AI VIET NAM
@aivietnam.edu.vn

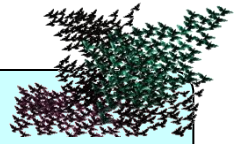
Randomness and its Applications to Optimization

Quang-Vinh Dinh
PhD in Computer Science

Year 2024

Objectives

Evolutionary Alg.

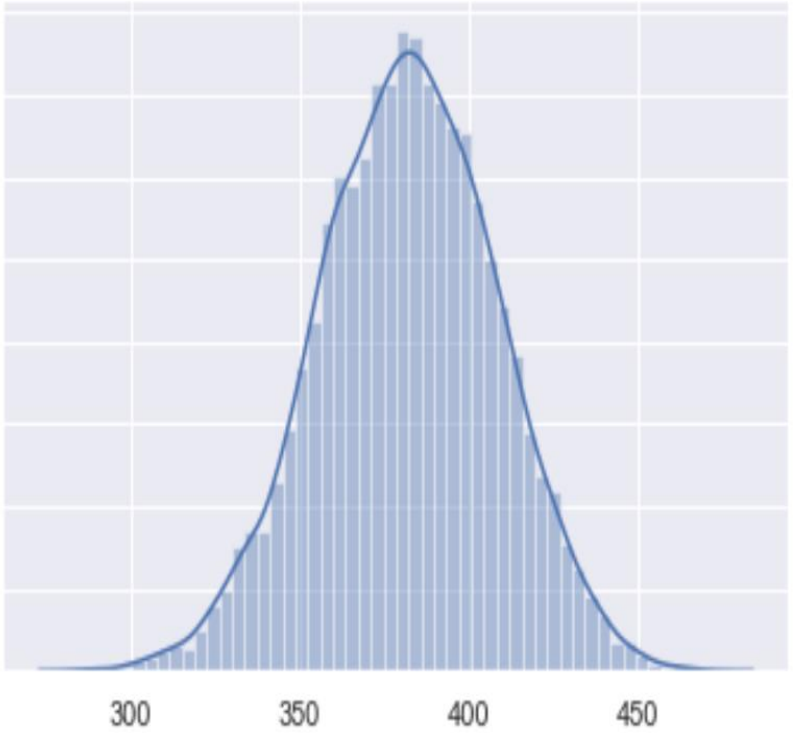


PSO
(Particle Swarm Optimizer)

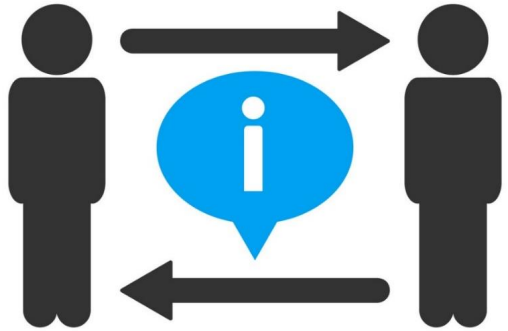
ACO
(Ant Colony Optimizer)

AIS
(Artificial Immune System)

Randomness



Steps to GAs



Outline

SECTION 1

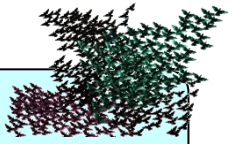
Evolutionary Algorithms

SECTION 2

Randomness

SECTION 3

Steps to GAs



PSO
(Particle Swarm Optimizer)

ACO
(Ant Colony Optimizer)

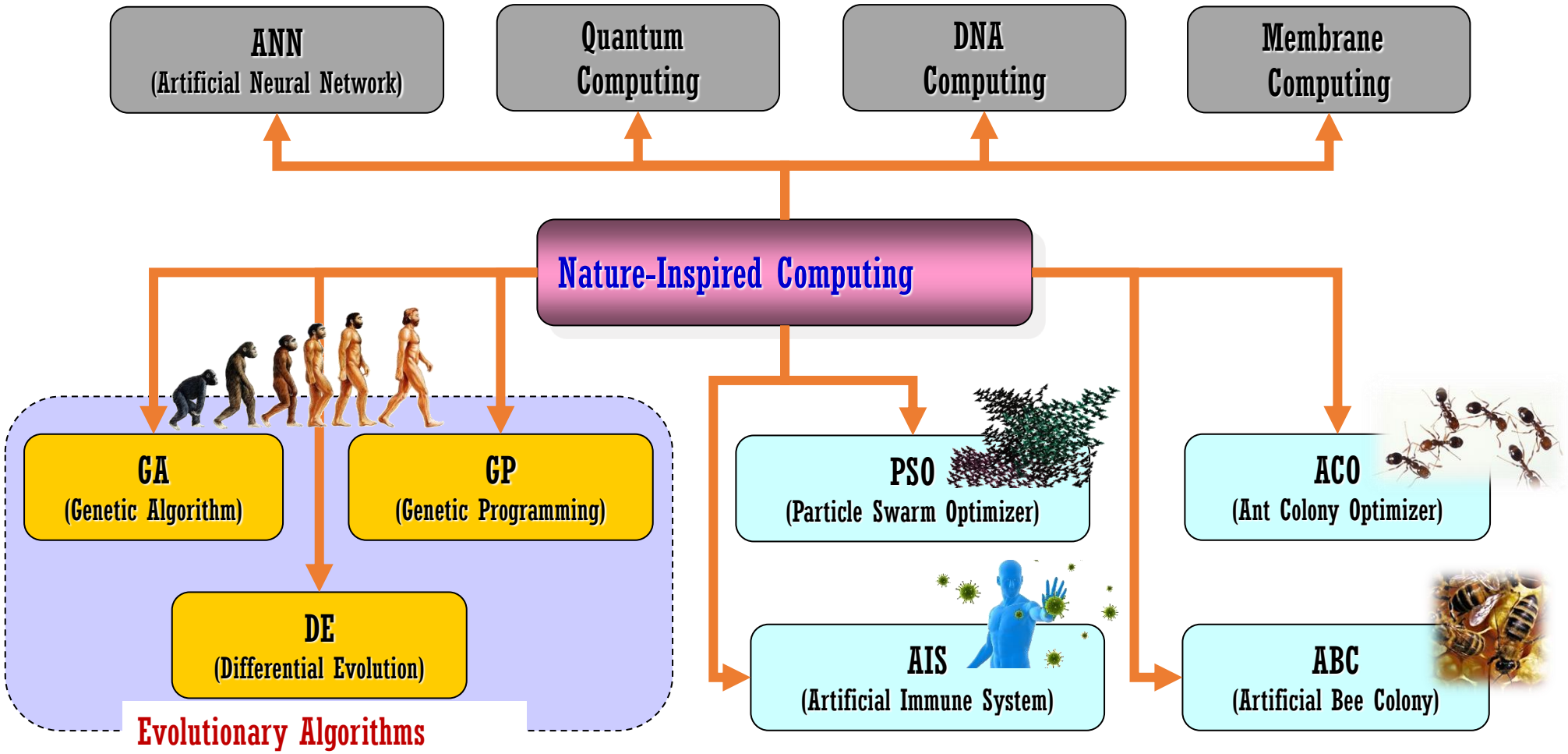
AIS
(Artificial Immune System)

ABC
(Artificial Bee Colony)

Applications

Slide from Prof. Ahn

Nature-Inspired Computing (Algorithms): Examples

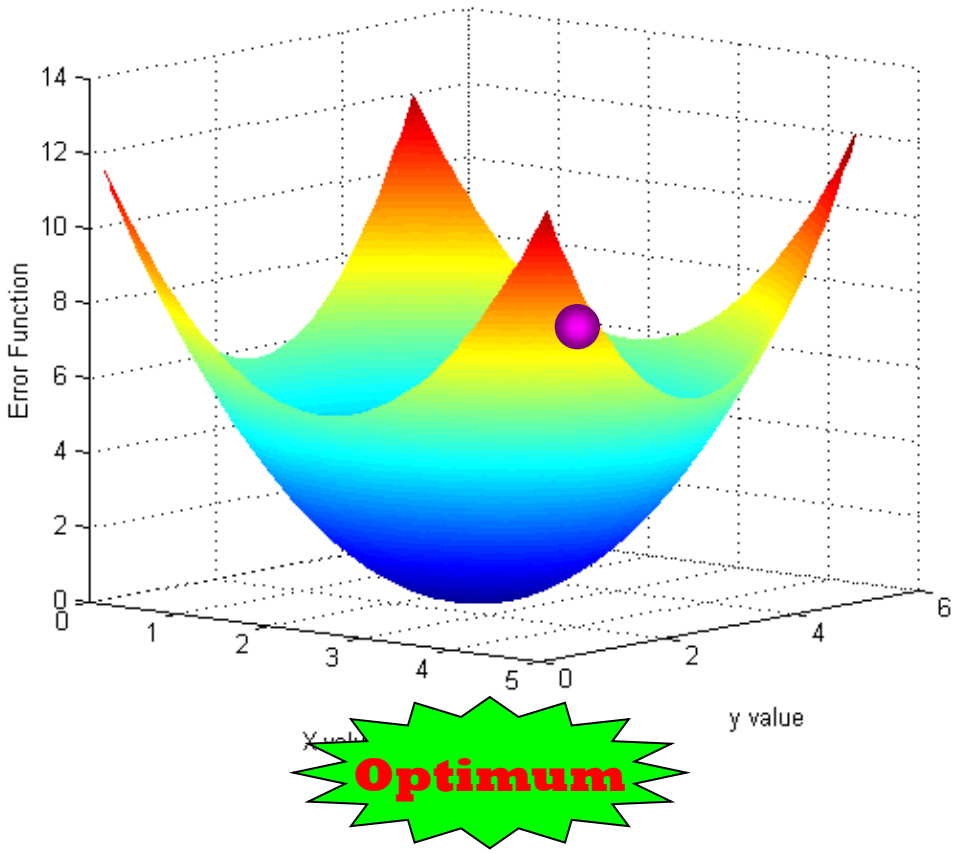


Applications

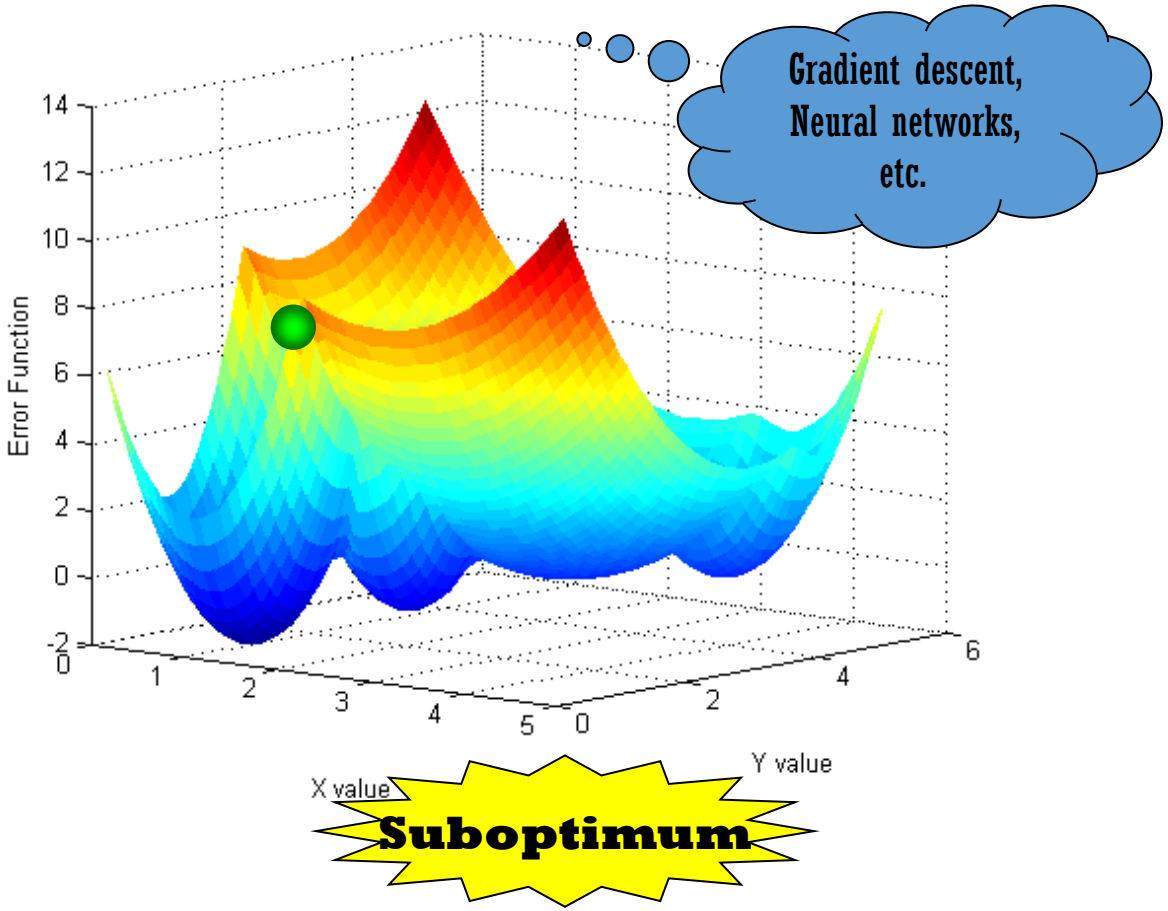
What's the Problem of Existing (Deterministic) Methods?

Slide from Prof. Ahn

Single modal case



Multiple modal case

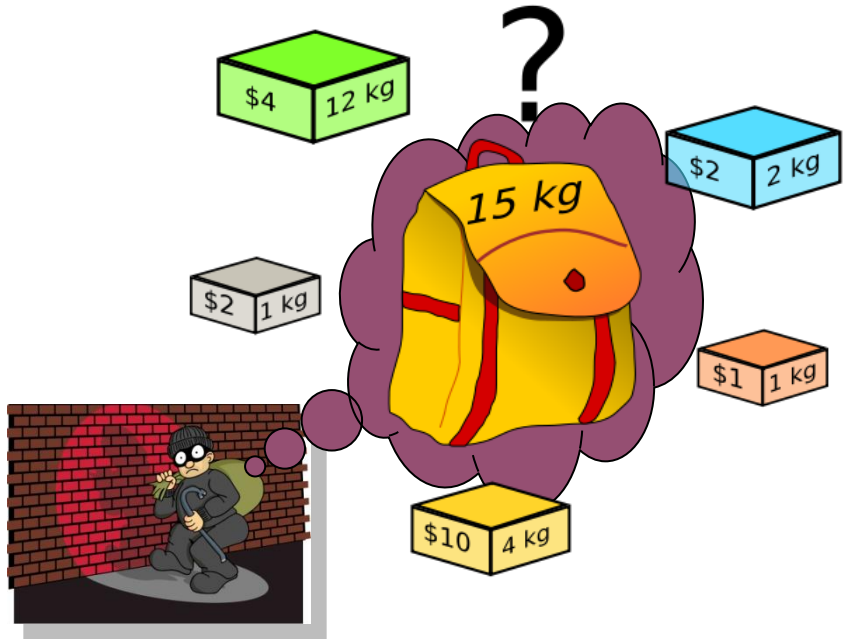


Classical Combinatorial Problems

Slide from Prof. Ahn

Traveling Salesman Problem

Knapsack Problem

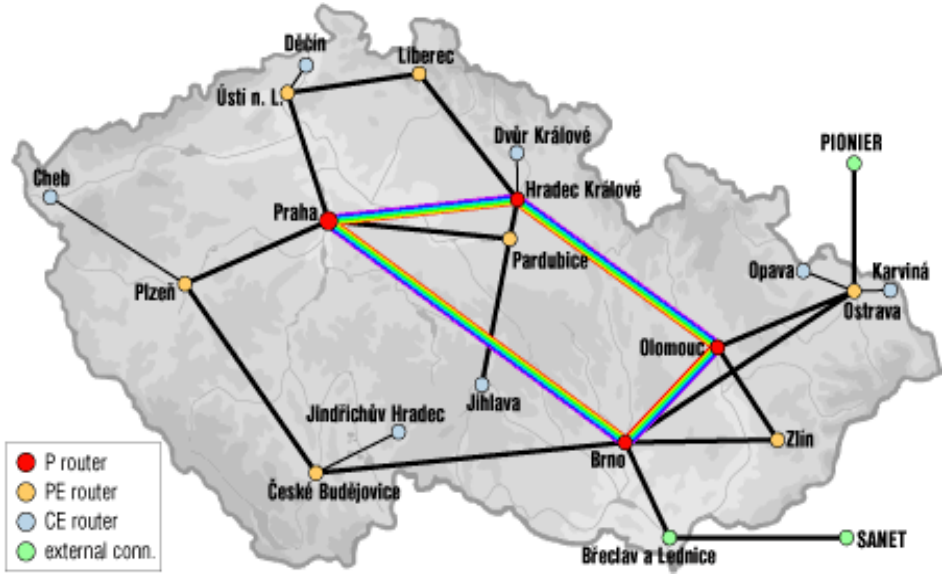


- Maximize the amount of profits (e.g., money) while still keeping the overall weight under or equal to a given limit!

Communication & Networks

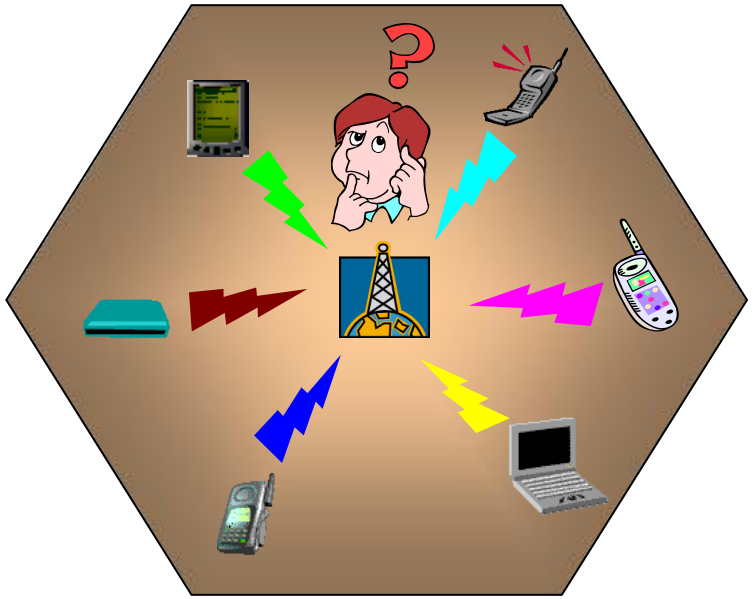
Slide from Prof. Ahn

Multicast Routing



- Minimize the cost of multicast tree while satisfying delay and bandwidth constraints

Resource Allocation

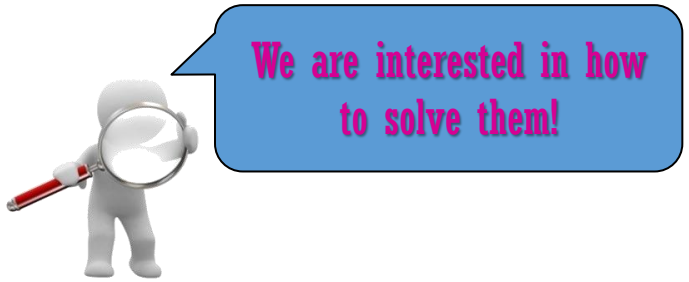


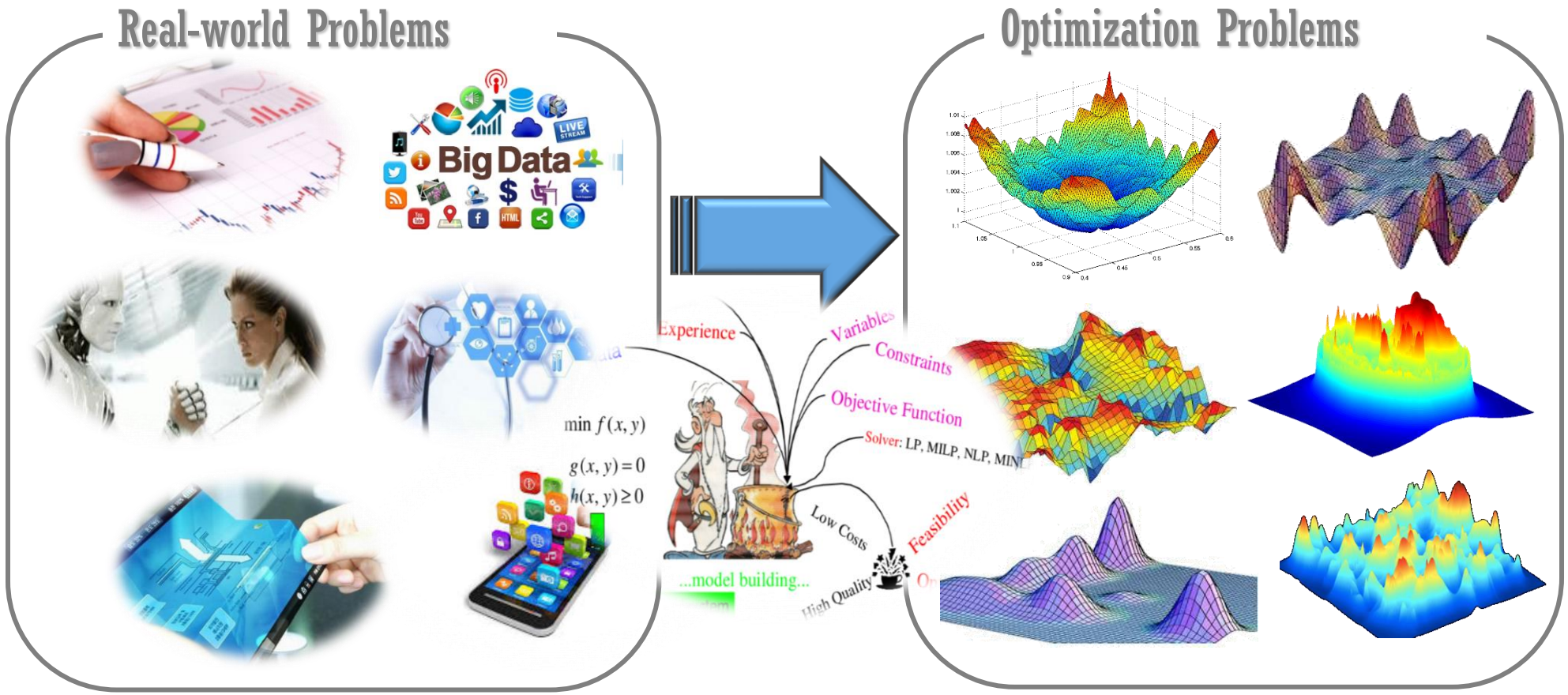
- Maximize resource utilization by fairly distributing wireless resources among the connections

Applications

What is the Target of Interest?

