Lecture 3: Genetic Algorithms Examples

Sabah Sayed

Department of Computer Science

Faculty of Computers and Artificial Intelligence

Cairo University

Egypt

Implementing Randomization in GAs

Canonical Genetic Algorithm

```
Canonical_Genetic_Algorithm()
{
   Initialize the Population = G0; // population size is constant …

   For(i=1 to Max_Generations) // Continue Evolution …
   {
        Evaluate Fitness of Individuals of Gi-1; // By Objective Function …

        Select Parents for Reproduction; // Roulette Wheel?

        Crossover; // According to probability of crossover…

        Mutation;  // According to probability of mutation

        Replacement;  // Replacing old generation Gi-1 with new generation Gi
   }
}
```

Crossover Probabilities

① • P_c fixed parameter in GA [0.4 → 0.7]
  – Probability that crossover will occur between two selected chromosomes
• Generate random number $r_1 \in [1, L-1]$
  – Where L = length of chromosome
② • Crossover point = r_1
  – Crossover will occur between two individuals after r_1 genes

1 → 7

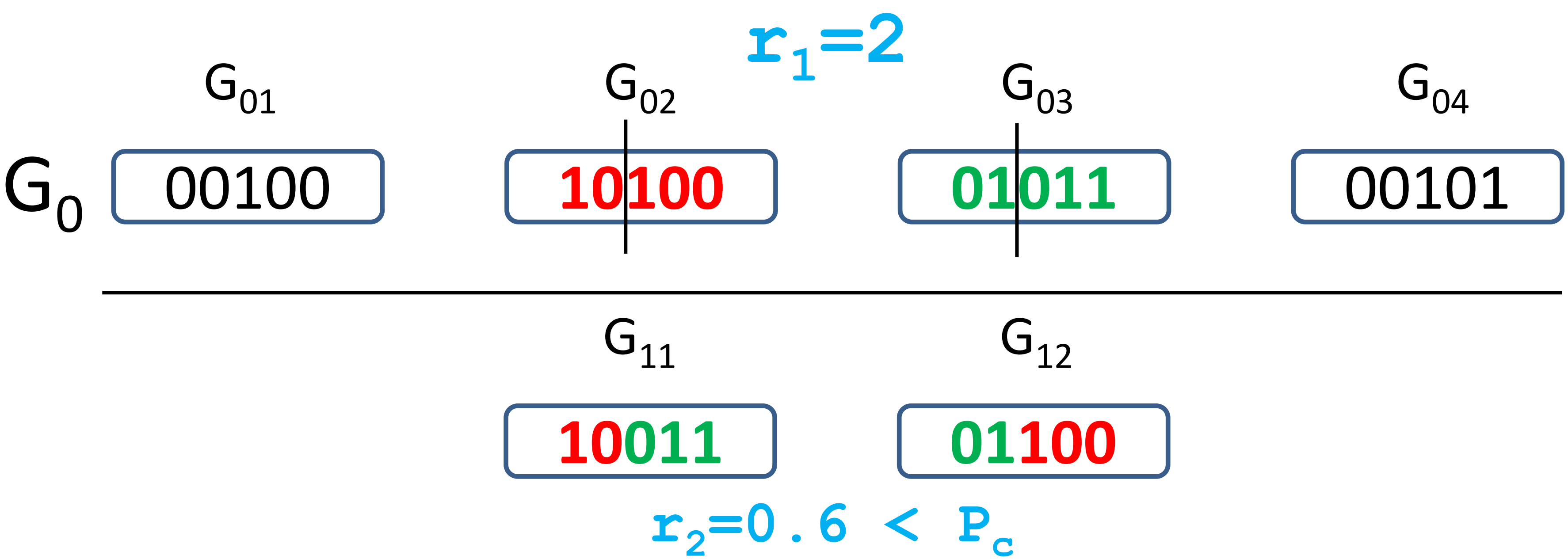
③ • Generate random number $r_2 \in [0, 1]$
  – if $r_2 \leq P_c$ then perform crossover
  – else no crossover occur …
    • Offspring1 = Parent1
    • Offspring2 = Parent2

② ③

• Can steps of generating r_1 and r_2 be swapped?

