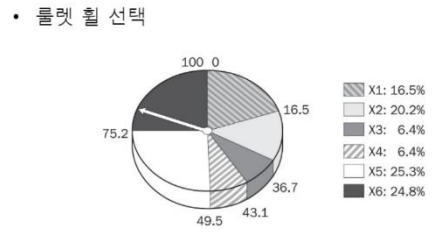
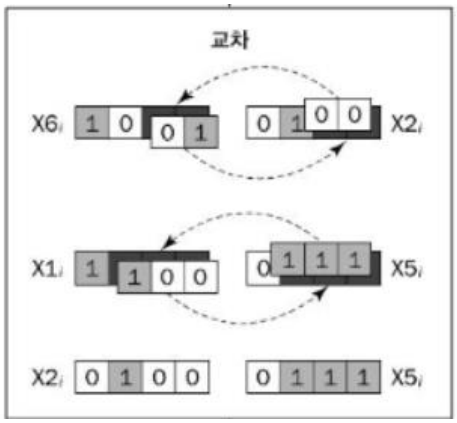
유전자 알고리즘

유전자 알고리즘이란 자연선택 또는 적자생존의 원칙에 입각한 알고리즘으로 개체군 중에서 환경에 대한 적합도(fitness)가 높은 개체의 경우 재생산을 통해, 환경에 적응하는 개체군을 만들어내는 알고리즘이다.

유전자 알고리즘에는 자연선택, 교차, 돌연변이를 통해 적합도가 높은 개체군을 생산해 나간다.

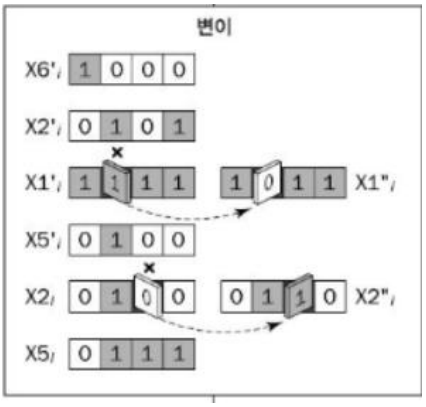


자연선택이란 한 세대에서 다음 세대로 전해지는 해의 후보가 되는 해들을 선택하는 과정으로 대표적으로는 룰렛 휠 선택이 있다. 이는 어떤 방법을 쓰느냐에 따라 최적해로 다가가는 속도가 더디게 되거나, 아니면 지역 최적해에 빠지는 경우가 생길 수도 있기 때문에 해의 선택은 유전 알고리즘의 성능에 큰 영향을 미친다.



교차란 생명체가 세대 내에서의 교배를 통해 다음 세대를 생성하는 것처럼 유전 알고리즘에서도 자연 선택된 해들은 교배를 통하여 다음 세대의 해들을 생성하게 된다. 일반적으로 두 개의 해를 선택한 후 둘 간에 교배 연산을 수행하게 되며, 이를 통해 생성된 해는 각각의 부모 해의 교차 연산을 통해서 서로 겹치지 않는 위치의 유전인자를 받아 새로운 유전자를 구성하게 된다.

이때 모든 해에 대해 교차연산을 수행할 경우 우수한 해가 다른 해와의 교배를 통해서 우수성을 잃어버림에 따라 세대의 최고 우수해가 보존되지 않는 상황이 발생할 수 있기 때문에 이런 경우를 방지하기 위하여 적합도를 통해 교차를 확률적으로 수행하여 우수한 해가 변형되지 않고 그대로 다음 세대에 전해질 수 있도록 해야한다.

