Master of Technology

Computational Intelligence II

Genetic Algorithms

Dr. Zhu Fangming
Institute of Systems Science,
National University of Singapore
Email: isszfm@nus.edu.sg

Objectives

- Upon the completion of this lecture, the students will be able
  - To understand concepts of genetic algorithms.
  - To explain the basic operations and characteristics of genetic algorithms.
  - To explain how genetic algorithm works.
  - To apply a simple genetic algorithm for solving problems.

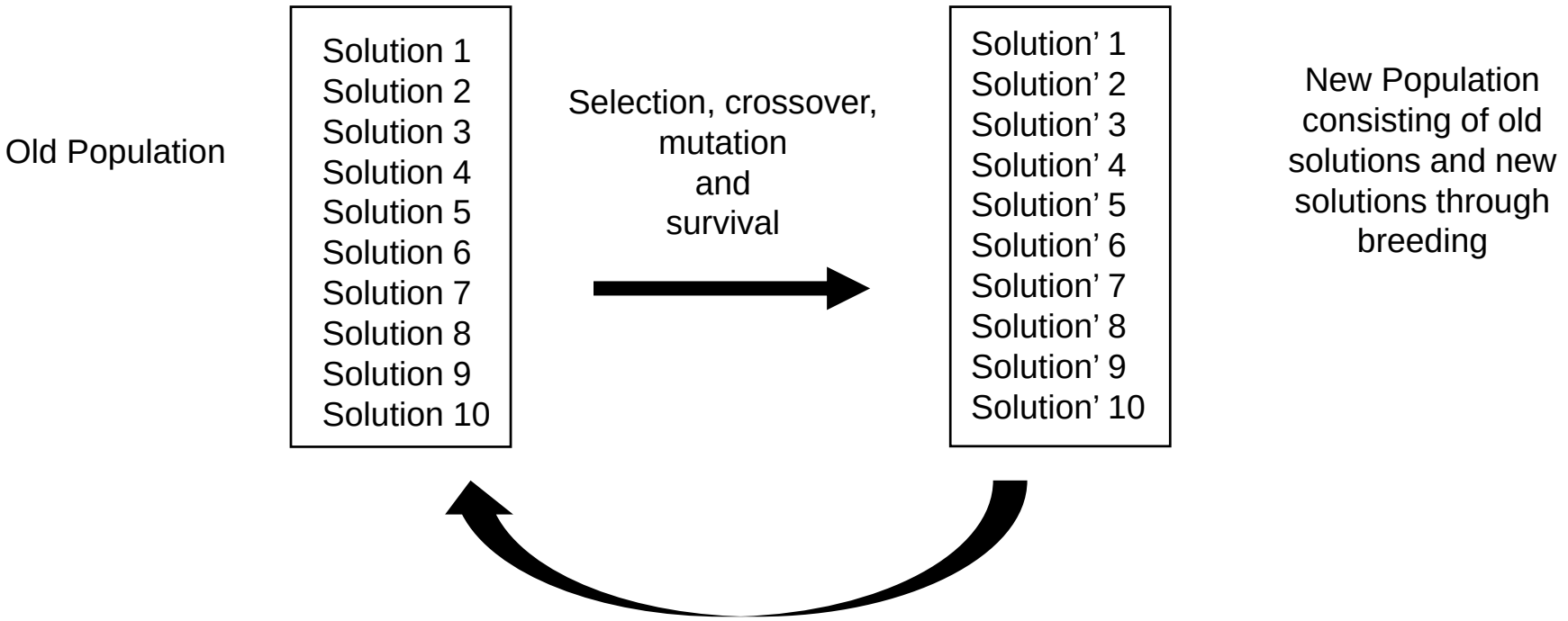
Topics

- Overview of Genetic Algorithms (GA)
- Core GA concepts
- Applications of GA
- Exercise
- Appendix: GA tools

Genetic Algorithms

- Genetic algorithms are general-purpose search and optimization algorithms that use principles inspired by natural population genetics to evolve solutions to problems
- GAs operate on a population of individuals representing potential solutions to a given problem.
- GAs seek to produce better (fitter) individuals (solutions) by combining the better of the existing ones (through breeding - crossover and mutation).

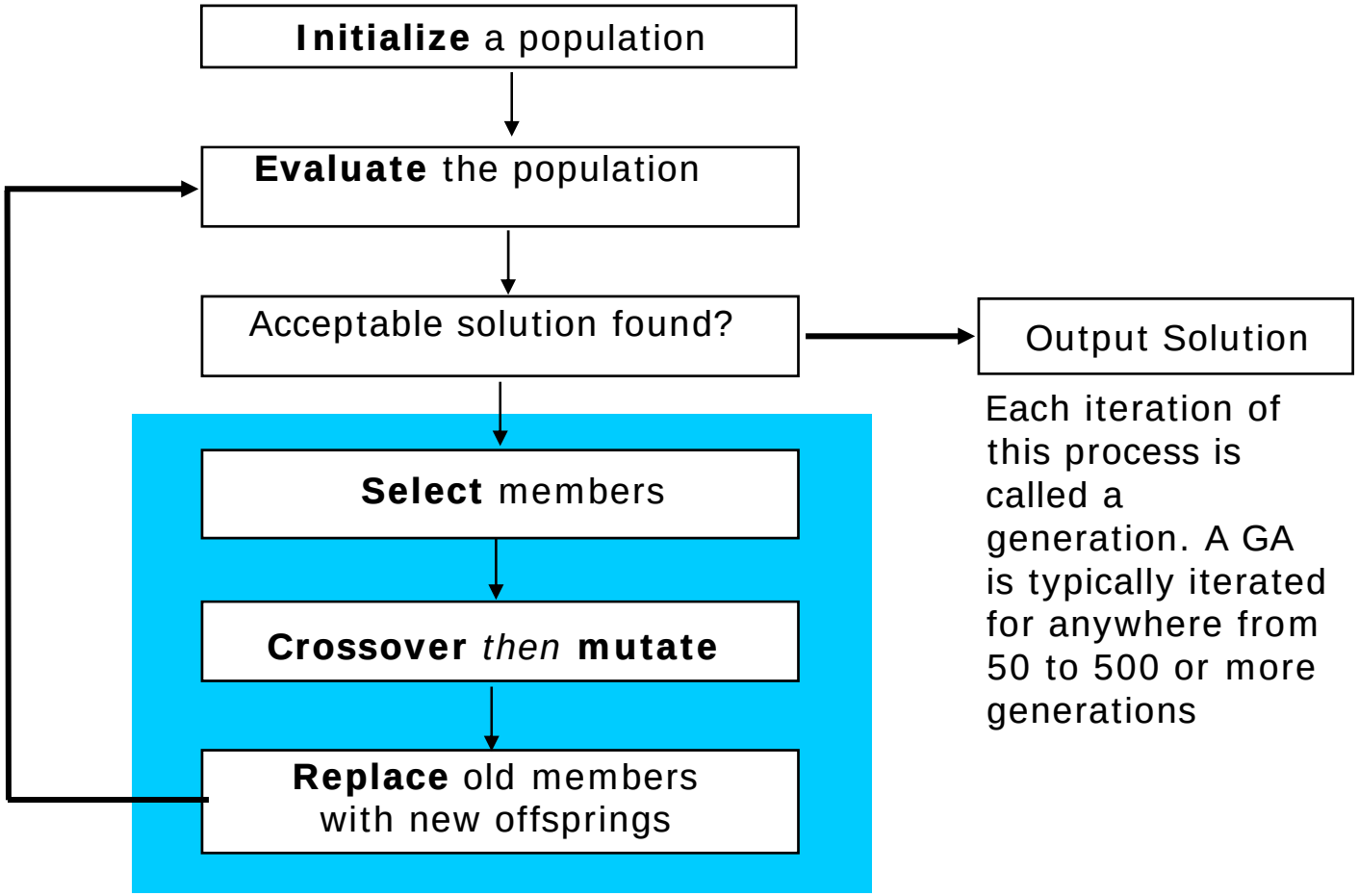
Genetic Algorithms



Genetic Algorithms

- The representation of the solution is encoded into strings (a sequence of values). Each value represents a property of the solution.
- The search is conducted using a population of solutions rather than considering only one solution.
- The search process is inherently parallel.
- The stochastic nature of GA means that searches are non-deterministic.

