Part I Evolutionary Computation

Chapter 3
Genetic Algorithms

GA

- 3-1 History of GA
- 3-2 General Structure of GA
- 3-3 Example with Simple GA
- 3-4 Implentation of GA
- 3-5 Control parameters
- 3-6 GA variants and others
- 3-7 Applications of GA
- 3-8 Schema theorem

3-6 GA variants and Application

- Based on the general GA, different implementations of a GA can be obtained by using operators combinations of selection, crossover, and mutation operations.
- Although different operator combinations result in different bahaviors, the same algorithmic flow is used.

GA variants

- 1 Genaration Gap methods
- The GAs as discussed thus far differ from biological models of evolution in that population sizes are fixed. This allows the selection process to be described by:
  - Parent selection
  - A replacement strategy that decides if offspring will replace parents, and which parents to replace.

Gap

- Two main classes of GA based on the replacement strategy used.
  - 世代遗传算法（generational genetic algorithms GGAs）
  - 稳态遗传算法（steady state geneticalgorithms, SSGAs），又称为增量GAs（incremental GAs）

Gap

- GGA
  - replace all parents with their offspring after all offspring have been created and mutated.
  - no overlap between the current population and the new population.
- SSGA
  - a decision is made immediately after an offspring is created and mutated as to whether the parent or the offspring survives to the next generation.
  - overlap

- The amount of overlap between the current and new populations is referred to as the generation gap(代沟)
  - 世代遗传算法GGAs have a zero generation gap
  - 稳态遗传算法SSGAs generally have large generation gaps.

Replacement strategies

- Replace worst替换最差：offspring replaces the worst individual of the current population.
- Repalce random随机替换: offspring replaces a randomly selected individual of the current population.
- Kill tournement死亡锦标赛：a group of individuals is randomly selected, and the worst individual of the group is replaced with the offspring. Alternatively, a tournament size of two is used, the worst individual is replaced with a probability $0.5<p_r<1$.

Replacement stratigies

- Replace oldest替换最老：first-in-first-out, replace the oldest individual of the current population which has a high probability of replacing one of the best individuals.
- Conservative selection保守选择： combine the above two. one of the two individuals in tournament is always the oldest individuals of the current population, which ensure the oldest one will not be lost if it is the fittest.
- Elitist精英策略：the best individuals is excluded from selection.
- Parent-offspring competition父代-子代竞争：a selection strategy is used to decide if an offspring replaces one of its parents.

others

- 杂乱遗传算法
- 交互进化
- 岛屿遗传算法
- 小生境遗传算法

3-7 Applications of GA

- 遗传算法在人工智能的众多领域便得到了广泛应用。例如，机器学习、聚类、控制（如煤气管道控制）、规划（如生产任务规划）、设计（如通信网络设计、布局设计）、调度（如作业车间调度、机器调度、运输问题）、配置（机器配置、分配问题）、组合优化（如TSP、背包问题）、函数的最大值以及图像处理和信号处理等等。

- 1. Unconstraint Optimization(无约束优化)
- 2. Constrained Optimization(约束优化)
- 3. Combinatorial optimization（组合优化）

1.1 Introduction to Unconstraint Optimization

- For a real function of several real variables, we want to find an argument vector which corresponds to a minimal function value:
- The optimization problem:

  Find $x^* = \text{argmin}_x \{ f(x) | x_L \le x \le x_R \}$(最优解定义)

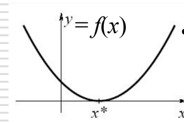
  where, the function f is called the objective function and x^* is the minimum
  - In some cases we want a maximum of a function. This is easily determined if we find a minimum of the function with opposite sign(极大值优化和极小值优化之间可以相互转化)

13

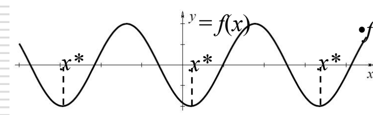
1.1 Introduction to Unconstraint Optimization

Example

- In this example we consider functions of one variable
- The function $f(x) = (x-x^*)^2$ has one unique minimum x^*.



- Periodical optimization: the function $f(x) = -2\cos(x-x^*)$ has many minimums.
  - $x = x^* + 2p\pi$, (最优点)   where p is an integer.

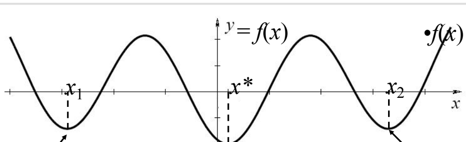


•$-4 \le x \le 4$

•14

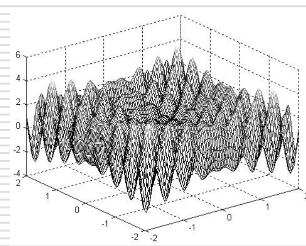
1.1 Introduction to Unconstraint Optimization

- **Multi-modal optimization**
  - the function $f(x) = 0.015(x-x^*)^2-2\cos(x-x^*)$ has a unique global minimum x^*. Besides that, it also has several so-called local minimums such as x_1 and x_2, each giving the minimal function value inside a certain region.



•15

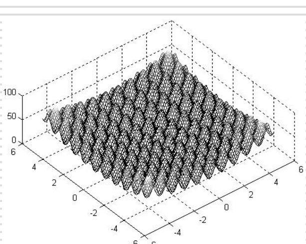
多峰函数



$$F(x) = \sum_{i=1}^{2} -(x_i * 250)\sin\sqrt{|x_i * 250|}$$

•16

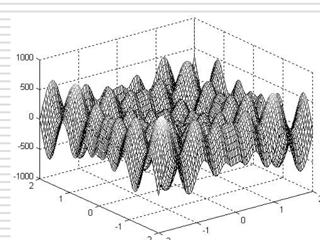
多峰函数



$$G(x,y) = 20 + x^2 - 10\cos 2\pi x + y^2 - 10\cos 2\pi y$$

•17

多峰函数



$$F(x,y) = x\sin(4\pi x) - y\sin(4\pi y) + 1$$

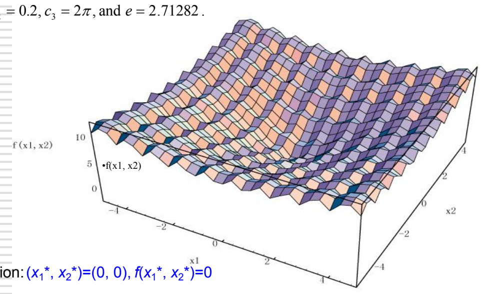
•18

•Ackley's Function

$$\min\ f(x_1, x_2) = -c_1 \cdot \exp\left(-c_2 \sqrt{\frac{1}{2}\sum_{j=1}^{2} x_j^2}\right) - \exp\left(\frac{1}{2}\sum_{j=1}^{2}\cos(c_3 \cdot x_j)\right) + c_1 + e$$

$$-5 \le x_j \le 5, \quad j = 1, 2$$

where $c_1 = 20, c_2 = 0.2, c_3 = 2\pi$, and $e = 2.71282$.