- All the Problems in Any Field!
- That is, a sort of Optimization Problem!

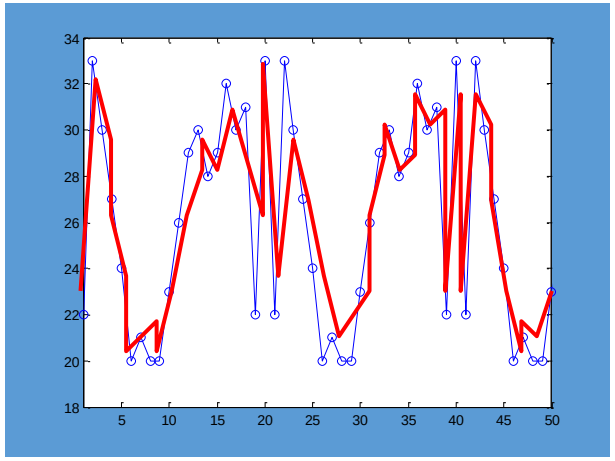




Economic Science

Slide from Prof. Ahn

Time-Series Forecasting



- Predicting some future outcomes from a set of historical events
- Stock prediction, Weather forecasting, etc.

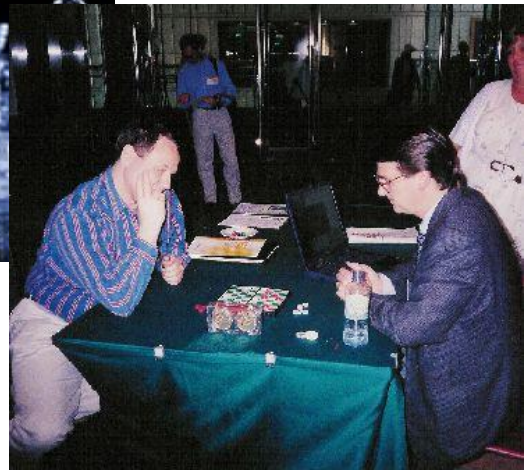
Decision in Dilemma

- Choosing a decision in conflict objectives
- Prisoner's dilemma, Game theory, etc.

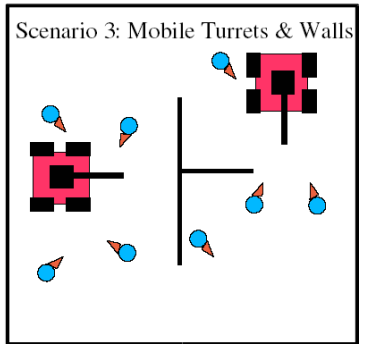
Game

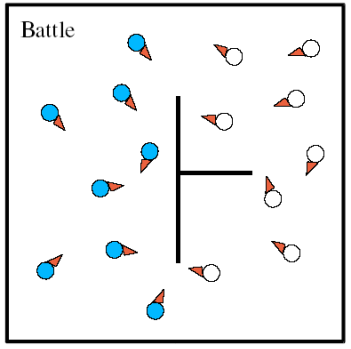
Slide from Prof. Ahn

- **Evolutionary Checker**
  - 8X8 board, 12 checkers for each player
    - Diagonal moves, Jumps are forced, etc.
  - Neural Networks + Evolutionary Prog.
    - Checkerboards are evaluated by NNs
    - NNs and King value are evolved with EP
  - Almost the expert level without knowledge



- **Video Game: NERO**
  - Univ. of Texas at Austin
  - Player's role
    - Train agents for competition
  - No prepackaged or scripted agents
  - Evolve in real-time

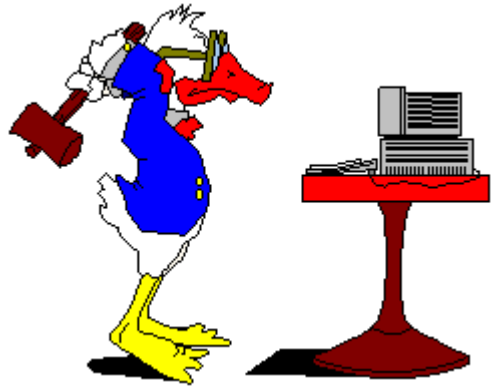




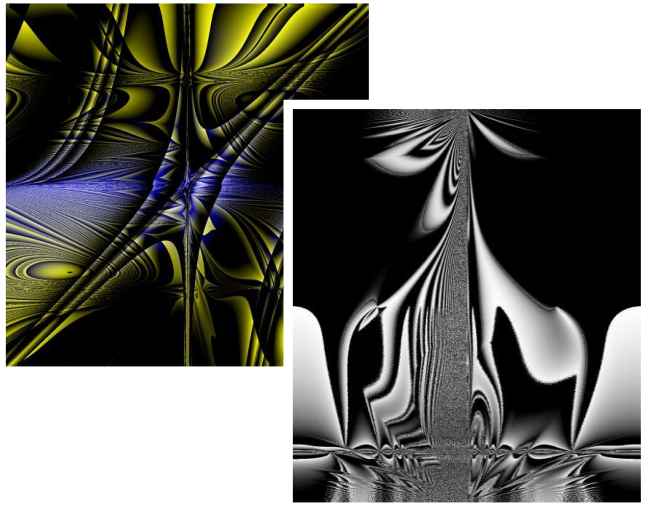
Art

Slide from Prof. Ahn

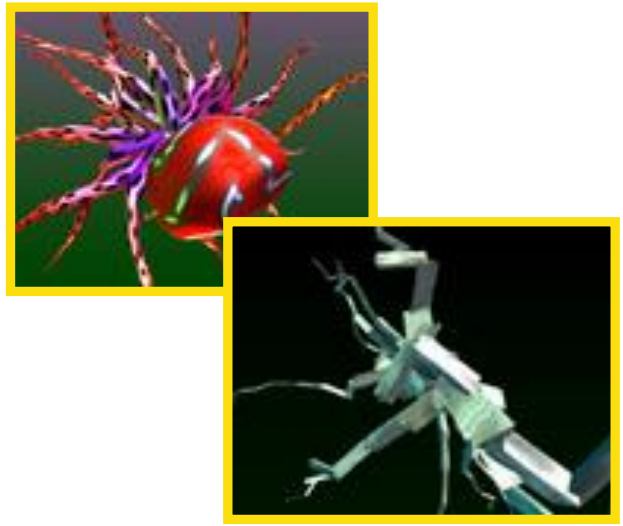
- **What's Evolutionary Art?**
  - Technically, it is creating pieces of art through human-computer interaction
  - Computer runs evolutionary algorithms and human applies subjective selection
    - Role of computers: offer choices and create diversity
    - Role of human: make (subjective) choices and reduce diversity
  - Selection (aesthetic/subjective) steers towards implicit user preferences



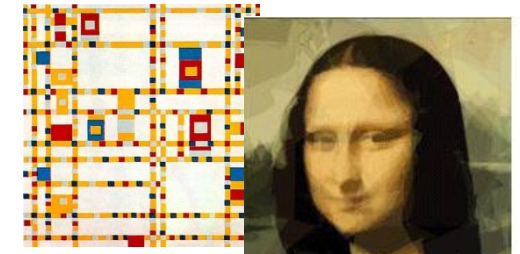
Evol. Art by Kleiweg

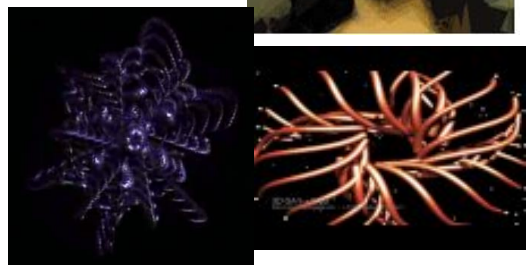


Galapagos by Karl Sims



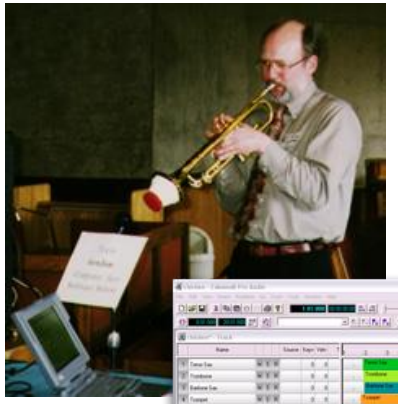
Other Examples





Music

- GenJam (Genetic Jammer)
  - Developed in 1993~94 by Prof. John Al Biles
  - Interactive GA that learns to play jazz solos
  - GemJam's repertoire: Over 250 jazz-style tunes
  - Evolving by special fitness operators;
    e.g., rhythm conformity
  - What can it be done?
    - ✓ Playing full-chorus improvised solos
    - ✓ Listening to trumpet and responds interactively when we trade fours
    - ✓ Engaging in collective improvisation; we both solo simultaneously and GenJam performs a smart echo of improvisation
    - ✓ Listening to me and play the head of a tune and breeds my measures



Source: http://www.it.rit.edu/~jab/GenJam.html

Source: http://phoenix.inf.upol.cz/~dostal/evm.html

Virtual quintet

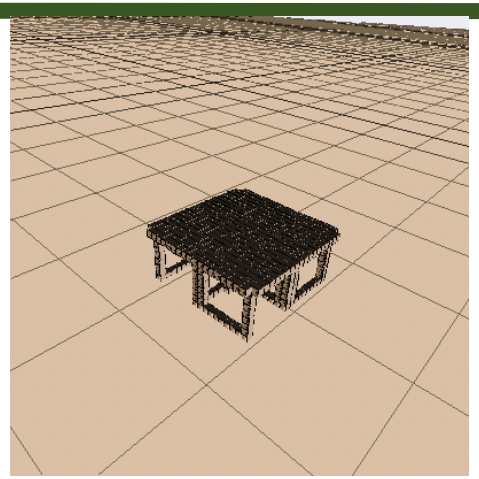
MusiGenesis

Design

Slide from Prof. Ahn

Structure Design

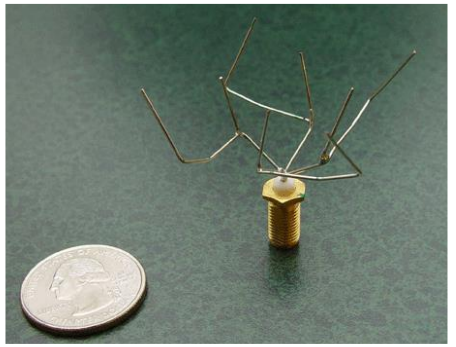
- Bridge structure optimization
- Building structure design

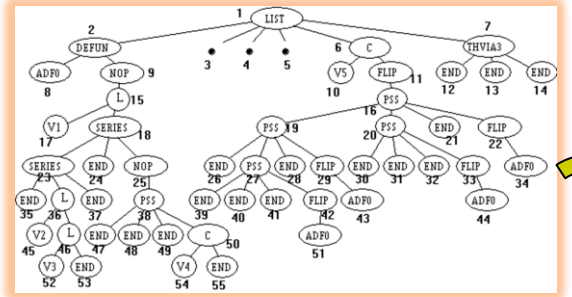


Circuit Design

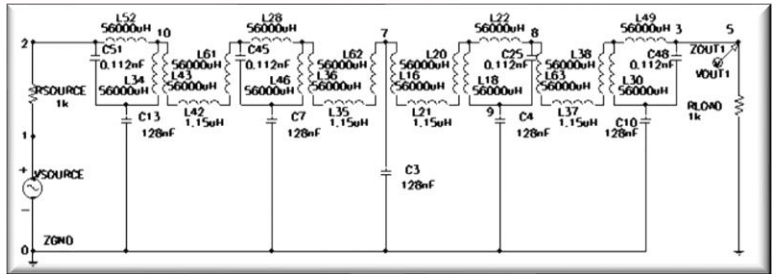
- Automatic synthesis of topology & sizing of analog electrical circuits

Evolved antenna (NASA, 2004)





Lowpass filter

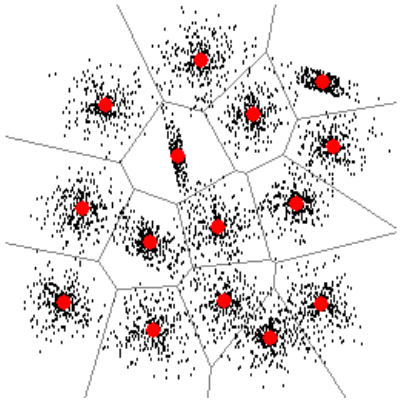


Aviation System Design

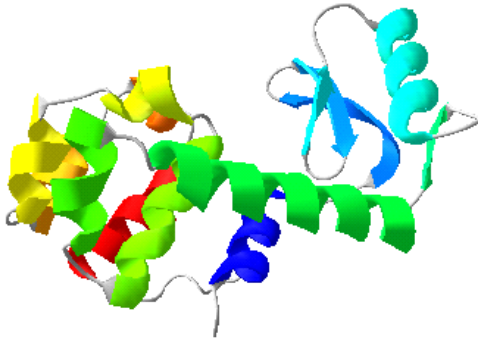
- Airfoil, wing, and antenna designs
- Space platform structure optimization
- Jet aircraft model optimization

Information Mining

- **Clustering**
  - Data clustering
  - Text mining
  - Web search



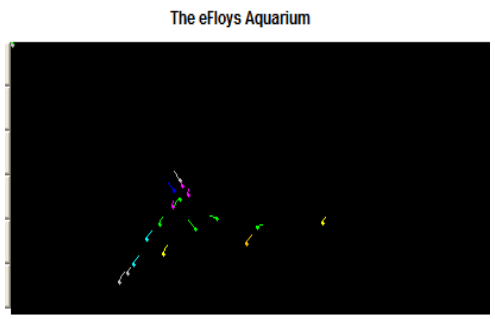
- **Bioinformatics**
  - Drug discovery
  - Protein folding
  - Cancer diagnosis

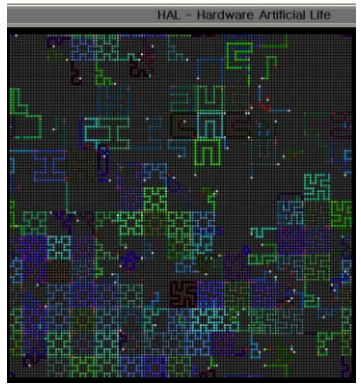


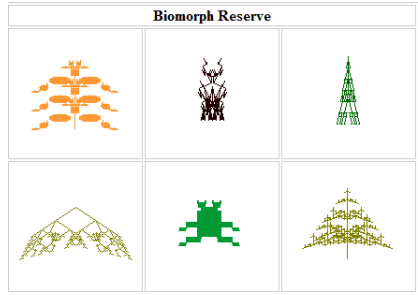
Artificial Creatures & Robotics

Slide from Prof. Ahn

- **Artificial Creatures**
  - eFly, Biomorph, HAL,
  - Self-replicating Worms
  - Gozilla, Solitaire







- **Robotics**
  - Humanoid Robots; e.g., e.g., ASIMO
  - Genetic Robots; e.g., Gene
  - Others; e.g., Six-Legged Robot
    Robot Snake

Outline

SECTION 1

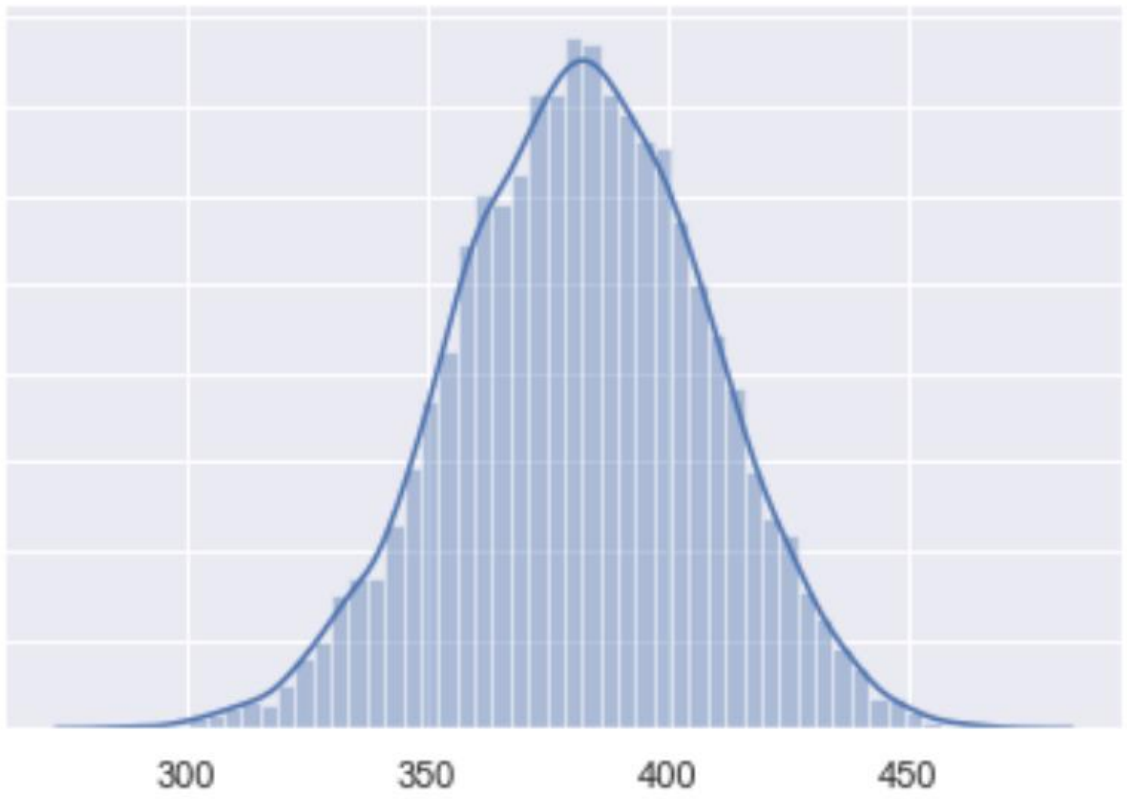
Evolutionary Algorithms

SECTION 2

Randomness

SECTION 3

Steps to GAs



Randomness Applications

❖ Simulation of coin tossing

Count #heads

Count #tails

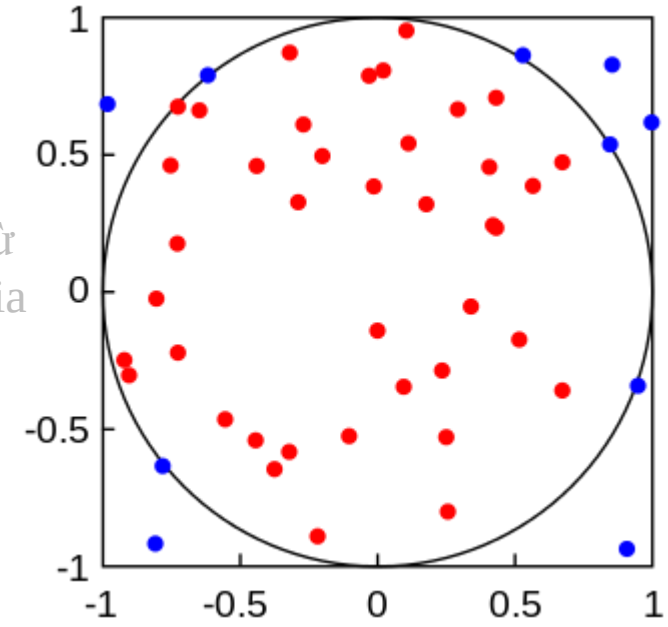
Check if the two numbers are similar

```
1.  # aivietnam.ai
2.  import random
3.
4.  # Tổng số lần búng đồng xu
5.  total_flips = 0
6.
7.  # số lần mặt sau xuất hiện
8.  num_tails = 0
9.
10. # số lần mặt trước xuất hiện
11. num_heads = 0
12.
13. for _ in range(1000):
14.     # sinh số ngẫu nhiên nằm trong khoảng [0,1)
15.     n = random.random()
16.     if n < 0.5:
17.         num_tails = num_tails + 1
18.     else:
19.         num_heads = num_heads + 1
20.
21.     # code ở vị trí này không thuộc khối else
22.     total_flips = total_flips + 1
```

Randomness Applications

❖ PI estimation

hình từ wikipedia



circle radius $r = 1$

circle_area $A_c = \pi r^2$

square side $s = 2$

square_area $A_s = s^2$

$$\frac{A_s}{A_c} \approx \frac{N_s}{N_c}$$

$$\frac{s^2}{\pi r^2} \approx \frac{N_s}{N_c}$$

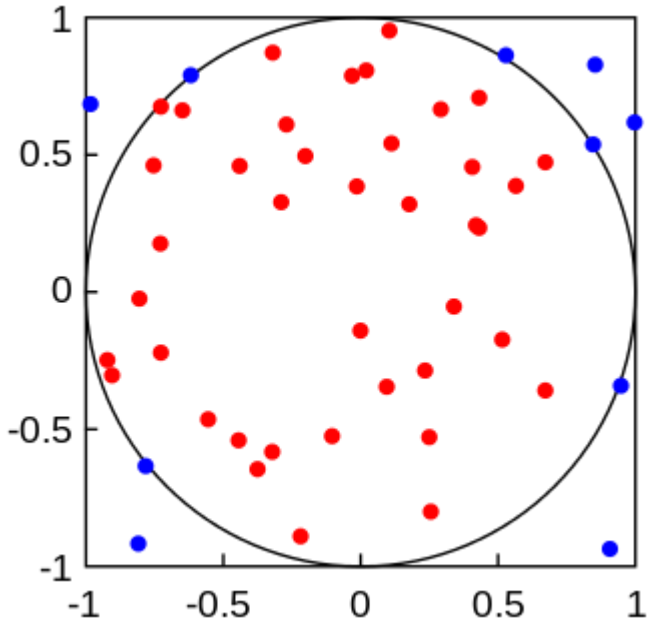
$$\pi \approx \frac{s^2 N_c}{N_s}$$

N_s is #random samples within the square generated according to uniform distribution

N_c is #random samples within the circle generated according to uniform distribution

Randomness Applications

❖ PI estimation



(hình từ wikipedia)

$$\pi \approx \frac{s^2 N_c}{N_s}$$

```python
# aivietnam.ai
import random
import math


# Tổng số điểm p được sinh ra
N = 100000

# số điểm thuộc tình tròn
N_T = 0

# Sinh ra N điểm ngẫu nhiên
for i in range(N):
    # sinh ra x, y thuộc [-1, 1].
    x = random.random()*2 - 1
    y = random.random()*2 - 1

    x2 = x**2
    y2 = y**2

    # kiểm tra p có nằm trong đường tròn
    if math.sqrt(x2 + y2) <= 1.0:
        N_T = N_T + 1

# tính PI
pi = (N_T / N) * 4
print(pi)
```

