REASONING SYSTEMS
DAY 3

Dr Zhu Fangming
Institute of Systems Science
National University of Singapore
fangming@nus.edu.sg

DAY 3 AGENDA

3.1 Reasoning using Optimization Techniques – Evolutionary Computation and Genetic Algorithms

3.2 Evolutionary Computation and Genetic Algorithm Applications

3.3 Workshop – Genetic Algorithms

3.1
REASONING USING OPTIMIZATION TECHNIQUES – EVOLUTIONARY COMPUTATION AND GENETIC ALGORITHMS

Reasoning Systems

- **Problem Solving AI**

  Plan; Optimize; Search for solutions

- **Reasoning systems need conduct search and optimization to get optimal (sub-optimal) solutions.**
- **There are different types of optimization techniques available.**

Optimization Problems

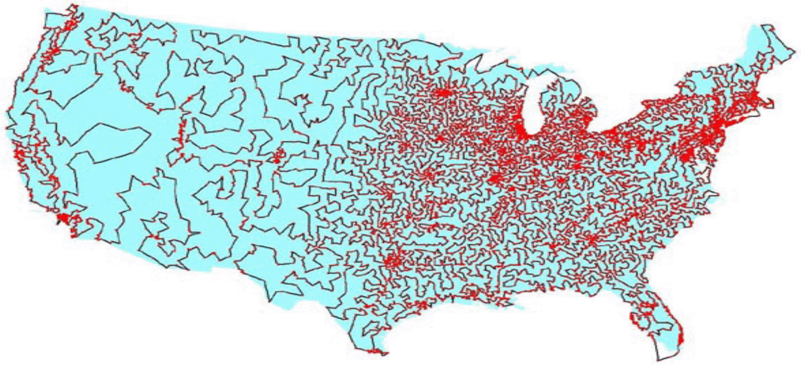


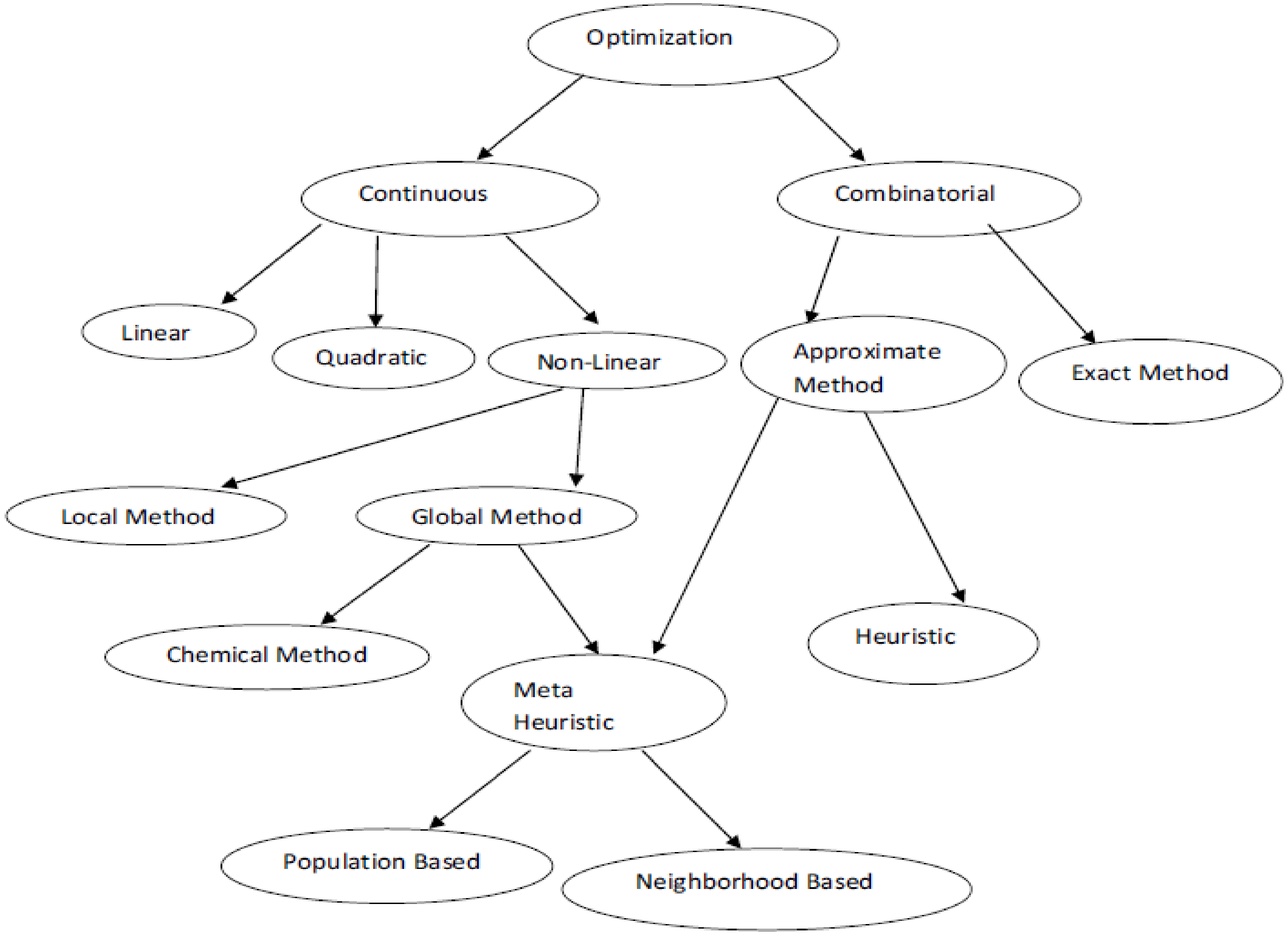
Figure credit: David Applegate, Robert Bixby, Vasek Chvatal and William Cook.

