Population-Based Search

Lecture 7, CMSC 170

John Roy Daradal / Instructor

Previously on CMSC 170

Local Search:

- Neighbors, Legal Neighbors, Selection, Objective function
- Hill Climbing
- Simulated Annealing
- Tabu Search

Today's Topics

Population-Based Search:

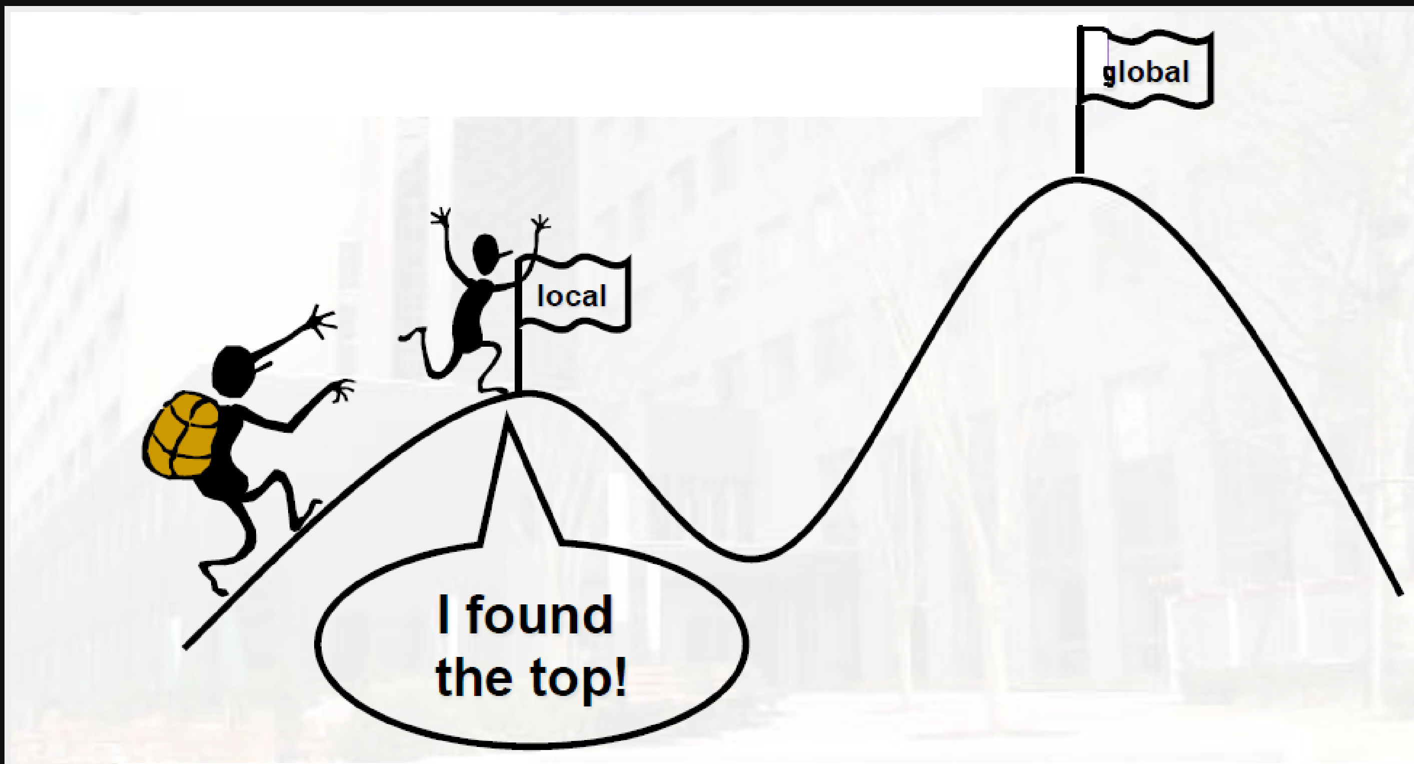
- Genetic Algorithms
- Swarm Algorithms

Constrained Problems

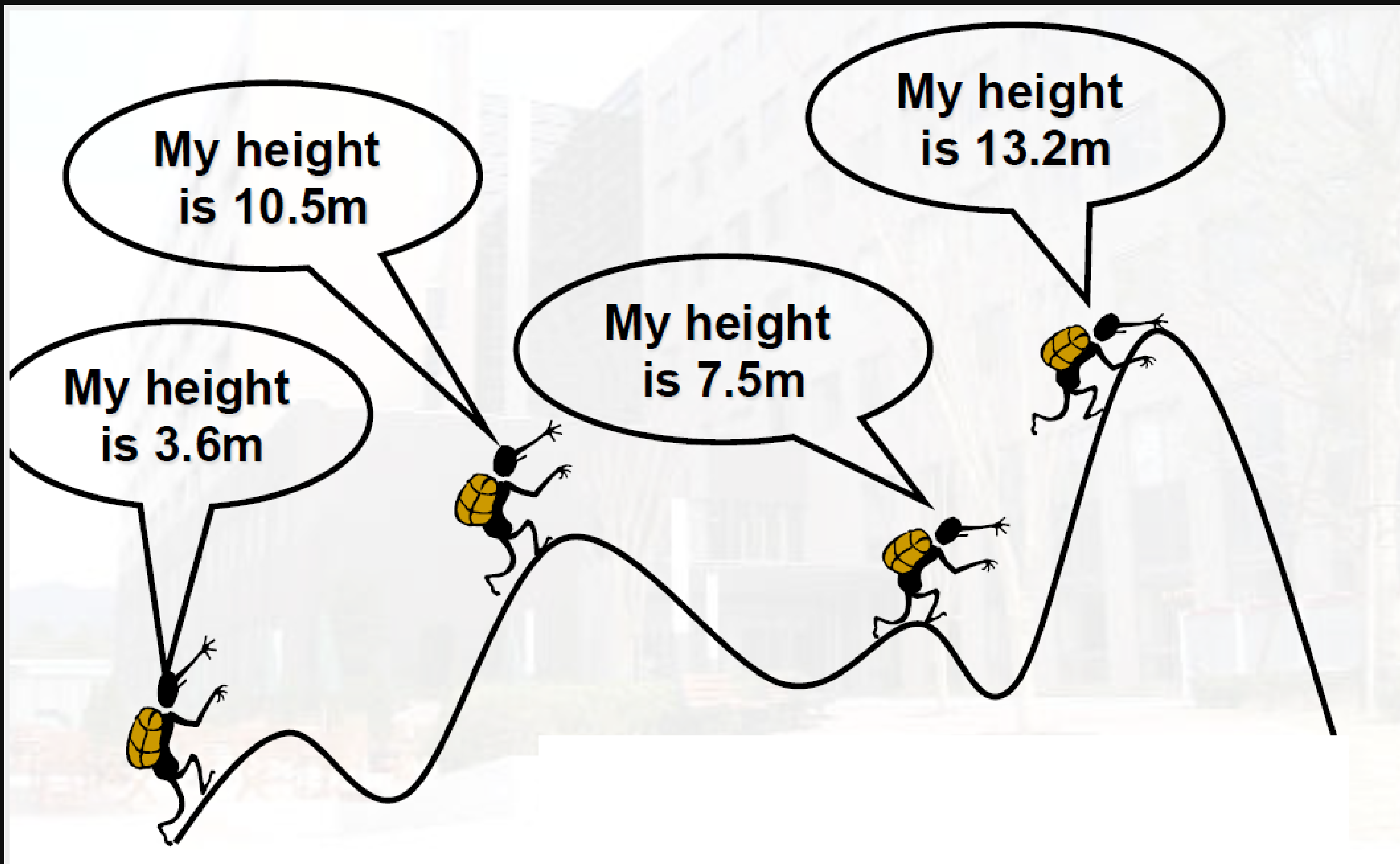
Solutions:

