Part I Evolutionary Computation

Chapter 3 Genetic Algorithms

引子

•遥远的世界有一群贝壳快乐的生活着

•都把火狐作为自己的图腾

•如何让变得更像火狐呢？

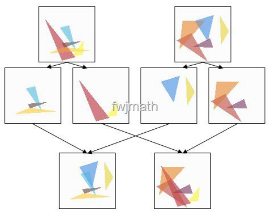
•孟德尔：我们每个个体都是有不同的多边形组成，如果能互换一些多边形，也许有更好的个体产生。或者将自己内部的多边形做一些改变

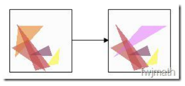
引子

•国王发布了两条法令：

1. 允许贝壳之间交换图形

2. 允许贝壳自行改变自己的颜色





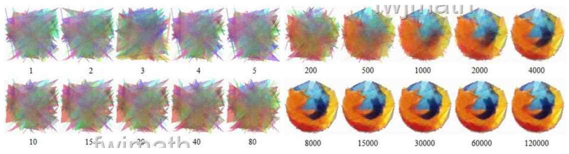
•很快产生了更像“火狐”的贝壳，但却很快终止了。

引子

•达尔文说：“物竞天择，适者生存”

•国王发布了新的法令：每一年，贝壳世界将淘汰长得最不像“火狐”的贝壳，同时用长得最像的贝壳的后代取代。

•日子一天天的逝去，奇迹终将出现..



Genetic Algorithms

- The Genetic Algorithms (GA) as powerful and broadly applicable stochastic searchand optimization techniques, are perhaps the most widely known types of Evolutionary Computation methodstoday.
- In past few years, the GA community has turned much of its attention to the optimization problems of industrial engineering resulting in a fresh body of research and applications.

GA

3-1 History of GA

– Genetic Algorithms (GA), developed by Dr. Holland.

- Holland, J.: *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975
- Goldberg, D.: *Genetic Algorithms in Search, Optimization and Machine Learning* Addison-Wesley, Reading, MA, 1989.

– Fogel, D.: *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence,* IEEE Press, Piscataway, NJ, 1995.
– Back, T.: *Evolutionary Algorithms in Theory and Practice,* Oxford University Press, New York, 1996.
– Michalewicz, Z.: *Genetic Algorithm + Data Structures = Evolution Programs.* 3rd ed., New York: Springer-Verlag, 1996.
– Gen, M. & R. Cheng: *Genetic Algorithms and Engineering Design,* John Wiley, New York, 1997.
– Gen, M. & R. Cheng: *Genetic Algorithms and Engineering Optimization,* John Wiley, New York, 2000.
– Deb, K.: *Multi-objective optimization Using Evolutionary Algorithms,* John Wiley, 2001.

3-2 General Structure of GA

- In general, a GA has five basic components.

•A genetic representation of potential solutions to the problem.
•A way to create a population (an initial set of potential solutions).
•An evaluation function rating solutions in terms of their fitness.
•Genetic operators that alter the genetic composition of offspring (selection, crossover, mutation, etc.).
•Parameter values that genetic algorithm uses (population size, probabilities of applying genetic operators, etc.).

Procedure of Simple GA

```
input: GA parameters
output: best solution
begin
  t ← 0;                                    // t: generation number
  initialize P(t) by encoding routine;      // P(t): population of chromosomes
  fitness eval(P) by decoding routine;
  while (not termination condition) do
      crossover P(t) to yield C(t);         // C(t): offspring
      mutation P(t) to yield C(t);
      fitness eval(C) by decoding routine;
      select P(t+1) from P(t) and C(t);
      t ← t+1;
  end
  output best solution;
end
```

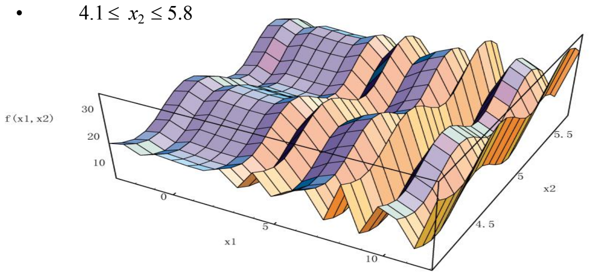
3-3 Example with Simple GA

- We explain in detail about how a genetic algorithm actually works with a simple examples.
- The numerical example of unconstrained optimization problem is given as follows:

•max $f(x_1, x_2) = 21.5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(20\pi x_2)$
•s. t. $-3.0 \le x_1 \le 12.1$
• $4.1 \le x_2 \le 5.8$

Example with Simple Genetic Algorithms

•max $f(x_1, x_2) = 21.5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(20\pi x_2)$
•s. t. $-3.0 \le x_1 \le 12.1$
• $4.1 \le x_2 \le 5.8$



1 Representation

- **Binary String Representation**
  - The **domain of x_j** is $[a_j, b_j]$ and the required precision is four places after the decimal point.
  - The **precision requirement** implies that the range of domain of each variable should be divided into at least $(b_j - a_j)\times10^4$ size ranges.
  - The **required bits** (denoted with m_j) for a variable is calculated as follows:

$$2^{m_j-1} < (b_j - a_j)\times 10^4 \le 2^{m_j} - 1$$

  - The mapping from a binary string to a real number for variable x_j is completed as follows:

$$x_j = a_j + \mathrm{decimal}(substring_j)\times\frac{b_j - a_j}{2^{m_j}-1}$$

13

1 Representation

- **Binary String Encoding**
  - The **precision requirement** implies that the range of domain of each variable should be divided into at least $(b_j - a_j)\times10^4$ size ranges.
  - The **required bits** (denoted with m_j) for a variable is calculated as follows:

x_1 : $(12.1-(-3.0))\times 10{,}000 = 151{,}000$

$2^{17} < 151{,}000 \le 2^{18}-1$, $m_1 = 18$ bits

x_2 : $(5.8-4.1)\times 10{,}000 = 17{,}000$

$2^{14} < 17{,}000 \le 2^{15}-1$, $m_2 = 15$ bits

•precision requirement: $m = m_1 + m_2 = 18 + 15 = 33$ bits

$$v_j:\ \overbrace{\underbrace{000001010100101001}_{\substack{18\text{ bits}\\ x_1}}\ \underbrace{101111011111110}_{\substack{15\text{ bits}\\ x_2}}}^{33\text{ bits}}$$

14

1 Representation

- **Procedure of Binary String Encoding**

•**input:** domain of $x_j \in [a_j, b_j]$, (j=1,2)

•**output:** chromosome v

step 1: The **domain of x_j** is $[a_j, b_j]$ and the required precision is four places after the decimal point.

step 2: The **precision requirement** implies that the range of domain of each variable should be divided into at least $(b_j - a_j)\times10^4$ size ranges.

step 3: The **required bits** (denoted with m_j) for a variable is calculated as follows:

$$2^{m_j-1} < (b_j - a_j)\times 10^4 \le 2^{m_j} - 1$$

step 4: A chromosome v is randomly generated, which has the number of genes m, where m is sum of m_j (j=1,2).

