GAs/EAs/SAs with Real Decision Variables

- Real Decision Variable Coding
- Real Coded Operators for GAs
- Evolution Strategy (EA)
- Differential Evolution (EA)
- Particle Swarm Optimization (SA)

SA- Swarm Algorithm; GA- Genetic Algorithm
EA-Evolutionary Algorithm

Real Decision Coding

- Many real word optimization problems have real decision variables.
- It is not natural to use binary coding to solve those problems.
- Another issue is computational complexity – if a problem has for example 30 real valued variables and each requires 30 bits, then the solution string is 900 bits long!!
- If we employ real number coding, then there are just 30 floating numbers in a solution string.
- Recently numerous crossover/mutation operators have been developed for real coded GAs/EAs/SAs.

Operators Derived from SGA

- Many operators developed for binary solution representation in Simple Genetic Algorithm can be used in real coded GAs.
- For example, 1-point, 2-point and uniform crossover operators can be used with real coding.
- If two real-coded parent strings are:

$$x = [x_1 \quad x_2 \quad \ldots \quad x_m] \text{ and } y = [y_1 \quad y_2 \quad \ldots \quad y_m]$$

then the result of 1-point xover is:

$$\begin{aligned} x' &= [x_1 \quad x_2 \quad \ldots \quad x_k \quad y_{k+1} \quad \ldots \quad y_m] \\ y' &= [y_1 \quad y_2 \quad \ldots \quad y_k \quad x_{k+1} \quad \ldots \quad x_m] \end{aligned}$$

- Similarly, uniform and 2-point xover operations can be performed.

Mutation

- Mutation operator has to be changed, *i.e*. flipping between 0 and 1 is not useful.
- There are a few different mutation operators
  - Uniform mutation: this operator just replaces a decision variable in the solution string by a randomly generated value within the search range. This is the same as re-initializing the decision variable.
  - Gaussian mutation: perturbs the current value by a randomly generated number with a given variance and zero mean: e.g.: $x_{new}=x_{old}+N(0, \sigma)$
- There is no need to restrict the operators to be similar to those used in the binary Simple GA. For example, the crossover operator can be modified significantly.

Arithmetical Operators

- If *x* and *y* are two parents, then the xover is of the following form: $\lambda_1 x + \lambda_2 y$
- The 2 offspring are: *y’*= $\lambda_1 y + \lambda_2 x$ $x' = \lambda_1 x + \lambda_2 y$
- Depending on the conditions satisfied by λ_1 and λ_2, three different types can be identified:
  – Convex crossover when $\lambda_1 + \lambda_2 = 1, \quad \lambda_1, \lambda_2 > 0$
    - A special case is average crossover when $\lambda_1 = \lambda_2 = \frac{1}{2}$

Arithmetical Operators

- If λ_1 or λ_2 is permitted to take –ve values, while $\lambda_1 + \lambda_2=1$, then it is called *affine crossover*.
- If λ_1 and λ_2 are +ve, but $\lambda_1 + \lambda_2=c$, where *c* is +ve constant (eg. *c*=2), then it is called *linear crossover*.
- These crossover operators are capable of generating offspring within a substantial region of the solution space.
- Hence, there is a need to ensure that the generated offspring are in the feasible search region.

Arithmetic Mutation

- As opposed to uniform or Gaussian mutation, non-uniform mutation operator is defined as follows, when we mutate x_k in an offspring $x' = [x_1 \quad \dots \quad x'_k \quad \dots \quad x_m]$

where x'_k is the mutated value.

- x'_k is selected with equal probability between the following two choices:

$$x'_k = x_k + r(b_k - x_k)\left(1 - \frac{t}{T}\right)^b \qquad or \qquad x'_k = x_k - r(x_k - a_k)\left(1 - \frac{t}{T}\right)^b$$

- where r is uniform random number in [0,1], t is the current generation number, T is the maximum number of generations, b determines the degree of non-uniformity, a_k and b_k are the upper and lower bounds for the k^{th} decision variable.

Direction Based Crossover Operator

- The previously defined operators do not make use of any domain knowledge to increase the possibility of improved performance by the offspring.
- if we have two solutions x and y with $f(y) \geq f(x)$, then an offspring can be generated as follows (fitness maximization problem): $x' = y + r(y - x)$

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

- If $f(x) \geq f(y)$, we can modify the above expression accordingly by interchanging x and y.
- The justification is that we generate offspring in the direction of ascent of the fitness values. This is true if x and y are close to each other.

Direction Based Mutation Operator

- This operator also uses the same justification as the direction based crossover operator as follows: *x'=x+rd,* where *r* is a uniform random number in [0,1], and *d* is the fitness ascent direction.
- Direction *d* is the same as the gradient of *f(x).* But, we cannot make use of the gradient information as GAs do not expect a functional form for the fitness.
- The gradient component can be numerically estimated as follows:

$$\frac{\partial f}{\partial x_k} \approx \frac{f(x_1, \ldots, x_k + \Delta x_k, \ldots, x_m) - f(x_1, \ldots, x_k, \ldots, x_m)}{\Delta x_k}$$

- Δx_k is a small real number. Make sure that the mutated offspring does not move out of the feasible range. In that situation, choose a random direction *d*.

Additional Real Coded Operators

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

- If interested, you can investigate the above mentioned operators.

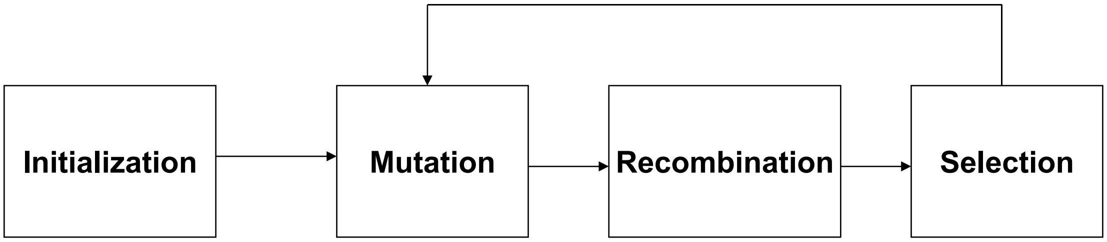
Evolution Strategy (ES) (1960s)

- Randomly generate an initial population of M solutions. Compute the fitness values of these M solutions.
- Use all initial vectors as parents to create nb offspring vectors by Gaussian mutation i.e. Xnew=Xold+N(0, σ)
- Calculate the fitness of nb vectors, and prune the population to M fittest vectors.
- 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

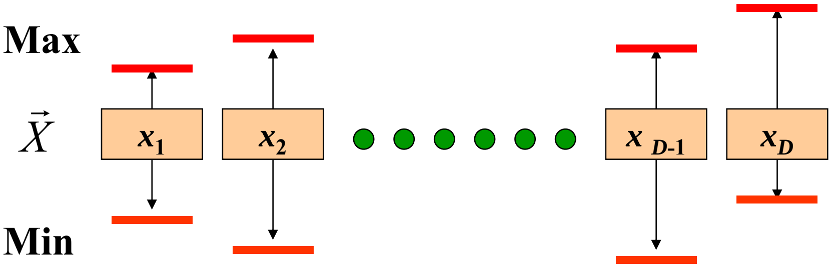
- **A stochastic population-based algorithm for continuous function optimization (Storn and Price, 1995)**
- **Finished 3rd at the First International Contest on Evolutionary Computation, Nagoya, 1996 (*icsi.berkley.edu/~storn*)**
- **Outperformed several variants of GA and PSO over a wide variety of numerical benchmarks over past several years.**
- **Continually exhibited remarkable performance in competitions on different kinds of optimization problems like dynamic, multi-objective, constrained, and multi-modal problems held under IEEE congress on Evolutionary Computation (CEC) conference series.**
- **Very easy to implement in any standard programming language.**
- **Very few control parameters (typically three for a standard DE) and their effects on the performance have been well studied.**
- **Spatial complexity is very low as compared to some of the most competitive continuous optimizers like CMA-ES.**

- DE is an Evolutionary Algorithm
- This Class also includes GA, Evolutionary Programming and Evolutionary Strategies



Basic steps of an Evolutionary Algorithm

Solution Representation

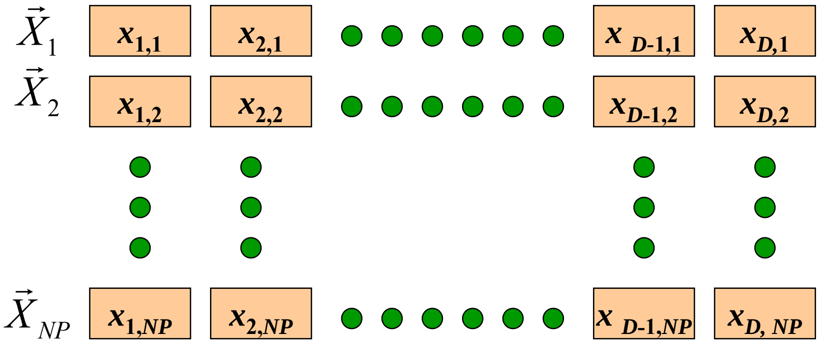


Solutions are represented as vectors of size *D* with each value taken from some domain.

May wish to constrain the values taken in each domain above and below.

Maintain Population - *NP*

We will maintain a population of size *NP*



$\vec{X}_1$	$x_{1,1}$	$x_{2,1}$	● ● ● ● ● ●	$x_{D-1,1}$	$x_{D,1}$
$\vec{X}_2$	$x_{1,2}$	$x_{2,2}$	● ● ● ● ● ●	$x_{D-1,2}$	$x_{D,2}$
	● ● ●	● ● ●		● ● ●	● ● ●
$\vec{X}_{NP}$	$x_{1,NP}$	$x_{2,NP}$	● ● ● ● ● ●	$x_{D-1,NP}$	$x_{D, NP}$

The population size *NP*

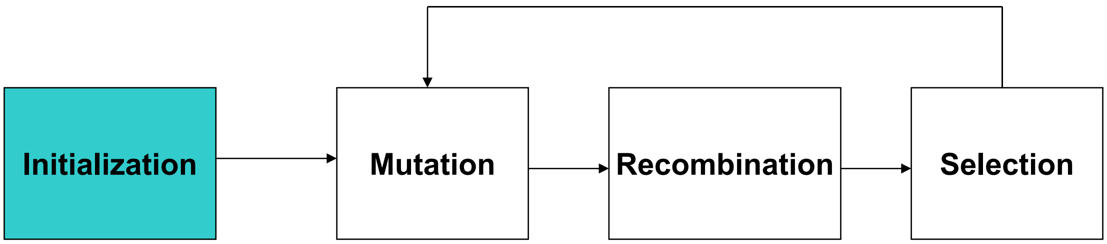
1) The influence of *NP* on the performance of DE is yet to be extensively studied and fully understood.

