



**UIT**

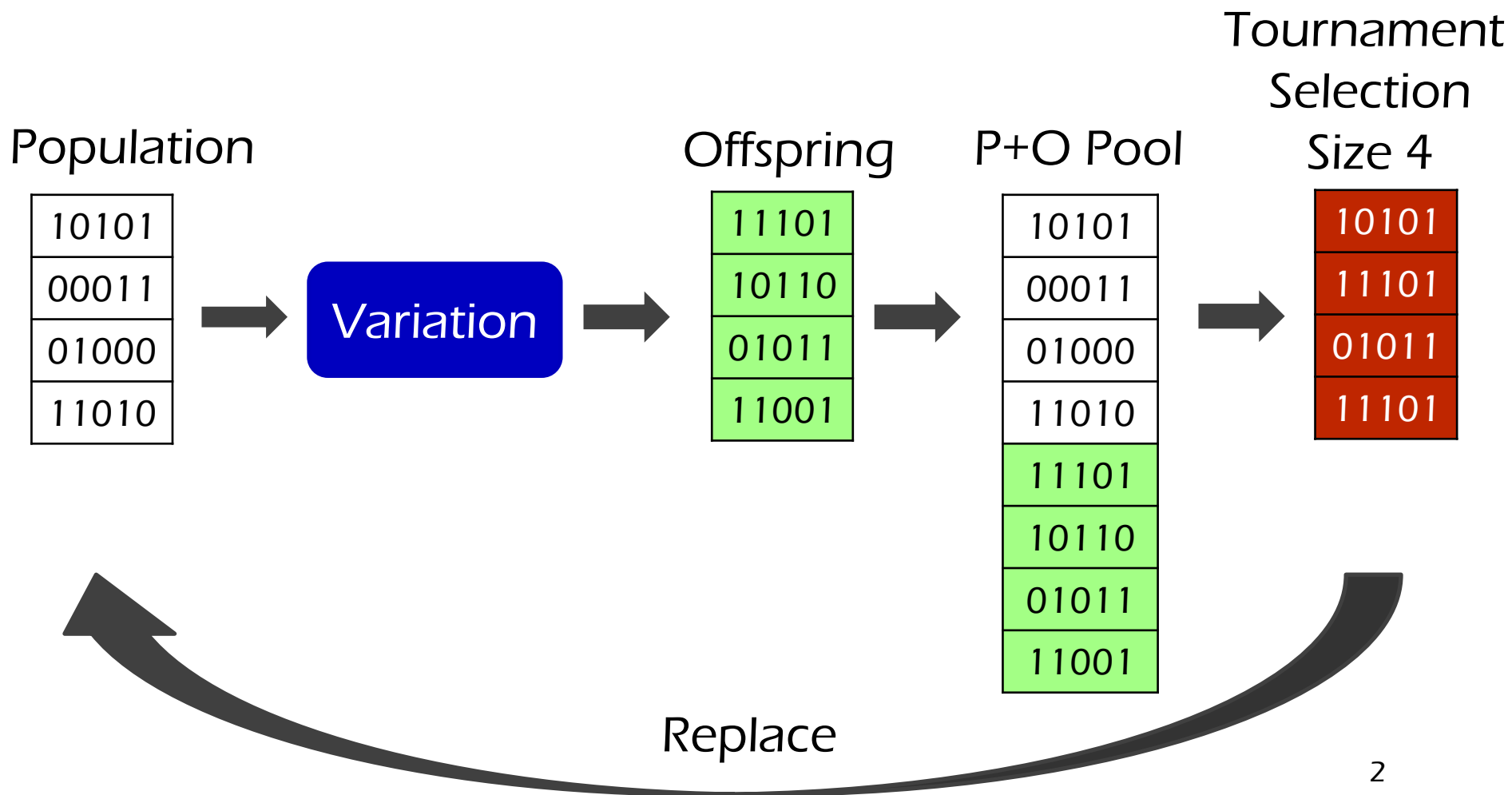
TRƯỜNG ĐẠI HỌC  
CÔNG NGHỆ THÔNG TIN

---

# **DIFFERENTIAL EVOLUTION & PARTICLE SWARM OPTIMIZATION**



# Simple Genetic Algorithm (sGA)



# Simple Genetic Algorithm (sGA)

Giải thuật di truyền đơn giản (Simple Genetic Algorithm– sGA) có thể giải bài toán sau không?

Tối thiểu hoá

$$f(x_0, x_1) = (x_0 - 0.5)^2 + (x_1 - 0.5)^2$$
$$x_0, x_1 \in [0, 1] \subset \mathbb{R}$$

# Simple Genetic Algorithm (sGA)

Tối thiểu hoá

$$f(x_0, x_1) = (x_0 - 0.5)^2 + (x_1 - 0.5)^2$$
$$x_0, x_1 \in [0, 1] \subset \mathbb{R}$$

$x_0$	$x_1$
0.25	0.80
0.40	0.97

$$f(0.40, 0.80) = 0.1$$

# Simple Genetic Algorithm (sGA)

Tối thiểu hoá

$$f(x_0, x_1) = (x_0 - 0.5)^2 + (x_1 - 0.5)^2$$
$$x_0, x_1 \in [0, 1] \subset \mathbb{R}$$

$x_0$	$x_1$
0.25	0.80
0.40	0.97
0.12	0.34
0.67	0.72

$$f(0.40, 0.34) = 0.0356$$



# Simple Genetic Algorithm (sGA)

Tối thiểu hoá

$$f(x_0, x_1) = (x_0 - 0.5)^2 + (x_1 - 0.5)^2$$

$$x_0, x_1 \in [0,1] \subset \mathbb{R}$$

$x_0$	$x_1$
0.25	0.80
0.40	0.97
0.12	0.34
0.67	0.72
0.55	0.81
0.52	0.04
0.85	0.23
0.76	0.65

$$f(0.40, 0.34) = 0.0229$$

# Simple Genetic Algorithm (sGA)

---

sGA có thể giải quyết các bài toán với các  
**biến số thực** không?

# sGA – Simple Binary Encoding

Một chuỗi nhị phân  $(a_i)$  với  $q$  bit

$$(a_i), a_i \in \{0,1\}, i = 0,1, \dots, q-1$$

có thể biểu diễn một biến số thực  $x \in [b_L, b_U]$ , và giá trị của  $x$  là

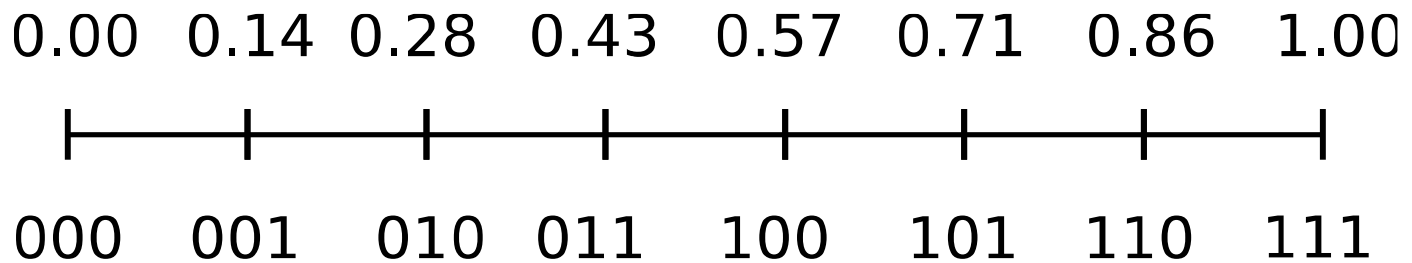
$$x = b_L + \frac{b_U - b_L}{2^q - 1} \cdot \sum_{i=0}^{q-1} a_i 2^i$$



# sGA – Simple Binary Encoding

$$x = b_L + \frac{b_U - b_L}{2^q - 1} \cdot \sum_{i=0}^{q-1} a_i 2^i$$

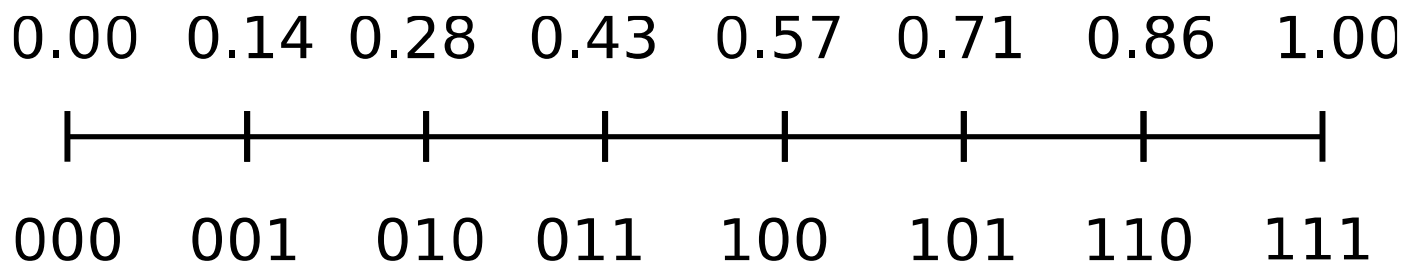
Ví dụ: Sử dụng  $q=3$  bit để biểu diễn một biến số thực  $x \in [0,1]$





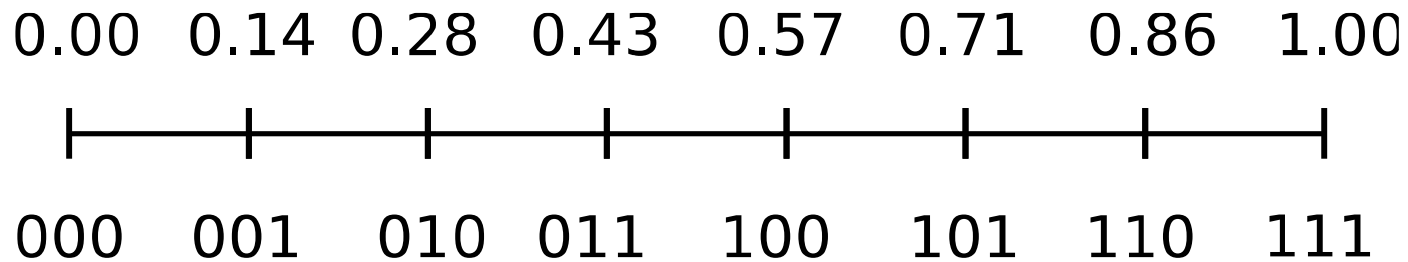
# sGA – Simple Binary Encoding

Có vấn đề gì với cách biểu diễn nhị phân này?



# sGA – Simple Binary Encoding

Có vấn đề gì với cách biểu diễn nhị phân này?



- Thuật toán phải xử lý nhiều biến hơn (số biến tăng  $q$  lần).
- Các giá trị gần nhau có thể có cách biểu diễn khác hẳn nhau. Ví dụ: 0.43 (011) và 0.57 (100).
- ...



**UIT**  
TRƯỜNG ĐẠI HỌC  
CÔNG NGHỆ THÔNG TIN

---

# DIFFERENTIAL EVOLUTION

# DE – Giới thiệu

- Differential Evolution (DE) được đề xuất bởi Storn và Price vào năm 1995.
- Ý tưởng chính: Phép biến đổi được thực hiện bằng cách cộng vector khác biệt giữa hai cá thể được chọn ngẫu nhiên vào một cá thể thứ ba được chọn ngẫu nhiên khác.