15

1 Representation

- **Binary String Decoding**
  - The mapping from a binary string to a real number for variable x_j is completed as follows:

$$v_j:\ \overbrace{\underbrace{000001010100101001}_{\substack{18\text{ bits}\\ x_1}}\ \underbrace{101111011111110}_{\substack{15\text{ bits}\\ x_2}}}^{33\text{ bits}}$$

	Binary Number	Decimal Number
x_1	000001010100101001	5417
x_2	101111011111110	24318

$$x_j = a_j + \mathrm{decimal}(substring_j)\times\frac{b_j - a_j}{2^{m_j}-1}$$

$$x_1 = -3.0 + 5417 \times \frac{12.1-(-3.0)}{2^{18}-1} = -2.687069$$

$$x_2 = 4.1 + 24318 \times \frac{5.8-4.1}{2^{15}-1} = 5.361653$$

16

1 Representation

- **Procedure of Binary Sring Decoding**

•**input:** substring$_j$

•**output:** a real number x_j

step 1: Convert a substring (a binary string) to a decimal number.

step 2: The mapping for variable x_j is completed as follows:

$$x_j = a_j + \mathrm{decimal}(substring_j)\times\frac{b_j - a_j}{2^{m_j}-1}$$

作业

17

依据上例编写程序，给定任意N位的染色体，将其转换为解空间为[a, b]的实数。

- (1) 转化为整数

$$x = \sum_{i=1}^{N} a_i 2^{i-1}$$

- (2)转化为实数

$$x = a + \frac{m}{2^L - 1}\times(b-a)$$

```
double decoding(int chrom [])
{  int i;
   double x = 0;
 • for (i = 0; i < N; i++)
 •   {   x = x + pow(2, N - i - 1) * chrom[i];
 •   }
   • x = a + (x / (pow(2, N) - 1)
   •          * (b - a);
   return x;
}
```

2 Initial Population

- **Initial population is randomly generated as follows:**
  - •$\boldsymbol{v}_1$ = [000001010100101001101111011111110] = [x_1 x_2] = [-2.687969 5.361653]
  - •$\boldsymbol{v}_2$ = [001110101110011000000010101001000] = [x_1 x_2] = [0.474101 4.170144]
  - •$\boldsymbol{v}_3$ = [111000111000001000010101001000110] = [x_1 x_2] = [10.419457 4.661461]
  - •$\boldsymbol{v}_4$ = [100110110100101101000000010111001] = [x_1 x_2] = [6.159951 4.109598]
  - •$\boldsymbol{v}_5$ = [000010111101100010001110001101000] = [x_1 x_2] = [-2.301286 4.477282]
  - •$\boldsymbol{v}_6$ = [111110101011011000000010110011001] = [x_1 x_2] = [11.788084 4.174346]
  - •$\boldsymbol{v}_7$ = [110100010011111000100110011101101] = [x_1 x_2] = [9.342067 5.121702]
  - •$\boldsymbol{v}_8$ = [001011010100001100010110011001100] = [x_1 x_2] = [-0.330256 4.694977]
  - •$\boldsymbol{v}_9$ = [111110001011101100011101000111101] = [x_1 x_2] = [11.671267 4.873501]
  - •$\boldsymbol{v}_{10}$ = [111101001110101010000010101101010] = [x_1 x_2] = [11.446273 4.171908]

19

3 Evaluation

- **The process of evaluating the fitness of a chromosome consists of the following three steps:**

```
•  input: chromosome $v_k$, $k$=1, 2, ..., popSize
•  output: the fitness $eval(v_k)$
•  step 1: Convert the chromosome's genotype to its phenotype, i.e., convert binary
   string into relative real values $x_k$ =($x_{k1}$, $x_{k2}$),   $k$ = 1,2, ..., popSize.
•  step 2: Evaluate the objective function $f(\boldsymbol{x}_k)$,   $k$ = 1,2, ..., popSize.
•  step 3: Convert the value of objective function into fitness. For the maximization
   problem, the fitness is simply equal to the value of objective function:
       •  $eval(\boldsymbol{v}_k) = f(x_k)$,   $k$ = 1,2, ..., popSize.
```

$$eval(v_k) = f(x_i) \qquad (k = 1, 2, \cdots, popSize)$$

$$(i = 1, 2, \cdots, n)$$

$$\bullet f(x_1, x_2) = 21.5 + x_1 \cdot \sin(4\pi x_1) + x2 \cdot \sin(20\pi x_2)$$

•Example: (x_1=-2.687969, x_2= 5.361653)

•$eval(\boldsymbol{v}_1) = f(-2.687969, 5.361653) = 19.805119$

20

3 Evaluation

- **An evaluation function plays the role of the environment, and it rates chromosomes in terms of their fitness.**
- **The fitness function values of above chromosomes are as follows:**
  - •$eval(\boldsymbol{v}_1)$ = f(-2.687969, 5.361653) =19.805119
  - •$eval(\boldsymbol{v}_2)$ = f(0.474101, 4.170144) = 17.370896
  - •$eval(\boldsymbol{v}_3)$ = f(10.419457, 4.661461) = 9.590546
  - •$eval(\boldsymbol{v}_4)$ = f(6.159951, 4.109598) = 29.406122
  - •$eval(\boldsymbol{v}_5)$ = f(-2.301286, 4.477282) = 15.686091
  - •$eval(\boldsymbol{v}_6)$ = f(11.788084, 4.174346) = 11.900541
  - •$eval(\boldsymbol{v}_7)$ = f(9.342067, 5.121702) = 17.958717
  - •$eval(\boldsymbol{v}_8)$ = f(-0.330256, 4.694977) = 19.763190
  - •$eval(\boldsymbol{v}_9)$ = f(11.671267, 4.873501) = 26.401669
  - •$eval(\boldsymbol{v}_{10})$ = f(11.446273, 4.171908) = 10.252480
- **It is clear that chromosome $\boldsymbol{v}_4$ is the strongest one and that chromosome $\boldsymbol{v}_3$ is the weakest one.**

21

4 Genetic Operators

- **Selection:**
  - **In most practices, a roulette wheel approach is adopted as the selection procedure, which is one of the fitness-proportional selection and can select a new population with respect to the probability distribution based on fitness values.**
  - **The roulette wheel can be constructed with the following steps:**

```
•input: population $P(t-1)$, $C(t-1)$
•output: population $P(t)$, $C(t)$
•step 1: Calculate the total fitness for the population $F = \sum_{k=1}^{popSize} eval(\boldsymbol{v}_k)$
•step 2: Calculate selection probability $p_k$ for each chromosome $\boldsymbol{v}_k$
        $p_k = \frac{eval(\boldsymbol{v}_k)}{F}$,        $k = 1, 2, \ldots, popSize$
•step 3: Calculate cumulative probability $q_k$ for each chromosome $\boldsymbol{v}_k$
        $q_k = \sum_{j=1}^{k} p_j$,        $k = 1, 2, \ldots, popSize$
•step 4: Generate a random number $r$ from the range [0, 1].
step 5: If $r \le q_1$, then select the first chromosome $\boldsymbol{v}_1$; otherwise, select
the $k$th chromosome $\boldsymbol{v}_k$ ($2 \le k \le popSize$) such that $q_{k-1} < r \le q_k$.
```

22

Evaluation

- •$eval(\boldsymbol{v}_1)$ = f(-2.687969, 5.361653) =19.805119
- •$eval(\boldsymbol{v}_2)$ = f(0.474101, 4.170144) = 17.370896
- •$eval(\boldsymbol{v}_3)$ = f(10.419457, 4.661461) = 9.590546
- •$eval(\boldsymbol{v}_4)$ = f(6.159951, 4.109598) = 29.406122
- •$eval(\boldsymbol{v}_5)$ = f(-2.301286, 4.477282) = 15.686091
- •$eval(\boldsymbol{v}_6)$ = f(11.788084, 4.174346) = 11.900541
- •$eval(\boldsymbol{v}_7)$ = f(9.342067, 5.121702) = 17.958717
- •$eval(\boldsymbol{v}_8)$ = f(-0.330256, 4.694977) = 19.763190
- •$eval(\boldsymbol{v}_9)$ = f(11.671267, 4.873501) = 26.401669
- •$eval(\boldsymbol{v}_{10})$ = f(11.446273, 4.171908) = 10.252480

23

4 Genetic Operators

- **Illustration of Selection:**

input: population $P(t-1)$, $C(t-1)$

output: population $P(t)$, $C(t)$

step 1: Calculate the total fitness F for the population.

$$F = \sum_{k=1}^{10} eval(\boldsymbol{v}_k) = 178.135372$$

step 2: Calculate selection probability p_k for each chromosome $\boldsymbol{v}_k$.

$$p_1 = 0.111180,\ p_2 = 0.097515,\ p_3 = 0.053839,\ p_4 = 0.165077,$$
$$p_5 = 0.088057,\ p_6 = 0.066806,\ p_7 = 0.100815,\ p_8 = 0.110945,$$
$$p_9 = 0.148211,\ p_{10} = 0.057554$$

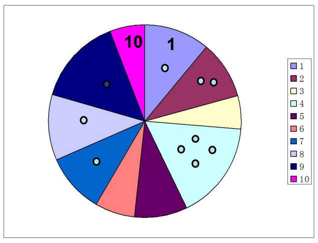
step 3: Calculate cumulative probability q_k for each chromosome $\boldsymbol{v}_k$.

$$q_1 = 0.111180,\ q_2 = 0.208695,\ q_3 = 0.262534,\ q_4 = 0.427611,$$
$$q_5 = 0.515668,\ q_6 = 0.582475,\ q_7 = 0.683290,\ q_8 = 0.794234,$$
$$q_9 = 0.942446,\ q_{10} = 1.000000$$

step 4: Generate a random number r from the range [0,1].