•Optimal solution: $(x_1^*, x_2^*)=(0, 0)$, $f(x_1^*, x_2^*)=0$

1.2 Genetic Approach for Ackley's Function

- To minimize Ackley's function, we simply use the following implementation of the Genetic Algorithm (GA):
  - Real Number Encoding
  - Arithmetic Crossover
  - Nonuniform Mutation
  - Top *popSize* Selection(最优确定选择)
- **Real Number Encoding**
  - $\mathbf{v} = [x_1, x_2, \ldots, x_n]$
  - x_i: real number, $i = 1, 2, \ldots, n$
- **Arithmetic Crossover**
  - The arithmetic crossover is defined as the combination of two chromosomes $\mathbf{v}_1$ and $\mathbf{v}_2$:

    $\mathbf{v}_1' = \mathbf{v}_1 + (1-\lambda)\mathbf{v}_2$

    $\mathbf{v}_2' = \mathbf{v}_2 + (1-\lambda)\mathbf{v}_1$

    where $\lambda \in (0, 1)$.

•20

1.2 Genetic Approach for Ackley's Function

- **Nonuniform Mutation**
  - For a given parent $\mathbf{v}$, if the element x_k of its selected for mutation.
  - The resulting: offspring is

    $\mathbf{v}' = [x_1, \ldots, x_k, \ldots, x_n]$

    where x_k is randomly selected from two possible choices:

    $x_1' = x_k + \rho(t, x_k^U - x_k)$ or

    $x_2' = x_k - \rho(t, x_k - x_k^L)$

    where x_k^U and x_k^L are the upper and lower bounds for x_k. The function $\rho(t, y)$ returns a value in the range $[0, y]$ such that the value of $\rho(t, y)$ approaches to 0 as t increases (t is the generation number) as follows:

$$\Delta(t, y) = y \cdot r \cdot \left(1 - \frac{t}{T}\right)^b$$

where r is a random number from [0, 1], T is the maximal generation number, and b is a parameter determining the degree of nonuniformity.

•21

1.2 Genetic Approach for Ackley's Function

- **Top *popSize* Selection**
  - Top *popSize* selection produces the next generationby selecting the best *popSize* chromosome from parents and offspring(确定排序最优选择)
  - For this case, we can simply use the values of objective as fitness values chromosomes according to these values.
- **Fitness Function**
  - $eval(\mathbf{v}) = f(\mathbf{x})$
- **Parameter setting of Genetic Algorithm (GA)**
  - The parameters of GA are set as follows:
    - Population size: *popSize* = 10
    - Crossover probability: p_C = 0.80
    - Mutation probability: p_M = 0.03
    - Maximum generation: *maxGen* =1000

•22

1.2 Genetic Approach for Ackley's Function

- GA for Unconstraint Optimization

```
procedure: GA for Unconstraint Optimization(uO)
input: uO data set, GA parameters
output: best solution
begin
    t ← 0;
    initialize P(t) by real number encoding;
    fitness eval(P);
    while (not termination condition) do
        crossover P(t) to yield C(t) by arithmetic crossover;
        mutation P(t) to yield C(t) by nonuniform mutation;
        fitness eval(C);
        select P(t+1) from P(t) and C(t) by top popSize selection;
        t ← t + 1;
    end
    output best solution;
end
```

•23

1.2 Genetic Approach for Ackley's Function

- After the 1000th generation, we have the following **chromosomes**:

	x_1	x_2
v_1 =	[-0.000002,	-0.000000]
v_2 =	[-0.000002,	-0.000000]
v_3 =	[-0.000002,	-0.000000]
v_4 =	[-0.000002,	-0.000000]
v_5 =	[-0.000002,	-0.000000]
v_6 =	[-0.000002,	-0.000000]
v_7 =	[-0.000002,	0.000000]
v_8 =	[-0.000002,	-0.000000]
v_9 =	[-0.000002,	-0.000000]
v_{10} =	[-0.000002,	-0.000000]

- The **fitness value**: $f(x_1^*, x_2^*) = -0.005456$

•24

1.2 Genetic Approach for Ackley's Function

- After the 1000th generation, we have the following **chromosomes**:

$$\begin{array}{rl} & \quad x_1 \qquad\qquad x_2 \\ v_1 = & [-0.000002,\ -0.000000] \\ v_2 = & [-0.000002,\ -0.000000] \\ v_3 = & [-0.000002,\ -0.000000] \\ v_4 = & [-0.000002,\ -0.000000] \\ v_5 = & [-0.000002,\ -0.000000] \\ v_6 = & [-0.000002,\ -0.000000] \\ v_7 = & [-0.000002,\ \ 0.000000] \\ v_8 = & [-0.000002,\ -0.000000] \\ v_9 = & [-0.000002,\ -0.000000] \\ v_{10} = & [-0.000002,\ -0.000000] \end{array}$$

- The **fitness value**: $f(x_1^*, x_2^*) = -0.005456$

2. Nonlinear Programming(约束优化)

- **Nonlinear programming** (or **constrained optimization**) deals with the problem of optimizing an objective function in the presence of equality and/or inequality constraints.
- Many practical problems can be successfully modeled as nonlinear program (NP). The **general NP model** may be written as follows:

$$\begin{aligned} \max \quad & f(\boldsymbol{x}) \\ \text{s. t.} \quad & g_i(\boldsymbol{x}) \le 0, \quad i = 1, 2, \ldots, m_1 \\ & h_i(\boldsymbol{x}) = 0, \quad i = m_1 + 1, \ldots, m_1 + m_2 \\ & \boldsymbol{x} \in \boldsymbol{X} \end{aligned}$$

where f is objective function, $g_i(\boldsymbol{x}) \le 0$ is inequality constraint, and each of the constraints $h_i(\boldsymbol{x}) = 0$ is equality constraint, the set $\boldsymbol{X}$ is domain constraint which include lower and upper bounds on the variables.