# DE – Các ký hiệu

- Quần thể  $P_{x,g}$  ở thế hệ  $g$  có chứa  $N$  vector  $\mathbf{x}_{i,g}$   
 $P_{x,g} = (\mathbf{x}_{i,g}), i = 0, 1, \dots, N - 1, g = 0, 1, \dots, g_{\max}$   
 $\mathbf{x}_{i,g} = (x_{j,i,g}), j = 0, 1, \dots, D - 1$   
 $x_{j,i,g} \in [b_{j,L}, b_{j,U}] \subset \mathbb{R}$
- $N$ : kích thước quần thể.
- $D$ : số biến – số tham số.
- $g_{\max}$ : số thế hệ tối đa.
- $[b_{j,L}, b_{j,U}]$ : miền giá trị của biến thứ  $j$

# DE – Khởi tạo quần thể

$$g = 0$$

$$P_{x,0} = (\mathbf{x}_{i,0})$$

$$x_{j,i,0} \in \text{rand}_j(0,1) \times (b_{j,U} - b_{j,L}) + b_{j,L}$$

$$0 \leq \text{rand}_j(0,1) \leq 1$$

# DE – Tạo ra vector đột biến

Quần thể đột biến (mutant population)  $P_{v,g}$  chứa N vector đột biến (mutant vector)  $\mathbf{v}_{i,g}$

$$P_{v,g} = (\mathbf{v}_{i,g}), i = 0, 1, \dots, N - 1, g = 0, 1, \dots, g_{\max}$$

$$\mathbf{v}_{i,g} = (v_{j,i,g}), j = 0, 1, \dots, D - 1$$

F: hệ số scale, thường thì  $F \in (0, 1)$

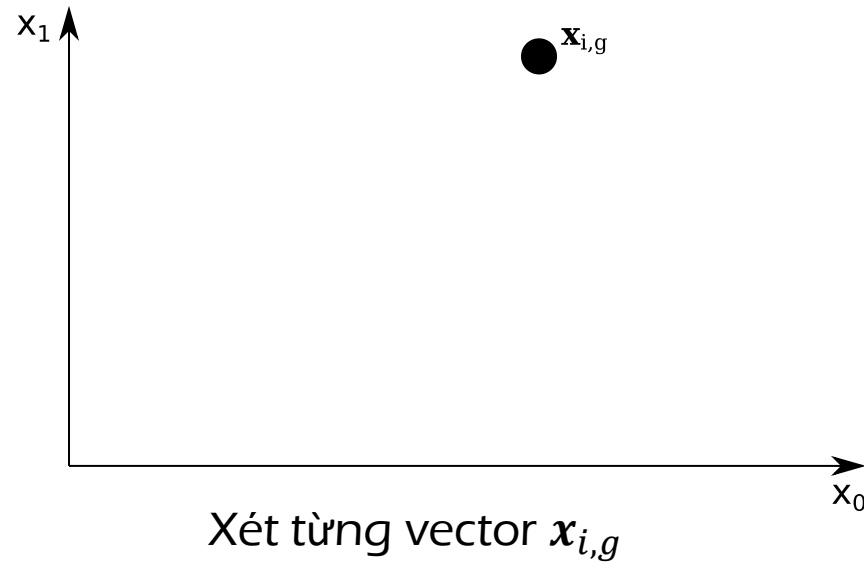
Với mỗi vector  $\mathbf{v}_{i,g}$ :

- Chọn ngẫu nhiên  $r_0, r_1, r_2$  từ  $\{0, 1, \dots, N - 1\} \setminus \{i\}$
- $\mathbf{x}_{r_0,g}, \mathbf{x}_{r_1,g}, \mathbf{x}_{r_2,g} \in P_{x,g}$

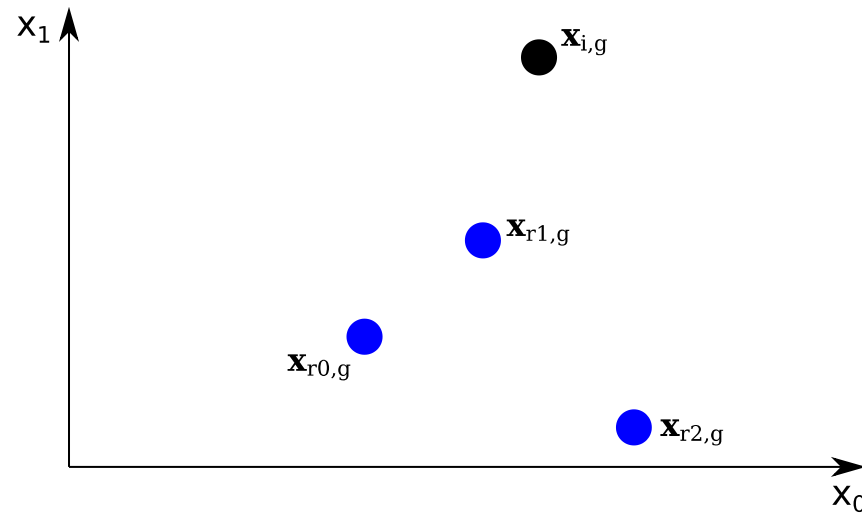
$$\mathbf{v}_{i,g} = \mathbf{x}_{r_0,g} + F \times (\mathbf{x}_{r_1,g} - \mathbf{x}_{r_2,g})$$



# DE – Tạo ra vector đột biến



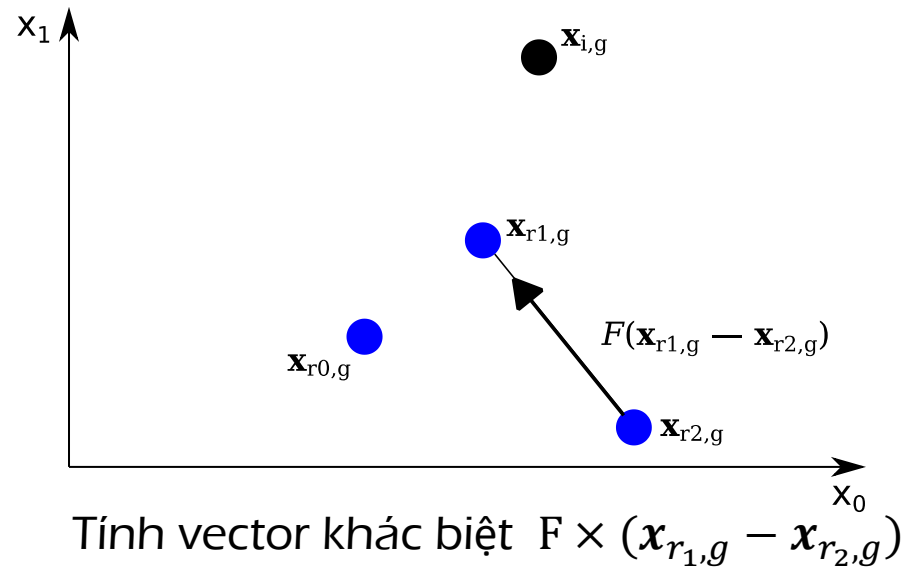
# DE – Tạo ra vector đột biến



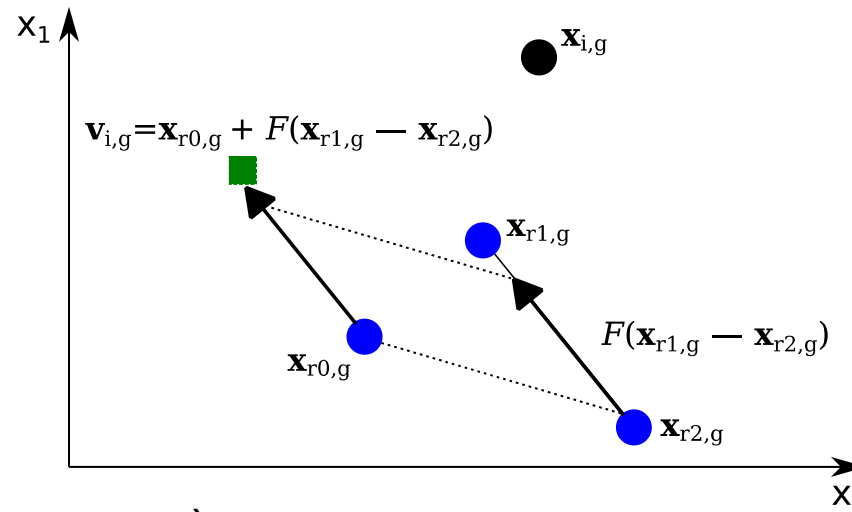
Chọn ngẫu nhiên 3 vector

$x_{r0,g}, x_{r1,g}, x_{r2,g}$

# DE – Tạo ra vector đột biến



# DE – Tạo ra vector đột biến



Tạo ra vector đột biến  $\mathbf{v}_{i,g}$  bằng cách cộng vector khác biệt vào vector  $\mathbf{x}_{r0,g}$

# DE – Lai ghép

Quần thể thử nghiệm (trial population)  $P_{u,g}$  chứa N vector thử nghiệm (trial vector)  $\mathbf{u}_{i,g}$

$$P_{u,g} = (\mathbf{u}_{i,g}), i = 0, 1, \dots, N - 1, g = 0, 1, \dots, g_{\max}$$

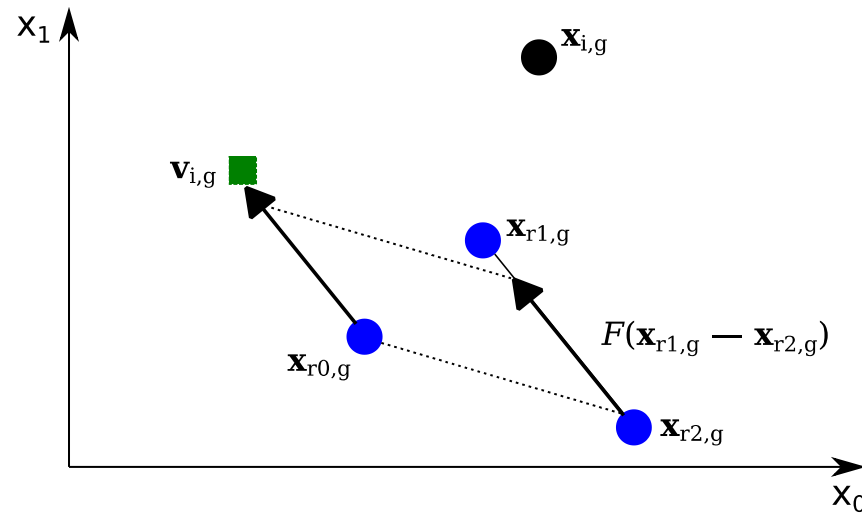
$$\mathbf{u}_{i,g} = (u_{j,i,g}), j = 0, 1, \dots, D - 1$$

Cr: Xác suất lai ghép. Với mỗi vector  $\mathbf{u}_{i,g}$ :

- $j_{\text{rand}}$ : Một chỉ số ngẫu nhiên được chọn từ  $\{0, 1, \dots, D - 1\}$
- Vector đích (target vector)  $\mathbf{x}_{i,g} \in P_{x,g}$

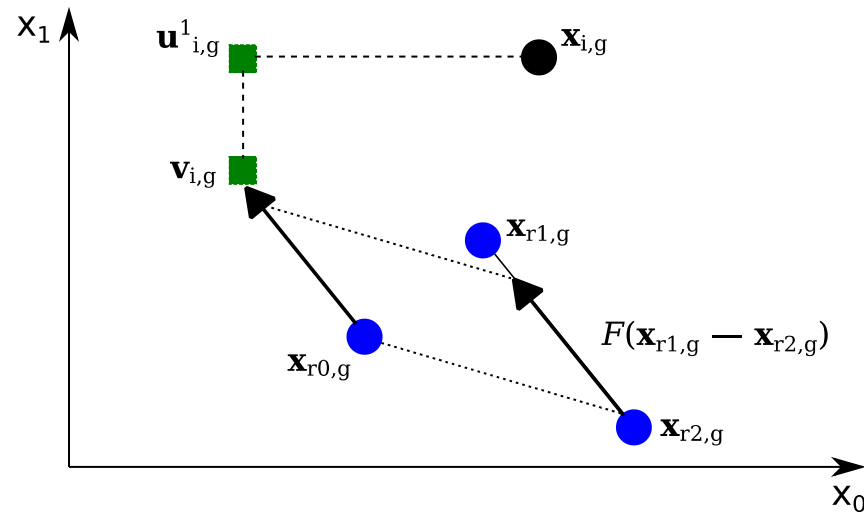
$$u_{j,i,g} = \begin{cases} v_{j,i,g} & \text{nếu } \text{rand}_j(0,1) \leq \text{Cr} \text{ hoặc } j = j_{\text{rand}} \\ x_{j,i,g} & \text{trong trường hợp ngược lại} \end{cases}$$