0.301431, 0.322062, 0.766503, 0.881893, 0.350871,
0.583392, 0.177618, 0.343242, 0.032685, 0.197577

$q_1 = 0.111180,\ q_2 = 0.208695,\ q_3 = 0.262534,\ q_4 = 0.427611,$
$q_5 = 0.515668,\ q_6 = 0.582475,\ q_7 = 0.683290,\ q_8 = 0.794234,$
$q_9 = 0.942446,\ q_{10} = 1.000000$



0.301431, 0.322062, 0.766503, 0.881893, 0.350871,
0.583392, 0.177618, 0.343242, 0.032685, 0.197577

25

4 Genetic Operators

- **Illustration of Selection:**

•step 5: $q_3 < r_1 = 0.301432 \le q_4$, it means that the chromosome $\boldsymbol{v}_4$ is selected for new population; $q_3 < r_2 = 0.322062 \le q_4$, it means that the chromosome $\boldsymbol{v}_4$ is selected again, and so on. Finally, the new population consists of the following chromosome.

•$\boldsymbol{v}_1'$ = [100110110100101101000000010111001] ($\boldsymbol{v}_4$)
•$\boldsymbol{v}_2'$ = [100110110100101101000000010111001] ($\boldsymbol{v}_4$)
•$\boldsymbol{v}_3'$ = [001011010100001100010110011001100] ($\boldsymbol{v}_8$)
•$\boldsymbol{v}_4'$ = [111110001011101100011101000111101] ($\boldsymbol{v}_9$)
•$\boldsymbol{v}_5'$ = [100110110100101101000000010111001] ($\boldsymbol{v}_4$)
•$\boldsymbol{v}_6'$ = [110100010011111000100110011101101] ($\boldsymbol{v}_7$)
•$\boldsymbol{v}_7'$ = [001110101110011000000010101001000] ($\boldsymbol{v}_2$)
•$\boldsymbol{v}_8'$ = [100110110100101101000000010111001] ($\boldsymbol{v}_4$)
•$\boldsymbol{v}_9'$ = [000001010100101001101111011111110] ($\boldsymbol{v}_1$)
•$\boldsymbol{v}_{10}'$ = [001110101110011000000010101001000] ($\boldsymbol{v}_2$)

26

```
#include "stdio.h"
#include "stdlib.h"
#include "time.h"
#define N 4
double prob[N] = {0.14, 0.49, 0.06, 0.31};
void main()
{  int i, j;
   srand((unsigned int)time(0));
   double sum_prob[N];
   sum_prob[0] = prob[0];
   for (i = 1; i < N; i++)
      sum_prob[i] = sum_prob[i-1] + prob[i];
   for (i = 0; i < N; i++)
   {   double x = (double)rand() / RAND_MAX;
      for (j = 0; j < N; j++)
         if (x < sum_prob[j])
         {   printf("%d selected!\n", j+1);
             break; }
   }
```

•计算累计概率

•生成随机数，选择个体

•搜索累计概率空间，确定个体

4 Genetic Operators

- **Crossover (One-cut point Crossover)**
  - **Crossover used here is one-cut point method, which random selects one cut point**
  - **Exchanges the right parts of two parents to generate offspring.**
  - **Consider two chromosomes as follow and the cut point is randomly selected after the 17th gene:**

•crossing point at 17th gene

•$\boldsymbol{v}_1$ = [100110110100101101000000010111001]

•$\boldsymbol{v}_2$ = [001110101110011000000010101001000]

$\Downarrow$

•$\boldsymbol{c}_1$ = [100110110100101100000010101001000]

•$\boldsymbol{c}_2$ = [001110101110011001000000010111001]

28

4 Genetic Operators

- **Procedure of One-cut Point Crossover:**

```
•procedure: One-cut Point Crossover
•input: $p_C$, parent $P_k$, $k$=1, 2, ..., $popSize$
•output: offspring $C_k$
•begin
•    for $k \leftarrow 1$ to $\lceil popSize/2 \rceil$ do        // $popSize$: population size
•       if $p_c \ge$ random [0, 1] then      // $p_C$: the probability of crossover
•          $i \leftarrow 0$;
•          $j \leftarrow 0$;
•          repeat
•             $i \leftarrow$ random [1, $popSize$];
•             $j \leftarrow$ random [1, $popSize$];
•          until ($i \ne j$ )
•          $p \leftarrow$ random [1, $l$ -1];        // $p$: the cut position, $l$: the length of chromosome
•          $C_i \leftarrow P_i$ [1: $p$-1] // $P_j$ [$p$: $l$ ];
•          $C_j \leftarrow P_j$ [1: $p$-1] // $P_i$ [$p$: $l$ ];
•       end
•    end
•    output offspring $C_k$;
•end
```

29

4 Genetic Operators

- **Mutation**
  - **<u>Alters one or more genes</u> with a probability equal to the mutation rate.**
  - **Assume that the 16th gene of the chromosome $\boldsymbol{v}_1$ is selected for a mutation.**
  - **Since the gene is 1, it would be flipped into 0. So the chromosome after mutation would be:**

•mutating point at 16th gene

•$\boldsymbol{v}_1$ = [100110110100101101000000010111001]

$\Downarrow$

•$\boldsymbol{c}_1$ = [100110110100101000000010101001000]

30

Example with Simple Genetic Algorithms

```
•procedure: Mutation
•input: $p_M$, parent $P_k$, $k$=1, 2, ..., popSize
•output: offspring $C_k$
•begin
•    for $k \leftarrow$ 1 to popSize do                        // popSize: population size
•        for $j \leftarrow$ 1 to $l$ do                        // l: the length of chromosome
•            if $p_M \geq$ random [0, 1] then             // $p_M$: the probability of mutation
•                $p \leftarrow$ random [1, $l$ -1];            // p: the cut position
•                $C_k \leftarrow P_k$ [1: $j$-1] // $P_k$ [ $j$ ] // $P_k$[ $j$+1: $l$ ];
•            end
•        end                                                    改变
•    end
•    output offspring $C_k$;
•end
```

- **Illustration of Mutation:**
  - Assume that $p_M = 0.01$

bitPos	*chromNum*	*bitNo*	*randomNum*
105	4	6	0.009857
164	5	32	0.003113
199	7	1	0.000946
329	10	32	0.001282

31

Example with Simple Genetic Algorithms

- **Next Generation**
  - v_1' = [100110110100101101000000010111001], f(6.159951, 4.109598) = 29.406122
  - v_2' = [100110110100101101000000010111001], f(6.159951, 4.109598) = 29.406122
  - v_3' = [001011010100001100010110011001100], f(-0.330256, 4.694977) = 19.763190
  - v_4' = [111111001011101100011101000111101], f(11.907206, 4.873501) = 5.702781
  - v_5' = [100110110100101101000000001011111], f(8.024130, 4.170248) = 19.91025
  - v_6' = [110100010011111000100110011101101], f(9.34067, 5.121702) = 17.958717
  - v_7' = [100110110100101101000000010111001], f(6.159951, 4.109598) = 29.406122
  - v_8' = [100110110100101101000000010111001], f(6.159951, 4.109598) = 29.406122
  - v_9' = [000001010100101001101111011111110], f(-2.687969, 5.361653) = 19.805199
  - v_{10}'= [001110101110011000000001010100110], f(0.474101, 4.170248) = 17.370896