Crossover Probabilities

- Assume $P_c = 0.7$



- Single point crossover after 2 bits (Genes)

Crossover Probabilities

- Assume $P_c=0.7$

$r_1=2$

	G_{01}	G_{02}	G_{03}	G_{04}
G_0	00100	10100	01011	00101

G_{11}	G_{12}
10100	01011

$r_2=0.8 > P_c$

- Single point crossover after 2 bits (Genes)
- No Crossover occurs

Crossover Probabilities

- Assume $P_c=0.7$

$r_1=3$

G_0

	G_{01}	G_{02}	G_{03}	G_{04}
G_0	00100	101 | 00	010 | 11	00101

G_{11}	G_{12}
10111	01000

$r_2=0.45 < P_c$

- Single point crossover after 3 bits

Mutation Probabilities

- P_m fixed parameter in GA [0.001 → 0.1]
  - Probability that mutation will occur for a gene/bit in some chromosomes
- Chromosome = bits[1…L]

```
for(i=1 to L)
{
    Generate Random number r_i ∈ [0, 1]
    if(r_i ≤ P_m)
        flip bit[i]
    elseif(r_i > pm)
        no change to bit[i]
}
```

Mutation Probabilities

- $P_m = 0.1$

r_i	Value
r_1	0.2
r_2	0.01
r_3	0.5
r_4	0.11
r_5	0.1

10101

21

11100

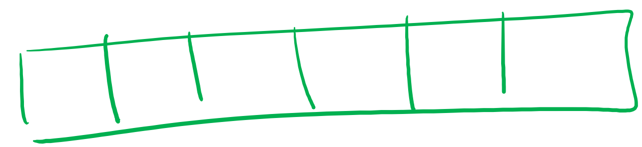
21→28

Mutation

Example: $f(x) = x^2$

- **Find the maximum of a function:**
  - $\boldsymbol{f(x) = x^2}$
  - Range [0, 63]

$x \longrightarrow$



Example : $f(x) = x^2$

- **Finding the maximum of a function:**
  - ***$f(x) = x^2$***
  - Range [0, 63]
  - Binary representation: string length 6 → 64 numbers (0-63)

genotype	0 0 0 1 0 1	
mapping	… 2^2 2^1 2^0 … 4 2 1	
phenotype	… 1*4+0*2+1*1 = 5	
fitness	25	= f(x)

$f(x) = x^2$

Initial Random Population

pop_sz = 5

	binary	value x	fitness x^2
String 1	000110 ✓	6	36
String 2	000011 ✓	3	9
String 3	001010 ✓	10	100
String 4	010101 ✓	21	441
String 5	000001 ✓	1	1

$f(x) = x^2$ Selection

	binary	value	fitness
String 1	000110	6	36
String 2	000011	3	9
String 3	001010	10	100
String 4	010101	21	441
String 5	000001	1	1

- Worst one can be removed

$f(x) = x^2$

Selection

	binary	value	fitness
String 1	000110	6	36
String 2	000011	3	9
String 3	001010	10	100
String 4	010101	21	441
String 5	000001	1	1

