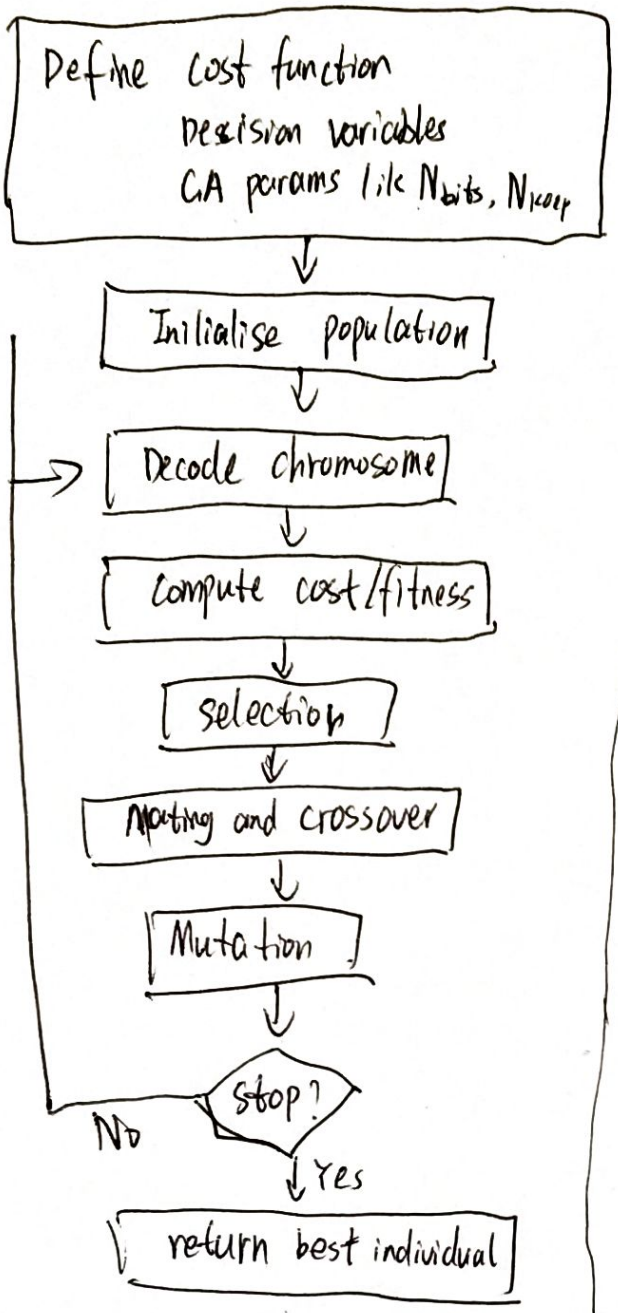
Binary Genetic Algorithm.



Binary encoding.

determine x_{lo}, x_{hi}, m, d.

$$\frac{x_{hi} - x_{lo}}{10^{-d}} \leq 2^m - 1$$

Binary decoding

$$x = x_{lo} + \text{decimal}(\text{gene}) \cdot \frac{x_{hi} - x_{lo}}{2^m - 1}$$

Selection.

determine who could reproduce and survive

Xrate approach.

Only the best N_{keep} individuals could survive

$N_{pop} * X_{rate}$

Thresholding approach.

Only the individuals whose cost lower than a user-defined threshold could survive.

If no survival, a new population generated.

Mating: select two chromosomes from mating pool to reproduce and crossover

if Xrate approach adopted, mating pool is a fixed size of N_{keep}, and $N_{pop} - N_{keep}$ offsprings generated.

① paring from top to bottom

② random pairing

③ Tournament Selection. (suitable for very large population)

1. Randomly sample n chromosomes from mating pool ($2 \leq n < N_{keep}$)
2. Chromosome in the sample with lowest cost selected.