Gaussian Distribution

Công thức

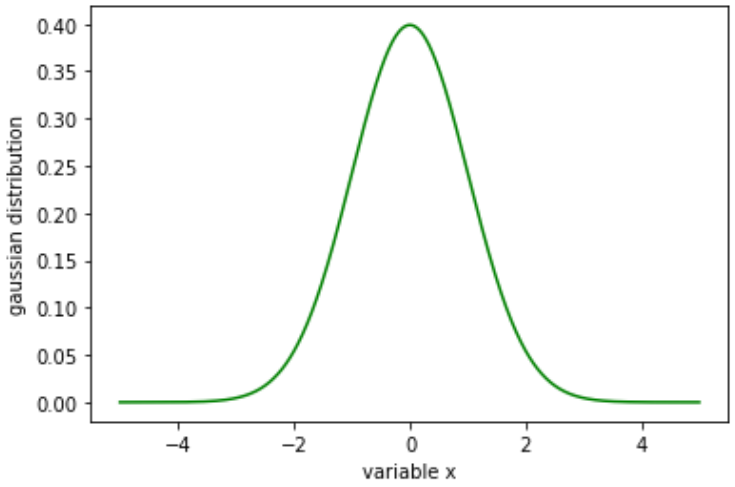
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$$

$-\infty < x < \infty$

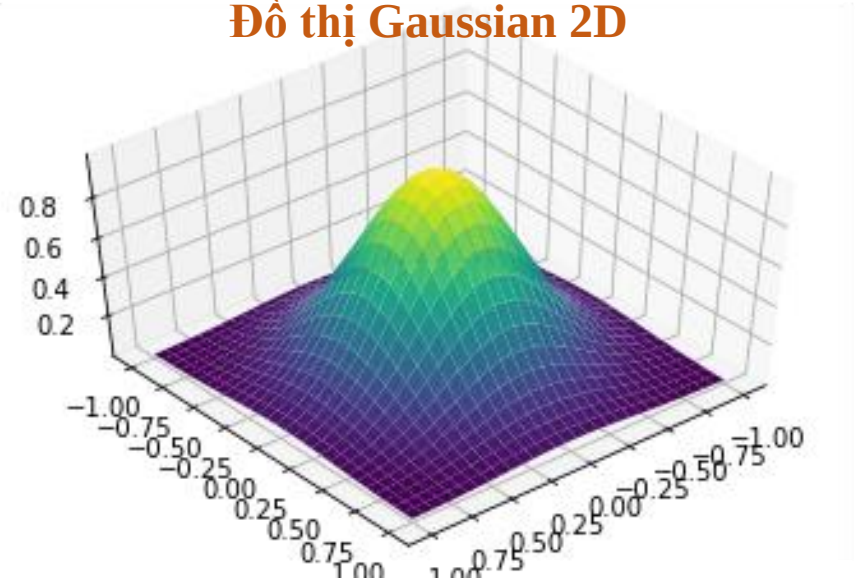
μ: mean

σ^2: variance

Đồ thị Gaussian 1D



Đồ thị Gaussian 2D



Ví dụ Cho $\mu = 0$ và $\sigma^2 = 1$. Tính giá trị hàm Gaussian với những giá trị x sau

$$f(x=-4) = \frac{1}{1\sqrt{2\pi}} e^{-\frac{1}{2*1}(-4-0)^2} = \frac{e^{-8}}{\sqrt{2\pi}} = \frac{1}{e^8\sqrt{2\pi}} = 1.3e{-}04$$

x = [-4 -3 -2 -1 0 1 2 3]

$f(x)$ =[1.3e-04 4.4e-03 5.3e-02 2.4e-01 3.9e-01 2.4e-01 5.3e-02 4.4e-03]

Distribution

❖ Definition

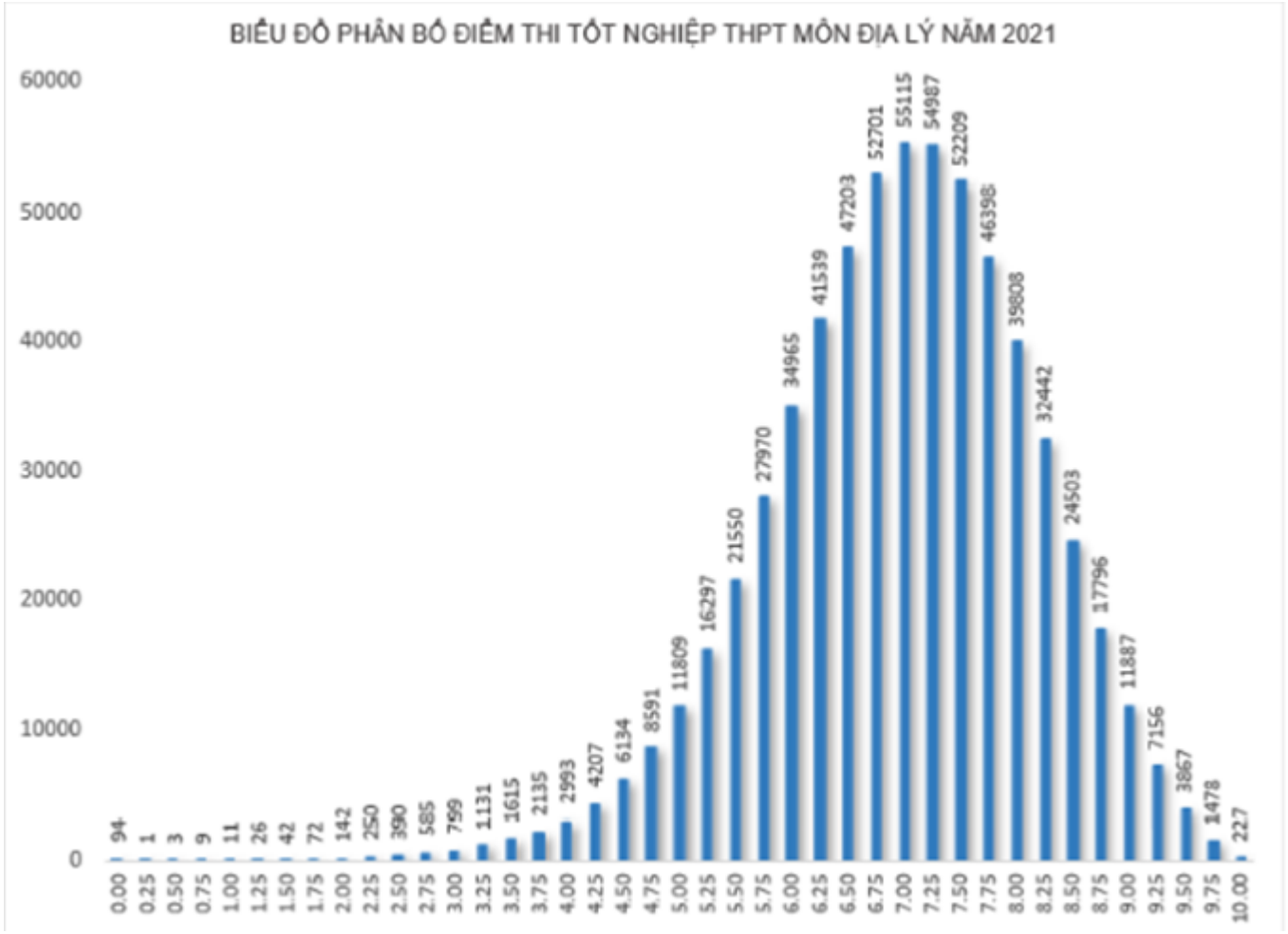
A distribution is simply a collection of data on a variable

Data are arranged in order

Statistics in Plain English

The probability distribution for a random variable describes how the probabilities are distributed over the values of the random variable.

https://www.britannica.com/science/statistics/Random-variables-and-probability-distributions



