# Lecture 5 - Popular Evolutionary Algorithms Variants

Yuhui Shi

CSE, SUSTech

# Outline of This Lecture

- Differential Evolution
- Evolution Strategies
- Particle Swarm Optimisation
- Brain Storm Optimization
- Summary of this Lecture
- Reading List

# Outline of This Lecture

- **Differential Evolution**
- Evolution Strategies
- Particle Swarm Optimisation
- Brain Storm Optimization
- Summary of this Lecture
- Reading List

# Differential Evolution [1]
# -A Powerful Engine for Continuous Optimization

- The Big Idea: What is Differential Evolution and why is it different?
- Core Terminology: Key parameters that define a DE strategy.
- The Algorithm's Heart: The mutation and crossover steps.
- A Step-by-Step Walkthrough: Seeing DE in action on a simple problem.
- Strengths, Weaknesses, and Variants.

# The Big Idea: Evolution by Difference

What is Differential Evolution?

- Proposed by Storn and Price in 1995 [2]
- A population-based stochastic optimization algorithm for continuous-valued functions.
- Key Innovation: Uses vector differences between population members to perturb individuals.
- Standard EA: Relies on fixed probability distributions for mutation (e.g., Gaussian).
- Differential Evolution: Uses self-guided mutation. The step size and direction automatically adapt based on the current population spread.
- Analogy:
  - ✓ Standard Mutation: "Take a random step somewhere nearby."
  - ✓ DE Mutation: "Look at where your peers are, and take a step in a promising direction relative to them."

# Core Terminology and Strategy Notation

- A DE strategy is denoted by: DE/x/y/z
  - ✓x: The base vector to be perturbed.
    - ➢ rand (a random individual) or best (the best individual found).
  - ✓y: The number of difference vectors used.
    - ➢ Usually 1 or 2.
  - ✓z: The crossover scheme.
    - ➢ bin (Binomial) or exp (Exponential).
- Example: DE/rand/1/bin is the most classic and widely used variant.

# The DE Algorithm: Main Loop

1. Initialization: Generate a random population of NP candidate solutions (real-valued vectors).
2. For each target vector in the population, create a mutant vector through mutation.
3. Create a trial vector by mixing the target and mutant vectors through crossover.
4. Selection: Greedily choose whether the target or trial vector survives to the next generation.
5. Repeat steps 2-4 until a termination criterion is met.

# Step 1: Mutation - Generating the Mutant Vector
## *-The "Differential" Step*

- For each target vector $x_i$, generate a mutant/donor vector $v_i$.
- For DE/rand/1: $v_i = x_{r1} + F * (x_{r2} - x_{r3})$
  - ✓$x_{r1}$, $x_{r2}$, $x_{r3}$ are three distinct randomly selected population vectors.
  - ✓F is the Differential Weight or Scaling Factor (typically in [0, 2]).
  - ✓$(x_{r2} - x_{r3})$ is the difference vector that provides the direction and step size.

# Step 2: Crossover - Generating the Trial Vector
## -*Mixing Genes: Binomial Crossover*

- Create the trial vector $u_i$ by mixing coordinates from the target vector $x_i$ and the donor vector $v_i$.
- For each dimension j: $u_i[j] = \{ v_i[j]$ if (rand() < CR) or (j == $j_{rand}$) else $x_i[j] \}$
  - ✓Crossover Rate (CR): Probability [0, 1] of taking the parameter from the donor.
  - ✓$j_{rand}$: A randomly chosen dimension. Guarantees the trial vector gets at least one component from $v_i$.

# Step 3: Greedy Selection
## -*Survival of the Fitter*

- The newly created trial vector $u_i$ is pitted directly against its corresponding target vector $x_i$.
- For a minimization problem: $x_i$ {next_gen} = { $u_i$ if $f(u_i) <= f(x_i)$ else $x_i$ }
  - ✓The better of the two vectors advances to the next generation.
  - ✓This greedy selection provides strong selection pressure, driving the population toward optima quickly.

