Week 3-5

GAs\ES\DE with Real Decision Variables

▼ Floating Decision Variable Coding and Operators.

For some real optimization problems using binary coding is not natural. And another issue is computational complexity.

So it's better to use real number (integer or floating) for some certain problems. However, the operators of crossover and mutation are different

▼ Operators for real-coded problems

▼ Crossover

1-point crossover or 2-point crossover is changed as:

E.g. 1-point crossover, two parents are $X = [x_1, x_2, ...x_m] \, and \, Y = [y_1, y_2, ...y_m]$

The result of crossover is:

$X' = [x_1, x_2, ..., y_k, y_{k+1}, ..., y_m]$ and $Y' = [y_1, y_2, ..., x_k, x_{k+1}, ..., x_m]$

/# I think it maybe a little stupid...

Arithmetical Operator

$y_{new} = \lambda_1 y_{old} + \lambda_2 x_{old}$, and $x_{new} = \lambda_2 y_{old} + \lambda_1 x_{old}$

λ_1 and λ_2 are both manually set parameters.

Some specific situations can be identified:

- Convex crossover: $\lambda_1 + \lambda_2 = 1, \quad \lambda_1, \lambda_2 > 0$
- Average crossover: $\lambda_1 = \lambda_2 = \frac{1}{2}$
- Affine crossover: $\lambda_1 + \lambda_2 = 1$ and one of them is permitted to take negative values.
- Linear crossover: $\lambda_1 + \lambda_2 = c, \quad \lambda_1, \lambda_2 > 0$, and c is a positive constant, eg. c=2

This strategy is creative enough but there is a need to ensure the generated offspring are in the feasible region.

Direction Based Crossover: $x_{new} = x_{old} + r(y_{old} - x_{old}) \ \ if: f(y_{old}) \geq f(x_{old})$

r is a uniform random number in [0,1]

Some Other Operators

PCX – Parent Centric Crossover

SBX – Simulated Binary Crossover
BLX – Blend Crossover
SPX - Simplex Crossover
UNDX- unimodal Normal Distribution Crossover

\# These can be investigated if interested. Obviously I am not the interested student, neither are you.

▼ Mutation

Uniform mutation: Just generate a new value within the search range

Gaussian mutation: $x_{new} = x_{old} + N(o, \sigma)$, where $N(0, \sigma)$ is Gaussian distribution.

Arithmetic mutation: $x_{new} = x_{old} + r(b_{up} - x_{old})(1 - \frac{t}{T})^b$ or

$x_{new} = x_{old} - r(x_{old} - b_{low})(1 - \frac{t}{T})^b$

Two thoices are selected with equal probability.

Where r is uniform random number in [0,1], t is the current generation number, T is the maximum number of generations. b is the manually set parameter. b_{up} and b_{low} are the upper and lower bounds for this variable.

Direction Based Mutation

$x_{new} = x_{old} + rd$

r is a uniform random number in [0,1] and d is the fitness ascent direction, as same as the gradient of f(x). However, usually it is obtained by numerically estimated.

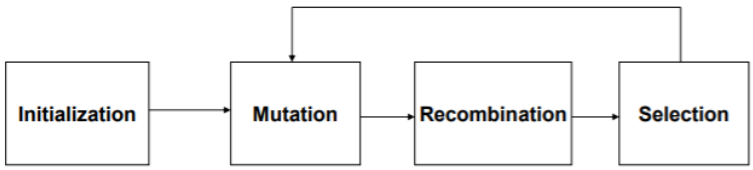
If the mutated offspring move out of the fesible range, choose a random direction d to instead.

▼ Evolution Strategy (ES)

\# Basically it's all unimportant nonsense, so I copied the slide directly.

