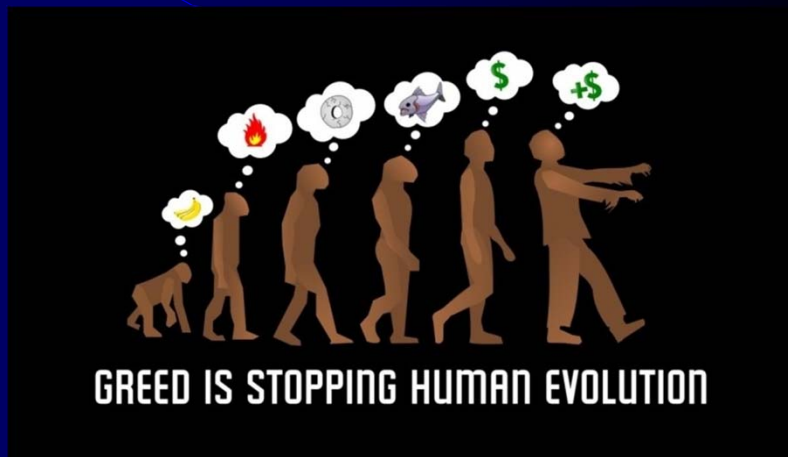
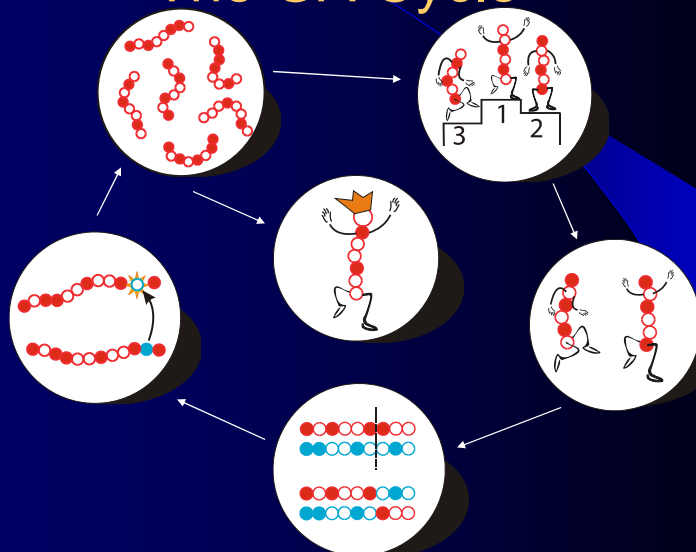


## GA Application and Issues

Lecture 4 and 5



### The GA Cycle



2

## Simple Problem

- The objective is to minimize the function

$$f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$


- In the interval  $0 \leq x_1, x_2 \leq 6$
- True solution is (3,2)

3

## Step 1

- Choose binary coding for  $x_1, x_2$
- 10-bits are chosen for each variable
- Total string length equal to 20.
- With 10-bits we can get a solution accuracy of  $(6-0)/(2^{10} - 1) = 0.006$  in the interval (0,6).
- Chose roulette-wheel selection
- Crossover probability = 0.8
- Mutation probability = 0.05
- Population size = 20
- $t_{max} = 30$

4



	String										Mating Pool			
	Substring 2	Substring 1	$x_2$	$x_1$	$f(x)$	$F(x)$	A	B	C	D	E	F	Substring 2	Substring 1
1	1110010000	1100100000	5.349	4.692	959.680	0.001	0.13	0.007	0.007	0.472	10	0	0010100100	1010101010
2	0001001101	0011100111	0.452	1.355	105.520	0.009	1.10	0.055	0.062	0.108	3	1	1010100001	0111001000
3	1010100001	0111001000	3.947	2.674	126.685	0.008	0.98	0.049	0.111	0.045	2	1	0001001101	0011100111
4	1001000110	1000010100	3.413	3.120	65.026	0.015	1.85	0.093	0.204	0.723	14	2	1110011011	0111000010
5	1100011000	1011100011	4.645	4.334	512.197	0.002	0.25	0.013	0.217	0.536	10	0	0010100100	1010101010
6	0011100101	0011111000	1.343	1.455	70.868	0.014	1.71	0.086	0.303	0.931	19	2	0011100010	1011000011
7	0101011011	0000000111	2.035	0.041	88.273	0.011	1.34	0.067	0.370	0.972	19	1	0011100010	1011000011
8	1110101000	1110101011	5.490	5.507	1436.563	0.001	0.12	0.006	0.376	0.817	17	0	0111000010	1011000110
9	1001111101	1011100111	3.736	4.358	265.556	0.004	0.49	0.025	0.401	0.363	7	1	0101011011	0000000111
10	0010100100	1010101010	0.962	4.000	39.849	0.024	2.96	0.148	0.549	0.189	4	3	1001000110	1000010100
11	1111101001	0001110100	5.871	0.680	814.117	0.001	0.14	0.007	0.556	0.220	6	0	0011100101	0011111000
12	0000111101	0110011101	0.358	2.422	42.598	0.023	2.84	0.142	0.698	0.288	6	3	0011100101	0011111000
13	0000111110	1110001101	0.364	5.331	318.746	0.003	0.36	0.018	0.716	0.615	12	1	0000111101	0110011101
14	1110011011	0111000010	5.413	2.639	624.164	0.002	0.24	0.012	0.728	0.712	13	1	0000111110	1110001101
15	1010111010	1010111000	4.094	4.082	286.800	0.003	0.37	0.019	0.747	0.607	12	0	0000111101	0110011101
16	0100011111	1100111000	1.683	4.833	197.556	0.005	0.61	0.030	0.777	0.192	4	0	1001000110	1000010100
17	0111000010	1011000110	2.639	4.164	97.699	0.010	1.22	0.060	0.837	0.386	9	1	1001111101	1011100111
18	1010010100	0100001001	3.871	1.554	113.201	0.009	1.09	0.054	0.891	0.872	18	1	1010010100	0100001001
19	0011100010	1011000011	1.326	4.147	57.753	0.017	2.08	0.103	0.994	0.589	12	2	0000111101	0110011101
20	1011100011	1110100000	4.334	5.724	987.955	0.001	0.13	0.006	1.000	0.413	10	0	0010100100	1010101010