- Best individual: can be reproduced twice → keep population size constant

$f(x) = x^2$ Selection

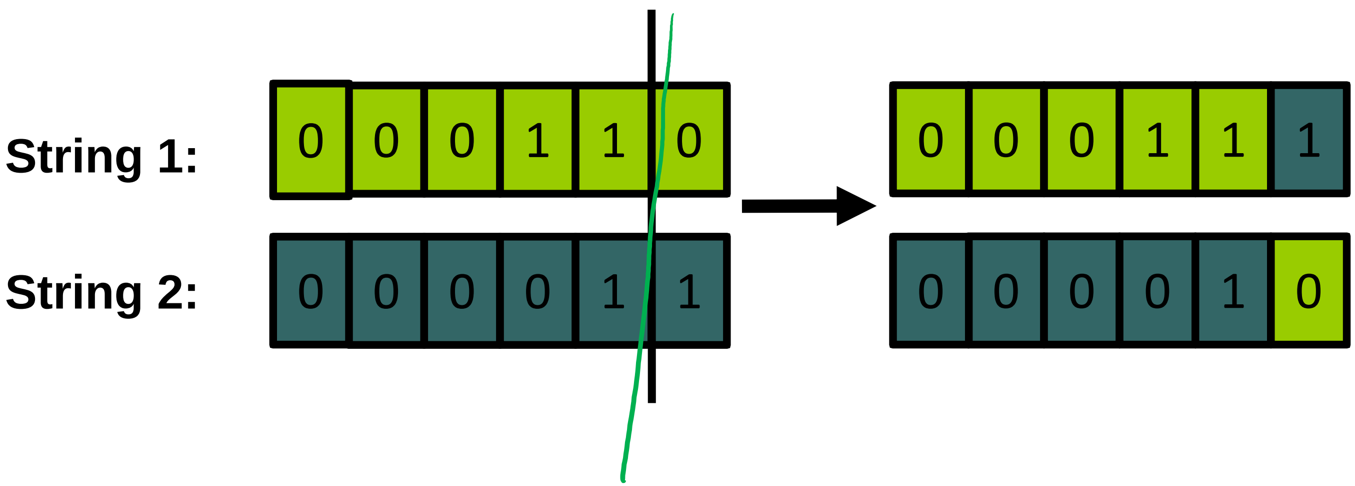
	binary	value	fitness
String 1	000110	6	36
String 2	000011	3	9
String 3	001010	10	100
String 4	010101	21	441
String 5	000001	1	~~1~~