# DE – Lai ghép



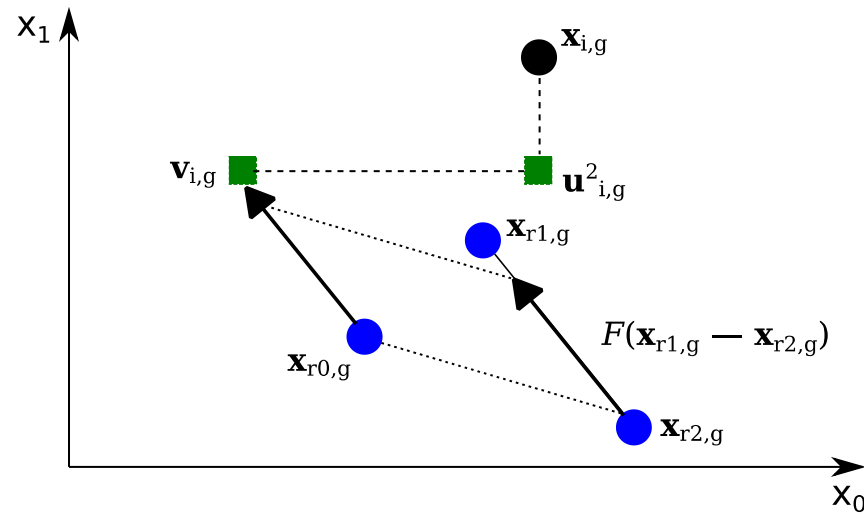
Thực hiện phép lai ghép giữa vector  $\mathbf{x}_{i,g}$  và vector đột biến  $\mathbf{v}_{i,g}$

# DE – Lai ghép



$\mathbf{u}_{i,g}^1$ :  $x_1$  lấy từ vector  $\mathbf{x}_{i,g}$  và  $x_0$  lấy từ vector đột biến  $\mathbf{v}_{i,g}$

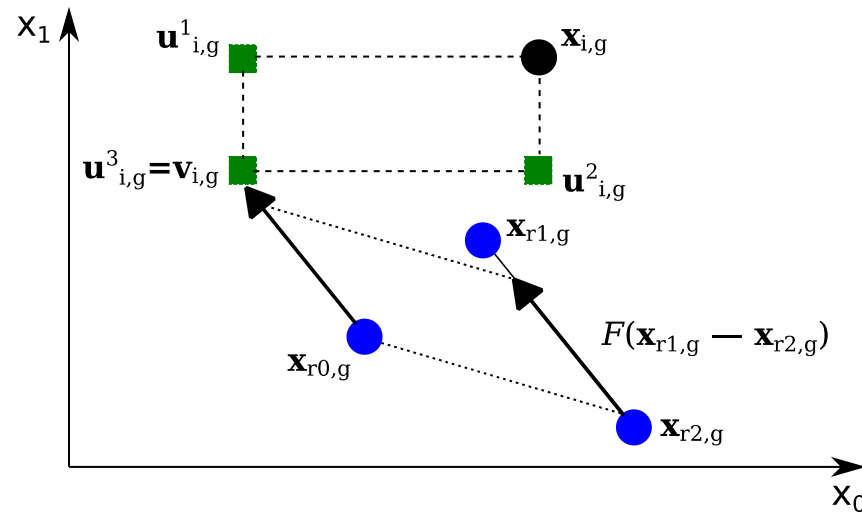
# DE – Lai ghép



$u^2_{i,g}$ :  $x_0$  lấy từ vector  $x_{i,g}$  và  $x_1$  lấy từ vector đột biến  $v_{i,g}$



# DE – Lai ghép



$u^3_{i,g}$ : Cả  $x_0$  và  $x_1$  lấy từ vector đột biến. Do đó,  $u^3_{i,g} = v_{i,g}$

# DE – Chọn lọc

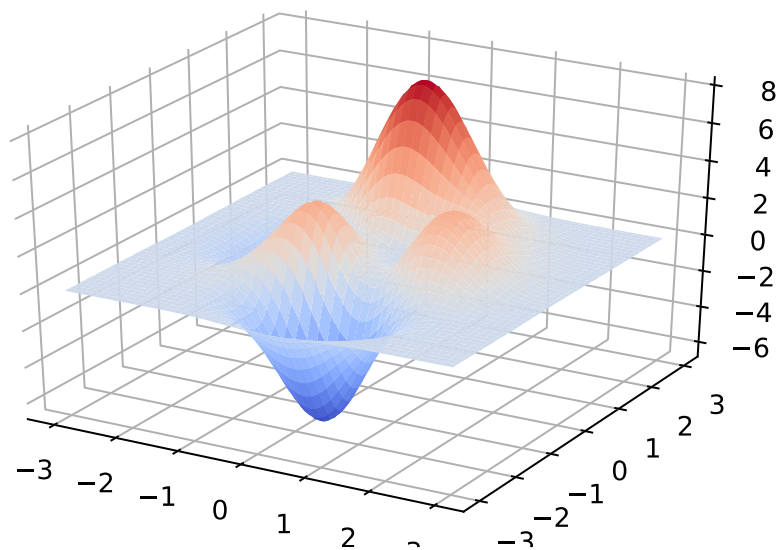
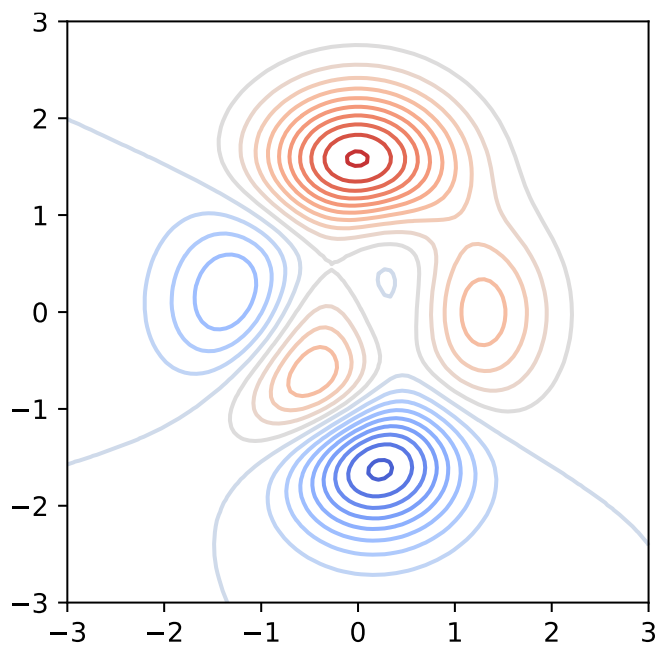
Quần thể trong thế hệ tiếp theo  $P_{x,g+1}$  chứa N vector  $\mathbf{x}_{i,g+1}$

$$P_{x,g+1} = (\mathbf{x}_{i,g+1}), i = 0, 1, \dots, N - 1, g = 0, 1, \dots, g_{\max}$$

$$\mathbf{x}_{i,g+1} = \begin{cases} \mathbf{u}_{i,g} & \text{nếu } f(\mathbf{u}_{i,g}) \text{ tốt hơn } f(\mathbf{x}_{i,g}) \\ \mathbf{x}_{i,g} & \text{trong trường hợp ngược lại} \end{cases}$$

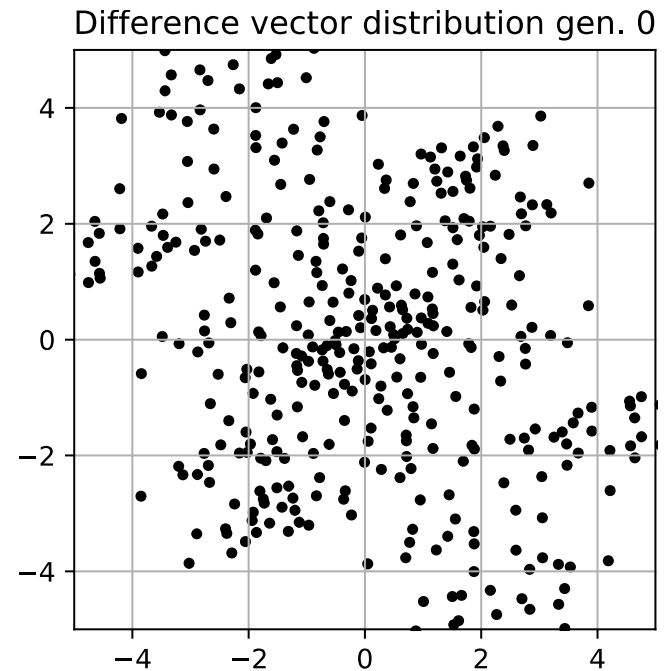
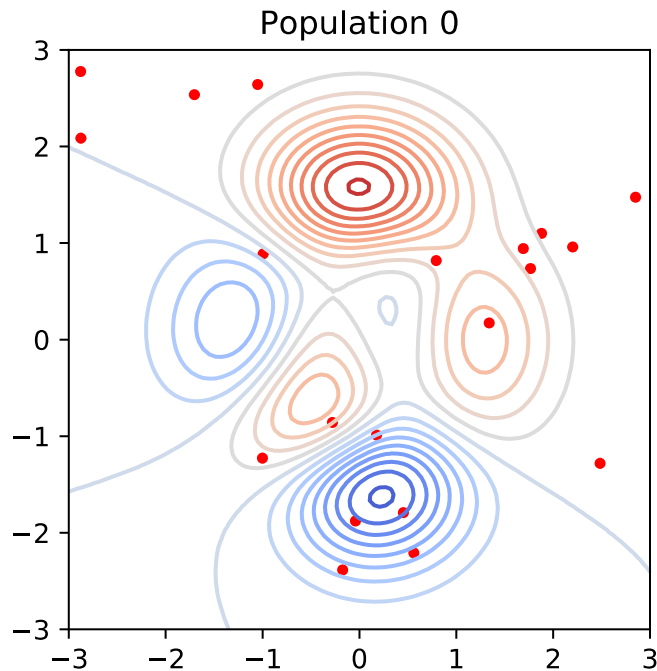