https://moet.gov.vn/tintuc/Pages/tin-tong-hop.aspx?ItemID=7451

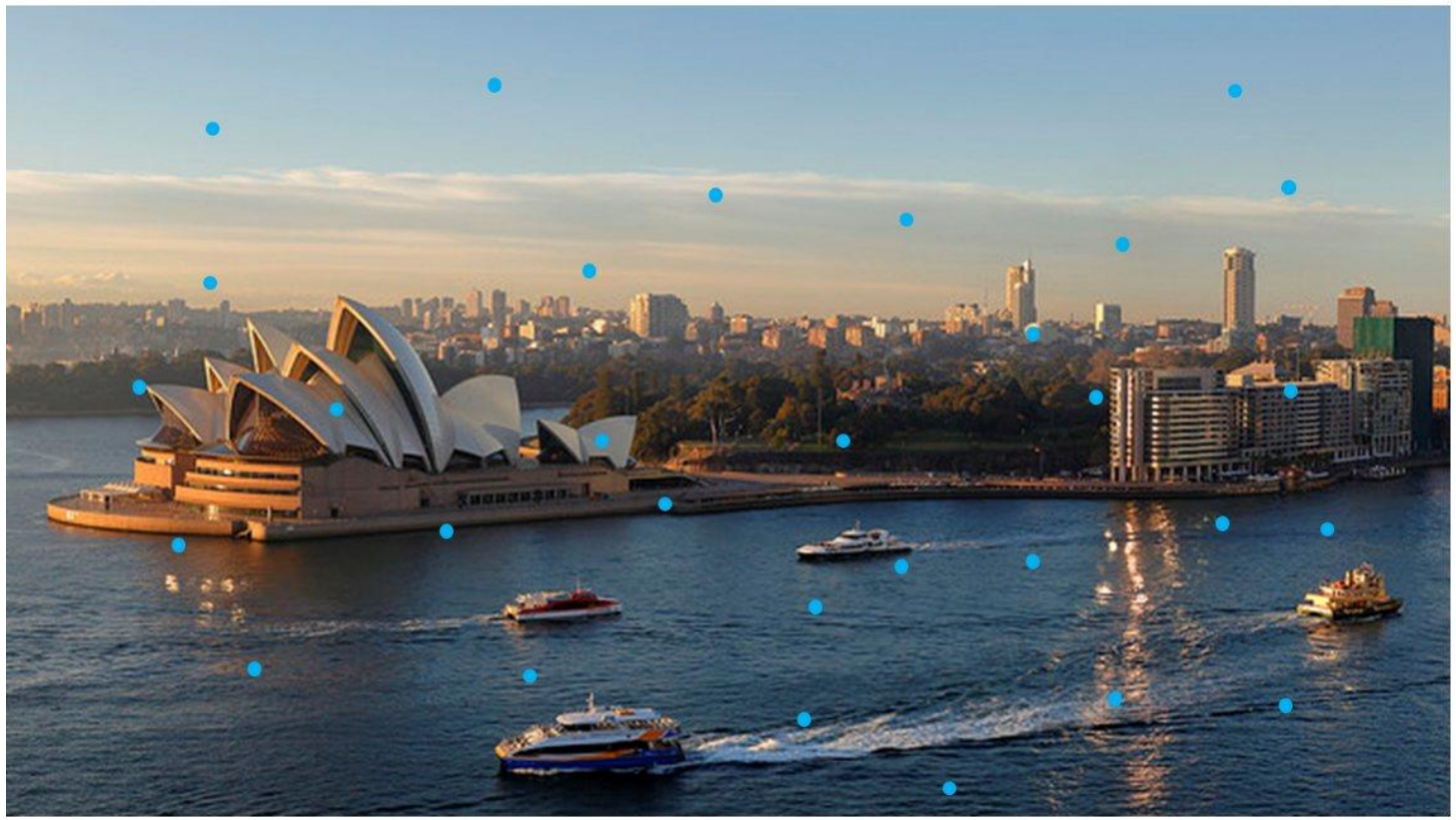
Central Limit Theorem

Randomly selected 30 pixels

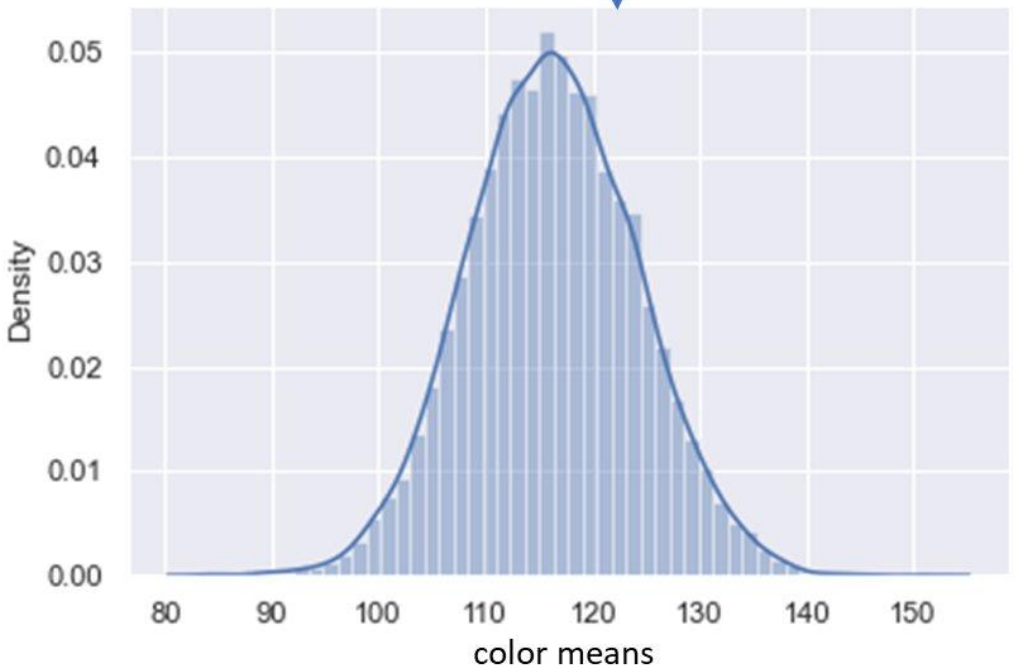
Compute color mean for the 30 pixels

Repeat 10000 times

Histogram of the 10000 color means



Population: pixel colors of the entire image



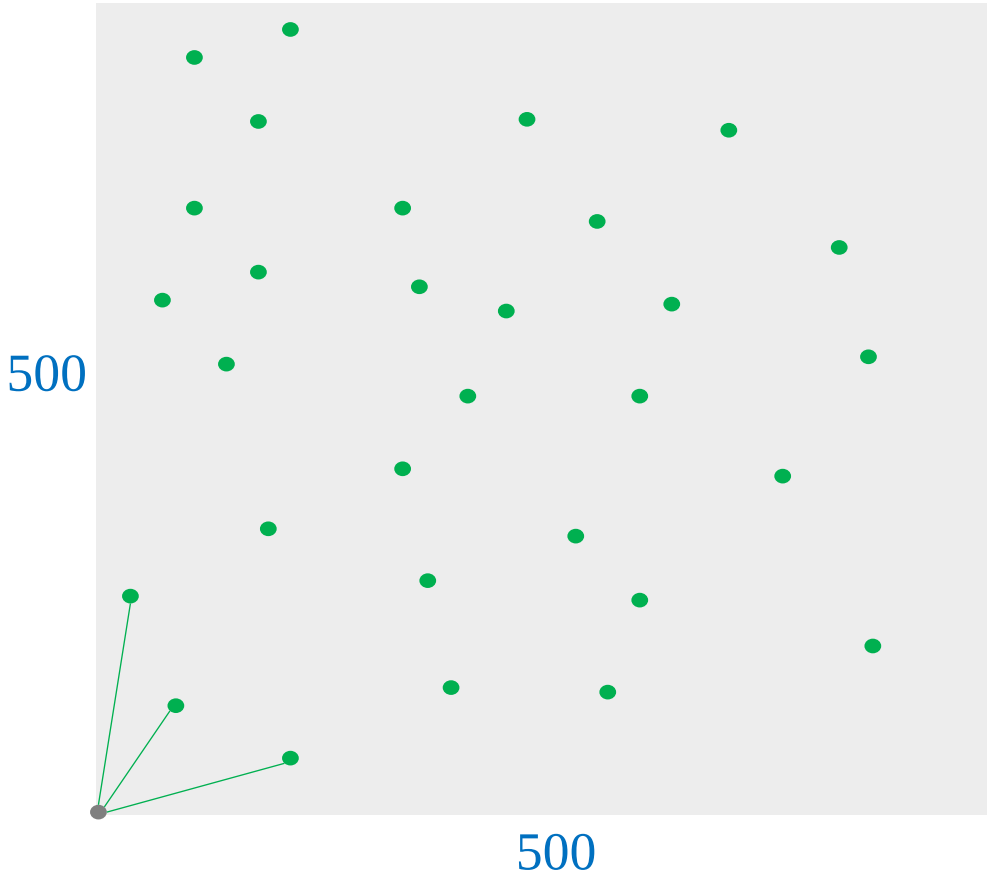
Example

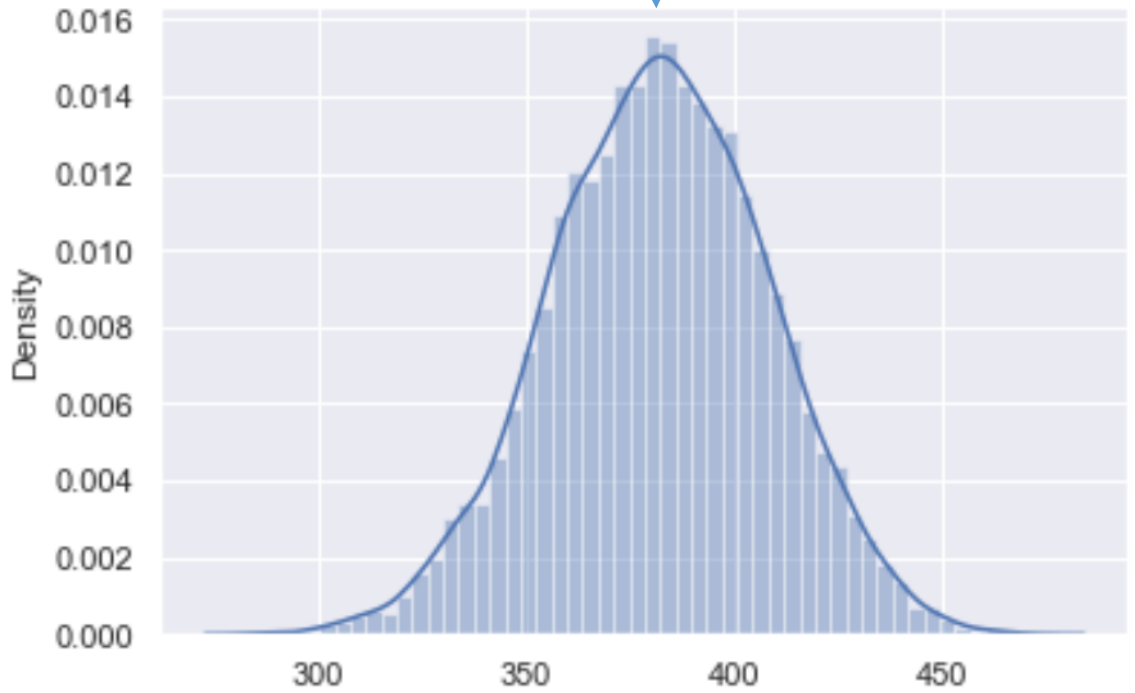
Randomly selected 30 pixels

Compute distance mean

Repeat 10000 times

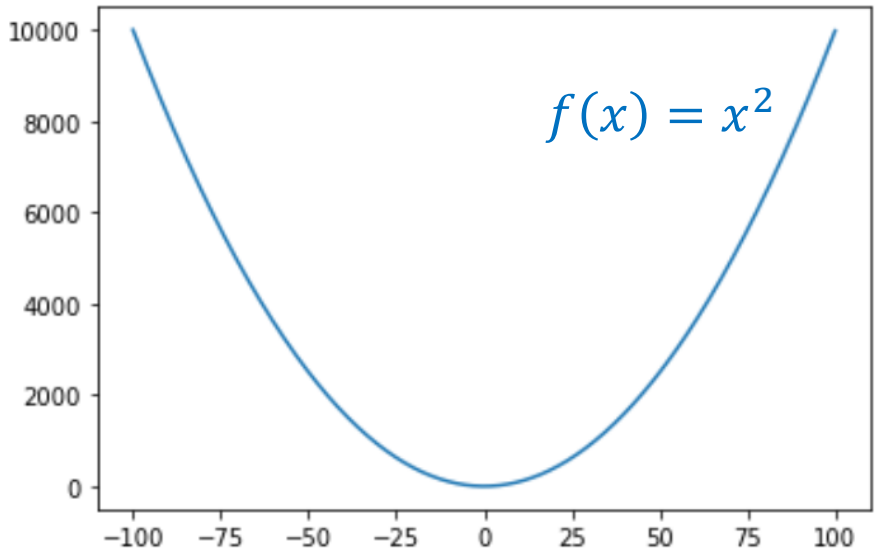
Histogram of 1000 distance means





Randomness Applications

❖ Optimization

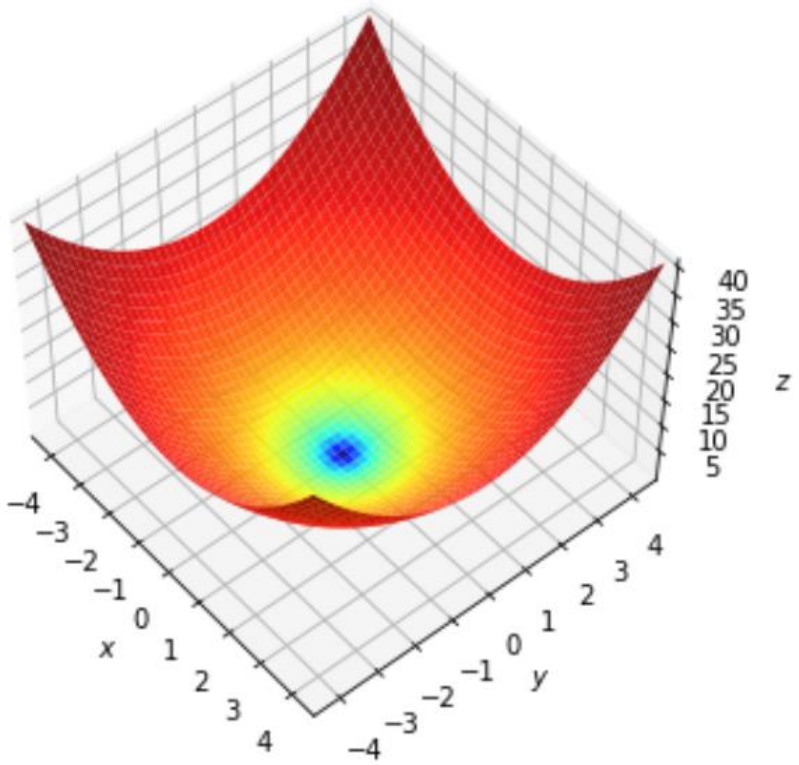


$$-100 \leq x \leq 100$$

$$x \in \mathbb{N}$$

$$f(x, y) = x^2 + y^2$$

$$-100 \leq x, y \leq 100; x \in \mathbb{N}$$



Outline

SECTION 1

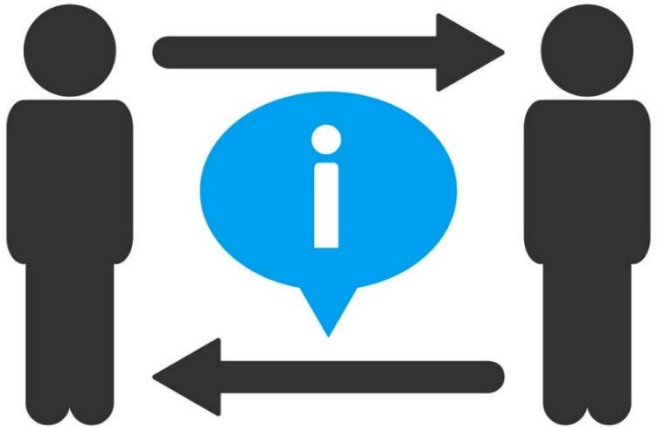
Evolutionary Algorithms

SECTION 2

Randomness

SECTION 3

Steps to GAs



NP-Hard Problems

❖ One-max problem

Each vector v has the length of n

$n = 10$

vector v1

1	1	0	1	0	1	1	0	0	1

secret(v1) → 6

vector v2

1	0	0	0	1	0	0	1	0	0

secret(v2) → 3

vector v1

1	1	0	1	0	1	1	0	0	1

secret(v1) → 4

vector v2

1	0	0	0	1	0	0	1	0	0

secret(v2) → 7

Possible

Secret information

$$\text{secret}(v) = \sum_i v_i$$

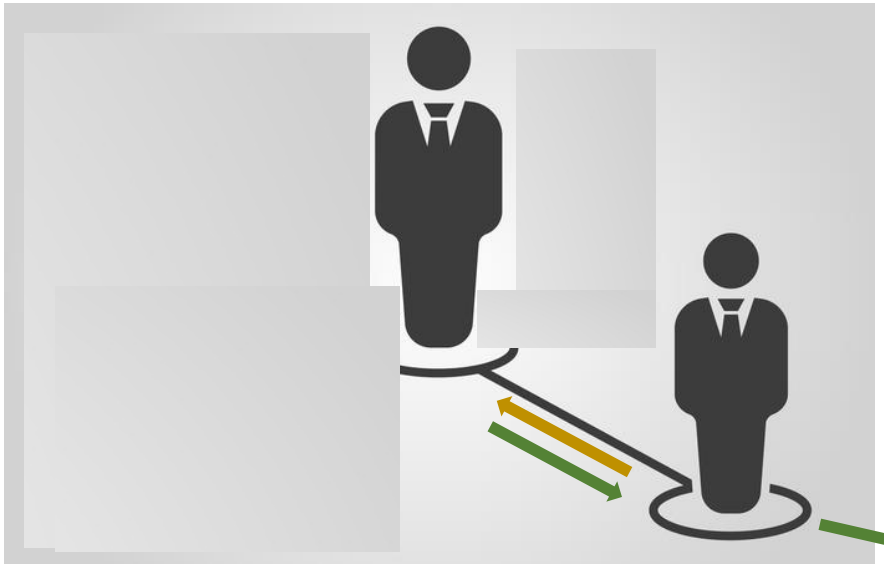
The body of the secret function is unknown

Design an algorithm for general purpose

Scenario 1

Searching Space

Decision maker (M)



Worker W1

1) Only one worker
2) Enough $ only for one round

Assign a random location

Assign a random location

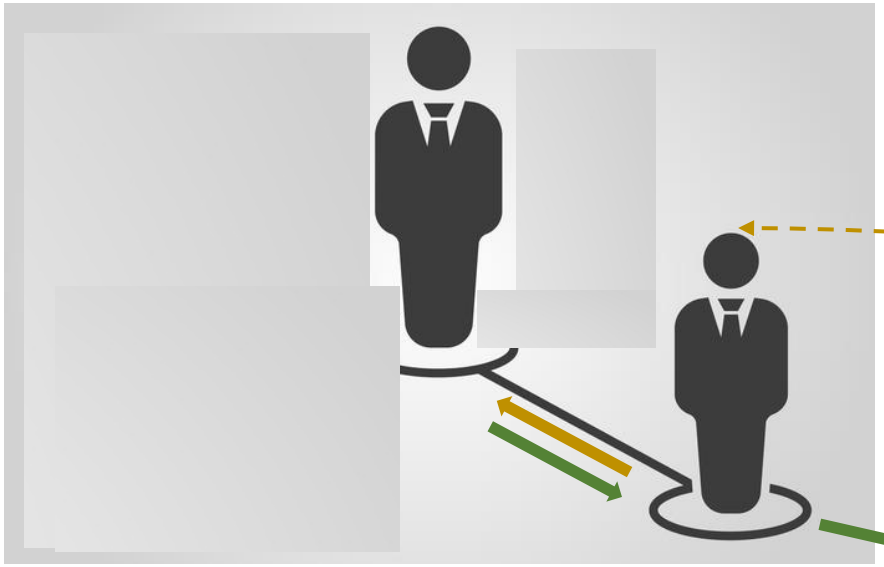
Feedback its score

Scenario 1

The score from W1 is 10

Searching Space

Decision maker (M)



Worker W1

1) Only one worker
2) Enough $ only for one round

Assign a random location

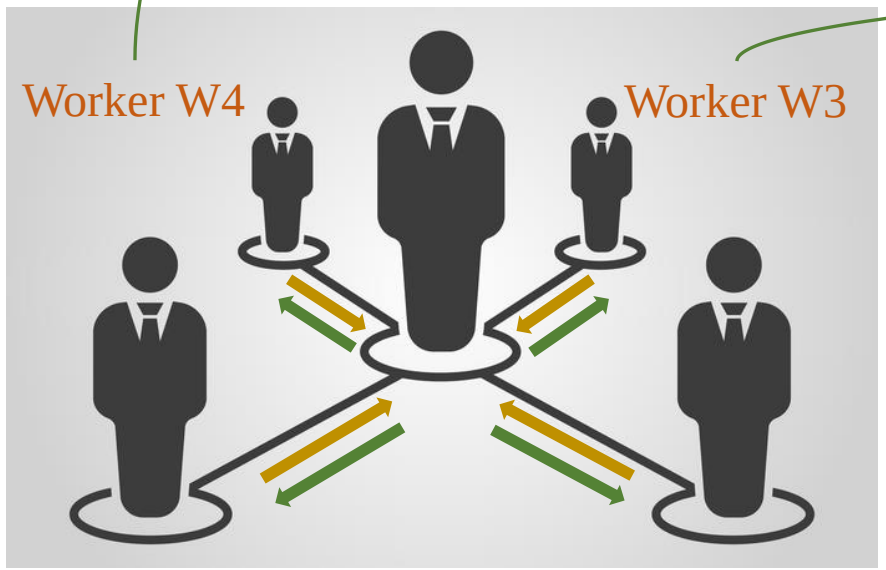
Assign a random location

Feedback its score

Scenario 2: Given some $

Searching Space

Decision maker (M)



1) Multiple workers
2) Work independently
3) Enough $ only for one round

Assign a random location

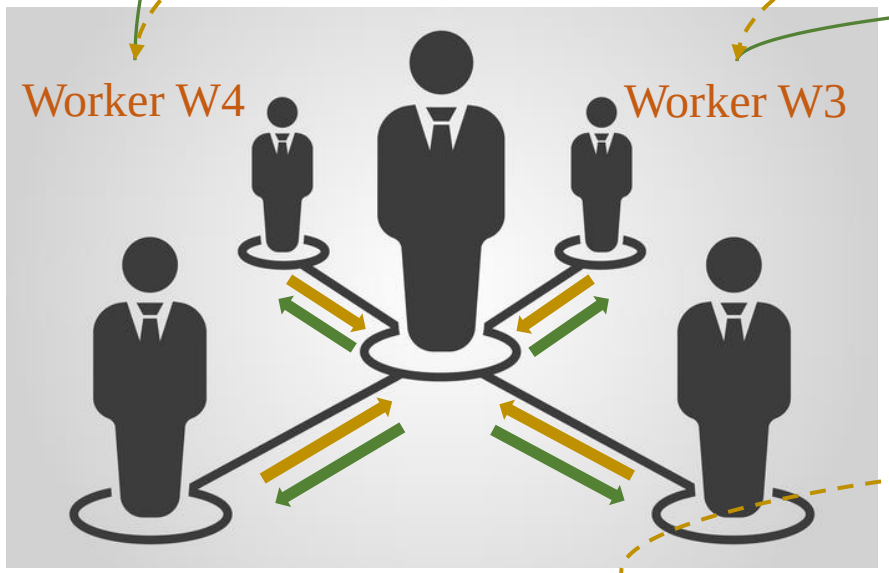
Feedback its score

Scenario 2: Given some $

Score1: 5
Score2: 7
Score3: 2
Score4: 8

Searching Space

Decision maker (M)



1) Multiple workers
2) Work independently
3) Enough $ only for one round

Assign a random location

Feedback its score

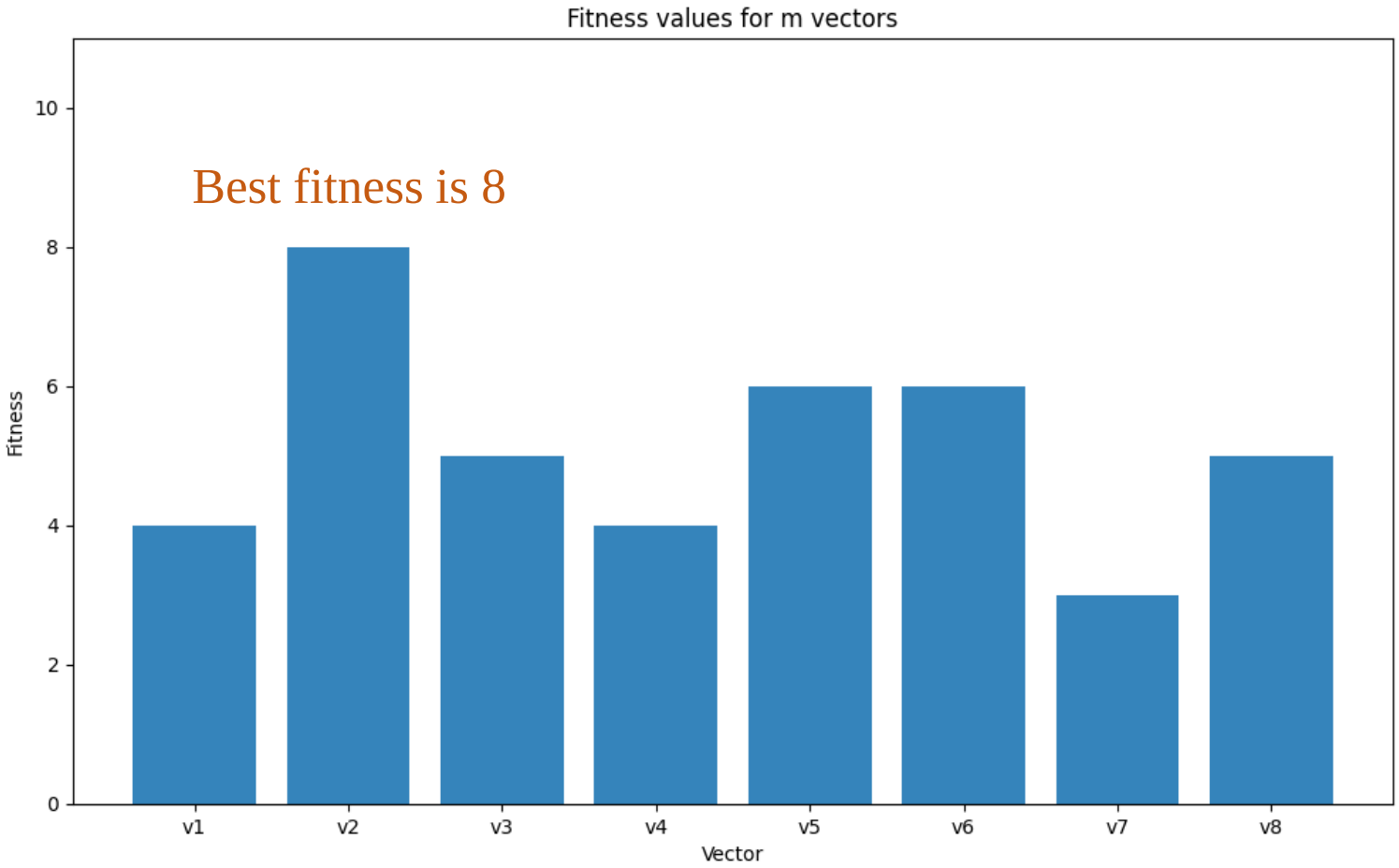
Random Search

❖ How to obtain vectors with good scores

Generate m vectors randomly

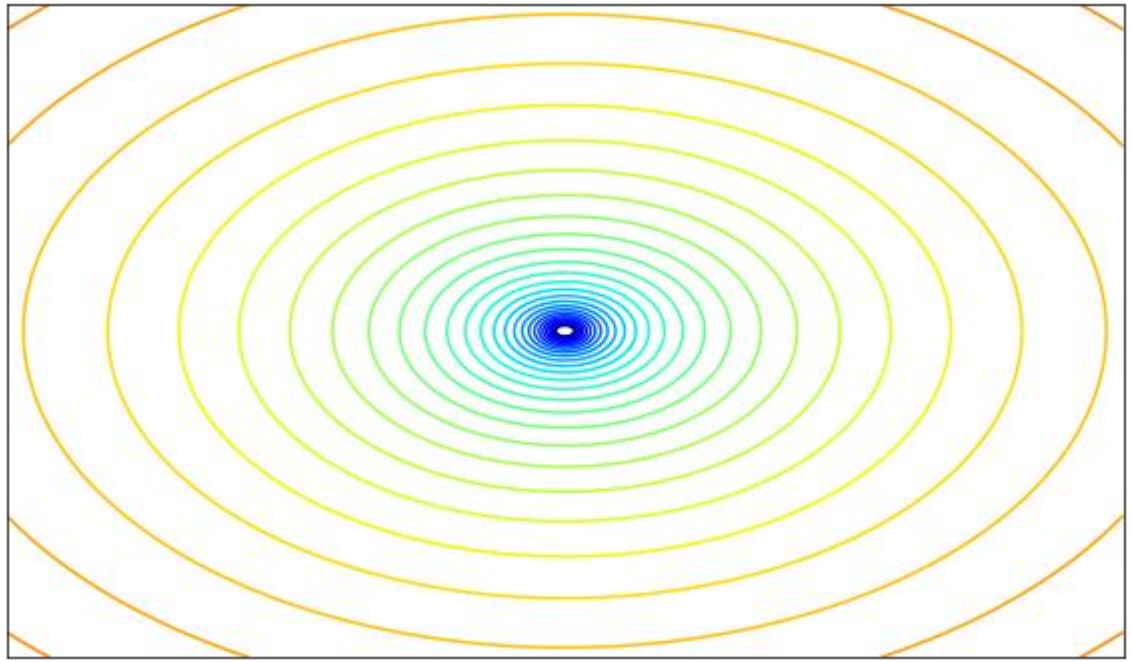
v_i receives value 0 or 1 randomly

```python
def generate_vectors(n=10, m=8):
    vectors = [[0]*n for _ in range(m)]

    for i in range(m):
        for j in range(n):
            if random.random() >= 0.5:
                vectors[i][j] = 1

    return vectors
```

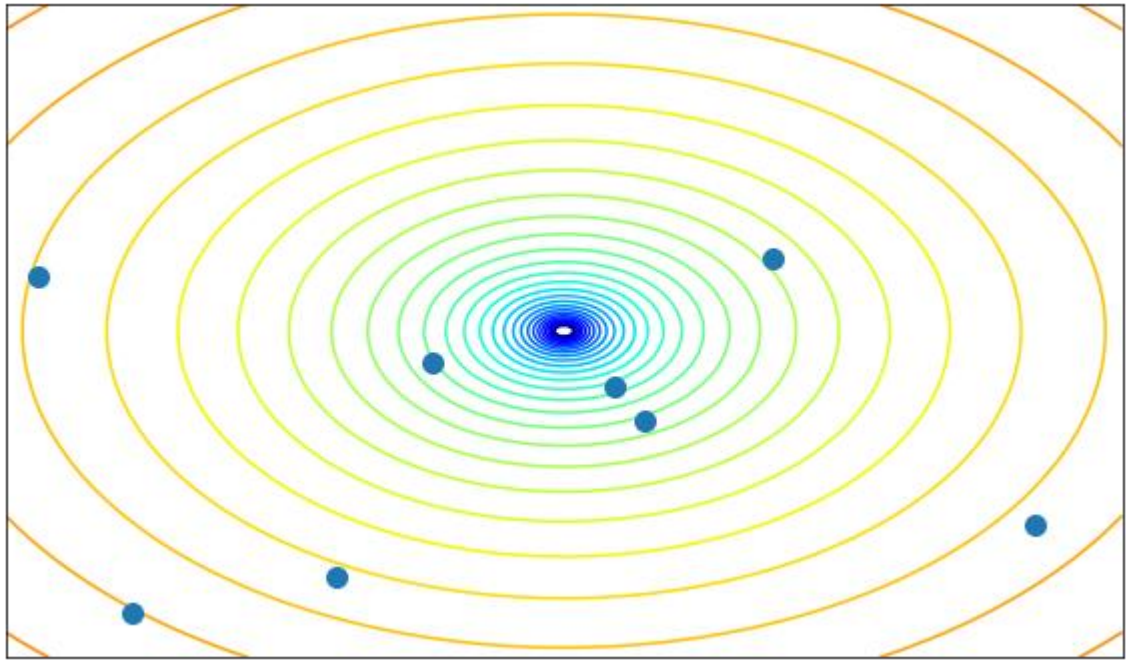


Random Search

Searching space



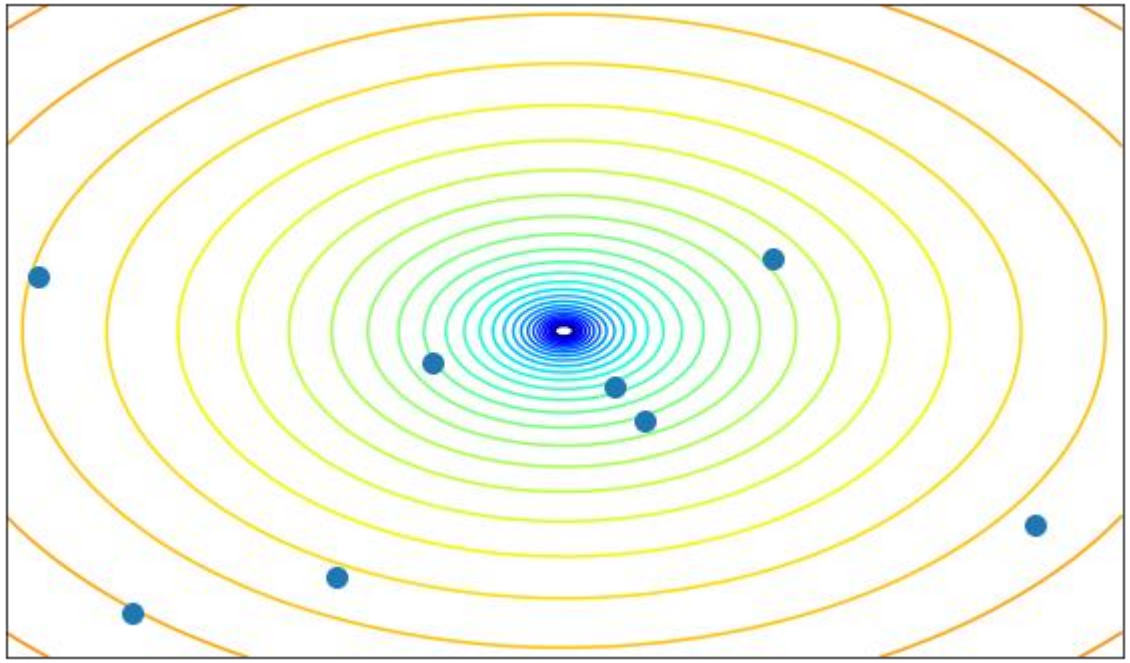
M randomly generated vectors



Random Search

❖ **Increase m to infinity**

m randomly generated vectors

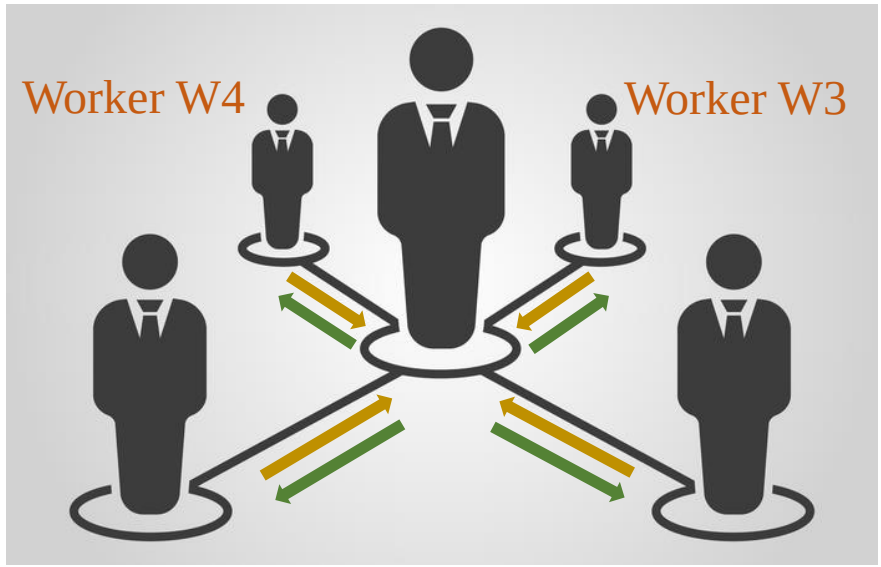


Impractical because of the limitation of resources

Need to use  rather than 

Scenario 3
Given more $

Decision maker (M)