A = Expected Count =  
 B = Probability Selection  
 C = Cumulative probability of selection  
 D = Random number between 0 and 1  
 E = String Number  
 F = True Count in the Mating pool


5

5

## Step 2

- Evaluate each string in the population
- First sub-string - 1100100000  
 $\rightarrow (2^9 + 2^8 + 2^5)$   
 $\rightarrow 800$   
 $\rightarrow 0 + (6-0) \times 800 / (1024-1) = 4.692$
- Second sub-string 1110010000  $\rightarrow 5.349$
- $f(x) = 959.680$
- Fitness function value at this value using the transformation rule:  $F(x) = 1.0 / (1.0 + 959.680) = 0.001$
- Calculate for other populations

6



String	Substring 2	Substring 1	$x_2$	$x_1$	$f(x)$	$F(x)$	A	B	C	D	E	F	Mating Pool	Substring 2	Substring 1
1	1110010000	1100100000	5.349	4.692	959.680	0.001	0.13	0.007	0.007	0.472	10	0	0010100100	1010101010	
2	0001001101	0011100111	0.452	1.355	105.520	0.009	1.10	0.055	0.062	0.108	3	1	1010100001	0111001000	
3	1010100001	0111001000	3.947	2.674	126.685	0.008	0.98	0.049	0.111	0.045	2	1	0001001101	0011100111	
4	1001000110	1000010100	3.413	3.120	65.026	0.015	1.85	0.093	0.204	0.723	14	2	1110011011	0111000010	
5	1100011000	1011100011	4.645	4.334	512.197	0.002	0.25	0.013	0.217	0.536	10	0	0010100100	1010101010	
6	0011100101	0011111000	1.343	1.455	70.868	0.014	1.71	0.086	0.303	0.931	19	2	0011100010	1011000011	
7	0101011011	0000000111	2.035	0.041	88.273	0.011	1.34	0.067	0.370	0.972	19	1	0011100010	1011000011	
8	1110101000	1110101011	5.490	5.507	1436.563	0.001	0.12	0.006	0.376	0.817	17	0	0111000010	1011000110	
9	1001111101	1011100111	3.736	4.358	265.556	0.004	0.49	0.025	0.401	0.363	7	1	0101011011	0000000111	
10	0010100100	1010101010	0.962	4.000	39.849	0.024	2.96	0.148	0.549	0.189	4	3	1001000110	1000010100	
11	1111101001	0001110100	5.871	0.680	814.117	0.001	0.14	0.007	0.556	0.220	6	0	0011100101	0011111000	
12	0000111101	0110011101	0.358	2.422	42.598	0.023	2.84	0.142	0.698	0.288	6	3	0011100101	0011111000	
13	0000111110	1110001101	0.364	5.331	318.746	0.003	0.36	0.018	0.716	0.615	12	1	0000111101	0110011101	
14	1110011011	0111000010	5.413	2.639	624.164	0.002	0.24	0.012	0.728	0.712	13	1	0000111110	1110001101	
15	1010111010	1010111000	4.094	4.082	286.800	0.003	0.37	0.019	0.747	0.607	12	0	0000111101	0110011101	
16	0100011111	1100111000	1.683	4.833	197.556	0.005	0.61	0.030	0.777	0.192	4	0	1001000110	1000010100	
17	0111000010	1011000110	2.639	4.164	97.699	0.010	1.22	0.060	0.837	0.386	9	1	1001111101	1011100111	
18	1010010100	0100001001	3.871	1.554	113.201	0.009	1.09	0.054	0.891	0.872	18	1	1010010100	0100001001	
19	0011100010	1011000011	1.326	4.147	57.753	0.017	2.08	0.103	0.994	0.589	12	2	0000111101	0110011101	
20	1011100011	1110100000	4.334	5.724	987.955	0.001	0.13	0.006	1.000	0.413	10	0	0010100100	1010101010	

A = Expected Count =  
B = Probability Selection  
C = Cumulative probability of selection  
D = Random number between 0 and 1  
E = String Number  
F = True Count in the Mating pool

7

7

## Step 3

- Since  $t < t_{max}$  We proceed to Step 4.

8

## Step 4

- Picking up good strings – Use roulette-wheel selection procedure.
- Calculate average fitness of the population

$$\underline{F} = \sum F_i / n = 0.008$$

- Column A:** Expected count of each string  $F(x) / \underline{F}$
- Column B:** Probability of each string being copied in the mating pool  
= Value of Column A / Population size