- The nonlinear programming problem is to find a **feasible point** $\boldsymbol{y}$ such that $f(\boldsymbol{y}) \ge f(\boldsymbol{x})$ for any feasible point $\boldsymbol{x}$.

Handling Constraints(约束处理算法)

- Several techniques have been proposed to handle constraints with Genetic Algorithms (GAs):
  - **Rejecting Strategy(抛弃策略)**
    - Rejecting strategy discards all infeasible chromosomes created throughout the evolutionary process.
  - **Repairing Strategy(修理策略)**
    - Repairing a chromosome involves taking an infeasible chromosome and generating a feasible one through some repairing procedure.
    - Repairing strategy depends on the existence of a deterministic repair procedure to convert an infeasible offspring into a feasible one.

- **Modifying Genetic Operator Strategy(修改遗传算子)**
  - One reasonable approach for dealing with the issue of feasibility is to invent problem-specific representation and specialized genetic operators to maintain the feasibility of chromosomes(设计针对问题的遗传算子和编码方法)
- **Penalty Strategy**(惩罚策略)
  - These strategies above have the advantage that they never generate infeasible solutions but have the disadvantage that they consider no points outside the feasible regions.（上述方法的缺点在于在遗传搜索过程这些不可行点均被抛弃或间接抛弃）

3.Traveling Salesman Problem

- **The Traveling Salesman Problem (TSP) is one of the most widely studied combinatorial optimization problems.**
- **Its statement is deceptively simple: A salesperson seeks the shortest tour through *n* cities.**

George Dantzig, Ray Fulkerson, and Selmer Johnson (1954)
a description of a method for solving the TSP :49 cities

•Table for non-directed graph

Traveling Salesman Problem

Existing Instances(实例)

- 49 city problem
  - **Dagtzig, G., D. Fulkerson and S. Johnson: "Solution of a large scale traveling salesman problems", *Operations Research*, vol. 2, pp. 393-410, 1954.**
- 120 city problem
  - **Grötschel, M.: "On the symmetric traveling salesman problem: solution of a 120 city problem", *Mathematical Programming Studies*, vol. 12, pp. 61-77, 1980.**
- 318 city problem
  - **Crowder, H. and M. Padberg "Solving large scale symmetric traveling salesman problems to optimality", *Management Science*, vol. 22, pp. 15-24, 1995.**
- 532 city problem
  - **Padberg, M. and G. Rinaldi "Optimization of 532 city symmetric traveling salesman problem by branch and cut", *Operations Research Letters*, vol. 6, pp. 1-7, 1987.**

Traveling Salesman Problem

- 666 city problem
  - **Grötschel, M. and O. Holland** "Solution of large scale symmetric traveling salesman problems", *Mathematical Programming Studies*, vol. 51, pp. 141-202, 1991.
- 2392 city problem
  - **Padberg, M. and G. Rinaldi** "A branch and cut algorithm for the resolution of large scale symmetric traveling salesman problem", *SIAM Review*, vol. 33, pp. 60-100, 1991.
- The earlier studies using the genetic algorithm to solve TSP
  - **Grefenstette, J.**: *Proceedings of the First International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.
- TSP has become a target for the genetic algorithm community
  - **Michalewicz, Z**: *Genetic Algorithm + Data structure = Evolution Programs*, 2nd ed.,

    Springer-Verlag, New York, 1994.

Traveling Salesman Problem

- **Notations(符号)**
  - **Indices**

    i, j : the index of city, i, j =1, 2, …, n
  - **Parameters**

    n: the total number of cities

    d_{ij} : the distance city i to city j, *i.e.*, the distance of route (i,j);
    the distance matrix (d_{ij}) is symmetric. (距离矩阵是对称的)
  - **Decision Variables**

    x_{ij} : the 0,1 decision variable; 1, if route (i,j) is selected, and 0, otherwise.

例如：四个城市X、Y、Z、W

城市名	访问顺序标示			
	1	2	3	4
X	0	1	0	0
Y	0	0	0	1
Z	1	0	0	0
W	0	0	1	0

路径

仅当每个城市只被访问一次

仅当每次只访问一个城市

Traveling Salesman Problem

Mathematical Model of TSP

$$\min \quad z = \sum_{i=1}^{n}\sum_{j=1}^{n}\left(d_{ij}x_{ij} + d_{n1}x_{n1}\right)$$

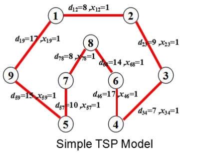
$$\text{s.t.} \quad \sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1,2,3,\cdots,n$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1,2,3,\cdots,n$$

$$x_{ij} = \begin{cases} 1, & \text{if route (i, j) is selected} \\ 0, & \text{otherwise} \end{cases}$$

data set for non-directed graph

i	j	d_{ij}	i	j	d_{ij}
1	2	8	3	7	7
	3	5		8	12
	4	9		9	12
	5	12	4	5	3
	6	14		6	17
	7	12		7	10
	8	16		8	7
	9	17		9	15
2	3	9	5	6	8
	4	15		7	10
	5	17		8	6
	6	8		9	15
	7	11	6	7	9
	8	18		8	14
	9	14		9	8
3	4	7	7	8	8
	5	9		9	6
	6	11	8	9	11



Simple TSP Model

定义适应值函数

- 我们将一个合法的城市序列s=（c_1, c_2, …, c_n）作为一个个体。这个序列中相邻两城之间的距离之和的倒数就可作为相应个体s的适应值，从而适应值函数就是

$$f(s) = \frac{1}{\sum_{i=1}^{n} d(c_i, c_{i+1})}$$

（c_{n+1}就是c_1）

1. Representation

- **Permutation Representation(置乱描述)**
  - This direct representation is perhaps the most natural representation of a TSP, where cities are listed in the order in which they are visited.