Businesses make pricing decisions to maximize profits

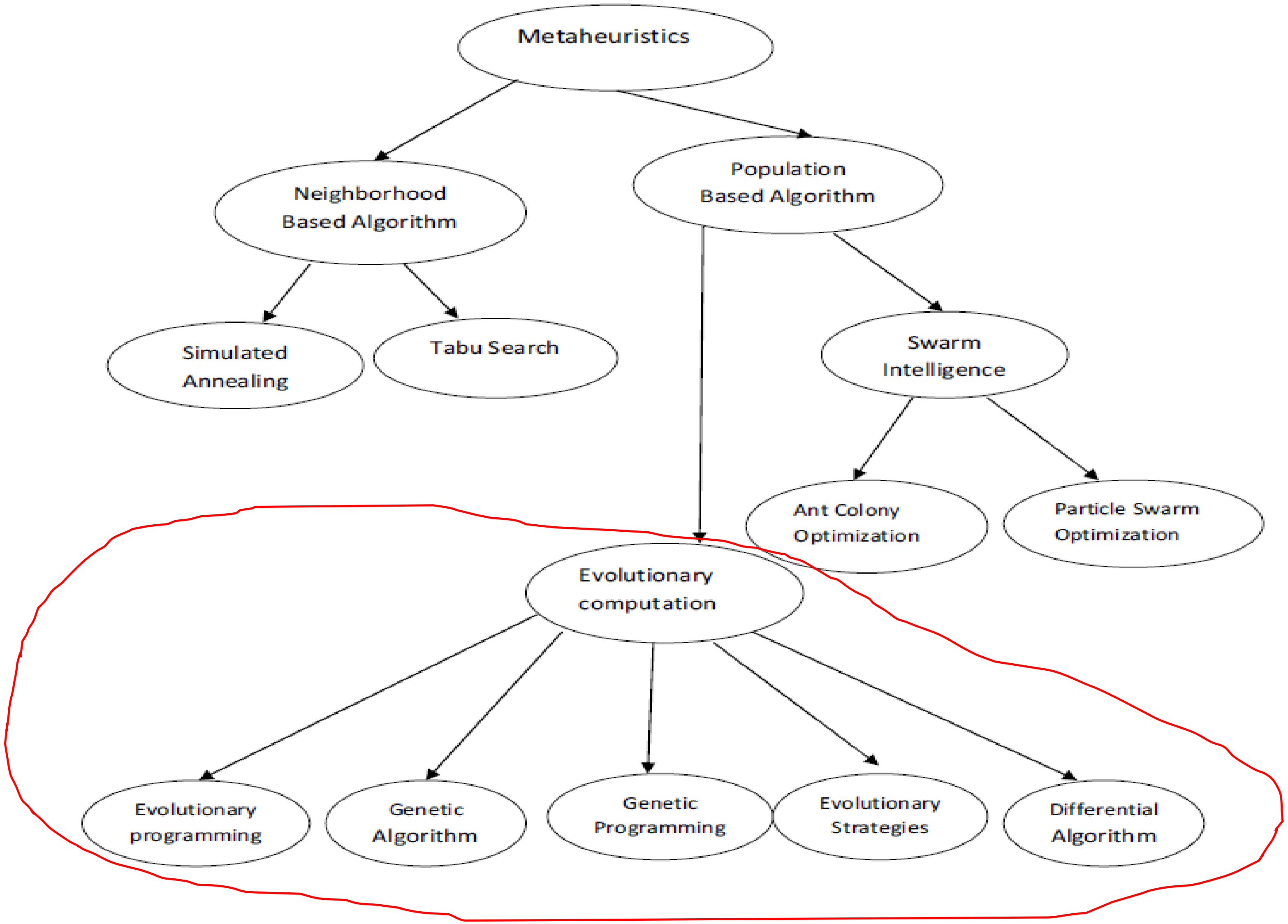
Consumers make investment decisions to maximize return

Travelling Salesman Problem (TSP) to minimize traveling distance

Optimization Techniques



Optimization Techniques



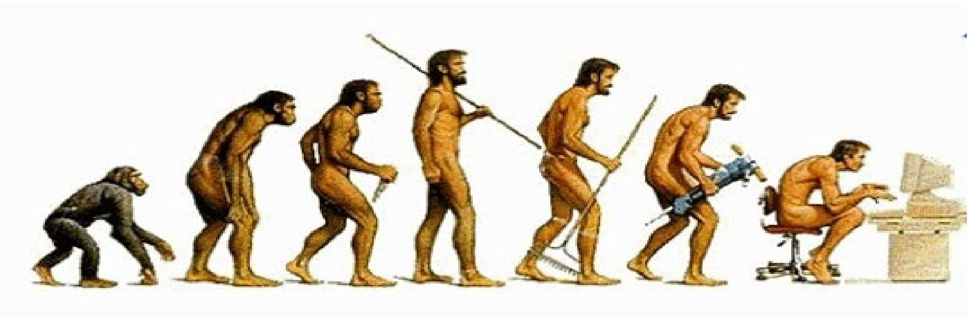
Introduction to Evolutionary Computation

"In computer science, **evolutionary computation** is a subfield of artificial intelligence (more particularly computational intelligence) that involves continuous optimization and combinatorial optimization problems. Its algorithms can be considered global optimization methods with a metaheuristic or stochastic optimization character and are mostly applied for black box problems (no derivatives known), often in the context of expensive optimization."

---Wikipedia

Darwinian Evolution

- **Inspired by Darwinian natural evolution:**
  - Survival of the fittest
  - Selection on phenotype through environment
  - Genotypic inheritance
  - Reproduction
  - Blind variation



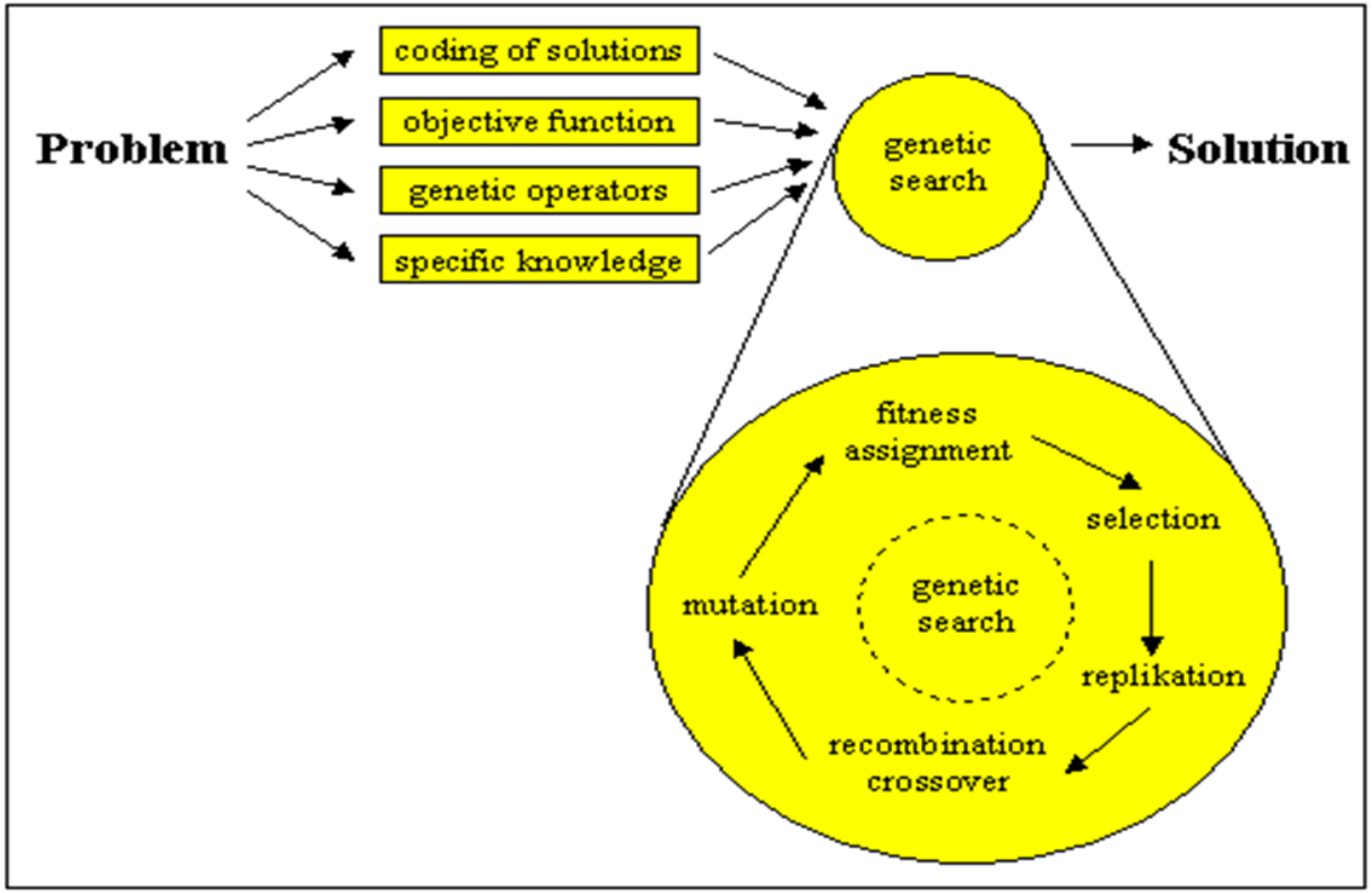
Evolutionary Computation Metaphor

EVOLUTION		PROBLEM SOLVING
Environment	⟷	Problem
Individual	⟷	Candidate Solution
Fitness	⟷	Quality

