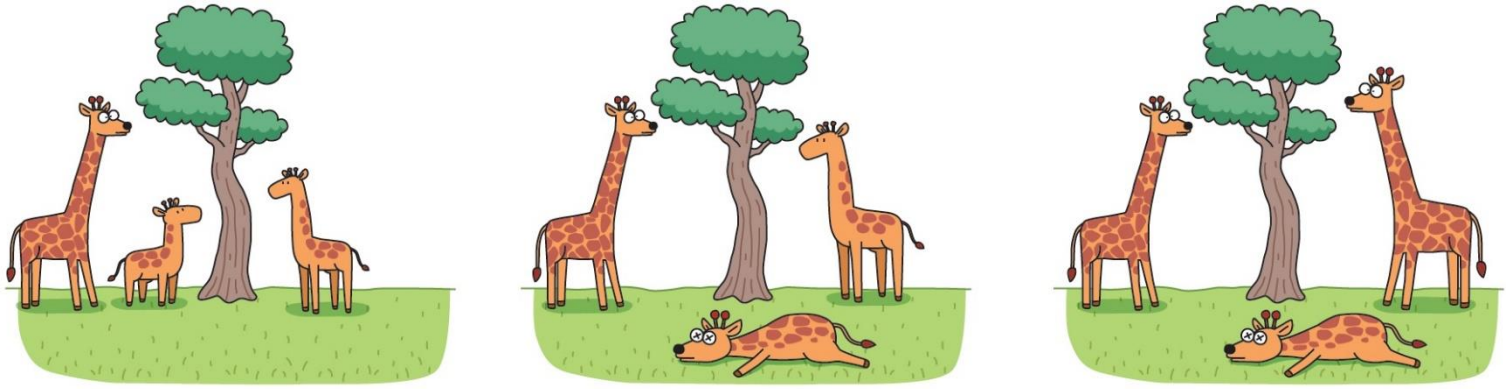
제8장 유전자 알고리즘

학습 목표

- 유전자 알고리즘의 기본 개념을 이해한다.
- 유전자 알고리즘을 여러 가지 문제에 응용해본다.
- 유전자 프로그래밍의 작동방식을 살펴본다.

자연계에서의 진화

- 유전자 알고리즘은 생물체의 염색체가 유전되는 현상에서 영감을 얻은 최적화 알고리즘으로서 적자생존 원칙에 기반을 두고 교차, 돌연변이, 도태 등의 과정을 통하여 우성 유전자만이 살아남는 자연계의 현상을 알고리즘으로 만든 것이다.