	1	2	3	4	5	6	7	8	9
chromosome	3	2	5	4	7	1	6	9	8

tour list 3 - 2 - 5 - 4 - 7 - 1 - 6 - 9 – 8

  - This representation is also called a path representation or order representation.

```
procedure: Permutation Encoding
Input: city set, total number of cities N
output: chromosome v
begin
  for j=1 to N
    v(j)← j;
  for i=1 to ⌈n/2⌉
    repeat
      j←random[1, N];
      l←random[1, N];
    until l≠j
    swap(v(j), v(l));
  output chromosome v;
end
```

交换

```
procedure: Permutation Decoding
Input: chromosome v,
       total number of cities N
output: tour list L
begin
  L← φ;
  for i =1 to N
    L ← L ∪v(i);
  output tour list L;
end
```

randperm

1 Representation

- **Random Keys Representation(随机数编码)**
  - This indirect representation encodes a solution with random numbers from (0,1).
  - These values are used as sort keys to decode the solution.

	1	2	3	4	5	6	7	8	9
chromosome	0.23	0.82	0.45	0.74	0.87	0.11	0.56	0.69	0.78

where position i in the list represents city i.

tour list   6 - 1 - 3 - 7 - 8 - 4 - 9 - 2 - 5

```
procedure: Random Keys Encoding
Input: city set,
      total number of cities N
output: chromosome v
begin
  for i =1 to N
    v[i] ← random[0,1];
  output chromosome v;
end
```

```
procedure: Random Keys Decoding
Input: chromosome v,
      total number of cities N
output: tour list L
begin
  L← ϕ;
  for i =1 to N
    L ← L ∪ i;
  sort L by v[i];
  output tour list L;
end
```

2 Crossover Operators(交叉算子)

- During the past decade, several crossover operators have been proposed for permutation representation, such as partial-mapped crossover (PMX), order crossover (OX), cycle crossover (CX), position-based crossover, order-based crossover, heuristic crossover, and so on.
- These operators can be classified into two classes:
  - Canonical approach
    - The canonical approach can be viewed as an extension of two-point or multipoint crossover of binary strings to permutation representation.(一般的方法可以看作两点交叉和多点交叉的推广)
  - Heuristic approach
    - The application of heuristics in crossover intends to generate an improved offspring.

2 Crossover Operators

1）**Partial-Mapped Crossover**(PMX 部分映射交叉)

```
procedure : PMX crossover
input : chromosome v1, v2,
        length of chromosome l
output : offspring v1', v2'
begin
    R ← ϕ;
    // step 1: select two positions
    at random
    s ← random[1:l-1] ;
    t ← random[s+1:l] ;
    // step 2: exchange two substrings
    v1' ← v1[1:s-1] // v2[s: t] // v1[t +1:l] ;
    v2' ← v2[1:s-1] // v1[s: t] // v2[t +1:l] ;
    // step 3: determine the mapping
               relationship
    R ← relation(v1[s:t] , v2[s:t]) ;
    // step 4: legalize offspring
    legalize( v1', v2' , R );
    output offspring v1' , v2' ;
end
```

step 1 : select two positions at random (s, t)

parent 1	1	2	3	4	5	6	7	8	9
parent 2	5	4	6	9	2	1	7	8	3

step 2: exchange two substrings

proto-child 1	1	2	6	9	2	1	7	8	9
proto-child 2	5	4	3	4	5	6	7	8	3

step 3 : determine the mapping relationship

6	9	2	1
3	4	5	6

1 ↔ 6 ↔ 3
2 ↔ 5
9 ↔ 4

step 4 : legalize offspring

offspring 1	3	5	6	9	2	1	7	8	4
offspring 2	2	9	3	4	5	6	7	8	1

v_1 : parent chromosome 1 v_2 : parent chromosome 2
l : length of chromosome v_1' : offspring chromosome 1
v_2' : offspring chromosome 2 R : relationships
s : start position of substring t : end position of substring
relation(v_1,v_2) : searching relationship between v_1 and v_2
legalize(v_1,v_2,R) change genes value of v_1, v_2 based on relationship R

2 Crossover Operators

2） **OX crossover**

```
procedure : Order Crossover (OX)
input :chromosome v1, v2,
       length of chromosome l
output : offspring
begin
    w ←1;
     // step 1: select substring at random
    s ← random[1: l-1] ;
    t ← random[s+1: l] ;
     // step 2: produce a proto-child by
              copying the substring
      ← v1[s: t] ;
     // step 3: produce offspring by filing
              unfixed positions of proto-
              child from parent 2
    for i=1 to s-1
        for j=w to l
           fg ←0;
           for k=s to t
              if v2[j] =v1[k] then
                 fg ← 1; break;
           if fg=0 then
              [i] ← v2[j] ;
              w ←j +1; break;
```

parent 1	1	2	3	4	5	6	7	8	9
proto-child			3	4	5	6			
offspring	7	9	3	4	5	6			
parent 2	5	7	4	9	1	3	6	2	8

v_1 : parent chromosome 1 v_2 : parent chromosome 2
l : length of chromosome v' : offspring chromosome
w: working data fg : flag
s : start position of substring t : end position of substring

offspring	7	9	3	4	5	6	1	2	8
parent 2	5	7	4	9	1	3	6	2	8

2 Crossover Operators

3） **Position-based Crossover**(PBX，基于位置交叉)

```
procedure : Position-based Crossover
input : chromosome v1, v2,
        length of chromosome l
output : offspring v'
begin
    T ← φ, S ← φ, w ← 1;
     // step 1: select a set of positions
                from parent 1 at random
    N ← random[1:l] ;
     // step 2: produce proto-child by
                copying values of
                selected positions
    for i=1 to N
        j ← random[1:l];
        v'[j] ← v1[j] ;
        T ← T∪j ;
        S ← S ∪ v1[j] ;
```

step 1 : select a set of positions from parent 1 at random (j)

parent 1	1	2	3	4	5	6	7	8	9

step 2 : produce proto-child by copying values of selected positions

proto-child		2			5	6			9

v_1 : parent chromosome 1 v_2 : parent chromosome 2
l : length of chromosome v' : offspring chromosome
N : total number of selected positions
T={t[j]} , j=1,2,..., N : selected positions set
S ={s[m]}, m=1,2,..., N : genes value set of selected positions
fg_1 : flag 1 fg_2 : flag 2
w : working data