Randomly generate an initial population of M solutions.
Compute the fitness values of these M solutions.
Use all initial vectors as parents to create M offspring solutions by Gaussian mutation i.e. Xnew=Xold+N(0, σ)
N(0, σ): normal distribution with zero mean variance σ
Calculate the fitness of M solutions and prune the population to M fittest solutions.
Go to the next step if the stopping condition is satisfied.
Otherwise, go to Step 2.
Choose the fittest one in the population in the last generation as the optimal solution.
ES would be more effective, if we are able to obtain good initial solutions instead of randomly generated initial solutions.
State of the art: Covariance Matrix Adapted ES – CMA-ES

▼ Differential Evolution (DE)



Basic steps of an Evolutionary Algorithm

▼ Initialization

$X_j = [x_{1,j}, x_{2,j} ..., x_{i,j}]$ is the j^{th} solution vector for i_{th} dimensionality, i.e. individual

$X = [X_1, X_2, ...X_j]^T$ is a set of solution vector, i.e. population.

In other word, j is the size of population and i is the number of decision variables.

j could be chosen between $5i$ to $10i$

Just randomly initialize the population in fesible region.

▼ Mutation

$$\vec{V}_{i,G} = \vec{X}_{r_1^i,G} + F \cdot (\vec{X}_{r_2^i,G} - \vec{X}_{r_3^i,G}).$$

$X_{r_1}, X_{r_2}, X_{r_3}$ are all randomly select from the population, G is the number of generation. F is a constant from (0, 1.5) in the traditional DE.

▼ Recombination

Uniform Crossover:

Let j_{rand} be a randomly chosen integer between 1,...,D.

$$u_{j,i,G} = \begin{cases} v_{j,i,G}, & \text{if } (rand_{i,j}[0,1) \le Cr \text{ or } j = j_{rand}) \\ x_{j,i,G}, & \text{otherwise,} \end{cases}$$

Cr is an algorithmic parameter to be set.

v and x here are parents.

Three possible trial/offspring/child solutions:

i) $\vec{U}_{i,G} = \vec{V}_{i,G}$ such that both the components of $\vec{U}_{i,G}$ are inherited from $\vec{V}_{i,G}$.

ii) $\vec{U}'_{i,G}$, in which the first component ($j = 1$) comes from $\vec{V}_{i,G}$ and the second one ($j = 2$) from $\vec{X}_{i,G}$.

iii) $\vec{U}''_{i,G}$, in which the first component ($j = 1$) comes from $\vec{X}_{i,G}$ and the second one ($j = 2$) from $\vec{V}_{i,G}$.

Low value of Cr is good for separable problems and high level of Cr is good for non-separable problems.

Exponential Crossover:

$$u_{j,i,G} = \begin{cases} v_{j,i,G}, & \text{for } j = \langle n \rangle_D, \langle n+1 \rangle_D, \ldots, \langle n+L-1 \rangle_D \\ x_{j,i,G}, & \text{for all other } j \in [1, D], \end{cases}$$

$<>_D$ denote a modulo function with modulus L

Pseudo-code for choosing *L*:

```
L = 0;
DO
{
   L = L+1;
} WHILE (((rand[0,1) ≤ Cr) AND (L < D)) ;
```