32

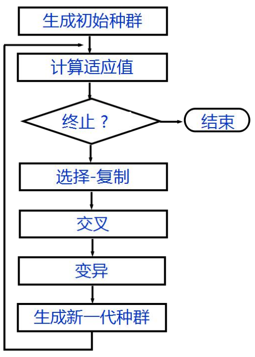
Example with Simple Genetic Algorithms

- **Procedure of GA for Unconstrained Optimization**

```
•procedure: GA for Unconstrained Optimization (uO)
•input: uO data set, GA parameters
•output: best solution
•begin
•    t ← 0;
•    initialize P(t) by binary string encoding;
•    fitness eval(P) by binary string decoding;
•    while (not termination condition) do
•        crossover P(t) to yield C(t) by one-cut point crossover;
•        mutation P(t) to yield C(t);
•        fitness eval(C) by binary string decoding;
•        select P(t+1) from P(t) and C(t) by roulette wheel selection;
•        t ← t+1;
•    end
•    output best solution;
•end
```

33

遗传算法基本流程框图



Example with Simple Genetic Algorithms

- **Final Result**
  - **The test run is terminated after 1000 generations.**
  - **We obtained the best chromosome in the 884th generation as follows:**

$$\max\ f(x_1, x_2) = 21.5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(20\pi x_2)$$

$$\text{s. t.}\quad -3.0 \leq x_1 \leq 12.1$$

$$4.1 \leq x_2 \leq 5.8$$

$$eval(v^*) = f(11.622766,\ 5.624329) = 38.737524$$

$$x_1^* = 11.622766$$

$$x_2^* = 5.624329$$

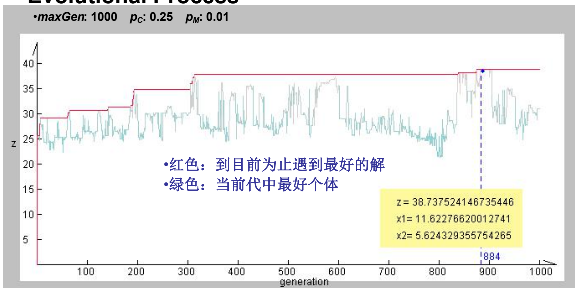
$$f(x_1^*, x_2^*) = 38.737524$$

35

Example with Simple Genetic Algorithms

- **Evolutional Process**

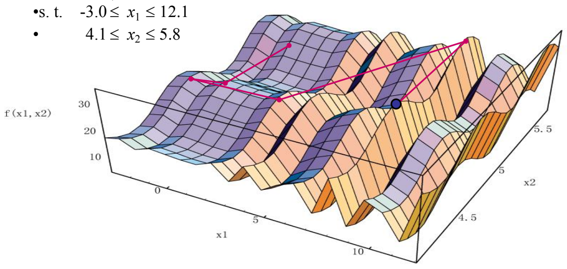
maxGen: 1000 p_C: 0.25 p_M: 0.01



36

Example with Simple Genetic Algorithms

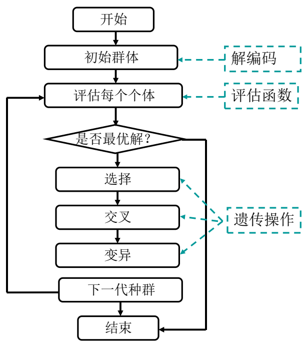
- **Evolutionary Process**
  - •max $f(x_1,x_2)=21.5+x_1\cdot\sin(4\pi x_1)+x_2\cdot\sin(20\pi x_2)$
  - •s. t. $-3.0\le x_1\le 12.1$
  - • $4.1\le x_2\le 5.8$



37

3-4 Implementation of GA

- 标准的遗传算法是具有“生成 + 测试”（generate-and-test）迭代过程的搜索算法。
- 遗传算法通过迭代进化，从一群初始解找到所期望的解。



1. Encoding

- **The classical representation scheme for GAs is binary vectors of fixed length.**
  - **In the case of an n_x-dimensional search space, each individual consist of n_x variables of each encoded as a bit string.**
  - **If variables have 0/1, the length of each chromosome is n_x bits.**
  - **In the case of nomimal-valued（标量值）variables, each nominal value can be encoded as an n_d-dimensonal bit vector, where 2^{n_d} is the total number of nominal values for that variable.**

- 如果问题解空间是整数，其编码需要两步确定
1. 确定编码长度。
- 如果问题解空间范围为x，那么编码长度为
- $l=\lceil \log_2 x\rceil$
2. 确定编码:将整数转换为二进制表示形式
- 方法：除2取余法。
- 例如：$9=(1001)_2$

- 如果所求解空间包含负值。
- 如果解空间为整数，可使用添加符号位的方法解决，即正数符号位为0，负数为1
- $+9=(01001)_2$
- $-9=(11001)_2$

Binary encoding

- **To solve optimization problems with continuous-values variables, the continuous search space problem can be mapped into a discrete programming problem.**
  - ● 如果所求解空间为实数空间？

  假设范围：［a，b］ 精度：小数点后4位

  1. 确定编码长度

  至少需要 $(b-a)\times 10^4$ 的空间存储

  码长： $L=\lceil log_2((b-a)\times 10^4)\rceil$

  2. 编码

  (1) 转换为整数 $m=[\frac{x-a}{b-a}\times(2^L-1)]$

  (2) 转换为二进制

例1　问题的解空间为[-3.0, 12.1]，求1.02编码后的结果，精度为小数点后4位。

1. 根据精度和解空间大小，确定编码长度

$$L = [log_2((b-a)\times 10^4)]$$
$$= [log_2((12.1+3.0)\times 10^4)]$$
$$= 18$$

•把一个浮点数转换为一个n_d位比特串，对于向量的每个元素，最大可以获得的精度为

$$\frac{x_{\max,j} - x_{\min,j}}{2^{n_d} - 1}，\ j = 1,\ldots,n_x$$

---

(1)转化为整数

$$m = [\frac{x-a}{b-a} \times (2^L - 1)]$$
$$= [\frac{1.02-(-3.0)}{12.1-(-3.0)} \times (2^{18} - 1)]$$
$$= 69789$$

(2)转化为二进制

$$69789 \xrightarrow{\quad} 10001000010011101$$

---

decoding

(1)转化为整数

假设 $chrom = a_N a_{N-1} a_{N-2} \ldots a_2 a_1$

转换公式：$x = \sum_{i=1}^{N} a_i 2^{i-1}$

1100100

$$x = \sum_{i=1}^{N} a_i 2^{i-1}$$
$$= 1 * 2^6 + 1 * 2^5 + 1 * 2^2$$
$$= 100$$

---

(2)转化为实数

$$x = a + \frac{m}{2^L - 1} \times (b-a)$$
$$= -3.0 + \frac{100}{2^{18} - 1} \times (12.1 - (-3.0))$$
$$= -2.9942$$

- 二进制编码优点

- **1.编解码简单易行**
- **2.遗传操作（交叉、变异）易于实现**
- **3.便于理论分析（模式定理）**

多参数映射编码

- 优化问题经常碰到带优化的参数不止一个的情况，因此需要采用多参数映射编码。
- 思想：现将每个参数进行二进制串编码，然后将这个子串生成一个完整的染色体。
- 例如：
- （2，5，6）---- 010 101 110
- 注：每个子串的码长可以不同。
- (2, 5, 100) ---- 010 101 1100100