1) Multiple workers
2) Work independently
3) Enough $ only for several rounds
4) Generations are independent

Assign a random location

Feedback its score

Scenario 3
Given more $

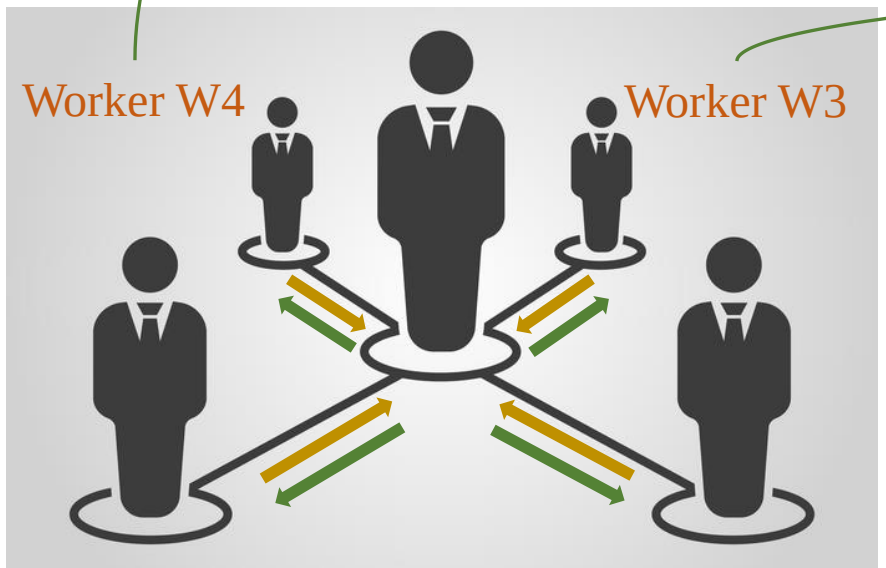
Searching Space

Score1: 5
Score2: 7
Score3: 2
Score4: 8

• • •

Score1: 4
Score2: 9
Score3: 1
Score4: 5

Decision maker (M)



1) Multiple workers
2) Work independently
3) Enough $ only for several rounds
4) Rounds are independent

Assign a random location

Feedback its score

Scenario 3
Given more $

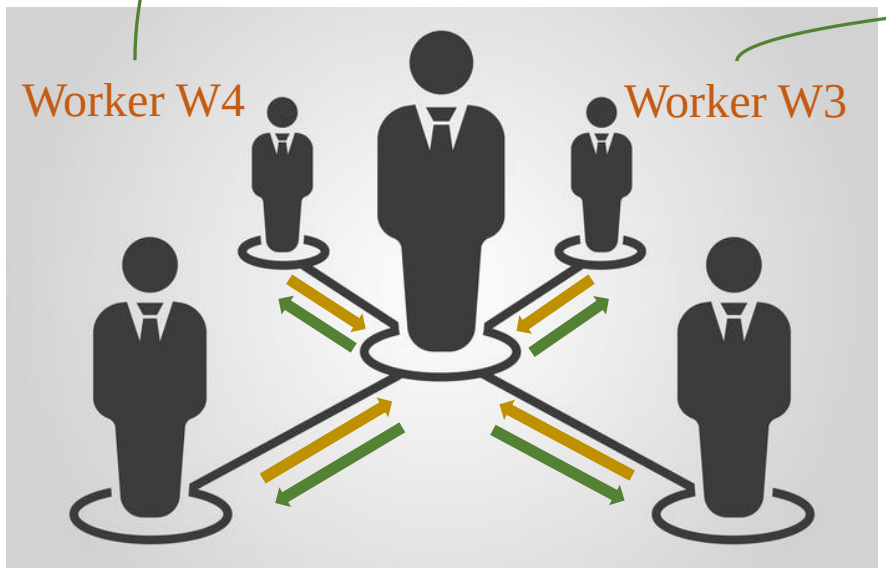
Searching Space

Score1: 5
Score2: 7
Score3: 2
Score4: 8

• • •

Score1: 2
Score2: 7
Score3: 5
Score4: 1

Decision maker (M)



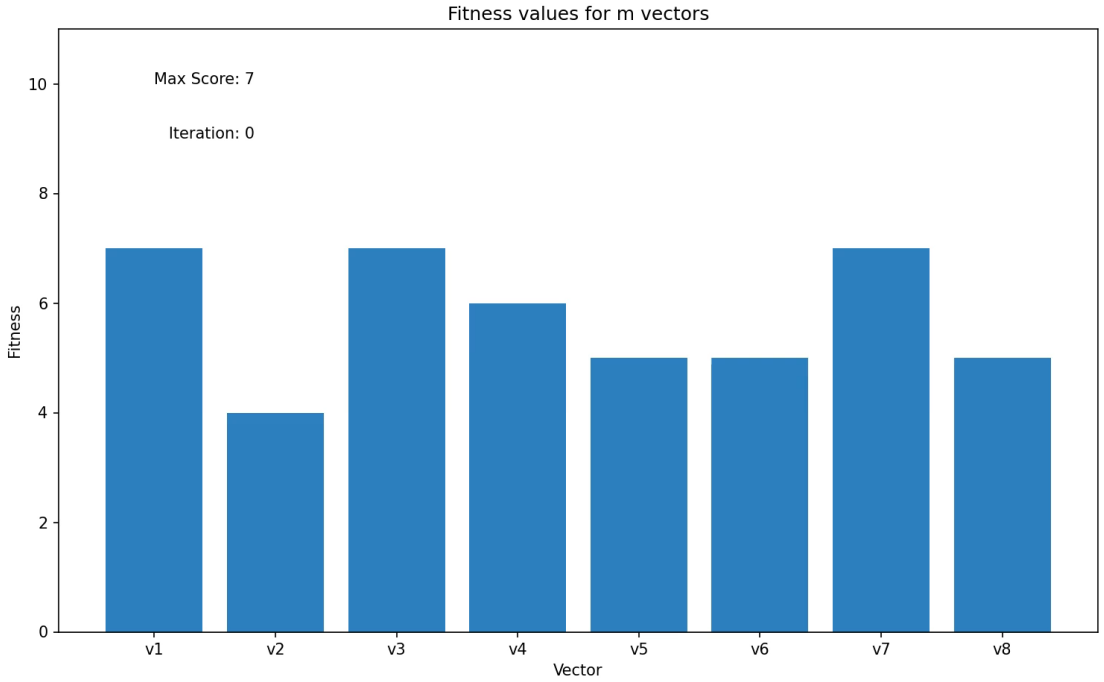
1) Multiple workers
2) Work independently
3) Enough $ only for several rounds
4) Rounds are independent

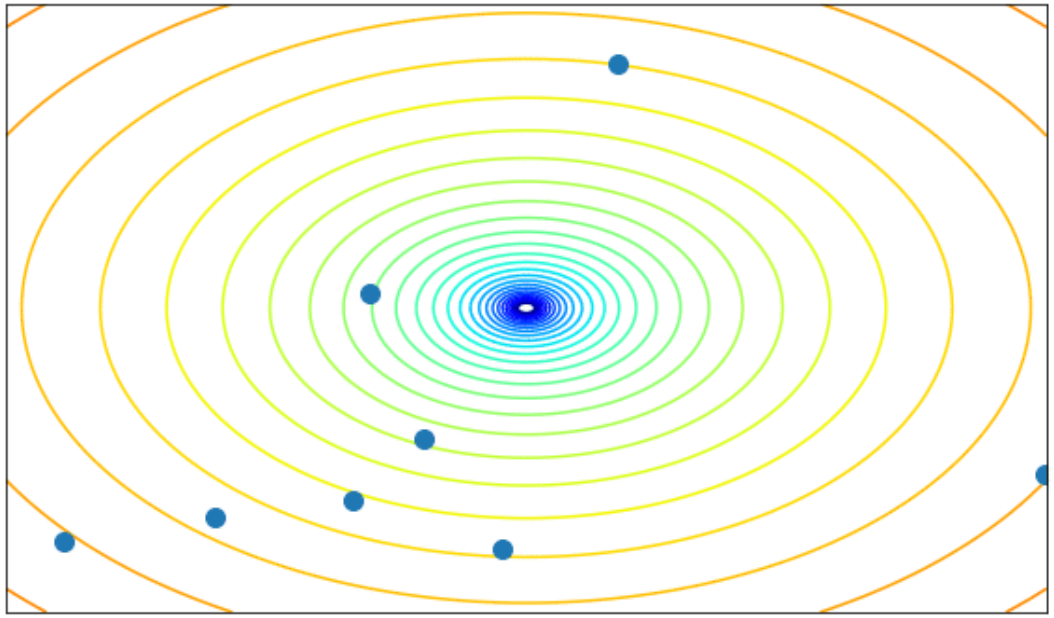
Assign a random location

Feedback its score

Random Search

❖ Generate different groups of m vectors





Scenario 4
Inheritance

max_score = 8

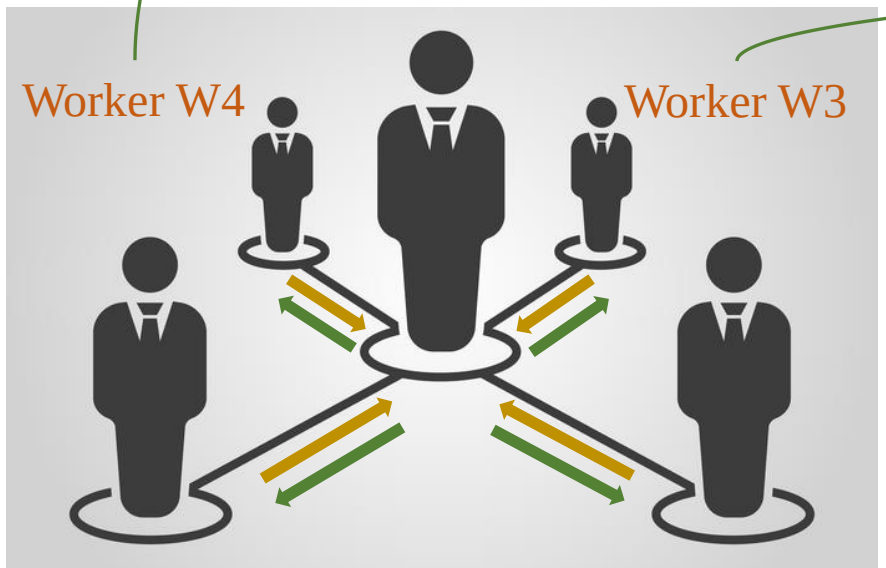
Score1: 5
Score2: 7
Score3: 2
Score4: 8

• • •

Score1: 2
Score2: 7
Score3: 5
Score4: 1

Searching Space

Decision maker (M1)



1) Multiple workers
2) Work independently
3) Enough $ only for several rounds
4) Rounds are somehow dependent

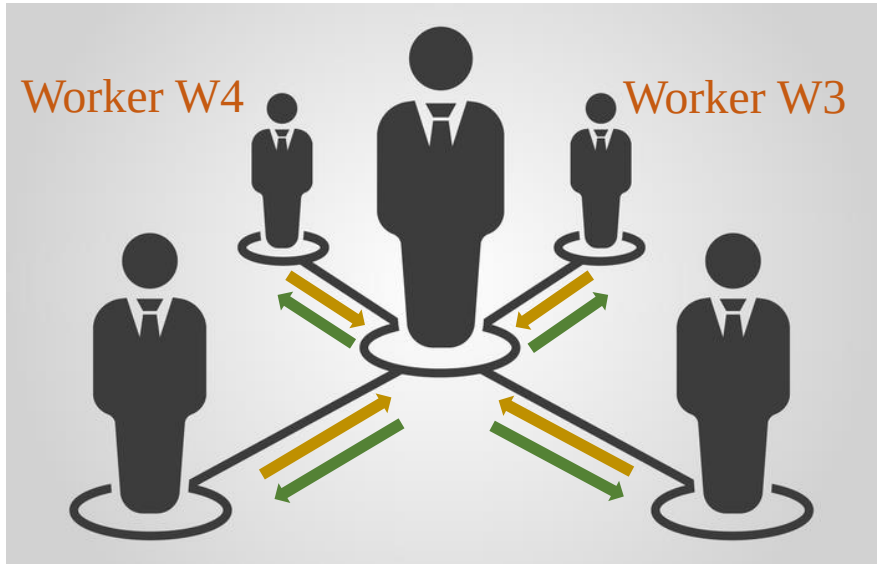
Assign a random location

Feedback its score

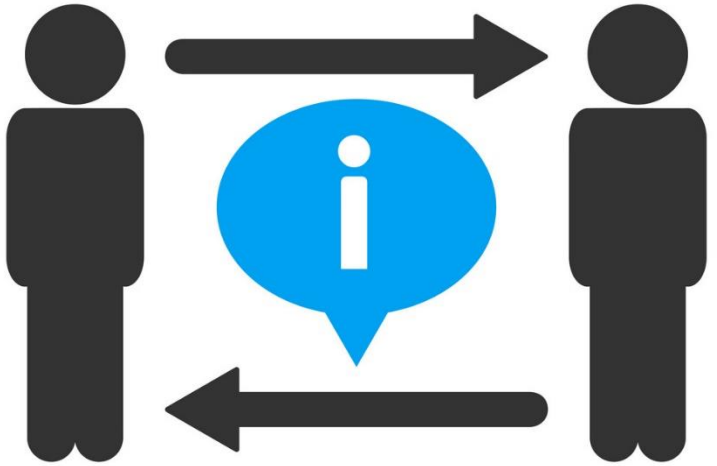
1.1.NaiveRandomSearch.ipynb

Discussion

❖ Problems



1) Multiple workers
2) Work independently
3) Enough $ only for several rounds
4) Rounds are somehow dependent



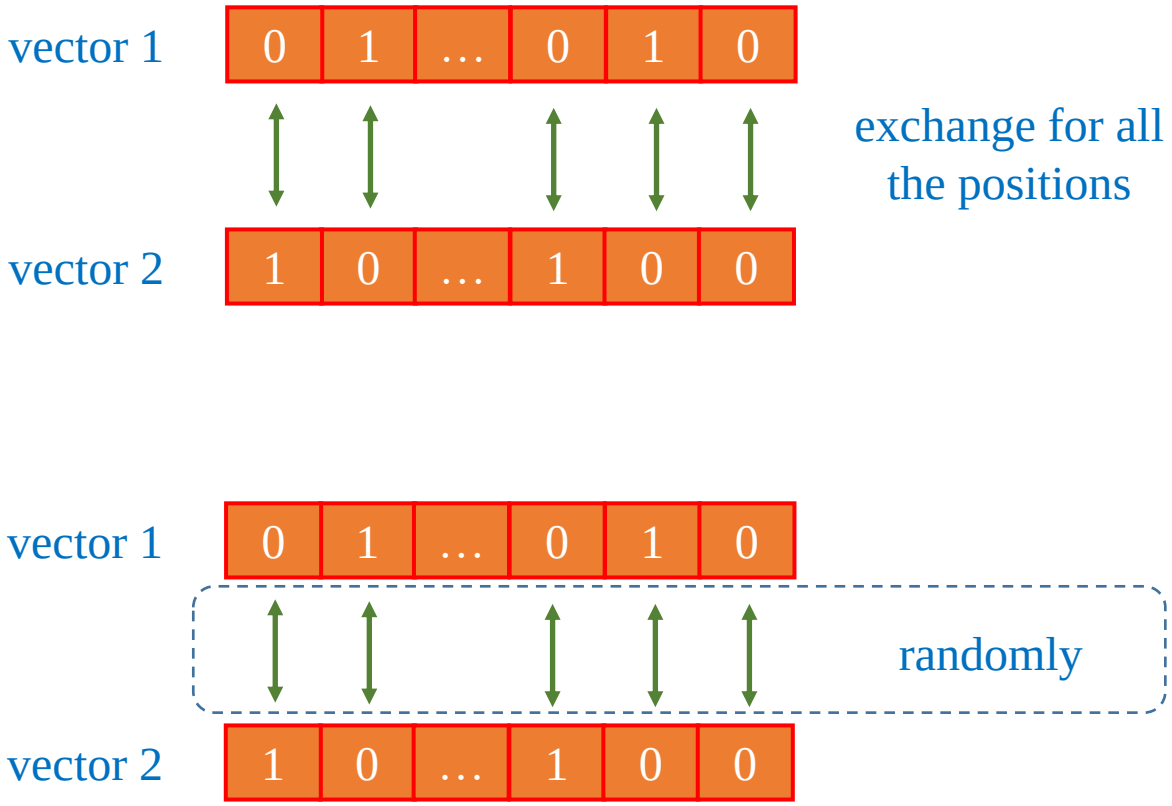
Discussion

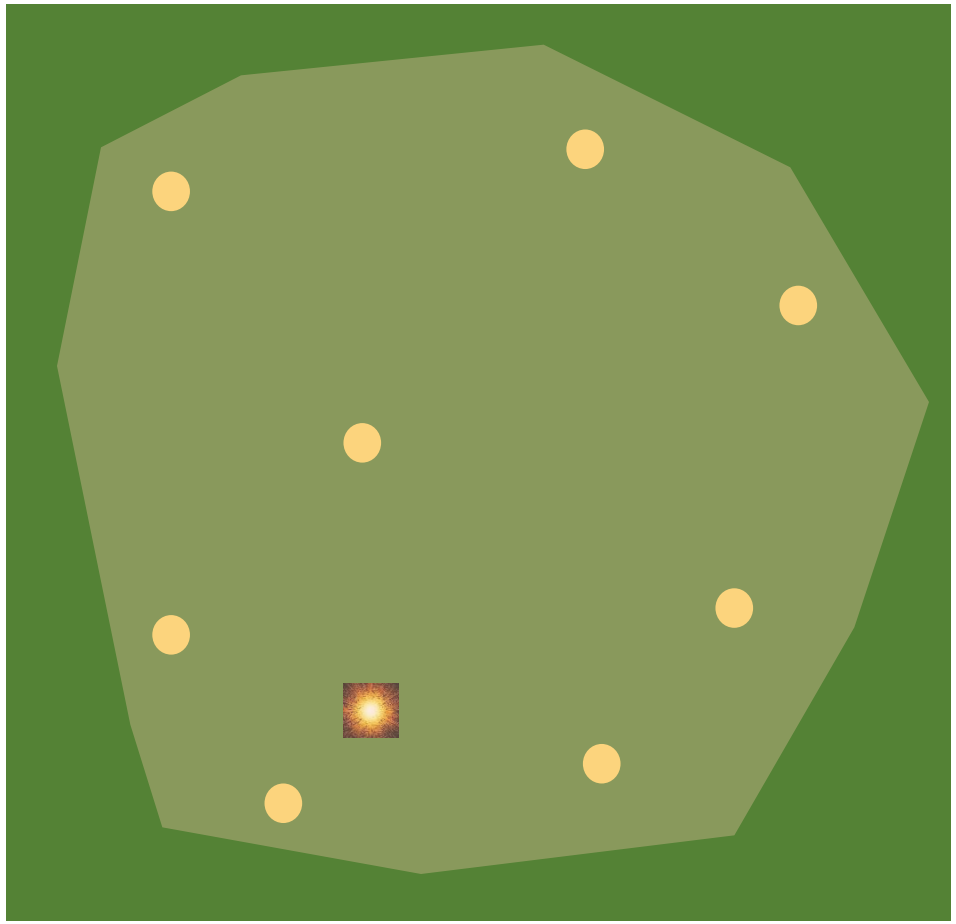
Generated points

❖ Information exchange

Explore or exploit?

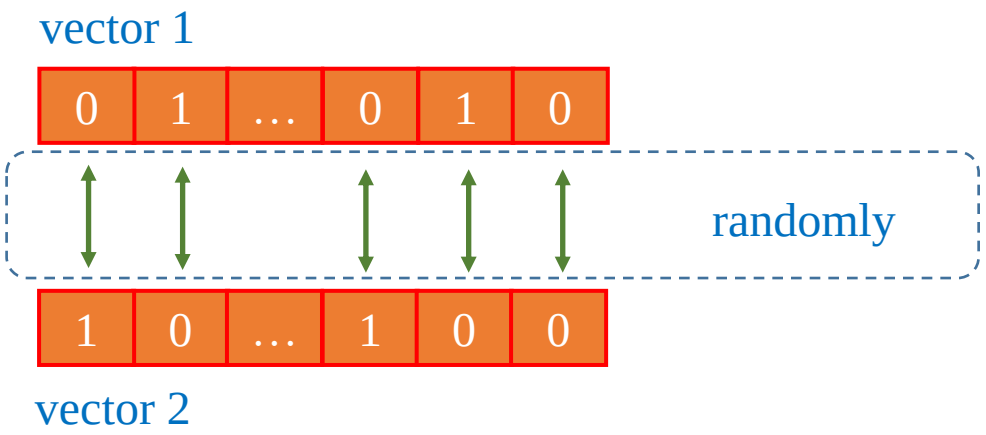
Search locally or globally?