GA Search Process

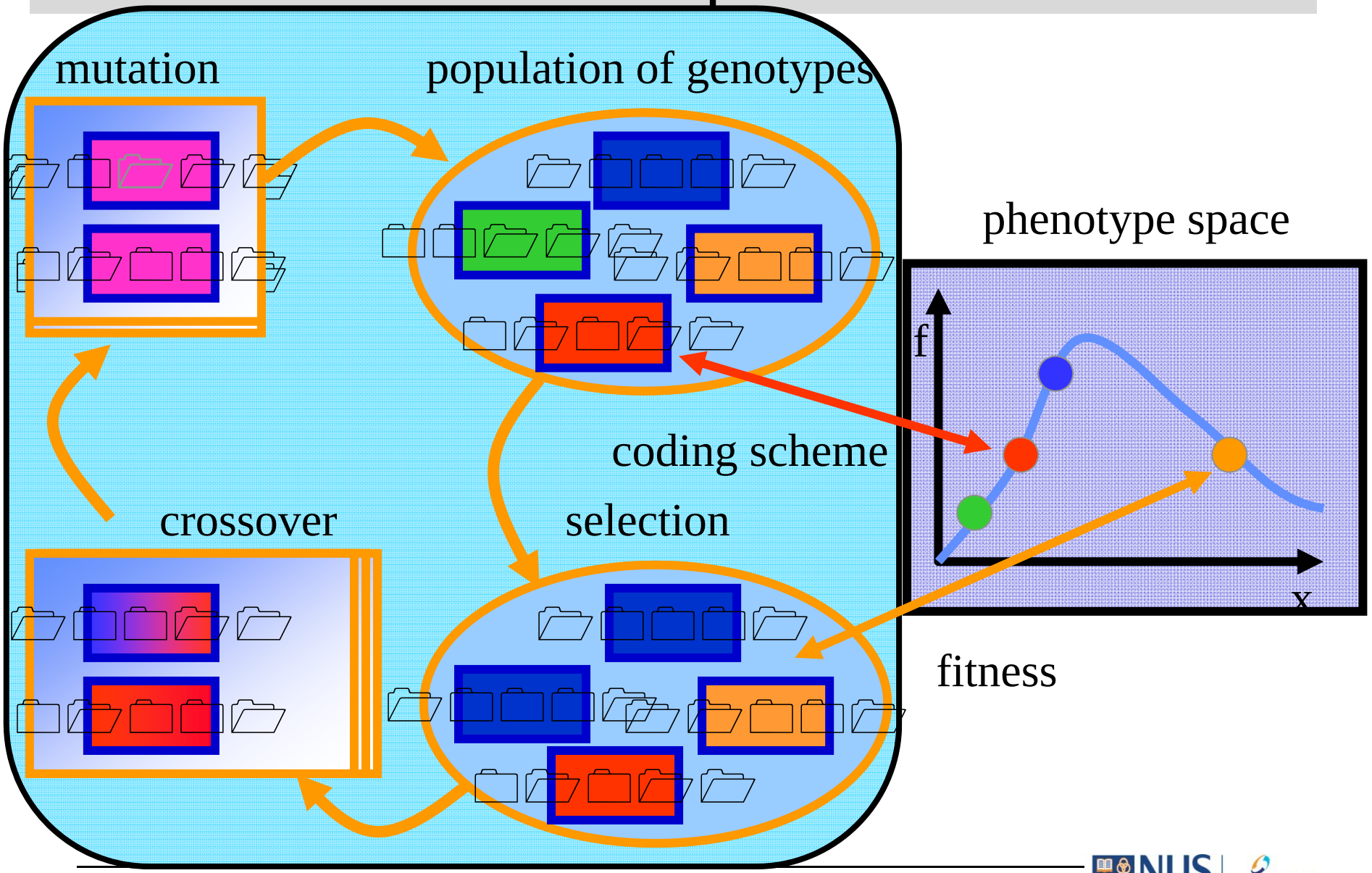


Each iteration of this process is called a generation. A GA is typically iterated for anywhere from 50 to 500 or more generations

GA Pseudococde

Generate the initial population P(0);
t=0;
repeat
    Evaluate the fitness of each individual in P(t);
    Select parents from P(t) based on their fitness;
    Applying crossover and mutation to parents to create O(t);
    Obtain population P(t + 1) by combining P(t) and O(t);
    t =t + 1;
until termination criterion satisfied;

Evolution as Optimisation
GA search process



Representing Solutions in GA

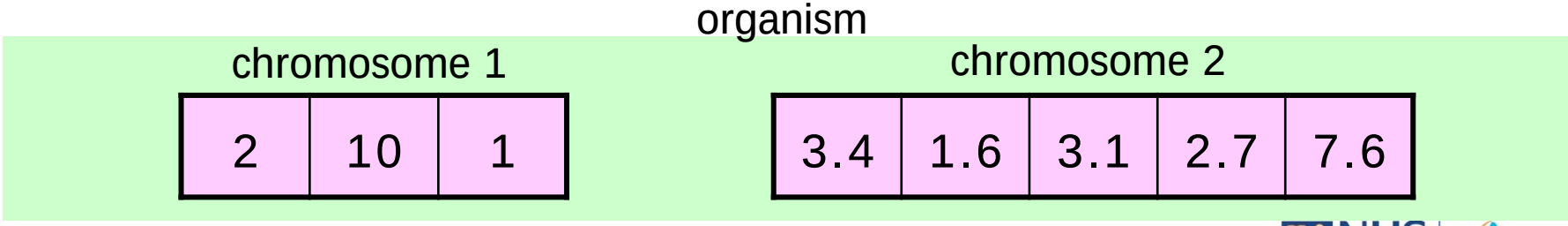
- GAs encode a problem's solution in a form that is analogous to nature's chromosomes or sequences of DNA.
  - Solutions are simply strings of values
    - Traditional primitive type is a single bit.
    - Integers, floating points or even higher level entities can be used.
- GAs are inherently independent of the application domain.