二进制编码的不足

- While binary coding is frequently used, it has the disadvantage of introducing Hamming cliffs海明悬崖。
  - A Hamming cliff is formed when two numerically adjacent values have bit representations that are far apart.
- 例如，十进制数7和8，相应的二进制表示串（用4位比特串表示）为7＝0111和8＝1000，它们的海明距离为4。
- a small change in variables should result in a small change in fitness.
  - 例如，假设7为最优解，当前最优解为8，要能搜索到更优值7，则需要改变很多比特位。

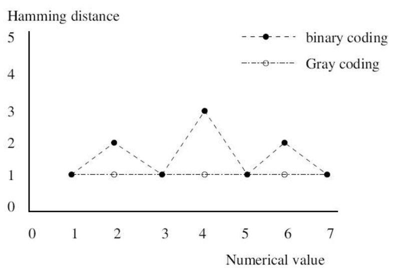
Gray Coding

- 格雷编码
  - 在该编码中，连续数值的海明距离为1。

	Binary	*Gray*
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

格雷编码

- Hamming distance



Conversion

- 二进制编码可以简单地转换为格雷编码：

$$g_1 = b_1$$
$$g_l = b_{l-1}\bar{b}_l + \bar{b}_{l-1}b_l$$

其中，b_l是二进制编码$b_1b_2\cdots b_{n_b}$的第l位，b_1是最高有效位（the most significant bit），$\bar{b}_l$表示非b_l，＋标似乎逻辑或，乘表示逻辑与。

2. Fitness function

- 适应值(fitness)就是借鉴生物个体对环境的适应程度,而对问题中的个体对象所设计的表征其优劣的一种测度。
- 适应值函数(fitness function)就是问题中的全体个体与其适应值之间的一个映射关系。它一般是一个实值函数。该函数就是遗传算法中指导搜索的评价函数。

- 原则
- 1、最优解对应最大的适应度
- 2、反映个体的优劣差异
- 3、计算量应该小
- 一般情况下由目标函数进行转换

构造方法

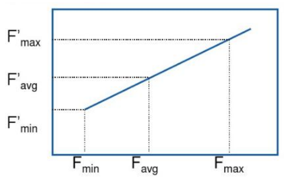
- 1、直接法
- 最大化问题 f(x)=F(x)
- 最小化问题 f(x)=-F(x)
- 2、界限构造
- 最大化问题 $f(x)=F(x)-C_{min}$
- 最小化问题 $f(x)=C_{max}-F(x)$
- 3、倒数构造
- 最大化问题 $f(x)=1/(1+C_{max}-F(x))$
- 最小化问题 $f(x)=1/(1+F(x)-C_{min})$

尺度变换

- 两个现象
- 1）个别个体适应度过高，能力太突出，影响选择，算法陷入局部最优。
- 2）算法后期适应度差别变小，算法停滞。
- 目的：
- 适当的放大缩小个体差别，提高算法性能
- 方法
- F'=a*F+b

Scaling

- 条件
- 1）变换前后平均适应度不变
- 2）变换后的最大适应度是变换前最大适应度的c倍



- Note：
- 1）可以根据条件1、2求出a和b
- 2）b>0，小于平均适应度的个体适应度将增大
- 大于平均适应度的个体适应度将减小
- 3）为预防出现负值，可以采用截断法。
- $aF_{min}+b=0$
- 4）每一代都需要重新计算a和b

Fitness scaling

– has a twofold intention(适应度变换有两个目的)
  - To maintain a reasonable differential between relative fitness ratings of chromosomes.(维持个体之间的合理差距，加速竞争)
  - To prevent a too-rapid takeover by some supper chromosomes in order to meet the requirement to limit competition early on, but to stimulate it later(避免个体之间的差距过大，限制竞争)

3.Selection

- The principle behind genetic algorithms is essentially Darwin natural selection.
- The selection directs GA search towards promising regions in the search space.
- During last few years, many selection methods have been proposed, examined, and compared.

Random selection

- Random selection is the simplest selection operator, where each individual has the same probability.

$$\frac{1}{n_s}, \quad n_s \text{ is the population size}$$

- No fitness information is used.
- The best and the worst indiduals have exactly the same probability of survising to the next generation.
- The lowest selective pressure.

Proportional selection

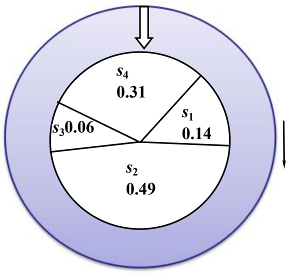
- Proportional selection, proposed by Holland, biases selection towards the most-fit individuals.
- A probability distribution proportional to the fitness is created.

$$\varphi_s(\mathbf{x}_i(t)) = \frac{f_\gamma(\mathbf{x}_i(t))}{\sum_{i=1}^{n_s} f_\gamma(\mathbf{x}_i(t))}$$

其中，n_s是种群中个体的总数，$\varphi_s(\mathbf{x}_i(t))$为个体$\mathbf{x}_i(t)$被选择的概率，$f_\gamma(\mathbf{x}_i(t))$是缩放后的$\mathbf{x}_i(t)$的适应度，为正浮点数。

Proportional selection

- Two popular sampling methods used in proportional selection is Roulette wheel sampling and stochastic universal sampling
- 轮盘赌和随机普遍采样。
  - Selection is directly proportional to fitness
  - Strong individual may dominate in producing offspring.
  - Limit the diversity of the new population.
  - High selective pressure

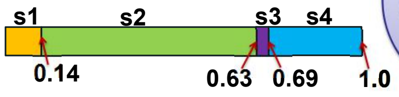


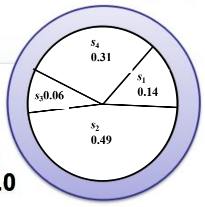
Roulette wheel selection

- Roulette wheel selection is an example proportional selection operator where fitness values are normalized(dividing each fitness by the maximum fitness value)
  - The probability distribution can then be seen as the roulette wheel, where the size of each slice is proportional to the normalized selection prabability of an individual.
  - Selection can be likened to spinning of a roulette wheel and recording which slice ends up at the top. the corresponding individual is then selected.

•程序实现

•为了方便分析，我们将轮盘区间展开





•随机产生0-1之间的随机数x。

x<0.14, 选择s1

0.14<=x<0.63 , 选择s2

0.63<=x<0.69 , 选择s3

0.69<=x , 选择s4

•步骤:

•1. 计算累计概率

•2. 生成随机数

•3. 根据范围选择

Roulette wheel selection

```
1. 令i=1，i标记染色体下标；
2. 计算$\varphi_s(x_i)$；
3. sum= $\varphi_s(x_i)$；
4. r~U(0, 1);
5. while   sum < r   do
6.     i=i + 1;  //前进到下一个染色体
7.     sum=sum+ $\varphi_s(x_i)$；
8. end
•  返回$x_i$作为选中的个体；
```

Roulette wheel selection

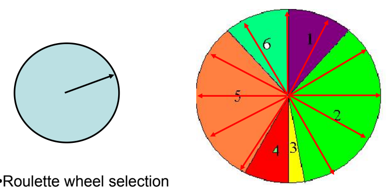
- When roulette wheel selection is used to create offspring to replace the retire population, n_s independent calls are needed. It was found that this results in a high variance in the number of offspring created by each individual.
  - It may happed that the best individual is not selected to produce offspring during a given generation.

Stochastic Universal Sampling

Baker proposed Stochastic Universal Sampling to prevent such problem.

Determine the number of offspring for each individual with only one call to the algorithm.