# DE – Ví dụ

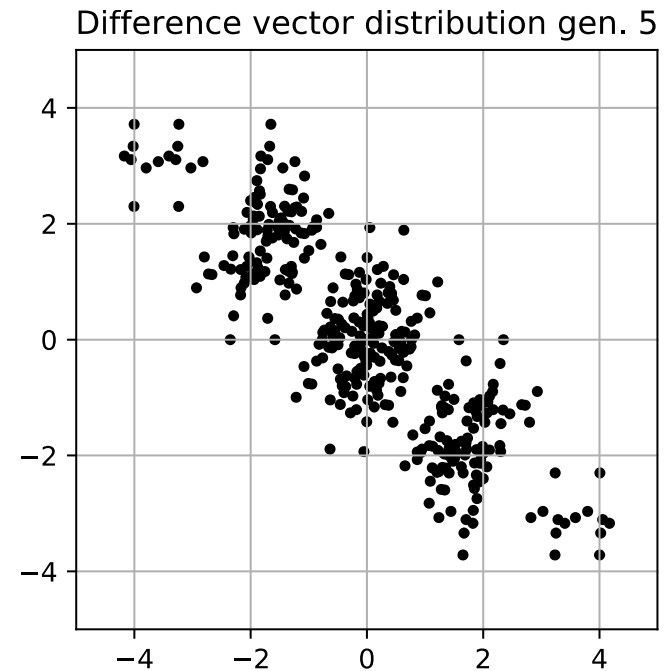
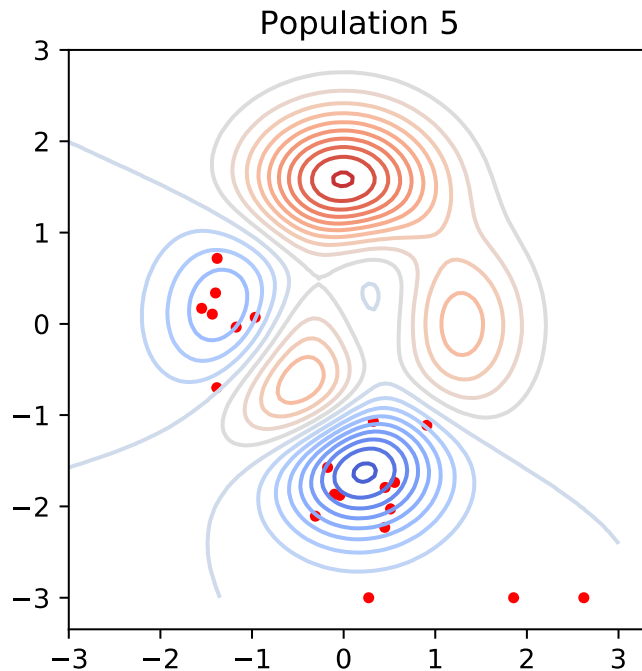




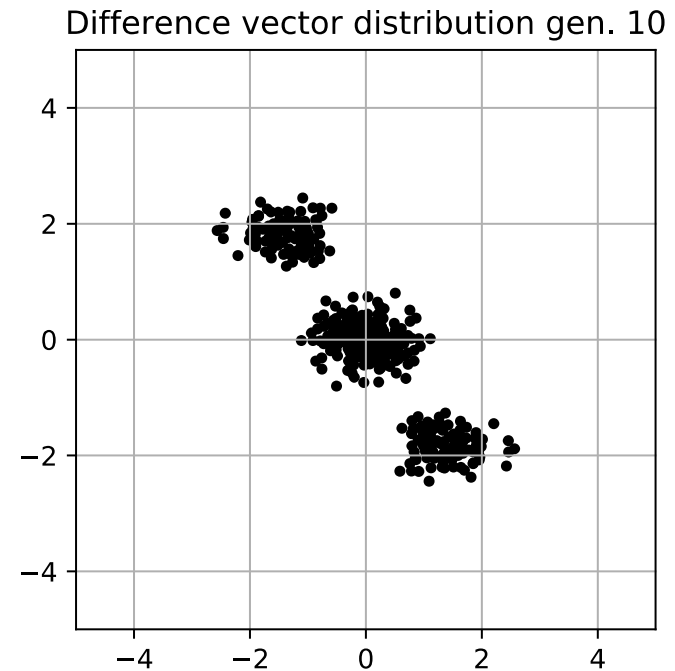
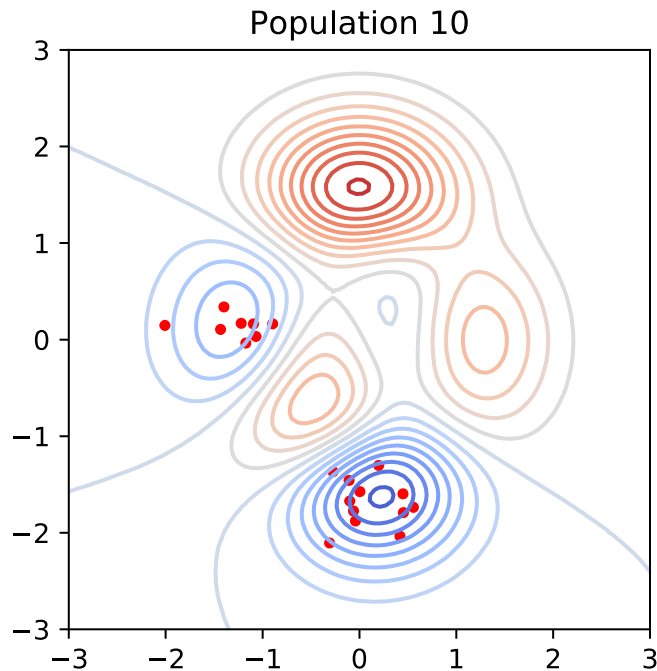
# DE – Ví dụ



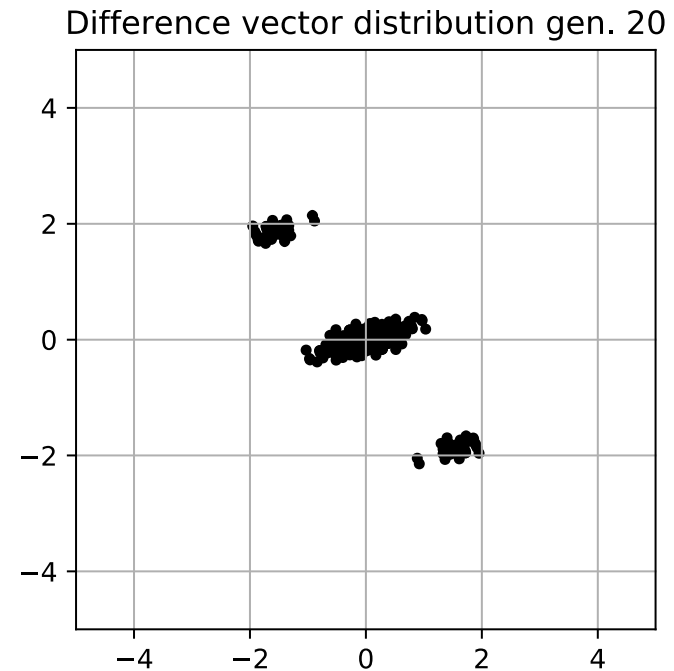
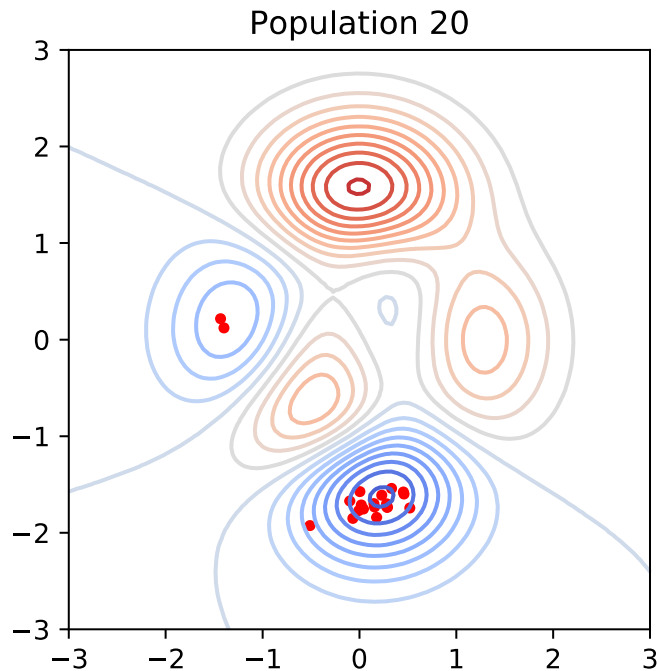
# DE – Ví dụ



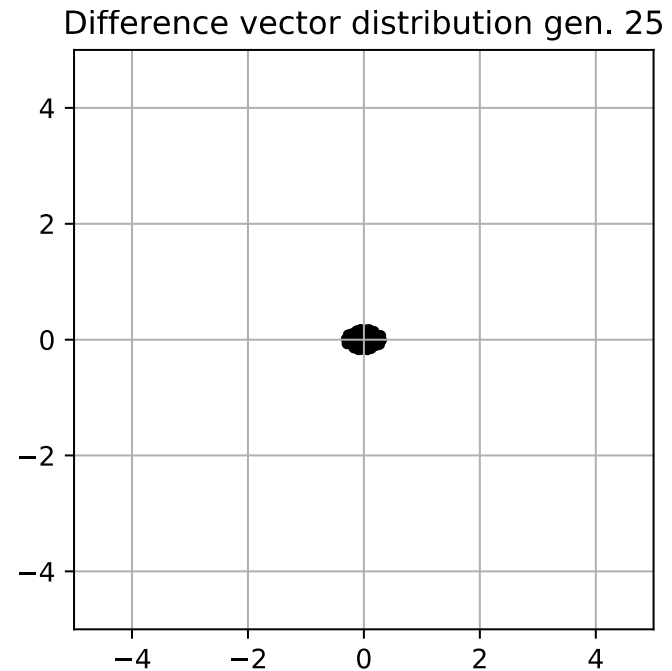
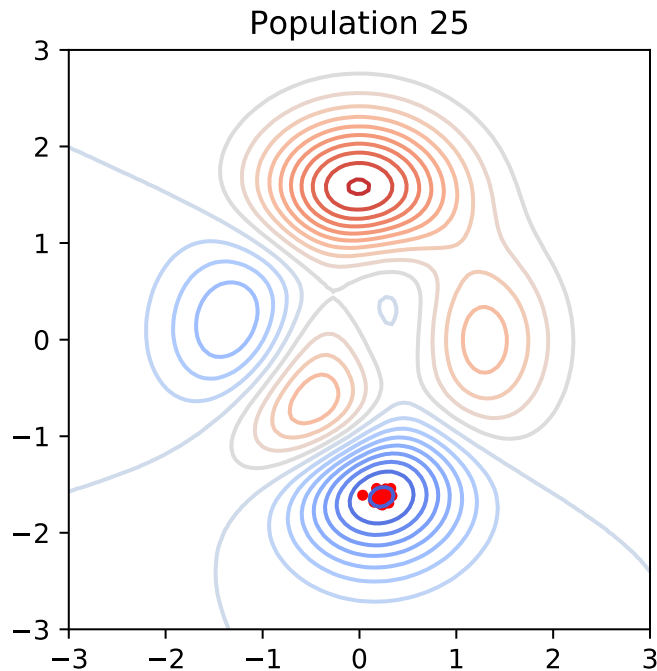
# DE – Ví dụ