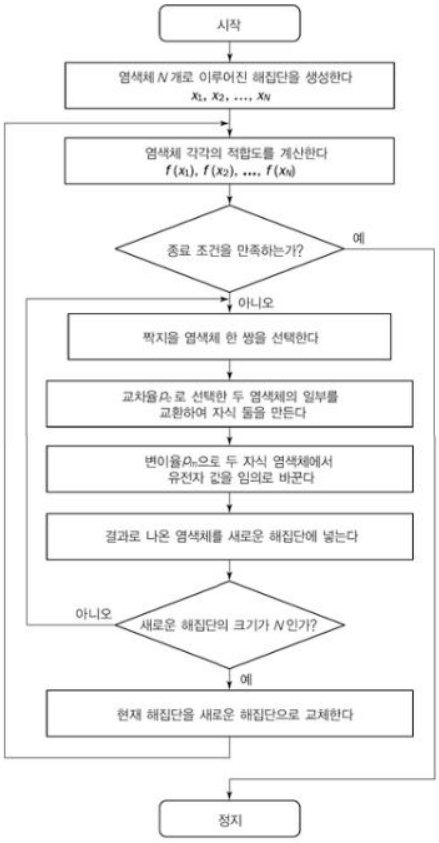


돌연변이란 일반 생명체에서 유전자의 교배 뿐 아니라, 하나의 유전자가 직접적으로 변이를 일으켜서 주어진 환경에서 살아남을 확률이 존재하기 때문에 유전자 알고리즘에서 또한 변이 연산은 주어진 해의 유전자 내의 유전 인자의 순서 혹은 값이 임의로 변경되어 다른 해로 변형되는 연산을 의미한다. 이러한 돌연변이의 방법을 통해 우리는 약간의 확률로 변이 연산을 수행하여 전체 세대가 함께 지역 최적해에 빠져드는 경우를 방지할 수 있으며, 해집단의 다양성을 높여줄 수 있다.

유전자 알고리즘은 일반적으로 계산 자원을 사용하며, 종료 조건이 없으면 알고리즘이 무한히 실

행될 수 있으며, 특정 문제나 응용에는 시간 제한이 있을 수 있다. 또한 연구 목적에 따라 종료 조건을 설정하면 알고리즘의 동작을 분석하고 평가하기 쉽기 때문에 유전자 알고리즘에는 종료 조건이 필요하다.

유전자 알고리즘에는 전형적인 종료조건과 적응적 종료조건이 있다. 전형적인 종료조건은 만족할 만한 해를 찾을 때까지 계속 해를 구하는 방식이고, 적응적 종료조건은 정해진 세대수가 되면 유전 알고리즘을 종료하고 해집단에서 가장 좋은 염색체를 찾는 방식이다.



<유전 알고리즘의 기본 동작 원리>

1) 문제 변수 영역을 고정된 길이의 염색체로 나타낸다. 해집단 크기 N, 교차율 Pc, 변이율 Pm을 정의한다. (보통 교차율은 0.7, 변이율은 0.001로 설정)

2) 문제 영역에서 개별 염색체의 성능, 적 적합도를 재는 적합도 한수를 정의한다. (적합도 함수는 재생산 과정에서 짝지어지는 염색체를 선택하는 근거이기 때문)

3) 염색체 N개로 이루어진 초기 해집단을 임의로 생성한다.

4) 염색체 각각의 적합도를 계산한다.

5) 현재 해집단에서 짝지을 염색체 한 쌍을 선택한다. 적합도에 따라 확률적으로 부모 염색체를 선택한다. 적합도가 높은 염색체는 적합도가 낮은 염색체보다 선택될 확률이 높다.

6) 유전 연산자인 교차와 변이를 적용하여 가식 염색체 한 쌍을 만든다.

7) 만들어진 자식 염색체를 새로운 해집단에 넣는다.

8) 새로운 해집단의 크기가 초기 해집단 크기인 N이 될 때까지 5단계를 반복한다.

9) 초기 해집단을 새로운 해집단으로 교체한다.

10) 4단계로 가서 종료 조건을 만족할 때까지 이 과정을 반복한다.

유전자 알고리즘 코드 정리

필요한 라이브러리를 호출한다. (numpy: 수학적 연산을 수행하기 위한 라이브러리 / random: 난수 생성을 위한 라이브러리 / matplotlib.pyplot: 데이터 시각화를 위한 라이브러리)

```
import numpy as np
import random
import matplotlib.pyplot as plt
```

fitness_function(x, y): 유전 알고리즘의 목표 함수로 위 코드의 경우 주어진 피크함수를 풀어 사용하였다. 이 함수는 입력으로 x와 y를 받아 목표 함수의 값을 계산한다.

```
def fitness_function(x, y):
    return (1 - x) ** 2 * np.exp(-x**2 - (y + 1)**2) - (x - x**3 - y**3) * np.exp(-x**2 - y**2)
```

initialize_population(population_size): 초기 해집단을 생성하는 함수로 주어진 population_size만큼 무작위로 x와 y 값을