- **Backtracking**: **extend** *partial* solution
- **Local Search**: **modify** *complete* solution
- **Population-Based Search**: use **multiple** solutions

Local Search



Population-Based Search



Genetic Algorithm

Genetic Algorithm

- **Population** of solutions
- Keep N **best** solutions at each step
- **Natural selection**: only the *fittest* survive

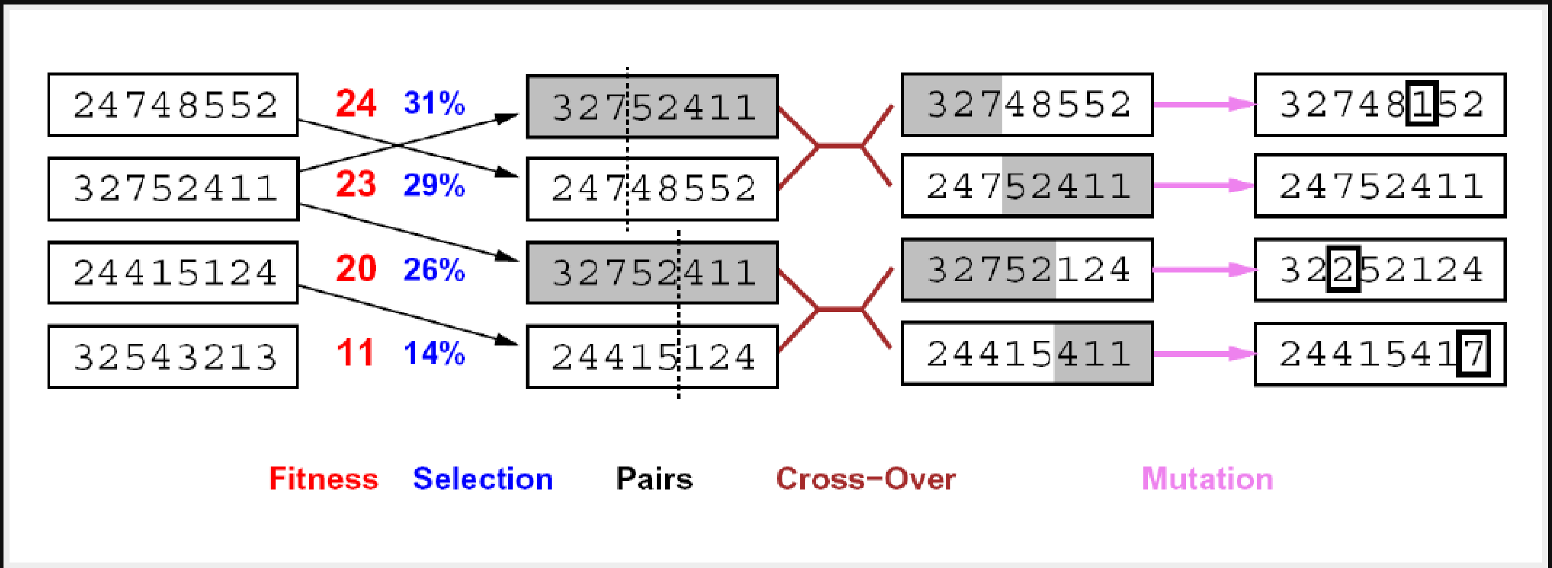
Genetic Algorithm

- **Evolutionary** algorithm
- Inspired by *population genetics* and *evolution*
- Constrained & unconstrained optimization

Applications

- Optimization
- Scheduling
- Game Theory
- Training Neural Networks
- Image Processing

Genetic Algorithm



Genetic Algorithm

- **Fitness function**: objective function
- **Selection**: parents of next generation
- **Crossover**: mating parents / breeding
- **Mutation**: modify solution

GA Terminology

- **Population**: set of solutions
- **Generation**: population in an iteration
- **Chromosome**: one solution
- **Gene**: one element of solution

GA Solutions

- Represented as **strings** or **vectors**
- *Common*: binary strings, array of integers
- Easier to perform *genetic operations*

Example

FIRE STATION LOCATION PROBLEM

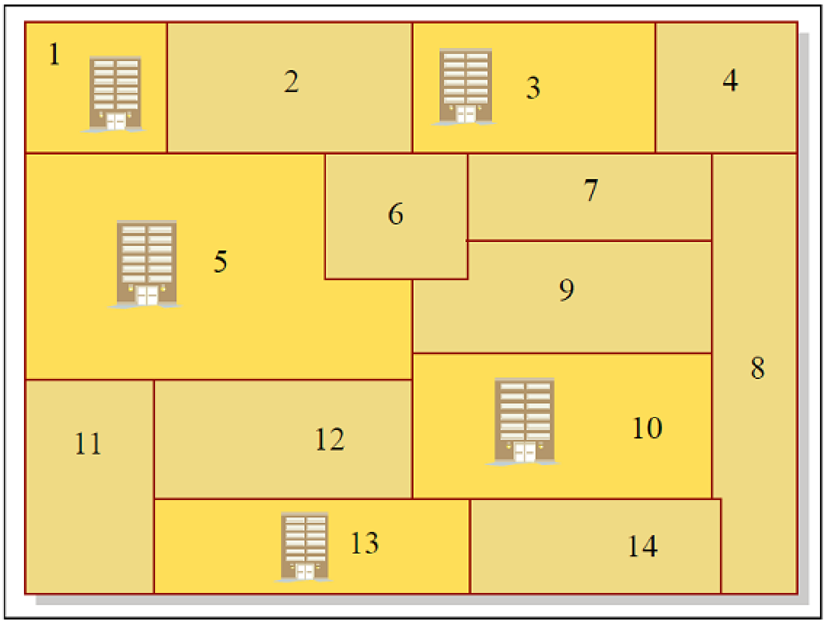


Image by MIT OpenCourseWare.