9

String												Mating Pool			
	Substring 2	Substring 1	$x_2$	$x_1$	$f(x)$	$F(x)$	A	B	C	D	E	F	Substring 2	Substring 1	
1	1110010000	1100100000	5.349	4.692	959.680	0.001	0.13	0.007	0.007	0.472	10	0	0010100100	1010101010	
2	0001001101	0011100111	0.452	1.355	105.520	0.009	1.10	0.055	0.062	0.108	3	1	1010100001	0111001000	
3	1010100001	0111001000	3.947	2.674	126.685	0.008	0.98	0.049	0.111	0.045	2	1	0001001101	0011100111	
4	1001000110	1000010100	3.413	3.120	65.026	0.015	1.85	0.093	0.204	0.723	14	2	1110011011	0111000010	
5	1100011000	1011100011	4.645	4.334	512.197	0.002	0.25	0.013	0.217	0.536	10	0	0010100100	1010101010	
6	0011100101	0011111000	1.343	1.455	70.868	0.014	1.71	0.086	0.303	0.931	19	2	0011100010	1011000011	
7	0101011011	0000000111	2.035	0.041	88.273	0.011	1.34	0.067	0.370	0.972	19	1	0011100010	1011000011	
8	1110101000	1110101011	5.490	5.507	1436.563	0.001	0.12	0.006	0.376	0.817	17	0	0111000010	1011000110	
9	1001111101	1011100111	3.736	4.358	265.556	0.004	0.49	0.025	0.401	0.363	7	1	0101011011	0000000111	
10	0010100100	1010101010	0.962	4.000	39.849	0.024	2.96	0.148	0.549	0.189	4	3	1001000110	1000010100	
11	1111101001	0001110100	5.871	0.680	814.117	0.001	0.14	0.007	0.556	0.220	6	0	0011100101	0011111000	
12	0000111101	0110011101	0.358	2.422	42.598	0.023	2.84	0.142	0.698	0.288	6	3	0011100101	0011111000	
13	0000111110	1110001101	0.364	5.331	318.746	0.003	0.36	0.018	0.716	0.615	12	1	0000111101	0110011101	
14	1110011011	0111000010	5.413	2.639	624.164	0.002	0.24	0.012	0.728	0.712	13	1	0000111110	1110001101	
15	1010111010	1010111000	4.094	4.082	286.800	0.003	0.37	0.019	0.747	0.607	12	0	0000111101	0110011101	
16	0100011111	1100111000	1.683	4.833	197.556	0.005	0.61	0.030	0.777	0.192	4	0	1001000110	1000010100	
17	0111000010	1011000110	2.639	4.164	97.699	0.010	1.22	0.060	0.837	0.386	9	1	1001111101	1011100111	
18	1010010100	0100001001	3.871	1.554	113.201	0.009	1.09	0.054	0.891	0.872	18	1	1010010100	0100001001	
19	0011100010	1011000011	1.326	4.147	57.753	0.017	2.08	0.103	0.994	0.589	12	2	0000111101	0110011101	
20	1011100011	1111010000	4.334	5.724	987.955	0.001	0.13	0.006	1.000	0.413	10	0	0010100100	1010101010	


A = Expected Count =  
 B = Probability Selection  
 C = Cumulative probability of selection  
 D = Random number between 0 and 1  
 E = String Number  
 F = True Count in the Mating pool

10

## Step 4 (Contd.)

- **Column C:** Cumulative probability of selection
- **Column D:** In order to form the mating pool, creat random numbers between zero and one and identify the particular string which is specified by each of these random numbers. e.g. for first string it has been 0.472 → that string occupies the interval (0.401,0.549) as shown in Column C → **String 10 is identified for Column E**

11



	String												Mating Pool		
	Substring 2	Substring 1	x <sub>2</sub>	x <sub>1</sub>	f(x)	F(x)	A	B	C	D	E	F	Substring 2	Substring 1	
1	1110010000	1100100000	5.349	4.692	959.680	0.001	0.13	0.007	0.007	0.472	10	0	0010100100	1010101010	
2	0001001101	0011100111	0.452	1.355	105.520	0.009	1.10	0.055	0.062	0.108	3	1	1010100001	0111001000	
3	1010100001	0111001000	3.947	2.674	126.685	0.008	0.98	0.049	0.111	0.045	2	1	0001001101	0011100111	
4	1001000110	1000010100	3.413	3.120	65.026	0.015	1.85	0.093	0.204	0.723	14	2	1110011011	0111000010	
5	1100011000	1011100011	4.645	4.334	512.197	0.002	0.25	0.013	0.217	0.536	10	0	0010100100	1010101010	
6	0011100101	0011111000	1.343	1.455	70.868	0.014	1.71	0.086	0.303	0.931	19	2	0011100010	1011000011	
7	0101011011	0000000111	2.035	0.041	88.273	0.011	1.34	0.067	0.370	0.972	19	1	0011100010	1011000011	
8	1110101000	1110101011	5.490	5.507	1436.563	0.001	0.12	0.006	0.376	0.817	17	0	0111000010	1011000110	
9	1001111101	1011100111	3.736	4.358	265.556	0.004	0.49	0.025	0.401	0.363	7	1	0101011011	0000000111	
10	0010100100	1010101010	0.962	4.000	39.849	0.024	2.96	0.148	0.549	0.189	4	3	1001000110	1000010100	
11	1111101001	0001110100	5.871	0.680	814.117	0.001	0.14	0.007	0.556	0.220	6	0	0011100101	0011111000	
12	0000111101	0110011101	0.358	2.422	42.598	0.023	2.84	0.142	0.698	0.288	6	3	0011100101	0011111000	
13	0000111110	1110001101	0.364	5.331	318.746	0.003	0.36	0.018	0.716	0.615	12	1	0000111101	0110011101	
14	1110011011	0111000010	5.413	2.639	624.164	0.002	0.24	0.012	0.728	0.712	13	1	0000111110	1110001101	
15	1010111010	1010111000	4.094	4.082	286.800	0.003	0.37	0.019	0.747	0.607	12	0	0000111101	0110011101	
16	0100011111	1100111000	1.683	4.833	197.556	0.005	0.61	0.030	0.777	0.192	4	0	1001000110	1000010100	
17	0111000010	1011000110	2.639	4.164	97.699	0.010	1.22	0.060	0.837	0.386	9	1	1001111101	1011100111	
18	1010010100	0100001001	3.871	1.554	113.201	0.009	1.09	0.054	0.891	0.872	18	1	1010010100	0100001001	
19	0011100010	1011000011	1.326	4.147	57.753	0.017	2.08	0.103	0.994	0.589	12	2	0000111101	0110011101	
20	1011100011	1111010000	4.334	5.724	987.955	0.001	0.13	0.006	1.000	0.413	10	0	0010100100	1010101010	

A = Expected Count =  
 B = Probability Selection  
 C = Cumulative probability of selection  
 D = Random number between 0 and 1  
 E = String Number  
 F = True Count in the Mating pool

12

12

## Step 4 (Contd.)

- **Column F** : True count in the mating pool (Column E)
- **Observation** : Columns A and F reveal that the theoretical expected count and the true count of each string more or less agree with each other.