선택하여 개체를 만들고 이를 리스트로 반환한다.

```
def initialize_population(population_size):
    population = []
    for _ in range(population_size):
        x = random.uniform(-3, 3)
        y = random.uniform(-3, 3)
        population.append([x, y])
    return population
```

crossover(parent1, parent2, crossover_rate): 두 부모 개체를 교차하여 자식 개체를 생성하는 함수로 crossover_rate 확률로 교차를 수행한다.

```
def crossover(parent1, parent2, crossover_rate):
    child = parent1.copy()
    if random.random() < crossover_rate:
        crossover_point = random.randint(0, len(parent1) - 1)
        child[crossover_point:] = parent2[crossover_point:]
    return child
```

mutate(individual, mutation_rate): 개체를 돌연변이시키는 함수로 각 요소에 대해 mutation_rate 확률로 작은 변화를 적용시킨다.

```
def mutate(individual, mutation_rate):
    for i in range(len(individual)):
        if random.random() < mutation_rate:
            individual[i] += random.uniform(-0.1, 0.1)  # 임의의 작은 변화
    return individual
'
```

genetic_algorithm(population_size, generations, crossover_rate, mutation_rate): 유전 알고리즘을 실행하는 함수로 초기 해집단을 생성하고 여러 세대를 걸쳐 진화를 수행한다. 각 세대에서 선택, 교차, 돌연변이 등의 과정을 거쳐 새로운 인구를 생성하고, 최적의 해를 찾는다. 최적의 해와 각 세대에서의 최적 해의 히스토리를 반환한다.

```
def genetic_algorithm(population_size, generations, crossover_rate, mutation_rate):
    population = initialize_population(population_size)
    best_solution = None
    best_fitness = float('-inf')
    best_fitness_history = []

    for generation in range(generations):
        new_population = []
        for _ in range(population_size):
            parent1 = random.choice(population)
            parent2 = random.choice(population)
            child = crossover(parent1, parent2, crossover_rate)
            child = mutate(child, mutation_rate)
            new_population.append(child)

        population = new_population

        best_individual = max(population, key=lambda ind: fitness_function(ind[0], ind[1]))
        current_fitness = fitness_function(best_individual[0], best_individual[1])

        if current_fitness > best_fitness:
            best_solution = best_individual
            best_fitness = current_fitness

        best_fitness_history.append(current_fitness)

        print(f"Generation {generation}: Best Fitness = {current_fitness:.6f}")

    return best_solution, best_fitness_history
```

실행에 필요한 매개변수 설정(population_size: 초기 해집단 / generations: 세대 수 / crossover_rate: 교차 확률 /

mutation_rate: 돌연변이 확률)

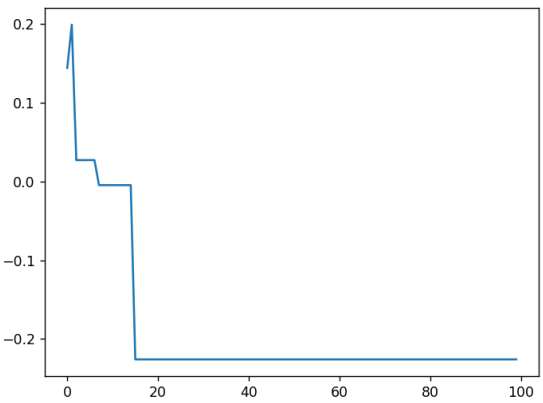
```
population_size = 6
generations = 100
crossover_rate = 0.7
mutation_rate = 0.001

best_solution, best_fitness_history = genetic_algorithm(population_size, generations, crossover_ra
te, mutation_rate)
```