Roulette wheel selectionand Stochastic universal sampling



•区别在于：后者采用均匀分布的且个数等于种群规模的旋转指针这种方法的基本原则是保证每个染色体在下一代中复制的次数与期望值相差不大。

70

Stochastic universal sampling

```
1.  for   i=1,…,$n_s$   do
2.     $\lambda_i(t)$ = 0;  //每个个体被选择的次数的初值为0
3.  end
4.  r ~ U(0, 1 / $\lambda$);  // $\lambda$是子代个体的总数
5.  sum=0.0;
6.  for   i=1,…,$n_s$   do
7.     sum = sum + $\varphi_s(x_i)$；
8.     while   r < sum   do
9.        $\lambda_i$ + +;
10.       r = r + 1 / $\lambda$;
11.    end
12. end
13. 返回$\lambda = (\lambda_1, \lambda_2, …, \lambda_{ns})$；
```

锦标赛选择Tournament seleciton

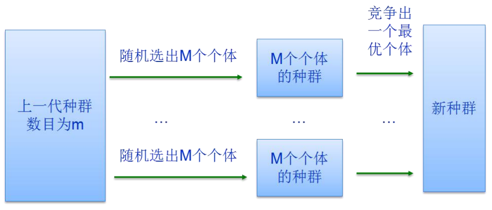
① select a group of n_{ts} individuals randomly from the population, n_{ts}<n_s.

② the performation of the selected n_{ts} individuals is compared and the best individuals from this group is selected and returned by this operator.

- While n_{ts} is not two large, the selection prevents the best individuals from dominating, thus having a lower selective pressure.
- If n_{ts} is too small, the chances increase that bad individuals are seleted.

Tournament seleciton

- Note that selective pressure is directly related to n_{ts}
  - if $n_{ts}=n_s$, the best individual will always be selected , resulting in a very high selective pressure.
  - if $n_{ts}=1$, random selection is obtained.



排序选择Rank-based selection

Rank-based seletion uses the rank ordering of fitness values to determine the probabilitu of selection, not the absolute fitness values.

- Selection is therefore independent of actual fitness values.
- The best individuals will not dominate in the selection process.

Rank-based selection

- Non-determinstic linear sampling select an individual $\mathbf{x}_i$

  $i \sim U(0, U(0, n_s-1))$,

  where the individuals are sorted in decreasing order of fitness value.

  It is assumed that the rank of the best individual is 0, the worst one is n_s-1。

排序选择

- 线性排序：

假设最优个体产生$\hat{\lambda}$个后代，最差个体产生$\tilde{\lambda}$个后代，其中

$1 \le \hat{\lambda} \le 2$，$\tilde{\lambda} = 2 - \hat{\lambda}$。

每个个体的选择概率由下式计算：

$$\varphi_s(\mathbf{x}_i(t)) = \frac{1}{n_s}\left(\tilde{\lambda} + \frac{f_\gamma(\mathbf{x}_i(t))}{n_s - 1}(\hat{\lambda} - \tilde{\lambda})\right)$$

其中，$f_\gamma(\mathbf{x}_i(t))$表示$\mathbf{x}_i(t)$的排列序号。

排序选择

- 非线性排序使用类似于下列公式的方法来计算选择概率：

$$\varphi_s(\mathbf{x}_i(t)) = \frac{1 - e^{-f_\gamma(\mathbf{x}_i(t))}}{\beta}$$

或

$$\varphi_s(\mathbf{x}_i(t)) = \nu(1-\nu)^{n_s - 1 - f_\gamma(\mathbf{x}_i(t))}$$

其中，$f_\gamma(\mathbf{x}_i(t))$是$\mathbf{x}_i(t)$的排列序号（即个体在排序队列中的位置），$\beta$是归一化常量，$\nu$表示选择下一个个体的概率。

- 排序选择算子可以用于任意采样方法中，如轮盘赌和随机普遍采样等。

Boltzmann selection

- 玻尔兹曼选择：**based on the thermodynamical principles of simulated annealing.**
- 玻尔兹曼选择有多种方式。
  – 其中一种用于计算选择概率的方式为：

$$\varphi_s(\mathbf{x}_i(t)) = \frac{1}{1+e^{\frac{f(\mathbf{x}_i(t))}{T(t)}}}$$

其中，$T(t)$为温度参数，可以使用一种温度调节方案把$T(t)$从初始的大数值减少到一个小的数值。

- 初始的大数值能保证每个个体都有相同的概率被选择。随着**T(t)**减小，选择会更关注于好的个体。

玻尔兹曼选择

- 玻尔兹曼选择可以用于两个个体间的选择。
- 例如，确定一个父个体$\mathbf{x}_i(t)$是否能被它的子个体$\mathbf{x}'_i(t)$所替代，如果

$$U(0,1) > \frac{1}{1+e^{\frac{f(\mathbf{x}_i(t))-f(\mathbf{x}'_i(t))}{T(t)}}}$$

则$\mathbf{x}'_i(t)$被选择，否则$\mathbf{x}_i(t)$被选择。

4. Crossover

- **Crossover operators can be devided into three main categories based on the number of parents used.**
  – 无性（**asexual**）：子代由一个父个体产生。
  – 有性（**sexual**）：由两个父个体产生一个或两个个体。
  – 多亲重组（**multi-recombination**）：由多于两个的父个体产生一个或多个个体。
- **based on the representation scheme used.**
  binary-specifc operator
  floating-point specific operator

Crossover

- **Parents are seleted using selection.**
- **Recombination is applied probabilistically.**
- **Each pair of parents have a probability** p_c **of producing offspring.**
  – p_c 交叉概率（**the crossover rate**）
  – **a high** p_c **is usually used**

Crossover

- **For parents selection:**
  ① **It may happen that the same individual is selected as both parents, in which case the generated offspring will be a copy of the parent(prevent)**
  ② **It is also possible that the same individual takes part in more than one application of crossover operation. (problem)**

Crossover

- **Replacement**

① **If one offspring is generated, the offspring may replace the worst parent based on that the offspring must be more fit that the worst parent.**
② **Boltzmann selection can be used to decide if the offspring should replace the worst parent.**

Binary representations

- Let $x_1(t)$ and $x_2(t)$ denote the two parents.

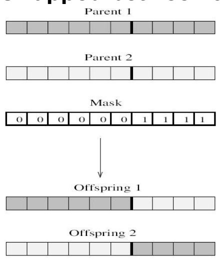
```
//位串交叉的遗传算法bitstring
1.  令$\tilde{y}_1(t) = x_1(t)$和$\tilde{y}_2(t) = x_2(t)$;
2.  if $U(0,1) \le p_c$ then
3.    计算二元掩码m(t); //binary mask，确定哪些位应交换
4.    for $j=1, …, n_x$ do
•       if $m_j=1$ then
•         $\tilde{y}_{1j}(t) = x_{2j}(t)$;  $\tilde{y}_{2j}(t) = x_{1j}(t)$;
•       end
•     end
•   end
```

Binary representations

- 不同的掩码计算方式，不同的交叉算子：
  - 单点交叉（One-point crossover）
  - 双点交叉（Two-point crossover）
  - 均匀交叉（Uniform crossover）

Binary representations

- one-point crossover: select a crossover point randomly and the bitstrings after that point are swapped between the two parents.



```
//单点交叉的掩码计算
mask computation
1.  select the crossover point $\xi \sim U(1, n_x-1)$ ;
2.  for $j=1, …, n_x$ do
3.    $m_j(t)=0$;
4.  end
5.  for $j=\xi+1, …, n_x$ do
6.    $m_j(t)=1$;
7.  end
```

•例　给定两个染色体解码后的值分别为13、8，使用4位编码，假设在第3位后进行单点交叉操作，结果分别为多少？

•(1) 转化为二进制

•13 ⟶ •1101　　•8 ⟶ •1000

•(2) 交叉操作

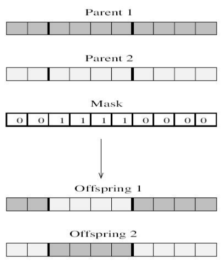
•110¦1　　　•110¦0
　　•交叉后 ⟶
•100¦0　　　•100¦1