# Why Use DE? Advantages and Challenges

| Advantages (Pros) | Challenges (Cons) |
| --- | --- |
| ☑Simple to implement. Few control parameters (NP, F, CR). | ❌Primarily for continuous optimization. |
| ☑Excellent performance on many nonlinear, multimodal problems. | ❌Parameter tuning (F, CR) can be problem-specific. |
| ☑Self-adaptive step size. No need for a separate PDF. | ❌Not as theoretically well-founded as some other EAs. |
| ☑Good convergence properties. | ❌Can get stuck in local optima on complex problems. |

# DE Summary & Key Takeaways

Differential Evolution is a robust and efficient optimizer for continuous spaces.

- Its power comes from using vector differences to guide the mutation.
- A strategy is defined by the notation DE/x/y/z (e.g., DE/rand/1/bin).
- The three key parameters are:
  - ✓NP (Population Size)
  - ✓F (Scaling Factor)
  - ✓CR (Crossover Rate)
- It operates through a cycle of Mutation -> Crossover -> Greedy Selection.

# Outline of This Lecture

- Differential Evolution
- **Evolution Strategies**
- Particle Swarm Optimisation
- Brain Storm Optimization
- Summary of this Lecture
- Reading List

# Evolution Strategies [1]
## *-Mastering Continuous Optimization through Self-Adaptation*

- Origins & Philosophy: Where did ES come from and what problem does it solve?
- The Core Idea: Representation and the magic of self-adaptive mutation.
- Algorithm Anatomy: The (μ/ρ +, λ) notation and selection strategies.
- The Algorithm: A step-by-step breakdown.
- Key Operators: Mutation with strategy parameters and recombination.
- Why ES? Strengths, weaknesses, and modern variants.

# Origins & Philosophy

- Developed in the 1960s in Germany by Ingo Rechenberg and Hans-Paul Schwefel [3,4].
- Original goal: Optimize real-valued parameters for hydrodynamic shape design (e.g., a bent pipe for minimal pressure loss).
- Key Philosophy:
  - ✓Focus on continuous parameter optimization.
  - ✓The algorithm must adapt its parameters (like mutation step size) during the search.
  - ✓Heavily inspired by the idea of evolution at the phenotypic level (tweaking parameters directly).

# The Core Idea: Self-Adaptation

The Genius of ES

- In many algorithms, you must manually set parameters like mutation strength. What if the algorithm could learn this itself?
- The ES Solution: An individual is not just a set of object variables ($x_1, x_2, ..., x_n$). It also contains strategy parameters ($\sigma_1, \sigma_2, ..., \sigma_n$) that control the mutation of x.
  - ✓Chromosome: ($x_1, x_2, ..., x_n; \sigma_1, \sigma_2, ..., \sigma_n$)
  - ✓σ (sigma) is the mutation step size (standard deviation) for each individual/dimension.
  - ✓These σ values are mutated first, and then the new σ' values are used to mutate the x values.
  - ✓This allows the algorithm to automatically learn how to search.

# Algorithm Notation: (μ/ρ +, λ) -*Cracking the Code*

ES uses a specific notation to describe its selection mechanism:

- μ (mu): The number of parents in the population.
- λ (lambda): The number of offspring created from the parents.
- ρ (rho): The number of parents used to create one offspring (mixing number).
- + or ,: The selection type.
  - ✓(μ + λ)-ES: Select the best μ individuals from the combined pool of μ parents and λ offspring. (Elitist)
  - ✓(μ, λ)-ES: Select the best μ individuals only from the λ offspring (λ > μ). (Non-elitist)
- Example: (100, 600)-ES means 100 parents create 600 offspring, and the next generation is chosen only from those 600 offspring.