Evolutionary Algorithms History

- Evolutionary Programming
  - L. Fogel 1962 (San Diego, CA)
- Genetic Algorithms
  - J. Holland 1962 (Ann Arbor, MI)
- Evolution Strategies
  - I. Rechenberg & H.-P. Schwefel 1965 (Berlin, Germany)
- Genetic Programming
  - J. Koza 1989 (Palo Alto, CA)

Problem Solution Using Evolutionary Algorithms



Genetic Algorithms – As typical EC 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

Old Population

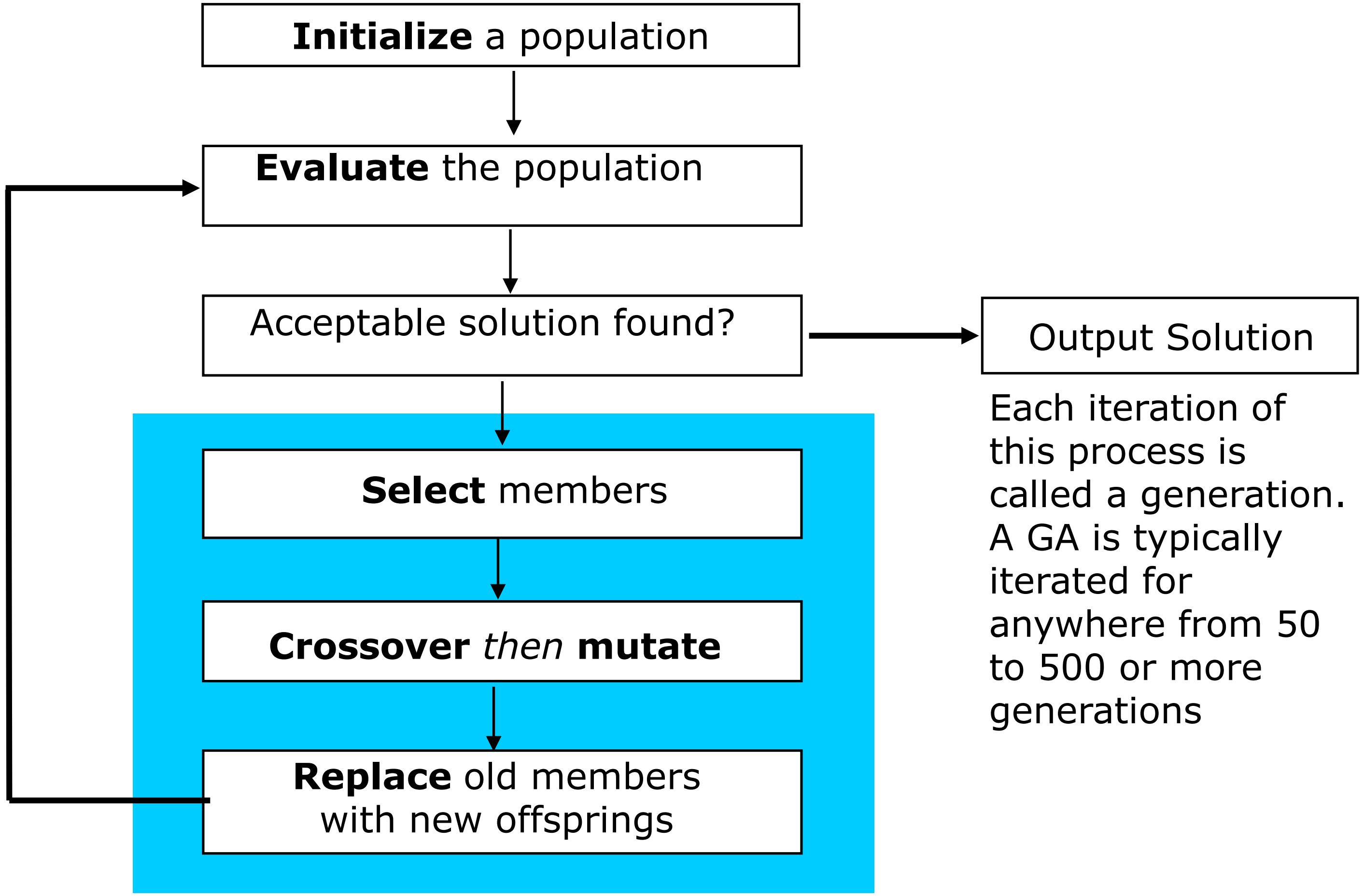
Solution 1
Solution 2
Solution 3
Solution 4
Solution 5
Solution 6
Solution 7
Solution 8
Solution 9
Solution 10

Selection, crossover,
mutation
and
survival

Solution’ 1
Solution’ 2
Solution’ 3
Solution’ 4
Solution’ 5
Solution’ 6
Solution’ 7
Solution’ 8
Solution’ 9
Solution’ 10

New Population
consisting of old
solutions and new
solutions through
breeding

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 Pseudocode

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;

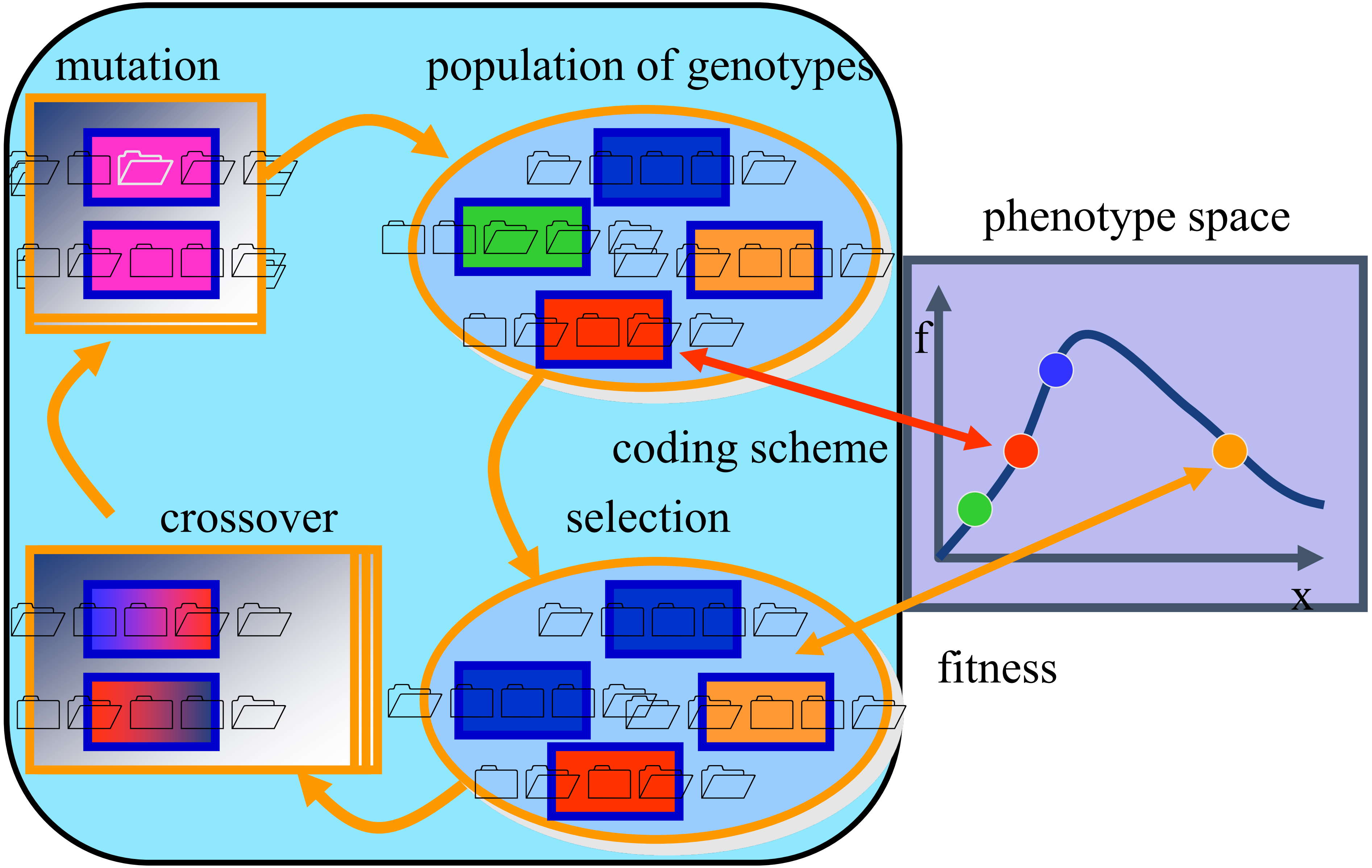
Applying crossover and mutation to parents to create O(t);