•(3) 转为十进制

•1100 ⟶ •12　　•1001 ⟶ •9

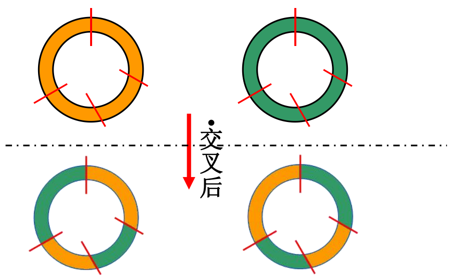
Binary representations

- Two-point crossover: select two bit positions randomly and the bitstring between the two points are swapped.



```
//两点交叉的掩码计算
1.  选择交叉点$\xi_1 \sim U(1, n_x-1)$ ;
2.  选择交叉点$\xi_2 \sim U(1, n_x-1)$ ;
3.  for $j=1, …, n_x$ do
4.    $m_j(t)=0$;
5.  end
//假设$\xi_1 \le \xi_2$
6.  for $j=\xi_1+1, …, \xi_2$ do
7.    $m_j(t)=1$;
8.  end
```

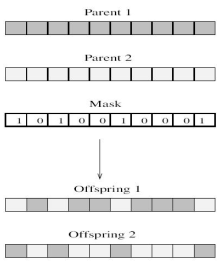
• 多点交叉



Binary representations

- **Uniform crossover:** n_x-dimensional mask is created randomly. p_x is the bit-swapping probability.

```
//均匀交叉的掩码计算
1. for j=1, …, $n_x$ do
2.     $m_j(t)=0$;
3. end
4. for j=$\xi_1$+1, …, $\xi_2$ do
5.     if $U(0,1) \le p_x$ then
6.         $m_j(t)=1$;
7.     end
8. end
```



Binary representations

- **Bremermann proposeda multi-parent crossover operators. Given n_μ parent vectors, generates one offspring using**

$$\tilde{\mathbf{x}}_{ij}(t)=\begin{cases}0 & \text{如果}n'_\mu \ge \dfrac{n_\mu}{2}, \quad l=1,\dots,n_\mu \\ 1 & \text{其它情况}\end{cases}$$

其中，n'_μ是$\mathbf{x}_{lj}(t)=0$的父个体的数目。

Binary representations

- **Jose developed a crossover hill-climbing operator**
  - 从两个父个体开始，不断产生子个体，直到达到最大的交叉次数或生成更好的子个体（至少有一个子个体更好）。
  - 交叉爬山不断使用新生成的子个体作为下次生成过程的父个体。
  - 如果在指定的时间内找不到更好的个体，则用随机个体替换较差的父个体。

Floating-point representation

- 线性算子（**the linear operator**），最早的浮点交叉算子，**Wright**提出
- 带方向启发的交叉算子（**a directional heuristic crossover operator**），**Wright**提出
- 算术交叉算子（**the arithmetic crossover operator**），**Michalewicz**提出
- 混合交叉（**BLX-α，the blend crossover operator**）算子，**Eshelman**和**Schaffer**提出
- 几何交叉算子（**the geometrical crossoveroperator**），**Michalewicz**等人提出
- 模拟二进制交叉（**SBX，the simulated binary crossover**），**Deb**和**Agrawal**提出

交叉—浮点表示

- 线性算子：通过父个体$\mathbf{x}_1(t)$和$\mathbf{x}_2(t)$产生三个备选子个体

$$\mathbf{x}_1(t)+\mathbf{x}_2(t),$$
$$1.5\mathbf{x}_1(t)-0.5\mathbf{x}_2(t),$$
$$-0.5\mathbf{x}_1(t)+1.5\mathbf{x}_2(t),$$

其中最好的两个解被选为子个体。

- 带方向启发的交叉算子：

$$\tilde{\mathbf{x}}_{ij}(t)=U(0,1)\left(\mathbf{x}_{2j}(t)-\mathbf{x}_{1j}(t)\right)+\mathbf{x}_{2j}(t)$$

其中，父个体$\mathbf{x}_2(t)$不比$\mathbf{x}_1(t)$差。

交叉—浮点表示

- 算术交叉算子：多父个体的重组策略，它根据权重从两个父个体获得信息。
  - 子个体生成公式为：

$$\tilde{x}_{ij}(t)=\sum_{l=1}^{n_\mu}\gamma_l x_{lj}(t), \quad 其中\sum_{l=1}^{n_\mu}\gamma_l=1。$$

当$n_\mu=2$时，可得算术交叉算子的一个特例：

$$\tilde{x}_{ij}(t)=(1-\gamma)x_{1j}(t)+\gamma x_{2j}(t), \quad 其中\gamma\in[0,1]。$$

若$\gamma=0.5$，则子个体为父个体的简单平均。

交叉—浮点表示

- 混合交叉（BLX-α）：

$$\tilde{x}_{ij}(t) = (1-\gamma_j)x_{1j}(t) + \gamma_j x_{2j}(t)$$

其中$\gamma_j = (1+2\alpha)U(0,1) - \alpha.$

- BLX-α算子随机生成的元素，其范围为：

假设$x_{1j}(t) < x_{2j}(t)$，

$$\left[x_{1j}(t) - \alpha\left(x_{2j}(t) - x_{1j}(t)\right),\ x_{1j}(t) + \alpha\left(x_{2j}(t) - x_{1j}(t)\right)\right]$$

- Eshelman与Schaffer的研究认为，α=0.5时，算子的效果最好。

- 在混合交叉算子中，子个体的位置依赖于父个体间的距离。如果距离较大，则子个体与父个体的距离也会比较大。

交叉—浮点表示

- 几何交叉算子：两个父个体几何交叉生成子个体。

$$\tilde{x}_{ij}(t) = \left(x_{1j}(t)x_{2j}(t)\right)^{0.5}$$

- 几何交叉算子可以泛化为多亲重组算子：

$$\tilde{x}_{ij}(t) = \left(x_{1j}^{\alpha_1} \cdot x_{2j}^{\alpha_2} \cdot \cdots \cdot x_{n_\mu j}^{\alpha_{n_\mu}}\right)$$

其中，n_μ为父个体数目，且$\sum_{l=1}^{n_\mu} \alpha_l = 1$。

交叉—浮点表示

- 模拟二进制交叉：模拟二进制表示的单点交叉。

$$\tilde{x}_{1j}(t) = 0.5 \times \left[(1+\gamma_j)x_{1j}(t) + (1-\gamma_j)x_{2j}(t)\right]$$
$$\tilde{x}_{2j}(t) = 0.5 \times \left[(1-\gamma_j)x_{1j}(t) + (1+\gamma_j)x_{2j}(t)\right]$$

其中，

$$\gamma_j = \begin{cases} (2u_j)^{\frac{1}{\eta+1}} & \text{如果} u_j \le 0.5 \\ \left(\dfrac{1}{2(1-u_j)}\right)^{\frac{1}{\eta+1}} & \text{其它情况} \end{cases}$$

且$u_j \in \mathrm{U}(0,1)$，$\eta > 0$为分布指数。Deb和Agrawal建议$\eta = 1$。

交叉—浮点表示

- 用于浮点表示的交叉算子（多亲算子）
  - 单模分布算子（UNDX，unimodal distributed operator），Ono与Kobayashi提出
  - 单纯形交叉算子（SPX，the simplex crossover operator）
  - 基因扫描技术
  - 对角交叉算子

- 多亲交叉算子的主要目标时加强探索能力。通过综合多个父个体的信息，子代和父代之间的相似性与双亲算子相比平均起来更小，获得了更多的破坏。

Mutation

- The aim of mutation is to introduce new genetic material into an existing individual:
  - add diversity to the genetic characteristics of the population.
  - support crossover .
- Mutation is applied at a certain probability p_m, produce the mutated offspring $\tilde{y}_i(t)$ for $\tilde{y}_i(t)$.
- Mutation probabolity,also referred to as the mutation rate. $p_m \in [0, 1]$ 。
  - p_m is usually a small value to ensure the good solutions are not distorted too much.