Eg. parents are :$X_{i,G} = [1,2,3,4,5]^T$, $V_{i,G} = [6,7,8,9,10]^T$

Suppose n=3 and L=3 for this specific example

$< n >_D, < n+1 >_D, \ldots, < n+L-1 >_D = < 3 >_3, < 4 >_3, < 5 >_3$ =0,1,2

when j=1 or j=2, the value of offspring comes from $V_{i,G}$, i.e. 6,7

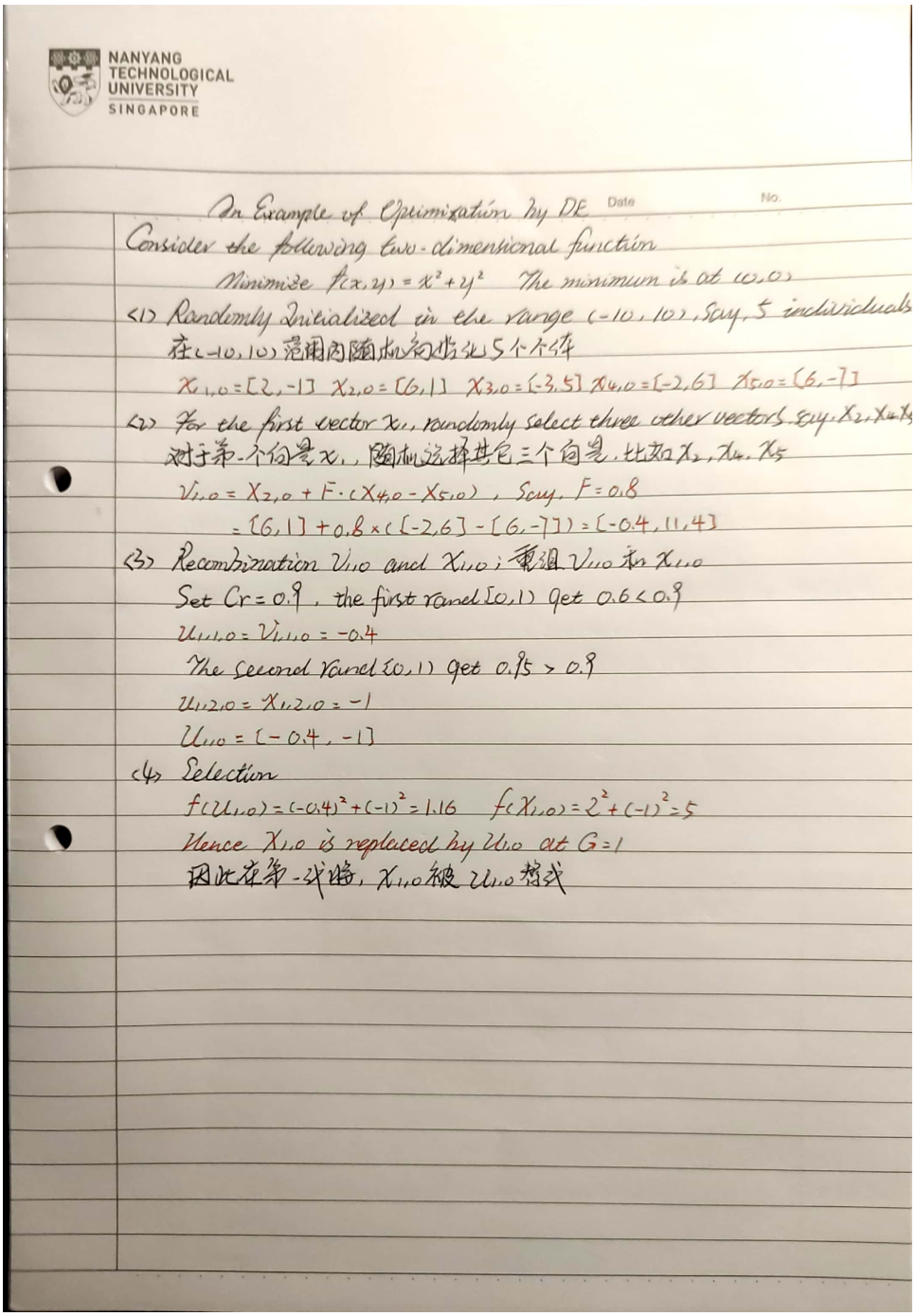
Otherwise, the value of offspring comes from $X_{i,G}$, i.e. 3,4,5

Thus the offspring is $u_{i,G} = [6, 7, 3, 4, 5]$

▼ **Selection**

$X_{i,G+1} = u_{i,G}\, if\, f(u_{i,G} \geq X_{i,G})\, Otherwise X_{i,G}$ (for maximum problem)

▼ **Example**



▼ **Improved mutation strategy**

"DE/rand/1": $\vec{V}_i(t) = \vec{X}_{r_1^i}(t) + F \cdot (\vec{X}_{r_2^i}(t) - \vec{X}_{r_3^i}(t))$.