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

0101001000010111100001100100011110001
5 6 1 9 8 3 7 4 2 10
14.5 79.0 22.8 9.3

Core GA Concepts

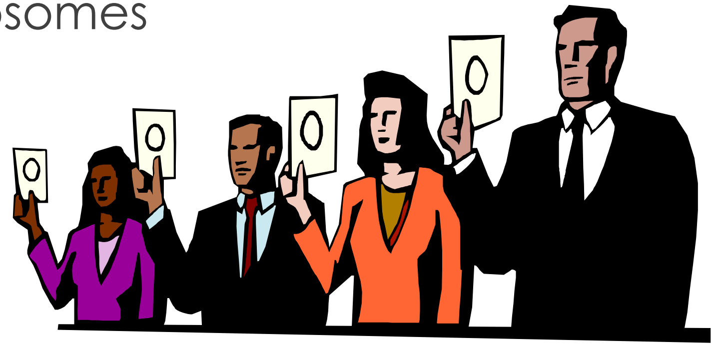
- **Initial population**
  - Can be initialized using whatever knowledge is available about the possible solutions
  - Should represent a random sample of the search space
  - Each member of the population is evaluated and assigned a measure of its fitness as a solution
- **New population**
  - Structures 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

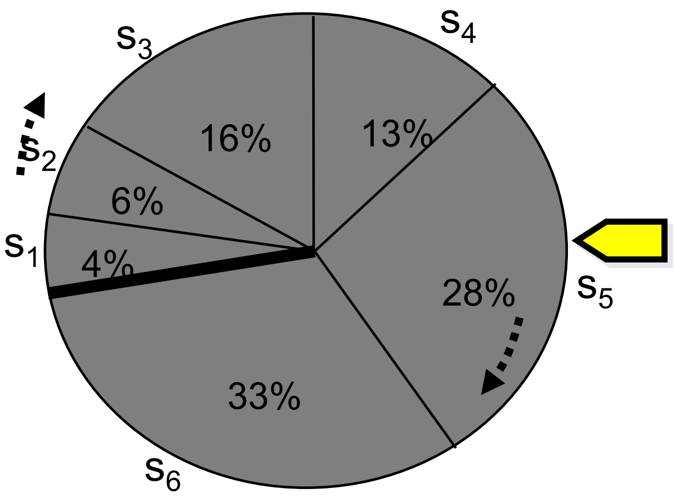
- Selection and reproduction in GA always in the given sequence
  1. **Select** fitter solutions (or individuals) to contribute genetic material to the next generation
     → selection method
  2. **Crossover** or recombine genetic materials from two parents to create new offsprings
     → crossover 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

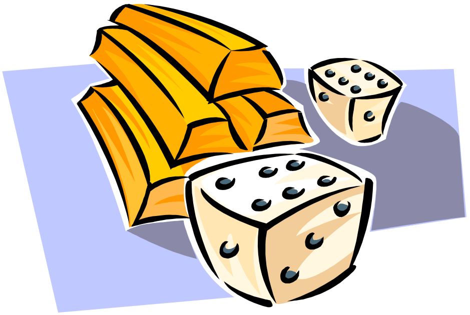
- **Fitness-Proportionate Selection with "Roulette Wheel" *aka* Wheel of Fortune**
  - Select new population with respect to the probability distribution based on fitness values
  - A roulette wheel with slots sized according to the fitness is used
    - Calculate the fitness value $f(s_i)$ for each 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 $q_{i-1}<r<\mathbf{q_i}$



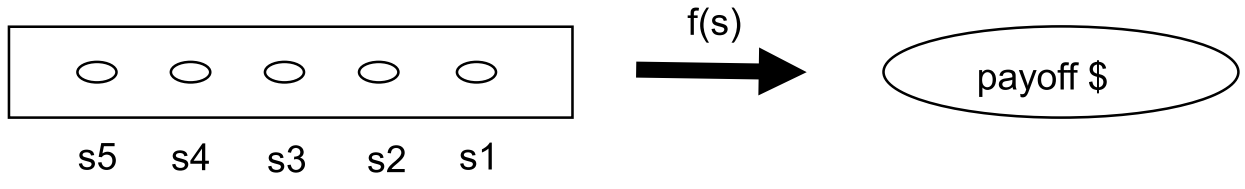
A “roulette-wheel sampling” giving each individual a slice of a circular roulette wheel equal in area to the individual’s fitness



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.**

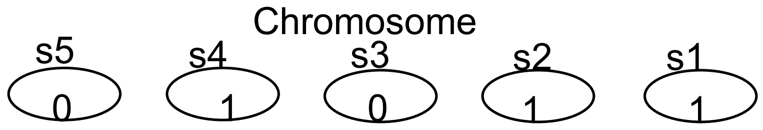
$f(s) = s^2$ on the integer interval [0,31]



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



s = 0*16+1*8+0*4+1*2+1*1 = 11
f(s) = 11*11 = 121 (fitness value)

Example: Selection (Roulette Wheel)

Fitness score for individual

Relative fitness to population fitness

Cumulative relative-fitness with sum = 1.0

Random number for each individual

s_i	f_i	p_i	$q_i=\sum p_{i..j}$
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		
:		
:		
:		

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**

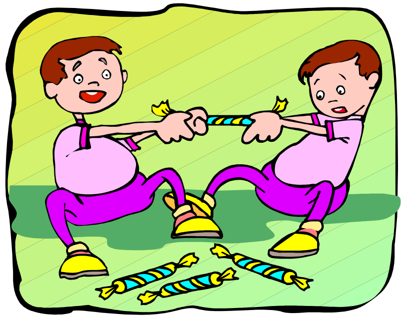
after selection

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

This represents an improvement of +2513

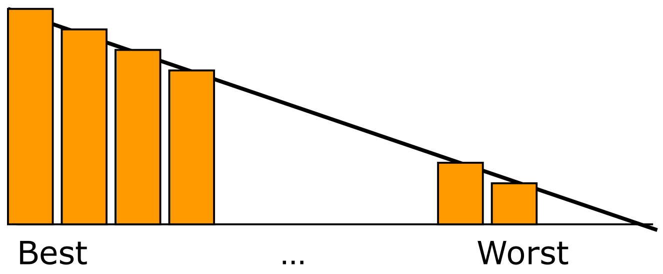
Other Selection Methods

- **Tournament selection method**
  - Choosing individuals for reproduction randomly.
  - **Binary tournament**
    - Two individuals are chosen at random and the better of the two individuals is selected.
  - **Larger or N-tournament**
    - N individuals are chosen at random from the population, with the best being selected for reproduction.