13

[illegible]

A = Expected Count =  
B = Probability Selection  
C = Cumulative probability of selection  
D = Random number between 0 and 1  
E = String Number  
F = True Count in the Mating pool

14

## Step 5

- Mating Pool are used for Crossover.
- In single-point crossover, two strings are selected at random and crossed at a random site.
- e.g. At random select strings 3 and 10 for first cross-over operation.
- First a coin is flipped with a probability  $p_c = 0.8$  to check whether a cross-over is desired or not.
- Let us say – YES.

15

## Step 5 (Contd.)

- Find the **cross-site at random**.
- We choose a site by creating a random number between **(0, l-1) or (0,19)**
- In this case, with 10 pairs Crossover operation will be carried out.
- Store the revised strings as **Intermediate Population**

16



## Step 6

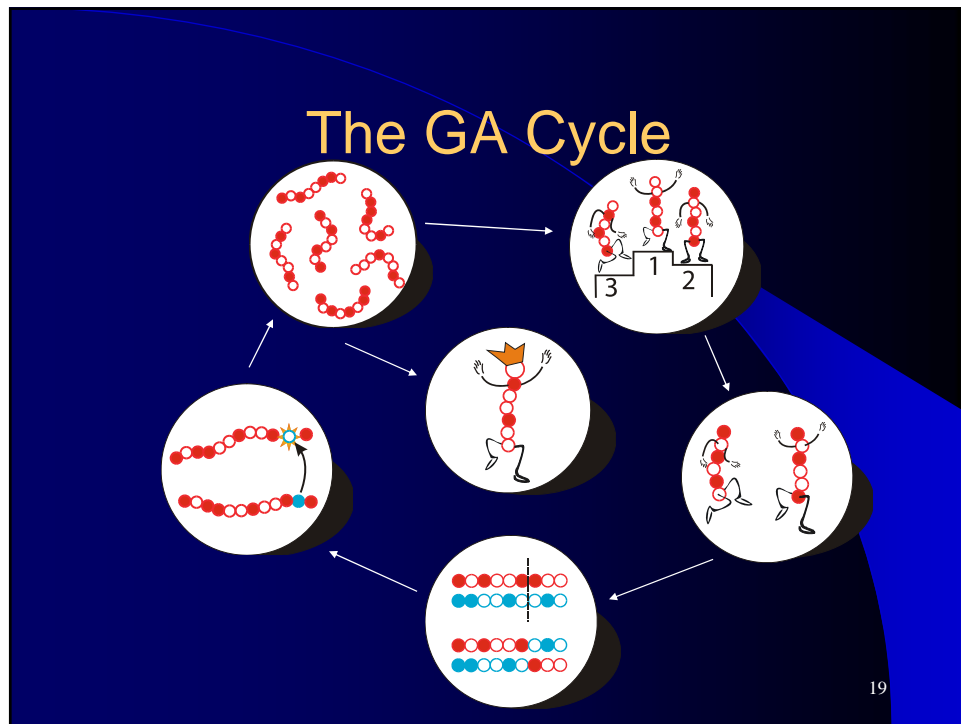
- Mutation process on intermediate population
- For bit-wise mutation, flip the coin with a probability  $p_m = 0.05$  for every bit.
- In this case, maximum  $0.05 \times 20 \times 20 = 20$  bits in population (population size 20, string length 20).

17

## Step 7

- Resulting population becomes the new population. Carryout steps of 2-6.
- Go on till we find the point at which the fitness value is near to 1.0