2 Crossover Operators

```
// step 3: produce offspring by filing unfixed
           positions of proto-child from parent 2
    for i=1 to l
       fg1 ← 0;
       for j =1 to N
          if i=t[j] then fg1 ← 1;
       if fg1 =1 then continue;
       for k=w to l
          fg2 ← 0;
          for m=1 to N
             if v2[k]=s[m] then
               fg2 ← 1; break;
          if fg2 =0 then
            v'[i]← v2[k ];
            w ← k +1 ; break ;
  output offspring v';
end
```

step 3 : produce offspring by filing unfixed positions of proto-child from parent 2

offspring	4	2	3	1	5	6	7	8	9
parent 2	5	4	6	3	1	9	2	7	8

v_1 : parent chromosome 1 v_2 : parent chromosome 2
l : length of chromosome v' : offspring chromosome
N : total number of selected positions
T={t[j]} , j=1,2,..., N : selected positions set
S ={s[m]}, m=1,2,..., N : genes value set of selected positions
fg_1 : flag 1 fg_2 : flag 2
w : working data

2 Crossover Operators

4）**Order-Based Crossover**(OBX)

procedure : Order-Based Crossover
input : chromosome v_1, v_2,
length of chromosome l
output : offspring v'
begin
$S \leftarrow \varphi$, $T \leftarrow \varphi$, $w \leftarrow 1$;
// step 1: select a set of positions from parent 1 at random
$N \leftarrow$ random[1:l] ;
for i=1 **to** N
$j \leftarrow$ random[1:l];
$S \leftarrow S \cup v_1[j]$;

step 1 : select a set of positions from parent 1 at random

		j			j	j			j
parent 1	1	2	3	4	5	6	7	8	9

v_1 : parent chromosome 1　　v_2 : parent chromosome 2
l : length of chromosome　　v' : offspring chromosome
N : total number of selected positions
$T=\{t[j]\}$, j=1,2,…, N : positions set of assigned genes from v_2 to v'
$S=\{s[i]\}$, i=1,2,…, N : genes value set of selected positions
fg_1 : flag 1　　fg_2 : flag 2
w : working data

Crossover Operators

// step 2: produce proto-child by copying non-selected value from parent 2
for i=1 **to** l
$fg_1 \leftarrow 0$;
for j=1 **to** N
if $v_2[i] = s[j]$ **then**
$fg_1 \leftarrow 1$; break;
if fg_1=0 **then**
$v'[i] \leftarrow v_2[i]$,
$T \leftarrow T \cup i$;
// step 3: produce offspring by copying selected values from parent 1
for i=1 **to** l
$fg_2 \leftarrow 0$;
for j=1 **to** N
if $i = t[j]$ **then** $fg_2 \leftarrow 1$;
if fg_2=1 **then** continue;
$v'[i] \leftarrow s[w]$;
$w \leftarrow w+1$;
output offspring v';
end

step 2: produce proto-child by copying non-selected value from parent 2

parent 2	5	4	6	3	1	9	2	7	8
proto-child		4		3	1			7	8

step 3 : produce offspring by copying selected values from parent 1

parent 1	1	2	3	4	5	6	7	8	9
proto-child	2	4	5	3	1	6	9	7	8

v_1 : parent chromosome 1　　v_2 : parent chromosome 2
l : length of chromosome　　v' : offspring chromosome
N : total number of selected positions
$T=\{t[j]\}$, j=1,2,…, N: positions set of assigned genes from v_2 to v'
$S=\{s[i]\}$, i=1,2,…, N : genes value set of selected positions
fg_1 : flag 1　　fg_2 : flag 2
w : working data

2 Crossover Operators

5) **Cycle Crossover**(CX)

procedure : CX crossover
input : chromosome v_1, v_2,
length of chromosome l
output : offspring v'
begin
$S \leftarrow \varphi$, $T \leftarrow \varphi$, $w \leftarrow 1$;
// step 1: find the cycle between parents
$C \leftarrow$ cy (v_1, v_2);
// step 2: produce proto-child by copying gene values in cycle from parent 1
for i=1 **to** l
for j=1 **to** N-1
if $v_1[i] = c[j]$ **then**
$v'[i] \leftarrow v_1[i]$;
$T \leftarrow T \cup i$;
$S \leftarrow S \cup v_1[i]$;

step 1 : find the cycle between parents

parent 1	1	2	3	4	5	6	7	8	9
parent 2	5	4	6	9	2	3	7	8	1

cycle 1 → 5 → 2 → 4 → 9 → 1

step 2 : produce proto-child by copying gene values in cycle from parent 1

proto-child	1	2		4	5				9

v_1 : parent chromosome 1　　v_2 : parent chromosome 2
l : length of chromosome　　v' : offspring chromosome
N : total number of values in cycle
$T=\{t[n]\}$, n=1,2,…, N-1: positions set of S in proto-child
$S=\{s[k]\}$, k=1,2,…, N-1: proto-child genes value set in cycle
$C=\{c[j]\}$, j=1,2,…, N-1: value set of cycle
fg_1 : flag 1　　fg_2 : flag 2
w : working data
cy (v_1,v_2):finding the cycle which is defined by the corresponding positions between parents

2 Crossover Operators

// step 3: produce offspring by filling unfixed position from parent 2
for i=1 **to** l
$fg_1 \leftarrow 0$;
for n=1 **to** $|T|$
if $i=t[n]$ **then** $fg_1 \leftarrow 1$;
if fg_1 =1 **then** continue;
for $j=w$ **to** l
$fg_2 \leftarrow 0$;
for k=1 **to** $|S|$
if $v_2[j]=s[k]$ **then**
$fg_2 \leftarrow 1$; break;
if fg_2 =0 **then**
$v'[i] \leftarrow v_2[j]$;
$w \leftarrow j+1$; break;
output offspring v' ;
end

step 3 : produce offspring by filling unfixed position from parent 2

parent 2	5	4	6	9	2	3	7	8	1
offspring	1	2	6	4	5	3	7	8	9

v_1 : parent chromosome 1　　v_2 : parent chromosome 2
l : length of chromosome　　v' : offspring chromosome
N : total number of values in cycle
$T=\{t[n]\}$, n=1,2,…, N-1: positions set of S in proto-child
$S=\{s[k]\}$, k=1,2,…, N-1: proto-child genes value set in cycle
$C=\{c[j]\}$, j=1,2,…, N-1: value set of cycle
fg_1 : flag 1　　fg_2 : flag 2
w : working data
cycle(v_1,v_2): searching cycle between v_1 and v_2