# DE – Ví dụ



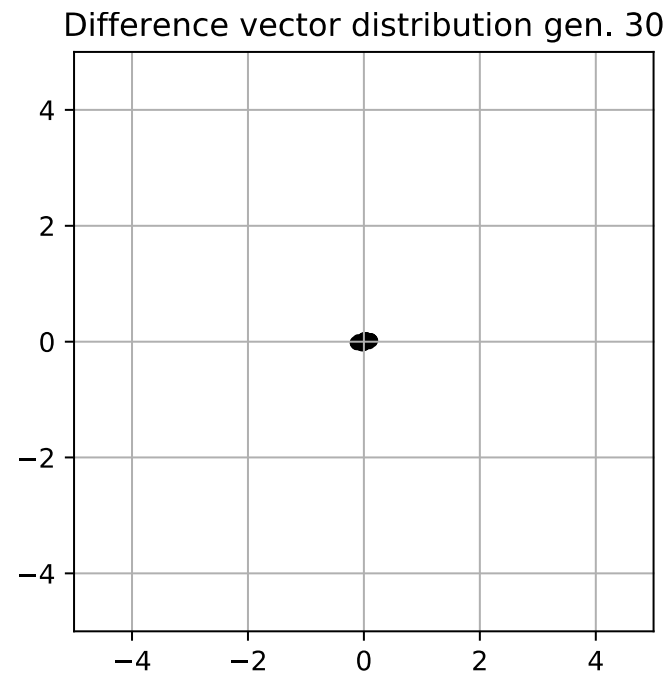
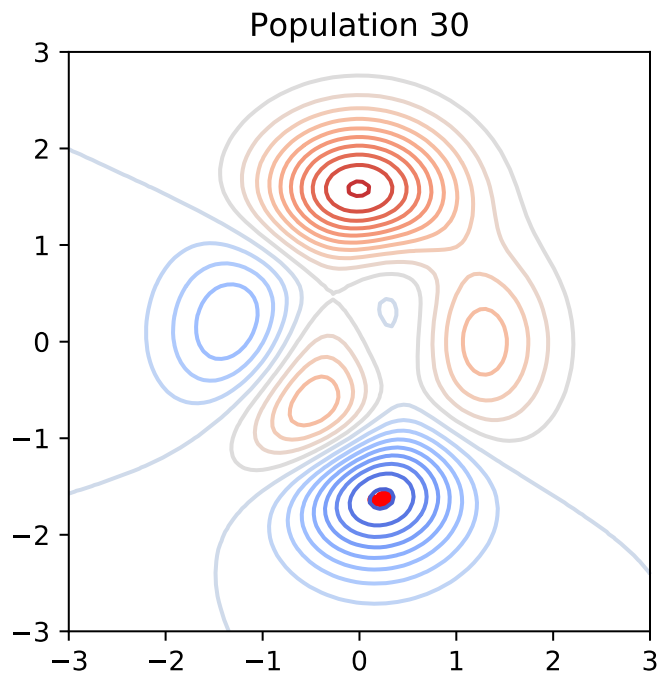
# DE – Ví dụ







# DE – Ví dụ



# DE – Các tham số điều khiển

Hiệu suất của DE bị ảnh hưởng bởi các tham số điều khiển:

- Kích thước quần thể
- Cách các vector  $\mathbf{x}_{r_0,g}$ ,  $\mathbf{x}_{r_1,g}$ ,  $\mathbf{x}_{r_2,g}$  được lựa chọn
- Hệ số scale F
- Cách phép lai ghép được thực hiện
- ...



**UIT**  
TRƯỜNG ĐẠI HỌC  
CÔNG NGHỆ THÔNG TIN

---

# PARTICLE SWARM OPTIMIZATION

# PSO – Giới thiệu

- Những thử nghiệm đầu tiên được thực hiện bởi Kennedy và Eberhart năm 1995.
- Ý tưởng chính: PSO duy trì một bầy đàn (swarm, tương tự như một quần thể population) gồm các phần tử (particle) di chuyển trong không gian tìm kiếm. Mỗi phần tử có cách di chuyển được xác định bởi:
  1. Vị trí hiện tại.
  2. Vị trí tốt nhất từ trước tới giờ.
  3. Vị trí của tốt nhất từ trước tới giờ từng được tìm ra bởi các phần tử trong lân cận.

# PSO – Ký hiệu

- Bầy đàn (quần thể)  $P_{x,g}$  tại thế hệ  $g$  chứa  $N$  phần tử

$\mathbf{x}_{i,g}$

$$P_{x,g} = (\mathbf{x}_{i,g}), i = 0, 1, \dots, N - 1, g = 0, 1, \dots, g_{\max}$$

$$\mathbf{x}_{i,g} = (x_{j,i,g}), j = 0, 1, \dots, D - 1$$

$$x_{j,i,g} \in [b_{j,L}, b_{j,U}] \subset \mathbb{R}$$

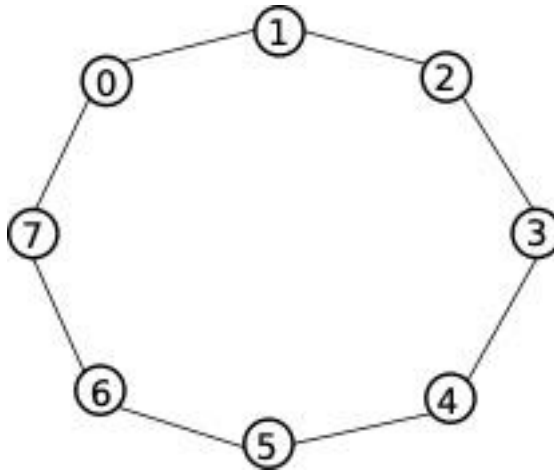
- $N$ : kích thước bầy đàn.
- $D$ : Số biến, số tham số.
- $g_{\max}$ : Số thế hệ tối đa.

# PSO – Ký hiệu

Mỗi phần tử  $i$  có:

- $\mathbf{x}_{i,g}$ : Vị trí hiện tại của phần tử  $i$ .
- $\mathbf{v}_{i,g}$ : Vector vận tốc hiện tại của phần tử  $i$ .
- $\mathbf{y}_{i,g}$ : Vị trí tốt nhất từng được tìm ra bởi phần tử  $i$ .
- $\mathbf{z}_{i,g}$ : Vị trí tốt nhất từng được tìm ra bởi các phần tử trong lân cận  $\mathcal{N}_i$  của phần tử  $i$ .

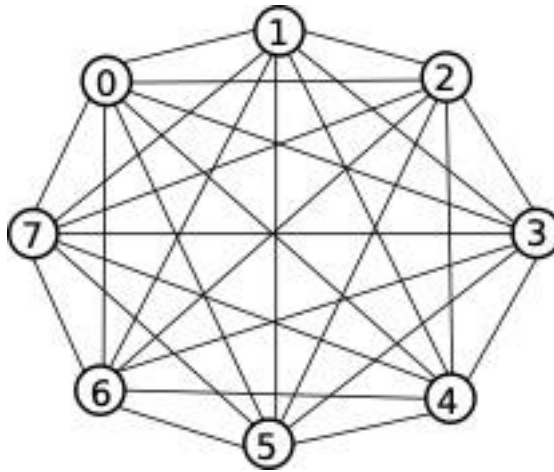
# PSO – Luân cận dạng vòng



Luân cận dạng vòng. Ví dụ  $\mathcal{N}_3 = \{2,3,4\}$ ,  $\mathcal{N}_0 = \{7,0,1\}$

$\mathbf{z}_{i,g}$  là vị trí tốt nhất từng được tìm ra bởi các phần tử trong lân cận của phần tử đang xét (lân cận bao gồm chính phần tử đó và các phần tử liền kề).

# PSO – Lân cận hình sao

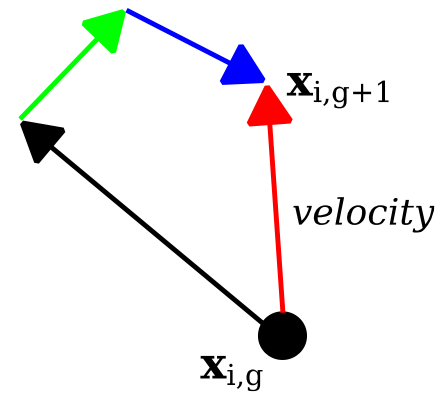
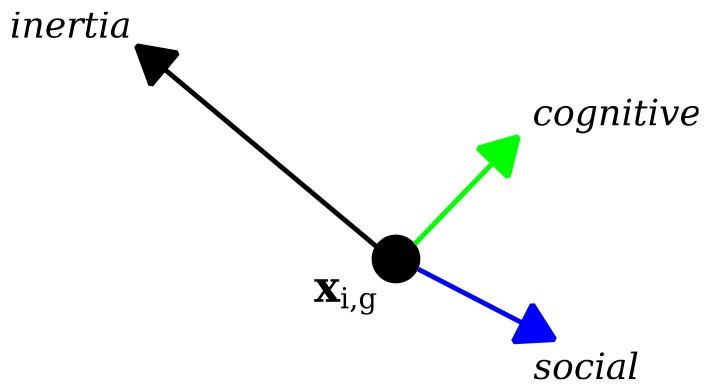


Lân cận hình sao. Với mọi  $i$ ,  $\mathcal{N}_i = \{0,1,2,3,4,5,6,7\}$

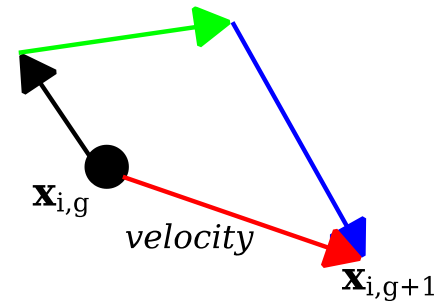
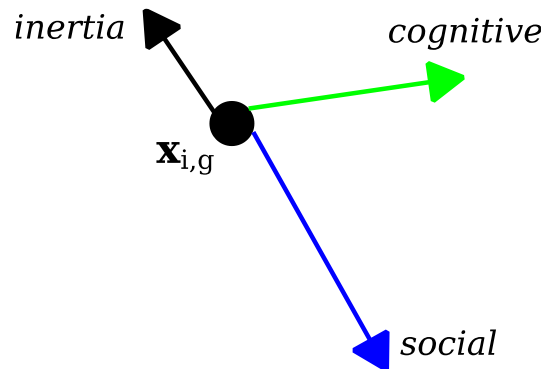
$\mathbf{z}_{i,g}$  là vị trí tốt nhất từng được tìm ra tất cả bầy đàn (quần thể).



# PSO – Cập nhật vị trí



# PSO – Cập nhật vị trí



# PSO – Cập nhật vị trí

Ở mỗi thế hệ, vector vận tốc của phần tử  $i$  được cập nhật:

$$\mathbf{v}_{i,g+1} = w\mathbf{v}_{i,g} + c_1\mathbf{r}_1 \otimes (\mathbf{y}_{i,g} - \mathbf{x}_{i,g}) + c_2\mathbf{r}_2 \otimes (\mathbf{z}_{i,g} - \mathbf{x}_{i,g})$$

- $w$  : Trọng số quán tính.
- $c_1, c_2$ : Hằng số gia tốc (acceleration constants)
- $\mathbf{r}_1, \mathbf{r}_2$ : Vector ngẫu nhiên với các giá trị trong khoảng  $(0, 1)$

Các thành phần của vector vận tốc (velocity vector).