18



### Example

## Traveling Salesman Problem

20

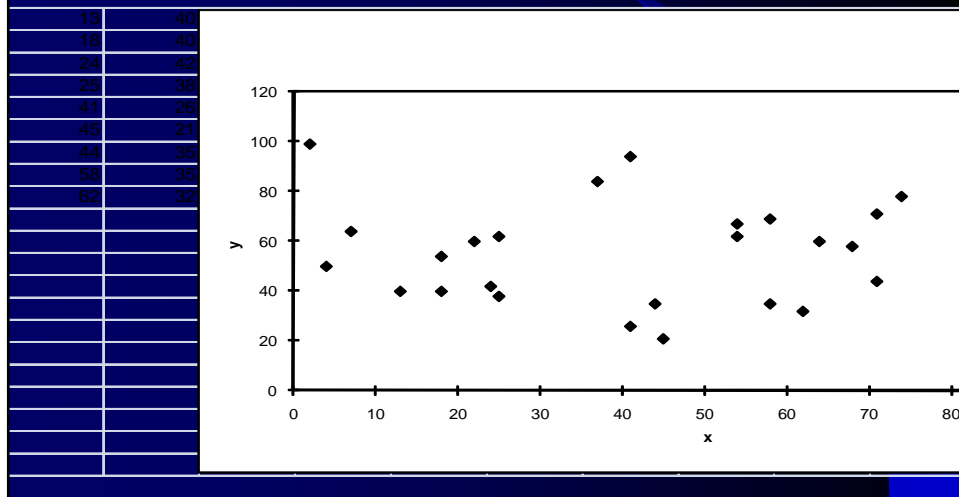
# Traveling Salesman Problem

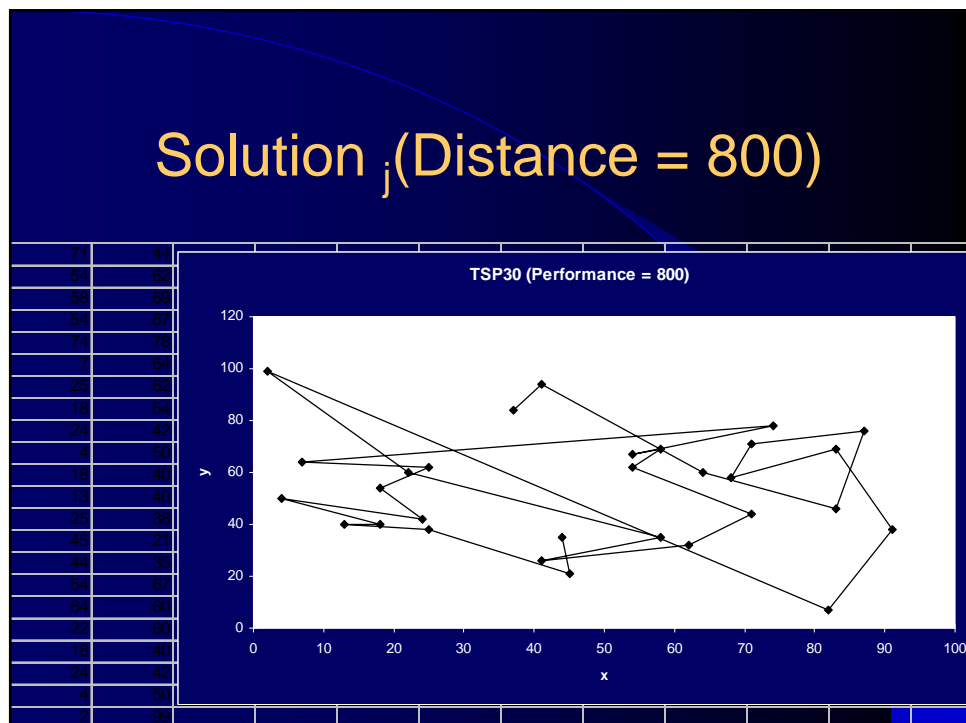
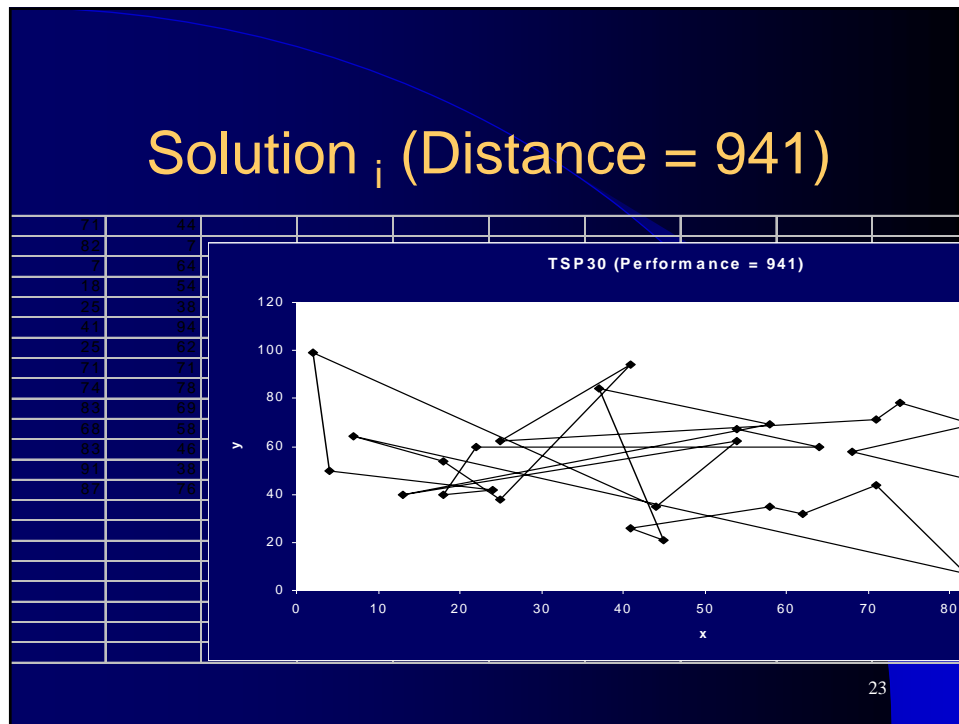
Find a tour of a given set of cities so that