- All others are reproduced once

$f(x) = x^2$ Recombination

- Parents and x-position randomly selected

	partner	x-position
String 1	**String 2**	~~**5**~~ r_1
String 3	String 4	3 r_1



$f(x) = x^2$ Recombination

- Parents and x-position randomly selected

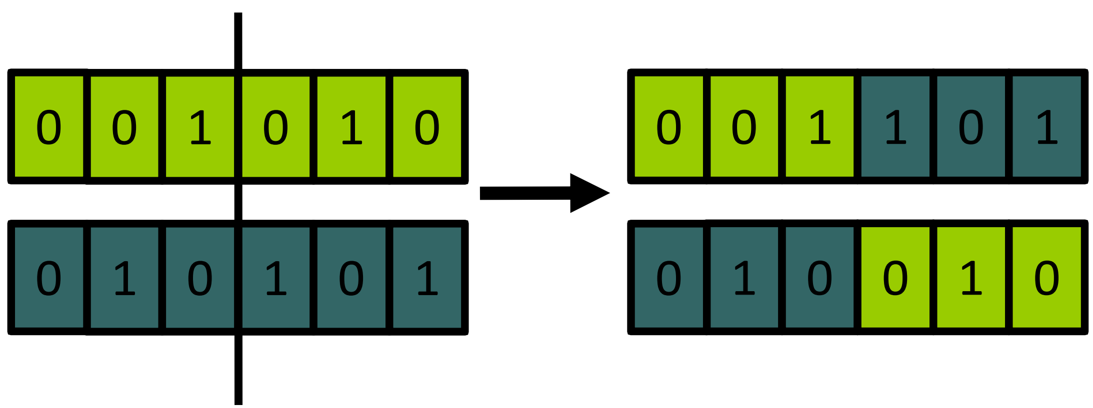
r_2

r_2

	partner	x-position
String 1	String 2	5
String 3	**String 4**	**3**

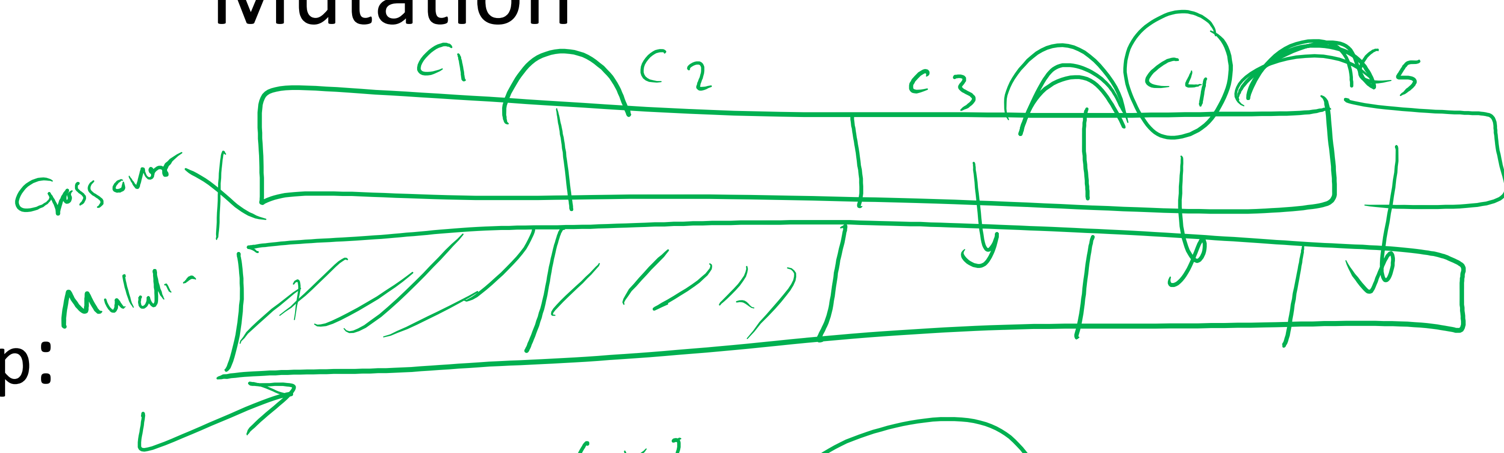
String 3:

String 4:



$f(x) = x^2$

Mutation



- bit-flip:
  - Offspring-String 1: 0**0**0111 (7) → 0**1**0111 (23)
  - String 5: 010**1**01 (21) → 010**0**01 (17)

6 r's

$f(x) = x^2$

Old Generation

	binary	value	fitness
String 1	000110	6	36
String 2	000011	3	9
String 3	001010	10	100
String 4	010101	21	441
String 5	000001	1	1

$f(x) = x^2$

New Generation

63×63

- All individuals in the parent population are replaced by offspring in the new generation
  - (generations are ***discrete***!)

Full

- **New population (Offspring)**:

① Max fitness

② Average "

	binary	value	fitness
String 1	010111	23	529
String 2	000010	2	4
String 3	001101	13	169
String 4	010010	18	324
String 5	010001	17	289

$f(x) = x^2$

When to stop?

- Iterate until termination condition reached, e.g.:
  - 1 – Best fitness < optimal solution Accuracy > 95%
  - 2 – Number of generations max
  - 3 – No New Chromosomes
  - 4 – No improvements after a number of generations

50 → 90

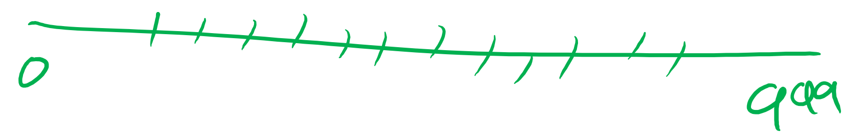
55 → 90

5

- Result after one generation:
  - Best individual: 010111 (23) – fitness 529

Drilling for Oil Example

- Imagine you had to drill for oil somewhere along a single 1km desert road
- Problem: choose the best place on the road that produces the most oil per day
- We could represent each solution as a position on the road



- Say, a whole number between [0..1000]

Where to drill for oil?