2) Storn and Price have indicated that a reasonable value for *NP* could be chosen between 5*D* and 10*D* (*D* being the dimensionality of the problem).

3) Brest and Maučec presented a method for gradually reducing population size of DE. The method improves the efficiency and robustness of the algorithm and can be applied to any variant of DE.

4) But, recently, all best performing DE variants used populations ~50-100 for dimensions from 50D to 1000D for the following scalability Special Issue:

F. Herrera M. Lozano D. Molina, "Test Suite for the Special Issue of Soft Computing on Scalability of Evolutionary Algorithms and other Metaheuristics for Large Scale Continuous Optimization Problems". Available: http://sci2s.ugr.es/eamhco/CFP.php.

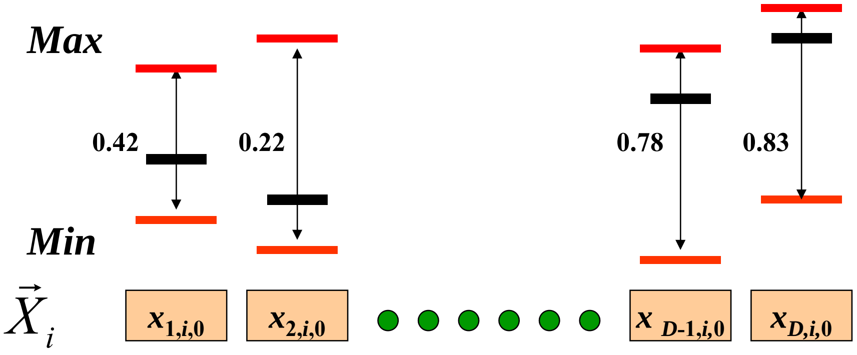


Initialization

Mutation

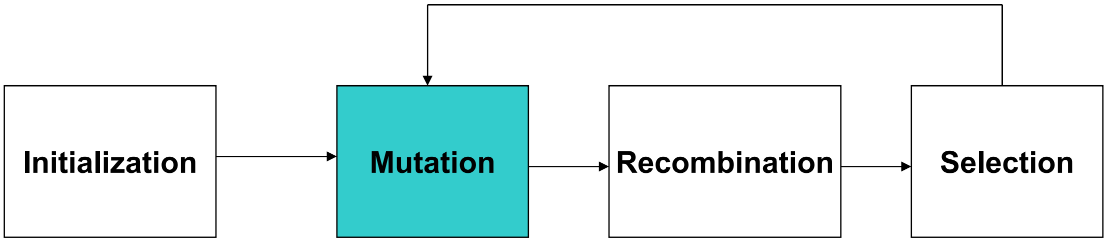
Recombination

Selection



$$x_{j,i,0} = x_{j,\min} + rand_{i,j}[0,1]\cdot(x_{j,\max} - x_{j,\min})$$

Different $rand_{i,j}[0,1]$ values are instantiated for each *i* and *j*.



Initialization

Mutation

Recombination

Selection

- **For each population member, select 3 other popln members (r_1, r_2, r_3 in the eqn below) randomly.**

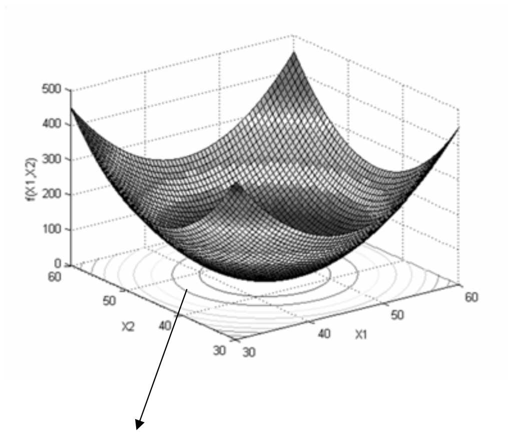
- **Add the weighted difference of two of the parameter vectors to the third to form a donor/mutant vector (most commonly seen form of DE-mutation):**

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

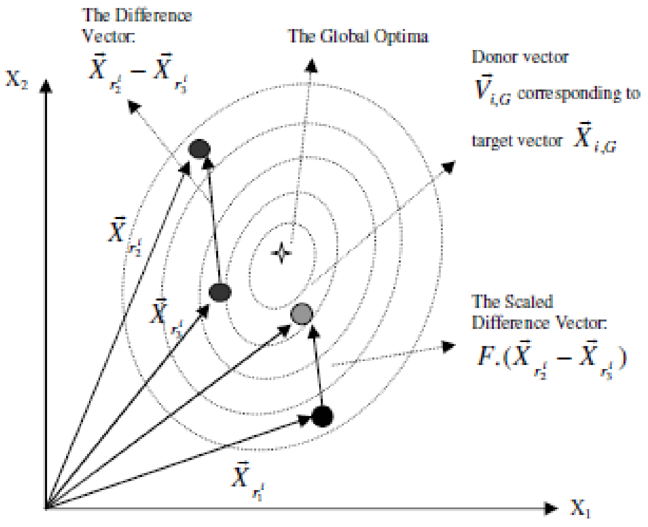
- **The scaling factor *F* is a constant from (0, 1.5) in the traditional DE**

- **Self-referential Mutation:**

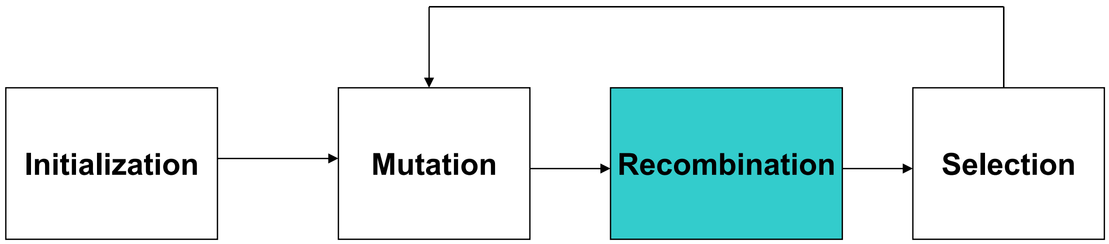
Example of formation of donor vector over two-dimensional constant cost contours



Constant cost contours of Sphere function



The Difference Vector:
$\vec{X}_{r_2^i} - \vec{X}_{r_3^i}$
The Global Optima
Donor vector
$\vec{V}_{i,G}$ corresponding to
target vector $\vec{X}_{i,G}$
X_2
$\vec{X}_{r_2^i}$
$\vec{X}_{r_3^i}$
The Scaled Difference Vector:
$F.(\vec{X}_{r_2^i} - \vec{X}_{r_3^i})$
$\vec{X}_{r_1^i}$
X_1



Initialization

Mutation

Recombination

Selection

Binomial (Uniform) Crossover:

Components of the donor vector enter into the trial offspring vector in the following way:

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

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

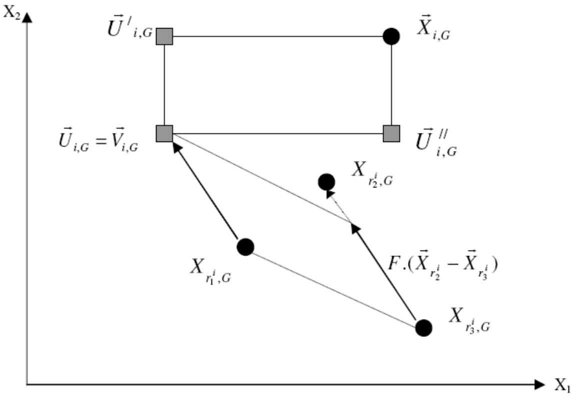
An Illustration of Binomial Crossover in 2-D Parametric Space:

Three possible trial vectors:

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 $\vec{V}_{i,G}$.



Exponential (two-point modulo) Crossover:

First choose integers *n* (as starting point) and *L* (number of components the donor actually contributes to the offspring) from the interval [1,*D*]

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

where the angular brackets $\langle \; \rangle_D$ denote a modulo function with modulus *D*.

Pseudo-code for choosing *L*:

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

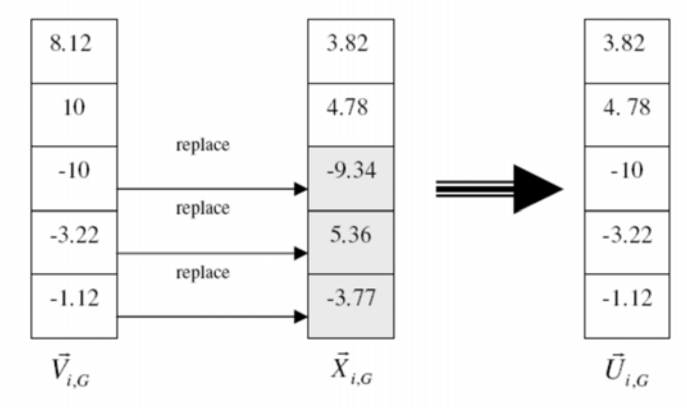
Exploits linkages among neighboring decision variables.
Existence of linkage means some decision variables must be changed together.

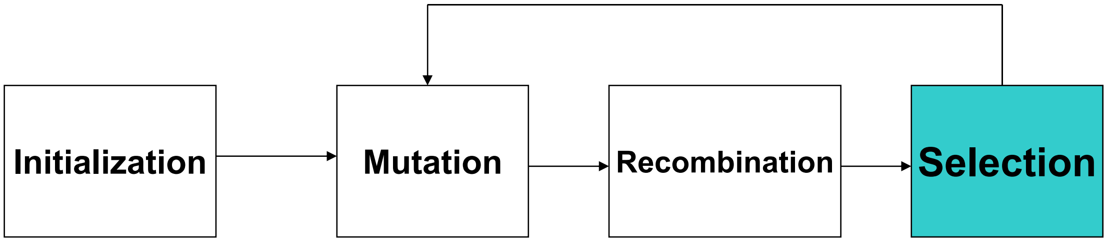
If benchmark problems have this feature, it performs well.
Similarly, for real-world problems with neighboring linkages.

