there are critical values of α (for abrupt change in behaviour). But these critical values are not α_c and α_d from theory. The metaheuristics fail much earlier.

RWSAT

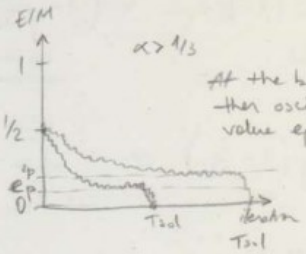
- Initialize all variables at random.
- Compute E (how many eq. are not satisfied)
- $t = 0$
- while (E>0 and $t < t_{max}$)
  - choose at random a non satisfied equation.
  - choose at random one of the 3 variable in this equation
  - change its value to its complement. → by this, change the equation is now satisfied.
  - compute E - $t = t+1$
- end while
- print E, t

30

This works well for small values of α:



It turns out that for RWSAT there is a critical α at $\alpha = 1/3$



At the beginning E/M drops but then oscillates around a plateau value ϵ_p.

After a while, there might be a ~~[illegible]~~ fluctuation which is large enough to take E/M to zero, which ends the process.

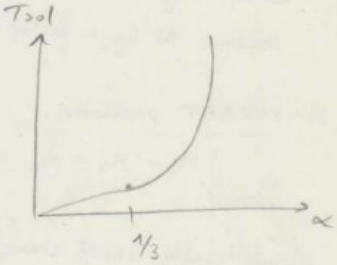
So we observe the following behavior:

- $\alpha < \alpha_E = 1/3$

  $\langle T_{sol} \rangle = N t_{sol} = O(N)$ → constant which increase with α.

  average value to find solution

  RWSAT has a linear time to solution



- $\alpha > \alpha_E = 1/3$

  $\langle T_{sol} \rangle = \exp(N t_{sol})$ → constant that increases with α.

  RWSAT is exponentially slow, no better than exhaustive search.

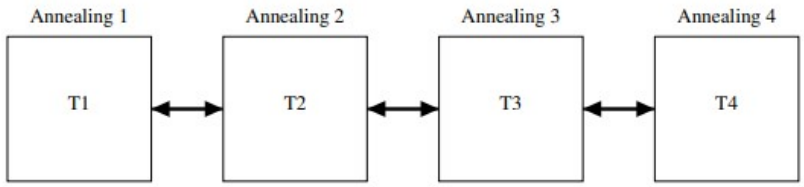
There is a phase transition in the complexity of RWSAT at $\alpha = \alpha_E$ (linear complexity to exponential complexity)

28- The parallel tempering.

-Parallel version of SA

-Come from physic

-we have several replicas of SA



-Each replica has a constant temperatur

-all run in parellel and interact by exchanging configuration

-temperature chosen as : t1<t2<t3<t4<…..<th

-neighborng sytems can xchange their current solution according to a probability law

-with 2 temperatures with 2 diffenrents solution c, we exchnge solution with probability :

$$p_{ij} = \min\left(1, e^{-\Delta_{ij}}\right)$$

where

$$\Delta_{ij} = (E_i - E_j)\left(\frac{1}{T_j} - \frac{1}{T_i}\right)$$

-both sytem se the same exchange proba

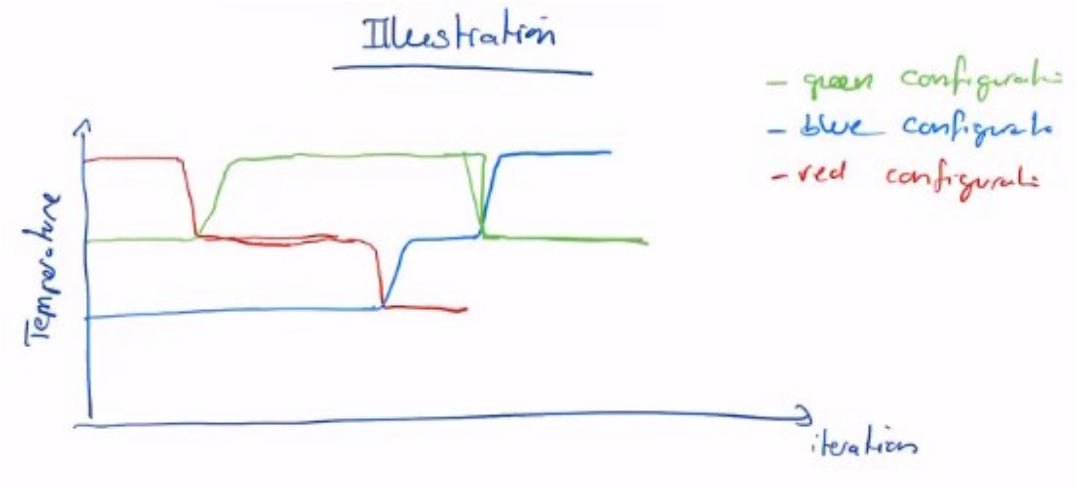
If $T_i < T_j$ and $E_i > E_j$, Δ_{ij} is negative and $p_{ij} = 1$

-means that good solution at higth T will always exchange with a less good solution at lower temperature.

-inverse olso possible

-the temp schedule is distributed acroos replicas instead of iteration

-higth temps explore other solutions and search space,low temps exploit one solution



Guiding parameters :
-How many sa ? usually M = sqrt(N) where N = problem size
-frequece of exchange config ? for instance when both systhem have reached a equilibrium
-range of temperature ?t1 small enough to allow convergence,tn big enouh to allow exploration

- These temperature can be fine tuned :
  - If the exchange rate is too large (> 2%)

    one increase the temperature, between all systems (difference)

    by $\Delta T = 0.1 (T_{i+1} - T_i)$
  - if the exchange rate is too low (less than $\frac{1}{2}$%)

    all ΔT are decreased, by the same rule.