1 0 1 0 1 0 0 0 0 1 0 0 1 0

"1" represents a fire station

Genetic Operators

- Crossover, Mutation
- Generate **next generation** of solutions
- Bulk of GA work = **designing** genetic operators

Genetic Algorithm

- Repeatedly **modify** population of solutions
- Select **parents** at random from current population (**fitter** solution → higher chance)
- Perform **genetic operations** on parents to produce *next generation*
- Population *evolves* towards **optimal solution** over time

Genetic Algorithm

Input: $Population_{size}$, $Problem_{size}$, $P_{crossover}$, $P_{mutation}$
Output: S_{best}
Population $\leftarrow$ `InitializePopulation`($Population_{size}$, $Problem_{size}$);
`EvaluatePopulation`(Population);
$S_{best} \leftarrow$ `GetBestSolution`(Population);
while $\neg$`StopCondition`() **do**
    Parents $\leftarrow$ `SelectParents`(Population, $Population_{size}$);
    Children $\leftarrow \emptyset$;
    foreach $Parent_1$, $Parent_2$ $\in$ Parents **do**
        $Child_1$, $Child_2$ $\leftarrow$ `Crossover`($Parent_1$, $Parent_2$, $P_{crossover}$);
        Children $\leftarrow$ `Mutate`($Child_1$, $P_{mutation}$);
        Children $\leftarrow$ `Mutate`($Child_2$, $P_{mutation}$);
    end
    `EvaluatePopulation`(Children);
    $S_{best} \leftarrow$ `GetBestSolution`(Children);
    Population $\leftarrow$ `Replace`(Population, Children);
end
return S_{best};

Termination

Stop when:

- **Solution** found satisfying minimum criteria
- Fixed number of **generations** reached
- Best solution's fitness has no more **improvement**

Initial Population

- **Randomly** generated
- Generated by **heuristic** (e.g. greedy)

Population Models

Steady State

- generate k offsprings, k = 1 or 2
- replace k solutions from current population
- aka *incremental* GA

Population Models

Generational

- current population size = N
- generate N offsprings
- replace entire population with new generation

Fitness Function

- Objective function
- How good / fit is this solution?
- Should be fast to compute
- Maximize or minimize

Min → Max

Convert minimization → maximization:

- N - score
- 1 / score
- -score

Selection

Choosing **parents** of next generation

Age-based

- each solution can only live for **fixed number** of generations (kick out after)

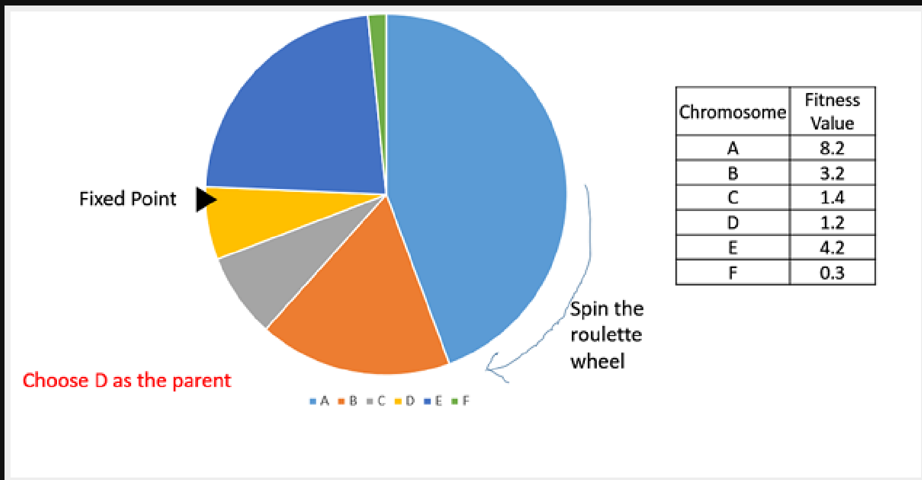
Fitness-based

- choose based on solution's **fitness value**

Selection

- **Fitness-based**: select **best solutions** based on fitness values
- **Stochastic**: more fit → higher probability
- *Idea*: fit solutions have **better genes** to pass on to next generation
- Purely random selection is *discouraged*

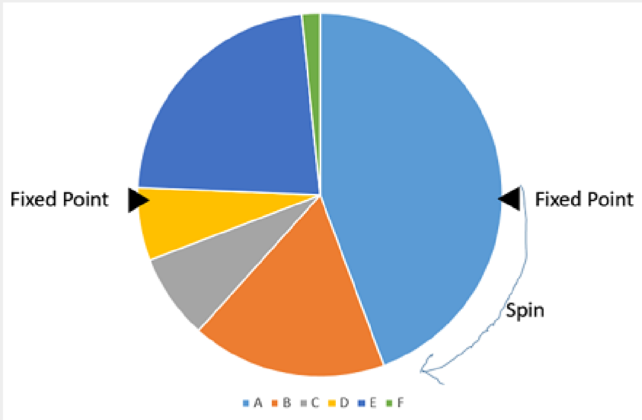
Roulette Wheel Selection



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Stochastic Universal Sampling