# The ES Algorithm: Main Loop
## *-Step-by-Step Process for (μ, λ)-ES*

1. Initialization: Create an initial population of μ parents. Each parent is a tuple (x, σ).
2. Repeat for each generation:
   a) Recombination: Create λ offspring by selecting ρ parents and mixing their (x, σ) values.
   b) Mutation: Mutate the strategy parameters σ of each offspring, then use them to mutate the object variables x.
   c) Evaluation: Calculate the fitness of each offspring.
   d) Selection: Select the best μ offspring to become the parents of the next generation.
3. Terminate if a stopping criterion is met.

# Key Operator 1: Mutation
# -*The Heart of Search*

- Mutation is the primary variation operator in ES. It happens in two stages:
- For each offspring '(x, σ)':
  1. Mutate σ first: $\sigma'_i = \sigma_i * \exp(\tau * N(0,1))$
     - ✓ τ is a learning rate (often $1/\sqrt{n}$).
     - ✓ N(0,1) is a random number from a standard normal distribution.
     - ✓ This allows for adaptive and non-local step sizes.
  2. Mutate x using the new σ': $x'_i = x_i + \sigma'_i * N_i(0,1)$
     - ✓ The new step size σ' is used to perturb x.

# Key Operator 2: Recombination
# -*Mixing Information*

- Recombination in ES is applied to both the object variables (x) and the strategy parameters (σ). Common types:
  - ✓Discrete Recombination (Dominant): For each component of the child, a value is chosen uniformly at random from one of the ρ parents.
    - child_x_i = parentA_x_i OR parentB_x_i
  - ✓Intermediate Recombination: The child is the average of the ρ parents.
    - child_x_i = (parentA_x_i + parentB_x_i) / 2
- Recombination helps mix successful strategies and accelerates convergence.

# Why Use ES? Pros and Cons

| Advantages (Pros) | Challenges (Cons) |
| --- | --- |
| ☑Highly effective for numerical, continuous optimization. | ❌Primarily designed for continuous domains. |
| ☑Self-adaptation removes the need to manually tune mutation parameters. | ❌The algorithm can be complex to implement correctly. |
| ☑Proven theoretical foundations (e.g., convergence proofs). | ❌The (μ, λ) strategy can forget good solutions if λ is too high. |
| ☑Robust performance on a wide range of problems. | ❌Not as effective for discrete or combinatorial problems. |

# ES Summary & Modern Context

- Key Takeaways:
  - ✓ES is a powerful EA variant specialized for continuous optimization.
  - ✓Its defining feature is self-adaptive mutation of strategy parameters (σ).
  - ✓Selection is described by the (μ/ρ +, λ) notation.
  - ✓Mutation is the main driver of variation; recombination is secondary.
- Modern Context:
  - ✓CMA-ES (Covariance Matrix Adaptation ES): A state-of-the-art variant that learns the correlations between variables, not just the step sizes. This allows it to navigate ill-conditioned, non-separable problems very effectively. It is often a very good choice for black-box optimization.

# Outline of This Lecture

- Differential Evolution
- Evolution Strategies
- **Particle Swarm Optimisation**
- Brain Storm Optimization
- Summary of this Lecture
- Reading List

# Particle Swarm Optimization Subtitle [1]
## *-Harnessing Collective Intelligence for Optimization*

- The Inspiration: Where did the idea come from?
- The Core Concept: What is a "Particle" and a "Swarm"?
- The Algorithm's Engine: The velocity update equation.
- A Step-by-Step Walkthrough: The PSO main loop.
- Key Parameters: Inertia, cognition, and social components.
- Variants: Bare-bones, topology, and more.
- Strengths, Weaknesses, and Applications.

Objective: Understand how PSO uses social influence to guide a population towards an optimum.