결과

```
Generation 0: Best Fitness = 0.144397
Generation 1: Best Fitness = 0.199443
Generation 2: Best Fitness = 0.027301
Generation 3: Best Fitness = 0.027301
Generation 4: Best Fitness = 0.027301
Generation 5: Best Fitness = 0.027301
Generation 6: Best Fitness = 0.027301
Generation 7: Best Fitness = -0.004527
Generation 8: Best Fitness = -0.004527
Generation 9: Best Fitness = -0.004527
Generation 10: Best Fitness = -0.004527
Generation 11: Best Fitness = -0.004527
Generation 12: Best Fitness = -0.004527
Generation 13: Best Fitness = -0.004527
Generation 14: Best Fitness = -0.004527
Generation 15: Best Fitness = -0.226000
Generation 16: Best Fitness = -0.226000
Generation 17: Best Fitness = -0.226000
Generation 18: Best Fitness = -0.226000
Generation 19: Best Fitness = -0.226000
Generation 20: Best Fitness = -0.226000
Generation 21: Best Fitness = -0.226000
Generation 22: Best Fitness = -0.226000
Generation 23: Best Fitness = -0.226000
Generation 24: Best Fitness = -0.226000
Generation 25: Best Fitness = -0.226000
Generation 26: Best Fitness = -0.226000
Generation 27: Best Fitness = -0.226000
Generation 28: Best Fitness = -0.226000
Generation 29: Best Fitness = -0.226000
Generation 30: Best Fitness = -0.226000
Generation 31: Best Fitness = -0.226000
Generation 32: Best Fitness = -0.226000
Generation 33: Best Fitness = -0.226000
Generation 34: Best Fitness = -0.226000
Generation 35: Best Fitness = -0.226000
Generation 36: Best Fitness = -0.226000
Generation 37: Best Fitness = -0.226000
Generation 38: Best Fitness = -0.226000
Generation 39: Best Fitness = -0.226000
Generation 40: Best Fitness = -0.226000
Generation 41: Best Fitness = -0.226000
Generation 42: Best Fitness = -0.226000
Generation 43: Best Fitness = -0.226000
Generation 44: Best Fitness = -0.226000
Generation 45: Best Fitness = -0.226000
Generation 46: Best Fitness = -0.226000
Generation 47: Best Fitness = -0.226000
Generation 48: Best Fitness = -0.226000
Generation 49: Best Fitness = -0.226000
Generation 50: Best Fitness = -0.226000
Generation 51: Best Fitness = -0.226000
Generation 52: Best Fitness = -0.226000
Generation 53: Best Fitness = -0.226000
Generation 54: Best Fitness = -0.226000
Generation 55: Best Fitness = -0.226000
Generation 56: Best Fitness = -0.226000
Generation 57: Best Fitness = -0.226000
Generation 58: Best Fitness = -0.226000
Generation 59: Best Fitness = -0.226000
Generation 60: Best Fitness = -0.226000
Generation 61: Best Fitness = -0.226000
Generation 62: Best Fitness = -0.226000
Generation 63: Best Fitness = -0.226000
Generation 64: Best Fitness = -0.226000
Generation 65: Best Fitness = -0.226000
Generation 66: Best Fitness = -0.226000
Generation 67: Best Fitness = -0.226000
Generation 68: Best Fitness = -0.226000
Generation 69: Best Fitness = -0.226000
Generation 70: Best Fitness = -0.226000
Generation 71: Best Fitness = -0.226000
Generation 72: Best Fitness = -0.226000
Generation 73: Best Fitness = -0.226000
Generation 74: Best Fitness = -0.226000
Generation 75: Best Fitness = -0.226000
Generation 76: Best Fitness = -0.226000
Generation 77: Best Fitness = -0.226000
Generation 78: Best Fitness = -0.226000
Generation 79: Best Fitness = -0.226000
Generation 80: Best Fitness = -0.226000
Generation 81: Best Fitness = -0.226000
Generation 82: Best Fitness = -0.226000
Generation 83: Best Fitness = -0.226000
Generation 84: Best Fitness = -0.226000
Generation 85: Best Fitness = -0.226000
Generation 86: Best Fitness = -0.226000
Generation 87: Best Fitness = -0.226000
Generation 88: Best Fitness = -0.226000
Generation 89: Best Fitness = -0.226000
Generation 90: Best Fitness = -0.226000
Generation 91: Best Fitness = -0.226000
Generation 92: Best Fitness = -0.226000
Generation 93: Best Fitness = -0.226000
Generation 94: Best Fitness = -0.226000
Generation 95: Best Fitness = -0.226000
Generation 96: Best Fitness = -0.226000
Generation 97: Best Fitness = -0.226000
Generation 98: Best Fitness = -0.226000
Generation 99: Best Fitness = -0.226000
```

<결과 그래프>



유전 알고리즘은 초기에 무작위로 설정된 개체를 향상시키는 방향으로 진화하며, 시간이 지남에 따라 적합도가 향상되어 최적해에 가까워지는 알고리즘이다.

초기에 x와 y 값을 무작위로 설정했기 때문에 최적해를 찾는 과정에서 처음에 음수 적합도가 나왔으며 시간이 지남에 따라 적합도가 향상되어 최적해를 향해 수렴하게 된다. 따라서 세대수가 증가함에 따라 그래프의 안정성이 높아지며 최종적으로는 평평하게 보여진다.