"DE/best/1": $\vec{V}_i(t) = \vec{X}_{best}(t) + F.(\vec{X}_{r_1^i}(t) - \vec{X}_{r_2^i}(t))$.

"DE/current-to-best/1":

$$\vec{V}_i(t) = \vec{X}_i(t) + F.(\vec{X}_{best}(t) - \vec{X}_i(t)) + F.(\vec{X}_{r_1^i}(t) - \vec{X}_{r_2^i}(t)),$$

"DE/best/2": $\vec{V}_i(t) = \vec{X}_{best}(t) + F.(\vec{X}_{r_1^i}(t) - \vec{X}_{r_2^i}(t)) + F.(\vec{X}_{r_3^i}(t) - \vec{X}_{r_4^i}(t))$.

"DE/rand/2": $\vec{V}_i(t) = \vec{X}_{r_1^i}(t) + F_1.(\vec{X}_{r_2^i}(t) - \vec{X}_{r_3^i}(t)) + F_2.(\vec{X}_{r_4^i}(t) - \vec{X}_{r_5^i}(t))$.

The general convention used for naming the various mutation strategies is DE/x/y/z, where DE stands for Differential Evolution, x represents a string denoting the vector to be perturbed, y is the number of difference vectors considered for perturbation of x, and z stands for the type of crossover being used (exp: exponential; bin: binomial)
best: **The best solution in the current population.**

▼ Improved DE: Self-Adaptive DE (SaDE)

Mainly there are 3 parameters to be set: size of population, mutation probability and factor F

- Size of population is set by user
- A set of F are randomly generated from normal distribution $N(0.5, 0.3)$ and applied to each target vector in the current population
- A set of Cr are randomly generated from normal distribution $N(Cr_m, Std)$, Cr_m is initialized as 0.5
- SaDE gradually adjusts the range of Cr:
  - Initialized Cr by $N(0.5, 0.1)$
  - During LP number generations, record the actual Cr values generating better offspring.

- After LP number generations, update Cr_m by the average of all successful Cr values

▼ JADE

A. $V_{i,G} = X_{i,G} + F_i(X^p_{best,G} - X_{i,G}) + F_i(X_{r1,G} - X_{r2,G})$

$X^p_{best,G}$ is a randomly chosen vector from top p% individuals in current population

B. JADE can optionally make use of an external archive A, which stores the recently rejected inferior parents

$X^p_{best,G}$, $X_{i,G}$ and $X_{r1,G}$ are selected from current population but $X_{r2,G}$ is selected from $P \cup A$

C. For every generation, generate Cr with $N(\mu_{cr}, 0.1)$ and

update $\mu_{cr} = (1-c)\mu_{cr} + c.mean_A(S_{cr})$. S_{cr} is the set of all successful crossover Cr at generation G

For every generation, generate Cr with $C(\mu_F, 0.1)$ (Cauchy distribution) and

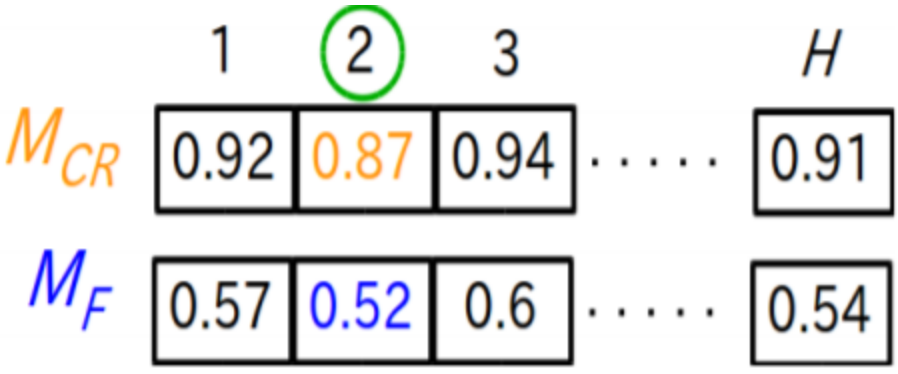
update $\mu_F = (1-c)\mu_F + c.mean_L(S_F)$. S_F is the set of all successful F at generation G and $mean_L$ is the Lehmer mean:

$mean_L(S_F) = \frac{\Sigma F^2}{\Sigma F}$

A rule of thumb: 1/c chosen from [5, 20] and p from [5%, 20%]

▼ Success-History based Adaptive DE (SHADE)

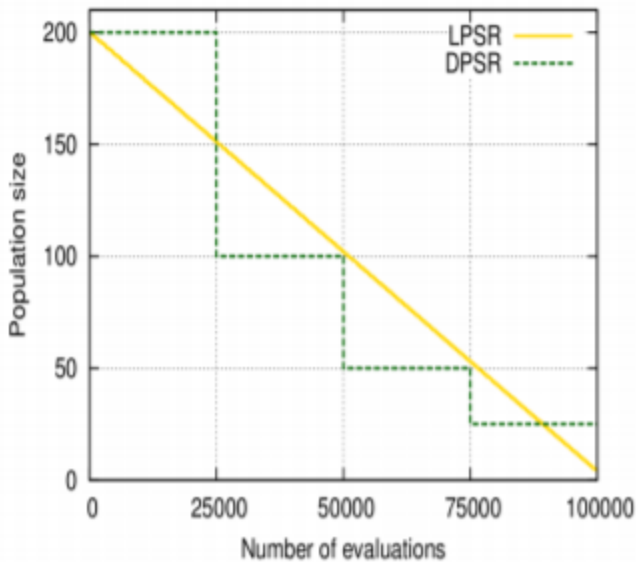
A historical memory M_{Cr} *and* M_F are used to store the successful parameters. And randomly select one pair of them:



Successful Cr and F will be recorded in the memory

At the end of generation, the contents of memory are updated by the mean values by S_{Cr} and S_F

▼ L-SHADE: SHADE with Linear Population Reduction



Simple Variable Population Sizing (SVPS) and Deterministic Population Size Reduction (DPSR)

Population size deceases as shown in the figure. **SVPS is more general.**

Partical Swarm Optimization (PSO)

▼ Variables in PSO

x-vector: the position of particle (individual) i.e. candidate solution vector

p-vector: the best x-vector so far

v-vector: direction the particle will move to

x-fitness: the fitness vector of x-vector

p-fitness: the fitness vector of p-vector

Update Equations:

For each decision variable:

$V_{new} = V_{old} + \varphi_1 * rand() * (p_{itself} - x_{old}) + \varphi_2 * rand() * (p_{global} - x_{old})$

$X_{new} = X_{old} + V$

φ_1 is called cognition component and φ_2 is called social component. Eiher of them can be set to 0. If both them are positive, the model is full model.

▼ Initialization

Randomly generate v-vector from $[-V_{max}, V_{max}]$.

Randomly generate x_vector in feasible region.

▼ Improvements on Velocities

PSO with Inertia Factor

$V_{new} = \omega * V_{old} + \varphi_1 * rand() * (p_{itself} - x_{old}) + \varphi_2 * rand() * (p_{global} - x_{old})$

Where ω is initialized to 1.0 and gradually reduced over generations.