# The Inspiration: Swarm Intelligence

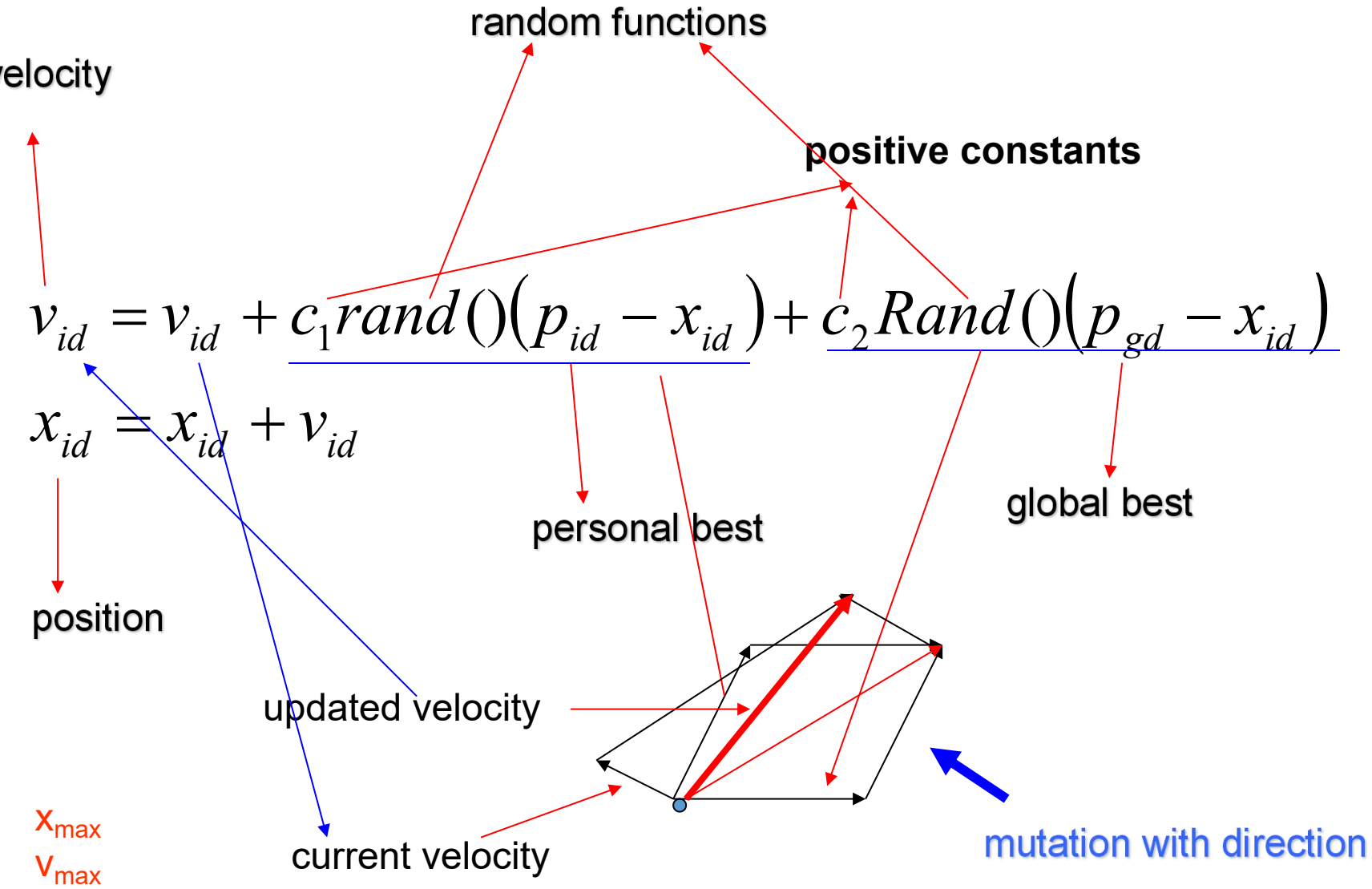
- Proposed by James Kennedy and Russell Eberhart in 1995 [5].
- Inspired by the social behavior of flocks of birds and schools of fish.
- Key Insight: Individuals in a group can accomplish complex tasks through simple rules and local communication.
  - ✓ No central control.
  - ✓ Each individual follows simple behaviors.
  - ✓ The complex, intelligent emergent behavior comes from their interaction.