④ weighted random pairing

- rank weighting

probability of chromosome i to be selected:

$$P_i = \frac{N_{keep} - i + 1}{\sum_{k=1}^{N_{keep}} k}$$

- cost weighting

probability of chromosome i to be selected:

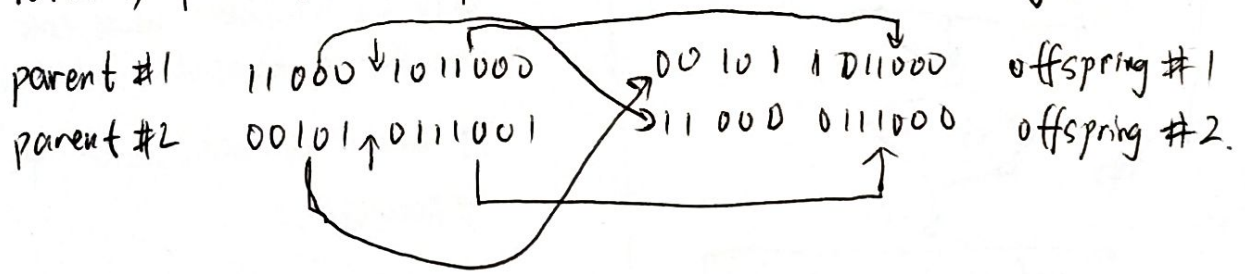
$$P_i = \left| \frac{C_i}{\sum_{m}^{N_{keep}} C_m} \right|$$

where

$$C_i = \begin{cases} C_i - C_{N_{keep}+1}, & N_{keep} > 0 \\ C_i - 2|C_{N_{pop}}|, & N_{keep} \geq 0 \end{cases}$$

Crossover : exchange information from two parents to create offsprings

randomly pick crossover points, and then exchange substrings



Mutation : Randomly alter a percentage of bits to introduce new information

N.B. Without mutation, GA could be failed to find global optima because search space is determined by initial population.

1. determine mutation rate μ.
2. calculate number of bits to alter

$$\#\text{ mutation bits} = \begin{cases} \mu \cdot N_{pop} \cdot N_{bits}, & \text{(elitism NOT implemented)} \\ \mu \cdot (N_{pop} - 1) \cdot N_{bits}, & \text{(elitism implemented)} \end{cases}$$

3. ~~Rand~~ Randomly choose #mutation bits bits and flip them.

Max percentage ~~(probability)~~ of possible solutions searched after k iteration:

$$P = \frac{N_{pop} \cdot N_{pop} \cdot k}{2^{N_{bits}}}$$

→ each iteration generate N_{pop}^2 chromosomes

→ $2^{N_{bits}}$ possible solutions in total.

Continous Genetic Algorithm.

- No encoding and decoding
- Genes in Chromosomes are real numbers instedd of binary strings.
- selection is the same as that of BGA.

BUT.

Crossover:

method ① swapping : same as that of BGA

② Blending

parent #1 $[P_{a1}, P_{a2}, \cdots, P_{an}]$
parent #2 $[P_{b1}, P_{b2}, \cdots, P_{bn}]$
offspring #1 $[P_{a'1}, P_{a'2}, \cdots, P_{a'n}]$
offspring #2 $[P_{b'1}, P_{b'2} \cdots, P_{b'n}]$

$$P_{a'i} = \beta P_{ai} + (1-\beta) P_{bi}$$
$$P_{b'i} = (1-\beta) P_{ai} + \beta P_{bi} \qquad (0 < \beta < 1)$$

where β is a user-defined param.

③ Extrapolation

$$P_{di} = P_{ai} - \beta (P_{ai} - P_{bi})$$
$$P_{b'i} = P_{bi} + \beta (P_{oi} - P_{bi}) \qquad (\beta \geq 0)$$

Note: Extrapolation may cause Genes outside allowed range, discard that offspring and run again.

Mutation:

1. calculat the number of genes (decision variables) to be altered.

$$\#\text{mutation} = \begin{cases} \mu \cdot N_{pop} \cdot N_{var} \\ \mu \cdot (N_{pop} - 1) \cdot N_{var} \quad \text{(elitism)} \end{cases}$$

2. randomly choose #mutation variables, and alter them as:

$$P'_i = P_i + \sigma N(0,1)$$

where σ is a user-defined param, and $N(0,1)$ is standard Goussian noise.

Advanced Topics of GA

① Expensive Cost Function: GA become time-consuming when computing complex cost function.

principle: try as little as possible to compute.

② Multiple Objective Optimisation

problem: multiple objectives to concern, i.e. multiple cost functions and objectives may conflict with one another.

possible approach: Aggregate them into one cost function with associated weights:

$$F(x) = \sum_{i=1}^{k} w_i f_i(x) \quad \text{where } w_i > 0 \ \& \ \sum_{i=1}^{k} w_i = 1$$

③ Gray Code

problem: Traditional binary representation has huge nonlinearity, which may cause many problem, typically reducing solution quality after crossover.