Allows weaker members of population to have chance of being chosen

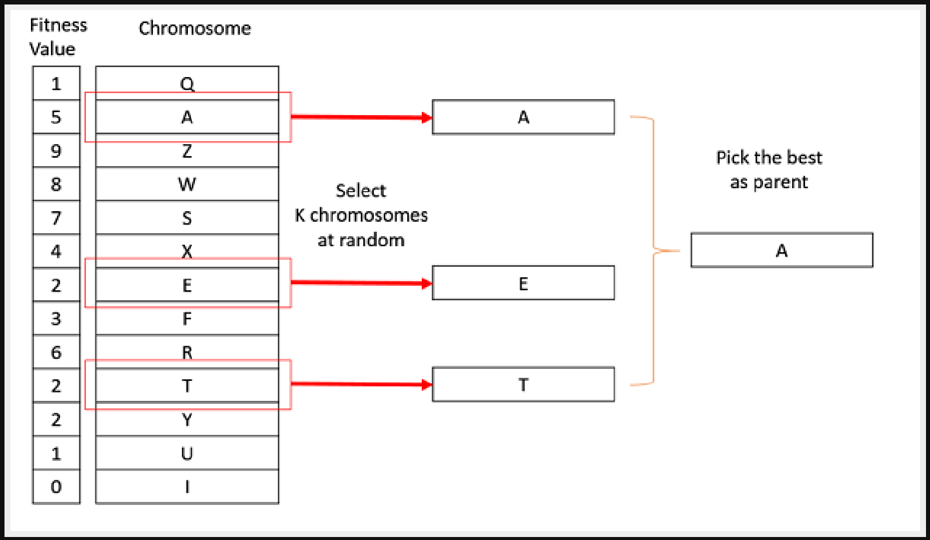


Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

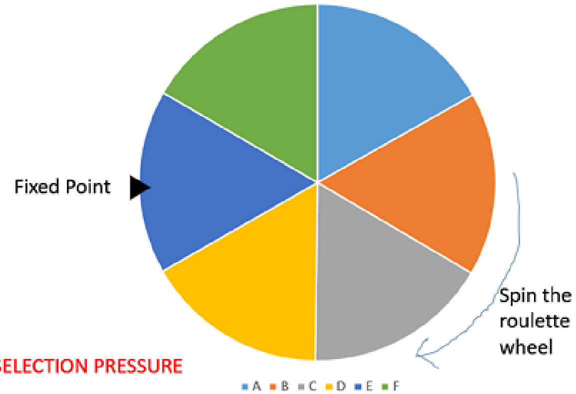
Fitness-Proportionate Selection

- Fit solutions more likely to be selected
- **Weakness**: when some members of population have very large fitness values compared to others (**domination**)

Tournament Selection



Rank Selection



NO SELECTION PRESSURE

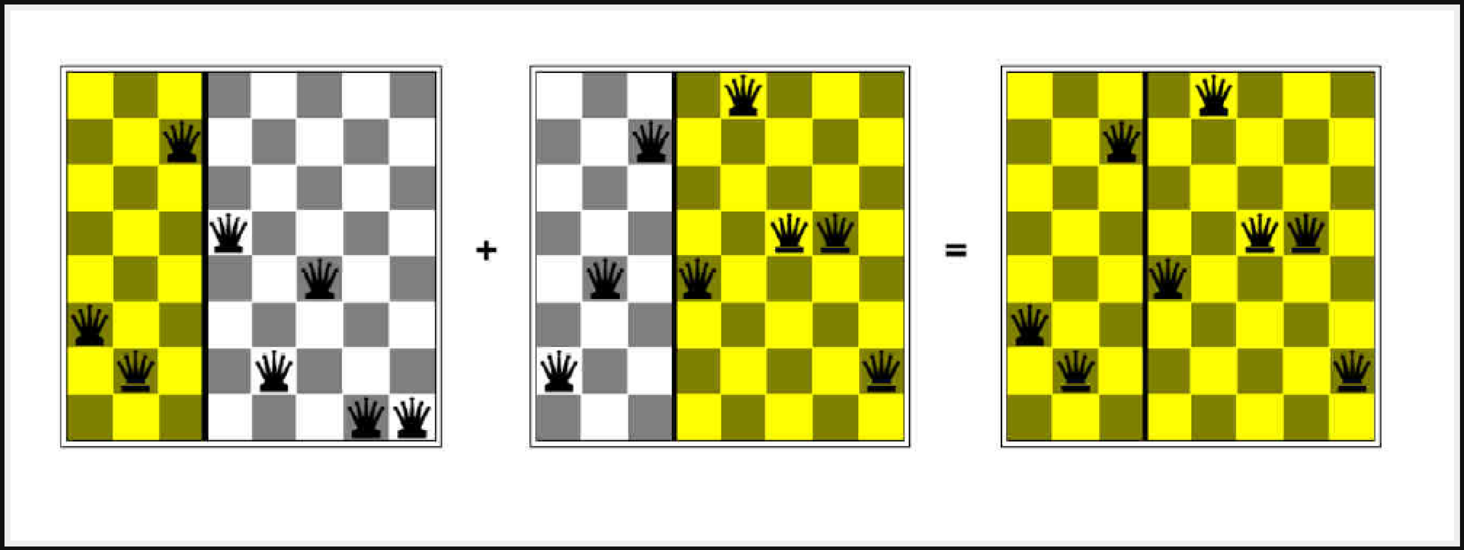
Chromosome	Fitness Value
A	8.1
B	8.0
C	8.05
D	7.95
E	8.02
F	7.99

Chromosome	Fitness Value	Rank
A	8.1	1
B	8.0	4
C	8.05	2
D	7.95	6
E	8.02	3
F	7.99	5

Crossover

- Genetic operator
- **Combine** genes from both parents
- Produce **new solutions** for next generation
- Usually given **high probability**