In PSO: Birds = Particles. Finding food = Finding the optimal solution.

# The Building Blocks of PSO

- A Particle represents a potential solution to the optimization problem.
- Each Particle has:
  - ✓A position (x): Its current solution in the search space.
  - ✓A velocity (v): Its direction and speed of movement.
  - ✓A personal best (pbest): The best position it has ever found.
  - ✓Knows the global best (gbest): The best position found by any particle in its neighborhood.
- The Swarm is the collection of all particles, exploring the search space collaboratively.

# Original Version



# PSO with inertia weight [6]

add inertia weight

$$v_{id} = v_{id} + c_1 rand()(p_{id} - x_{id}) + c_2 Rand()(p_{gd} - x_{id})$$

$$x_{id} = x_{id} + v_{id}$$

$$v_{id} = w v_{id} + c_1 rand()(p_{id} - x_{id}) + c_2 Rand()(p_{gd} - x_{id})$$

$$x_{id} = x_{id} + v_{id}$$

# How Particles Decide Where to Go

- The heart of PSO is how a particle's velocity is updated each iteration.

$$v_{new} = (\omega \times v_{old}) + (c_1 \times rand() \times (pbest - x)) + (c_2 \times rand() \times (gbest - x))$$

- Let's break down the three components:
  1. Inertia ($\omega \times v_{old}$): Momentum term. Encourages exploration.
  2. Cognitive Component ($c_1$...): Attraction to the particle's own best-found solution. (Memory)
  3. Social Component ($c_2$...): Attraction to the swarm's best-found solution. (Collaboration)
- rand() introduces stochasticity.

# Putting It All Together

- After updating the velocity, the particle moves to its new position:

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

- The PSO Algorithm (Main Loop):
  1. Initialize a swarm of particles with random positions and velocities.
  2. Evaluate each particle's fitness.
  3. Update each particle's pbest and the swarm's gbest.
  4. For each particle:
     a. Calculate its new velocity.
     b. Update its position.
  5. Repeat from Step 2 until a stopping criterion is met (e.g., max iterations, solution found).

# Tuning the Swarm's Behavior

- ω (Inertia Weight): Controls exploration vs. exploitation.
  - ✓ High ω (~0.9): Global exploration (large steps).
  - ✓ Low ω (~0.4): Local exploitation (small steps, fine-tuning).
- $c_1$ (Cognitive Acceleration Coefficient): How much the particle trusts its own experience.
- $c_2$ (Social Acceleration Coefficient): How much the particle trusts the swarm's experience.
- Swarm Size (S): Typically 20-50 particles. Problem-dependent.

Rule of Thumb: Often, setting $c_1 = c_2 = 2$ and tuning ω works well.

# Topology: Who Talks to Whom?

- The flow of information is defined by the topology (who informs whom about gbest).
  - ✓Global Topology (gbest): All particles are connected.
    - ➢ One gbest for the whole swarm.
    - ➢ Faster convergence, but higher risk of getting stuck in local optima.
  - ✓Local Topology (lbest): Particles are connected only to neighbors (e.g., a ring).
    - ➢ Multiple local bests.
    - ➢ Slower convergence, but better exploration.

# Why Use PSO? Pros and Cons

| Advantages (Pros) | Challenges (Cons) |
| --- | --- |
| ☑Simple concept and easy to implement. Few parameters. | ❌Primarily for continuous optimization. |
| ☑Fast convergence in early stages. | ❌Can converge prematurely on sub-optimal solutions. |
| ☑Derivative-free. Does not require gradient information. | ❌Theoretical analysis is less mature than for EAs. |
| ☑Very effective for a wide range of problems. | ❌Performance can be sensitive to parameter tuning. |

# Summary, Applications, & Variants

- PSO is a population-based optimizer inspired by social behavior.
- Potential solutions are particles that fly through the search space.
- Movement is guided by personal best (pbest) and neighborhood best (gbest/lbest).
- The balance between exploration and exploitation is controlled by $\omega$, $c_1$, and $c_2$.
- Applications: Parameter tuning, neural network training, antenna design, robotics.
- Common Variants: Bare-Bones PSO, Quantum PSO, Adaptive PSO.

