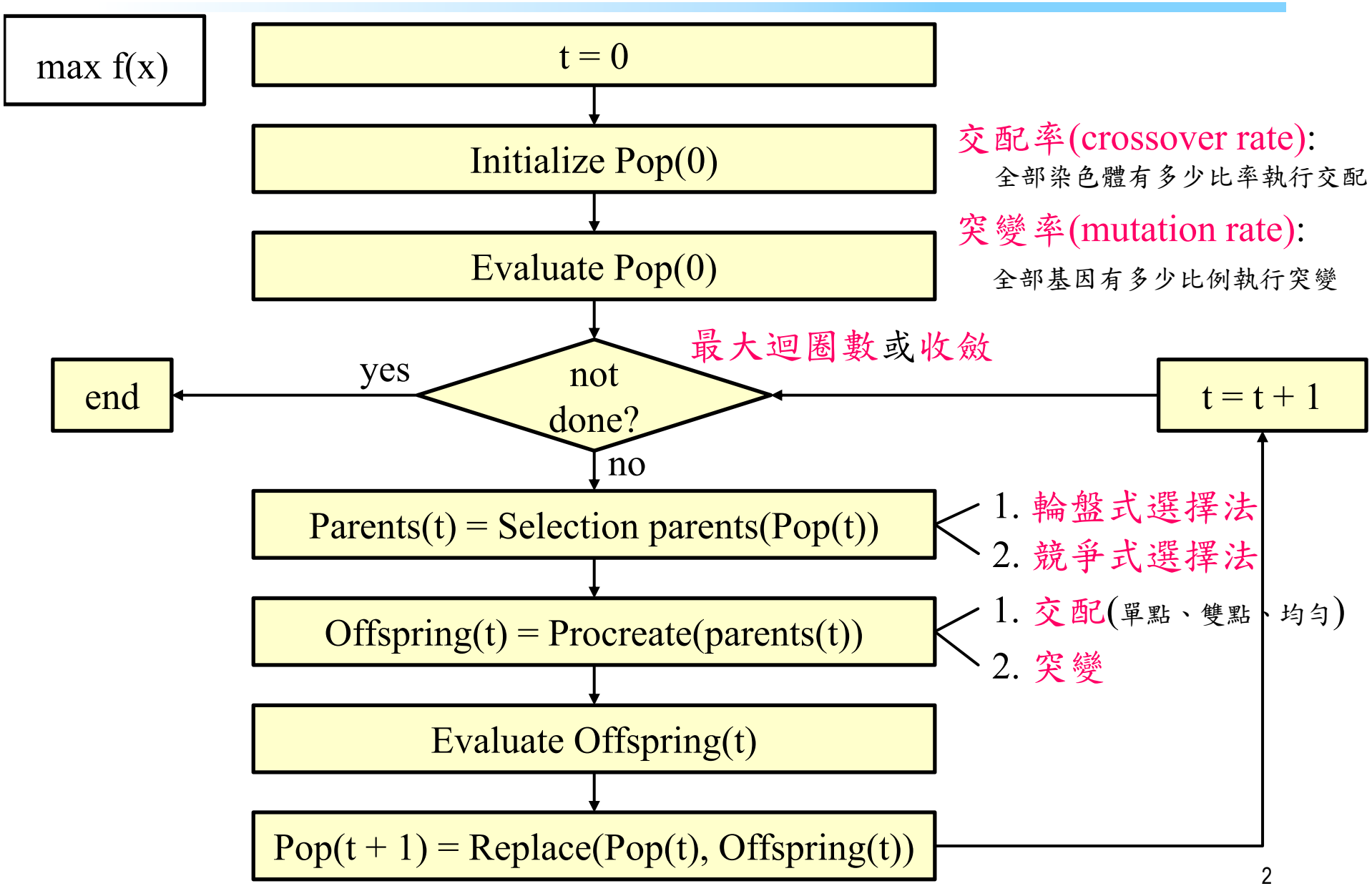
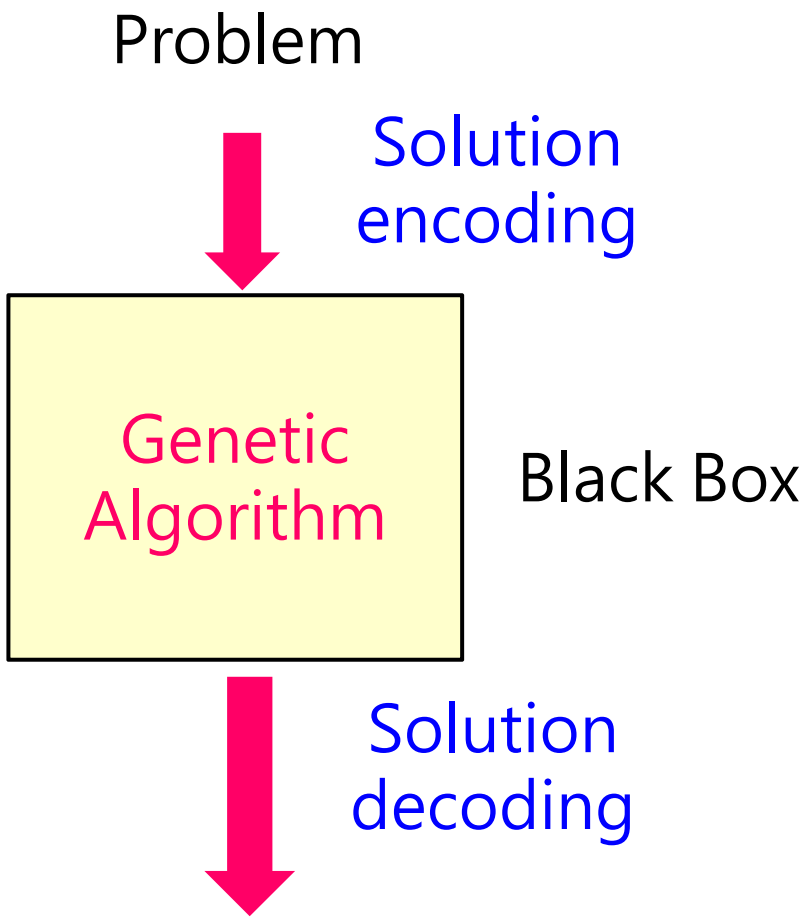
GA使用二元離散解的編碼