- $w\mathbf{v}_{i,g}$ : Thành phần quán tính (inertia)
- $c_1\mathbf{r}_1 \otimes (\mathbf{y}_{i,g} - \mathbf{x}_{i,g})$ : Thành phần nhận thức (cognitive)
- $c_2\mathbf{r}_2 \otimes (\mathbf{z}_{i,g} - \mathbf{x}_{i,g})$ : Thành phần xã hội (social)

# PSO – Cập nhật vị trí

Ở mỗi thế hệ, vector vận tốc của phần tử  $i$  được cập nhật:

$$\mathbf{v}_{i,g+1} = w\mathbf{v}_{i,g} + c_1\mathbf{r}_1 \otimes (\mathbf{y}_{i,g} - \mathbf{x}_{i,g}) + c_2\mathbf{r}_2 \otimes (\mathbf{z}_{i,g} - \mathbf{x}_{i,g})$$

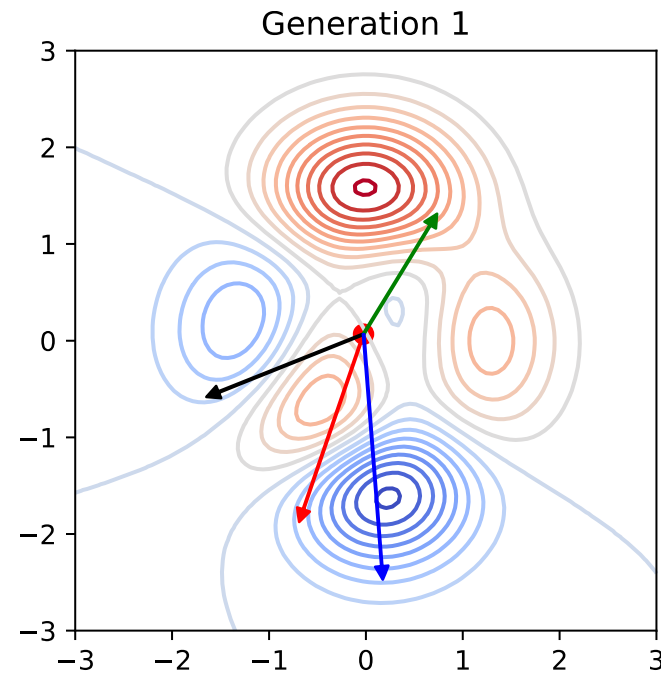
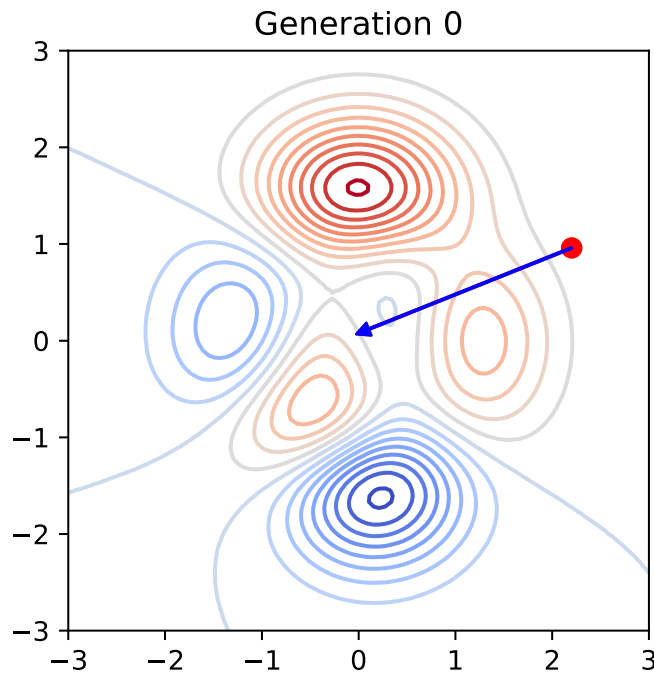
Các thành phần của vector vận tốc (velocity vector).

- $w\mathbf{v}_{i,g}$ : Thành phần quán tính (inertia)
- $c_1\mathbf{r}_1 \otimes (\mathbf{y}_{i,g} - \mathbf{x}_{i,g})$ : Thành phần nhận thức (cognitive)
- $c_2\mathbf{r}_2 \otimes (\mathbf{z}_{i,g} - \mathbf{x}_{i,g})$ : Thành phần xã hội (social)

Vị trí kế tiếp của phần tử  $i$  được cập nhật bởi:

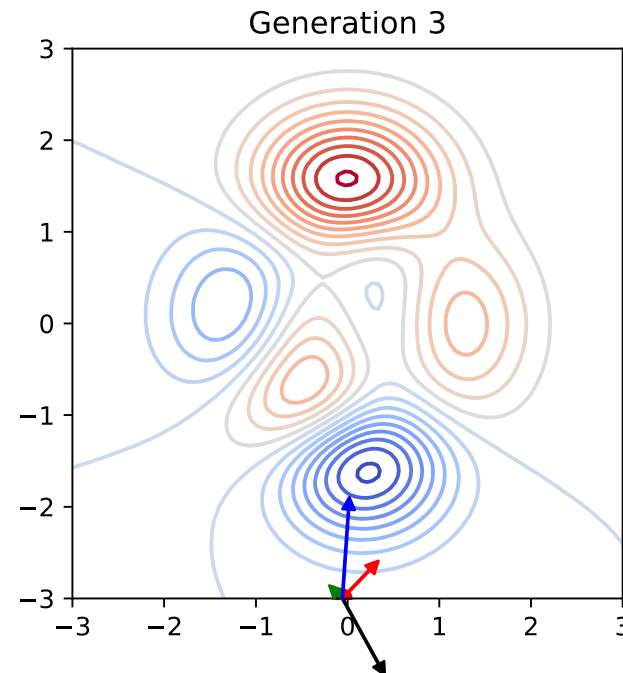
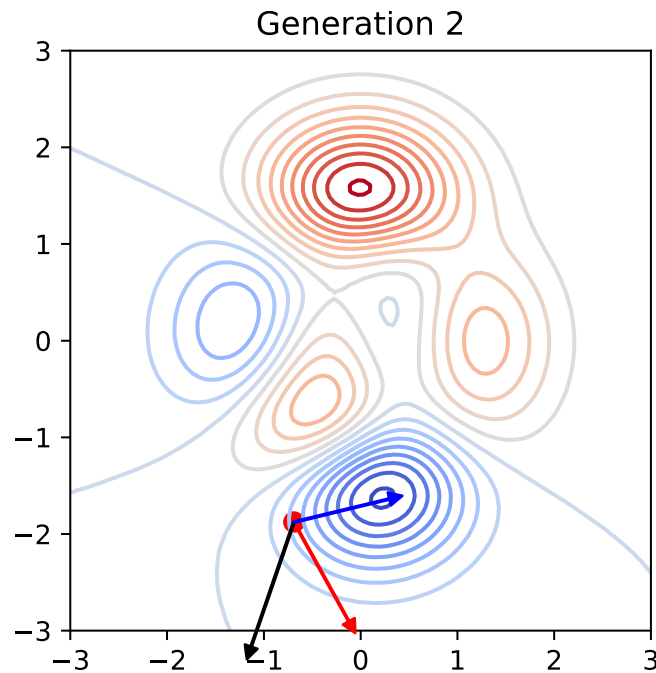
$$\mathbf{x}_{i,g+1} = \mathbf{x}_{i,g} + \mathbf{v}_{i,g+1}$$

# PSO – Ví dụ - Quan sát 1 phần tử



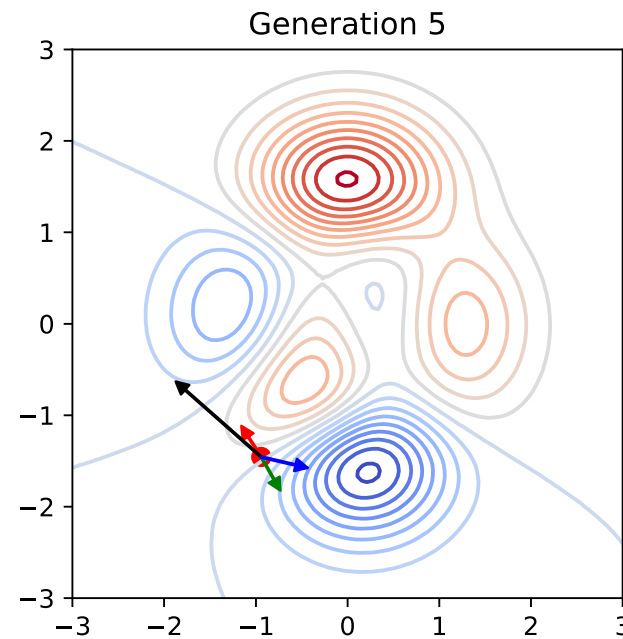
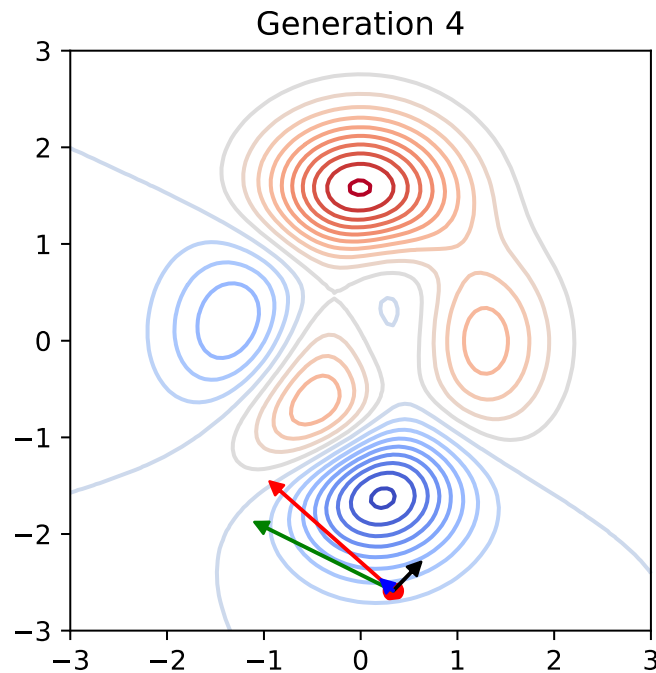
—▶ inertia    —▶ cognitive    —▶ social  
—▶ velocity


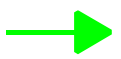


# PSO – Ví dụ - Quan sát 1 phần tử



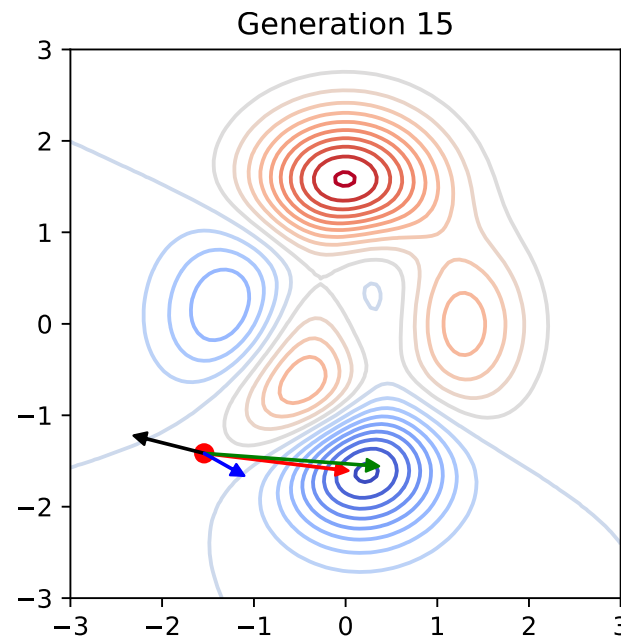
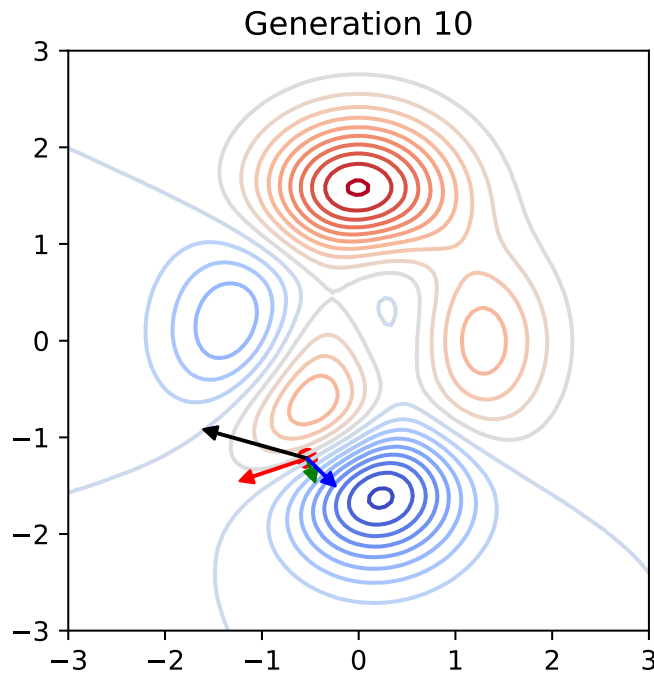
→ *inertia* → *cognitive* → *social*  
→ *velocity*