3 Mutation Operators

1) **Inversion Mutation**（倒位变异）

procedure : Inversion Mutation
input : chromosome v_1, v_2,
length of chromosome l
output : offspring v'
begin
// step 1: select subtour at random
$s \leftarrow$ random[1:l-1] ;
$t \leftarrow$ random[s+1:l] ;
// step 2: produce offspring by copying inverse string of substring
$S \leftarrow$ invert(v[s: t]) ;
$v' \leftarrow v$[1:s-1] // S // v[t+1:l] ;
output offspring v' ;
end

step 1: select subtour at random

			s			t			
parent	1	2	3	4	5	6	7	8	9

step 2 : produce offspring by copying inverse string of substring

offspring	1	2	6	5	4	3	7	8	9

v : parent chromosome　　l : length of chromosome
v' : offspring chromosome　　s : start position of substring
t : end position of substring　　S : inverse string of substring
invert(*string*) : inversely changing order of *string*

3 Mutation Operators

2） **Insertion Mutation**（插入变异）

procedure : Insertion Mutation
input : chromosome v_1, v_2,
length of chromosome l
output : offspring v'
begin
// step 1 : select a position in parent 1 at random
$i \leftarrow$ random[1:l] ;
// step 2: insert selected value in randomly selected position parent 2
$j \leftarrow$ random[1:l-1] ;
$W \leftarrow v$[1:i-1] // v[i+1:l] ;
$v' \leftarrow W$[1:j-1] // v[i] // W[j:l-1];
output offspring v' ;
end

step 1 : select a position in parent 1 at random

						i			
parent	1	2	3	4	5	6	7	8	9

step 2: insert selected value in randomly selected position of parent 2

			j						
offspring	1	2	6	3	4	5	7	8	9

v : parent chromosome　　l : length of chromosome
v' : offspring chromosome　　i : selected position in parent 1
j : selected position in parent 2　　W : working data set

3 Mutation Operators

3) Displacement Mutation（移位变异）

procedure : Displacement Mutation
input : chromosome v_1, v_2,
length of chromosome l
output : offspring v'
begin
// step 1: select subtour
$s \leftarrow$ random[1:l-1] ;
$t \leftarrow$ random[s+1:l] ;
// step 2: insert subtour in a random position
$n \leftarrow t$-(s-1);
$i \leftarrow$ random[1:l-n] ;
$W \leftarrow v$[1:s-1] // v[t+1:l];
$v' \leftarrow W$[1:i-1] // v[s : t] // W[i:l-n] ;
output offspring v' ;
end

step 1 : select subtour

				s		t			
parent	1	2	3	4	5	6	7	8	9

step 2 : insert subtour in a random position

			i						
offspring	1	2	5	6	7	3	4	8	9

v : parent chromosome
v' : offspring chromosome
t : end position of substring
i : insert position
W : working data set
l : length of chromosome
s : start position of substring
n : length of subtour

3 Mutation Operators

4）Swap Mutation（互换变异）

procedure : Swap Mutation
input : chromosome v_1, v_2,
length of chromosome l
output : offspring v'
begin
// step 1: select two position at random
$i \leftarrow$ random[1:l -1] ;
$j \leftarrow$ random[i+1 :l] ;
// step 2: produce offspring by swapping selected positions
$v' \leftarrow v$[1: i -1] // v[j] // v[i+1 : j -1]// v[i] // v[j+1 : l];
output offspring v' ;
end

step 1: select two position at random

			i			j			
parent	1	2	3	4	5	6	7	8	9

step 2 : produce offspring by swapping selected positions

			j			i			
offspring	1	2	6	4	5	3	7	8	9

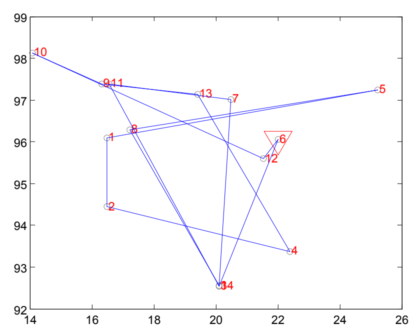
v : parent chromosome
v' : offspring chromosome
j : selected position
l : length of chromosome
i : selected position

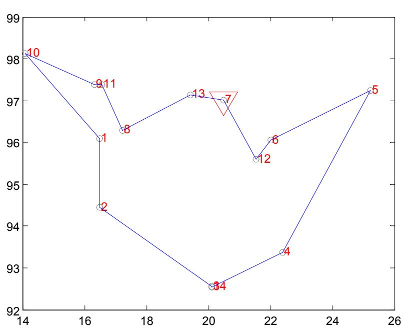
Overall Algorithm

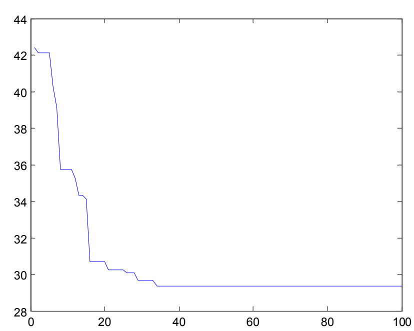
- **GA procedure for Traveling Salesperson Problem**

procedure: GA for Traveling Salesperson Problem (TSP)
Input: TSP data set, GA parameters
output: best tour route
begin
$t \leftarrow 0$;
initialize $P(t)$ by permutation encoding or random keys encoding
fitness $eval(P)$ by permutation decoding or random keys decoding
while (**not** termination condition) **do**
crossover $P(t)$ to yield $C(t)$ by partial-mapped crossover
mutation $P(t)$ to yield $C(t)$ by swap mutation
fitness $eval(C)$ by permutation decoding or random keys decoding
select $P(t+1)$ from $P(t)$ and $C(t)$;
$t \leftarrow t+1$;
end
output best tour route;
end

14个城市的TSP求解







- 初始种群一个随机解
- 6->3->11->7->14->8->5->1->2->4->13->9->10->12->6
- Rlength =
- 66.6070

- 最优解
- 7->12->6->5->4->3->14->2->1->10->9->11->8->13->7
- Rlength =
- 29.3405

Conclusion

- Combinatorial optimization problemsare characterized by afinite number of feasible solutionsand abounded in everyday life, particularly in various engineering design.
- One of the most challenging problemsin combinatorial optimization is to deal effectively with thecombinatorial explosion
- A major trend in solving such difficult problems isto use genetic algorithms.
- We explained how to solve the following Combinatorial optimization problems by genetic algorithms:
  - Knapsack Problem, - quadratic Assignment Problem - Minimum Spanning Tree, - Set-Covering Problem,
  - Bin-Packing Problem, - Traveling Salesman Problem.