基因演算法流程圖


max f(x)
t = 0
Initialize Pop(0)
Evaluate Pop(0)
not done?
yes
end
no
最大迴圈數或收斂
t = t + 1
Parents(t) = Selection parents(Pop(t))
1. 輪盤式選擇法
2. 競爭式選擇法
Offspring(t) = Procreate(parents(t))
1. 交配(單點、雙點、均勻)
2. 突變
Evaluate Offspring(t)
Pop(t + 1) = Replace(Pop(t), Offspring(t))
交配率(crossover rate):
全部染色體有多少比率執行交配
突變率(mutation rate):
全部基因有多少比例執行突變

Framework of using the GA



- Optimal solutions or approximation solutions
- Could solve large-scale problems

Solution encoding/decoding

- Encoding continuous decision variables

3.6	7.2	4.9	1.3	2.9

- Encoding discrete decision variables

1	0	0	1	1

3	7	4	3	2

- Encoding permutation solutions

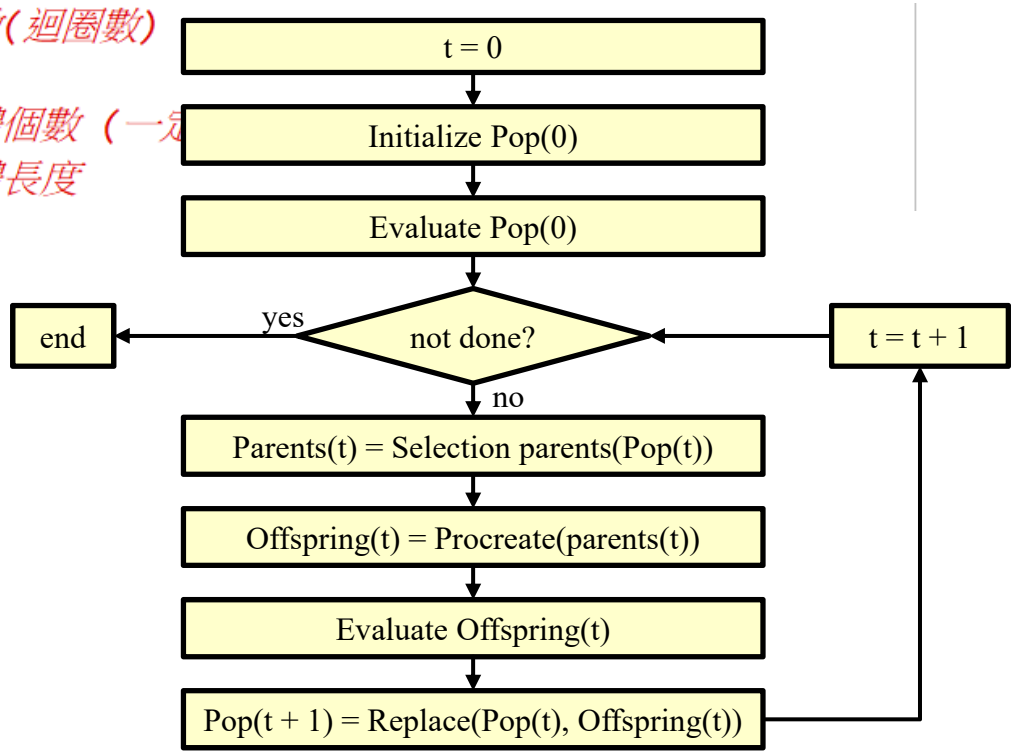
3	1	5	4	2

- Mixed encoding

3	1	5	4	2	1	0	0	1	1

```python
NUM_ITERATION = 20          # 世代數(迴圈數)

NUM_CHROME = 20             # 染色體個數 (一定
NUM_BIT = 6                 # 染色體長度
```




```python
# ==== 主程式 ====
pop = initPop()             # 初始化 pop
pop_fit = evaluatePop(pop)  # 算 pop 的 fit

for i in range(NUM_ITERATION) :
    parent = selection(pop, pop_fit)            # 挑父母
    offspring = crossover(parent)               # 交配
    mutation(offspring)                         # 突變
    offspring_fit = evaluatePop(offspring)      # 算子代的 fit
    pop, pop_fit = replace(pop, pop_fit, offspring, offspring_fit)    # 取代

    print('iteration %d: x = %s, y = %d'    %(i, pop[0], pop_fit[0]))
```

```python
NUM_ITERATION = 20              # 世代數(迴圈數)

NUM_CHROME = 20                 # 染色體個數 (一定要偶數)
NUM_BIT = 6                     # 染色體長度
```

1. 編碼：6個二元編碼，000000~111111
初始化：隨機設為6個二元編碼

```python
def initPop():                  # 初始化群體
    return np.random.randint(2, size=(NUM_CHROME,NUM_BIT)) # 產生 NUM_CHROME 個二元編碼
```

```python
def fitFunc(x):                 # 適應度函數
    # 將[1, 2, ..., NUM_BIT-1]的二元數轉成整數(第0數是符號數)
    fitness = int("".join(str(i) for i in x[1:NUM_BIT]), 2)