Example: Let us consider the following pair of donor and target vectors

$$\vec{X}_{i,G} = \begin{bmatrix} 3.82 \\ 4.78 \\ -9.34 \\ 5.36 \\ -3.77 \end{bmatrix} \qquad \vec{V}_{i,G} = \begin{bmatrix} 8.12 \\ 10 \\ -10 \\ -3.22 \\ -1.12 \end{bmatrix}$$

Suppose *n* = 3 and *L*= 3 for this specific example. Then the exponential crossover process can be shown as:





- **“Survival of the fitter” principle in selection: The trial offspring vector is compared with the target (parent) vector and the one with a better fitness is admitted to the next generation population.**

$$\vec{X}_{i,G+1} = \vec{U}_{i,G}, \text{ if } f(\vec{U}_{i,G}) \le f(\vec{X}_{i,G})$$
$$= \vec{X}_{i,G}, \text{ if } f(\vec{U}_{i,G}) > f(\vec{X}_{i,G})$$

- **Importance of parent-mutant crossover & parent-offspring competition-based selection**

An Example of Optimization by DE

Consider the following two-dimensional function

$f\,(x, y) = x^2+y^2$ *The minima is at (0, 0)*

Let's start with a population of 5 candidate solutions randomly initiated in the range (-10, 10)

$X_{1,0}$ = [2, -1] $X_{2,0}$ = [6, 1] $X_{3,0}$ = [-3, 5] $X_{4,0}$ = [-2, 6] $X_{5,0}$= [6,-7]

For the first vector X_1, randomly select three other vectors say X_2, X_4 and X_5

Now form the donor vector as, $V_{1,0} = X_{2,0}+F.\,(X_{4,0} - X_{5,0})$

$$V_{1,0} = \begin{bmatrix} 6 \\ 1 \end{bmatrix} + 0.8 \times \left\{ \begin{bmatrix} -2 \\ 6 \end{bmatrix} - \begin{bmatrix} 6 \\ -7 \end{bmatrix} \right\} = \begin{bmatrix} -0.4 \\ 10.4 \end{bmatrix}$$

Now we form the trial offspring vector by exchanging components of $V_{1,0}$ with the target vector $X_{1,0}$

Let *rand*[0, 1) = 0.6. If we set *Cr* = 0.9, since 0.6 < 0.9, $u_{1,1,0} = V_{1,1,0} = -\,0.4$

Again next time let *rand*[0, 1) = 0.95 > *Cr*
Hence $u_{1,2,0} = x_{1,2,0} = -\,1$

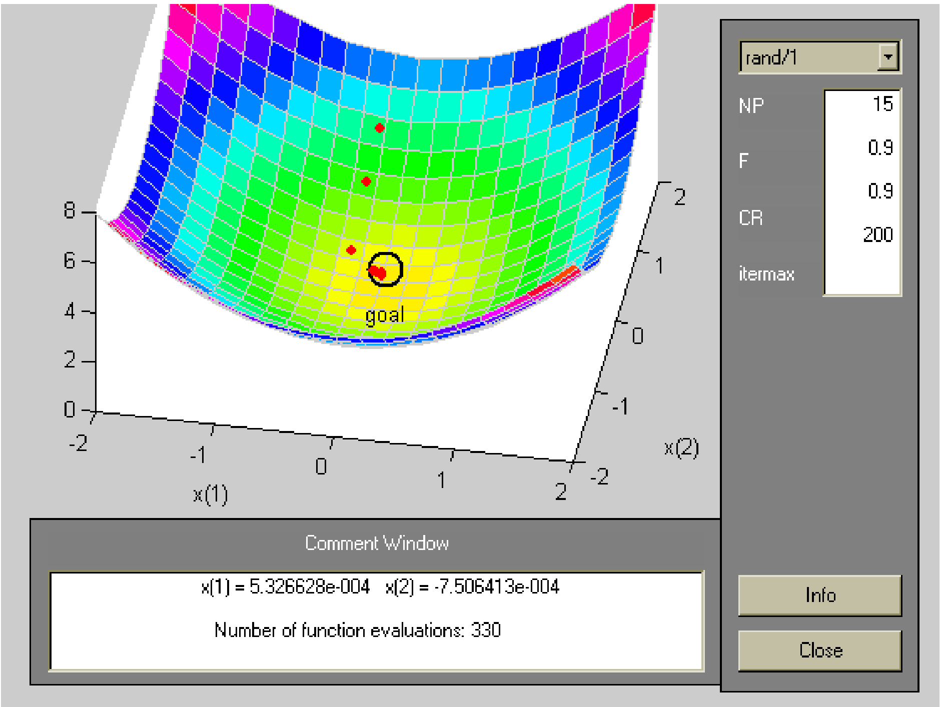
So, finally the offspring is $U_{1,0} = \begin{bmatrix} -0.4 \\ -1 \end{bmatrix}$

Fitness of parent:	Fitness of offspring
$f\,(2, -1) = 2^2 + (-1)^2 = 5$	$f\,(-0.4, -1) = (-0.4)^2 + (-1)^2 = 1.16$

Hence the parent is replaced by offspring at *G* = 1

Population at $G = 0$	Fitness at $G = 0$	Donor vector at $G = 0$	Offspring Vector at $G = 0$	Fitness of offspring at $G = 1$	Evolved population at $G = 1$
$X_{1,0}$ = [2,-1]	5	$V_{1,0}$ =[-0.4,10.4]	$U_{1,0}$ =[-0.4,-1]	1.16	$X_{1,1}$ =[-0.4,-1]
$X_{2,0}$= [6, 1]	37	$V_{2,0}$ =[1.2, -0.2]	$U_{2,0}$ =[1.2, 1]	2.44	$X_{2,1}$ =[1.2, 1]
$X_{3,0}$= [-3, 5]	34	$V_{3,0}$ =[-4.4, -0.2]	$U_{3,0}$ =[-4.4, -0.2]	19.4	$X_{3,1}$ =[-4.4, -0.2]
$X_{4,0}$= [-2, 6]	40	$V_{4,0}$ =[9.2, -4.2]	$U_{4,0}$ =[9.2, 6]	120.64	$X_{4,1}$ =[-2, 6]
$X_{5,0}$= [6, 7]	85	$V_{5,0}$ =[5.2, 0.2]	$U_{5,0}$ =[6, 0.2]	36.04	$X_{5,1}$ =[6, 0.2]

Locus of the fittest solution: DE working on 2D Sphere Function



rand/1

NP 15

F 0.9

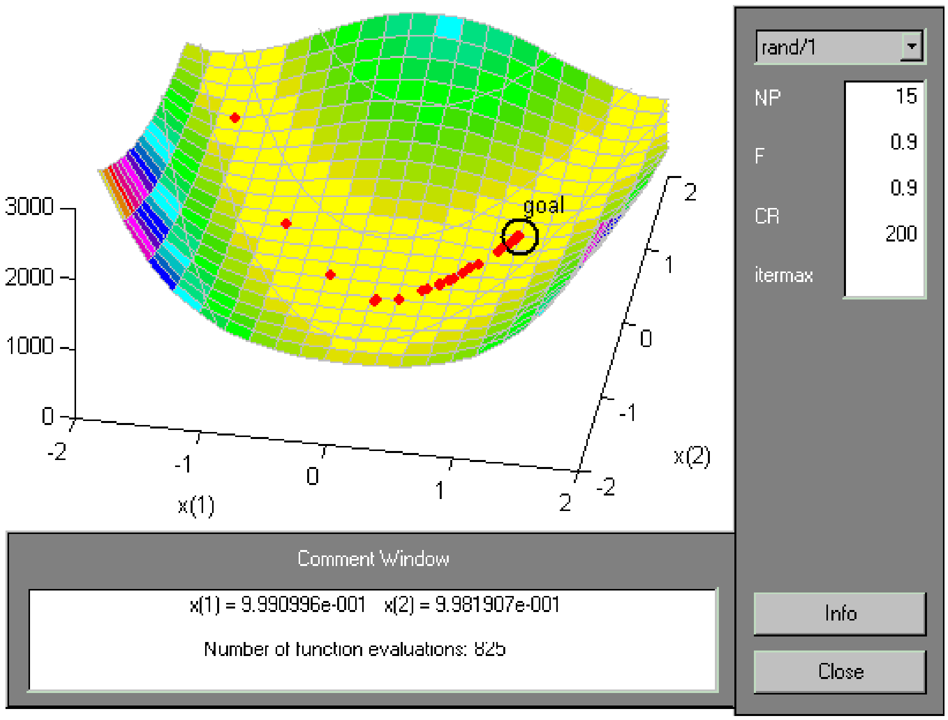
CR 0.9

itermax 200

x(1) = 5.326628e-004   x(2) = -7.506413e-004

Number of function evaluations: 330

Locus of the fittest solution: DE working on 2D Rosenbrock Function



rand/1

NP 15

F 0.9

CR 0.9

itermax 200

Comment Window

x(1) = 9.990996e-001 x(2) = 9.981907e-001

Number of function evaluations: 825

Five most frequently used DE mutation schemes

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

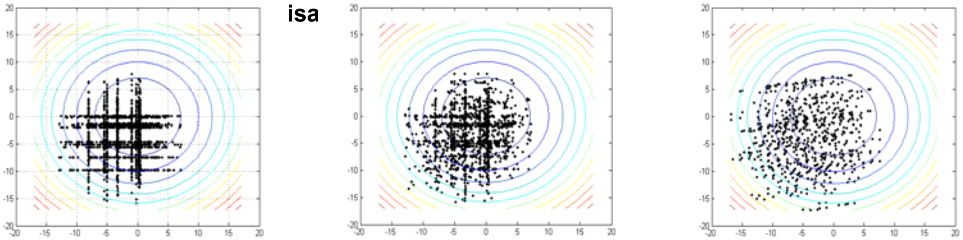
The general convention used for naming the various mutation strategies is DE/x/y/z, where DE stands for Differential Evolution, x represents a string denoting the vector to be perturbed, y is the number of difference vectors considered for perturbation of x, and z stands for the type of crossover being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

The Crossover Rate *Cr*

1) The parameter *Cr* controls how many parameters in expectation, are changed in a population member.
2) Low value of *Cr*, a small number of parameters are changed in each generation and the stepwise movement tends to be orthogonal to the current coordinate axes → Good for separable problems

3) High values of *Cr* (near 1) cause most of the directions of the mutant vector to be inherited prohibiting the generation of axis orthogonal steps → Good for non-separable problems

Empirical distribution of trial vectors for three different values of *Cr* has been shown. The plots were obtained by obtained by running DE on a single starting population of 10 vectors for 200 isa



(a) *Cr* = 0

(b) *Cr* = 0.5

(c) *Cr* = 1.0

For schemes like DE/rand/1/bin the performance is rotationally invariant only when *Cr* = 1.