Encoded forms of problem solutions

0101001000010111100001100100011100001
5 6 1 9 8 3 7 4 2 10
14.5 79.0 22.8 9.3

Representing Solutions in GA

- GAs use a vocabulary borrowed from natural genetics
  - Individuals (structures) in a population are often called organisms comprising one or more chromosomes
  - Genes are substrings (bits) that define the features or properties of chromosomes
    - Genes of certain characters are located at certain places of the chromosome which are called **loci** (string positions) – the position of each gene is significant



Representing Solutions in GA

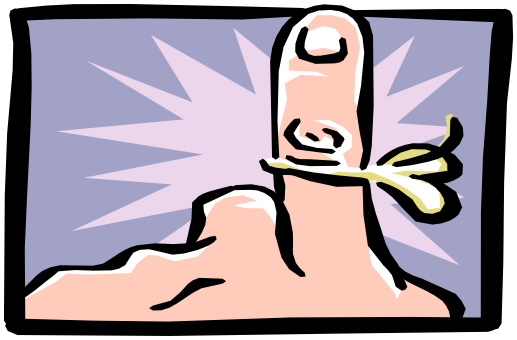
- The different possible settings for a gene are called **alleles** (feature values)
- Population is a collection of individuals (structures) each comprising one or more chromosomes
- Fitness function measures the fitness value of the individual
  - The fitter an individual, the more desirable it is

Problem Encoding

- The way in which candidate solutions are encoded is a central factor in the success of a genetic algorithm
  - Binary encodings
    - Most GA applications use **fixed-length, fixed-order** bit strings to encode solutions
    - Historical but unnatural (Holland's method)
  - Multiple-character or real-valued encodings

Problem Encoding

- How to decide on the correct encoding for a problem?
  - An experienced researcher, Lawrence Davis, strongly advocates using whatever encoding is the most natural for your problem, and then devising a GA that can use that encoding



Core GA Concepts

- Initial population
  - Can be initialized using whatever knowledge is available about the possible solutions
  - Should represent a random sample of the search space
  - Each member of the population is evaluated and assigned a measure of its fitness as a solution
- New population
  - Structures in the current population are selected for replication based on their relative fitness
  - Genetic operators (crossover, mutation) are performed to generate the new population

Core GA Concepts

- **Selection**
  - Reproduction focuses attention on **high fitness individuals**, thus exploiting on the available fitness information.
  - High-performing structures might be chosen several times for replication, while poor-performing structures might not be chosen at all.
- **Crossover**
  - Combines the features of two parent structures (new combinations of genes are generated from previous ones) to form two similar offspring → **conserves** genetic information.
- **Mutation**
  - Alters one or more components of a selected structure randomly (a direct analogy from nature and provides the way to introduce new information into the population) → population **diversity**

Core GA Concepts

- The resulting offspring are then evaluated and inserted back into the population, **replacing** older members.
  - Specific decisions about how many members are replaced during each iteration, and how members are selected for replacement, define a range of alternative implementations
  - Generational GA
    - GAs that replace the entire population
  - Steady-state GA
    - GAs that replace only a small fraction of chromosomes. Typically new chromosomes replace the worst chromosomes

Selection and Reproduction

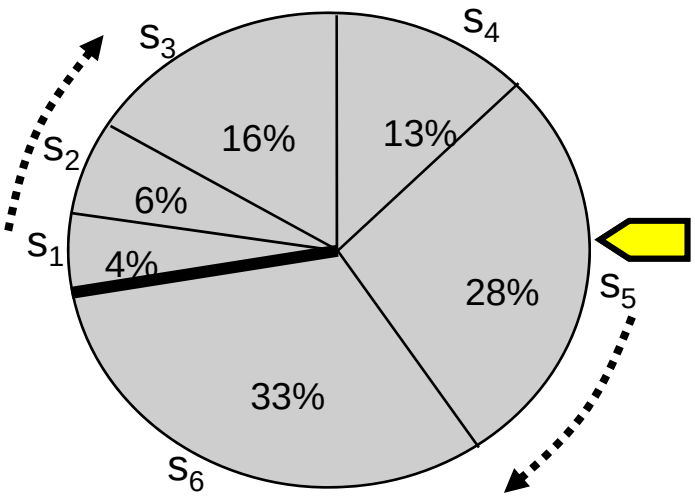
- Selection and reproduction in GA always in the given sequence
  1. **Select** fitter solutions (or individuals) to contribute genetic material to the next generation
     → selection method
  2. **Crossover** or recombine genetic materials from two parents to create new offsprings
     → crossover operator and crossover rate
  3. **Mutate** randomly selected components of new offsprings
     → mutation operator and mutation rate
  4. **Replace** some existing solutions in the current generation with the new offsprings to form the next generation
     → replacement strategy

Selection Methods
Roulette Wheel

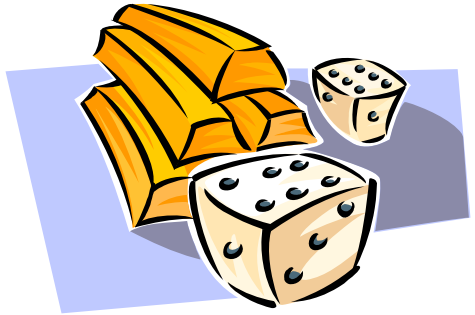
- Fitness-Proportionate Selection with "Roulette Wheel" *aka* **Wheel of Fortune**
  - Select new population with respect to the probability distribution based on fitness values
  - A roulette wheel with slots sized according to the fitness is used
    - Calculate the fitness value $f(s_i)$ for each chromosome s_i
    - Find the total fitness of the population
      - ✓ $F = \sum_i f(s_i)$
    - Calculate the probability of selection p_i for each chromosome s_i
      - ✓ $p_i = f(s_i)/F$
    - Calculate a cumulative probability q_i for each chromosome s_i
      - ✓ $q_i = \sum_j p_j$

Selection Methods
Roulette Wheel

- Spin the roulette wheel, each time a single chromosome for a new population is selected in the following way
  - Generate a random (float) number r from the range [0..1]
  - If $r < q_1$ then select the first chromosome; otherwise select the **i-th chromosome s_i** such that $q_{i-1} < r < \mathbf{q_i}$



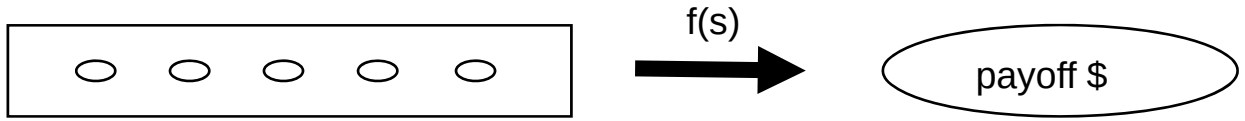
A "roulette-wheel sampling" giving each individual a slice of a circular roulette wheel equal in area to the individual's fitness



Example: Problem Encoding

Consider a **black box switching problem** which concerns a black box device with a bank of five input switches. For every setting of the five switches, there is an output signal f; mathematically f = f(s) where s is a particular setting of the five switches. The objective of the problem is to set the switches to obtain the maximum possible f value.