    return 1024 - fitness * fitness

def evaluatePop(p):             # 評估群體之適應度
    return [fitFunc(p[i]) for i in range(len(p))]
```

2. 適應度：轉成10進位x_d，
然後計算 fit = $1024 - x_d^2$

```python
# ==== 主程式 ====
pop = initPop()                 # 初始化 pop
pop_fit = evaluatePop(pop)      # 算 pop 的 fit

for i in range(NUM_ITERATION) :
    parent = selection(pop, pop_fit)                # 挑父母
    offspring = crossover(parent)                   # 交配
    mutation(offspring)                             # 突變
    offspring_fit = evaluatePop(offspring)          # 算子代的 fit
    pop, pop_fit = replace(pop, pop_fit, offspring, offspring_fit)    # 取代

    print('iteration %d: x = %s, y = %d'    %(i, pop[0], pop_fit[0]))
```

```python
Pc = 0.5                                # 交配率 (代表共執行Pc*NUM_CHROME/2次交配)
Pm = 0.01                               # 突變率 (代表共要執行Pm*NUM_CHROME*NUM_BIT次突變)

NUM_PARENT = NUM_CHROME                 # 父母的個數
NUM_CROSSOVER = int(Pc * NUM_CHROME / 2)    # 交配的次數
NUM_CROSSOVER_2 = NUM_CROSSOVER*2       # 上數的兩倍
NUM_MUTATION = int(Pm * NUM_CHROME * NUM_BIT)   # 突變的次數
```

0.5 * 20 /2 → 5

0.01 * 20 * 6 → 1

```python
def selection(p, p_fit):    # 用二元競爭式選擇法來挑父母
    a = []
    for i in range(NUM_PARENT):
        [j, k] = np.random.choice(NUM_CHROME, 2, replace=False)  # 任選兩個index
        if p_fit[j] > p_fit[k] :                    # 擇優
            a.append(p[j])
        else:
            a.append(p[k])

    return a
```

3. 選擇：二元競爭式選擇法

```python
def crossover(p):           # 用單點交配來繁衍子代
    a = []
    for i in range(NUM_CROSSOVER) :
        c = np.random.randint(1, NUM_BIT)           # 隨機找出單點(不包含0)
        [j, k] = np.random.choice(NUM_PARENT, 2, replace=False)  # 任選兩個index

        a.append(np.concatenate((p[j][0: c], p[k][c: NUM_BIT]), axis=0))
        a.append(np.concatenate((p[k][0: c], p[j][c: NUM_BIT]), axis=0))

    return a
```