At that setting, crossover is a vector-level operation that makes the trial vector a pure mutant i.e.

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

Self-Adaptive DE (SaDE)

- Includes both control parameter adaptation and strategy adaptation

Strategy Adaptation:

Four effective trial vector generation strategies: DE/rand/1/bin, DE/rand-to-best/2/bin, DE/rand/2/bin and DE/current-to-rand/1 are chosen to constitute a strategy candidate pool.

For each target vector in the current population, one trial vector generation strategy is selected from the candidate pool according to the probability learned from its success rate in generating improved solutions (that can survive to the next generation) within a certain number of previous generations, called the *Learning Period* (*LP*).

A. K. Qin and P. N. Suganthan, “Self-adaptive Differential Evolution Algorithm for Numerical Optimization”, *IEEE Congress on Evolutionary Computation*, pp.1785-1791, 2005.

A. K. Qin, V. L. Huang, and P. N. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization", *IEEE Trans. on Evolutionary Computation,* 13(2):398-417, April, 2009.

SaDE (Contd..)

Control Parameter Adaptation:

1) ***NP*** **is left as a user defined parameter.**
2) **A set of *F* values are randomly sampled from normal distribution *N*(0.5, 0.3) and applied to each target vector in the current population.**
3) ***CR*** **obeys a normal distribution with mean value** CR_m **and standard deviation *Std* =0.1, denoted by** $N(CR_m, Std)$ **where** CR_m **is initialized as 0.5.**
4) **SaDE gradually adjusts the range of *CR* values for a given problem according to previous *CR* values that have generated trial vectors successfully entering the next generation.**

JADE

1) Uses DE/current-to-*p*best strategy as a less greedy generalization of the DE/current-to-best/ strategy.
Instead of only adopting the best individual in the DE/current-to-best/1 strategy, the current-to-pbest/1 strategy utilizes the information of other good solutions.

Denoting $\vec{X}^{p}_{best,G}$ **as a randomly chosen vector from the top 100*p*% individuals in current population**

DE/current-to-*p*best/1 without external archive: $\vec{V}_{i,G} = \vec{X}_{i,G} + F_i \cdot (\vec{X}^{p}_{best,G} - \vec{X}_{i,G}) + F_i \cdot (\vec{X}_{r_1^i,G} - \vec{X}_{r_2^i,G})$

2) JADE can optionally make use of an external archive (A), which stores the recently explored inferior

solutions. In case of DE/current-to-pbest/1 with archive, $\vec{X}_{i,G}$, $\vec{X}^{p}_{best,G}$ **, and** $\vec{X}_{r_1^i,G}$ **are selected from the**

current population P, but $\vec{X}_{r_2^i,G}$ **is selected from** $P \cup A$

J. Zhang, and A. C. Sanderson, "JADE: Adaptive differential evolution with optional external archive", *IEEE Transactions on Evolutionary Computation*, 13(5) pp. 945-958, Oct. 2009.

JADE (Contd..)

3) JADE adapts the control parameters of DE in the following manner:

A) *Cr* for each individual and at each generation is randomly generated from a normal distribution $N(\mu_{Cr}, 0.1)$ and then truncated to [0, 1].

The mean of normal distribution is updated as: $\mu_{Cr} = (1-c).\mu_{Cr} + c.mean_A(S_{Cr})$

where S_{Cr} be the set of all successful crossover probabilities Cr_i s at generation *G*

B) Similarly for each individual and at each generation F_i is randomly generated from a Cauchy distribution $C(\mu_F, 0.1)$ with mean μ_F and scale parameter 0.1.

F_i is truncated if F_i > 1 or regenerated if F_i <= 0

The location parameter of the Cauchy distribution is updated as: $\mu_F = (1-c).\mu_F + c.mean_L(S_F)$

where S_F is the set of all successful scale factors at generation G and $mean_L$ is the Lehmer mean:

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

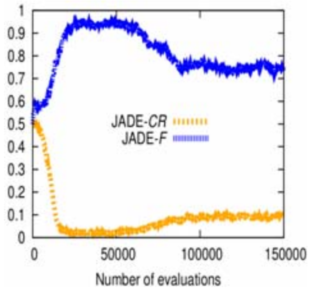
JADE usually performs best with 1/*c* chosen from [5, 20] and *p* from [5%, 20%]

Success-History based Adaptive DE (SHADE)

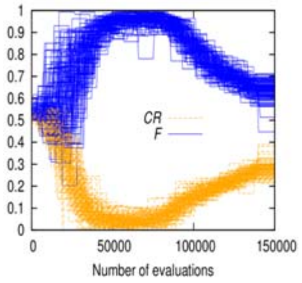
- An improved version of JADE
- Uses a success-history based adaptation
  - Based on a historical memory of successful parameter settings that were previously used found during the run
  - A historical memory M_{CR} , M_F are used, instead of adaptive parameter u_{cr} , u_F
  - This improves the robustness of JADE

Fig. Their adaptation behaviors on Rastrigin (30 dimensions)

JADE uses a single pair u_{cr}, u_F

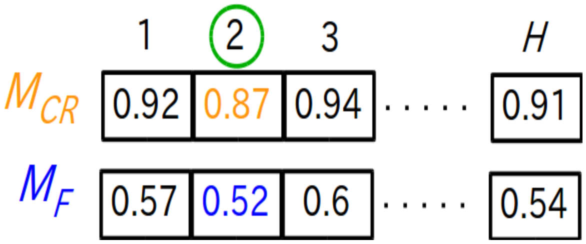


SHADE maintains a diverse set of parameters in a historical memory M_{CR} , M_F



SHADE

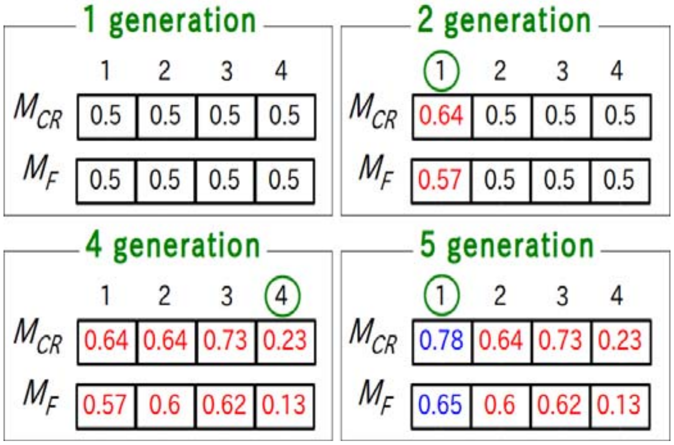
- The weighted Lehmer mean (in CEC'14 ver.) values of S_{CR} and S_F , which are successful parameters for each generation, are stored in a historical memory M_{CR} and M_F



	1	2	3		H
M_{CR}	0.92	0.87	0.94	· · · · ·	0.91
M_F	0.57	0.52	0.6	· · · · ·	0.54

- **CR_i and F_i are generated by selecting an index r_i randomly from [1, memory size H]**
- **Example: if selected index $r_i = 2$**
  - **CR_i = NormalRand(0.87, 0.1)**
  - **F_i = CauchyRand(0.52, 0.1)**

Example of memory update in SHADE



1 generation

	1	2	3	4
M_{CR}	0.5	0.5	0.5	0.5
M_F	0.5	0.5	0.5	0.5

2 generation

	1	2	3	4
M_{CR}	0.64	0.5	0.5	0.5
M_F	0.57	0.5	0.5	0.5

4 generation

	1	2	3	4
M_{CR}	0.64	0.64	0.73	0.23
M_F	0.57	0.6	0.62	0.13

5 generation

	1	2	3	4
M_{CR}	0.78	0.64	0.73	0.23
M_F	0.65	0.6	0.62	0.13

- The contents of both memories are initialized to 0.5 and the index counter is set 1
- The CR_i and F_i values used by successful individuals are recorded in S_{CR} and S_F
- At the end of the generation, the contents of memory are updated by the mean values of S_{CR} and S_F
- The index counter is incremented
- Even if S_{CR} , S_F for some particular generation contains a poor set of values, the parameters stored in memory from previous generations cannot be negatively impacted
- If the index counter exceeds the memory size H, the index counter wraps around to 1 again

Deterministic population reduction methods

- General Policy for Evolutionary Algorithms
  - Explorative search is appropriate for estimating the promising regions
  - Exploitative search is appropriate for finding the higher precision solutions
- Deterministic population reduction methods
  - They use a large population size as initial population and reduce its size
  - This mechanism makes EA more robust and effective.