유전자 알고리즘

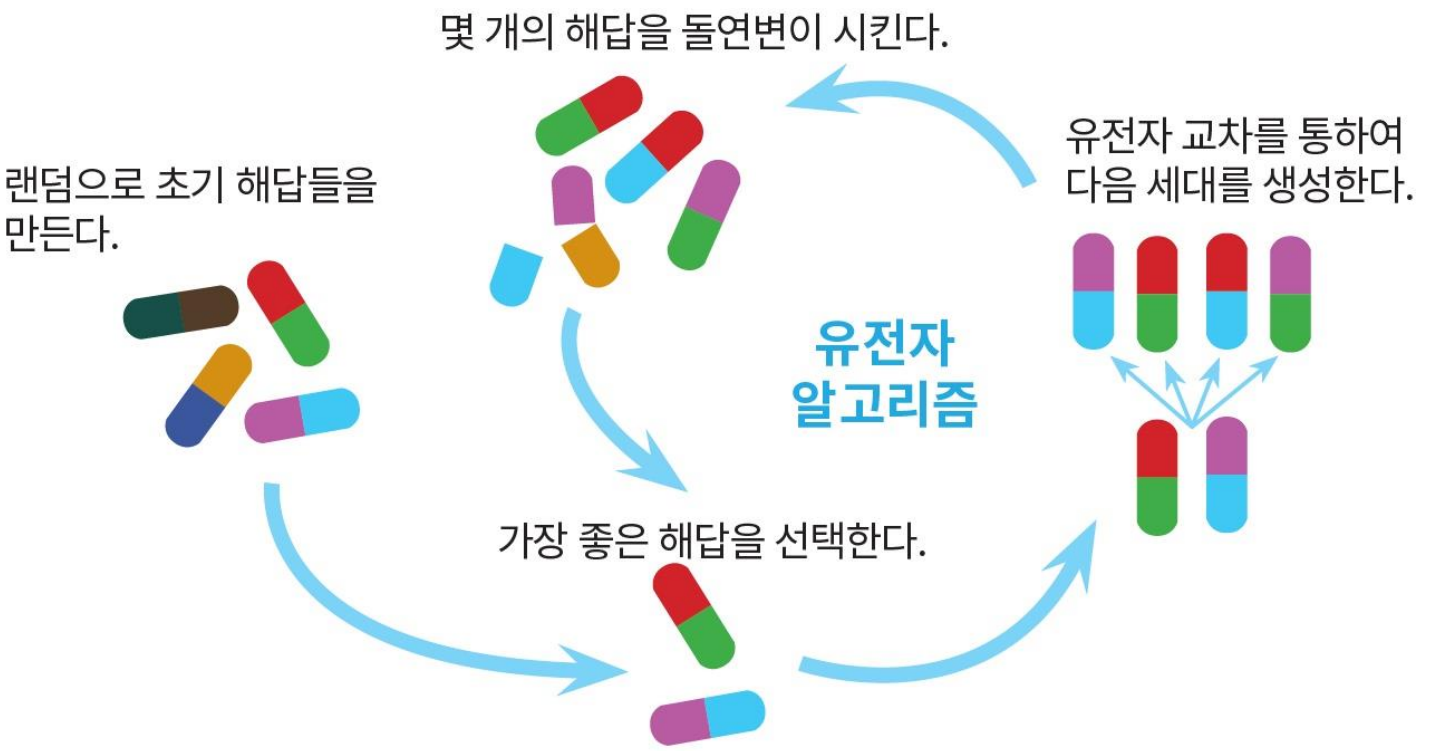


그림 8–3 유전자 알고리즘의 절차

염색체

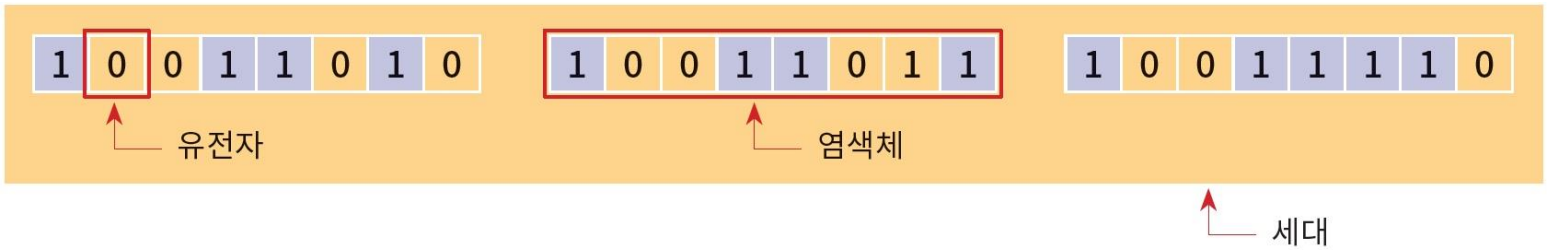


그림 8-4 유전자의 구조

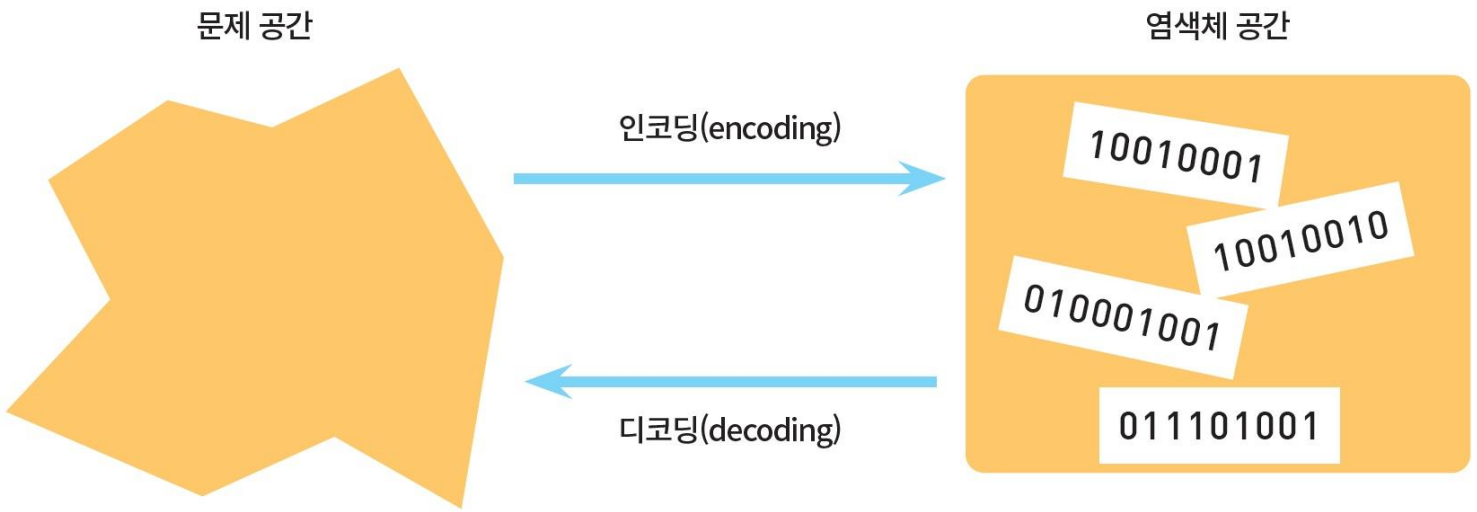


그림 8-5 인코딩과 디코딩

평가함수, 적합도, 교차

- 평가 함수(evaluation function)는 현재의 염색체가 얼마나 문제를 잘 해결하고 있는지를 나타내는 적합도(fitness)를 반환
- 재생산이 일어나면 부모의 염색체들의 일부는 교환된다. 이러한 연산을 교차(crossover)
- 돌연변이 연산자는 염색체에서 랜덤한 위치의 유전자의 값을 바꾼다.

유전자 알고리즘의 의사 코드

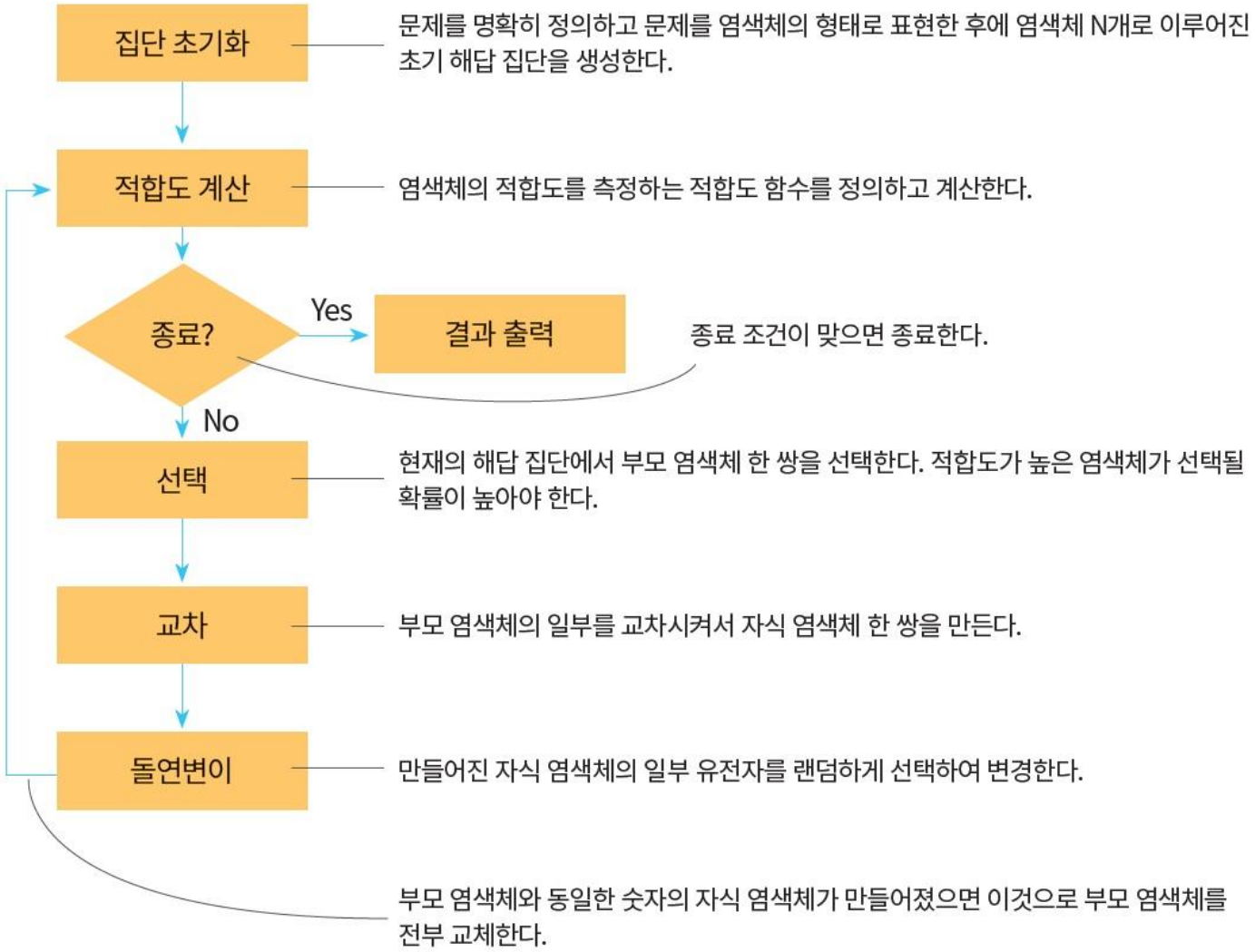
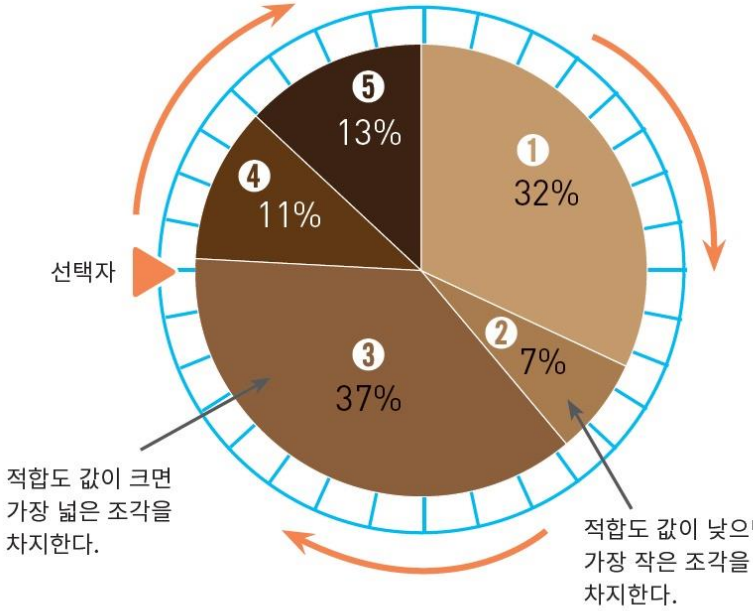