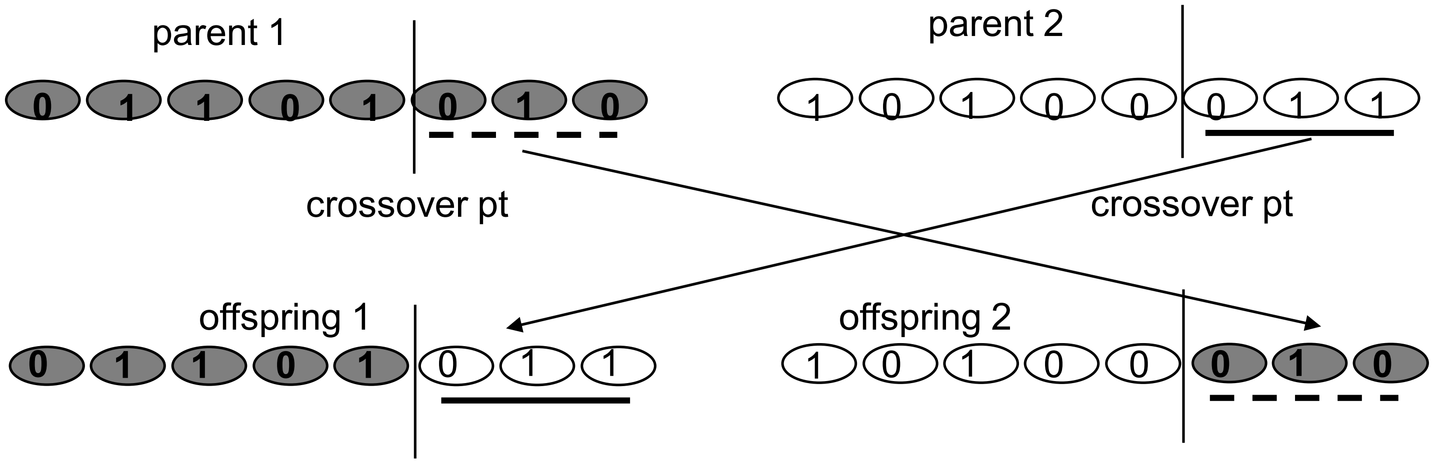
Other Selection Methods

- ## Rank selection method
  - The population is ordered according to the measured fitness values. A new fitness value is then ascribed, inversely related to their rank.
  - **Linear ranking**
    - Each chromosome in the population is ranked in increasing order of fitness, from 1 to N, and assigned a new subjective fitness using a linear ranking function.
  - **Exponential ranking**
    - First chromosome assigned fitness of 1; the second fitness of s (typically 0.99); the third fitness of s^2 and so on.



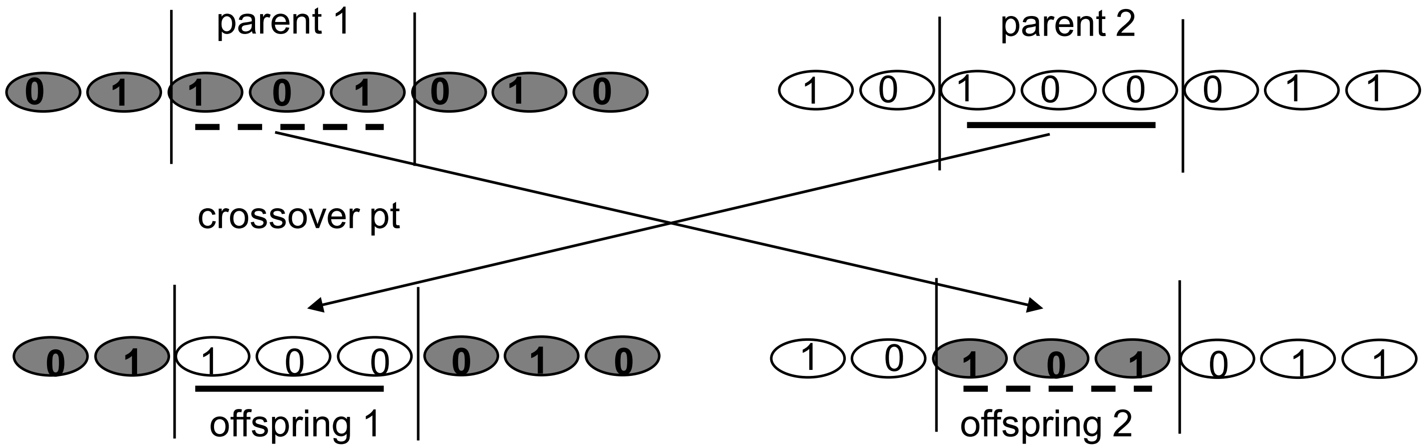
Crossover Operators

- **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)

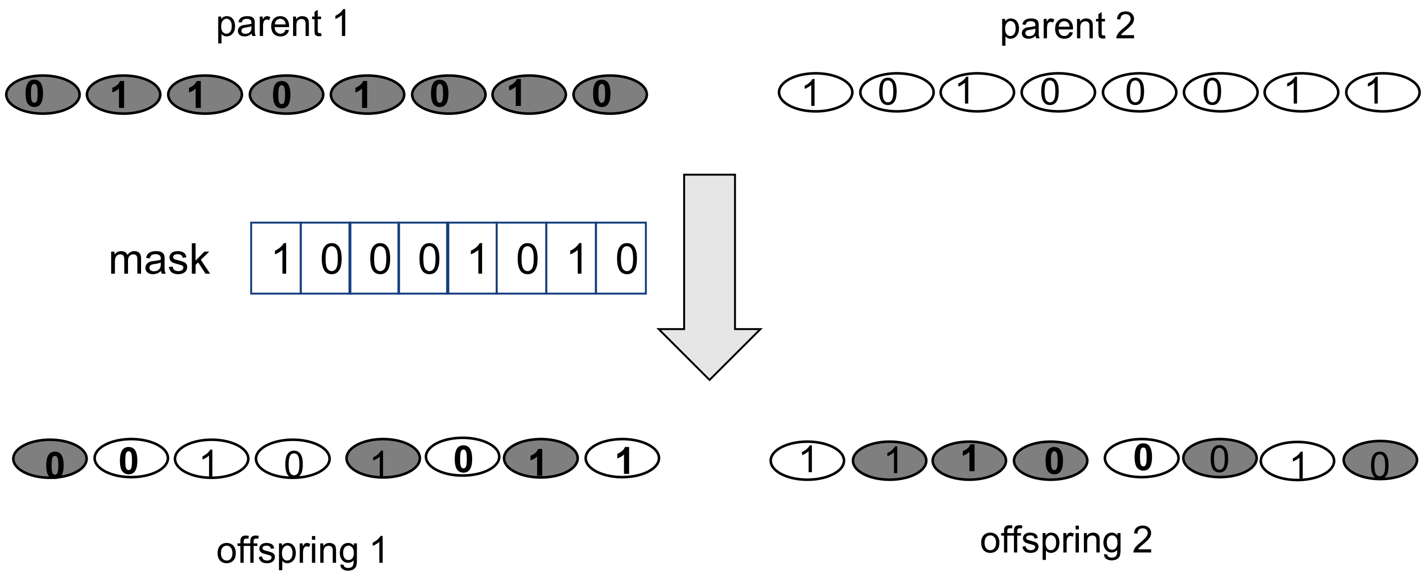


- Traditionally, the number of crossover points has been fixed at a very low constant value of 2.