# PSO – Ví dụ - Quan sát 1 phần tử



 *inertia*
 *cognitive*
 *social*  
 *velocity*

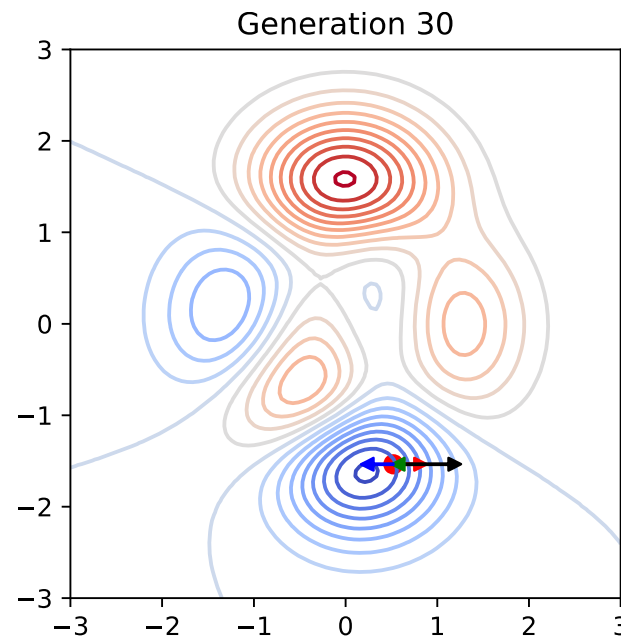
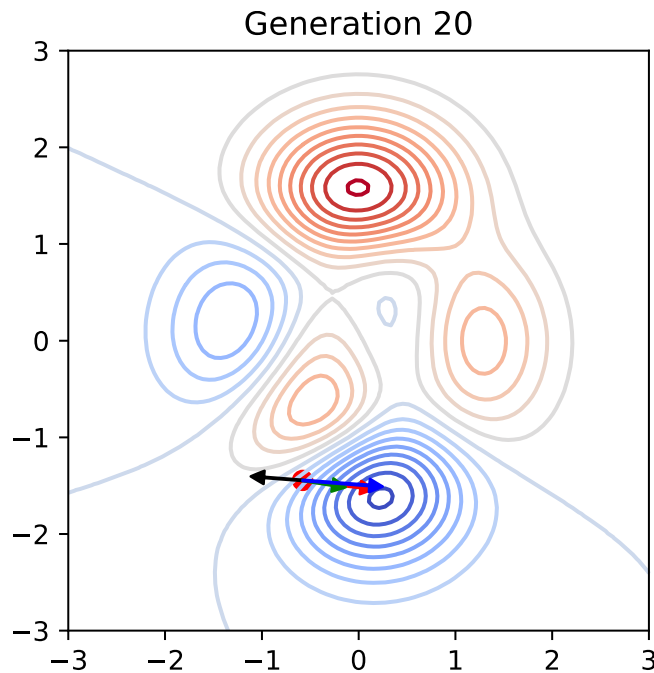
# PSO – Ví dụ - Quan sát 1 phần tử



—→ *inertia*    —→ *cognitive*    —→ *social*  
—→ *velocity*



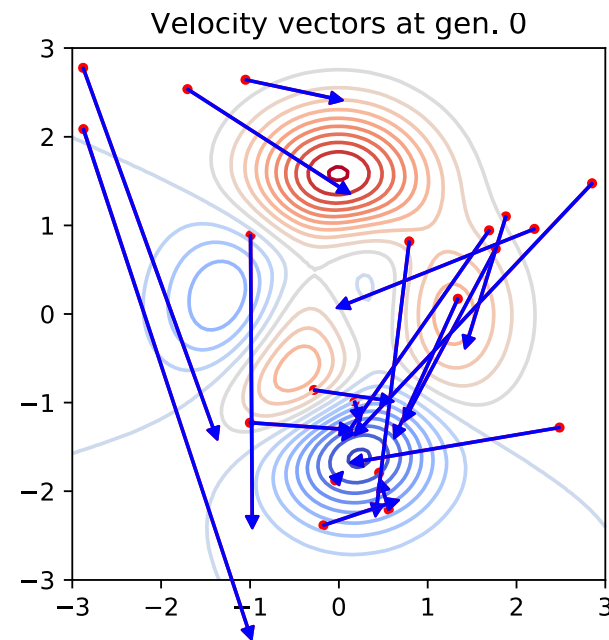
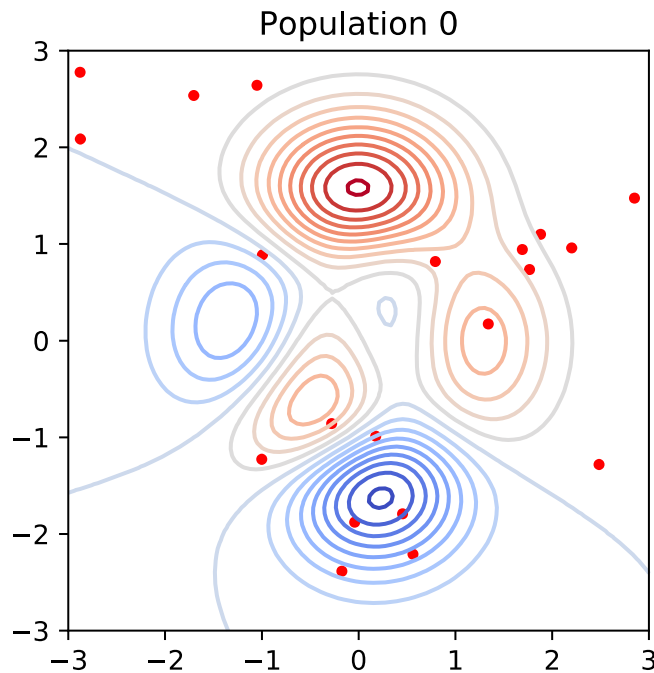
# PSO – Ví dụ - Quan sát 1 phần tử



—▶ inertia    —▶ cognitive    —▶ social  
—▶ velocity



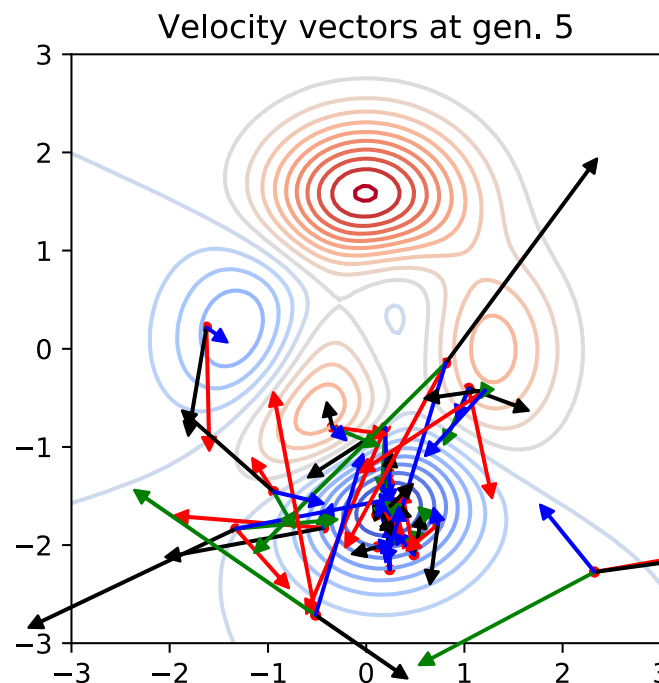
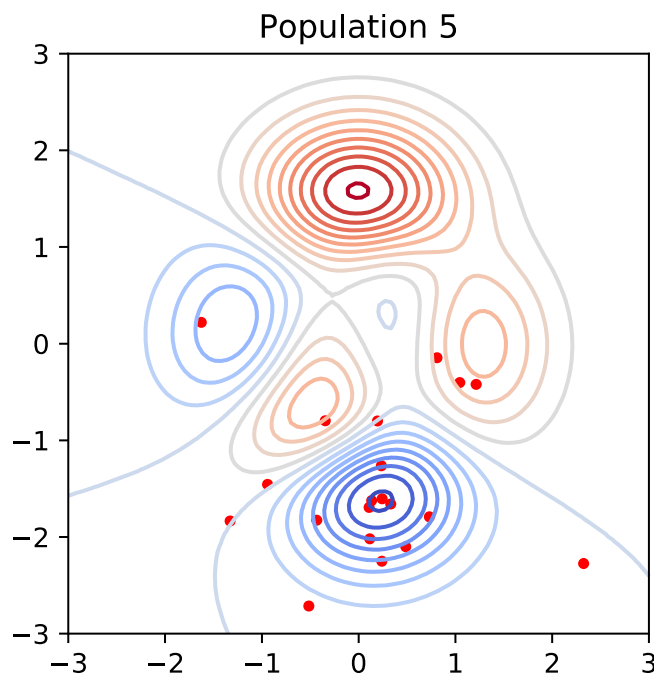
# PSO – Ví dụ - Quan sát cả bầy đàn



—▶ *inertia*    —▶ *cognitive*    —▶ *social*  
—▶ *velocity*



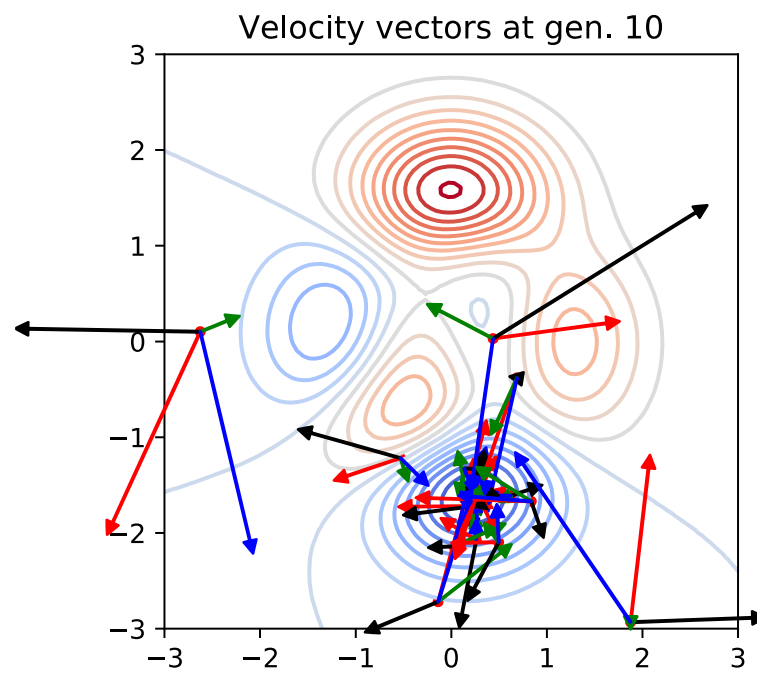
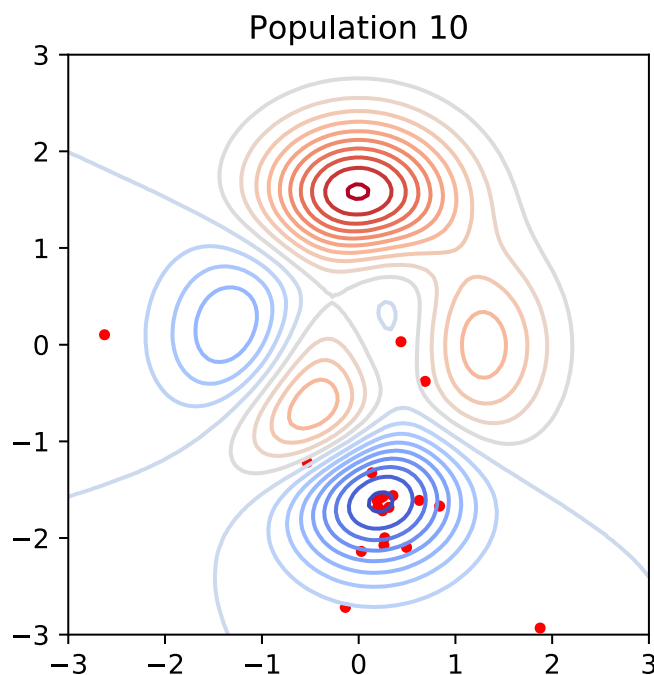
# PSO – Ví dụ - Quan sát cả bầy đàn