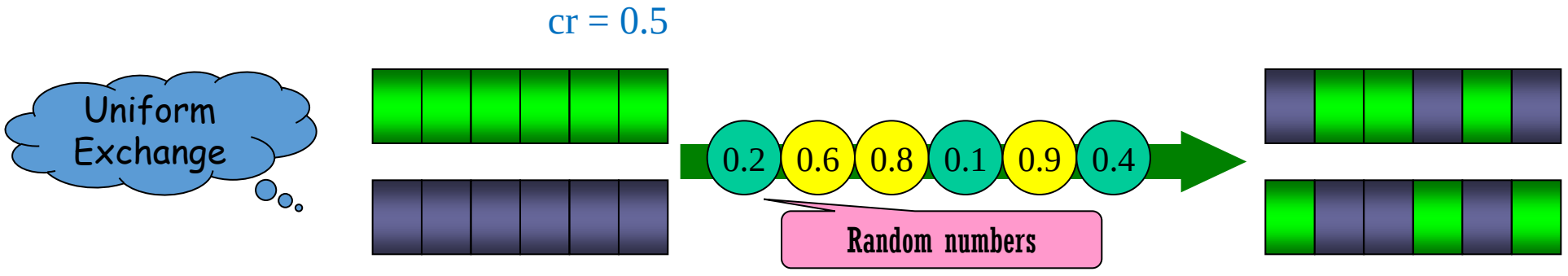


Discussion

❖ Information exchange



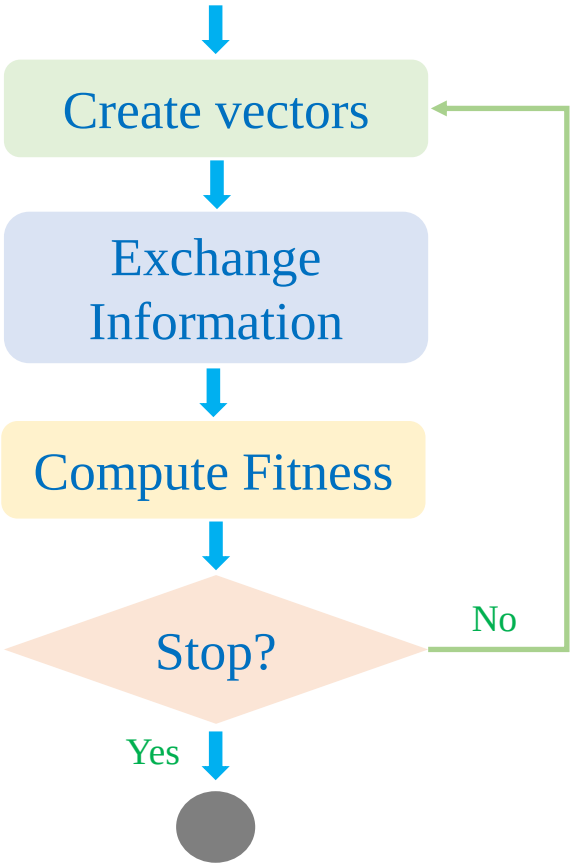

```python
def exchange(vector1, vector2, n, cr=0.9):
    vector1_new = vector1.copy()
    vector2_new = vector2.copy()

    for i in range(n):
        if random.random() < cr:
            vector1_new[i] = vector2[i]
            vector2_new[i] = vector1[i]

    return vector1_new, vector2_new
```



Random Search + Information Exchange

#vectors: m = 40

Vector length: n = 20

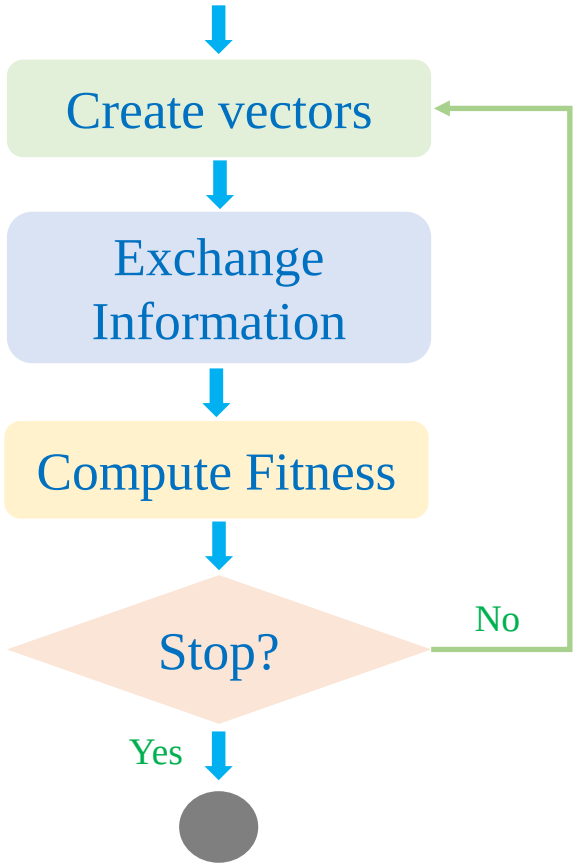



```python
import random

def generate_01():
    return random.randint(0, 1)

def compute_fitness(vector):
    return sum(gen for gen in vector)

def create_vector():
    return [generate_01() for _ in range(n)]

def exchange(vector1, vector2, n, cr=0.9):
    vector1_new = vector1.copy()
    vector2_new = vector2.copy()

    for i in range(n):
        if random.random() < cr:
            vector1_new[i] = vector2[i]
            vector2_new[i] = vector1[i]

    return vector1_new, vector2_new
```

Random Search + Information Exchange

#vectors: m = 40

Vector length: n = 20

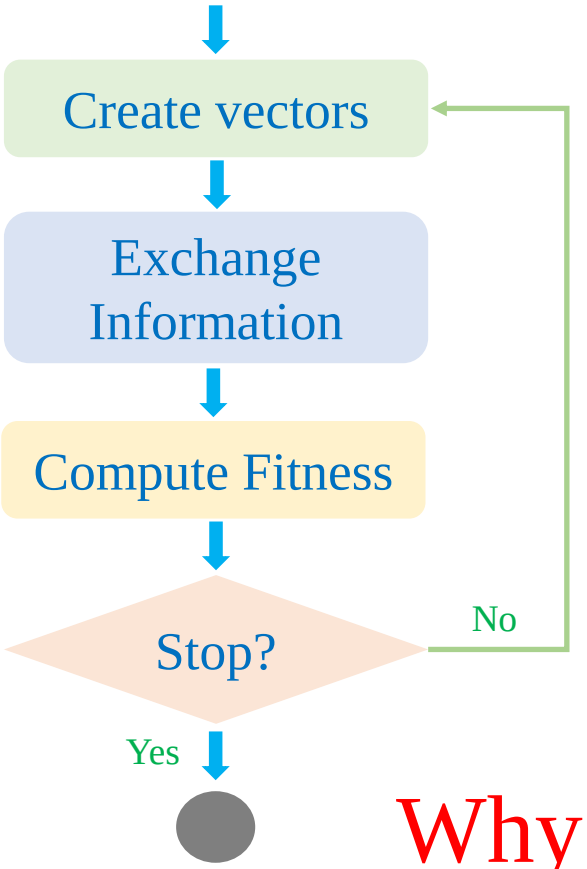


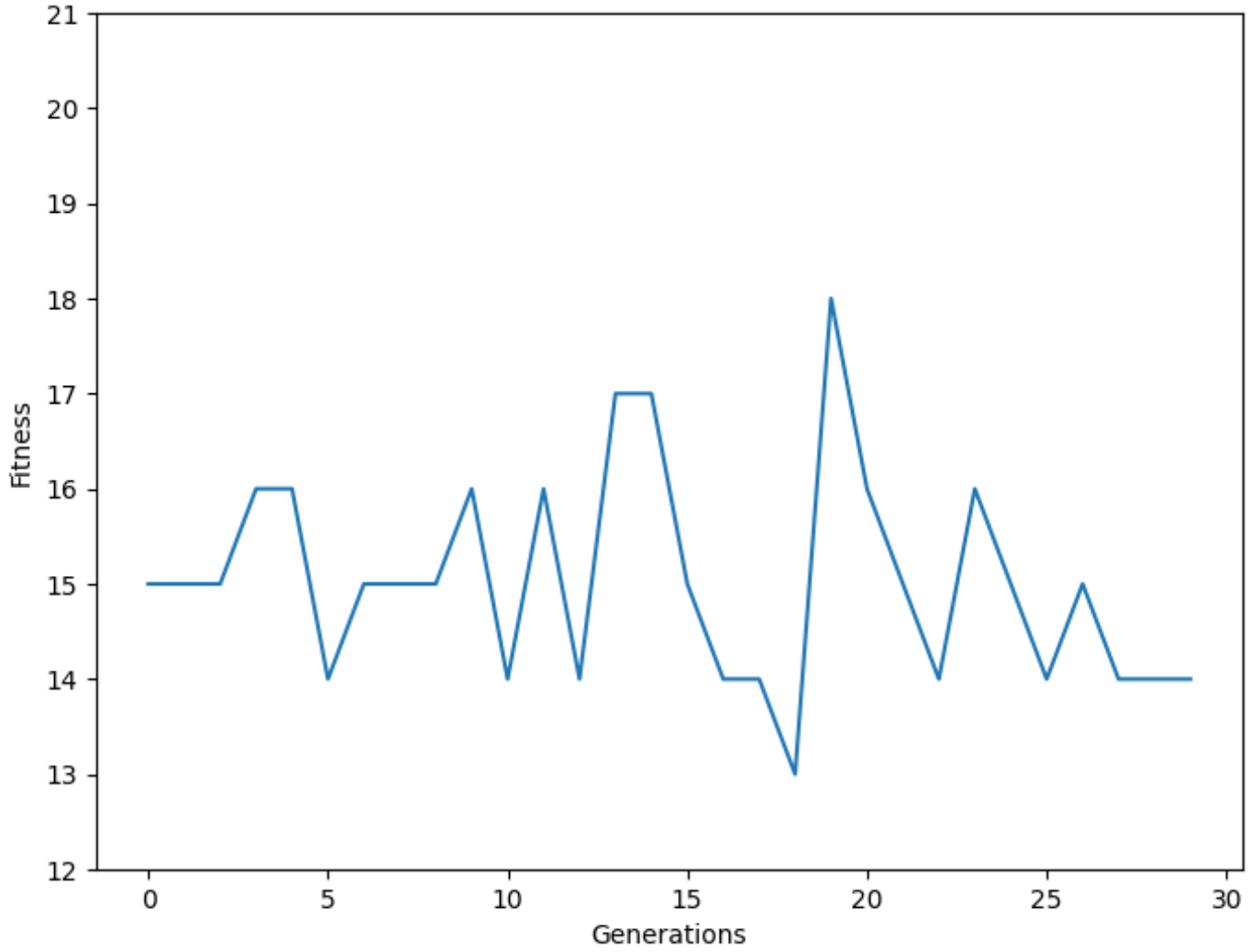

```python
for i in range(n_generations):
    # 1. create population
    population = [create_vector() for _ in range(m)]

    # 2. compute fitness ...

    # exchange
    new_population = []
    while len(new_population) < m:
        # get two vectors
        index1 = random.randint(0, m-1)
        while True:
            index2 = random.randint(0, m-1)
            if (index2 != index1):
                break

        vector1 = population[index1]
        vector2 = population[index2]

        # exchange
        vector1, vector2 = exchange(vector1, vector2, n)

        # save these two vectors
        new_population.append(vector1)
        new_population.append(vector2)

    # update
    population = new_population
```

Random Search + Information Exchange

#vectors: m = 40

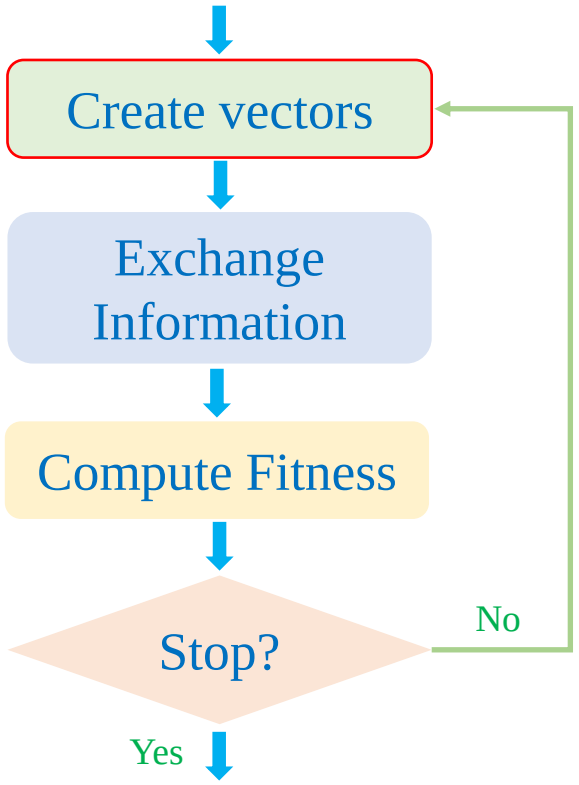
Vector length: n = 20

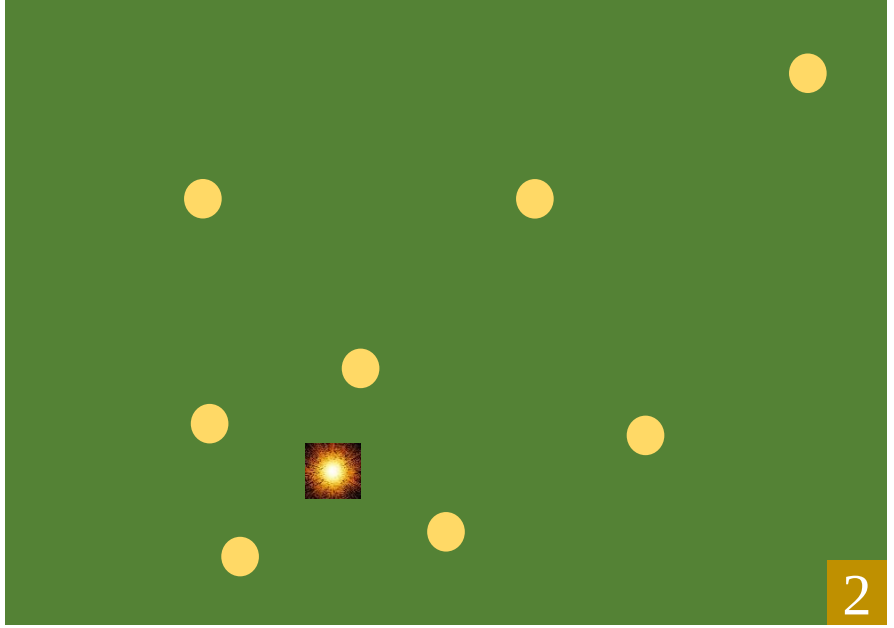


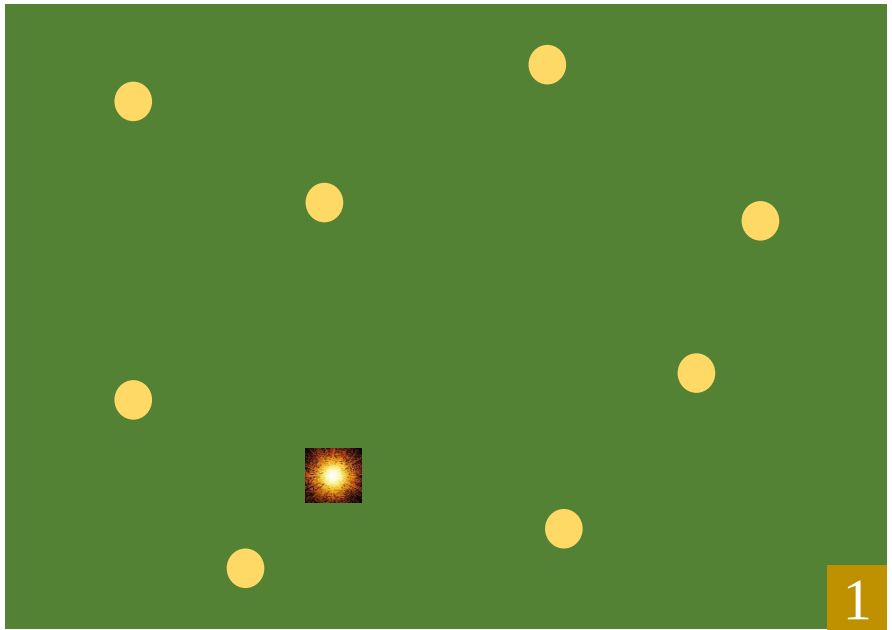


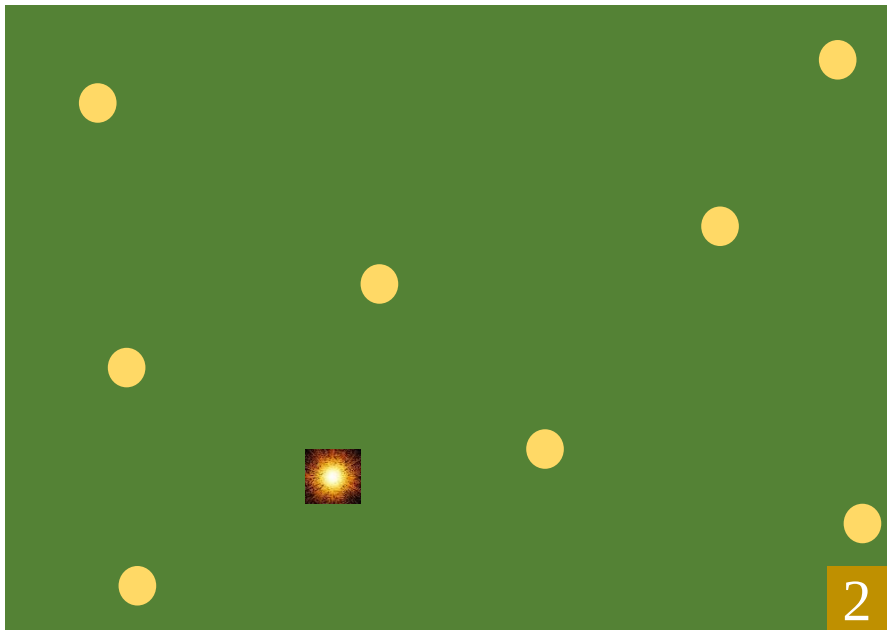
Why??