$f(s) = s^2$ on the integer interval [0,31]



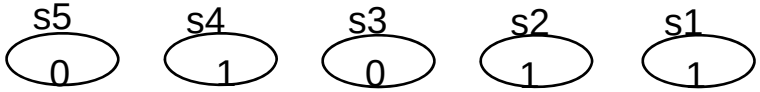
s5 s4 s3 s2 s1

s = s5*16+s4*8+s3*4+s2*2+s1*1

Population

00110	36
01010	100
10000	256
01001	81
00011	9
00100	16
00001	1
01100	144
11000	576
01110	196

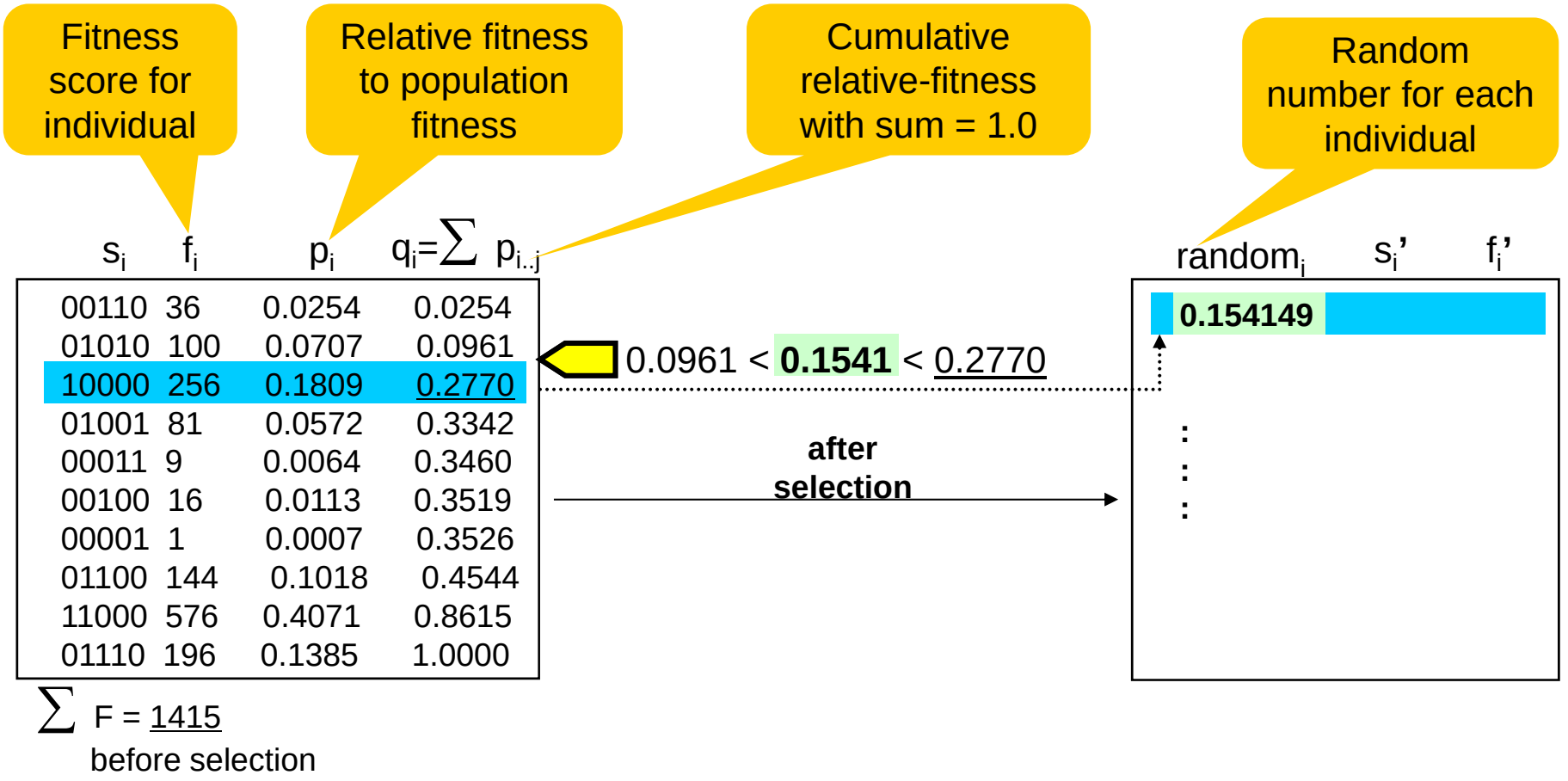
Chromosome



s = 0*16+1*8+0*4+1*2+1*1 = 11

f(s) = 11*11 = 121 (fitness value)

Example: Selection (Roulette Wheel)



Example: Selection (Roulette Wheel)

Generation 0

Probability distribution calculation based on fitness values

s_i	f_i	p_i	$\sum p_{i..j}$
00110	36	0.0254	0.0254
01010	100	0.0707	0.0961
10000	256	0.1809	0.2770
01001	81	0.0572	0.3342
00011	9	0.0064	0.3460
00100	16	0.0113	0.3519
00001	1	0.0007	0.3526
01100	144	0.1018	0.4544
11000	576	0.4071	0.8615
01110	196	0.1385	1.0000

$\sum$ F = 1415
before selection

0.0961 < **0.1541** < 0.2770

after selection

$random_i$	s_i'	f_i'
0.154149	10000	256
0.984649	01110	196
0.776116	11000	576
0.568041	11000	576
0.147954	10000	256
0.677175	11000	576
0.373852	01100	144
0.657063	11000	576
0.509262	11000	576
0.881771	01110	196