—▶ *inertia*    —▶ *cognitive*    —▶ *social*  
—▶ *velocity*

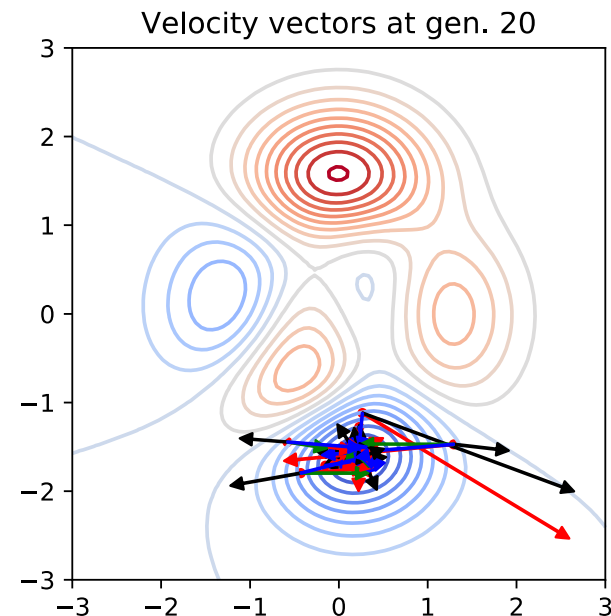
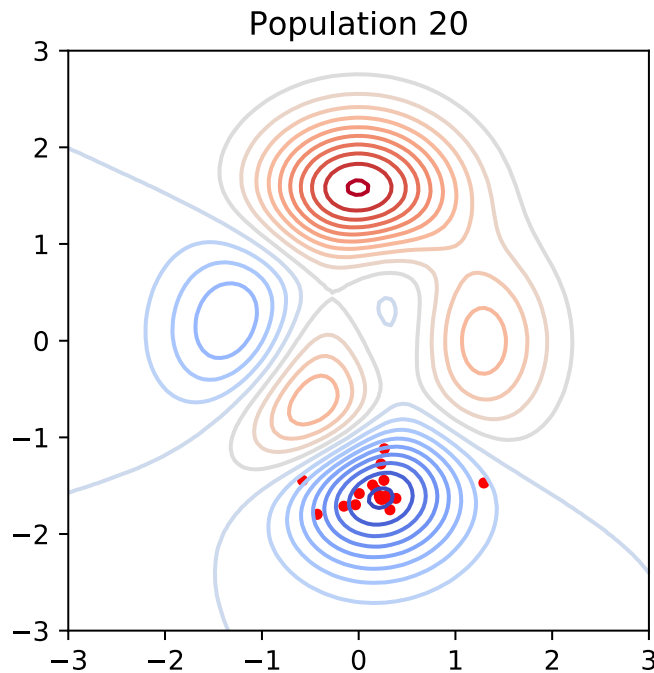


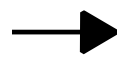
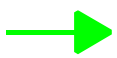


# PSO – Ví dụ - Quan sát cả bầy đàn



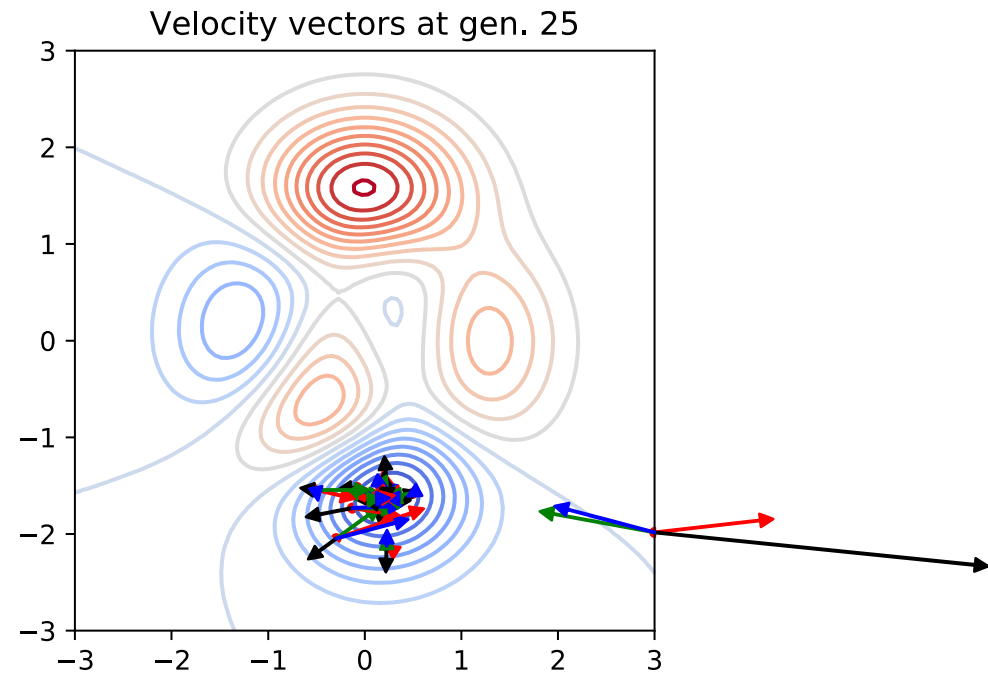
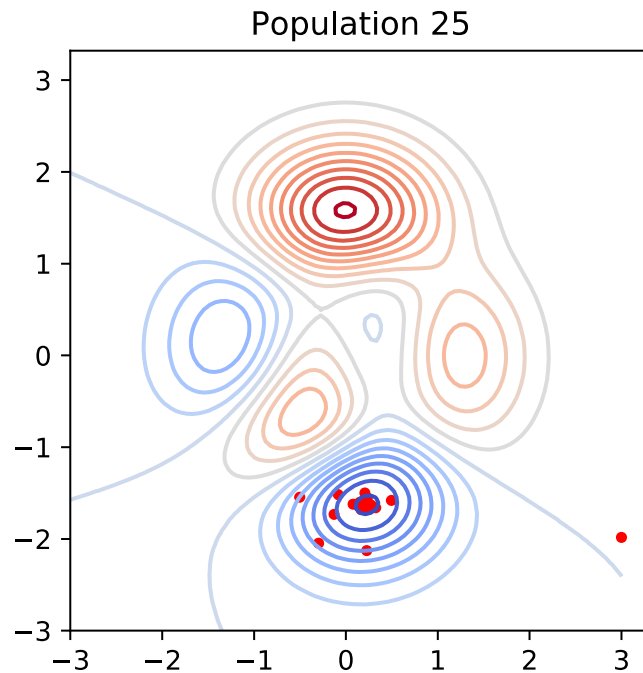
—▶ inertia —▶ cognitive —▶ social  
—▶ velocity

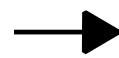
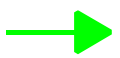


# PSO – Ví dụ - Quan sát cả bầy đàn



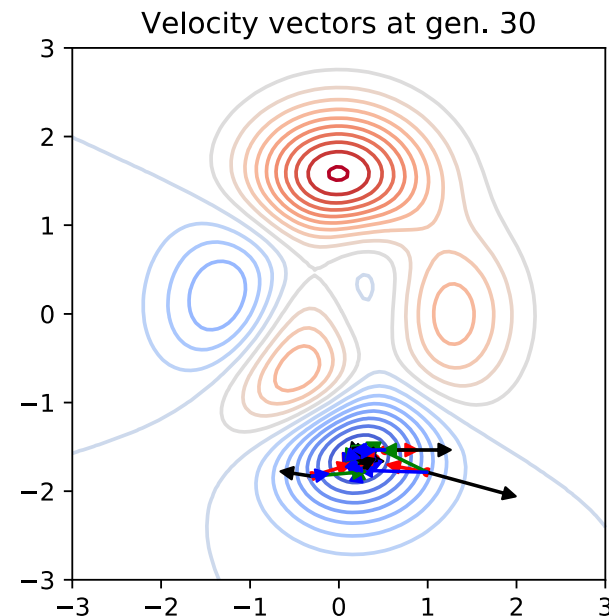
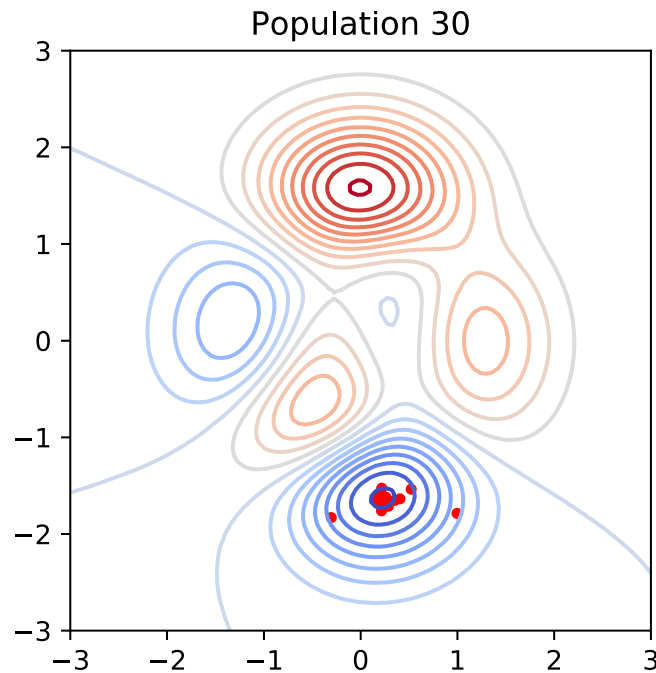
 *inertia*
 *cognitive*
 *social*  
 *velocity*

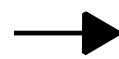
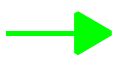


# PSO – Ví dụ - Quan sát cả bầy đàn



 *inertia*
 *cognitive*
 *social*  
 *velocity*

# PSO – Ví dụ - Quan sát cả bầy đàn



 *inertia*
 *cognitive*
 *social*  
 *velocity*

# PSO – Tham số điều khiển

- Hiệu suất của PSO cũng bị ảnh hưởng lớn bởi các tham số điều khiển:
- Kích thước bầy đàn.
- Định nghĩa về kiểu lân cận.
- Các hệ số: quán tính, gia tốc, ...



# Câu hỏi thảo luận

- Cách so sánh DE và PSO? Làm sao để so sánh các thuật toán tiến hoá khác nhau?
- Làm sao để cài đặt, điều chỉnh, và tinh chỉnh các tham số điều khiển của các thuật toán tiến hoá?
- Xét từng bài toán tối ưu hoá cụ thể mà ta cần giải, làm sao để chọn ra thuật toán tối ưu hoá thích hợp? Cách cấu hình riêng (customize) thuật toán cho từng trường hợp cụ thể?
- ...

# Tài liệu tham khảo

- Price K.V., Storn R.M., Lampinen J.A. (2005). The Differential Evolution Algorithm. Differential Evolution: A Practical Approach to Global Optimization, 37-134
- Riccardo P., Kennedy J., Blackwell T. (2007). Particle swarm optimization: An overview. Swarm intelligence 1 (1), 33-57