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

✓	0.254149	10000
	0.784649	01110
	0.976116	11000
✓	0.568041	11000
✓	0.147954	10000
✓	0.377175	11000
	0.773852	01100
	0.657063	11000
✓	0.509262	11000
✓	0.481771	01110

$\sum$ F = 3928

after selection, before crossover

after crossover

0.5092 < **0.6**

0.4817 < **0.6**

:
:
:

01000 ✓
11110 ✓

Random number for each individual

Individual selected if **random number < crossover rate**

Example: Crossover Operators

crossover rate= 0.6

✓	0.254149	10000
	0.784649	01110
	0.976116	11000
✓	0.568041	11000
✓	0.147954	10000
✓	0.377175	11000
	0.773852	01100
	0.657063	11000
✓	0.509262	**11**000
✓	0.481771	**01**110

$\sum$ F = 3928
after selection
before crossover

after crossover

0.5092 < **0.6**

0.4817 < **0.6**

10000 ✓
01110
11000
11000 ✓
11000 ✓
10000 ✓
01100
11000
01000 ✓
11110 ✓

$\sum$ F' = **4120**
after crossover

Parent	Offspring
1|0000	10000
1|1000	11000
10|000	11000
11|000	10000
11|000	**01**000
01|110	**11**110

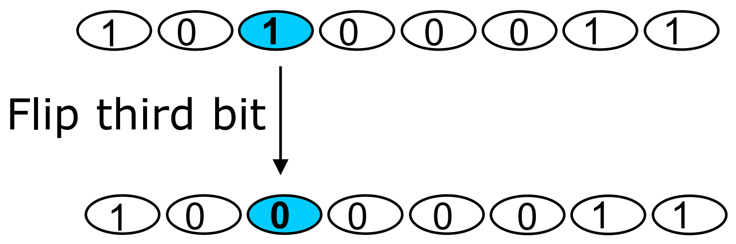
Crossover point randomly selected

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

	Before mutation (after crossover)		After mutation	Fitness
s_1	10000	s_1'	10000	256
	01110		01110	196
	11000		11000	576
	11000		11000	576
	11000		11000	576
	10000		1000<u>1</u>	289
	01100		<u>1</u> 1100	784
	11000		11000	576
	01000		01000	64
s_{10}	11110	s_{10}'	11110	900

$\sum$ F = <u>4120</u> after crossover, before mutation

after mutation →

$\sum$ F' = <u>**4793**</u> after mutation compared to

$\sum$ F = <u>1415</u> 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	<u>**0.010773**</u>
	<u>**0.077120**</u>	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**

<u>0.07712 < **0.1**</u>

<u>0.010773< **0.1**</u>

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

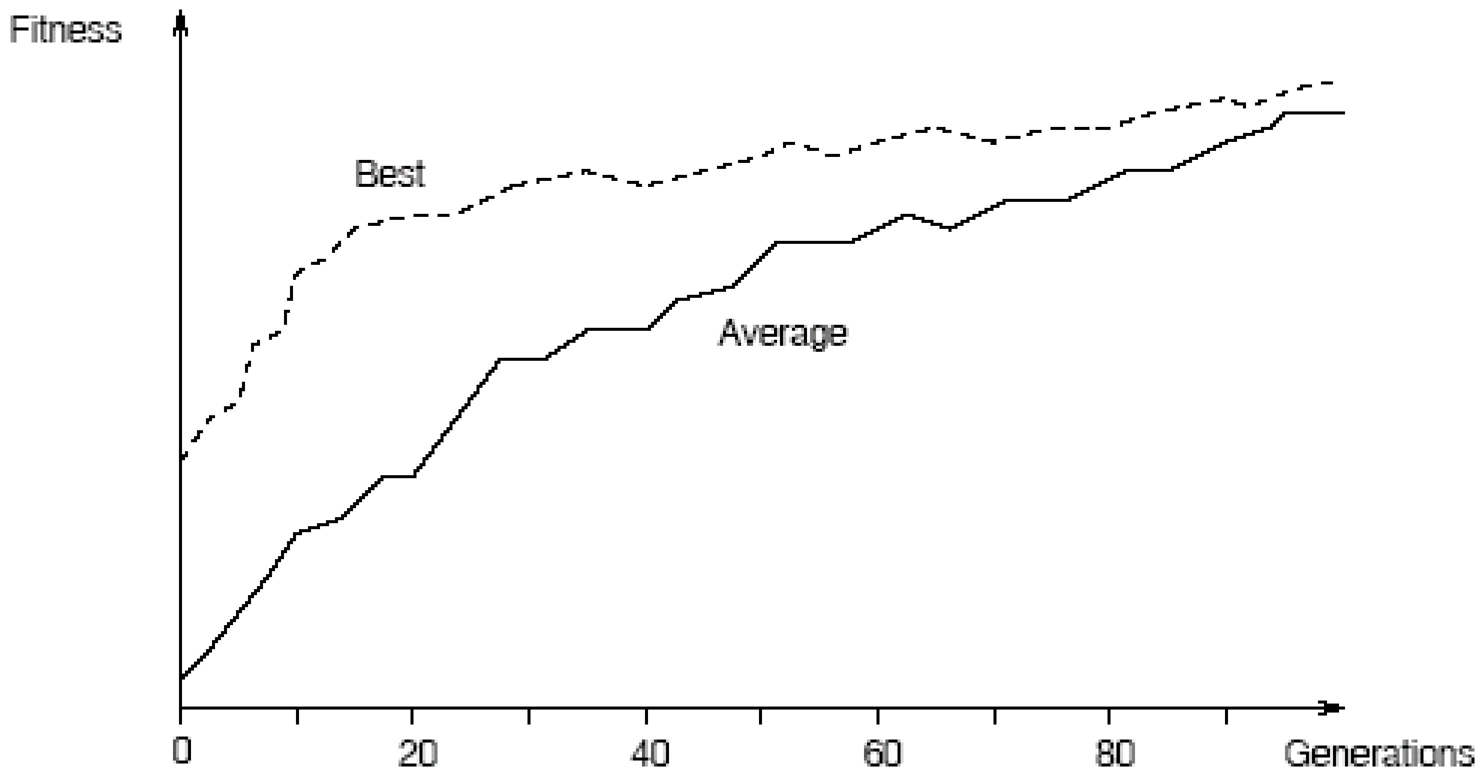
Parameters for GAs

- **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.

Trade-off Considerations on Control Parameters

- Increasing the crossover probability increases recombination of schemata, but it also increases the disruption of good chromosomes.
- Increasing the mutation probability tends to transform the genetic search into a random search, but it also helps re-introduce lost genetic material.
- Increasing the population size increases its diversity and reduces the probability that the GA will prematurely converge to a local optimum, but it also increases the time required for the population to converge to the optimal regions in the search space.