Possibility



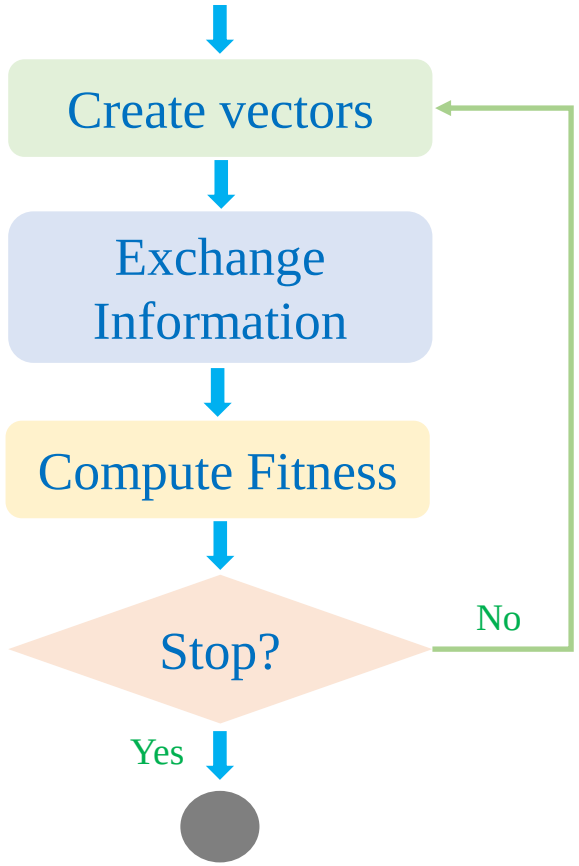




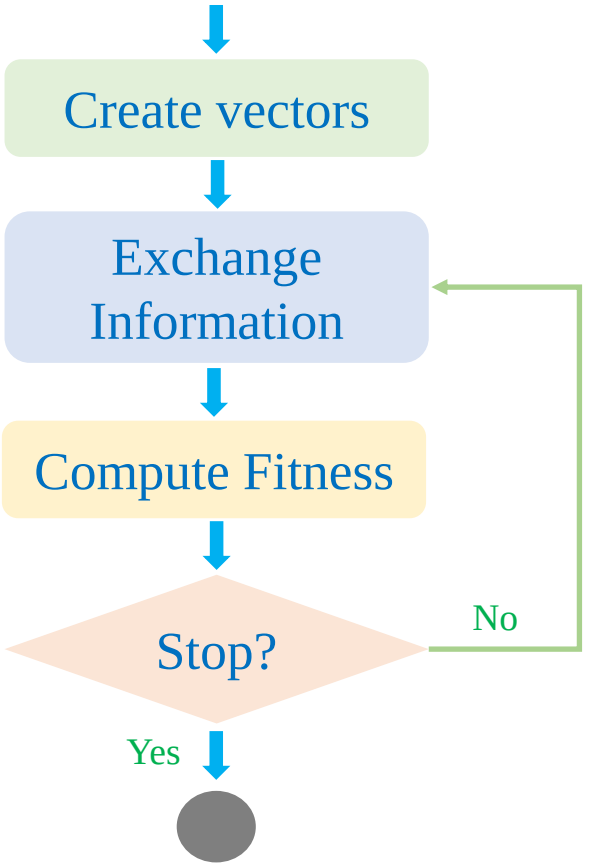


Discussion

❖ Reuse what has done



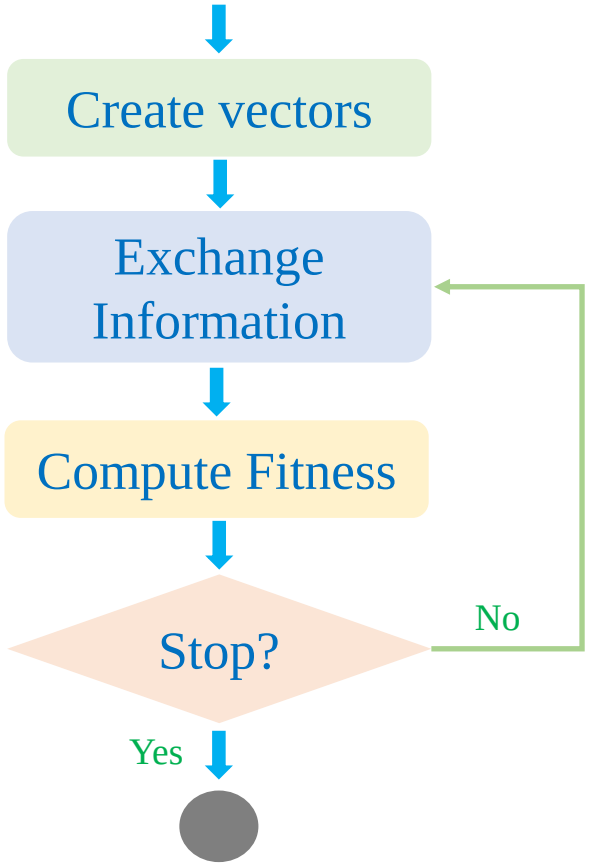
No inheritance



Inheritance

Information Exchange + Inheritance

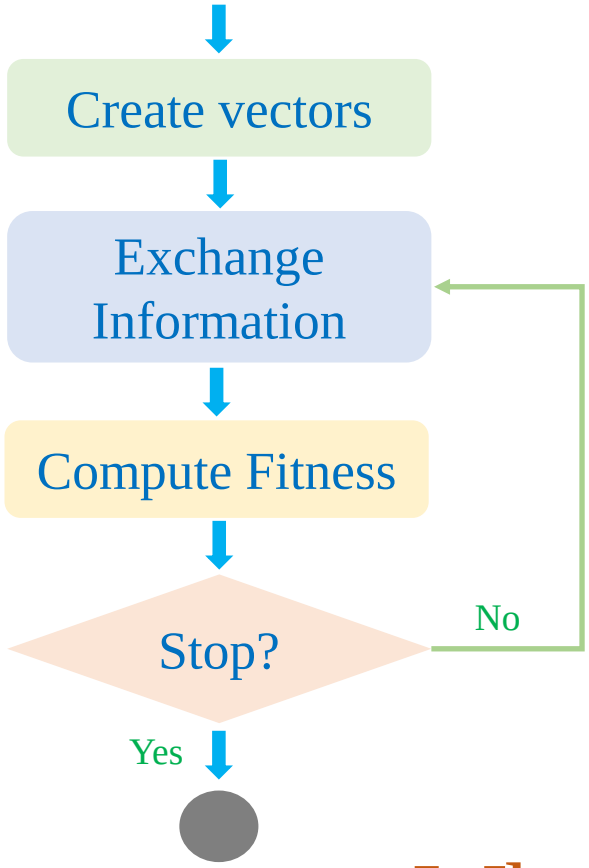
❖ Implementation

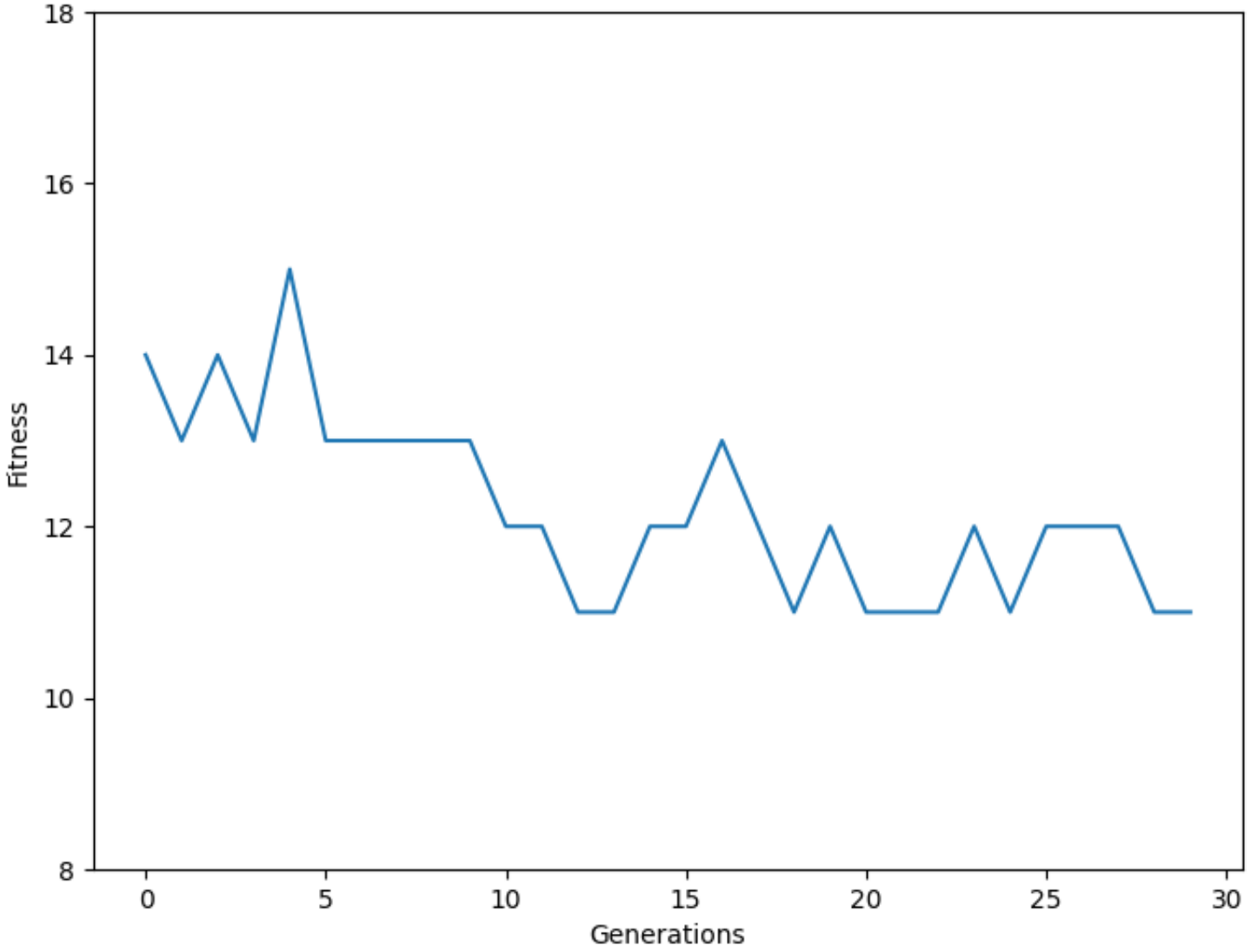



```python
# 1. create population
population = [create_vector() for _ in range(m)]

for i in range(n_generations):
    # 2. compute fitness ...

    # exchange
    new_population = []
    while len(new_population) < m:
        # get two vectors
        index1 = random.randint(0, m-1)
        while True:
            index2 = random.randint(0, m-1)
            if (index2 != index1):
                break

        vector1 = population[index1]
        vector2 = population[index2]

        # exchange
        vector1, vector2 = exchange(vector1, vector2, n)

        # save these two vectors
        new_population.append(vector1)
        new_population.append(vector2)

    # update
    population = new_population
```

Information Exchange + Inheritance

❖ **Result**





Why???

Revisit information exchange

Searching space

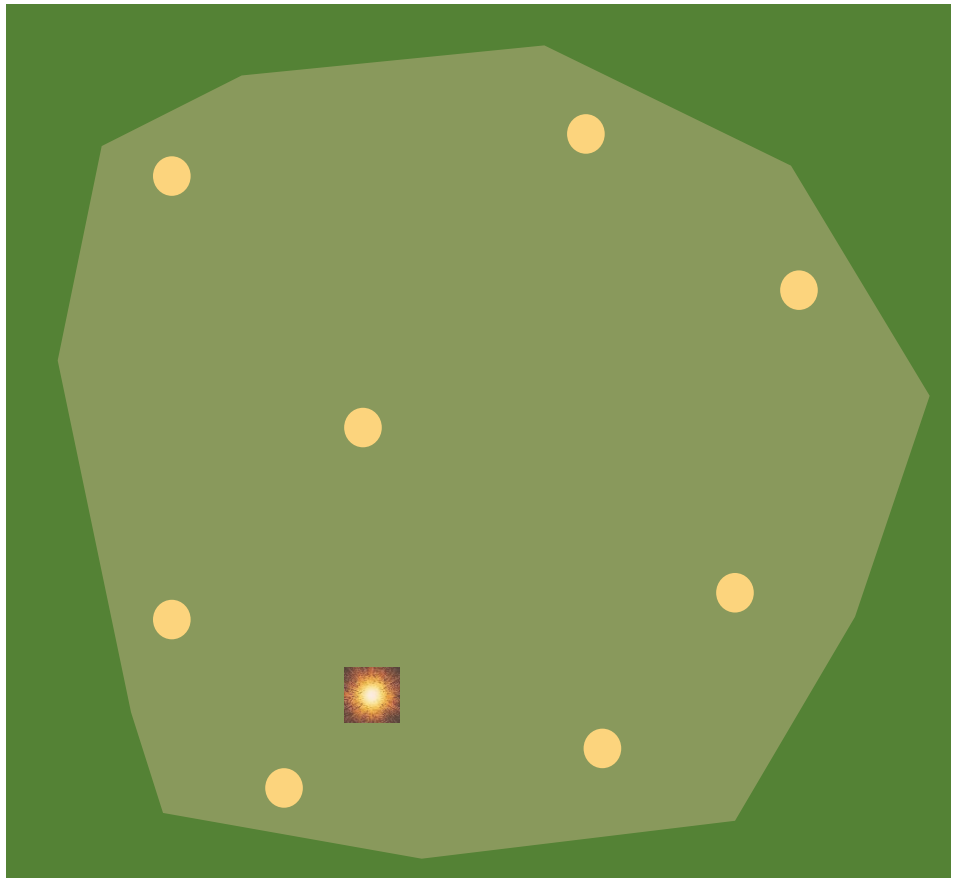
Bad information

Generated points

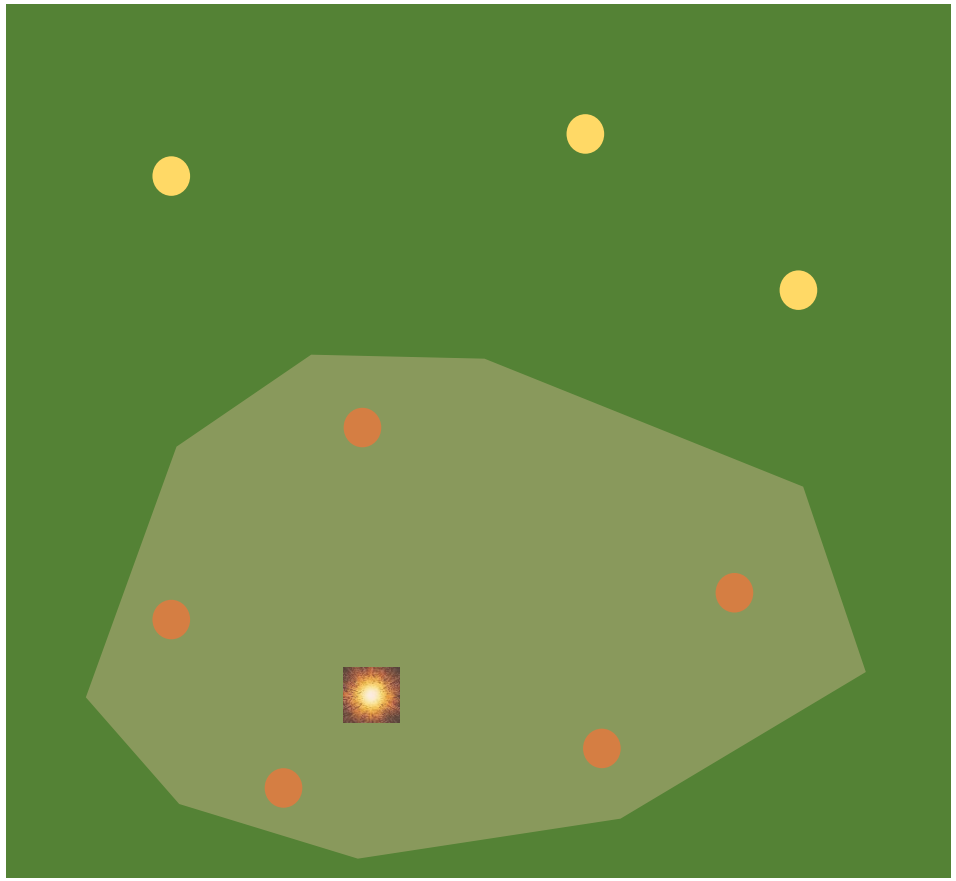
vector 1 | 0 | 1 | … | 0 | 1 | 0 |

randomly

vector 2 | 1 | 0 | … | 1 | 0 | 0 |

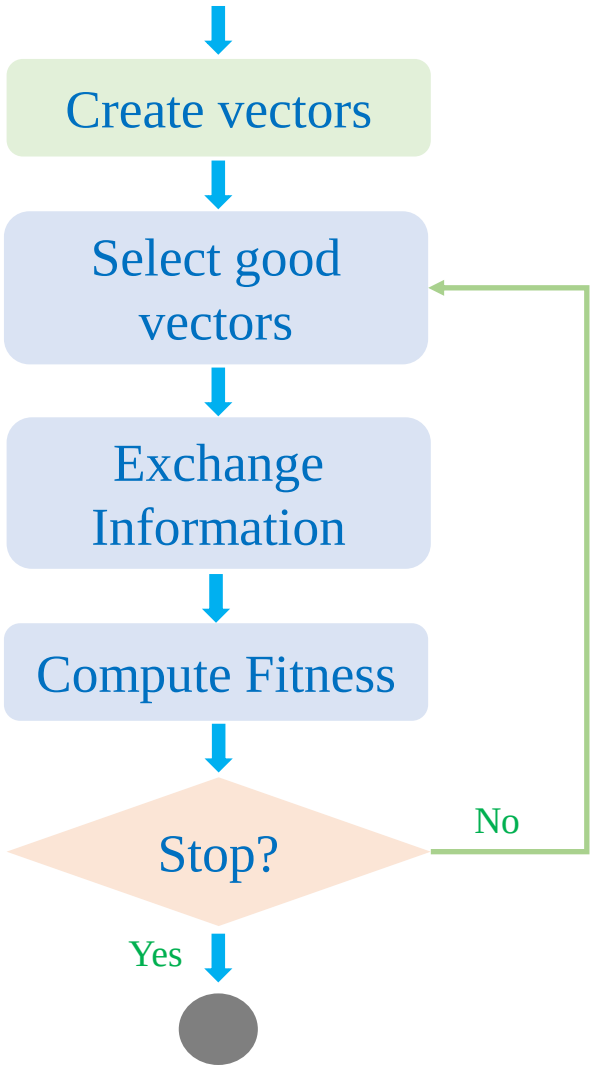


Only best vectors are used



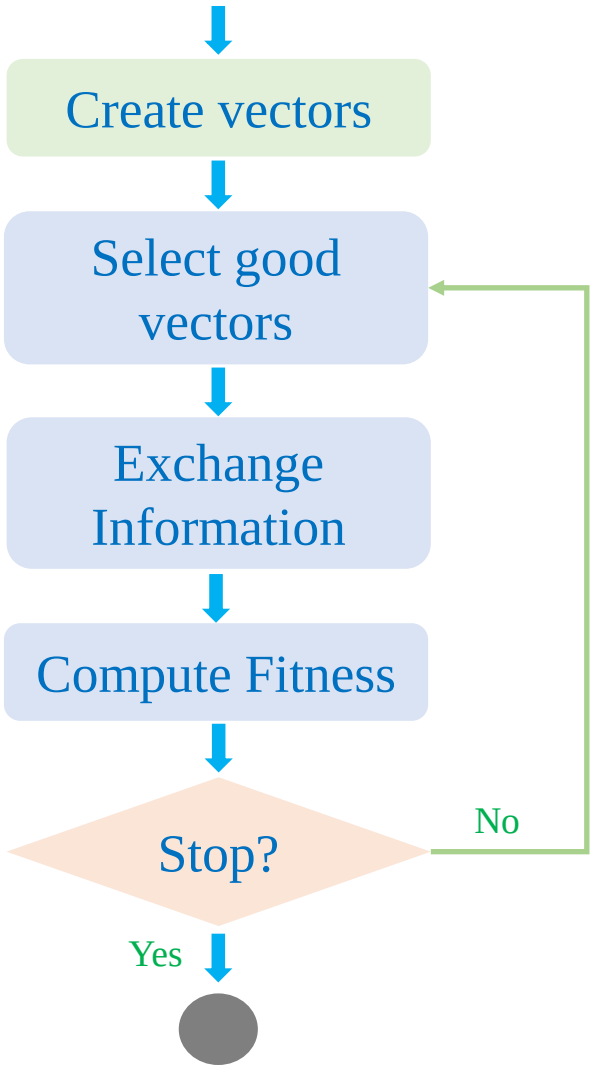
Exchange + Inheritance + Selection

❖ Using good vectors




```python
def select_better_vector(sorted_vectors):
    index1 = random.randint(0, m-1)
    while True:
        index2 = random.randint(0, m-1)
        if (index2 != index1):
            break

    vector = sorted_vectors[index1]
    if index2 > index1:
        vector = sorted_vectors[index2]

    return vector
```

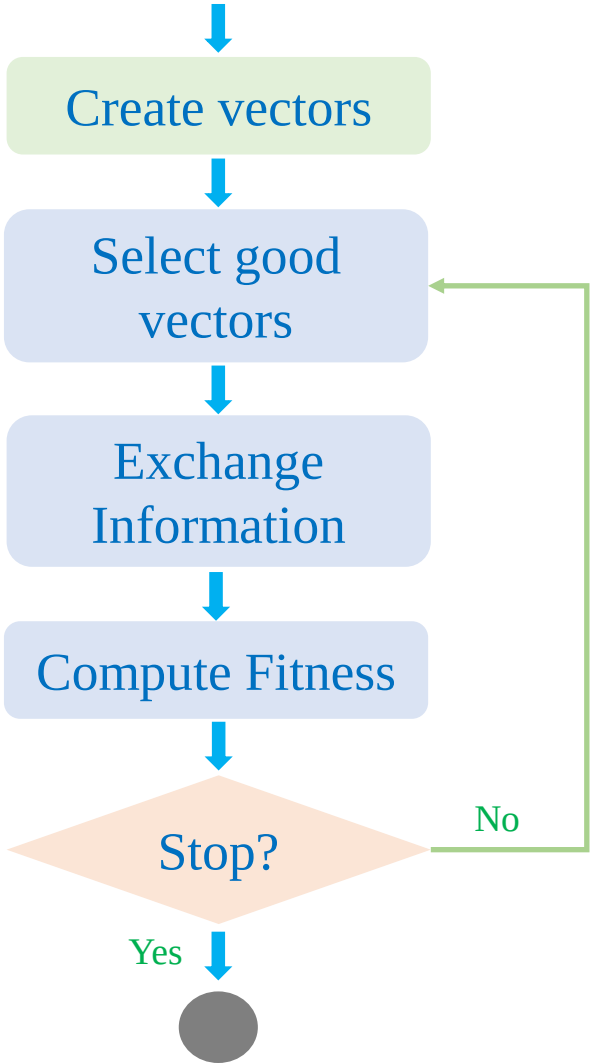
Exchange +Inheritance +Selection

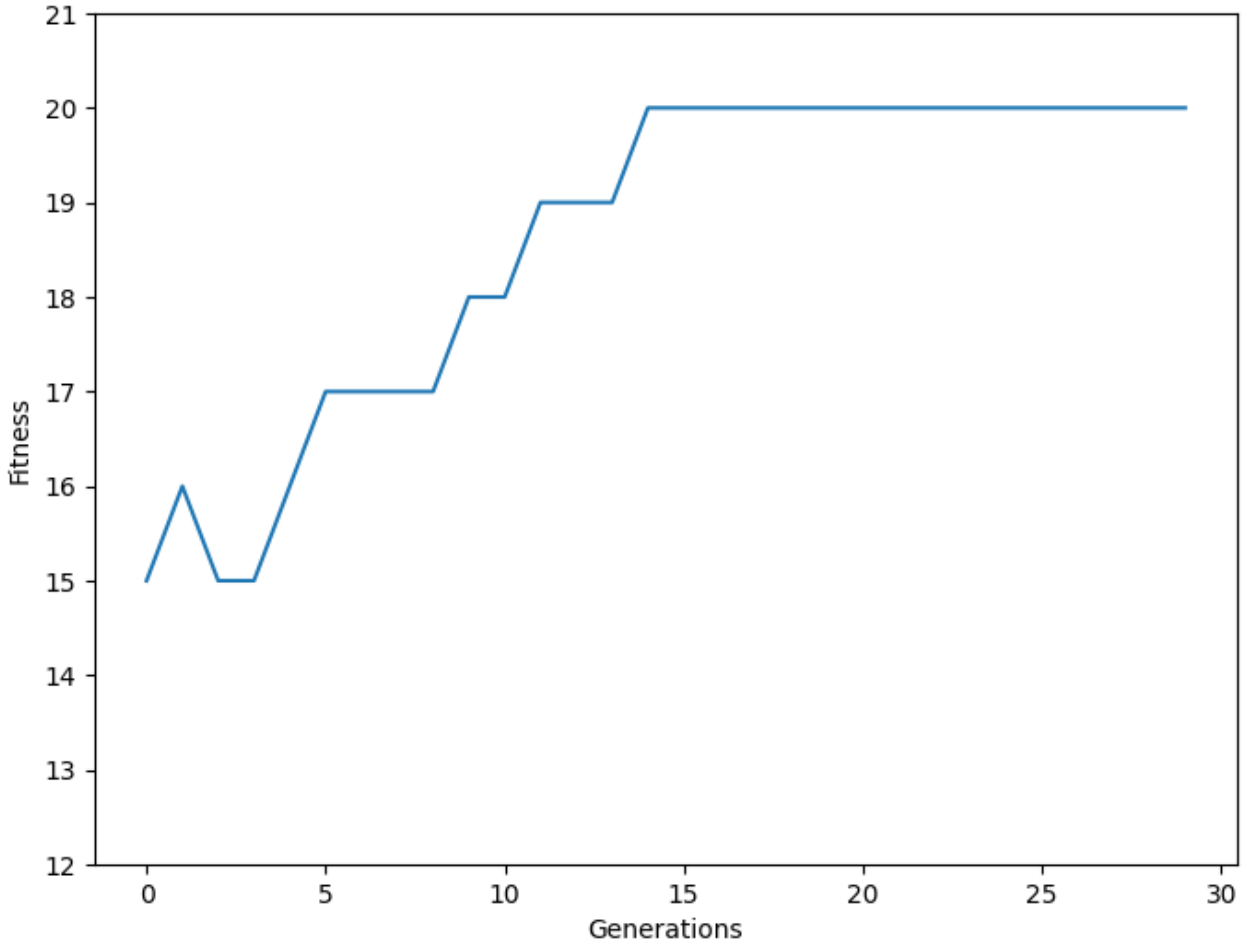



```python
# create population
population = [create_vector() for _ in range(m)]

# loops
for i in range(n_generations):
    # compute fitness (optional)
    sorted_vectors = sorted(population, key=compute_fitness)
    fitnesses.append(compute_fitness(sorted_vectors[m-1]))
    print("BEST:", compute_fitness(sorted_vectors[m-1]))

    new_population = []
    while len(new_population) < m:
        # selection
        vector1 = select_better_vector(sorted_vectors)
        vector2 = select_better_vector(sorted_vectors) # dup.

        # exchange
        vector1, vector2 = exchange(vector1, vector2, n)

        # save
        new_population.append(vector1)
        new_population.append(vector2)

    # update
    population = new_population
```