PSO with the Constriction Coefficient

$V_{new} = k * [V_{old} + \varphi_1 * rand() * (p_{itself} - x_{old}) + \varphi_2 * rand() * (p_{global} - x_{old})]$

Where $\varphi = \varphi_1 + \varphi_2$, $\varphi > 4$ and $k = 2/|2 - \varphi - sqrt(\varphi^2 - 4\varphi)|$

▼ Particle Update Strategy

Synchronously\Asynchronously

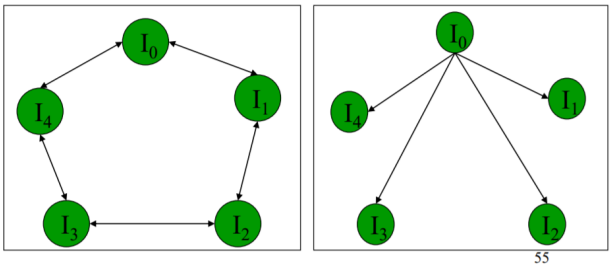
Global PSO: Adjust the particle's velocity according its personal best performance and the best performance achieved by all the particles (gbest).

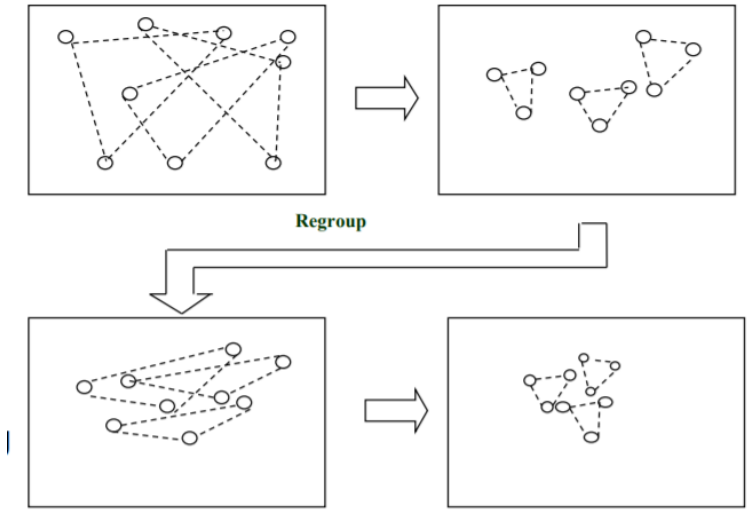
Local PSO: Adjust the particle's velocity according its personal best performance and the best performance achieved by its neighborhood (glocal).

▼ Dynamic multi-swarm PSO

In local PSO, there are several topologies. Two examples are

- Ring Topology (neighborhood of 3)
- Star Topology (global neighborhood)





1. Small sized swarms

2. Randomly re-group the swarm

▼ Comprehensive learning PSO (CLPSO)

$v_i^d = \omega * v_i^d + c * rand_i^d * (pbest_{f_i(d)}^d - x_i^d)$

$x_i^d += v_i^d$, $d = 1, 2, ..., D$ the number of decision variables.

$f_i = [f_i(1), f_i(2), ..., f_i(D)]$ denotes a set of particle indices with respect to each dimension of the particle i. i.e. pbest of different particles in all dimensions. It take the value i with probability Pc_i.

For each dimension of particle i, generate a random number, if the number is larger than Pc_i, the corresponding dimension of particle i will learn from its own

pbest, otherwise learn from another particle's pbest randomly selected or selected by tournament selection.

If selected by tournament selection, the size is usually 2.

We allow each particle to learn from its $pbest^d_{f(d)}$ until the particle stop to improve for a certain number of generations, called the refreshing gap m (usually 7 generations).

After that, we re-assign f_i for each particle i.

Difference:

Particle can learn from gbest of not only itself, but also other particles.

Different dimensions may learn from different particle's pbest.

Instead of learn from two exemplars (pbest and gbest), each dimension of particle only learn from one exemplar (pbest). It may reduce deficiency of the premature convergenece.