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

3.2
Evolutionary Computation and Genetic Algorithm Applications

Application Domains

- **Evolutionary Computation and Genetic Algorithms have been successfully used in various areas such as optimisation, learning and design.**
  - Numerical & Combinatorial Optimization
  - Engineering Design
  - Interactive Creative Design
  - Machine Learning
  - Scheduling and Control
  - Etc…

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 GA

- **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 GA

- **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

GA Tools

- **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/

GA Tools

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

3.3 WORKSHOPS

GA Workshop:
Solving Load Distribution Problem
Using Excel Solver

Problem Description

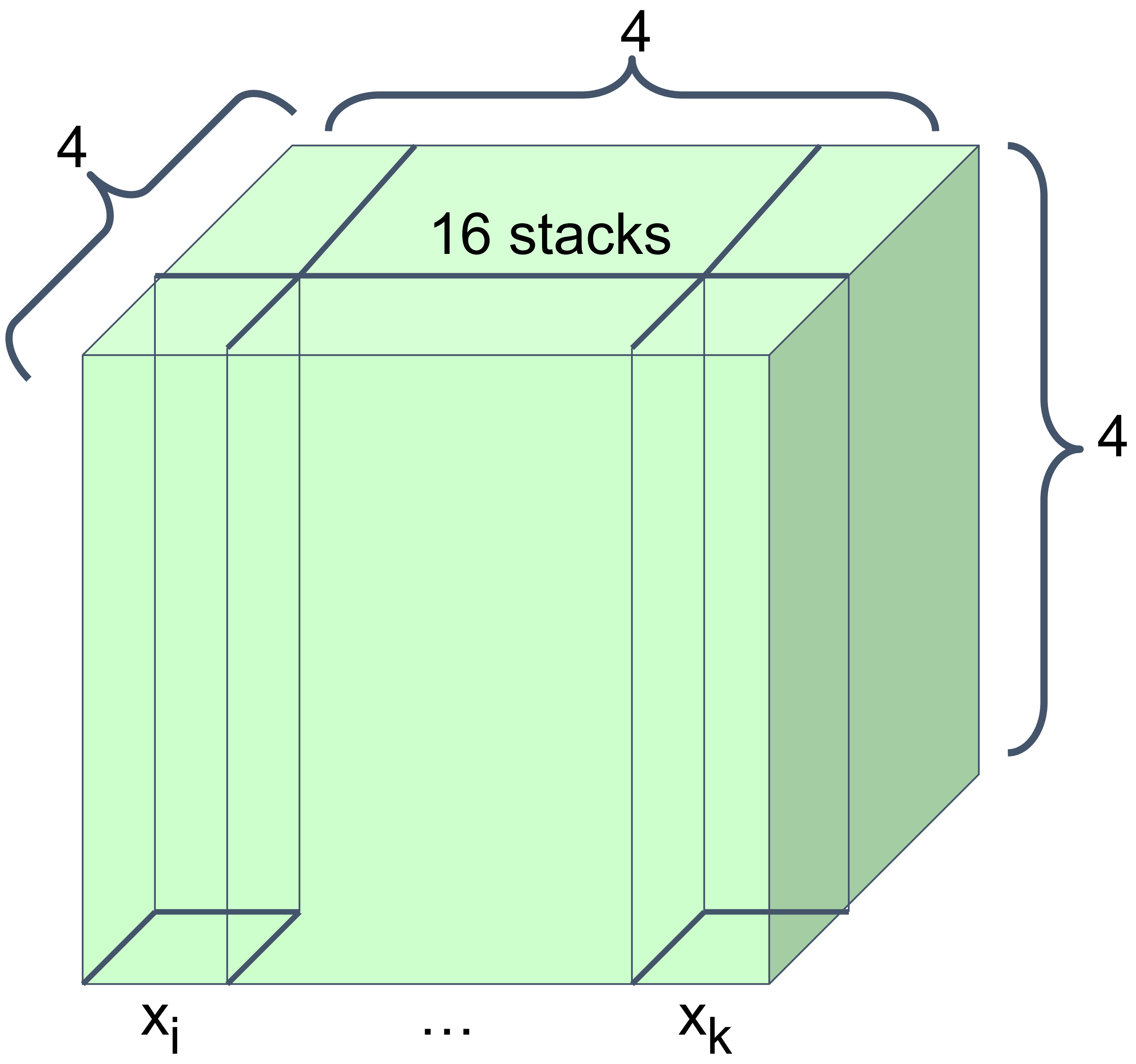
- In loading containers into a vessel or loading packages into an aircraft, the distribution of the weights should be more or less uniform.
- It is assumed the holding space for the containers or packages can be divided into 64 rectangular spaces.
- It is also necessary to ensure that the lighter containers or packages are on top of the heavier ones.
- The solution needs to determine how the packages are to be assigned to the rectangular spaces to ensure uniform distribution.

Problem Description

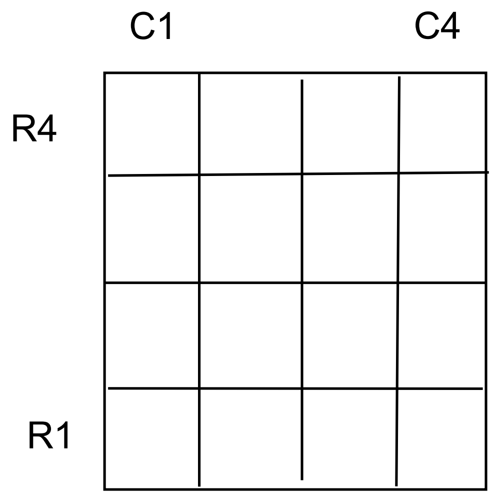
PackageID	Weight
1	27
2	57
3	33
4	55
5	65
6	26
7	21
8	28
9	43
10	40
11	27
12	62
13	25
14	33
15	67
16	30
17	46
18	31
19	58
20	59
21	37
22	55
23	61
24	25
25	38
26	65
27	51
28	30
29	40
30	69
31	33
32	61

PackageID	Weight
33	66
34	63
35	53
36	37
37	53
38	67
39	32
40	44
41	53
42	47
43	39
44	38
45	46
46	67
47	36
48	30
49	37
50	43
51	68
52	50
53	64
54	32
55	32
56	59
57	24
58	67
59	54
60	23
61	49
62	60
63	65
64	35

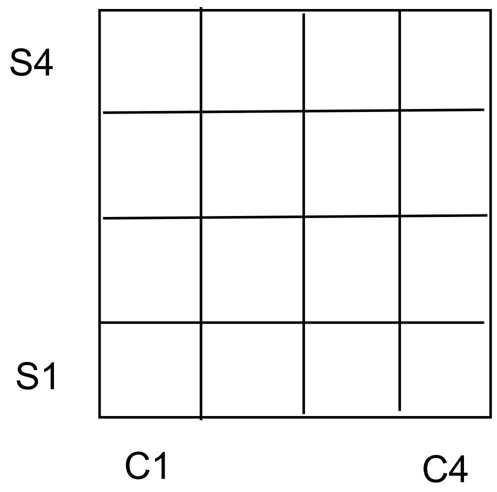
Graphical Representation



Graphical Representation



Looking from
the side of a
vessel



Looking from the
aeroplane view
of a vessel

Problem Model

1. Required information

- Supplied by client
  - Package ID
  - Package weight
  - Container/space location ID
- Derived by our model
  - Averages
  - Standard deviation

Problem Model

2. Representation of solution – we have some options

- Assigning packages to locations
  - The location remain static but the assigned packages change
  - In this model, the chromosome consists of Package IDs
- Assigning locations to packages
  - The packages remain static but the assigned locations change
  - In this model, the chromosome consists of location IDs

Problem Model

3. Fitness function

- Two different averages (of weights)
  - Overall across all packages (static)
  - For each stack (changes with assigned packages)
- Minimise the spread between the average stack weights and the overall average weight

Minimise $\sum_i (x_i - AV)^2$

where x_i is the average package weight in each stack and AV is the average of all the packages.