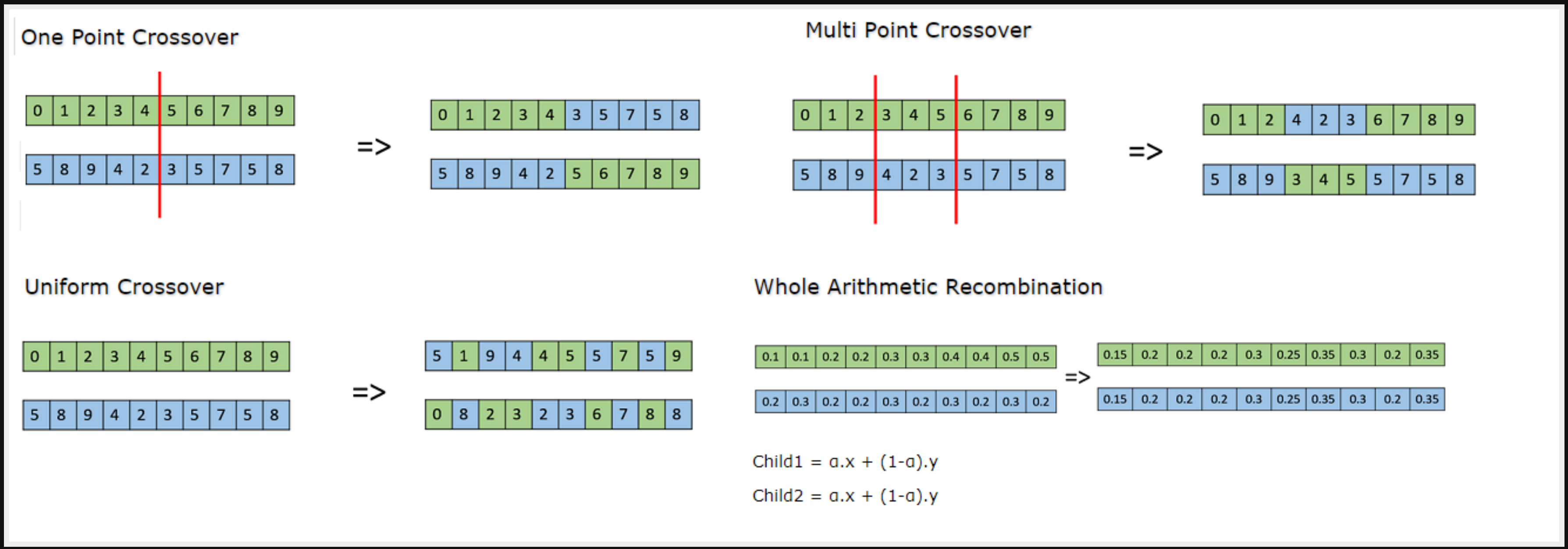
Example: N-Queens



Speciation Heuristic

- **Penalize** crossover between *similar solutions*
- Encourages **diversity**
- Prevents **premature convergence** to suboptimal solution

Crossover

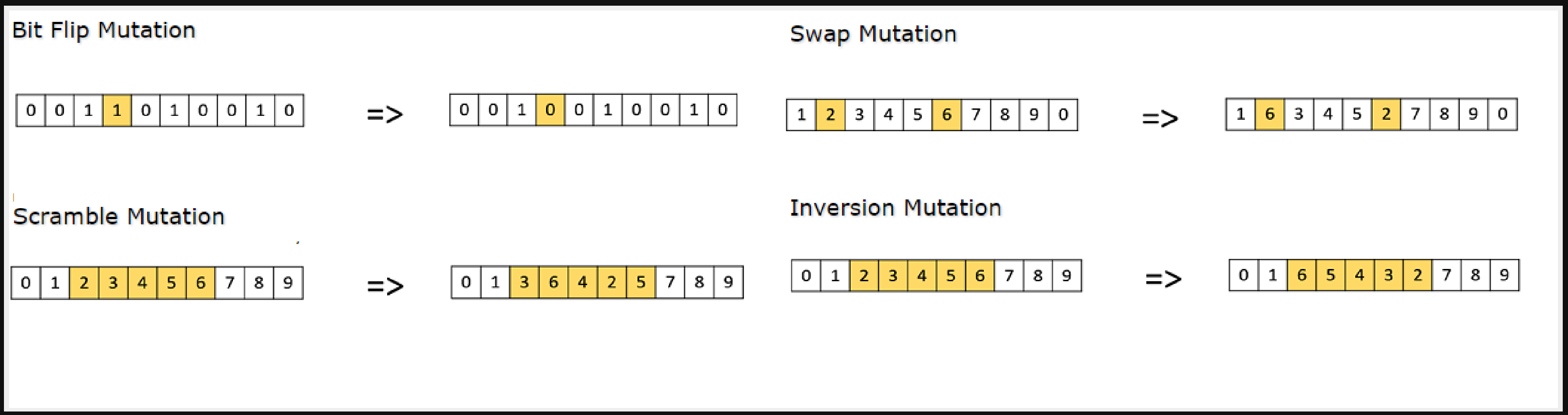

One Point Crossover
0 1 2 3 4 5 6 7 8 9
5 8 9 4 2 3 5 7 5 8
=>
0 1 2 3 4 3 5 7 5 8
5 8 9 4 2 5 6 7 8 9
Multi Point Crossover
0 1 2 3 4 5 6 7 8 9
5 8 9 4 2 3 5 7 5 8
=>
0 1 2 4 2 3 6 7 8 9
5 8 9 3 4 5 5 7 5 8
Uniform Crossover
0 1 2 3 4 5 6 7 8 9
5 8 9 4 2 3 5 7 5 8
=>
5 1 9 4 4 5 5 7 5 9
0 8 2 3 2 3 6 7 8 8
Whole Arithmetic Recombination
0.1 0.1 0.2 0.2 0.3 0.3 0.4 0.4 0.5 0.5
0.2 0.3 0.2 0.2 0.3 0.2 0.3 0.2 0.3 0.2
=>
0.15 0.2 0.2 0.2 0.3 0.25 0.35 0.3 0.2 0.35
0.15 0.2 0.2 0.2 0.3 0.25 0.35 0.3 0.2 0.35
Child1 = α.x + (1-α).y
Child2 = α.x + (1-α).y

Mutation

- Perform **local changes** to solution to slightly **modify** it
- **Diversify** solutions, **explore** search space
- No mutation = limited search space
- Usually given **low probability**

Mutation

Random **perturbations**:

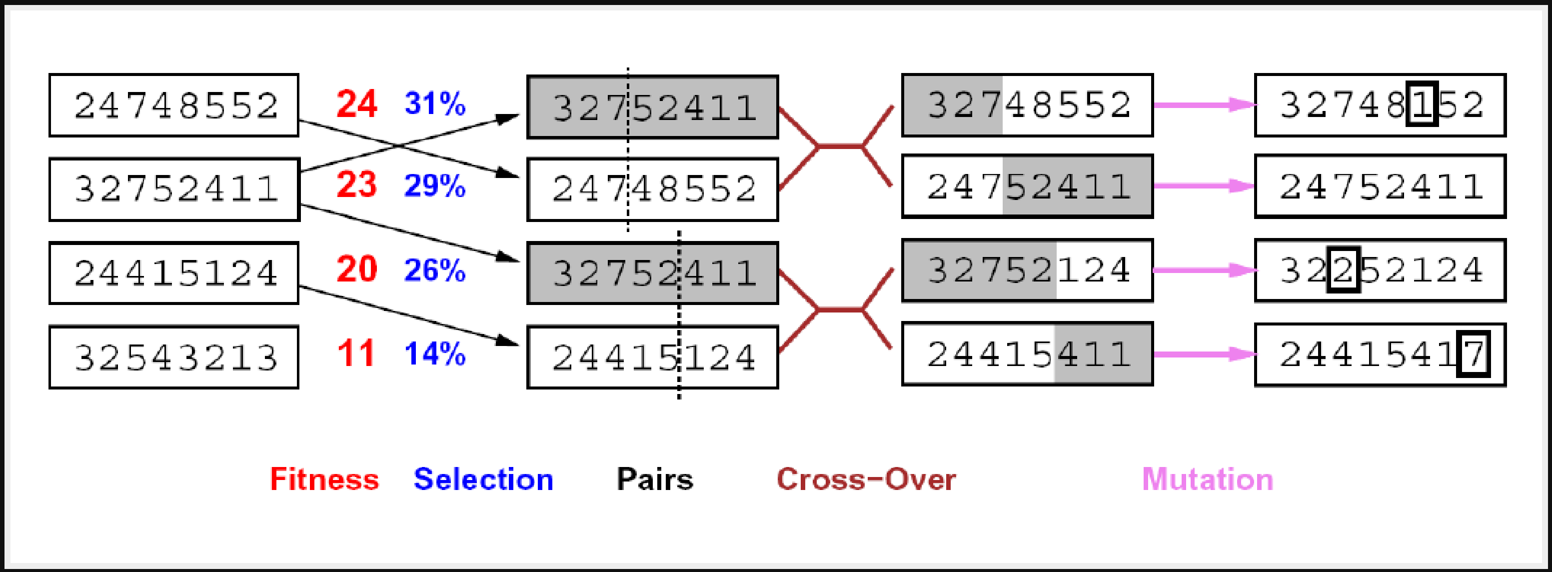


Handling Constraints

Implicit:

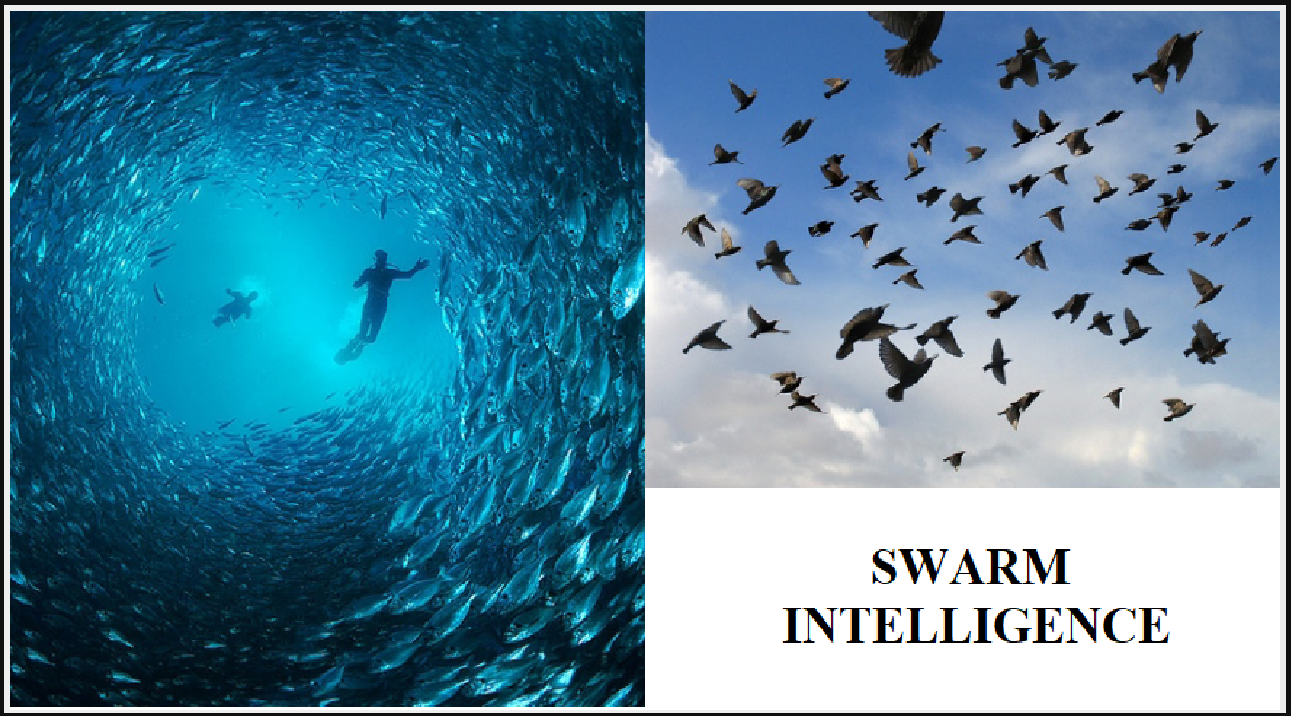
- **Fitness function**: penalty for violation
- **Selection**: reject constraint violators
- **Encoding**: only allow valid encodings

Genetic Algorithm



Demo: MaxOne problem

Swarm Algorithms



Swarm Algorithms

- **Collective behavior** of *self-organized* agents
- **Population** of agents **interact** locally with *one another* and with *environment*

Collective Intelligence

"The whole is more than the sum of its parts"

Swarm Algorithms

- Cooperation
- Competition
- Communication
- Self-Organization

Applications

- Swarm robotics
- Dynamic optimization
- Scheduling
- Medical Diagnosis
- Image Analysis
- Data mining, clustering

Swarm Algorithms

- Particle Swarm Optimization
- Ant-Colony Optimization
- Artificial Bee Colony Algorithm

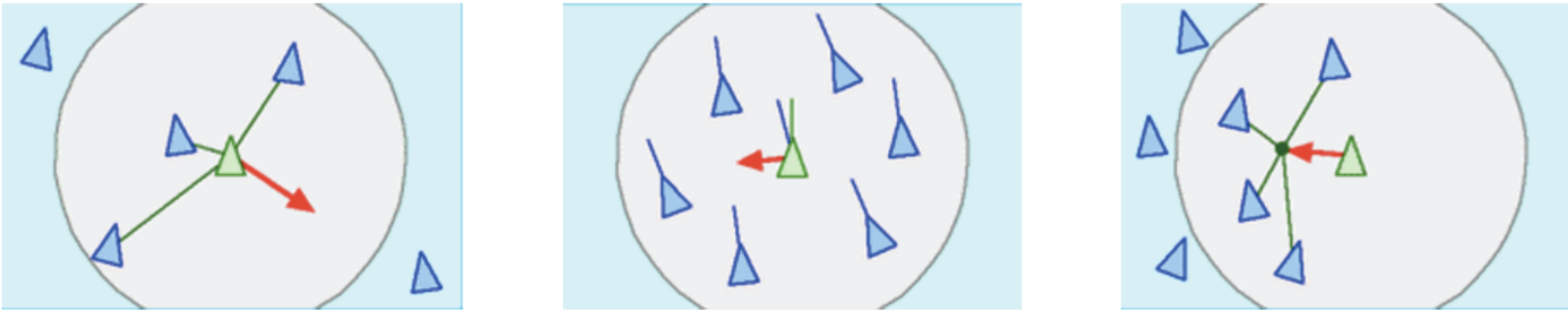
Particle Swarm Optimization

- **Social behavior**: bird flock, fish school
- **Population** of candidate solutions (particles)
- Move **particles** around search space
- Guided towards **best known positions** in search space