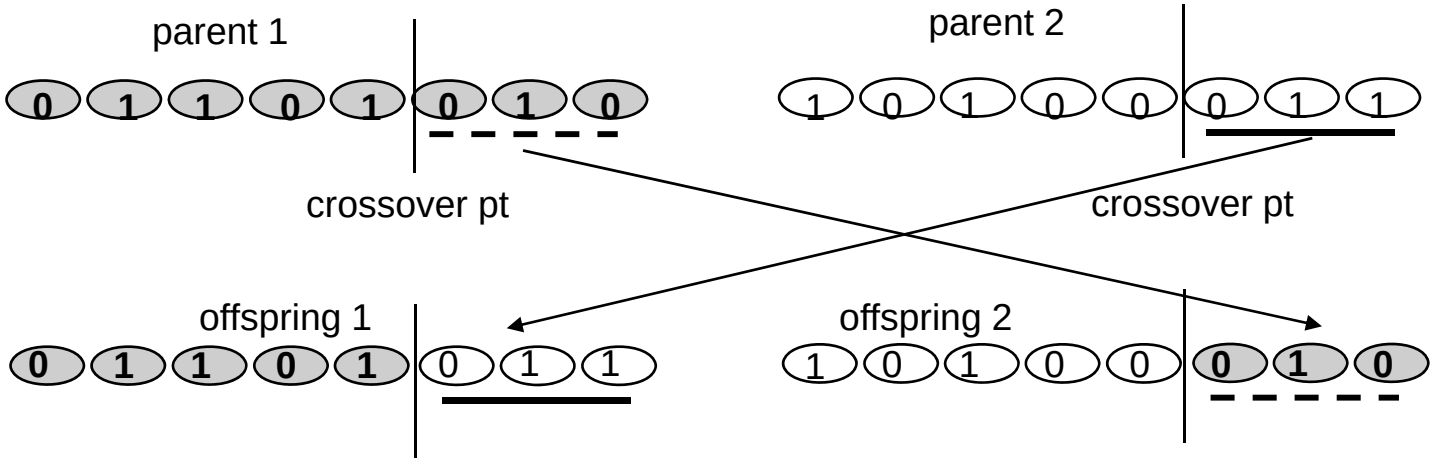
$\sum$ F' = **3928**
after selection

Select fitter chromosomes for a new population based on random number generation

This represents an improvement of +2513

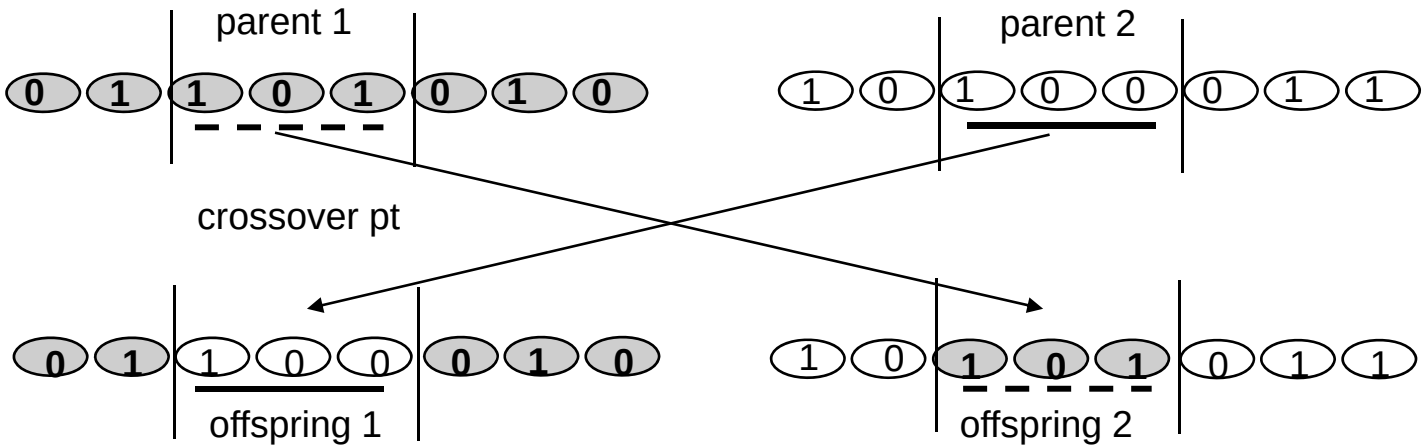
Crossover Operators

- Segments are cut-and-spliced between the chromosomes (strings)
  - Segments of two parent chromosomes are swapped producing two offsprings
  - Crossover site chosen randomly
- Single-point crossover



Crossover Operators

- Multi-point crossover (k-point crossover)

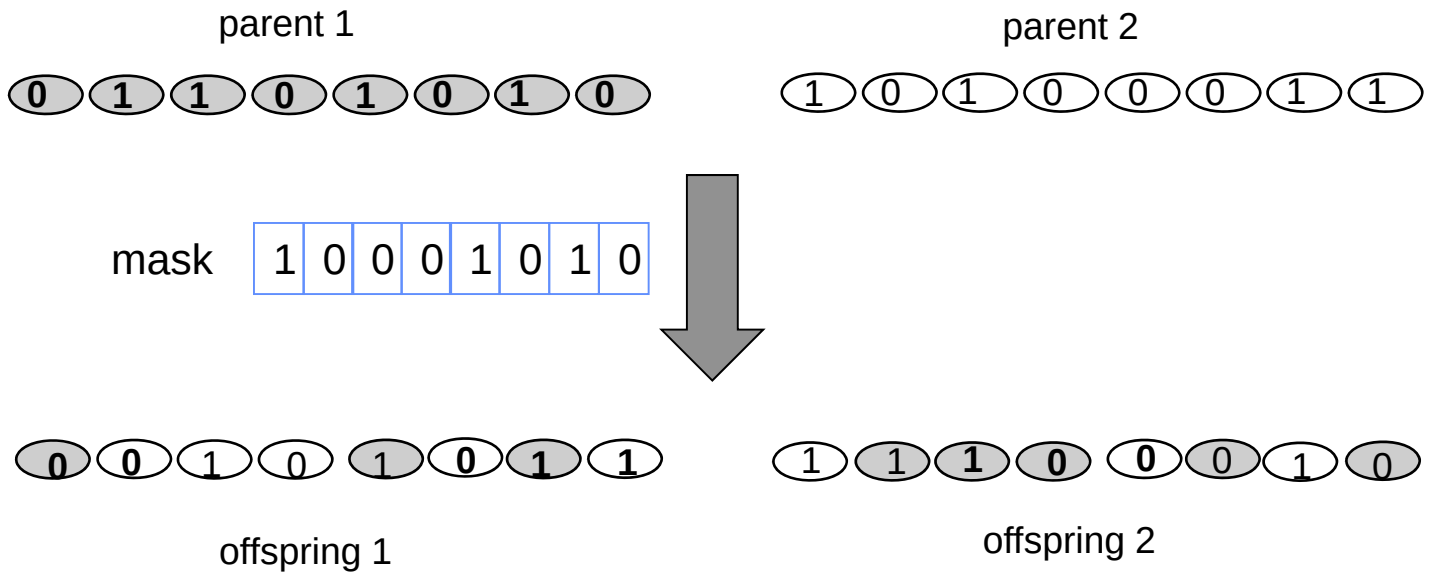


- Traditionally, the number of crossover points has been fixed at a very low constant value of 2.

Crossover Operators

- Uniform crossover

  - Genes are copied from the first parent or from the second parent based on a randomly created mask