# Outline of This Lecture

- Differential Evolution
- Evolution Strategies
- Particle Swarm Optimisation
- **Brain Storm Optimization**
- Summary of this Lecture
- Reading List

# Brain Storm Optimization Subtitle [1]
## *-Simulating Human Creativity for Complex Problem-Solving*

- The Inspiration: The psychology of human brainstorming.
- The Core Metaphor: How are ideas like individuals in a population?
- The Algorithm's Anatomy: The two key phases: Clustering and Idea Generation.
- A Step-by-Step Walkthrough: The BSO main loop.
- Why It's Unique: The role of convergence and divergence.
- Strengths, Weaknesses, and Applications.

Objective: Understand how BSO mimics collective creativity to balance exploration and exploitation.

# The Inspiration: Human Brainstorming

- Proposed by Shi in 2011 [7,8].
- Inspired by the collective human creativity process: brainstorming.
- Key Principles of Brainstorming:
  1. No bad ideas: Defer judgment to encourage divergence.
  2. Combine and improve ideas: Convergence builds on others' thoughts.
  3. Quantity leads to quality: Many ideas increase the chance of a breakthrough.
- The Algorithmic Metaphor:
  - ✓ Individuals = Ideas
  - ✓ Fitness = Quality of an Idea
  - ✓ Clustering = Grouping similar ideas
  - ✓ New Idea Generation = Combining or perturbing existing ideas

# Core Concept: The Two Fundamental Processes -*Convergence and Divergence*

- BSO's power comes from explicitly managing two competing forces:
  1. Convergence (Exploitation):
     - ✓ Action: Grouping similar ideas (solutions) into clusters.
     - ✓ Goal: Focus search on promising regions, refining good solutions.
     - ✓ "Let's focus on this one great approach and improve it."
  2. Divergence (Exploration):
     - ✓ Action: Creating new ideas by combining clusters or randomly perturbing ideas.
     - ✓ Goal: Explore new, potentially radical solutions, avoiding local optima.
     - ✓ "What if we took a piece from that completely different idea and mixed it with this one?"

# The Algorithm: Main Loop

1. Initialization: Generate n random ideas (individuals) in the search space.
2. Clustering: Group the n ideas into m clusters using a clustering algorithm (e.g., k-means).
3. Evaluate & Rank: Calculate the fitness of each idea and identify the best idea in each cluster.
4. Apply a disruptive operator to a cluster' best idea with a low probability.
5. Generate New Ideas: For each new idea to be created:
   - ✓ Select a cluster (with a bias towards better clusters).
   - ✓ Create a new idea based on one or two existing ideas in the cluster(s).
   - ✓ Select the next population from old and new ideas.
6. Repeat until a termination criterion is met.

# Step 1: The Clustering Phase
## -*Organizing the Ideas*

- Purpose: To group similar solutions, preventing premature convergence and managing diversity.
- Common Method: K-Means Clustering.
  - ✓ The number of clusters, k, is a key algorithm parameter.
  - ✓ Each cluster has a center (mean of its ideas) and a best idea (highest fitness member).
- Why it matters: The cluster structure directly guides the search. Operators treat ideas within a cluster differently than ideas from different clusters.