- each city is visited only once
- the total distance traveled is minimized

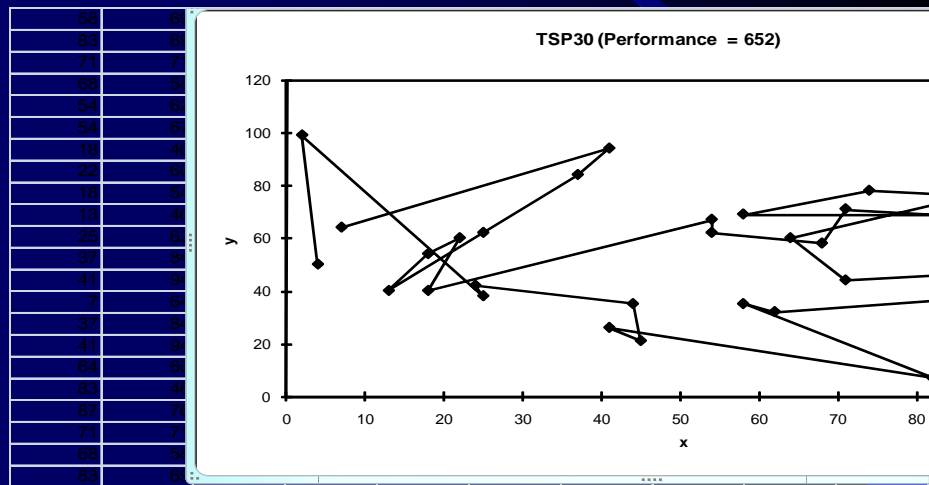
21

# TSP Example: 30 Cities

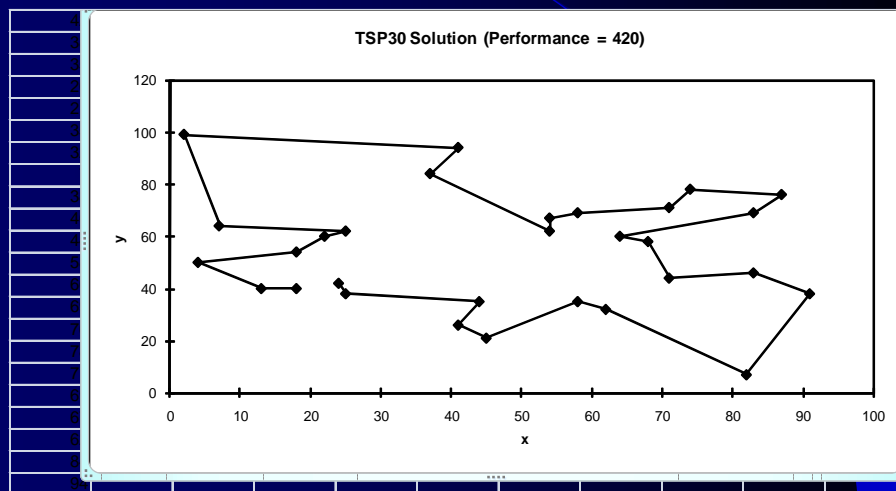




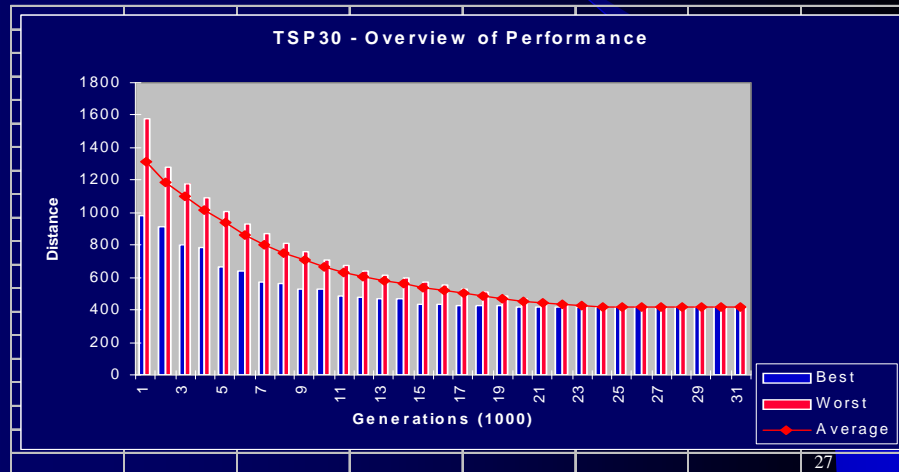
## Solution $k$ (Distance = 652)



## Best Solution (Distance = 420)



## Overview of Performance



## Some Issues

## Discussion on Issues

- Constraints Violation
- Selection
- Cross over
- Mutation
- Debate – Cross over Vs Mutation
- Coding Approaches

29

## Constraint Violation

30

# Constraint Violation

- Penalty Function

31

Penalty  
depending  
upon # of  
Constraints  
violation

String														Mating Pool	
Substring 2	Substring 1	$x_2$	$x_1$	$f(x)$	$F(x)$	A	B	C	D	E	F	Substring 2	Substring 1		
1	1110010000	1100100000	5.349	4.692	959.680	0.001	0.13	0.007	0.007	0.472	10	0	0010100100	1010101010	
2	0001001101	0011100111	0.452	1.355	105.520	0.009	1.10	0.055	0.062	0.108	3	1	1010100001	0111001000	
3	1010100001	0111001000	3.947	2.674	126.685	0.008	0.98	0.049	0.111	0.045	2	1	0001001101	0011100111	
4	1001000110	1000010100	3.413	3.120	65.026	0.015	1.85	0.093	0.204	0.723	14	2	1110011011	0111000010	
5	1100011000	1011100011	4.645	4.334	512.197	0.002	0.25	0.013	0.217	0.536	10	0	0010100100	1010101010	
6	0011100101	0011111000	1.343	1.455	70.868	0.014	1.71	0.086	0.303	0.931	19	2	0011100010	1011000011	
7	010101011	0000000111	2.035	0.041	88.273	0.011	1.34	0.067	0.370	0.972	19	1	0011100010	1011000011	
8	1110101000	1110101011	5.490	5.507	1436.563	0.001	0.12	0.006	0.376	0.817	17	0	0111000010	1011000110	
9	1001111101	1011100111	3.736	4.358	265.556	0.004	0.49	0.025	0.401	0.363	7	1	0101011011	0000000111	
10	0010100100	1010101010	0.962	4.000	39.849	0.024	2.96	0.148	0.549	0.189	4	3	1001000110	1000010100	
11	1111101001	0001110100	5.871	0.680	814.117	0.001	0.14	0.007	0.556	0.220	6	0	0011100101	0011111000	
12	0000111101	0110011101	0.358	2.422	42.598	0.023	2.84	0.142	0.698	0.288	6	3	0011100101	0011111000	
13	0000111110	1110001101	0.364	5.331	318.746	0.003	0.36	0.018	0.716	0.615	12	1	0000111101	0110011101	
14	1110011011	0111000010	5.413	2.639	624.164	0.002	0.24	0.012	0.728	0.712	13	1	0000111110	1110001101	
15	1010111010	1010111000	4.094	4.082	286.800	0.003	0.37	0.019	0.747	0.607	12	0	0000111101	0110011101	
16	0100011111	1100111000	1.683	4.833	197.556	0.005	0.61	0.030	0.777	0.192	4	0	1001000110	1000010100	
17	0111000010	1011000110	2.639	4.164	97.699	0.010	1.22	0.060	0.837	0.386	9	1	1001111101	1011100111	
18	1010010100	0100001001	3.871	1.554	113.201	0.009	1.09	0.054	0.891	0.872	18	1	1010010100	0100001001	
19	0011100010	1011000011	1.326	4.147	57.753	0.017	2.08	0.103	0.994	0.589	12	2	0000111101	0110011101	
20	1011100011	1111010000	4.334	5.724	987.955	0.001	0.13	0.006	1.000	0.413	10	0	0010100100	1010101010	