Example: $N_{bits} = 4$, optima = 6 or 9.

par #1 $0\downarrow111 = 7$ — crossover → offsp#1 $0\downarrow000 = 0$

par#2 $1\uparrow000 = 8$ offsp#2 $1\uparrow111 = 16$

both offsprings getting away from best solution

possible approach: use Gray code whose hamming distance of two adjacent number is only one.

binary → Gray:

Set binary code : $b_n b_{n-1} \cdots b_1$

Gray code : $g_n g_{n-1} \cdots g_1$

we have $g_i = \begin{cases} b_n, & i = n \\ b_i \text{ XOR } b_{i+1}, & 1 \le i \le n-1 \end{cases}$

par#1 $0\downarrow100 = 7$ — crossover → offsp#1 $0\downarrow100 = 7$

par#2 $1\uparrow100 = 8$ offsp#2 $1\uparrow100 = 8$

④ permutation

problem: optimisation that require to put things in right order.

e.g. Travel salesman problem.

possible approaches:

I. revised crossover.

1. partially matched crossover
2. Ordered crossover
3. cycled crossover
4. coding with referen list

II. revised mutation.

1. Inversion
2. Insertion and displacement
3. reciprocal exchange

See codelist.py

⑤ Penalty function. — A method to handle constraint.

$$f_p(x) = f(x) + \sum_{i=1}^{m} C_i(x)$$

usually, $C_i(x) = \begin{cases} \text{big number} & \text{, if constraint violated} \\ 0 & \text{, if constraint hold} \end{cases}$

So that cost function gets a high value when constraint being violated.

⑥ Genetic programming — evolutionary approach to come up with efficient computer programe for solving computation problem.

basic idea: In computer Science, a program is proved to be equivalent to a function, i.e. a combination of oprends (data structure) and operators (algorithm)

principle: for a problem, define a set of variables and operators. represent them as an expression tree. apply selection, crossove, mutation to those trees.

⑦ why GA works — Schema Theorem.

According to Schema Theorem, above-average schemata receive exponetially increasing trails in subsequence generations, yet below average schemata will die out.

Proof:

Set $f(s_i)$ as the cost of string s_i (chromosome)

$f(S,t)$ as the average cost of all string matched by schema S at iteration t.

$F(t)$ as sum of cost of the whole population

$$F(t) = \sum_{i=1}^{Npop} f(s_i)$$

$F(S,t)$ as sum of cost of all string matched by Schema S at iteration t.

then, we have probability of string s_i being selected:

$$P(s_i) = \frac{f(s_i)}{F(t)}$$

probability of a selected string matched by schema S:

$$P(S) = \frac{F(S,t)}{F(t)}$$

Set $\xi(S,t)$ as the number of string in the population matched by schema S at t iteration.

then, we have

$$\xi(S,t+1) = P(S) \cdot Npop$$

$$= \frac{F(S,t)}{F(t)} \cdot Npop$$

$$= \frac{f(S,t) \cdot \xi(S,t)}{F(t)} \cdot Npop$$

$$= \xi(S,t) \cdot \frac{f(S,t)}{F(t)/Npop} = \xi(S,t) \cdot \frac{f(S,t)}{\bar{F}(t)}$$

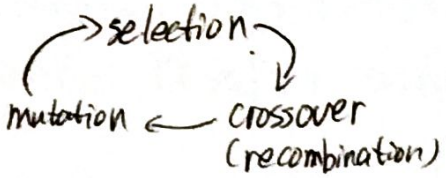
if $\frac{f(S,t)}{\bar{F}(t)} > 1$, meaning that schema S have above-average performance then $\xi(S,t+1) > \xi(S,t)$, meaning that number of string matched by S will increase in the subsequent generation.

Q.E.D.

Evolution Strategy

A technique of evolutionary computation like GA.

Basic components:



Individuals are realed vectors $\vec{x} = [x_1, x_2, \cdots, x_{n_x}]$, associated with a strategy parameter $\vec{\sigma} = [\sigma_1, \sigma_2, \cdots, \sigma_{n_x}]$

Selection.

- plus strategy $(\mu+\lambda)$-ES.

  Generate λ offspring from μ parents, the best μ individuals from parents and offsprings survive to the next generation.

  * Elitism Implemented !