그림 8-6 유전자 알고리즘의 순서도

선택 연산자

- 유전 알고리즘은 재생산을 위하여 우수한 성능을 보이는 부모 염색체 2개를 선택한다.
- 부모 염색체를 선택할 때 많이 사용되는 알고리즘이 룰렛 휠 선택(rootlet wheel selection)이다.



염색체	확률값
C1	32%
C2	7%
C3	37%
C4	11%
C5	13%

그림 8–7 룰렛 휠 알고리즘

교차 연산자

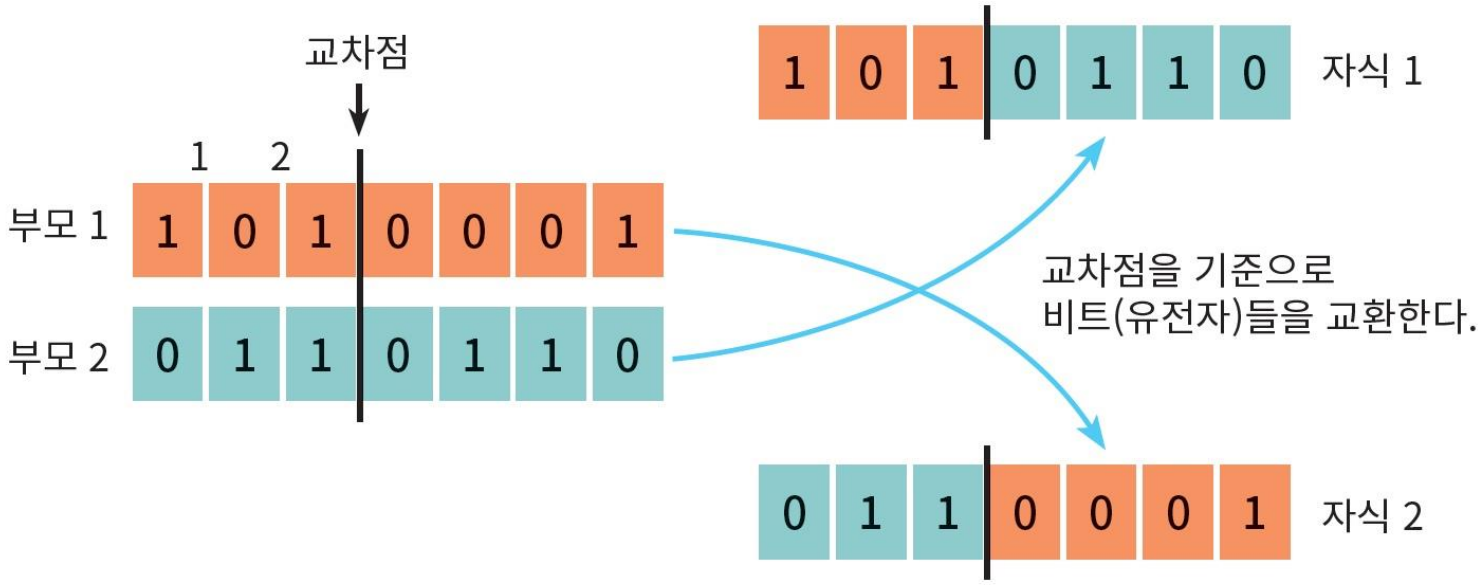
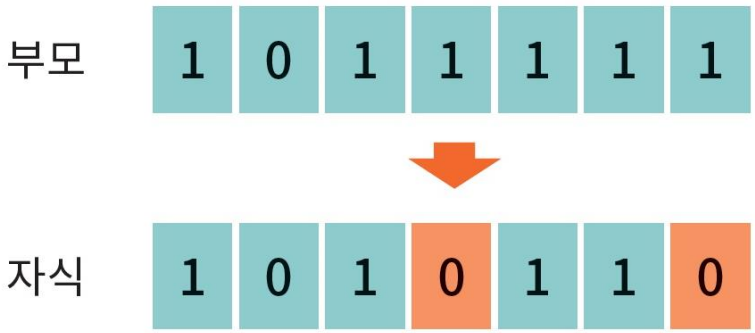


그림 8-8 교차 연산

돌연변이 연산자



의사 코드

```
GeneticAlgorithm(population, FitnessFunc)
{
        repeat
                new_population ← []
                for i=1 to size(population) do
                        father ←select(population, FitnessFunc)
                        mother ←select(population, FitnessFunc)
                        child ← crossover(father, mother)
                        if (난수 < 변이_확률) then child←mutate(child)
                        new_population ←new_population + child
                population = new_population
        until 충분히 적합한 개체가 얻어지거나 충분한 반복횟수가 지나면
        return 가장 적합한 개체
}
```