4. 交配：單點交配

```python
Pc = 0.5                                # 交配率 (代表共執行Pc*NUM_CHROME/2次交配)
Pm = 0.01                               # 突變率 (代表共要執行Pm*NUM_CHROME*NUM_BIT次突變)

NUM_PARENT = NUM_CHROME                                 # 父母的個數
NUM_CROSSOVER = int(Pc * NUM_CHROME / 2)                # 交配的次數
NUM_CROSSOVER_2 = NUM_CROSSOVER*2                       # 上數的兩倍
NUM_MUTATION = int(Pm * NUM_CHROME * NUM_BIT)           # 突變的次數
```

```python
def mutation(p):                        # 突變
    for _ in range(NUM_MUTATION) :
        row = np.random.randint(NUM_CROSSOVER_2)        # 任選一個染色體
        col = np.random.randint(NUM_BIT)                # 任選一個基因

        p[row][col] = (p[row][col] + 1) % 2             # 對應此染色體的此基因01互換
```

5. 突變：任選一染色體的一個基因，01互換

```python
def sortChrome(a, a_fit):               # a的根據a_fit由大排到小
    a_index = range(len(a))                                 # 產生 0, 1, 2, ..., |a|-1 的 list

    # a_index 根據 a_fit 的大小由大到小連動的排序
    a_fit, a_index = zip(*sorted(zip(a_fit,a_index), reverse=True))

    # 根據 a_index 的次序來回傳 a，並把對應的 fit 回傳
    return [a[i] for i in a_index], a_fit
```

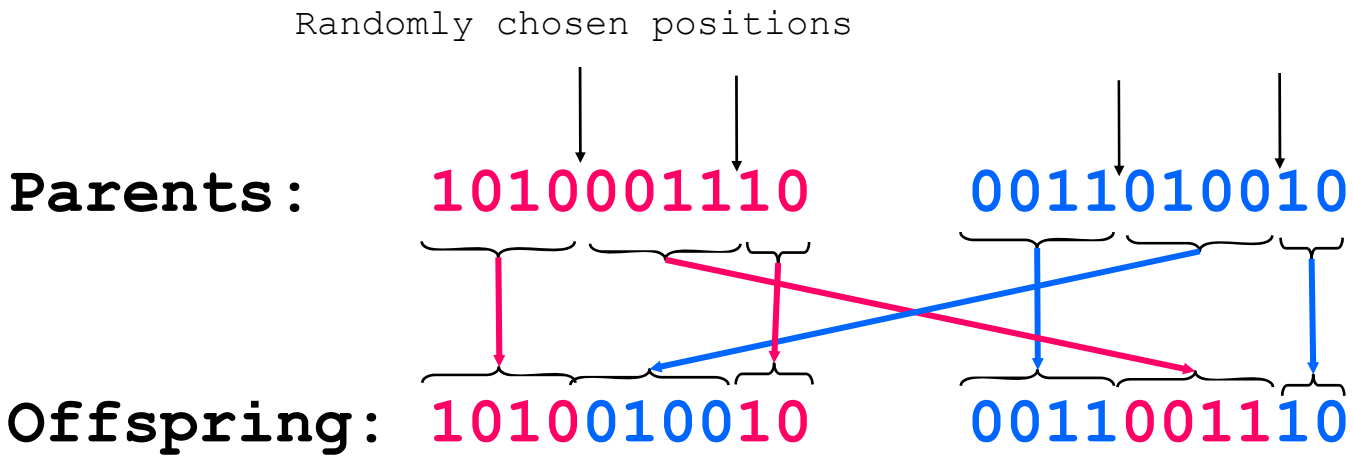
```python
def replace(p, p_fit, a, a_fit):
    b = np.concatenate((p,a), axis=0)                       # 把本代 p 和子代 a 合併成 b
    b_fit = p_fit + a_fit                                   # 把上述兩代的 fitness 合併成
    b, b_fit = sortChrome(b, b_fit)                         # b 和 b_fit 連動的排序

    return b[:NUM_CHROME], list(b_fit[:NUM_CHROME]) # 回傳 NUM_CHROME 個為新的一
```

6. 取代：Pop(t+1) = {Pop(t) - {worsts}} ∪ {kids}

Exercise

- Implement the two-point crossover operation in the sample code “GA.py”.



- (Optional)
  Implement the uniform crossover operation.