- Comma strategy (μ, λ)-ES

  Generate λ offspring from μ parents, the best μ individuls from ONLY offsprings survive to the next generation.

  * Elitism NOT Implemented !

Crossover (5 approaches)

① No recombination: selected parent directly become offspring.

② Load, discrete crossover: randomly mix elements of ~~$\vec{x}$ and $\vec{\sigma}$~~ 2 parents.

$$x_j(t) = \begin{cases} x_{1j}(t) & \text{if } r_j \le 0.5 \\ x_{2j}(t) & \end{cases}$$

$$\sigma_j(t) = \begin{cases} \sigma_{1j}(t) \\ \sigma_{2j}(t) \end{cases}, \quad \text{if } r_j \le 0.5$$

where $r_j \in [0, 1]$ is random variable

③ Local. Intermediate Crossover: average of two parents.

$$x_j(t) = r\,x_{1j}(t) + (1-r)\,x_{2j}(t)$$

$$\sigma_j(t) = r\,\sigma_{1j}(t) + (1-r)\,\sigma_{2j}(t)$$

where $r \in [0, 1]$ is random number.

④ ~~Global~~
Global, discrete crossover: randomly mix elements of multiple parents.

$$x_j(t) = \begin{cases} x_{1j}(t) & , \text{if } r_j \le 0.5 \\ x_{s_j j}(t) \end{cases}$$

$$\sigma_j(t) = \begin{cases} \sigma_{1j}(t) & , \text{if } r_j \le 0.5 \\ \sigma_{s_j j}(t) \end{cases}$$

where $r_j \in [0, 1]$ a random number, and

$s_j \in [1, N_P]$, a random number.
↳ number of parents selected to mate

⑤ Global, indermediat crossover: average of multiple parents

$$x_j(t) = \frac{1}{N_P} \sum_{k=1}^{N_P} x_{kj}(t)$$

$$\sigma_j(t) = \frac{1}{N_P} \sum_{k=1}^{N_P} \sigma_{kj}(t)$$

Mutation.

offspring mutation: $x'_j(t) = x_j(t) + \sigma_j(t) N_j(0,1)$

where $N_j(0,1)$ is standard Gaussian noise

strategy parameter mutation:

$$\sigma'_j(t) = \sigma_j(t) \qquad \text{(Non-adaptive)}$$

OR $$\sigma'_j(t) = \begin{cases} \sigma_j(t) \cdot e^{\frac{1}{3}} & , \text{relative frequency of successful mutation of certain period over } \frac{1}{5} \\ \sigma_j(t) / (e^{\frac{1}{3}})^{\frac{1}{4}} \end{cases}$$

OR when $t > 10 n_x$

$$\sigma'_j(t) = \begin{cases} \alpha\,\sigma_j(t) & , n_m < 2n_x \\ \sigma_j(t)/\alpha & , n_m > 2n_x \\ \sigma_j(t) & , n_m = 2n_x. \end{cases}$$

where n_m is number of successful mutation, $\alpha > 0$

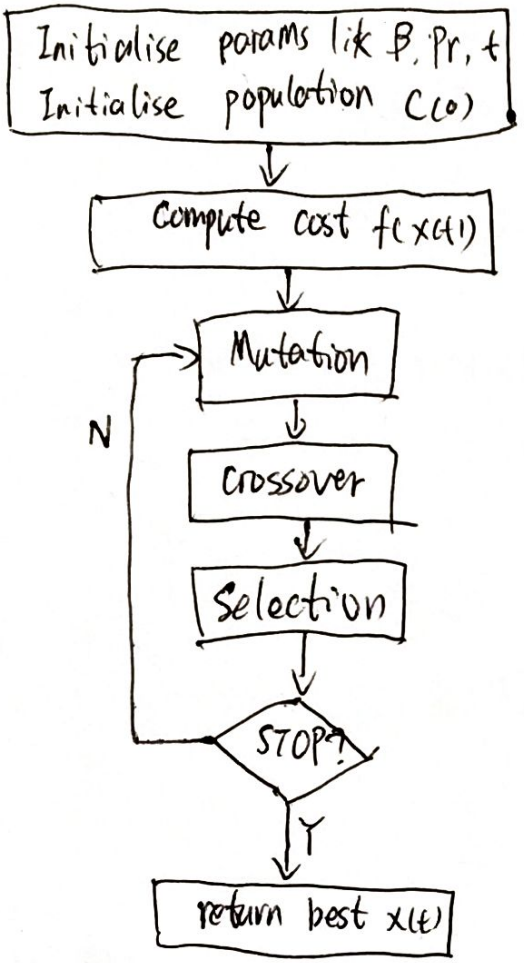
Note: only mutate after $10 n_x$ iteration, and at every n_x iteration, otherwis not mutate

Defferential Evolution.