32



## Selection

33

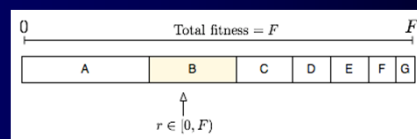
## Selection

- Fitness proportionate selection
- Stochastic universal sampling
- Tournament selection
- Reward-based selection

34

## Fitness Proportionate Selection

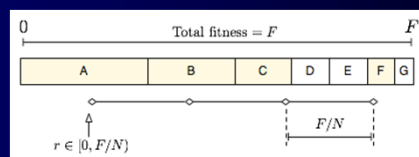
- Roulette wheel



35

## Stochastic Universal Sampling

- SUS uses a single random value to sample all of the solutions by choosing them at **evenly spaced intervals**. This gives weaker members of the population (according to their fitness) a chance to be chosen and thus reduces the unfair nature of fitness-proportional selection methods.



36

## Tournament Selection

- **Tournament selection** is a method of selecting an individual from a population of individuals in a genetic algorithm.
- Tournament selection involves running several "tournaments" among a few individuals chosen at random from the population.
- The winner of each tournament (the one with the best fitness) is selected for crossover.
- Selection pressure is easily adjusted by changing the tournament size.
- If the tournament size is larger, weak individuals have a smaller chance to be selected.

37

## Reward Based Selection

- **Reward-based selection** is a technique used in evolutionary algorithms for selecting potentially useful solutions for recombination.
- The probability of being selected for an individual is proportional to the cumulative reward, obtained by the individual.
- The cumulative reward can be computed as a sum of the individual reward and the reward, inherited from parents.

38

## Cross Over

39

## Cross Over Approaches

- One-point crossover
- Two-point crossover
- "Cut and splice"
- Uniform Crossover and Half Uniform Crossover
- Three parent crossover
- Crossover for Ordered Chromosomes

40

## One Point Crossover

- A single crossover point on both parents' organism strings is selected. All data beyond that point in either organism string is swapped between the two parent organisms. The resulting organisms are the children:

41

## Two Point Crossover

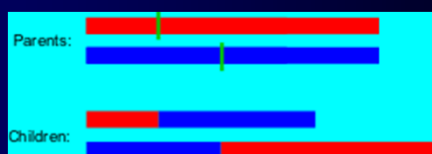
- Two-point crossover calls for two points to be selected on the parent organism strings. Everything between the two points is swapped between the parent organisms, rendering two child organisms:



42

## Cut and Splice

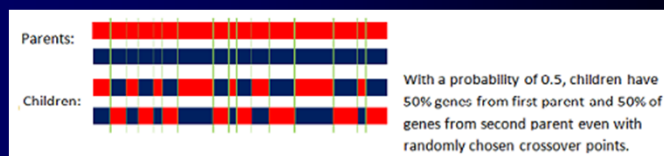
- Another crossover variant, the "cut and splice" approach, results in a change in length of the children strings. The reason for this difference is that each parent string has a separate choice of crossover point.



43

## Uniform cross over and half uniform crossover

- The Uniform Crossover uses a fixed mixing ratio between two parents. Unlike one- and two-point crossover, the Uniform Crossover enables the parent chromosomes to contribute the gene level rather than the segment level.
- If the mixing ratio is 0.5, the offspring has approximately half of the genes from first parent and the other half from second parent, although cross over points can be randomly chosen



44

## Three Parent Crossover

- In this technique, the child is derived from three parents. They are randomly chosen. Each bit of first parent is checked with bit of second parent whether they are same. If same then the bit is taken for the offspring otherwise the bit from the third parent is taken for the offspring. For example, the following three parents:
  - parent1 1 1 0 1 0 0 0 1 0
  - parent2 0 1 1 0 0 1 0 0 1
  - parent3 1 1 0 1 1 0 1 0 1produces the following offspring:
  - offspring 1 1 0 1 0 0 0 0 1

45

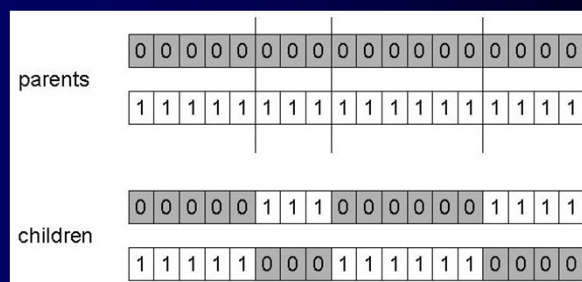
## Alternative Crossover Operators

- Performance with 1 Point Crossover depends on the order that variables occur in the representation
  - more likely to keep together genes that are near each other