Solution1 = 300

Solution2 = 900

Road

0

500

1000

Digging for Oil

- The set of all possible solutions [0..1000] is called the *search space* or *state space*
- Often GA’s code numbers in binary producing a bitstring representing a solution
- In our example we choose 10 bits which is enough to represent 0..1000

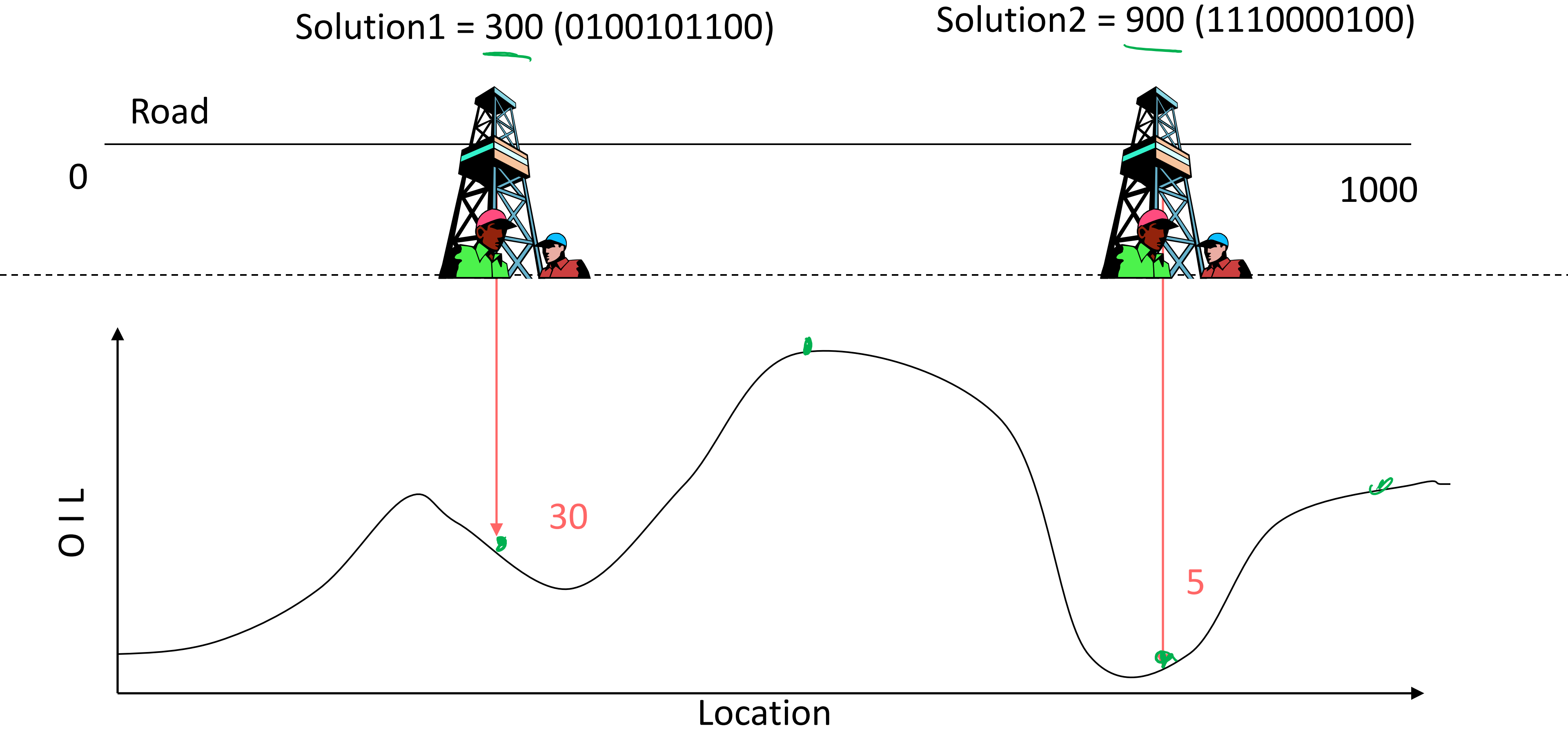
Convert to binary string

0 – 31 5 bits
0 – 63 6 bits

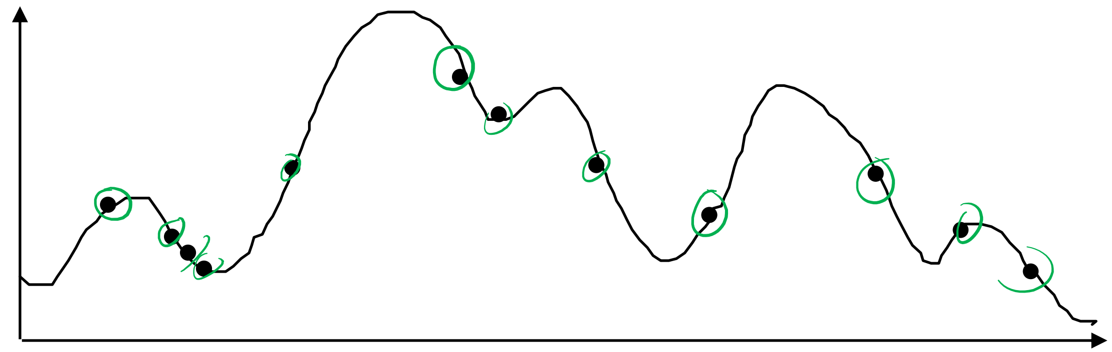
0 – 1000
0 – 1023 10 bits

	512	256	128	64	32	16	8	4	2	1
900	1	1	1	0	0	0	0	1	0	0
300	0	1	0	0	1	0	1	1	0	0
1023	1	1	1	1	1	1	1	1	1	1