Exchange + Inheritance + Selection

❖ Using good vectors

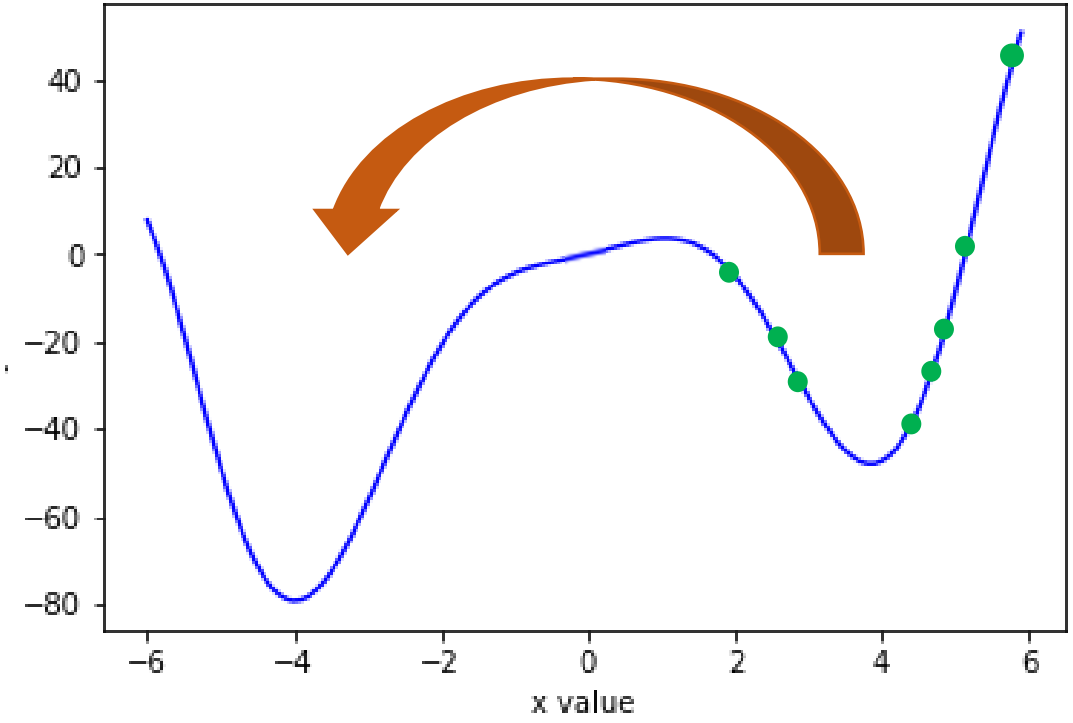


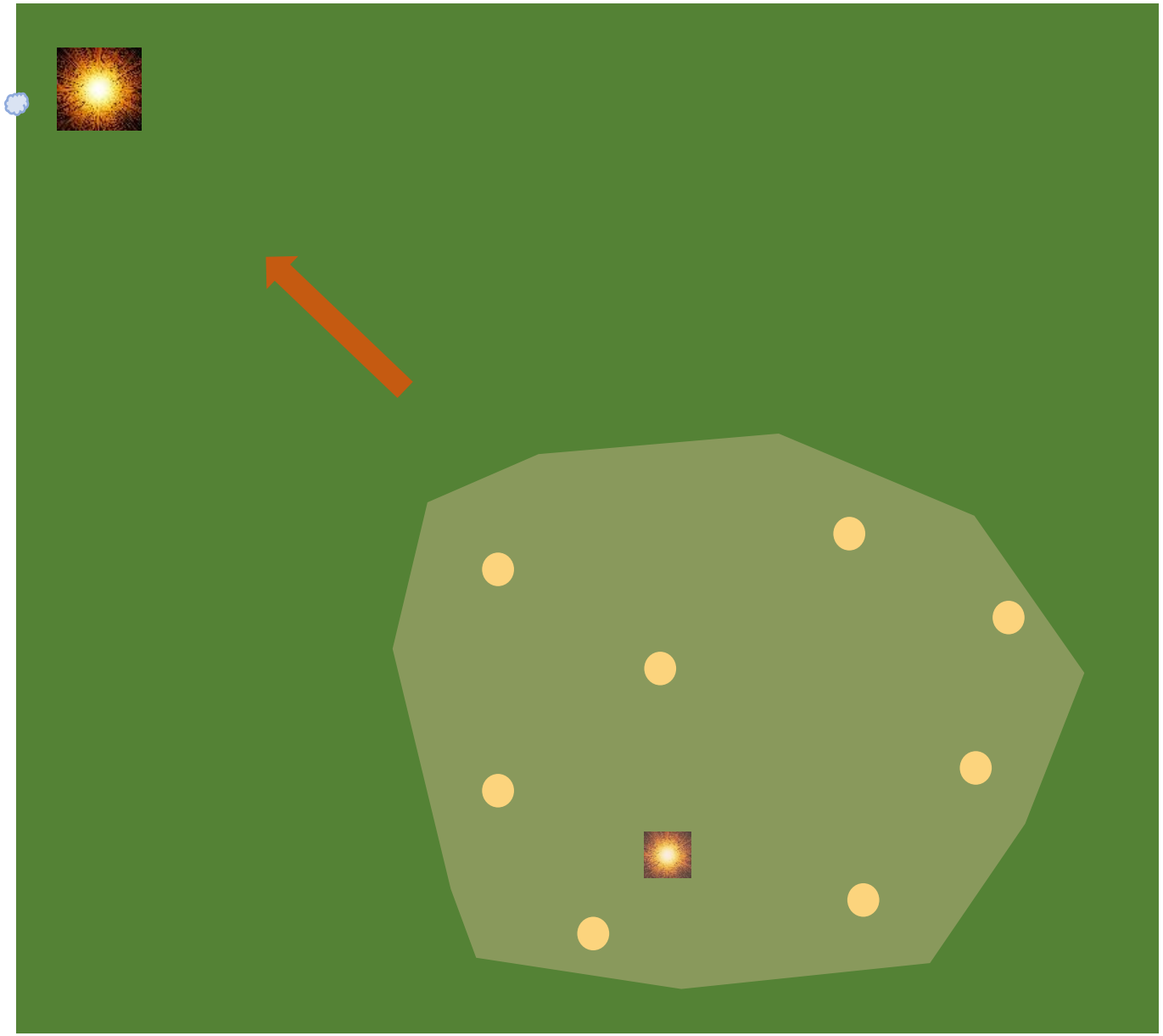


New Challenge

Better solution

How to get out of this local optimal?

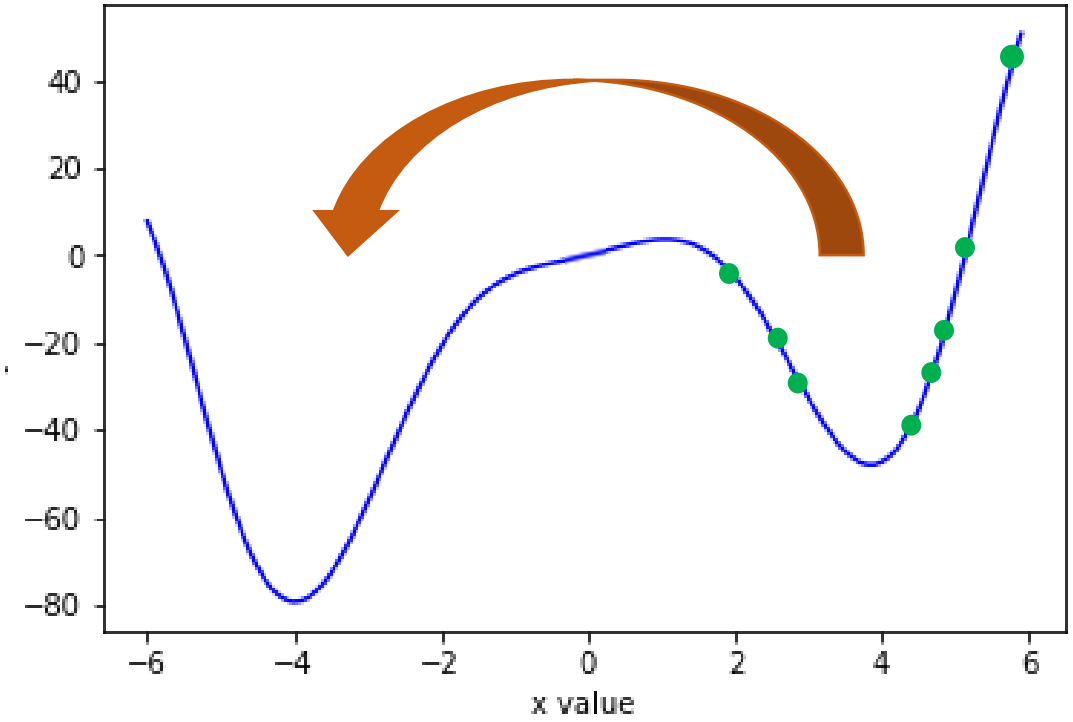


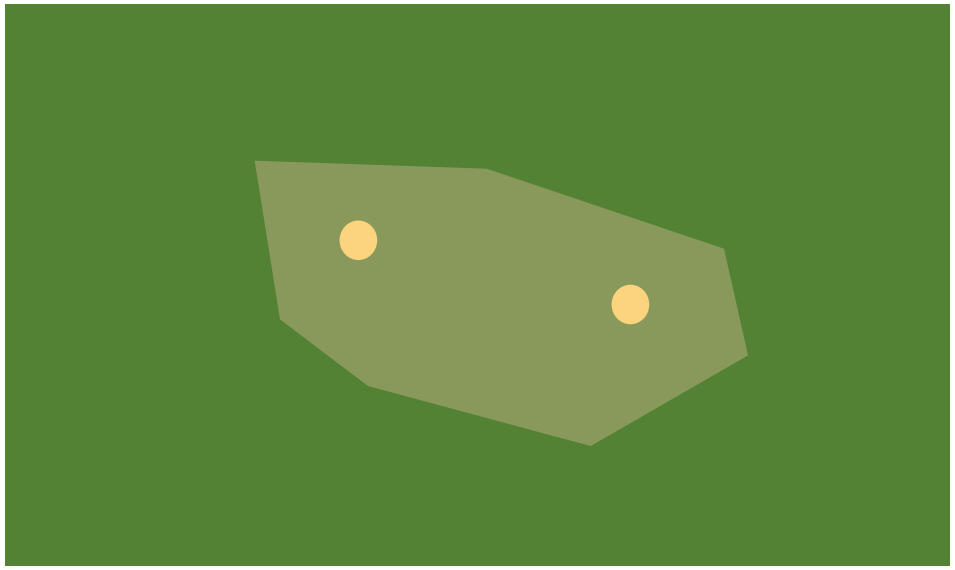


New Challenge

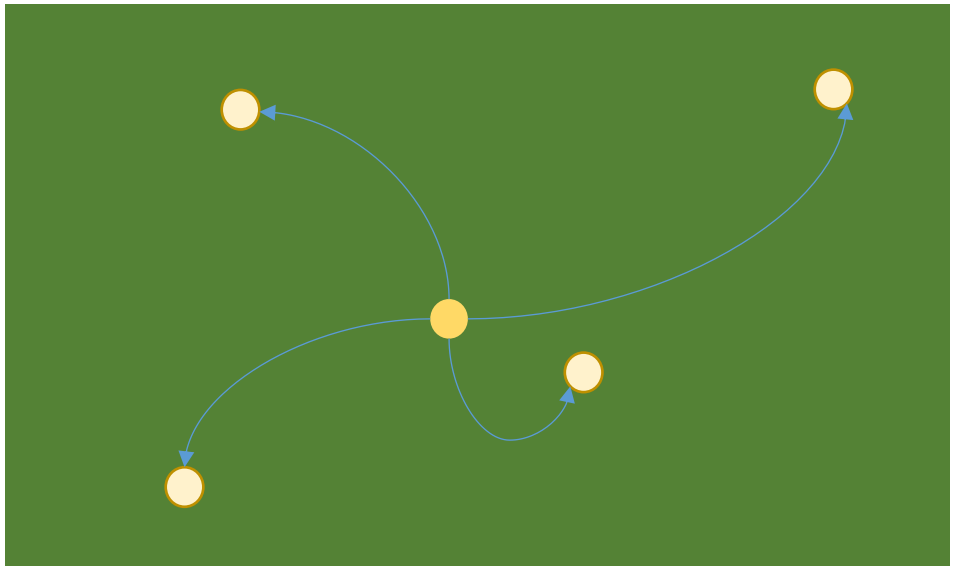
Generated points

How to get out of this local optimal?





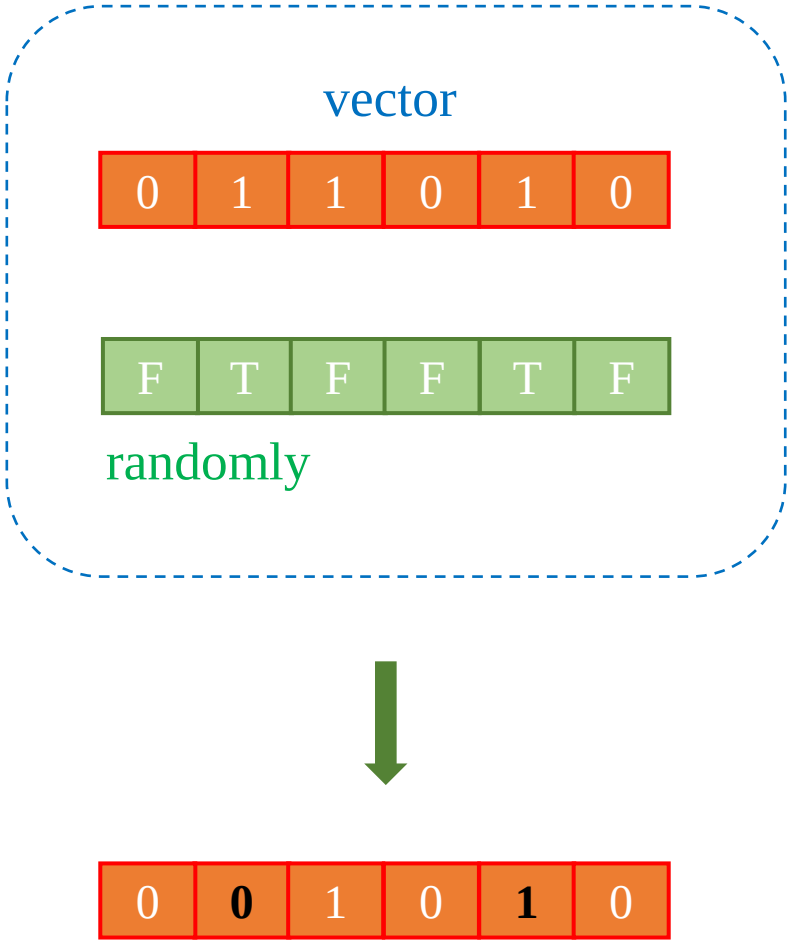
Exploitation (Information Exchange)

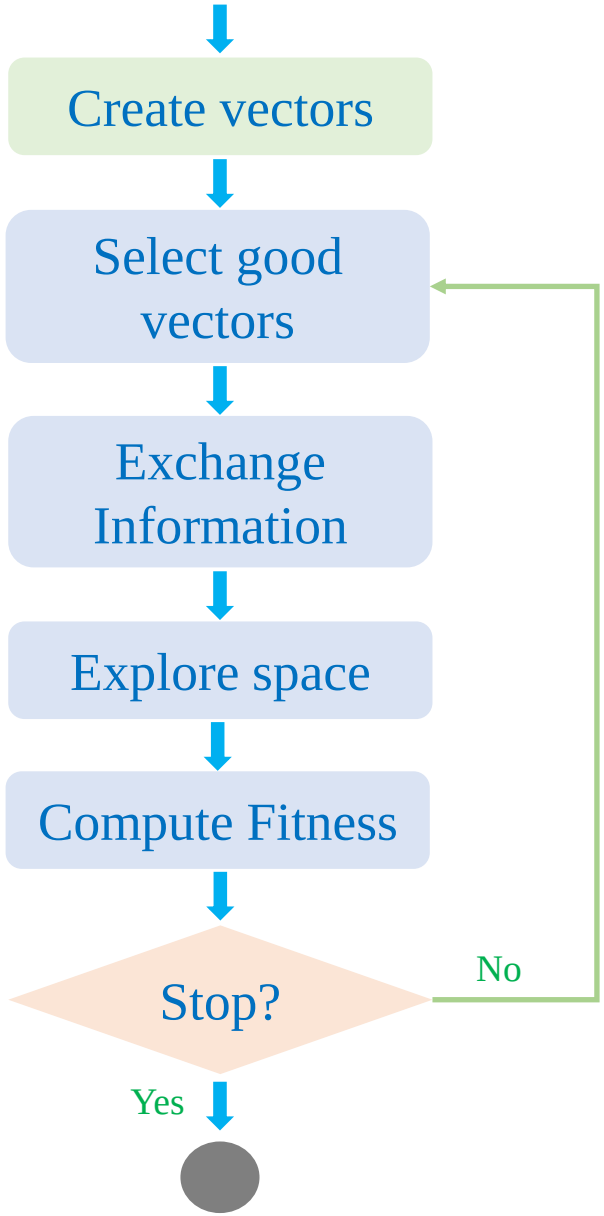


Exploration (Receiving a random value)

… + Exploration

❖ **Change a value of a vector randomly**



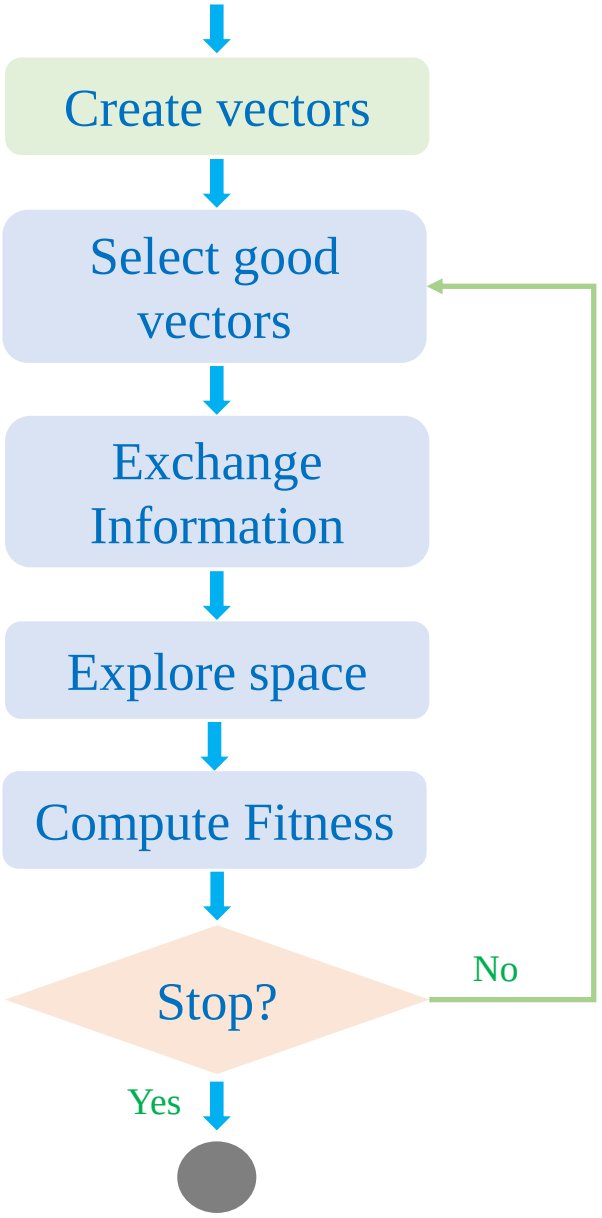


… + Exploration

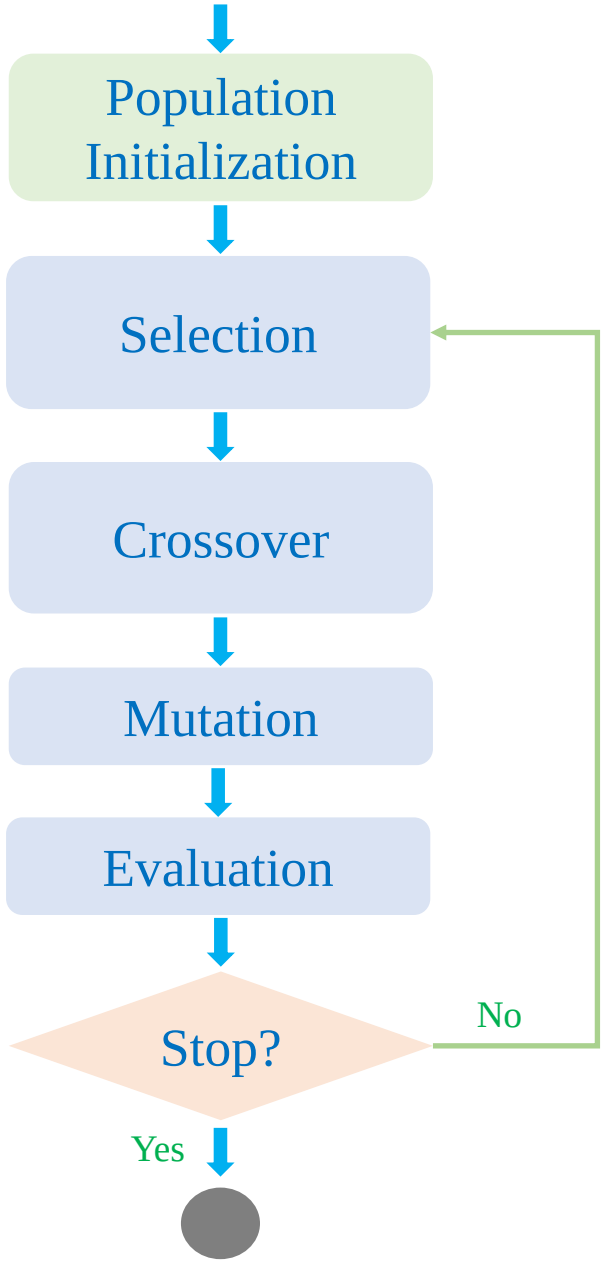
```python
def exchange(vector1, vector2, rate=0.9):
    vector1_new = vector1.copy()
    vector2_new = vector2.copy()

    for i in range(n):
        if random.random() < rate:
            vector1_new[i] = vector2[i]
            vector2_new[i] = vector1[i]

    return vector1_new, vector2_new

def explore(vector, rate=0.05):
    vector_m = vector.copy()

    for i in range(n):
        if random.random() < rate:
            vector_m[i] = generate_01()

    return vector_m
```

```python
# create population
population = [create_vector() for _ in range(m)]

# loops
for i in range(n_generations):
    #  sort population
    sorted_vectors = sorted(population, key=compute_fitness)

    new_population = []
    while len(new_population) < m:
        # selection
        vector_s1 = selection(sorted_vectors)
        vector_s2 = selection(sorted_vectors) # duplication

        # exchange
        vector_c1, vector_c2 = exchange(vector_s1, vector_s2)

        # explore
        vector_m1 = explore(vector_c1)
        vector_m2 = explore(vector_c2)

        # save
        new_population.append(vector_m1)
        new_population.append(vector_m2)

    # update
    population = new_population
```

Summary



Our algorithms



Genetic Algorithm