"Evaluating the performance of SHADE on CEC 2013 benchmark problems", Ryoji Tanabe and Alex Fukunaga, The University of Tokyo, Japan (**Codes-Results available**, **as SHADE_CEC2013)**

L-SHADE in CEC 2014: "Improving the Search Performance of SHADE Using Linear Population Size Reduction," By Ryoji Tanabe and Alex S. Fukunaga

L-SHADE: SHADE with Linear Population Reduction

- **Deterministic Population Size Reduction (DPSR)**
  - **reduces the population by half at predetermined intervals**
  - **The frequency of the population reduction has to be tuned to match the initial population size as well as the dimensionality of the problem…**
- **Simple Variable Population Sizing (SVPS)**
  - **is a more general framework in which the shape of the population size reduction schedule is determined according to two control parameters**
  - **Due to its general versatility, tuning the two control parameters is very hard…**
- **Linear Population Size Reduction (LPSR) [Tanabe CEC 2014]**
  - **is a special case of SVPS which reduces the population linearly, and requires only initial population sizes**
  - **L-SHADE is an extended SHADE with LPSR**

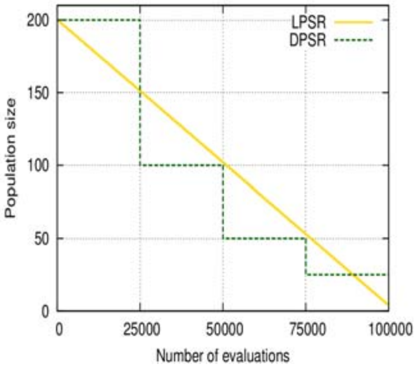


Fig. Comparison of population resizing schedule between LPSR and DPSR (# of reduction = 4)

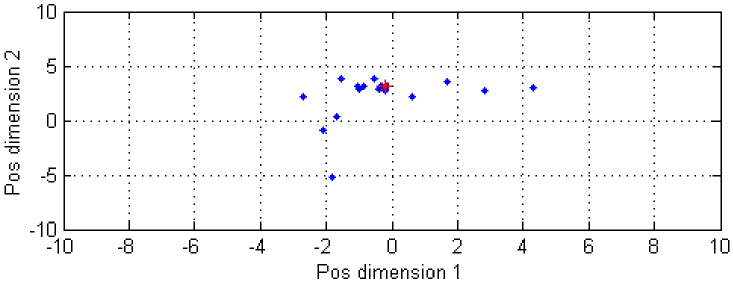
L-SHADE's C++ and Matlab/Octave code can be downloaded from Ryoji Tanabe's site (https://sites.google.com/site/tanaberyoji/)

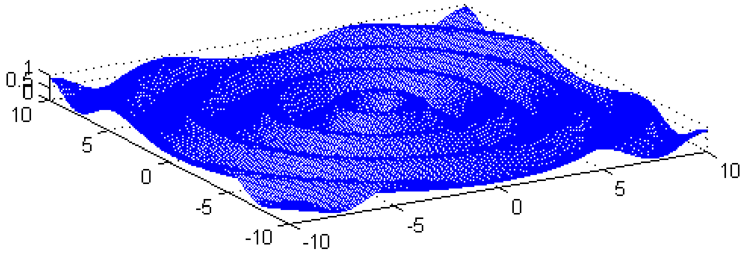
Particle Swarm Optimization

- Particle Swarm Optimization (PSO) applies concept of social interactions to problem solving.
- It was developed in 1995 by Jim Kennedy and Russ Eberhart [Kennedy, J. and Eberhart, R. (1995). "Particle Swarm Optimization", *Proc. of IEEE International Conference on Neural Networks*, pp. 1942-1948, IEEE Press.] (http://dsp.jpl.nasa.gov/members/payman/swarm/kennedy95-ijcnn.pdf)
- It has been applied successfully to a wide variety of search and optimization problems.
- In PSO, a swarm of *n* individuals communicate either directly or indirectly with others' search directions (gradients).
- PSO is a simple but powerful search technique.

PSO

- PSO is modeled after the social behavior of flocks of birds, swarm of insects, school of fish, etc.
- PSO is a population based search process where individuals, referred to as particles, are grouped into a swarm.
- Each particle in the swarm represents a candidate solution to the optimization problem.
- In a PSO system, each particle is "flown" through the multidimensional search space, adjusting its position in search space according to its own experience and that of other particles'.
- The position coordinates are used to evaluate the fitness function.
- There is no evolution, no crossover, no mutation, etc. Generation counter is usually replaced by an iteration counter.



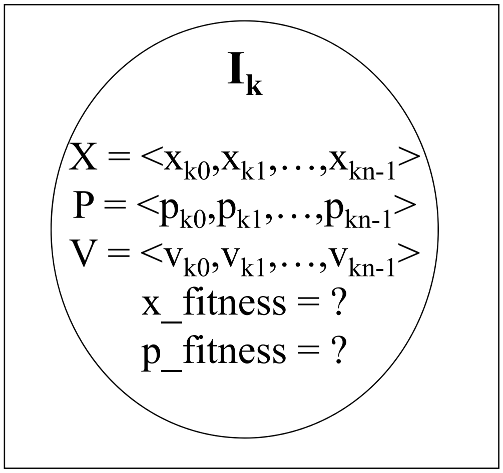


- Schaffer's:

f6(x,y)=0.5-(sin^2(sqrt(x^2+y^2))-0.5)/(1+0.001*(x^2+y^2))^2;

Particle Swarm Optimization: The Anatomy of a Particle

- A particle (individual) is composed of:
  - Three vectors:
    - The **x-vector** records the current position (location) of the particle in the search space,
    - The **p-vector** (*pbest*) records the location of the best solution found so far by the particle, and
    - The **v-vector** contains a gradient (direction) for which particle will travel in if undisturbed.
  - Two fitness values:
    - The **x-fitness** records the fitness of the x-vector, and
    - The **p-fitness** records the fitness of the p-vector, the best solution found by the particle so far.



$\mathbf{I_k}$

$X = <x_{k0},x_{k1},\ldots,x_{kn-1}>$

$P = <p_{k0},p_{k1},\ldots,p_{kn-1}>$

$V = <v_{k0},v_{k1},\ldots,v_{kn-1}>$

x_fitness = ?

p_fitness = ?

Particle Swarm Optimization

- In PSO, particles never die!
- Particles can be seen as simple agents that fly through the search space and record (and possibly communicate) the best solution that they have discovered.
- So the question now is, “How does a particle move from on location in the search space to another?”
- This is done by simply adding the v-vector to the x-vector to get another x-vector ($X_i \leftarrow X_i + V_i$).
- Once the particle computes the new X_i it then evaluates its new location. If x-fitness is better than p-fitness, then $P_i = X_i$ and p-fitness = x-fitness.

PSO: Update Equations

- Actually, we must adjust the v-vector before adding it to the x-vector as follows:

$$\begin{aligned} v_{id} \leftarrow v_{id} &+ \varphi 1 * rnd() * (p_{id} - x_{id}) \\ &+ \varphi 2 * rnd() * (p_{gd} - x_{id}); \end{aligned}$$

$$x_{id} \leftarrow x_{id} + v_{id};$$

- where *i* is the particle,
- $\varphi 1, \varphi 2$ are learning rates governing the **cognition** and **social** components
- Where *g* represents the index of the particle with the best p-fitness (also know as gbest or global best), and
- Where *d* is the d^{th} dimension.

PSO: Initialization

- Initially the values of the velocity vectors are randomly generated with the range *[-Vmax, Vmax*] where *Vmax* is the maximum value that can be assigned to any v_{id}.

- Random initialization of positions (or coordinates of the variables in the optimization problem) X_{id} within the given upper and lower bounds.

PSO: Swarm Types

- [Kennedy, J. (1997), “The Particle Swarm: Social Adaptation of Knowledge”, Proceedings of the 1997 International Conference on Evolutionary Computation, pp. 303-308, IEEE Press.] identifies 3 types of PSO based on φ1 and φ2 .
- Given: $v_{id} \leftarrow v_{id} + \varphi 1 * rnd() * (p_{id} - x_{id}) + \varphi 2 * rnd() * (p_{gd} - x_{id});$

  $x_{id} \leftarrow x_{id} + v_{id};$

  - Full Model (φ1, φ2 > 0)
  - Cognition Only (φ1 > 0 and φ2 = 0),
  - Social Only (φ1 = 0 and φ2 > 0)

PSO: Related Issues

- There are a number of related issues concerning PSO:
  - Controlling velocities (determining the best value for Vmax),
  - Swarm Size,
  - Neighborhood Size,
  - Updating X and Velocity Vectors,
  - Robust Settings for (φ1 and φ2),

- Carlisle, A. and Dozier, G. (2001). "An Off-The-Shelf PSO", *Proceedings of the 2001 Workshop on Particle Swarm Optimization*, pp. 1-6, Indianapolis, IN. (http://antho.huntingdon.edu/publications/Off-The-Shelf_PSO.pdf)

PSO: Controlling Velocities

- When using PSO, it is possible for the magnitude of the velocities to become very large.
- Performance can suffer if Vmax is inappropriately set.
- Two methods were developed for controlling the growth of velocities:
  - A dynamically adjusted inertia factor, and
  - A constriction coefficient.

PSO: With the Inertia Factor

- When the inertia factor is used, the equation for updating velocities is changed to:

  $V_{id} \leftarrow \omega * v_{id} + \varphi 1 * rnd() * (p_{id} - x_{id})$
  $\qquad + \varphi 2 * rnd() * (p_{gd} - x_{id});$

- Where ω is initialized to 1.0 and is gradually reduced over time/generations (measured by iterations or generations).

PSO: With the Constriction Coefficient

- In 1999, Maurice Clerc developed a constriction Coefficient for PSO.

$V^{(t+1)}_{id} \leftarrow K[v_{id} + \varphi 1 * rnd() * (p_{id} - x_{id})$
$\qquad + \varphi 2 * rnd() * (p_{gd} - x_{id})];$

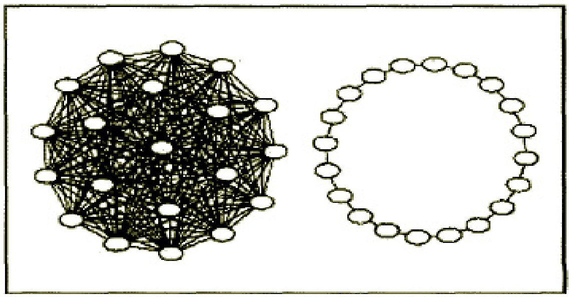
  - Where $K = 2/|2 - \varphi - sqrt(\varphi^2 - 4\varphi)|$,
  - $\varphi = \varphi 1 + \varphi 2$, and
  - $\varphi > 4$.

PSO: Particle Update Methods

- There are two ways that particles can be updated:
  - Synchronously
  - Asynchronously
- Asynchronous update allows for newly discovered solutions to be used more quickly
- These choices are valid for Evolutionary Algorithms too.