A stochastic, population-based search strategy, originally developed for contiuous-valued ~~prob~~ problem (different GA).

The biggest characteristic : use that distence and direction information to guide search.

Basic Process



General psudo code

```
Initialize params lik β, Pr.
Initialise population C(0).
While not STOP do.
   for each individual x⃗_i(t) ∈ C(t) do.
      compute cost f(x⃗_i(t))
      create trial vector u⃗_i(t)  (mutation)
      create offspring x⃗'_i(t)    (crossover)
      if f(x⃗'_i(t)) < f(x⃗_i(t)) then.
         add x⃗'_i(t) to C(t+1)
      else
         add x⃗_i(t) to C(t+1)
      end if
   end for
   t ← t+1
end while
return x⃗(t) with lowest cost.
```

trial vector $\vec{u}_i(t)$ = target vector + $\beta \Sigma$ (difference vector)

where β is a user-defined param

and difference vector means the ~~diff~~ subtraction of two randomly picked individuals.

offspring vector $\vec{x}'_i(t) = [x'_{i1}(t), \cdots, x'_{in}(t)]$

$$x'_{ij}(t) = \begin{cases} u_{ij}(t), & \text{if } j \in J \\ x_{ij}(t) \end{cases}$$

where $J = \begin{cases} \text{randomly sampled indices from } (1,2,\cdots,n) \\ \text{e.g. } \{1, 2, 5, 9\} \\ \text{randonly selected indices sequence} \\ \text{e.g. } \{2, 3, 4, 5\}. \end{cases}$

Various DE versions: DE/x/y/z.

x: ways to choose target vector

y: number of difference vectors

z: ways to crossover (binominol or exponetial)

DE/rand/1/z.

target vector $\vec{x}_{i1}(t)$: randomly chosen.

trail vector $\vec{u}_i(t) = \underbrace{\vec{x}_{i1}(t)}_{\text{target vector}} + \beta(\underbrace{\vec{x}_{i2}(t) - \vec{x}_{i3}(t)}_{\text{difference vector}})$

DE/best/1/z.

$x_{i1}(t)$: best individual $\vec{\hat{x}}(t)$ in current population $C(t)$

$\vec{u}_i(t) = \vec{\hat{x}}(t) + \beta(\vec{x}_{i2}(t) - \vec{x}_{i3}(t))$

DE/α/nv/z.

$\vec{x}_{i1}(t)$: randomly chosen

$\vec{u}_i(t) = \vec{x}_{i1}(t) + \beta \sum_{k=1}^{n_v} (\vec{x}_{i2}^{k}(t) - \vec{x}_{i3}^{k}(t))$

~~DE/rand/nv/z.~~

DE/rand-to-best/nv/z.

$\vec{x}_{i1}(t)$: randomly chosen. $\vec{\hat{x}}(t)$: best individual

$\vec{u}_i(t) = r\vec{\hat{x}}(t) + (1-r)\vec{x}_{i1}(t) + \beta \sum_{k=1}^{n_v} (\vec{x}_{i2}^{k}(t) - \vec{x}_{i3}^{k}(t))$

$r \in [0, 1]$ user-defined param.

DE/current-to-best/1+nv/z.

$\vec{x}_{i1}(t)$: parent vector $\vec{x}_i(t)$

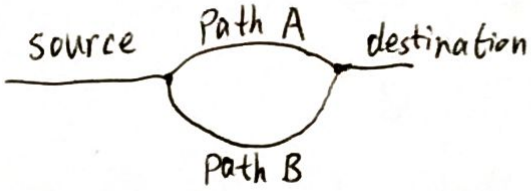
$\vec{u}_i = \vec{x}_i(t) + \beta(\vec{\hat{x}}(t) - \vec{x}_i(t)) + \beta \sum_{k=1}^{n_v} (\vec{x}_{i2}^{k}(t) - \vec{x}_{i3}^{k}(t))$

The Rest Are NOT Important.