The objective function

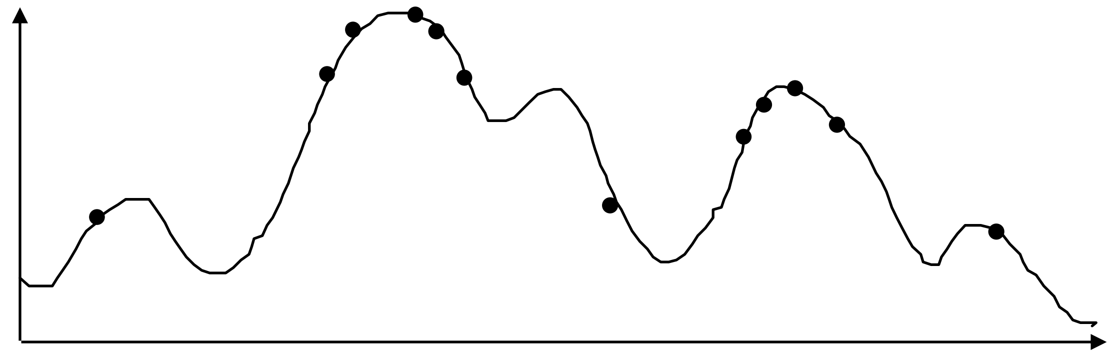


Individuals on Curve



Distribution of Individuals in Generation 0

1

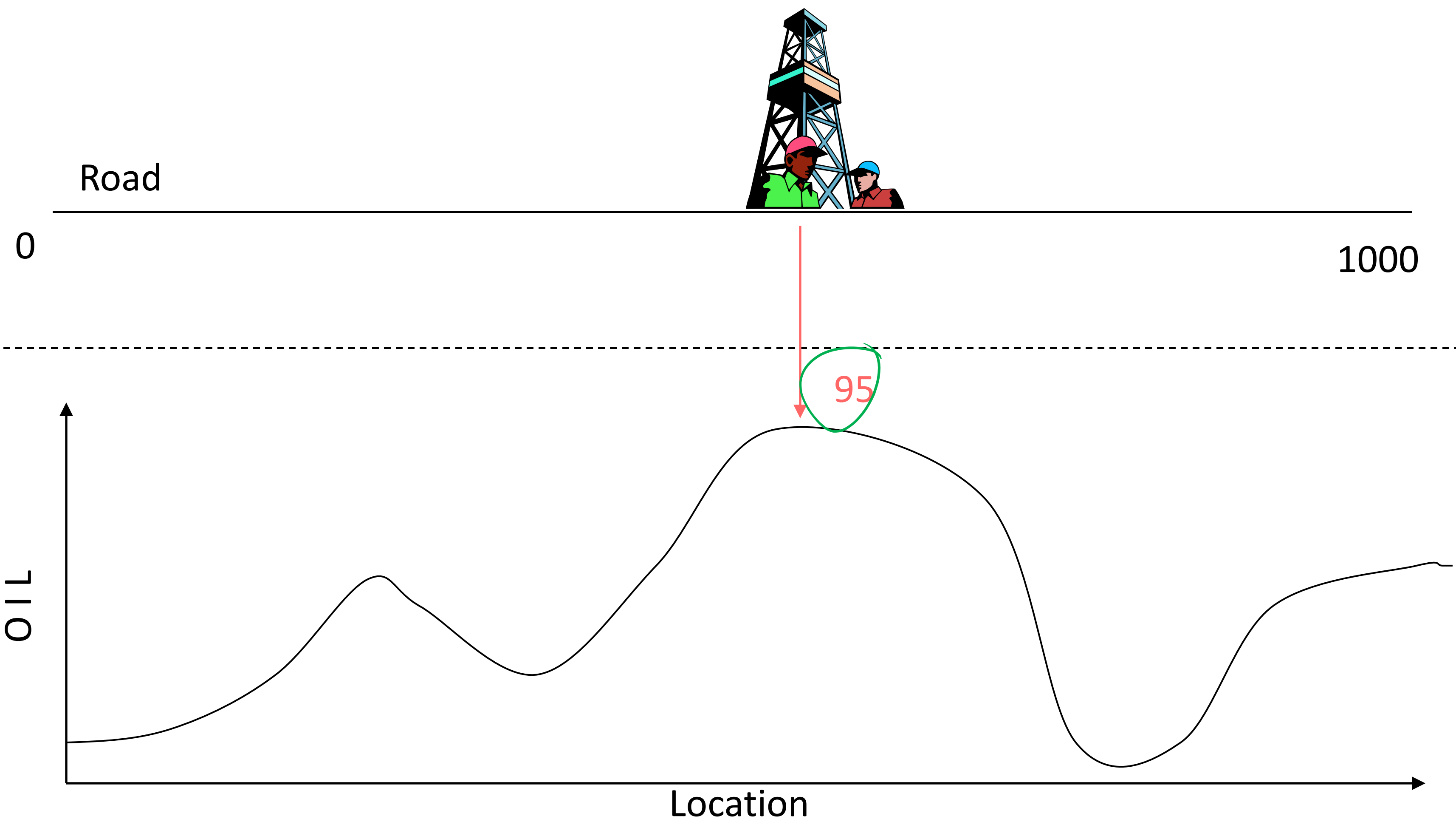


Distribution of Individuals in Generation N

2

Optimal Solution

Solution? = 600
(1001011000)



Summary

- Represent possible solutions as a number
- Encoded a number into a binary string
- Ensure that all genotypes correspond to feasible solutions
- Generate a score for each number given a *function* of "how good" each solution is
- Our oil example is really optimisation over a function f(x) where we adapt the parameter x