Local PSO

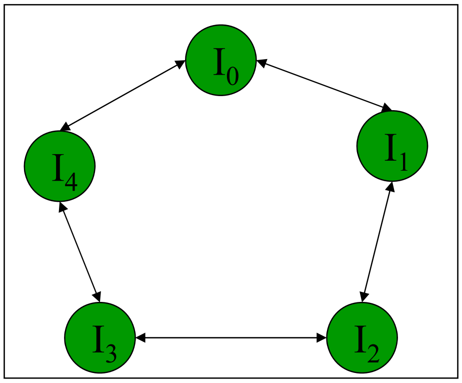
- There were two versions of PSO: the global version and the local version .
- In the global version of PSO, each particle' velocity is adjusted according to it's personal best performance achieved so far (*pbest*) and the best performance achieved so far by all the particles (*gbest*). All previous equations are for global version.
- In the local version, each particle's velocity is adjusted according to its *pbest* and the best performance achieved so far within its neighborhood (*lbest*). The neighborhood of each particle is defined as some of the topologically nearest particles.

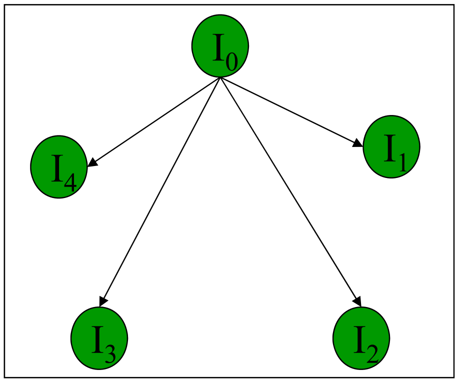


GBEST (LEFT) AND LBEST SOCIOMETRIC PATTERNS'.

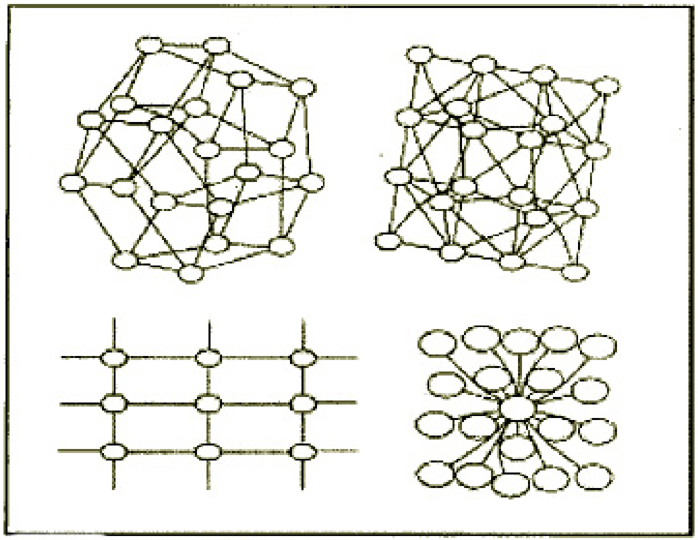
Particle Swarm Optimization: Swarm Topologies

- In local PSO, there are several topologies. Two examples are
  - Ring Topology (neighborhood of 3)
  - Star Topology (global neighborhood)





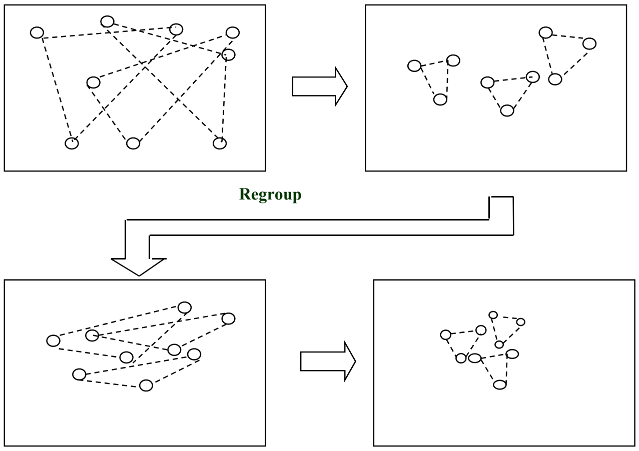
- Different neighborhood structures for the local PSO. Local neighborhoods are used to prevent premature convergence.



THE VON NEUMANN NEIGHBORHOOD IS SHOWN WRAPPED (TOP LEFT) AND ONE SECTION IS FLATTENED OUT (LOWER LEFT). THE PYRAMID SOCIOMETRY IS AT THE TOP RIGHT, AND THE STAR AT LOWER RIGHT.

Dynamic multi-swarm PSO (DMS-PSO)

- **DMS-PSO is constructed based on the local version of PSO with a novel neighborhood topology**
- **Two major characteristics of the novel neighborhood topology:**
  - **Small sized swarms**
  - **Randomized re-grouping scheme**



- The population is divided into several sub-swarms randomly.
- Each sub-swarm utilizes its own particles to search for better solutions and converge to some suboptimal solution.
- The whole population is re-grouped into new sub-swarms periodically. New sub-swarms continue the search procedure.
- This process continues until a termination criterion is satisfied.

Comprehensive learning PSO (CLPSO)

- CLPSO learning strategy:

$$v_i^d \leftarrow w \times v_i^d + c \times rand_i^d \times \left(pbest_{f_i(d)}^d - x_i^d\right)$$
$$x_i^d \leftarrow x_i^d + v_i^d$$

- $f_i = [f_i(1), f_i(2), \ldots, f_i(D)]$ denotes a set of particle indices with respect to each dimension of the particle *i*. $f_i(d)$ represents a comprehensive exemplar with each dimension composed of the value from the corresponding dimension of the **pbest** of particle. These indices take the value *i* itself with the probability *Pc*$_i$, referred to as the learning probability, which takes different values with respect to different particles.

- For each dimension of particle *i*, we generate a random number. If this random number is larger than *Pc*$_i$, the corresponding dimension of particle *i* will learn from its own **pbest**, otherwise it will learn from the **pbest** of another randomly chosen particle.

CLPSO

- Tournament selection with size 2 is used to choose the index $f_i(d)$

- To ensure that a particle learns from good exemplars and to minimize the time wasted on poor directions, we allow each particle to learn from the exemplars until such particle stop to improve for a certain number of generations, called the refreshing gap *m* (7 generations).
- After that, we re - assign $f_i = [f_i(1), f_i(2), \ldots, f_i(D)]$ for each particle *i*.

- The detailed description and algorithmic implementation can be found in Matlab codes including CLPSO and several state-of-the-art PSO variants available.

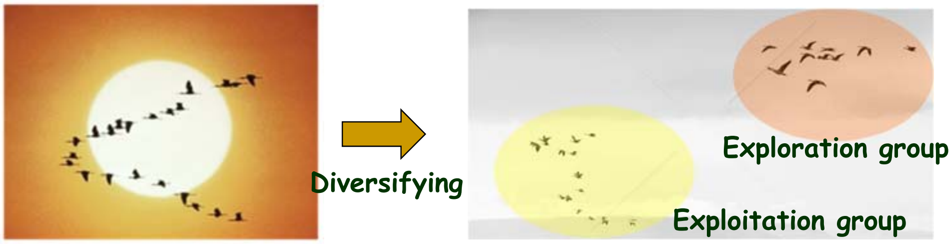
CLPSO

- Three major differences between CLPSO and the conventional PSO are highlighted:
  - Instead of using particle's **pbest** and **gbest** as the exemplars, all particles' **pbest**s can be used to guide a particle's flying direction.
  - Instead of learning from the same exemplar for all dimensions, different dimensions of a particle may learn from different exemplars within certain generations. In other words, at one iteration, each dimension of a particle may learn from the corresponding dimension of different particle's **pbest**.
  - Instead of learning from two exemplars (**pbest** and **gbest**) in every generation, each dimension of a particle in CLPSO learns from just one comprehensive exemplar within certain generations.
- Experimental results over a suite of 16 numerical test functions have demonstrated the promising performance of the CLPSO to solve the multi-modal optimization problems in comparison with 8 state-of-the-art PSO variants.

Comprehensive Social Learning Strategy

- Premature convergence is still the main deficiency of the PSO.
  - In the original PSO, each particle learns from its *pbest* and *gbest* simultaneously. The *gbest* is more likely to provide a larger momentum, as |*gbest* -*X*| is likely to be larger than the |*pbest* -*X*| and drag the particle in its direction even if the current *gbest* is far from the global optimum
- Instead of learning from two exemplars namely the *pbest* and *gbest* simultaneously, as in the original PSO, each dimension of a particle learns from just <u>one</u> exemplar.
- Instead of learning from the *pbest* and *gbest* for all dimensions, each dimension of a particle could learn from <u>a different exemplar</u> for different dimensions .

HCLPSO – Heterogeneous Swarm



- **Velocity update for Subpopulation group 1**

$$V_i^d = w * V_i^d + c * rand_i^d * \left(\boxed{pbest_{fi(d)}^d} - X_i^d\right)$$

- **Velocity update Subpopulation group 2**

$$V_i^d = w * V_i^d + c_1 * rand_{1i}^d * \left(\boxed{pbest_{fi(d)}^d} - X_i^d\right) + c_2 * rand_{2i}^d * \left(\boxed{gbest^d} - X_i^d\right)$$

- **Position update Subpopulation group 1 and 2**

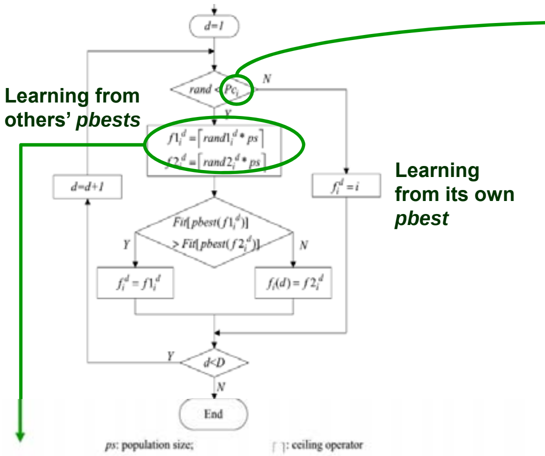
$$X_i^d = X_i^d + V_i^d$$

HCLPSO – Comprehensive Learning Strategy

- The exemplar $pbest_{fi(d)}^{d}$ is generated using the comprehensive learning strategy (CL).
- Instead of learning from the same exemplar for all dimensions, each dimension of a particle can learn from different particles for different dimensions.
- The particle can learn from *gbest*, its own *pbest* and other particles' *pbests* for different dimensions.
- The exemplar for each dimension are randomly selected according to probability *Pc* values called learning probability.