Boids

- Craig Reynolds (1986)
- Model of **coordinated** animal motion
- Three **local rules**: separation, alignment, cohesion

Boids



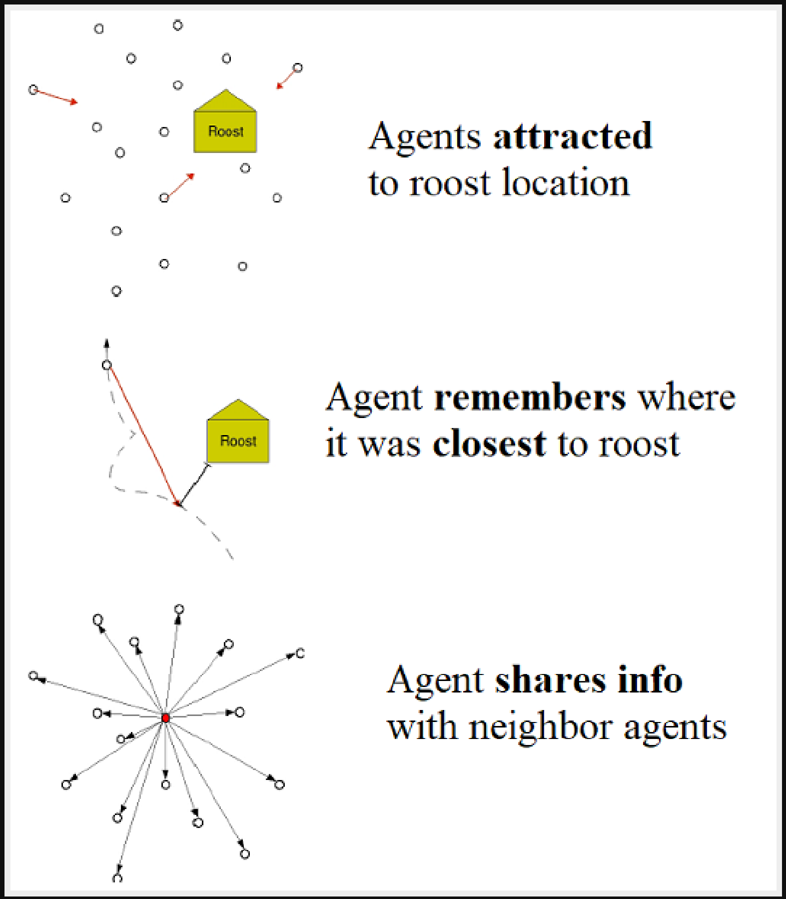
Separation: steer to avoid crowding local flockmates

Alignment: steer towards the average heading of local flockmates

Cohesion: steer to move toward the average position of local flockmates

[https://www.youtube.com/watch?v=QbUPfMXXQIY]

Boids + Roosting



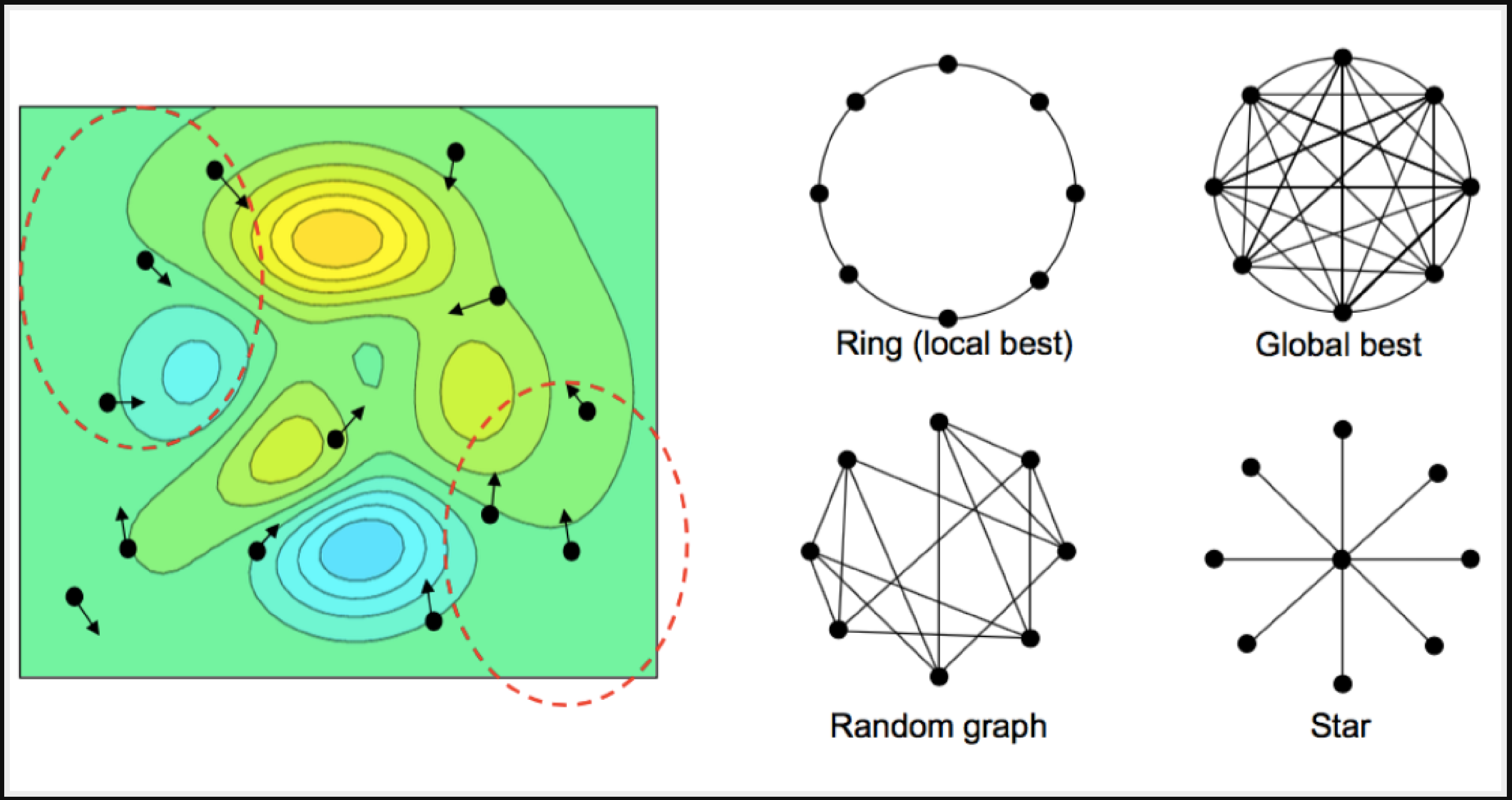
Boids + Roosting

- *1995*: J. Kennedy & R. Eberhart added **roost** (attraction point)
- Visualization of **bird flock** behavior
- Eventually, (almost) all agents land on roost
- *Particle swarm*: roost = **optimal solution**