# Step 2: The New Idea Generation Operator
# -*The Creative Engine*

This is the core variation operator. To create a new idea $X_{new}$:

1. Randomly generate a value between 0 and 1;
2. If the value is less than a probability p-one,
   i. Randomly select a cluster;
   ii. Generate a random value between 0 and 1;
   iii. If the value is smaller than a pre-determined probability p-select-best,
      - Select the cluster center and add random values to it to generate new individual.
   iv. Otherwise randomly select an individual from this cluster and add random value to the individual to generate new individual.
3. Otherwise randomly select two clusters to generate new individual
   I. Generate a random value and the two cluster are randomly selected;
   II. If it is less than a pre-determined probability p-select-best, the two clusters' centers are combined and then added with random values to generate new individual;
   III. Otherwise, two individuals from each selected cluster are randomly selected to be combined and added with random values to generate new individual.
4. The newly generated individual is compared with the existing individual with the same individual index, the better one is kept and recorded as the new individual;

# The "Disruptive" Operator for Innovation
## *-Simulating a "Eureka!" Moment*

- To prevent getting trapped in local optima, BSO includes a disruptive operator.
- With a very low probability (p_disrupt), a new idea is not created from existing clusters.
- Instead, it is generated randomly in the search space, just like in the initialization step.
- This represents a "wild" or "out-of-the-box" idea that is completely different from the current thinking.
- This mechanism is crucial for maintaining population diversity and global exploration capabilities.

# Why Use BSO? Pros and Cons

| Advantages (Pros) | Challenges (Cons) |
| --- | --- |
| ☑Novel approach excellent for multimodal and complex problems. | ❌Computationally expensive due to the clustering step in every iteration. |
| ☑Explicitly manages exploration vs. exploitation via clustering. | ❌Has more parameters to tune (k, p_select_best, p_one_idea, p_disrupt, etc.). |
| ☑Disruptive operator helps escape local optima effectively. | ❌Performance is sensitive to the choice of clustering algorithm and its parameters (e.g., k). |
| ☑Intuitive metaphor, easy to understand conceptually. | ❌Less established and studied compared to classic algorithms like PSO or GA. |

# Summary, Applications, & Conclusion

- BSO is a population-based algorithm that simulates human brainstorming.
- Its key innovation is using clustering to structure the population and guide the search.
- New ideas are created by perturbing or combining ideas from one or more clusters.
- A disruptive operator introduces radical new ideas to maintain diversity.
- Applications: Well-suited for complex, high-dimensional, multimodal optimization problems like:
  - ✓ Neural network architecture search
  - ✓ Data clustering (meta-optimization)
  - ✓ Engineering design problems
- Final Thought: BSO isn't just an optimizer; it's a computational model of collective creativity.

# Outline of This Lecture

- Differential Evolution
- Evolution Strategies
- Particle Swarm Optimisation
- Brain Storm Optimization
- **Summary of this Lecture**
- Reading List

# Summary of this Lecture

- Four popular EAs are described, i.e., to futher illustrate how EAs work
  - ✓Diefferential evolution
  - ✓Evolutionary strategy
  - ✓Particle swarm optimization
  - ✓Brain storm optimization

# Outline of This Lecture

- Differential Evolution
- Evolution Strategies
- Particle Swarm Optimisation
- Brain Storm Optimization
- Summary of this Lecture
- **Reading List**

# Reading List I


1. DeepSeek
2. R. Storn and K. V. Price, Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces, 1995.
3. Ingo Rechenberg. “Cybernetic solution path of an experimental problem”. In: Royal Aircraft Establishment Library Translation 1122 (1965).
4. H-P Schwefel. “Kybernetische Evolution als Strategie der experimentellen Forschung in der Stromungstechnik”. In: Diploma thesis, Technical Univ. of Berlin (1965).
5. Kennedy, James, and Russell Eberhart. "Particle swarm optimization." Proceedings of ICNN'95-international conference on neural networks. Vol. 4. ieee, 1995.
6. Shi, Yuhui, and Russell Eberhart. "A modified particle swarm optimizer." 1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360). Ieee, 1998.

# Reading List II


7. Shi, Yuhui. "Brain storm optimization algorithm." International conference in swarm intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.
8. Shi, Yuhui. "An optimization algorithm based on brainstorming process." Emerging Research on Swarm Intelligence and Algorithm Optimization. IGI Global Scientific Publishing, 2015. 1-35.