Ant Colony Optimisation

suitable for : routing and other graph-related optimisation problems.

Binary Bridge Experiment.



Q: Which path will an ant choose at the next iteration?

A: The ant tend to choose a path with more pheromone, which means it will choose according to the probability:

$$P_A(t+1) = \frac{(c+n_A(t))^\alpha}{(c+n_A(t))^\alpha + (c+n_B(t))^\alpha}$$

$$P_B(t+1) = \frac{(c+n_B(t))^\alpha}{(c+n_A(t))^\alpha + (c+n_B(t))^\alpha}$$

where

$n_A(t)$ and $n_B(t)$: amount of pheromone on path A and B at the t-th iteration

c: user-defined. weight of exploration

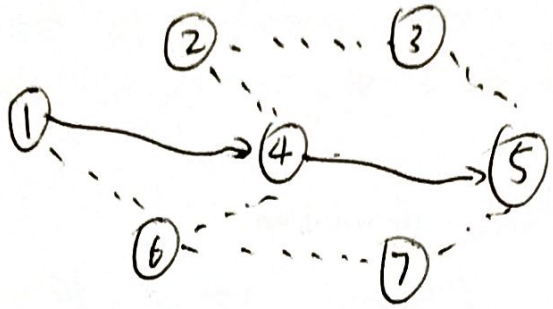
α: user-defined. weight of exploitation.

Characteristics of social colonies:

- Flexible
- Robast
- Decentrised
- Self-Organisation
  - positive feedback
  - nagative feedback
  - amplification of fluctuation
  - multiple interaction.

Ant colony for shortest path problem.

Given a graph



Q: How to figure out the shortest path from the souree node to the destination node?

A: Basic idea: ∵ ants left pheromone on path while walking.
the shorter a path, the more pheromone.
ants tend to choose paths with more pheromone.
∴ the shortest path have most pheromone, attract most ants.

~~Simple Ant Colony Optimization~~

Set $1, 2, \cdots, n$ as ants

$x^k(t)$ as the path of ant k from src to dest at t.

$\langle i, j \rangle$ as edge from node i to j.

$f(x)$ as the cost of path x.

$\tau_{ij}(t)$ as amount of pheromone on edge$\langle i, j \rangle$ at t.

AC ~~algo~~ psudocode.

```
Initialise τij(0), deterime params like ρ, Q.
while not STOP do
    for each ant k = 1, ..., n, do
        x^k(t) = { }    // start from src node
        while dest not reached do
            compute transition probability p_ij^k(t)
            add <i,j> to x^k(t) according to p_ij^k(t) and transition rule
        end while
        remove loops from x^k(t)
        compute f(x^k(t))
    end for
    for each edge <i,j> in graph do.
        evaporate pheromone
```

end for
for each edge $\langle i,j \rangle$ in graph do.
update pheromone amont.
end for
$t \leftarrow t+1$
end while
return $x^k(t)$ with lowest cost $f(x^k(t))$

Note: 1. Transition rule.
2. pheromone evaporation rule
3. Pheromone update rule
are the 3 main component of various Ant Colony algorithms

Simple Ant Colony Optimisation

Transition rule.

$$P_{ij}^k(t) = \begin{cases} \dfrac{\tau_{ij}(t)}{\sum_{u \in N_i^k(t)} \tau_{iu}(t)} & \text{, if } j \in N_i^k(t) \\ 0 \end{cases}$$

where $N_i^k(t)$ are all reachable nodes from i

Pheromone evaporation.

$$\tau_{ij}(t) \leftarrow (1-\rho)\,\tau_{ij}(t)$$

where $\rho \in [0,1]$ is a user-defined param.

phero mone update.

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

$$\text{wher } \Delta \tau_{ij}{}^k(t) = \begin{cases} \dfrac{Q}{f(x^k(t))} & \text{, if } \langle i,j \rangle \text{ in } x^k(t) \\ 0 & \text{, otherwise.} \end{cases}$$

Ant System.