▼ Bandit Problem & Exploration and Exploitation (EE) problem.

https://blog.csdn.net/wangh0802/article/details/87913867?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-3.control&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-3.control

▼ Heterogeneous CLPSO (HCLPSO)

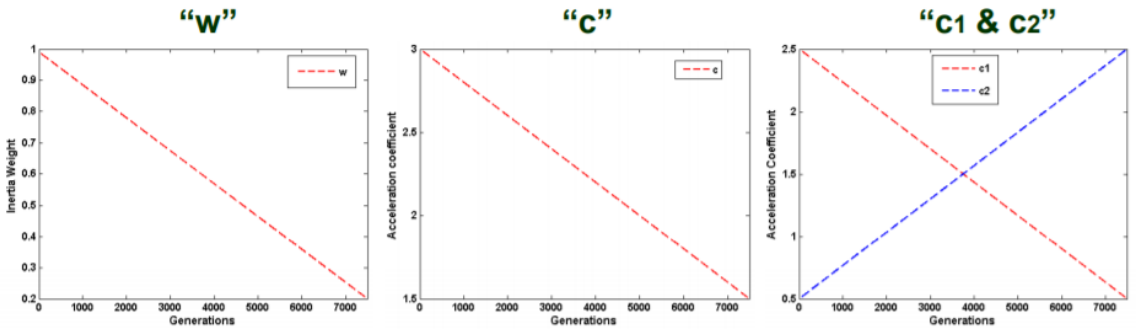
Particles are re-grouped into two sub populations:

1. Exploration group and 2. Exploitation group

For group 1: $v_i^d = \omega * v_i^d + c * rand_i^d * (pbest^d_{f_i(d)} - x_i^d)$

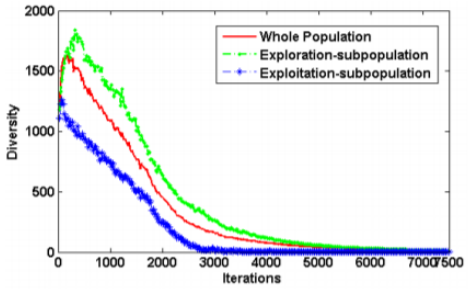
For group 2: $v_i^d = \omega * v_i^d + c_1 * rand_{1i}^d * (pbest^d_{f_i(d)} - x_i^d) + c_2 * rand_{2i}^d * (gbest^d - x_i^d)$

Adaptive Control Parameters



$$Diversity\,(S(t)) = \frac{1}{N}\sum_{i=1}^{N}\sqrt{\sum_{d=1}^{D}(X_i^d(t) - \overline{X^d(t)})^2}$$

$$\overline{X^d(t)} = \frac{\sum_{i=1}^{N} X_i^d(t)}{N}$$



- ✓ Exploration subpopulation g_1 : high diversity
- ✓ Exploitation subpopulation g_2 : low diversity
- ✓ The explorative particles are not allowed to access information from the exploitative particles.
- ✓ Thus, even if exploitation group suffers from premature convergence, exploration group has potential to rescue the exploitation-oriented group from the local optimum.
- ✓ Exploration and exploitation processes in HCLPSO are enhanced without one process adversely affecting the other.

66

Constraint Optimization

▼ Constraint Handling

The classical format of constraint problem:

Minimize $f(x), x = [x_1, x_2, ..., x_D]$

Subjected to $g_i(x) \leq 0, i = 1, ..., q$

and $h_j(x) = 0, j = q+1, ..., m$

For convenience, the equality constraint can be transformed into inequality form with ϵ by:

$|h_j(x)| - \epsilon \leq 0$

Consequently, the formula can be described as:

$$Minimize\, f(x), x = [x_1, x_2, ..., x_D]$$
$$Subjected\, to\, G_j(x) \leq 0,\ j = 1, ..., m$$
$$where\, G_{1,...,q} = g_{1,...,q}(x), G_{q+1,...,m} = |h_{q+1,...,m}(x) - \epsilon|$$

▼ Strategies in EA

Evolutionary algorithms (EAs) always perform unconstrained search. When solving constrained optimization problems, they require additional mechanisms to handle constraints.