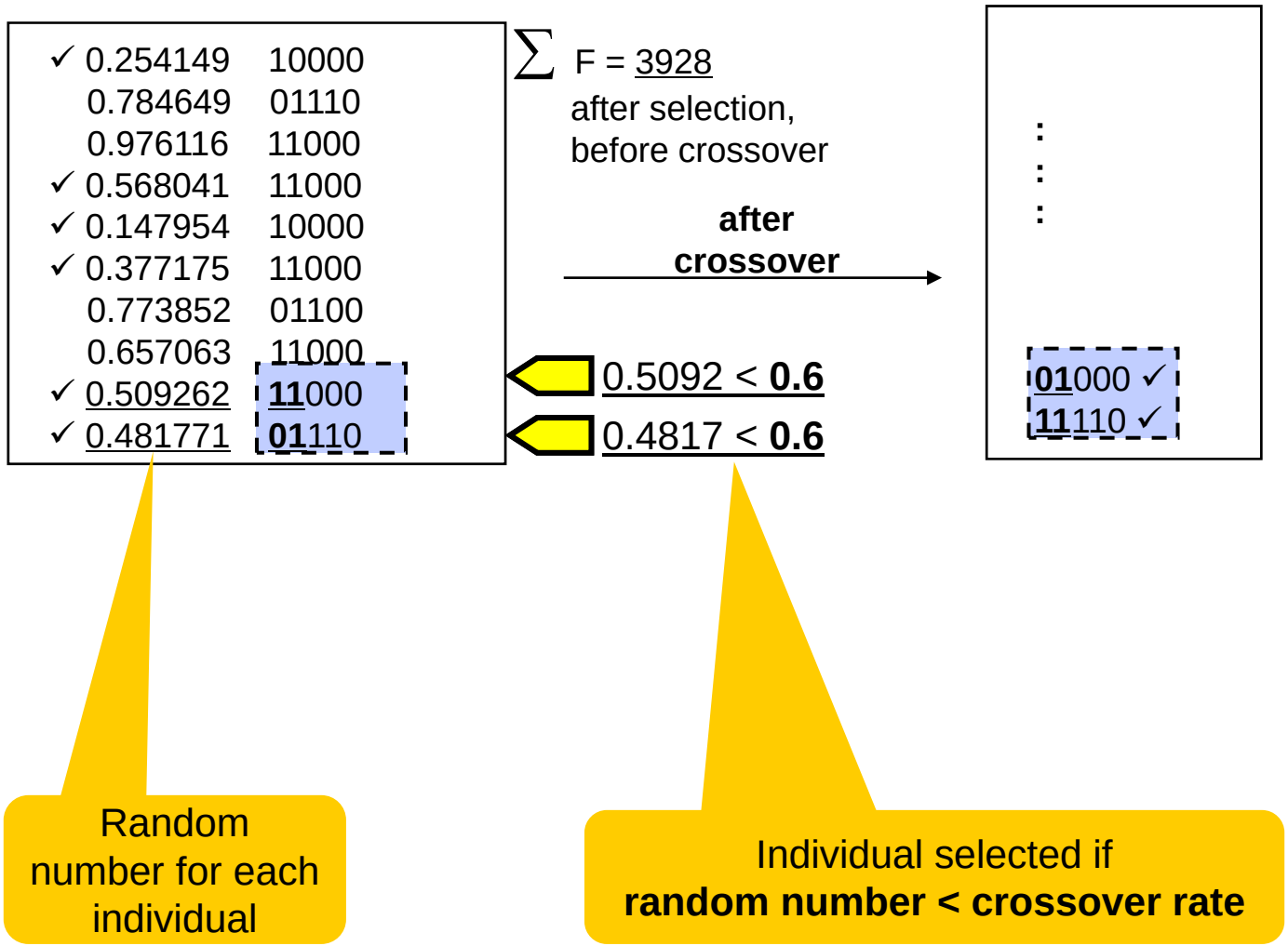
- Bear in mind that there are **other kinds of crossover operators** besides point-based ones

Crossover Rate

- The probability of crossover (crossover rate) p_c gives the expected number $p_c \times$ population size of chromosomes to undergo the crossover operation.
  - Generate a random number r from the range [0..1]
  - If $r < p_c$, select given chromosome for crossover.
  - For each pair of coupled chromosomes we generate a random integer number (the position of the crossover point) from the range [1..m-1] where m is the number of bits in a chromosome
    - The parent chromosomes are replaced by the offspring chromosomes.

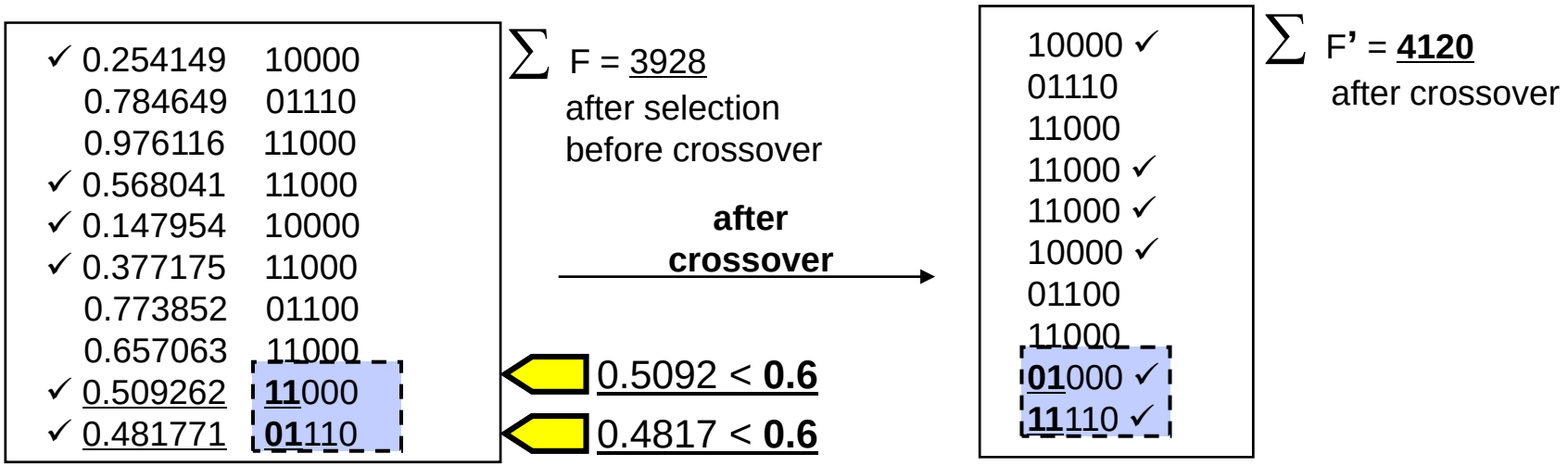
Example: Crossover Operators

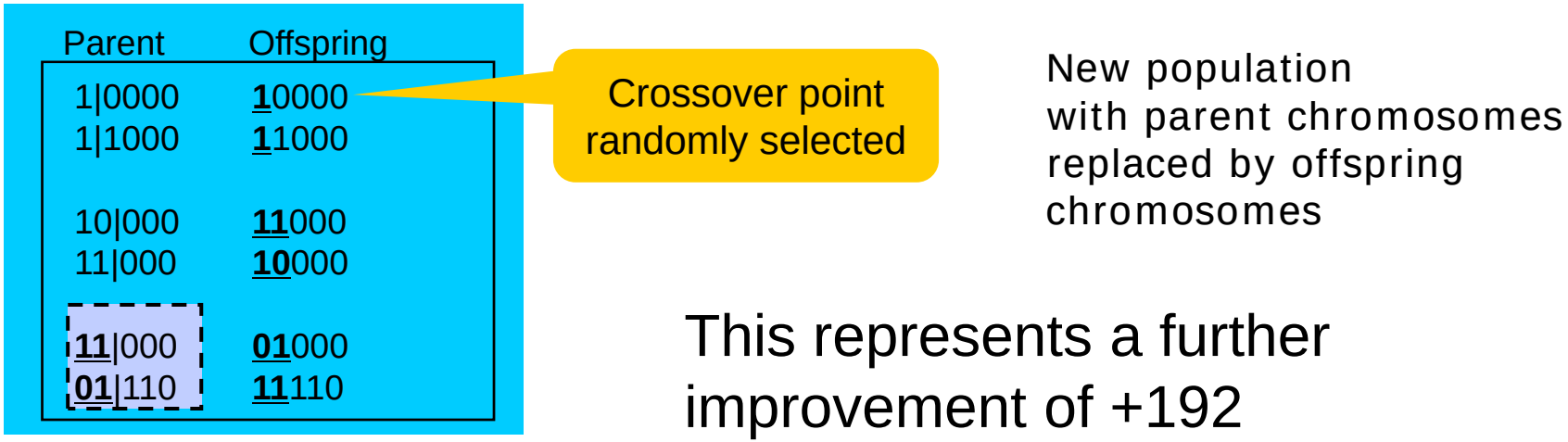
crossover rate= 0.6



Example: Crossover Operators

crossover rate= 0.6



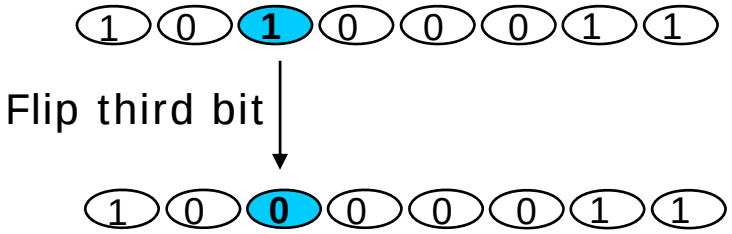


New population
with parent chromosomes
replaced by offspring
chromosomes

This represents a further
improvement of +192

Mutation Operators

- Replace the value at some randomly-chosen position by a new arbitrary value
  - The role of mutation is to maintain genetic diversity
  - A range of mutation operators have been proposed, ranging from completely random alterations to more heuristically motivated local search operators