유전자 알고리즘의 예제

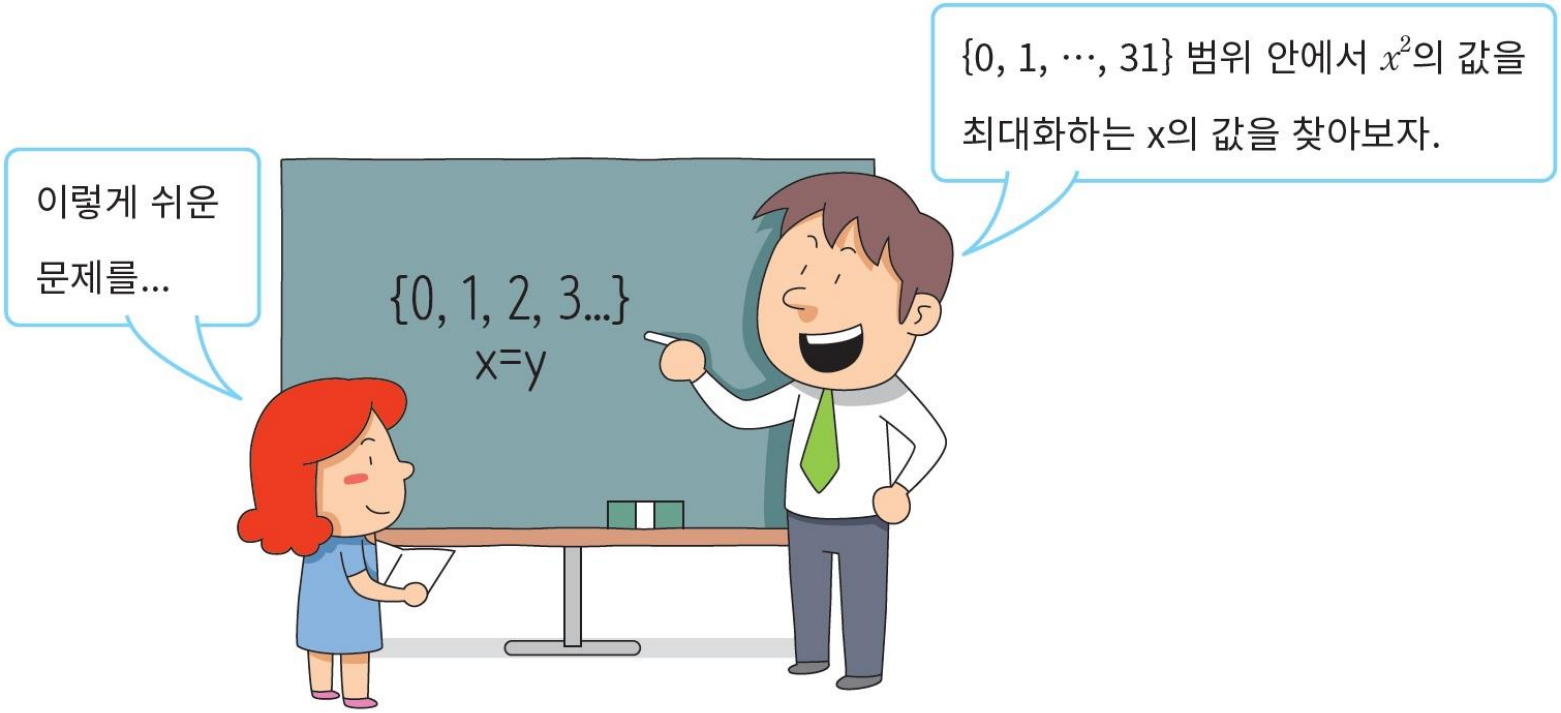


그림 8–10 유전자 알고리즘 예제

유전자 알고리즘의 예제

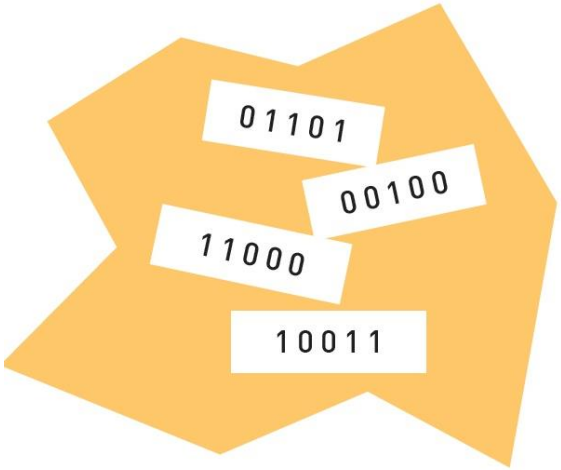
- 염색체를 어떻게 나타낼 것인가? - 0에서 31까지이므로 하나의 염색체를 크기가 5인 이진수나타낸다.

- 염색체의 개수를 몇 개로 하여야 할까? - 간단한 설명을 위하여 4개로 하자. 즉 해집단의 크기 N은 4이다.

- 교차와 돌연변이는 어떤 방법으로 할 것인가? 교차 확률 는 0.7, 돌연변이 확률 은 0.001로 한다.

초기 염색체

- 랜덤하게 생성한 1과 0을 이용하여 문자열 4개를 생성



적합도 계산

염색체 번호	초기 염색체	x값	적합도 $f(x) = x^2$	적합도 비율
1	0 1 1 0 1	13	169	15%
2	1 1 0 0 0	24	576	51%
3	0 0 1 0 0	4	16	1%
4	1 0 0 1 1	19	361	32%
합계			1,122	

룰렛 휠 선택 방법

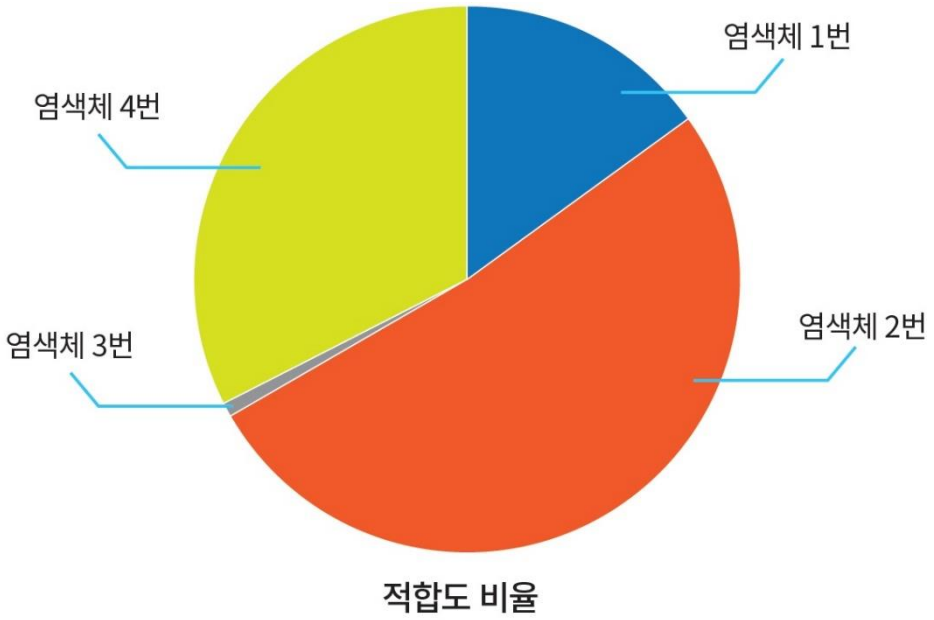


그림 8-12 룰렛 휠

우리의 예에서는 처음 두 번의 회전으로 1번과 2번이 선택되고, 다음 2번의 회전으로 2번과 4번이 선택되었다고 가정한다.

교차 연산

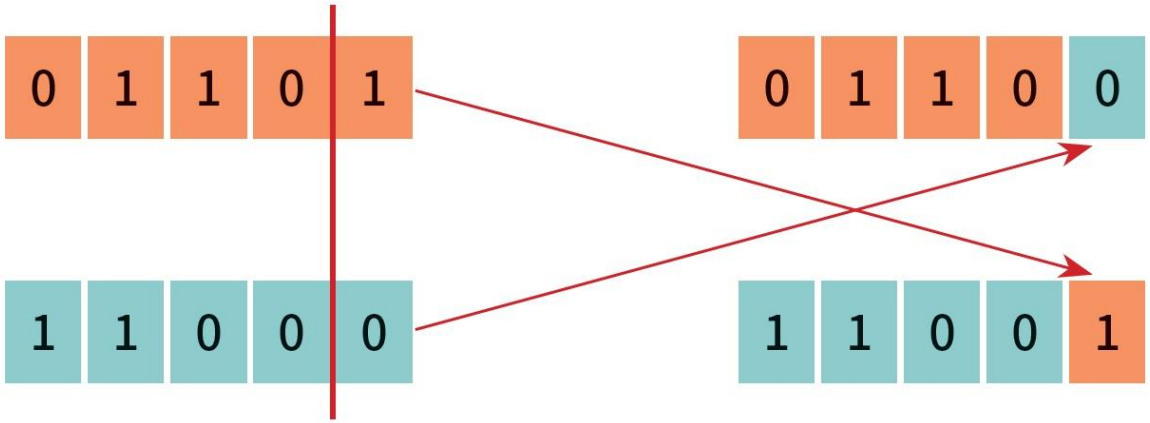


그림 8-13 교차 연산

교차 연산 후의 적합도

염색체 번호	초기 염색체	교차점	자식 염색체	x 값	적합도
1	0 1 1 0 | 1	4	0 1 1 0 0	12	144
2	1 1 0 0 | 0	4	1 1 0 0 1	25	625
2	1 1 | 0 0 0	2	1 1 0 1 1	27	729
4	1 0 | 0 1 1	2	1 0 0 0 0	16	256
합계					1,754