HCLPSO – Comprehensive Learning Strategy

- Generating the exemplar $pbest_{fi(d)}^{d}$

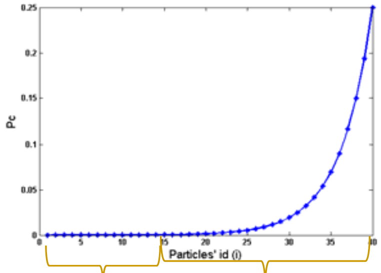

d=1
rand < Pc_i
N
Y
$f1_i^d = \lceil rand1_i^d * ps \rceil$
$f2_i^d = \lceil rand2_i^d * ps \rceil$
d=d+1
$f_i^d = i$
$Fit[pbest(f1_i^d)] > Fit[pbest(f2_i^d)]$
Y
N
$f_i^d = f1_i^d$
$f_i(d) = f2_i^d$
Y
d<D
N
End

ps: population size; ⌈ ⌉: ceiling operator

Learning from others' *pbests*

Learning from its own *pbest*

Learning Probability values,



Subpopulation　　Subpopulation

- ✓ Different particles has different levels of exploration and exploitation ability.
- ✓ g_1: learns mostly from itself and has strong exploration ability.
- ✓ g_2: learns from others and *gbest* has strong exploitation ability.

- Two particles are randomly selected and the particle with better fitness is chosen as the exemplar for that dimension.
- For g_1 : other particles are selected from its own population.

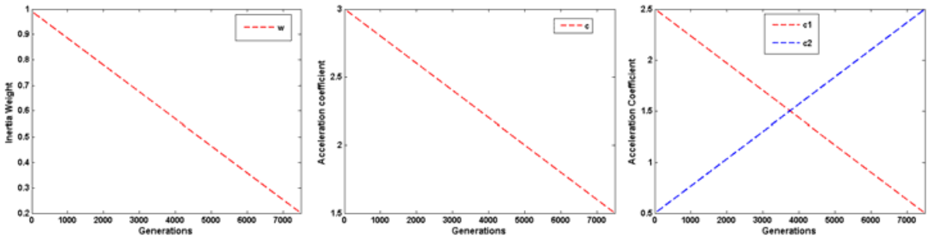
HCLPSO – Adaptive Control Parameters

- ❖ **Exploration Subpopulation group 1**

$$V_i^d = \boxed{w} * V_i^d + \boxed{c} * rand_i^d * \left(pbest_{fi(d)}^d - X_i^d\right)$$

- ❖ **Exploitation Subpopulation group 2**

$$V_i^d = \boxed{w} * V_i^d + \boxed{c_1} * rand_{1i}^d * \left(pbest_{fi(d)}^d - X_i^d\right) + \boxed{c_2} * rand_{2i}^d * \left(gbest^d - X_i^d\right)$$


"w" **"c"** **"c1 & c2"**

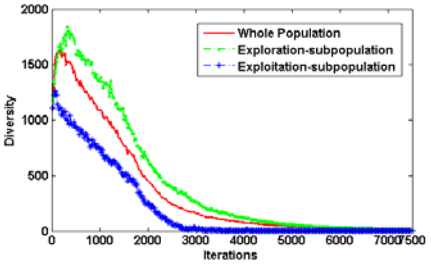
- ✓ **Adjusting the swarm's behaviour from exploration of the entire search space to exploitation of promising regions**
- ✓ **Enhancing Exploration at subpopulation group 1**
- ✓ **Enhancing Exploitation at subpopulation group 2**

HCLPSO – Is it working as designed?

- ❖ **Are the subpopulations are preforming exploration and exploitation?**
- ❖ **Swarm Diversity : "distance-to-average point" measure**
- ❖ **Diversity measure on unimodal and multimodal problems**

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

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



- ✓ **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**

Hybrid PSO

- Different algorithms can be combined, for example, Combine Advances of ES / EA / GAs with PSO
- PSO (and other SAs-Swarm algorithms) do not include cross-over.
- Including cross-over in SAs is a common hybridization.
- A common sense requirement for hybridization: When PSO is hybridized with DE, are we able obtain better optimization performance over the best of available PSO & DE algorithms?
- This is challenging…

Constrained Optimization

- In general, the constrained problems can be transformed into the following form:
- Minimize $f(\mathbf{x}), \mathbf{x} = [x_1, x_2, ..., x_D]$

subjected to: $g_i(\mathbf{x}) \le 0, i = 1, ..., q$

$h_j(\mathbf{x}) = 0, j = q+1, ..., m$

q is the number of inequality constraints and *m-q* is the number of equality constraints.

Constrained Optimization

- For convenience, the equality constraints can be transformed into inequality form: $|h_j(\mathbf{x})| - \varepsilon \le 0$

  where ε is the allowed tolerance.

- Then, the constrained problems can be expressed as

  Minimize

$$f(\mathbf{x}), \mathbf{x} = [x_1, x_2, ..., x_D]$$

  subjected to

$$G_j(\mathbf{x}) \le 0, j = 1, ..., m,$$

$$G_{1,...,q}(\mathbf{x}) = g_{1,...q}(\mathbf{x}), G_{q+1,...,m}(\mathbf{x}) = \left| h_{q+1,...m}(\mathbf{x}) \right| - \varepsilon$$

Constraint Handling

- Many optimization problems in science and engineering involve constraints. The presence of constraints reduces the feasible region and complicates the search process.
- Evolutionary algorithms (EAs) always perform unconstrained search.
- When solving constrained optimization problems, they require additional mechanisms to handle constraints

Constrained Optimization

- The main difficulty is that the GA often yields infeasible offspring (i.e. solutions that violate one or more of the constraints).
- Several approaches have been proposed recently:
  - » Rejecting strategy
  - » Repairing strategy
  - » Modifying genetic operators
  - » Penalty strategy
  - » Superiority of Feasible Solutions
  - » Epsilon Constraint handling method

Rejecting Strategy

- All infeasible solutions are discarded.
- It works reasonably well if the feasible region is convex.
- If the feasible region is fragmented, then the method may not yield good results.
- This method does not allow solutions to cross thro’ infeasible region to another feasible region.
- Likely to be highly inefficient if a large number of offpsirng are discarded.

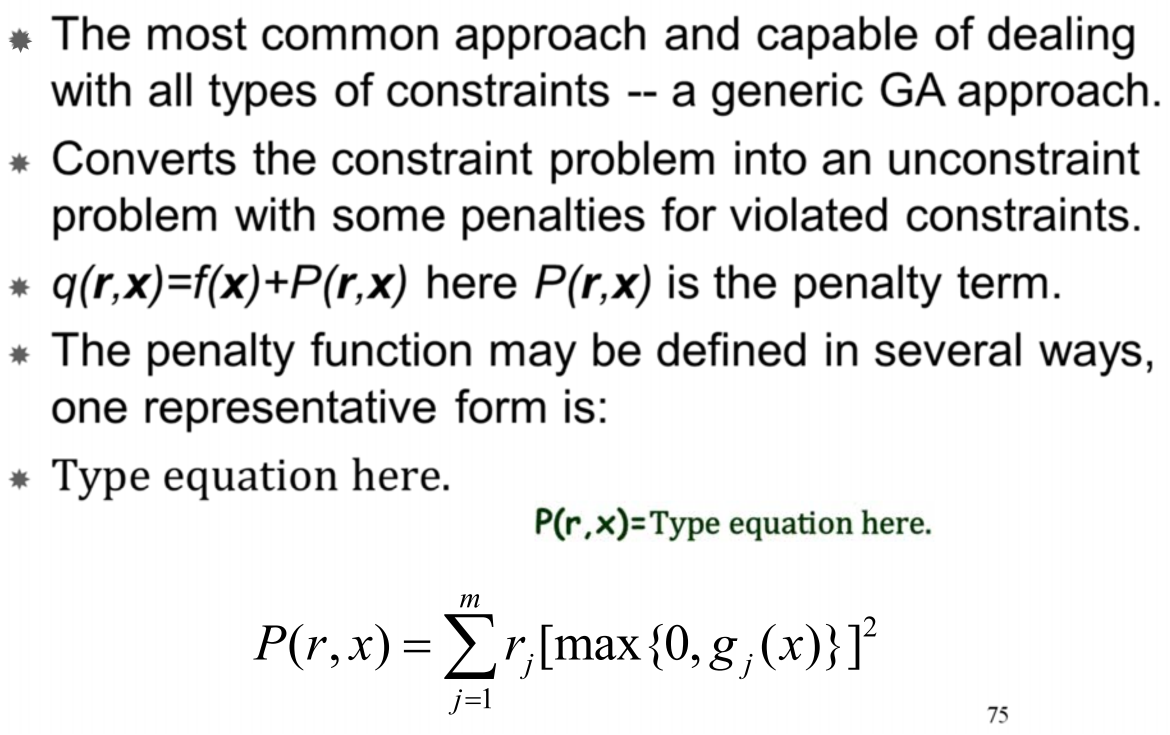
Repairing Strategy

- Infeasible offsprings are corrected to make them satisfy all the constraints.
- If it is possible to repair using some deterministic rules, then this method gives good results.
- Problem dependent rules have to be developed.
- Sometimes repairing process may be too complex.

Modifying Genetic Operators

- Develop genetic operators to give only valid offsprings satisfying all constraints.
- Special genetic operators have to be developed for every problem, i.e. problem dependent.
- All these 3 methods confine the search to within the feasible regions only. Sometimes, it is advantageous to allow the search to consider infeasible solutions too during the intermediate generations.

Penalty Function Approach



- The most common approach and capable of dealing with all types of constraints -- a generic GA approach.
- Converts the constraint problem into an unconstraint problem with some penalties for violated constraints.
- *q(**r**,**x**)=f(**x**)+P(**r**,**x**)* here *P(**r**,**x**)* is the penalty term.
- The penalty function may be defined in several ways, one representative form is:
- Type equation here.

P(r,x)=Type equation here.

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

Penalty Function Approach

- The parameters r_j's to be chosen.
- It can be set to reflect the degree of constraint violation and/or importance of the constraints.
- Usually set r_j's to small values initially.
- Then gradually increase them.
- This approach allows the GA to search feasible and infeasible regions initially and in the end with large r_j's, we force the GA to give valid solutions too.