3-8 模式理论

指导遗传算法的基本理论，是J.H.Holland教授创立的模式理论。该理论揭示了遗传算法的基本机理。

问题的引出

例：求 max $f(x)=x^2$ $x \in \{0,31\}$

表 1-1 遗传算法的第 0 代

个体编号	初始群体	x_i	适应度 $f(x_i)$	$f(x_i)/\Sigma f(x_i)$	$f(x_i)/\bar{f}$	下代个体数目
1	01101	13	169	0.14	0.58	1
2	11000	24	576	0.49	1.97	2
3	01000	8	64	0.06	0.22	0
4	10011	19	361	0.31	1.23	1

57

[分析]

- 当编码的最左边字符为“1”时，其个体适应度较大，如2号个体和4号个体，
  我们将其记为“1****”；
  其中2号个体适应度最大，其编码的左边两位都是1，我们记为“11***”；
- 当编码的最左边字符为“0”时，其个体适应度较小，如1号和3号个体，
  我们记为“0****”。

[结论]

从这个例子可以看比，我们在分析编码字符串时，常常只关心某一位或某几位字符，而对其他字符不关心。换句话讲．我们只关心字符的某些特定形式，如

1****，11***，0****。这种特定的形式就叫模式。

58

模式定理

模式：基于三值字符集{0, 1, *}所产生的能描述具有某些结构相似性的0，1字符串集的字符串称为模式。

•模式 *0100 描述：00100, 10100

•模式 **0*1 描述：

00001, 00011，01001，01011，10001，10011
11001, 11011

•模式 10001 描述：10001

- 长度为L的串隐含2^L个模式，规模为n的种群隐含着2^L ~ $n*2^L-(n-1)$个模式
- 例如串01具有2^2个模式 {01, 0*, 1*, **}
- 串110隐含着2^3个模式
- 群体{01，10}隐含着7个模式
- {01，0*，*1，**，10，1*，*0}

- 模式阶：
- 模式H中的确定位置的个数称为该模式的模式阶，记为O(H)
- 模式1*01**的阶数为 3
- 模式1*****的阶数为 1

- 显然，模式阶说明的是模式的确定程度，即模式阶数越高，模式的确定性越大，反之，确定性越小

- 模式的定义长度
- 模式H中第一个确定位置和最后一个确定位置之间的距离称为该模式的确定长度，记为δ(H)
- 模式1*01**的定义长度为 3
- 模式1*****的定义长度为 0

- 可见，模式定义长度描述的是模式中确定元素的紧凑程度

模式定理

- 假定在给定的时间步t,一个特定的模式s在群体P(t)中数目为m，记为m(s,t)。首先，我们暂不考虑交叉和变异操作。每个个体根据适应值的大小获得不同的复制概率。个体i的复制概率为：

$$p_i = \frac{f(i)}{\sum_{j=1}^{n} f(j)} \qquad (1)$$

模式定理

- 如果第i个个体在第t+1代被选择到的概率是pi，那么，第t+1代被选择到的数目为 $m_i = np_i$ ，因此第t+1代中含有模式s的个体数目为

$$m(s,t+1) = \sum_{i=1}^{m} m_i = n\sum_{i=1}^{m} p_i = n\sum_{i=1}^{m} \frac{f(i)}{\sum_{j=1}^{n} f(j)}$$

$$= \frac{m(s,t)n\sum_{i=1}^{m}\frac{f(i)}{m(s,t)}}{\sum_{j=1}^{n} f(j)} = m(s,t)n\frac{\overline{f}(s)}{\sum_{j=1}^{n} f(j)}$$

模式定理

- 则在群体P(t+1)中，模式s的代表个体的数量值为：

$$m(s,t+1) = m(s,t)\cdot n\cdot\frac{\overline{f}(s)}{\sum_{j=1}^{n} f(j)} \qquad (2)$$

- 其中，$\overline{f}(s)$表示在t时刻的所有含有模式s个体的适应值的均值，称为模式s的适应值。

模式定理

- 若记P(t)中所有个体的适应值的平均值为：

$$\overline{f} = \frac{\sum_{j=1}^{n} f(j)}{n}$$

- 则(2)式可以表示为：

$$m(s,t+1) = m(s,t)\cdot\frac{\overline{f}(s)}{\overline{f}} \qquad (3)$$

模式定理

- (3)式表明，模式s的代表串的数目随时间增长的幅度正比于模式s的适应值与群体平均适应值的比值。即：适应值高于群体平均值的模式在下一代的代表串数目将会增加，而适应值低于群体平均值的模式在下一代的代表串数目将会减少。
- 假设模式的适应值为$(1+c)\overline{f}$，其中c是一个常数，则(3)式可写为：

模式定理

$$m(s,t+1)=m(s,t)\cdot\frac{(1+c)\overline{f}}{\overline{f}}$$
$$=m(s,t)\cdot(1+c)=m(s,0)\cdot(1+c)^{t+1} \quad (4)$$

[结论]
在平均适应值之上（之下）的模式，将会按指数增长（衰减）的方式被复制。
选择算子可以保证优良个体按照指数规律遗传给后代。

模式定理

- 复制的结果并没有生成新的模式。因而，为了探索搜索空间中的未搜索部分，需要利用交叉和变异操作。
- 下面先探索交叉对模式的影响。
- 模式s1=“*1****0”和s2=“***10**”
- 交叉会改变模式的一部分，模式的长度越长，被破坏的概率越大。

这里以单点交叉算子为例研究。

[举例]

(1) 有两个模式 s_1: “ * 1 * * * * 0 ”
s_2: “ * * * 1 0 * * ”
它们有一个共同的可匹配的个体（可与模式匹配的个体称为模式的表示）
a: “ 0 1 1 1 0 0 0 ”