세대 교체

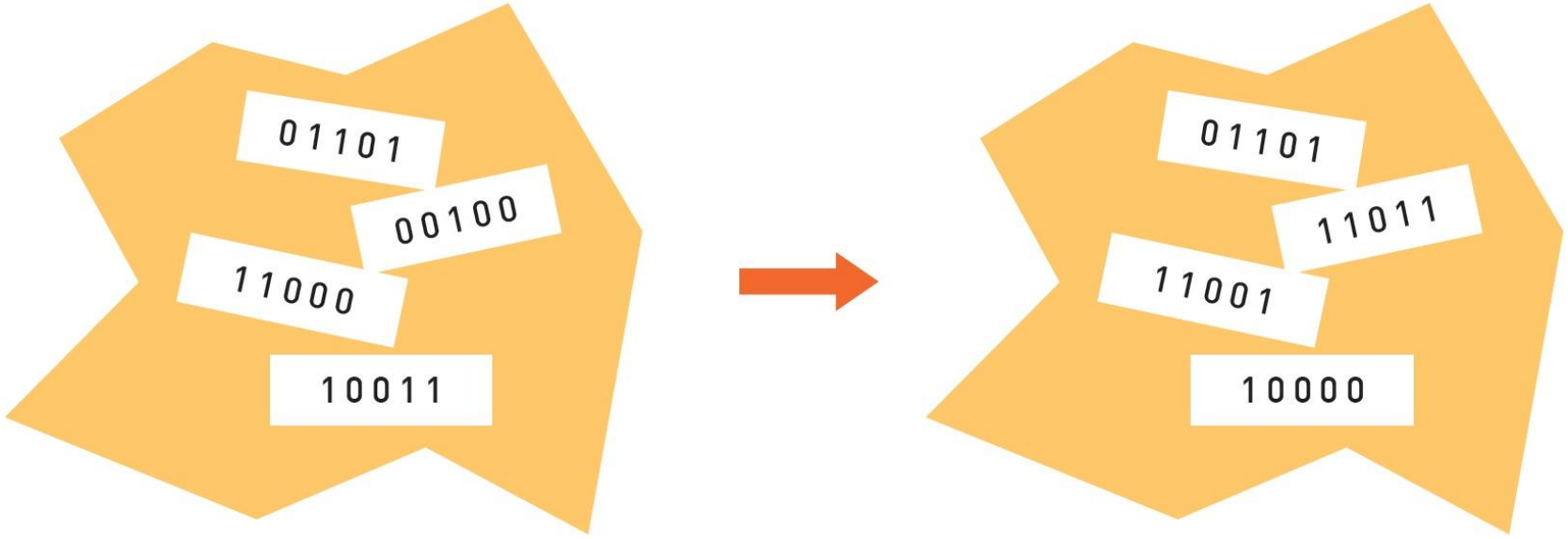


그림 8–14 교차 연산 후 염색체 집단

확률변수의 연산이 필요한 이유

- 지역 최대점 문제에 빠지는 것을 방지

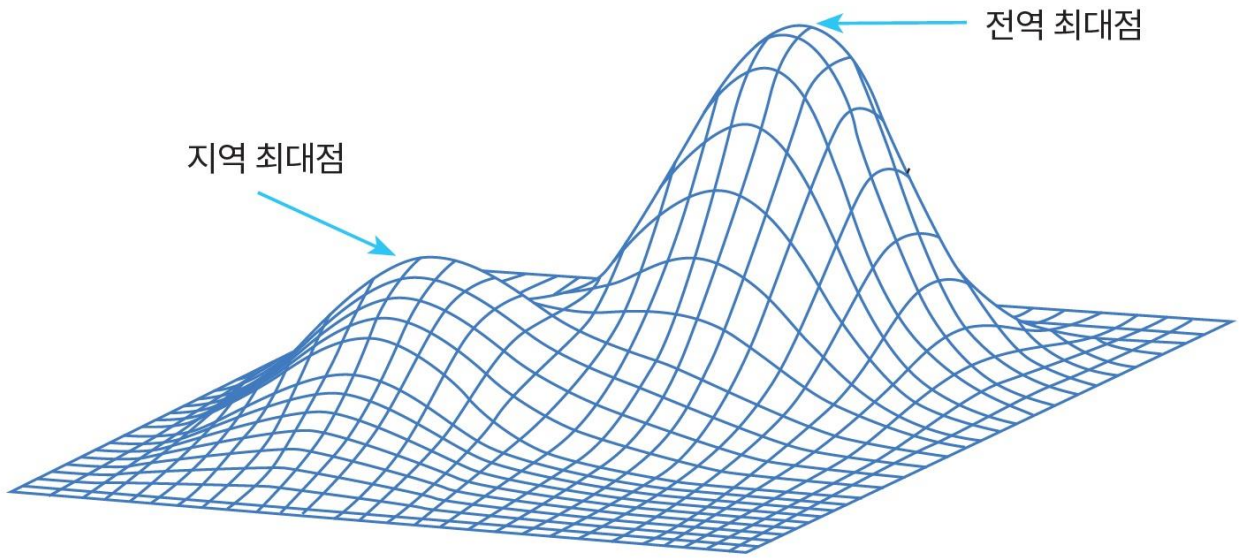


그림 8–15 지역 최대점 문제

돌연변이

새로운 염색체 번호	자식 염색체	돌연변이 위치	자식 염색체	x 값	적합도
1	0 1 1 0 0	4	0 1 1 1 0	14	196
2	1 1 0 0 1	0	1 1 0 0 1	25	625
3	1 1 0 1 1	0	1 1 0 1 1	27	729
4	1 0 0 0 0	0	1 0 0 0 0	16	256
합계					1,806

유전자 알고리즘 프로그램

```
# 염색체를 클래스로 정의한다.
class Chromosome:
    def __init__(self, g=[]):
        self.genes = g.copy()                    # 유전자는 리스트로 구현된다.
        self.fitness = 0                 # 적합도
        if self.genes.__len__()==0:    # 염색체가 초기 상태이면 초기화한다.
            i = 0
            while i<SIZE:
                if random.random() >= 0.5: self.genes.append(1)
                else: self.genes.append(0)
                i += 1

    def cal_fitness(self):               # 적합도를 계산한다.
        self.fitness = 0;
        value = 0
        for i in range(SIZE):
            value += self.genes[i]*pow(2,SIZE-1-i)
        self.fitness = value
        return self.fitness

    def __str__(self):
        return self.genes.__str__()
```