46

## N-point Crossover

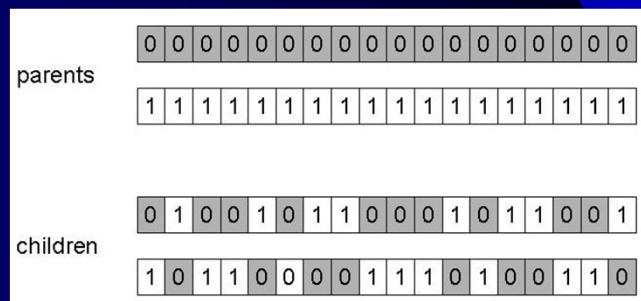
- Choose n random crossover points
- Split along those points
- Glue parts, alternating between parents
- Generalisation of 1 point (still some positional bias)



47

## Uniform Crossover

- Assign 'heads' to one parent, 'tails' to the other
- Flip a coin for each gene of the first child
- Make an inverse copy of the gene for the second child
- Inheritance is independent of position



48



## Mutation

49

## Mutation

- **Flip Bit** - This mutation operator takes the chosen genome and inverts the bits. (i.e. if the genome bit is 1, it is changed to 0 and vice versa)
- **Boundary** - This mutation operator replaces the genome with either lower or upper bound randomly. This can be used for integer and float genes.

50

## Mutation

- **Non-Uniform** - The probability that amount of mutation will go to 0 with the next generation is increased by using non-uniform mutation operator. It keeps the population from stagnating in the early stages of the evolution. It tunes solution in later stages of evolution. This mutation operator can only be used for integer and float genes.

51

## Mutation

- **Uniform** – This operator replaces the value of the chosen gene with a uniform random value selected between the user-specified upper and lower bounds for that gene. This mutation operator can only be used for integer and float genes.

52

## Mutation

- **Gaussian** - This operator adds a unit Gaussian distributed random value to the chosen gene. If it falls outside of the user-specified lower or upper bounds for that gene, the new gene value is clipped. This mutation operator can only be used for integer and float genes.

53

## Debate – Cross over Vs Mutation

54

## Crossover OR Mutation?

- Decade long debate: which one is better / necessary / main-background
- Answer (at least, rather wide agreement):
  - it depends on the problem, but
  - in general, it is good to have both
  - both have another role
  - mutation-only-EA is possible, crossover-only-EA would not work

55

## Crossover OR Mutation?

Exploration: Discovering promising areas in the search space, i.e. gaining information on the problem

Exploitation: Optimising within a promising area, i.e. using information

There is co-operation AND competition between them

- Crossover is explorative, it makes a *big* jump to an area somewhere “in between” two (parent) areas
- Mutation is exploitative, it creates random *small* diversions, thereby staying near (in the area of ) the parent

56

## Crossover OR Mutation?

- Only crossover can combine information from two parents
- Only mutation can introduce new information
- To hit the optimum you often need a 'lucky' mutation

57

## Coding Approaches

58

## Coding Representations

- Binary Coding
- Real Coding

59

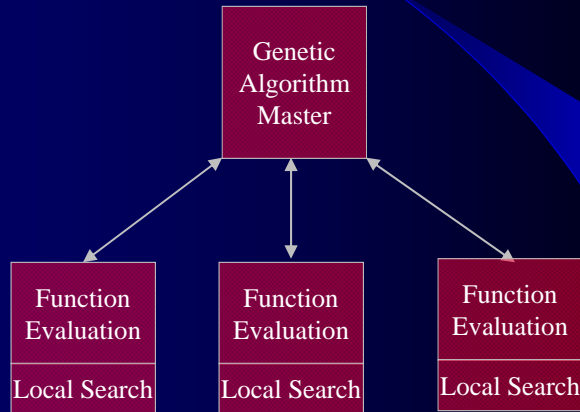
## Parallelisation Methods

Common parallel GA prototypes:

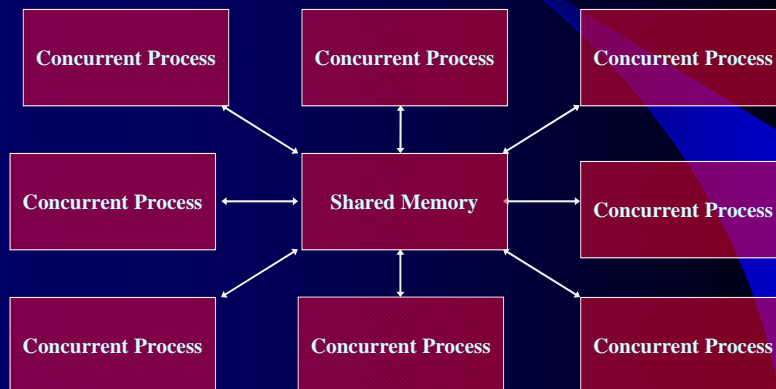
- Master Slave prototype.
- Distributed, Asynchronous Concurrent prototype.
- Network model.
- Island model.



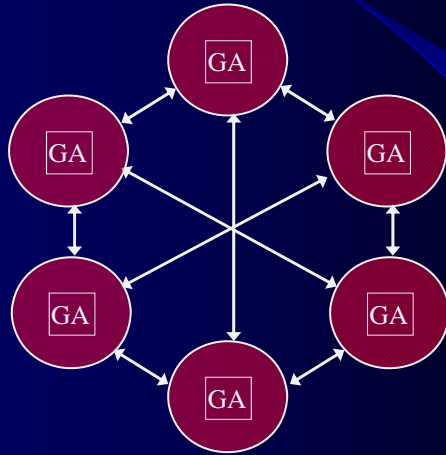
## Master Slave prototype



## Distributed, Asynchronous Concurrent prototype.



## Network Model



## Island Model