The algorithm may yields infeasible offspring.

Rejecting Strategy

All infeasible solutions are discarded. Likely to be highly inefficient if a large number of
offpsirng are discarded. Also, if the feasible region is fragmented, then the method may not yield good results.

Repairing Strategy

Infeasible offsprings are corrected to make them satisfy all the constraints. Complex to implement.

Modifying Genetic Operators

Develop genetic operators to give only valid offsprings satisfying all constraints. Complex to implement.

Penalty Function Approach

$q(r, x) = f(x) + P(r, x)$, $P(r, x)$ here is the penalty function. It can be designed as:

$$P(r, x) = \Sigma_{j=1}^{m} r_j \left[max\left\{0, g_j\left(x\right)\right\}\right]^2$$

r_j is to be chosen, which can reflect the importance of the constraints. Usually set it to small values initially then gradually increase them. This way allow GA to search in both feasible and infeasible regions but give valid solution in the end.

Sometime the equation may be transformed as:

$$Q(r, x) = C - f(x) - P(r, x)$$

So that the value of the function keep positive. Here C is a value to be set.

Epsilon Constraint Method

Relaxation of constraints is controlled by ε parameter

High quality solutions for problems with equality constraints

$$\varepsilon(0) = \upsilon(X_\theta)$$

X_θ : top θ th individual in initial population (sorted w. r. t. υ)

$$\varepsilon(k) = \begin{cases} \varepsilon(0)\left(1 - \dfrac{k}{T_c}\right)^{cp}, & 0 < k < T_c \\ 0, & k \geq T_c \end{cases}$$

The recommended parameter settings are:

$$T_c \in [0.1T_{\max}, 0.8T_{\max}] \qquad cp \in [2, 10]$$

▼ Variable reduction strategy (VRS)

For example at first:

$$\begin{gathered} min\, x_1^2 + x_2^2 \\ Subjected\, to\, x_1 + x_2 = 2 \\ 0 \leq x_1 \leq 5,\ 0 \leq x_2 \leq 5 \end{gathered}$$

We can obtain the relationship between two decision variables: $x_2 = 2 - x_1$, then:

$$\begin{gathered} min\, 2x_1^2 - 4x_1 + 4 \\ Subjected\, to\ 0 \leq x_1 \leq 2 \end{gathered}$$

Some corresponding very advanced looking formulas:

Minimize $f(x), x = [x_1, x_2, ..., x_D]$

Subjected to $g_i(x) \leq 0, i = 1, ..., q$

and $h_j(x) = 0, j = q+1, ..., m$

Handle it with $|h_j(x)| - \epsilon \leq 0$

Assume Ω_j denotes the collection of variables involved in $h_j(X) = 0$

For $h_j(X) = 0$, if we can obtian a relationship:

$$x_k = R_{k,j}(\{x_l | l \in \Omega_j, l \neq k\})$$

Then we can eliminate x_k.

In summary, a constraint problem can be described as:

$$\begin{gathered} Minimize: f(X) \\ Subjected\,to: g_i(X) \leq 0,\ i = 1, ..., p \\ h_j(X) = 0,\ j = 1, ..., m \\ l_k \leq x_k \leq u_k,\ k = 1, ..., m \end{gathered}$$

And VRS can be implemented as:

$$\begin{gathered} Minimize: f(X) \\ Subjected\,to: g_i(X) \leq 0,\ i = 1, ..., p \\ h_j(X_k | k \in C) = 0,\ j \in M_2 \\ l_k \leq R_{k,j}(\{x_l | l \in \Omega_j, l \neq k\}) \leq u_k,\ k \in C_1, j \in M_1 \\ l_k \leq x_k \leq u_k,\ k \in C_2 \end{gathered}$$

Empirically, we can reduce in following situations:

- One variable less than or equal to second-order
- All linear equality constraints and variables
- Variables separately operated by one operator (e.g. $x^n, cosx, lnx, a^x$)