Mutation

  - Bear in mind that there are **other kinds of mutation operators** besides bit-flipping

Mutation Rate

- Applying mutation operator to the individuals in the new population
  - The probability of mutation (mutation rate) p_m gives the expected number of mutated bits $p_m \times$ population size $\times$ number of bits in a chromosome
  - Every bit (in all chromosomes in the whole population) has an equal chance to undergo mutation
    - Generate a random number r from the range [0,1]
    - If $r < p_m \rightarrow$ mutate the bit

Example: Mutation Operators

mutation rate = 0.1

Chromosomes

s_1	10000
	01110
	11000
	11000
	11000
	10000
	01100
	11000
	01000
s_{10}	11110

$\sum$ F = 4120
after crossover, before mutation

after mutation

s_1'	
	:
	:
	1 1100 784
	:
s_{10}'	

Random numbers **for each gene**

s_1	0.228797	0.607837	0.151128	0.350780	0.151189
	0.387310	0.760491	0.915616	0.569292	0.922483
	0.405988	0.368755	0.612964	0.865047	0.507889
	0.823206	0.323008	0.454573	0.762017	0.447615
	0.924039	0.324229	0.847530	0.968780	0.443434
	0.269478	0.245827	0.839442	0.331889	0.010773
	0.077120	0.353801	0.242561	0.481948	0.360790
	0.491348	0.624195	0.674978	0.605823	0.770440
	0.285440	0.572649	0.137638	0.893033	0.585803
s_{10}	0.200568	0.167446	0.461562	0.409742	0.329386

Gene selected **random number < mutation rate**

0.07712 < **0.1**

Example: Mutation Operators

mutation rate = 0.1

Chromosomes

s_1	10000
	01110
	11000
	11000
	11000
	10000
	01100
	11000
	01000
s_{10}	11110

$\sum$ F = 4120 after crossover, before mutation

after mutation

s_1'	10000	256
	01110	196
	11000	576
	11000	576
	11000	576
	10001	289
	11100	784
	11000	576
	01000	64
s_{10}'	11110	900

$\sum$ F' = **4793** after mutation compared to

$\sum$ F = 1415 before selection (many slides ago)

New population
Fitness is evaluated

s_1	0.228797	0.607837	0.151128	0.350780	0.151189
	0.387310	0.760491	0.915616	0.569292	0.922483
	0.405988	0.368755	0.612964	0.865047	0.507889
	0.823206	0.323008	0.454573	0.762017	0.447615
	0.924039	0.324229	0.847530	0.968780	0.443434
	0.269478	0.245827	0.839442	0.331889	0.010773
0.07712 < **0.1**	**0.077120**	0.353801	0.242561	0.481948	0.360790
	0.491348	0.624195	0.674978	0.605823	0.770440
	0.285440	0.572649	0.137638	0.893033	0.585803
s_{10}	0.200568	0.167446	0.461562	0.409742	0.329386

Gene selected **random number < mutation rate**

0.0107 < **0.1**

This represents a further improvement of +673

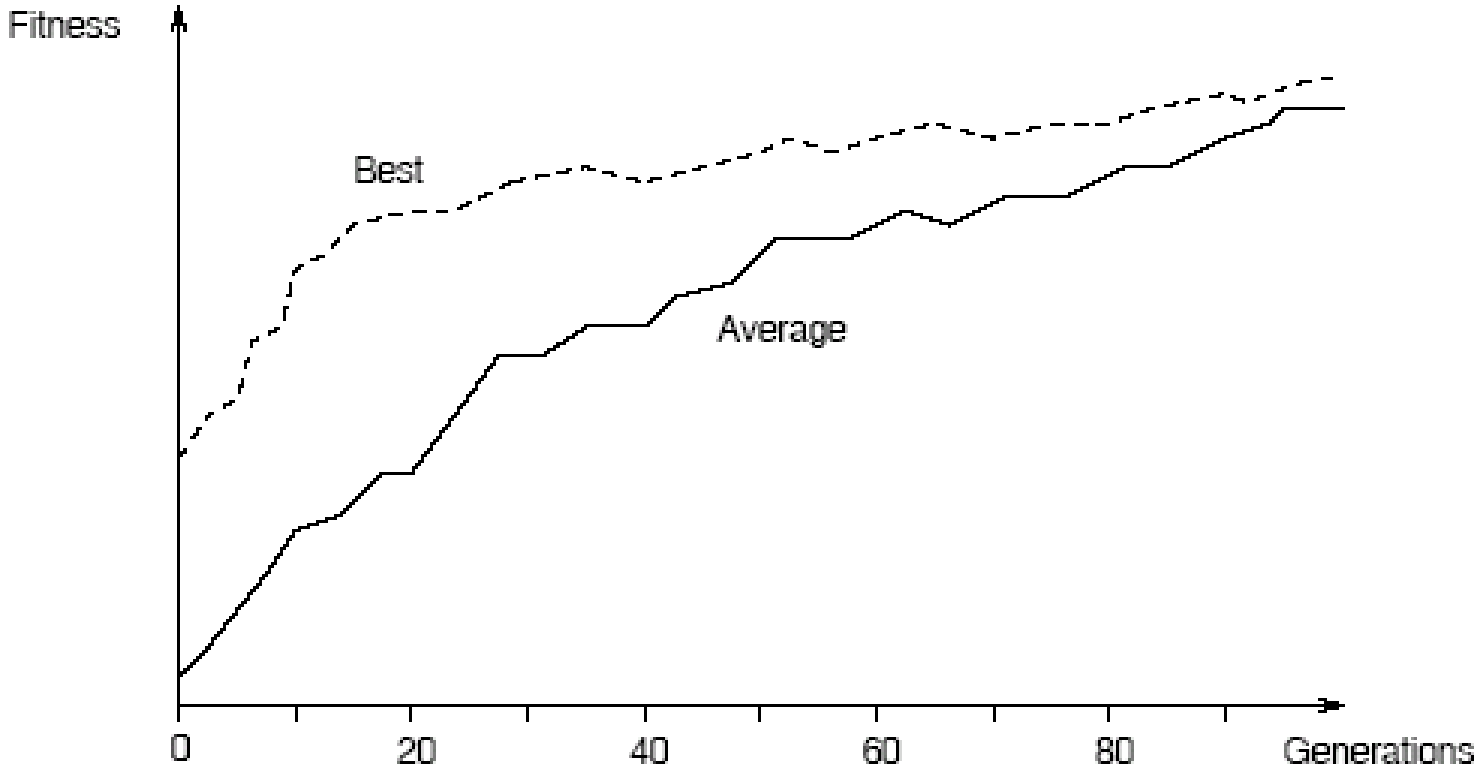
GA Evolution

- Following selection, crossover and then mutation, the new population is ready for its next evaluation.
- The rest of evolution is just cyclic repetition of these steps.
- The total fitness of the new population is higher than total fitness of the previous generation – or so, one would hope ☺
- A stopping criterion must be specified:
  - After a fixed number of generations
  - After a chromosome with a certain high fitness value is located
  - After all the chromosomes in the population have attained a certain degree of homogeneity