Example - Penalty Function

minimse $f(x) = (x_1 - 2)^2 + (x_2 - 1)^2$

subject to $g_1(x) = x_1 - 2x_2 + 1 = 0$

$$g_2(x) = \frac{x_1^2}{4} - x_2^2 + 1 \leq 0$$

Upper and lower bounds for both variables are 0 and 3. Minimum of 0.3125 at (1.5, 1.25). Set both penalty weights Equal at 1 initially, then increase by a factor of 10 after a Specified number of generations.

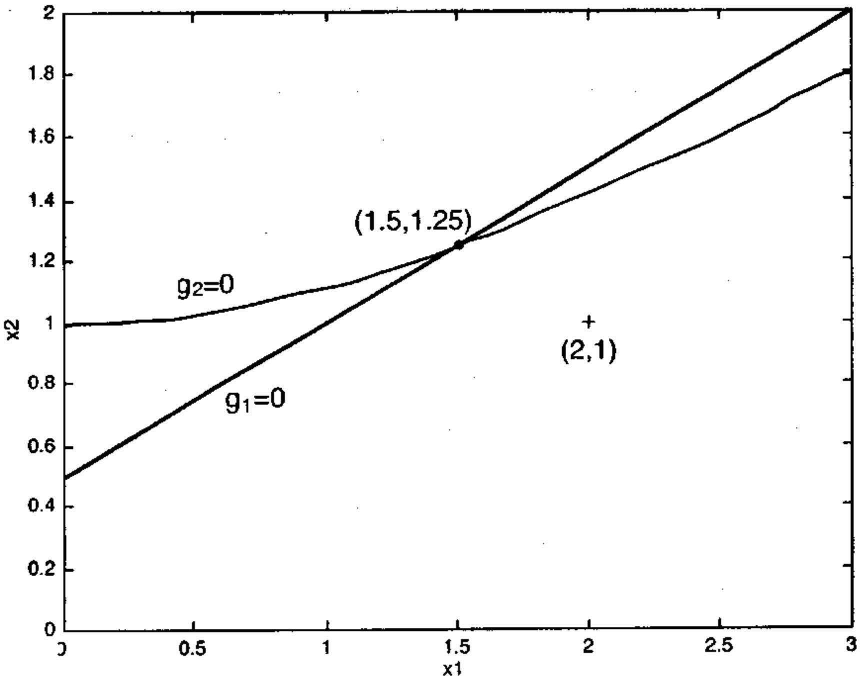


Figure 4.2 The constrained optimisation problem

Example - Penalty Function

- $P(r,x)=r_1[g_1(x)]^2 + r_2[max\{0,g_2(x)\}]^2$
- *Q(r,x)=C - f(x) - P(r,x), C* can be set to a value to ensure *Q(r,x)* is positive. This is required if we use SGA to solve this problem.
- The report function has to give the objective value and the constraints separately, instead of the overall fitness values.

Constraint Handling in EAs

- **Superiority of Feasible Method**

  Among X_i and X_j, X_i is regarded superior to X_j if :

  - Both infeasible & $\upsilon(X_i) < \upsilon(X_j)$

    push infeasible solutions to feasible region

  - Both feasible & $f(X_i) < f(X_j)$ (minimization problems)

    improves overall solution

  - X_i - feasible & X_j – infeasible

Constraint Handling in EAs

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 θth 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)

- Although EAs can treat optimization problems as black-boxes (e.g., academic benchmarks), evidences showing that the exploitation of specific problem domain knowledge can improve the problem solving efficiency.
- Technically, optimization can be viewed a process that an algorithm act on a problem. To promote this process, we can
  - Enhance the capability of optimization algorithms,
  - Make use of the domain knowledge hidden in the problem to reduce its complexity.
- We may think
  - Whether there exists general domain knowledge?
  - How to use such knowledge?

Guohua Wu , Witold Pedrycz, P. N. Suganthan, R. Mallipeddi, "A Variable Reduction Strategy for Evolutionary Algorithms Handling Equality Constraints," *Applied soft computing*, 37 (2015): 774-786

VRS: Motivation

- We utilize the domain knowledge of equality optimal conditions (EOCs) of optimization problems.
  - EOCs are expressed by equation systems;
  - EOCs have to be satisfied for optimal solutions;
  - EOCs are necessary conditions;
  - EOCs are general (e.g. equality constraints in constrained optimization and first derivative equals to zero in unconstrained optimization problem with fist-order derivative).
- Equality constraints are much harder to be completely satisfied when an EA is taken as the optimizer.
- The equality constraints of constrained optimization problems (COPs) are treated as EOCs to reduce variables and eliminate equality constraints.

VRS: Motivation

- In general, the constrained problems can be transformed into the following form: $f(\mathbf{x}), \mathbf{x} = [x_1, x_2, ..., x_D]$
- Minimize $g_i(\mathbf{x}) \le 0, i = 1, ..., q$

subjected to: $h_j(\mathbf{x}) = 0, j = q+1, ..., m$

q: number of inequality constraints & *m-q* : the number of equality constraints.

- When using EAs to solve COP, the equality constraints are converted into inequality constraints as: $|h_j(\mathbf{x})| - \varepsilon \le 0$
- To obtain highly feasible solutions, we need small ε. However, evidences show that a too small ε makes it harder to find feasible solutions, let alone high-quality ones.

VRS: Implementation

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

From $h_j(X)=0$ $(1 \le j \le m)$, if we can obtain a relationship

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

x_k can be actually calculated by relationship $R_{k,j}$ and the values of variables in $\{x_l \mid l \in \Omega_j, l \ne k\}$

Moreover, equality constraint $h_j(X)$ is always satisfied.

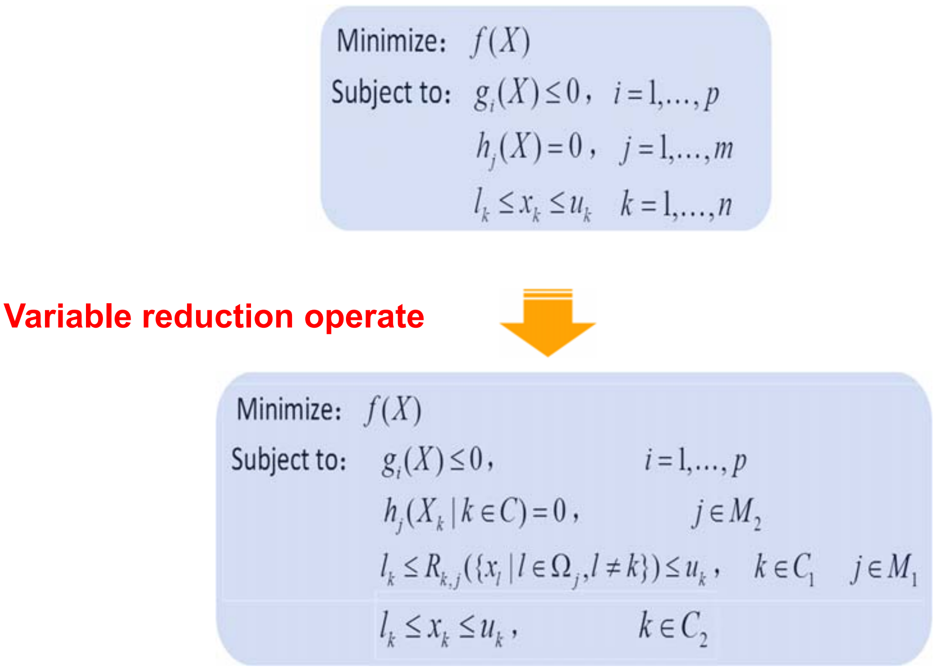
As a result, both $h_j(X)$ and x_k are eliminated.

VRS: Implementation

- **Some essential concepts:**
  - **Core variable(s):** The variable(s) used to represent other variables in terms of the variable relationships in equality constraints.
  - **Reduced variable(s):** The variable(s) expressed and calculated by core variables.
  - **Eliminated equality constraint(s):** The equality constraint(s) eliminated along with the reduction of variables due to full satisfaction by all solutions.

The aim of the variable reduction strategy is to find a set of core variables with minimum cardinality, such that maximum number of equality constraints and variables are reduced.

VRS: Implementation



Minimize: $f(X)$

Subject to: $g_i(X) \le 0$, $i = 1, \dots, p$

$h_j(X) = 0$, $j = 1, \dots, m$

$l_k \le x_k \le u_k$ $k = 1, \dots, n$

Variable reduction operate

Minimize: $f(X)$

Subject to: $g_i(X) \le 0$, $i = 1, \dots, p$

$h_j(X_k \mid k \in C) = 0$, $j \in M_2$

$l_k \le R_{k,j}(\{x_l \mid l \in \Omega_j, l \ne k\}) \le u_k$, $k \in C_1$ $j \in M_1$

$l_k \le x_k \le u_k$, $k \in C_2$

VRS: Implementation

- Naïve example, considering:

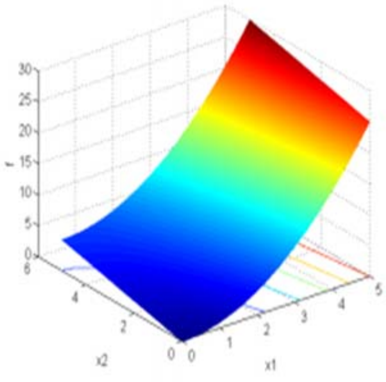
$$\min x_1^2 + x_2^2$$
$$x_1 + x_2 = 2$$
$$0 \le x_1 \le 5, 0 \le x_2 \le 5$$

- We can obtain the variable relationship $x_2 = 2 - x_1$ and substitute it into original problem, then we get

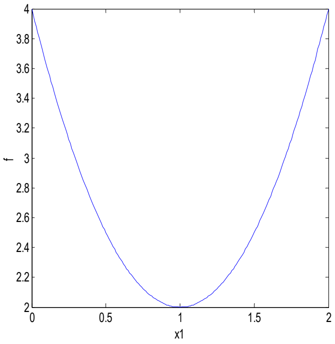
$$\min 2x_1^2 - 4x_1 + 4$$
$$0 \le x_1 \le 2$$

VRS: Implementation

- Solution space before and after the variable reduction process



(a) Original solution space



(b) Solution after VRS.

VRS: Implementation

- Since equality constraints can be nonlinear and very complex, it is still an open to design a unified method for variable reduction.
- 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 only by one operator (e.g. x^n,$\cos x$, $\ln x$, a^x).