Transition probability

$$P_{ij}^{k}(t) = \begin{cases} \dfrac{\tau_{ij}^{\alpha}(t) \cdot \eta_{ij}^{\beta}(t)}{\sum_{u \in N_i^k(t)} \tau_{ij}^{\alpha}(t) \cdot \eta_{ij}^{\beta}(t)} & , \text{if } \cancel{\langle i,j \rangle \text{ in } x^k(t)} \; j \in N_i^k(t) \\ 0 \end{cases}$$

where $\eta_{ij}^{k}(t) = \dfrac{1}{d_{ij}(t)}$

OR.

$$P_{ij}^{k}(t) = \begin{cases} \dfrac{\alpha \tau_{ij}(t) + (1-\alpha)\eta_{ij}(t)}{\sum_{u \in N_i^k(t)} \alpha \tau_{ij}(t) + (1+\alpha)\eta_{iu}(t)} & , \text{if } j \in N_i^{k}(t) \\ 0 \end{cases}$$

α, β are user-defined params

pheromone update rule.

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{k=1}^{m} \tau_{ij}^{k}(t)$$

where

$$\Delta \tau_{ij}^{k}(t) = \begin{cases} \dfrac{Q}{f(x^k(t))} & , \text{if } \langle i,j \rangle \text{ in } x^k(t) \\ 0 \end{cases}$$

OR

$$\Delta \tau_{ij}^{k}(t) = \begin{cases} Q & , \text{if } \langle i,j \rangle \text{ in } x^k(t) \\ 0 \end{cases}$$

OR

$$\Delta \tau_{ij}^{k}(t) = \begin{cases} \dfrac{Q}{d_{ij}(t)} & , \text{if } \langle i,j \rangle \text{ in } x^k(t) \\ 0 \end{cases}$$

Ant Colony System

Transition rule:

$$j = \begin{cases} \underset{j}{\operatorname{argmax}} \{ \tau_{ij}(t)\, \eta_{iu}^{\beta}(t) \}, & \text{if } r \le r_0 \\ J, & \text{if } r > r_0 \end{cases}$$

where $r \in [0, 1]$ is a random number

$r_0 \in [0, 1]$ is a user-defined param

and $J \in N_i^k(t)$ is randomly selected based on $P_{ij}^k(t)$

$$P_{ij}^k(t) = \begin{cases} \dfrac{\tau_{ij}(t)\, \eta^{\beta}(t)}{\sum_{u \in N_i^k(t)} \tau_{ij}(t)\, \eta_{iu}^{\beta}(t)}, & \text{if } J \in N_i^k(t) \\ 0 \end{cases}$$

Evaporation rule

$$\tau_{ij}(t) \leftarrow (1 - P_L)\, \tau_{ij}(t) + P_L\, \tau_0$$

where $P_L \in (0, 1)$ a user-defined param

and τ_0 is a sure defined small constant.

pheromone update rule.

$$\tau_{ij}(t+1) = (1 - P_a)\, \tau_{ij}(t) + P_a\, \Delta \tau_{ij}(t)$$

where

$$\Delta_{ij}(t) = \begin{cases} \dfrac{1}{f(x^+(t))}, & \text{if } \langle i, j \rangle \text{ in } x^+(t) \\ 0 \end{cases}$$

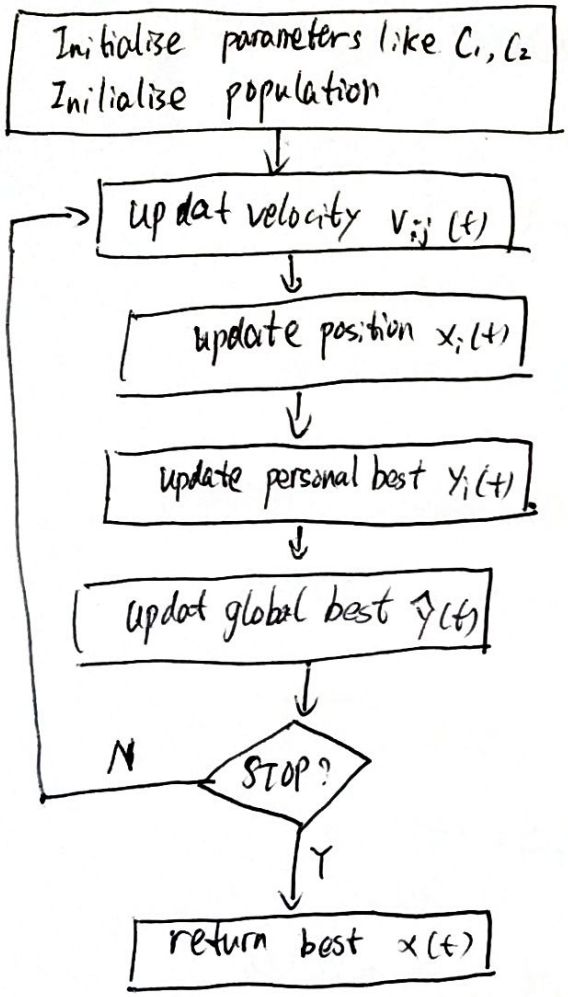
where $x^+(t)$ is the best solution at t iteration