유전자 알고리즘 프로그램

```
# 염색체와 적합도를 출력한다.
def print_p(pop):
    i = 0
    for x in pop:
        print("염색체 #", i, "=", x, "적합도=", x.cal_fitness())
        i += 1
    print("")

# 선택 연산
def select(pop):
    max_value  = sum([c.cal_fitness() for c in population])
    pick    = random.uniform(0, max_value)
    current = 0

    # 룰렛휠에서 어떤 조각에 속하는지를 알아내는 루프
    for c in pop:
        current += c.cal_fitness()
        if current > pick:
            return c
```

유전자 알고리즘 프로그램

```
# 교차 연산
def crossover(pop):
    father = select(pop)
    mother = select(pop)
    index = random.randint(1, SIZE - 2)
    child1 = father.genes[:index] + mother.genes[index:]
    child2 = mother.genes[:index] + father.genes[index:]
    return (child1, child2)

# 돌연변이 연산
def mutate(c):
    for i in range(SIZE):
        if random.random() < MUTATION_RATE:
            if random.random() < 0.5:
                c.genes[i] = 1
            else:
                c.genes[i] = 0
```

유전자 알고리즘 프로그램

```
# 메인 프로그램
population = []
i=0

# 초기 염색체를 생성하여 객체 집단에 추가한다.
while i<POPULATION_SIZE:
    population.append(Chromosome())
    i += 1

count=0
population.sort(key=lambda x: x.cal_fitness(), reverse=True)
print("세대 번호=", count)
print_p(population)
count=1
```

유전자 알고리즘 프로그램

```
while population[0].cal_fitness() < 31:
    new_pop = []

    # 선택과 교차 연산
    for _ in range(POPULATION_SIZE//2):
        c1, c2 = crossover(population);
        new_pop.append(Chromosome(c1));
        new_pop.append(Chromosome(c2));

    # 자식 세대가 부모 세대를 대체한다.
    # 깊은 복사를 수행한다.
    population = new_pop.copy();

    # 돌연변이 연산
    for c in population: mutate(c)

    # 출력을 위한 정렬
    population.sort(key=lambda x: x.cal_fitness(), reverse=True)
    print("세대 번호=", count)
    print_p(population)
    count += 1
    if count > 100 : break;
```

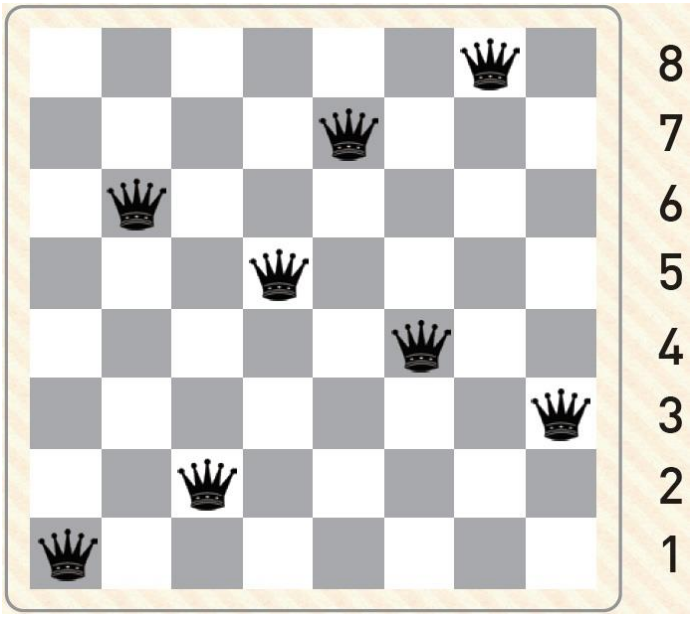
실행 결과

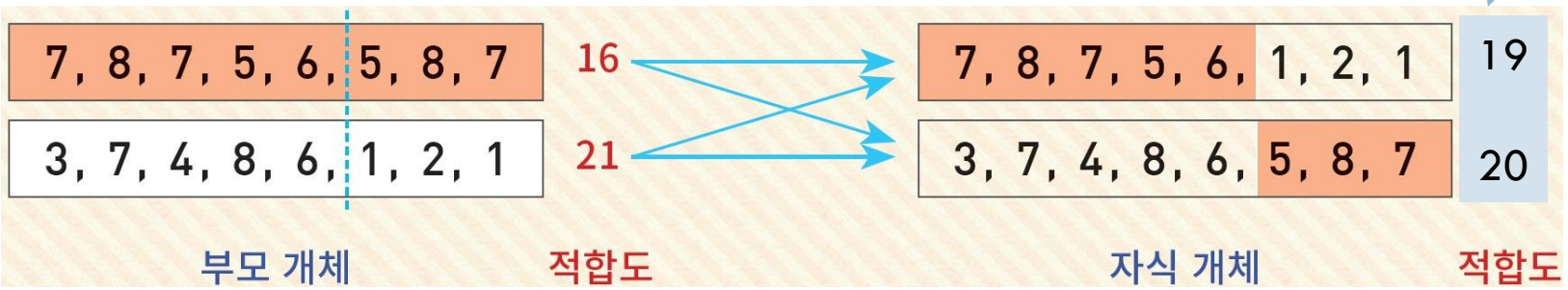
```
세대 번호= 0
염색체 # 0 = [1, 1, 1, 0, 0] 적합도= 28
염색체 # 1 = [0, 1, 1, 1, 0] 적합도= 14
염색체 # 2 = [0, 1, 1, 1, 0] 적합도= 14
염색체 # 3 = [0, 0, 0, 0, 0] 적합도= 0

세대 번호= 1
염색체 # 0 = [1, 1, 1, 1, 0] 적합도= 30
염색체 # 1 = [1, 1, 1, 0, 0] 적합도= 28
염색체 # 2 = [1, 1, 1, 0, 0] 적합도= 28
염색체 # 3 = [0, 1, 1, 0, 0] 적합도= 12
...
...
세대 번호= 9
염색체 # 0 = [1, 1, 1, 1, 0] 적합도= 30
염색체 # 1 = [1, 1, 1, 0, 1] 적합도= 29
염색체 # 2 = [1, 1, 0, 1, 1] 적합도= 27
염색체 # 3 = [1, 1, 0, 0, 0] 적합도= 24

세대 번호= 10
염색체 # 0 = [1, 1, 1, 1, 1] 적합도= 31
염색체 # 1 = [1, 1, 1, 1, 1] 적합도= 31
염색체 # 2 = [1, 1, 1, 0, 1] 적합도= 29
염색체 # 3 = [1, 1, 0, 1, 0] 적합도= 26
```

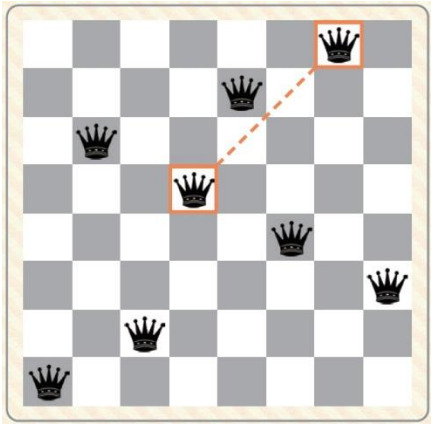
Lab: 8-queen


8
7
6
5
4
3
2
1


오타!!
7, 8, 7, 5, 6, 5, 8, 7
3, 7, 4, 8, 6, 1, 2, 1
16
21
7, 8, 7, 5, 6, 1, 2, 1
3, 7, 4, 8, 6, 5, 8, 7
19
20
부모 개체
적합도
자식 개체
적합도

Lab: 8-queen

- 8-queen 문제에서 적합도 함수는 어떻게 만들면 좋을까?
- 적합도 값으로 서로 공격하지 않는 queen 쌍의 개수를 사용하자. 만약 모든 queen 들이 서로를 공격하지 않는다면 28이 된다.
- 여기서 서로를 공격하는 queen 쌍의 개수를 뺀다. 따라서 적합도 함수는 28-h가 된다.