Parameters for GAs

- GA's performance in problem solving will depend on
  - The method for encoding candidate solutions
  - The method for selecting the individuals in the population to reproduce for the next generation
  - Choices of genetic operators
  - The parameter settings
    - Population size
    - Probability of crossover (crossover rate) gives the probability that a pair of chromosomes will be combined
    - Probability of mutation (mutation rate) gives the probability that a bit will be flipped.

A Sample GA Run



Taken from: Beasley, D. R. Bull, R. R. Martin. An Overview of Genetic Algorithms: Part 1, Fundamentals. University Computing 15:4 (1993) 170-181.

When & How to Use GAs?

- When would you want to apply GA on a problem?
  - No idea how to reasonably solve the problem
  - You cannot enumerate all possible solutions
  - You know how to evaluate how good or bad a solution is
- What do we need to use Genetic Algorithms?
  - A **goodness measure**

    → this is eventually translated into the fitness function
  - A **representation** of the solution

    → the more natural, the better
  - A **problem model** which defines the behaviour

Some Caveats and Issues

- Can a GA converge to a poor solution?
  - In a word, YES!
  - Possible reasons include poor problem representation, premature convergence, a poor fitness evaluation algorithm, or bad luck of the random numbers could generate a poor solution
- How do you know whether a GA solution is optimal (or very near optimal)?
  - If you knew how to find the optimal solution, you would not need to use a GA! Remember, these problems are often *very* hard to solve.
  - There is no guarantee that a GA will find an optimal solution.

Applications of GA

- GAs can be applied to a wide range of optimization and learning problems
  - Routing and scheduling, machine vision, engineering design optimization, gas pipeline control systems, machine learning
  - Hundreds of applications have now been discovered and a variety of commercial software tools have been introduced.

- Numerical function optimisation
  - GA has been shown to outperform conventional optimisation techniques on difficult, discontinuous, multimodal, noisy functions.

Applications of GA

- Combinatorial optimisation
  - Resource allocation problems
  - Classical problem of travelling salesperson / bin-packing / job shop scheduling
  - For example:
    - Assign jobs to machines over time such that the machines' idle time and the jobs' throughput time are minimized
    - Timetabling of examinations or classes in the universities, colleges

Applications of GA

- Financial forecasting
  - Searching for a set of rules or equations that will predict the ups and downs of a financial market, such as that for foreign currency and stocks.
  - Hypothesis: linear combination of technical indicators can be used to forecast periods of rising price movement in the stock market.
  - The contribution of each indicator in a forecasting model is indicated by a weighting coefficient.
  - The coefficients are derived by a genetic algorithm.
  - The resulting models are evaluated against historical price movement of the stocks.

Applications of GA

- Design optimisation
  - network routing
    - To maximise throughput for given bandwidth (first objective) while minimizing cost (second objective).
    - Use of network simulation tools to determine bandwidth utilisation and throughput.
  - Satellite orbit selection
    - Avoid collision and reduce blackout window

Applications of GAs

- Machine learning
  - Used to evolve aspects of particular machine learning systems, such as weights for neural networks, rules for learning symbolic production systems, and sensors of robots.
  - Classifier systems which evolve if-then rules for a specific problem domain
    - For example: rule induction for financial decision making

Applications of GAs

- Designing satellite antenna (NASA Space Technology 5)
  - Encode antenna structure into a genome and use a GA to evolve an antenna that best meets the desired antenna performance as defined in a fitness function.
- Generating self-animating characters
  - Dynamic Motion Synthesis
  - Breeding computer characters which walk ‘naturally’ without any supervision
  - Used in conjunction with neural networks – hybrid system

Pattern Recognition Example

Criminal suspect recognition using ***Faceprints***

- Developed by psychology department of New Mexico State University
- Uses GAs to aid witnesses in the identification of criminal suspects
- Easy to identify criminal from photo but hard to describe features
- Difficult to generate even from computer library of visual features

Pattern Recognition Example

- GA approach:
  - Randomly generate 20 faces on a computer screen
  - Witness rates each face on a 10 point scale
  - GA generates additional faces from 5 building blocks: eyes, mouth, nose, hair and chin. Each is a 7 bit string.
  - The five features are coded as a 35-bit binary string consisting of five 7-bit parameters (34 billion faces are possible)
  - Witness evaluates successive generations with 10 point scale. That is, the witness's memory acts as the fitness function.
  - Facial features are iteratively refined as the witness rewards and punishes a range of facial attributes
  - Convergence often occurs after 20 generations

Exercise

Use the simple Visual Basic GA program provided to solve the example problem in this lecture. You are required to observe:

- how the fitness of the population progress over each generation?
- how many generations are required to reach a stable state?
- is the optimal solution achievable?
- the degree of homogeneity of the chromosomes?

Experiment with the following parameters:

- the population size
- the crossover rate
- the mutation rate

Exercise

The user interface allows you to change the population size, crossover rate and mutation rate before initializing the first generation of population.

The steps for generating the next population can be observed by clicking the Selection button, then the Crossover button. During the crossover process, you can see the two selected parent chromosomes are combined to give two offspring chromosomes.

You can opt to step through each of this combination using the Next button or continue to the mutation process. Select the Mutation button to watch the chromosomes being mutated.

Exercise

Subsequently you can select:

- the Next button which will show you each generation of population,
- the StepIn button which is stepping through the above process,
- the Stop button to get the best current result, or
- the Reset button to start all over again with a different set of parameters.

Appendix: GA Tools

GA Tools

- Matthew's GAlib, A C++ Library of Genetic Algorithm Components developed at MIT Technology Center
  - The library includes tools for using genetic algorithms to do optimization in any C++ program using any representation and genetic operators.
  - http://lancet.mit.edu/ga/

- JGAP, Java Genetic Algorithms Package
  - a Genetic Algorithms and Genetic Programming component provided as a Java framework.
  - Easy to use, highly modular
  - Can plug in custom genetic operators
  - .Net version available
  - http://jgap.sourceforge.net/

GA Tools

- Excel-based GA tools
  - Excel Solver
    - http://www.solver.com/
  - Evolver
    - http://www.palisade.com/evolver/
  - SolveXL
    - http://www.solvexl.com/
  - GeneHunter
    - http://www.wardsystems.com/genehunter.asp

GA Tools

- R packages for GA:
  - Genalg
  - GA
  - RGP
  - ……
- Python packages for GA:
  - DEAP
  - Pyevolve
  - Pyvolution
  - ……