Particle Swarm Optimization

- **Multi-agent**: *swarm* of **particles**
- Particles **move** over *search space* (**exploration**) with certain **velocity**
- Particle is part of **neighborhood**

PSO Topologies



Particle Swarm Optimization

- Particle influenced by **self** and **swarm** *history*
- **Self-improvement** & **imitating** others
- *Dilemma*: Exploration vs Exploitation

Particle

Remembers:

- Current **position** in search space (solution + fitness)
- **Velocity** (speed + direction)
- **Individual best** position
- **Swarm** remembers **global best**

PSO Algorithm

- *Initial population*: random particles
- Evaluate **fitness** of particles (objective fn)
- Update individual and global **bests**
- Update **velocity**, **position** of each particle

PSO Algorithm

```
Input: ProblemSize, Population_size
Output: P_g_best
Population ← ∅;
P_g_best ← ∅;
for i = 1 to Population_size do
    P_velocity ← RandomVelocity();
    P_position ← RandomPosition(Population_size);
    P_cost ← Cost(P_position);
    P_p_best ← P_position;
    if P_cost ≤ P_g_best then
        P_g_best ← P_p_best;
    end
end
while ¬StopCondition() do
    foreach P ∈ Population do
        P_velocity ← UpdateVelocity(P_velocity, P_g_best, P_p_best);
        P_position ← UpdatePosition(P_position, P_velocity);
        P_cost ← Cost(P_position);
        if P_cost ≤ P_p_best then
            P_p_best ← P_position;
            if P_cost ≤ P_g_best then
                P_g_best ← P_p_best;
            end
        end
    end
end
return P_g_best;
```

Velocity Update

Search for new solutions

$$\mathbf{v}_i^{t+1} = \underbrace{\mathbf{v}_i^t}_{\textbf{Diversification}} + \underbrace{c_1 \mathbf{U}_1^t (\mathbf{pb}_i^t - \mathbf{p}_i^t) + c_2 \mathbf{U}_2^t (\mathbf{gb}^t - \mathbf{p}_i^t)}_{\textbf{Intensification}}$$

Exploits what good so far

$$\mathbf{v}_i^{t+1} = \underbrace{\mathbf{v}_i^t}_{inertia} + \underbrace{c_1 \mathbf{U}_1^t (\mathbf{pb}_i^t - \mathbf{p}_i^t)}_{personal\ \ influence} + \underbrace{c_2 \mathbf{U}_2^t (\mathbf{gb}^t - \mathbf{p}_i^t)}_{social\ \ influence}$$

c1, c2 = weight coefficients for personal, global best (usually 2)
u1, u2 = random variables between 0,1
pb = personal best of particle
gb = global best of swarm
p = position
v = velocity

Exploration vs Exploitation

- **Exploration**: explore more of search space, find new solutions
- **Exploitation**: take advantage of what you already know
- **Exploitation**: locally oriented search, approach (possibly local) optimum

Termination

- Best **solution** exceeds **quality** threshold
- Average **velocity** of agents falls below threshold (slow movement)
- After fixed number of **iterations**

Convergence

- **Failure**: swarm diverges, doesn't converge
- **Optimal**: global best = global optimum
- **Local**: global best = local optimum

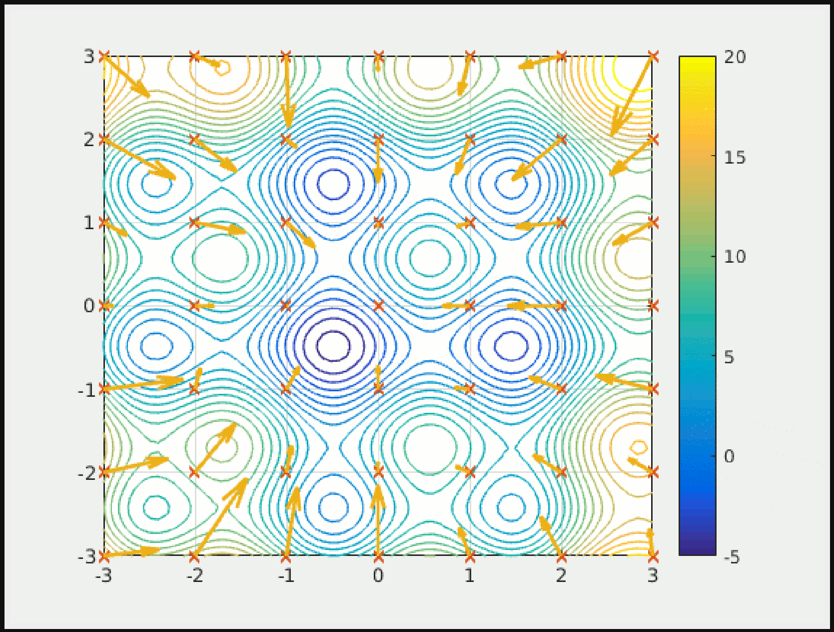
Advantages

- **Simple** implementation
- **Direct** search: no gradients, derivatives
- Few algorithm **parameters**
- **Asynchronous**: no central control
- **Efficient** global search algorithm

Disadvantages

- Tendency to **prematurely converge** to mid-optimum points
- **Slow convergence** in refined search

Particle Swarm Optimization



[https://www.youtube.com/watch?v=_bzRHqmpwvo]

Summary

Population-Based Search:

- **Genetic Algorithms**: fitness, selection, crossover, mutation
- **Particle Swarm Optimization**

Announcements

- **MP#2** within the week (Facebook Group)
- *Next Meeting*: MP#2 Discussion
- *Next Meeting*: Quiz on Local Search + Population-Based Search
- *Next Topic*: Machine Learning

References

- *Clever Algorithms*, J. Brownlee, 2011
- CS 188 Lec 5 slides, Dan Klein, UC Berkeley
- www.tutorialspoint.com/genetic_algorithms/
- www.swarmintelligence.org/tutorials.php

Questions?