Lab: TSP

- "도시 목록과 도시 간의 거리를 제공하면 각 도시를 방문하여 출발 도시로 돌아오는 가장 짧은 경로는 무엇인가?"

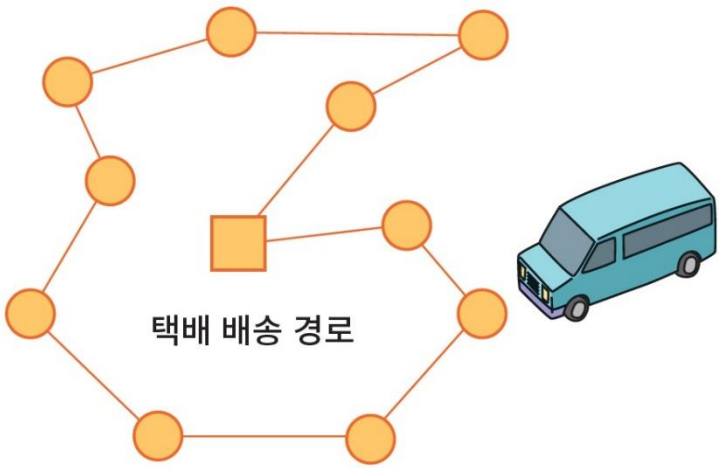


그림 8–16 택배 배송 경로

Lab: TSP

- 유전자: 하나의 도시 ((x, y) 좌표로 나타낸다.)
- 염색체: 하나의 경로
- 세대: 염색체의 모임
- 적합도 함수: 각 경로가 얼마나 좋은지를 알려주는 함수
- 돌연변이: 경로에서 도시들을 랜덤하게 바꾼다.

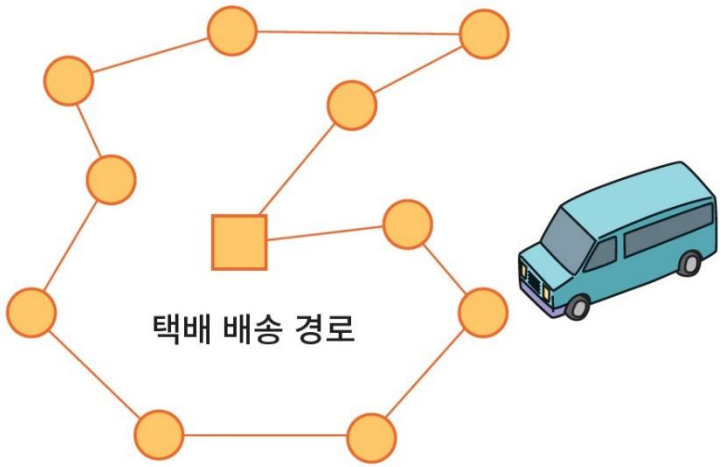


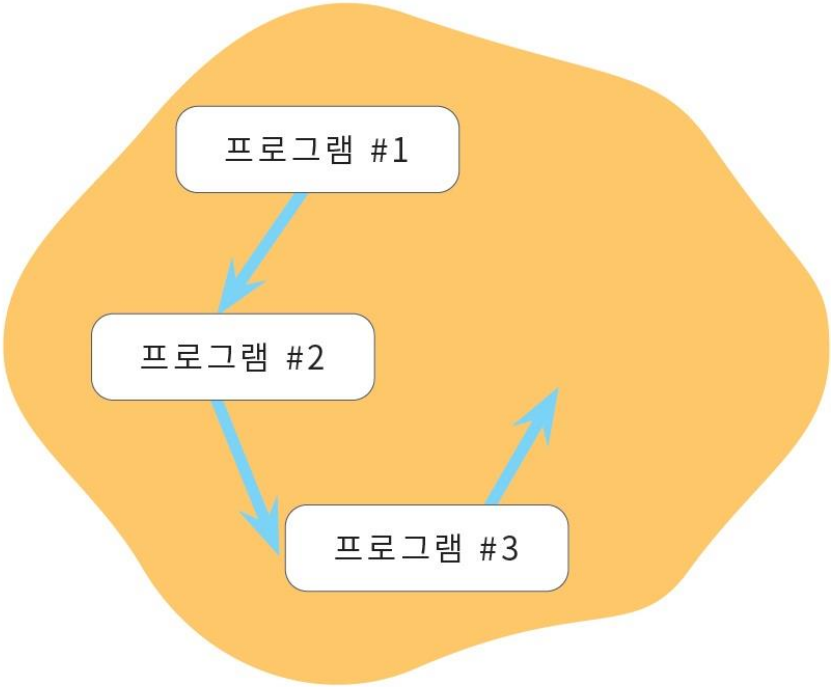
그림 8-16 택배 배송 경로

유전자 알고리즘의 장단점

- 결정론적 알고리즘이 존재하는 문제에 대해서는 유전자 알고리즘 접근법이 의미가 없다.
- 유전자 알고리즘은 본질적으로 확률적 특성으로 인해 수행 시간을 예측할 수 없다.
- 기존의 방법으로는 해결할 수 없는 제약 조건 만족 문제와 같이 완벽하게 최적의 솔루션이 필요 없는 어려운 문제에 종종 사용된다.

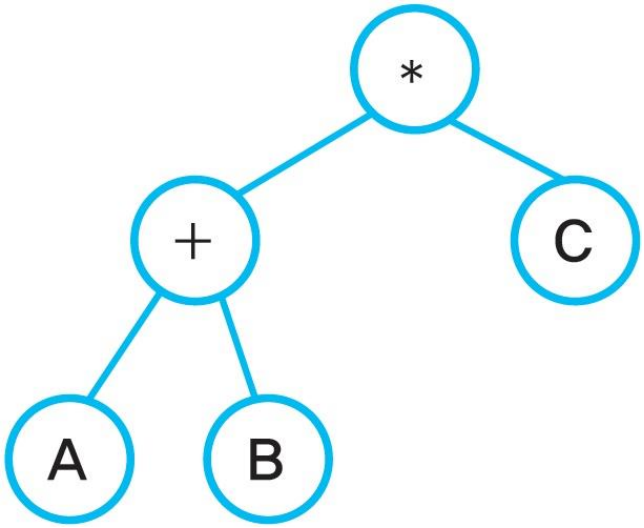
유전자 프로그래밍

- 유전자 프로그래밍(GP: Genetic Programming)은 유전자 알고리즘의 원리를 프로그래밍에 적용한 것이다.



유전자 프로그래밍

- 유전자 프로그래밍에서는 유전 연산자를 이용하여 트리 자체를 변경해야 하기 때문에, C언어나 파이썬과 같은 코드에는 적용할 수 없다. 적합한 언어가 하나 있는데 바로 LISP이다.