变异

给定每个基因以概率p_m变异，则个体$\tilde{\mathbf{x}}_i(t)$变异的概率为

$$\mathrm{Prob}\left(\tilde{\mathbf{x}}_i(t)\text{已变异}\right) = 1 - \left(1 - p_m\right)^{n_x},$$

其中，个体$\tilde{\mathbf{x}}_i(t)$包含n_x个基因。

假设采用二进制表示，如果$H\left(\tilde{\mathbf{x}}_i(t),\ \tilde{\mathbf{x}}_i'(t)\right)$是子代$\tilde{\mathbf{x}}_i(t)$和其变异体$\tilde{\mathbf{x}}_i'(t)$之间的海明距离，则变异体相似于原子代的概率为

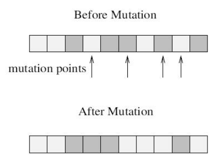
$$\mathrm{Prob}\left(\tilde{\mathbf{x}}_i'(t) \approx \tilde{\mathbf{x}}_i(t)\right) = p_m^{H\left(\tilde{\mathbf{x}}_i(t),\ \tilde{\mathbf{x}}_i'(t)\right)} \cdot \left(1 - p_m\right)^{n_x - H\left(\tilde{\mathbf{x}}_i(t),\ \tilde{\mathbf{x}}_i'(t)\right)}$$

变异—二进制表示

- 部分基于二进制表示的变异算子
  - 均匀变异，Uniform (random) mutation
  - 顺序变异，Inorder mutation
  - 高斯变异，Gaussian mutation

Binary representations

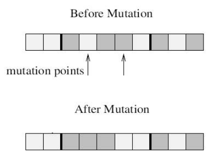
- **uniform mutation:** choose bit position randomly and the corresponding bit values negated.



```
//均匀（随机）变异
1. for j=1,…,$n_x$ do
2.    if U(0,1)≤$p_m$ do
3.        $x'_{ij}(t) = \neg\tilde{x}_{ij}(t)$;
4.    end
5. end
```

Binary representations

- **inorder mutation:** select two mutation points randomly and the bits between these mutation points undergo random mutation.



```
//顺序变异
1. 选择变异点$\xi_1$，$\xi_2$~U(1,…,$n_x$);
2. for j= $\xi_1$,…, $\xi_2$ do
3.    if U(0,1)≤$p_m$ do
4.        $x'_{ij}(t) = \neg\tilde{x}_{ij}(t)$;
5.    end
6. end
```

Binary representations

- **Gaussian mutation: convert back to floating-point representation and mutated with Gaussian noise.**
- 对每个染色体，从泊松分布（Poisson distribution）中产生一个随机数，决定该染色体要变异的基因的个数，随后转化表示这些基因的位串。
  - 对每个浮点值加上步长N(0, σ_j)，σ_j是相应浮点变量值域的0.1倍。
  - 最后，将变异后的浮点值转化回位串。
- Hinterding showed that Gaussian mutation on the floating-point representation provided superior results to bit flipping.

Binary representations

- For large dimensional bitstrings, mutation may significantly add to the computational cost of the GA.
- To reduce computational complexity, Birru divided the bitstring of each individual into a number of bins.
  - the mutation probability is applied to the bins
  - if a bin is to be mutated, one of its bits are randomly seleced and flipped.

Floating-point representation

- Uniform mutataion

$$x'_{ij}(t) = \begin{cases} \tilde{x}_{ij}(t) + \Delta(t, x_{\max,j} - \tilde{x}_{ij}(t)) & \text{如果随机值等于 } 0 \\ \tilde{x}_{ij}(t) + \Delta(t, \tilde{x}_{ij}(t) - x_{\min,j}) & \text{如果随机值等于 } 1 \end{cases}$$

其中，$\Delta(t,x)$返回$[0, x]$内的一个随机值。

Macromutation Operator

- 宏变异算子 Jonse
  - 又称为无头鸡算子（Headless Chicken）。
  - combine a parent individual with a randomly generated individual using crossover operation.
- the concept of inheritence does not exist.
- introduction of new randomly generated material.

用GA求解背包问题

- 0/1背包问题：已知n个物品和一个背包，每一个物品有一个价值v_i和一个重量w_i（i=1……n），而背包可以容纳总重量不超过C的物品。要求找出n个物体的一个子集，使所装入的物品的总价值最大，且物品的总重量不超过C，物品只能选择装还是不装。

$$\max \quad \sum_{i=1}^{n} v_i x_i$$
$$s.t. \quad \sum_{i=1}^{n} w_i x_i \le C,$$
$$x_i \in \{0,\ 1\},\ i \in \{1,\ldots,n\}.$$

用CGA求解背包问题

- 编码：

1	1	0	……	0	1	……	1
1	2	3		i	i+1		n

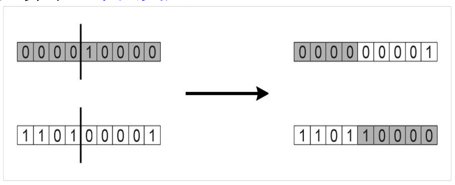
- 适应度函数：

$$f_0(x) = \begin{cases} \sum_{i=1}^{n} v_i x_i & \sum_{i=1}^{n} w_i x_i \le C \\ -\sum_{i=1}^{n} w_i x_i & \sum_{i=1}^{n} w_i x_i > C \end{cases}$$

$f(x) = f_0(x) - \min\{f(x);\ x \in C\}$，$C$为当前种群。

用CGA求解背包问题

- 交叉算子：单点交叉



- 变异算子：均匀变异，每一位以一定的概率变异。
  - Before: （1 1 1 1 0 0 1 0）
  - After: （1 1 0 0 0 1 1 0）

3-5 Control parameters

- 控制参数

– coding length	L 取决于精度
– population size	一般取20~100,或$2^{L/2}$
– crossover rate p_c	一般取0.4~0.99
– mutation rate p_m	一般取0.0001~0.1
– generation	T取100~1000

- 早期的GA：较低的p_m值，较高的p_c值。
  - 一般情况下，p_m和p_c维持不变。
- 普遍认为，p_m和p_c的最优设置可以极大地提高算法的性能。
  - 通过经验参数调整来寻找最优设置是非常耗时的过程。
  - 使用动态变化参数是寻找最优设置的一个办法。

Mutation rate

- dynamic, self-adjusting
- Forgarty
  - mutation rate exponentially decreases with generation number.

$$p_m(t) = \frac{1}{240} + \frac{0.11375}{2^t}$$

  - mutation rate per bit j=1,…,n_b n_b is the least significant bit.（Binary）

$$p_m(j) = \frac{0.3528}{2^{j-1}}$$

Mutation rate

- large mutation favorsexploration（探索）in the initial steps, and with a discrease in mutation rate as the genetation number increases, exploitation（开采）is facilitated.

- 可以用不同的方法来减少变异率。
  - 指数递减 exponential decrease
  - 线性递减 linear decrease

- A good strategy is to base the probality of being mutated on the fitness of the individual.
- the more fit individual is, the lower the probability that its genes will be mutated.
- the more unfit, the higher theprobability.

Crossover rate

- 交叉率也对性能有很大影响。
  - 最优值也依赖于问题。

- 对p_m的调整策略，也可用于调整p_c。

最佳进化算子

- 进化算子的选择也要根据不同问题来定。

- 经验表明，可以同时得到交叉、变异和选择的最优组合以及控制参数的最优值。

- 从根本上来说，寻找最优的算子集和控制参数值本身也是一个多目标优化问题。

小结

- 经典遗传算法
  - 基本操作及其实现
- 交叉
  - 各种交叉算子
- 变异
  - 各种变异算子
- 控制参数

作业3

1. 完成GA英文文献翻译。
2. 编写程序，给定任意N位的染色体转换为解空间为[a, b]的实数的编解码程序。
3. 编写赌轮法选择程序。