(2) 选择个体a 进行交叉

(3) 随机选择交叉点

s_1: “ * 1 | * | * | * | * | 0 ”　　交叉点选在第 2～6 之间都可能破坏模式s_1;

s_2: “ * * * 1 | 0 * * ”　　交叉点在 第 4～5之间才破坏s_2。

交换发生在模式s 的定义长度 δ(s)范围内，即模式被破坏的概率是：

$$p_d=\frac{\delta(s)}{l-1}$$

- 例：s_1 被破坏的概率为：5/6
- s_2 被破坏的概率为：1/6

70

模式定理

- 假定模式s在交叉后不被破坏的概率为p_s，则：

$$p_s \ge 1-\frac{\delta(s)}{l-1}$$

- 若交叉概率为p_c,则s不被破坏的概率为

$$p_s \ge 1-p_c\cdot\frac{\delta(s)}{l-1}$$

模式定理

- 所以，再考虑交叉时，(3)式可表示为

$$m(s,t+1)\ge m(s,t)\cdot\frac{\overline{f}(s)}{\overline{f}}\left[1-p_c\cdot\frac{\delta(s)}{l-1}\right] \quad (5)$$

[结论]
模式的定义长度对模式的存亡影响很大，模式的长度越大，越容易被破坏。模式长度越小，后代出现的次数越多。

模式定理

- 最后，考虑变异算子对模式的影响。变异算子以概率p_m随机地改变个体某一位的值。只有当o(s)个确定位的值不被破坏时，模式s才不被破坏。
- 模式s在变异后不被破坏的概率：

$$p_s = (1 - p_m)^{o(s)}$$

- P_m<<1,可近似地表示为

$$p_s \approx 1 - p_m \cdot o(s)$$

模式定理

- 因此，考虑交叉和变异时，(3)式可表示为

$$m(s,t+1) \geq m(s,t) \cdot \frac{\overline{f}(s)}{\overline{f}} \cdot \left[1 - p_c \cdot \frac{\delta(s)}{l-1} - o(s) \cdot p_m + p_c \cdot \frac{\delta(s)}{l-1} \cdot o(s) \cdot p_m\right]$$

$$\geq m(s,t) \cdot \frac{\overline{f}(s)}{\overline{f}} \cdot \left[1 - p_c \cdot \frac{\delta(s)}{l-1} - o(s) \cdot p_m\right] \quad (6)$$

模式定理

- 由(6)我们得到一个重要的定理。
- 模式定理(Schema Theorem)

  适应值在群体适应值之上的，长度较短的，低阶的模式在GA的迭代中将按指数增长方式被复制。

积木块假设

- Holland和Goldberg在模式定理的基础上提出了“积木块假设”(Building Block Hypothesis):
- 低阶、长度较短、高于平均适应度的模式(积木块)在遗传算子的作用下，相互结合，能生成高阶、长度较长、适应度较高的模式，并得到全局最优解。

模式对搜索空间的划分

[举例]

以 max $f(x)=x^2$ $x \in \{0,31\}$ 为例，

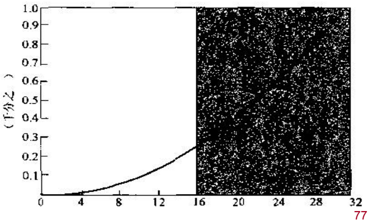
图中：横坐标表示x，

纵坐标代表适应度$f(x)=x^2$，用千分数表示，

弧线表示适应度曲线，

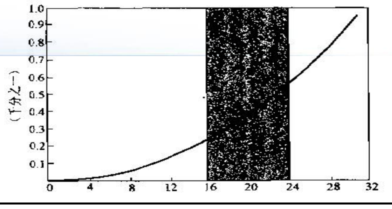
网点区代表所有符合此模式的个体集合。

模式1：1****

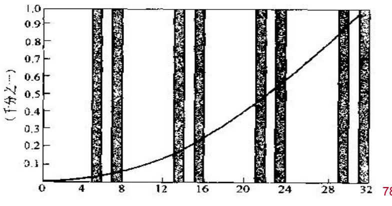


77

•模式2：10***



•模式3：**1*1



78

[结论]

模式能够划分搜索空间，而且模式的阶(已确定个数)越高，对搜索空间的划分越细致。

2 分配搜索次数

模式定理告诉我们：

GA根据模式的适应度、长度和阶次为模式分配搜索次数。
为那些适应度较高，长度较短，阶次较低的模式分配的搜索次数按指数率增长；
为那些适应度较低，长度较长，阶次较高的模式分配的搜索次数按指数率衰减。

3 建筑块假说

前面我们已经介绍了GA如何划分搜索空间和在各个子空间中分配搜索次数，
那么GA如何利用搜索过程中的积累信息加快搜索速度呢？

Holland 和 Goldberg在模式定理的基础上提出了“建筑块假说”。

79

[积木块(Buliding Block)]

——短定义长度、低阶、高适应度的模式。

之所以称之为建筑块（积木块），是由于遗传算法的求解过程并不是在搜索空间中逐一地测试各个基因的枚举组合，而是通过一些较好的模式，像搭积木一样、将它们拼接在一起，从而逐渐地构造出适应度越来越高的个体编码串。

[假说]

GA在搜索过程中将不同的“建筑块”通过遗传算子（如交叉算子）的作用结合在一起，形成适应度更高的新模式。这样将大大缩小GA的搜索范围。

80

[建筑块混合]

——建筑块通过遗传算子的作用集合在一起的过程称为“建筑块混合”。

当那些构成最优点（或近似最优点）的“建筑块”结合在一起时，就得到了最优点。

[建筑块混合的例子]

- 问题的最优用三个建筑块 BB_1, BB_2, BB_3 表示；
- 群体中有8个个体。
- 初始群体中个体1，个体2包含建筑块BB_1，个体3包含BB_3 ,个体5包含BB_2 。

	初始群体	第二代群体	第三代群体
P_1	BB_1	BB_2	BB_2
P_2	BB_1	BB_1	BB_1
P_3	BB_3	BB_1 BB_3	BB_1 BB_2 BB_3
P_4		BB_2	
P_5	BB_2	BB_3	BB_2 BB_3
P_6		BB_1	BB_1 BB_3
P_7		BB_3	BB_3
P_8		BB_1 BB_2	BB_1 BB_2

说明：
第三代群体中出现了一个包含三个“建筑块”的个体3。
个体3就代表这个问题的最优解。