Particle Swarm Optimisation

Analogy: Suppose a man was going to London New Year Firework, ~~he~~ didn't know the exact spot. He followed the crowd, and adjusted speed and direction according to people around him. Finally, he, as well as other people, arrived at destination.

Three principle to obey: Seperation, Alignment, Cohesion

Basic Process



General Psudocode

```
Initialise x_i(0), y_i(0), ŷ(0)
set t=0, v_i(0)=0, determine C_1, C_2.
while not STOP do
    for each particle i=1, 2, ..., n do
        update velocity v_ij(t+1)
        update position x_i(t+1)
    end for
    for each particle i=1, 2, ..., n do
        if f(x_i(t+1)) < f(y_i(t)) then
            y_i(t+1) = x_i(t+1)
        else
            y_i(t+1) = y_i(t)
        endif
        if f(y_i(t+1)) < f(ŷ(t)) then
            ŷ(t+1) = y_i(t+1)
        end if
    end for
    t ← t+1
end while
return best x(t)
```

Velocity update rule

$$v_{ij}(t+1) = \underbrace{v_{ij}(t)}_{\text{previous}} + \underbrace{c_1 r_{1j}(t)\left(y_{ij}(t) - x_{ij}(t)\right)}_{\text{self cognition}} + \underbrace{c_2 r_{2j}(t)\left(\hat{y}_j(t) - x_{ij}(t)\right)}_{\text{social information}}$$

where c_1, c_2 are user-defined parameters.

r_{1j}, r_{2j} are random numbers

particle update rule

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1)$$

personal best update rule.

$$y_i(t+1) = \begin{cases} y_i(t) \\ x_i(t+1) \end{cases}, \text{if } f(y_i(t)) < f(x_i(t+1))$$

global best update rule

$$\hat{y}(t+1) = \underset{y_i(t+1)}{\arg\min} \{ f(y_1(t+1)), \dots, f(y_n(t+1)) \}$$

The major difference between Local PSO and Global PSO is $\hat{y}(t)$

for Global PSO: $\hat{y}(t)$ is unique, selected from the whole population.

for Local PSO: $\hat{y}(t)$ is different for each particle, selected for neighbors of i.

The Rest Are NoT Important.

Stopping Criteria.

1. Swarm radius is small enough

$$R_{max}(t) = \max \| x_i(t) - \hat{y}(t) \| < \varepsilon$$

when ε is a user-defined small value

2. most particle clustered

$$\frac{|C|}{n} \geq \delta$$

where δ is a user-defined small value

and C is a set of particles.

---

for about σ times

$$\bar{x}(t) = \frac{\sum_{i}^{|C|} x_i(t)}{|C|}$$

for i in range(1, n)

if $\| x_i(t) - \bar{x}(t) \| \leq \varepsilon$

$C \leftarrow C \cup \{ x_i(t) \}$

end if

end for

end for

Velocity Explosion : similar to gradient explosion.

Velocity inorese too fast and become very large.

possible solutions.

- static approach : adjust velocity before updating position

$$V'_{ij}(t+1) = \begin{cases} V_{ij}(t+1) & , \text{if } V_{ij}(t+1) < V_{max,j} \\ V_{max,j} \end{cases}$$

where $V_{max,j}$ is user-defined for the i-th particle.

- dynamic approaches

$$V_{max,j}(t+1) = \begin{cases} r\, V_{max,j}(t) & , \text{if } f(\vec{y}(t)) \geq f(\vec{y}(t-t')) \\ V_{max,j}(t) \end{cases}$$

OR $$V_{max,j}(t+1) = \left(1 - \left(\frac{t}{n}\right)^{\alpha}\right) V_{max,j}(t)$$

OR $$V'_{ij}(t+1) = V_{max,j}(t+1) \tanh\left(\frac{V_{ij}(t+1)}{V_{max,j}(t+1)}\right)$$