기본 연산

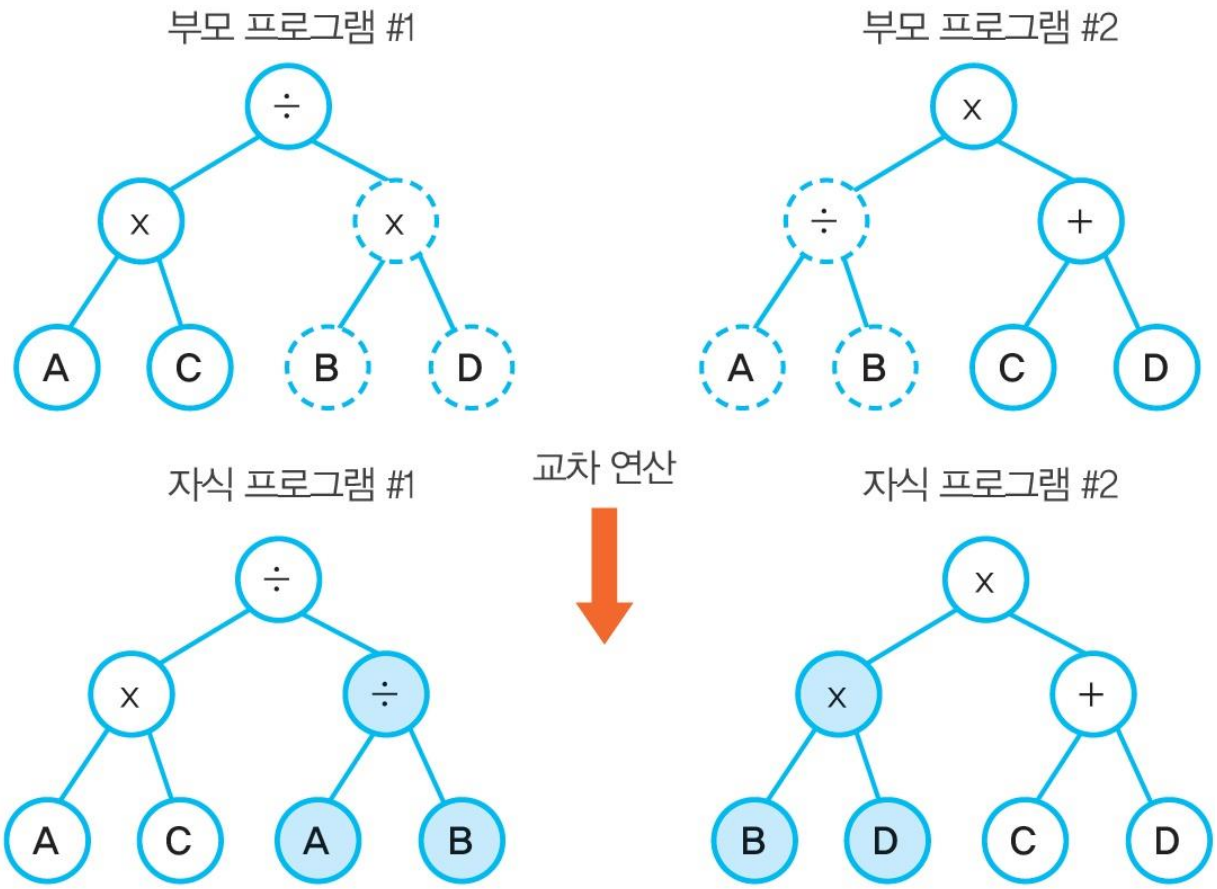


그림 8-18 수식 트리의 교차 연산

기본 연산

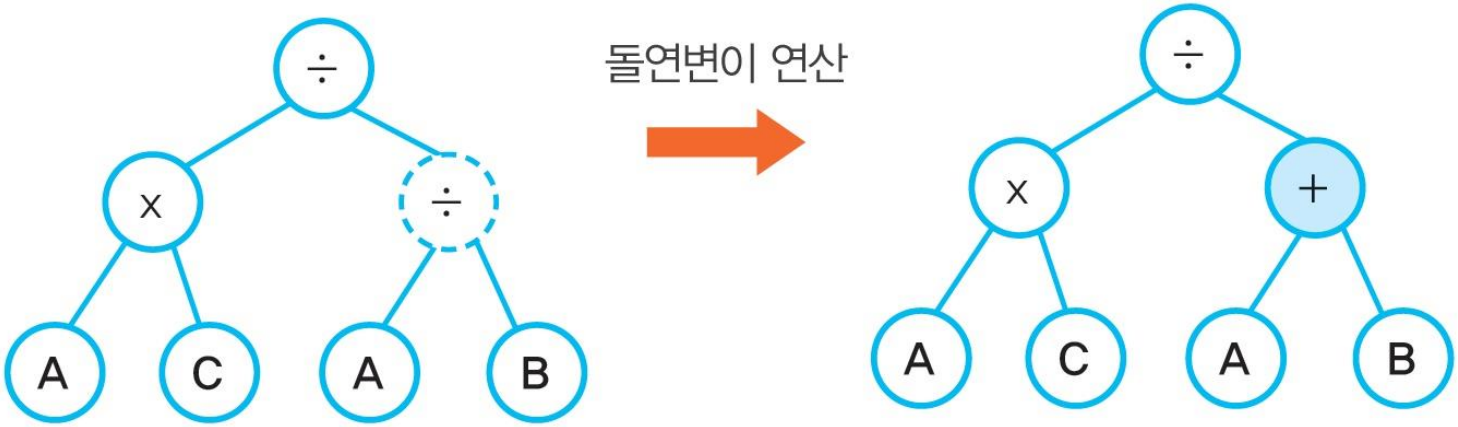


그림 8–19 수식 트리의 돌연변이 연산

GP 알고리즘

1. 초기 세대를 랜덤하게 생성한다.
2. 다음을 반복한다.
    1. 현재 세대의 각 프로그램을 주어진 데이터 집합에 대하여 평가한 후에 적합도를 계산한다. 2. 현재 세대 안의 프로그램을 랜덤하게 선택하여 적합도를 비교한다.
    3. 다음과 같은 3가지의 유전자 알고리즘의 연산자를 적용한다.
        * 교차 연산
        * 재생산 연산
        * 돌연변이 연산
    4. 결과 세대 안의 모든 프로그램을 평가한다.
3. 기준이 만족될 때까지 반복한다.

Summary

- 자연계의 진화 현상을 인공지능에 응용한 것이 유전자 프로그래밍이다. 유전자 알고리즘은 유전학에서 영감을 얻은 자연선택, 교차, 돌연변이를 사용한다. 염색체는 여러 개의 유전자로 이루어지며 하나의 유전자는 일반적으로 0과 1로 표시한다.

- 유전자 알고리즘에서는 개체 집단을 만들어 적합도를 평가하고 선택이나 교차, 돌연변이 연산자를 적용하여 새로운 해집단을 생성한다. 이 과정을 일정한 횟수만큼 되풀이 한다.

- 유전자 프로그래밍은 1990년대에 존 코자가 개척한 분야이다. 스스로 진화하는 프로그램을 작성하려는 시도였다. 유전자 알고리즘처럼 진화적인 방법을 적용하지만 대상이 이진 문자열이 아니고 트리 형태인 컴퓨터 프로그램이다.

Q & A