Problem Model

4. Constraints

- Exactly sixteen stacks, each with four levels
- Lighter packages on top of heavier ones – we don’t want to have crushed packages upon arrival
- Explain how each constraint can be addressed based on the solution representation

Handling Constraints

- We are asked to propose only those solutions with lighter packages above heavier ones
  BUT
  we don't need to use hard constraints
- Hard constraints have an adverse impact on performance due to the generate-and-test strategy GA uses

Handling Constraints

- **Options for handling weight-ordering constraint**
  - Using soft constraints
    - Factor in the degree of the violation
    - Include this as penalty to fitness function,
      BUT may need to scale penalty appropriately
  - Not using GA constraints at all!
    - Don’t feel trapped into using GA constraints for everything
    - Simply sort the packages in each stack after finding a good solution – fitness is not affected!

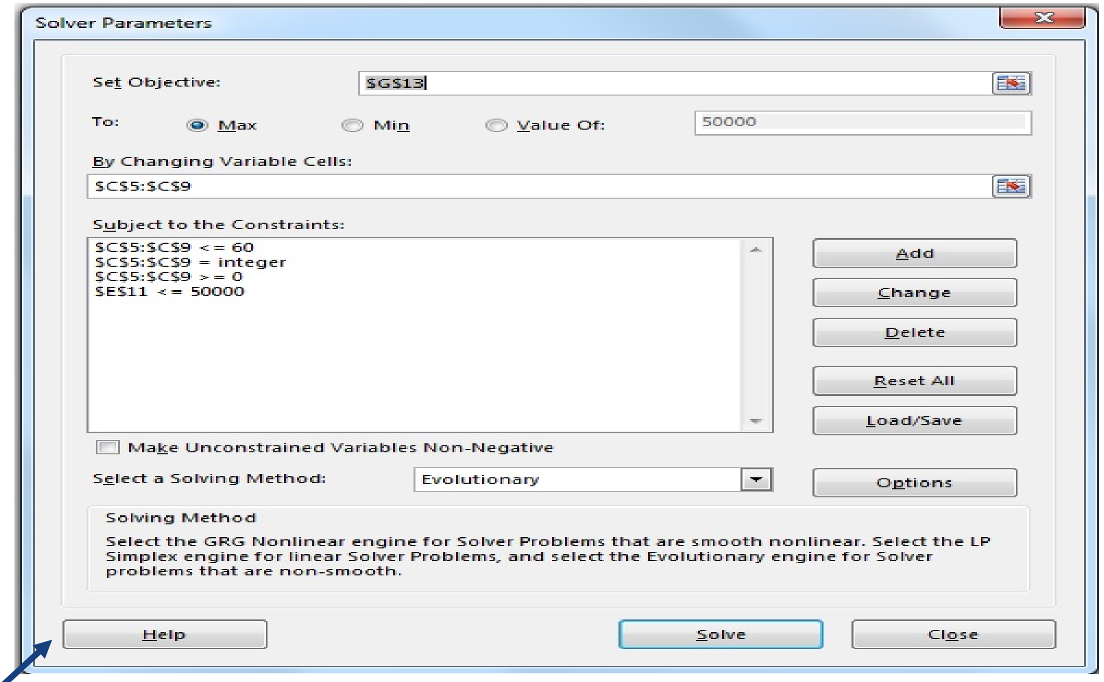
Appendix: Excel Solver

How to Load the Excel Solver Add-in

- **Open Excel**
- Click the **File** tab, and then click **Options**.
- Click **Add-Ins**, and then in the **Manage** box, select **Excel Add-ins**.
- Click **Go**.
- In the **Add-Ins available** box, select the **Solver Add-in** check box, and then click **OK**.
- If **Solver Add-in** is not listed in the **Add-Ins available** box, click **Browse** to locate the add-in.
- If you get prompted that the Solver add-in is not currently installed on your computer, click **Yes** to install it.
- After you load the Solver add-in, the **Solver** command is available in the **Analysis** group on the **Data** tab.

Excel Solver

- Excel Solver Help
  - http://www.solver.com/excel-solver-help



#Click HELP to get the help file.

GA Modelling – Excel Solver

Step 1:

- Decide what you are optimising for (e.g. maximise profit). This 'goodness measure' or objective will eventually translate into a fitness function

Step 2:

- Represent each solution chromosome as an array of real numbers or integers
- E.g. to find the best split of funds that will allow a company to maximise profits

advertising	marketing	production	salary		profit
8%	12%	50%	30%		$2780

One solution

Fitness function

GA Modelling – Excel Solver

Step 3:

- Write a **fitness function** that evaluates the goodness of a solution chromosome.
- This function takes as input a candidate solution and returns a number that indicates how good the solution is (e.g. the amount of expected profit)

Step 4:

- Define any constraints on the values of your solution chromosome.

Exercise 1

The following jobs can be processed on any of the 5 machines. How can these jobs be assigned to the machine so that the total processing time for the jobs is minimum. The time taken to process each job on each machine is known.

Set the Excel spreadsheet as shown.

	A	B	C	D	E	F
1		Machine ID	Process Time			
2	Job1	1	=INDEX($B$11:$F$15,1, B2)			
3	Job2	2	=INDEX($B$11:$F$15, 2, B3)			
4	Job3	3	=INDEX($B$11:$F$15, 3, B4)			
5	Job4	4	=INDEX($B$11:$F$15, 4, B5)			
6	Job5	5	=INDEX($B$11:$F$15, 5, B6)			
7		Total	=SUM(C2:C6)			
8						
9						
10	Machine ID	1	2	3	4	5
11	Job1	12	45	23	33	12
12	Job2	34	13	8	14	25
13	Job3	22	13	33	15	24
14	Job4	14	56	23	12	26
15	Job5	4	13	23	34	27

Exercise 2

You have a group of students whose average overall performance are known. You are required to divide them into 3 groups so that the members of each group can interact well with each other. To ensure that they can interact well with each other the deviations of their performance should be minimum.

Set the Excel spreadsheet as shown

	A	B	C	D	E	F
1	Student	Performance	Group	Group1	Group2	Group3
2	Tom	50	1	=IF(C2=1, B2,”0”)	=IF(C2=2, B2, ”0”)	=IF(C2=3, B2, ”0”)
3	Jerry	67	2	=IF(C3=1, B3, ”0”)	=IF(C3=2, B3, ”0”)	=IF(C3=3, B3, ”0”)
4	Ann	34	3	=IF(C4=1, B4, ”0”)	=IF(C4=2, B4, ”0”)	=IF(C4=3, B4, ”0”)
5	Bob	55	1	=IF(C5=1, B5, ”0”)	=IF(C5=2, B5, ”0”)	=IF(C5=3, B5, ”0”)
6	June	80	2	=IF(C6=1, B6, ”0”)	=IF(C6=2, B6, ”0”)	=IF(C6=3, B6, ”0”)
7	Nancy	90	3	=IF(C7=1, B7, ”0”)	=IF(C7=2, B7, ”0”)	=IF(C7=3, B7, ”0”)
8	Mike	77	1	=IF(C8=1, B8, ”0”)	=IF(C8=2, B8, ”0”)	=IF(C8=3, B8, ”0”)
9	Dolly	55	2	=IF(C9=1, B9, ”0”)	=IF(C9=2, B9, ”0”)	=IF(C9=3, B9, ”0”)
10	Paul	66	3	=IF(C10=1,B10, ”0”)	=IF(C10=2, B10, ”0”)	=IF(C10=3, B10, ”0”)
11	Lucy	83	1	=IF(C11=1,B11, ”0”)	=IF(C11=2, B11, ”0”)	=IF(C11=3, B11, ”0”)
12	John	44	2	=IF(C12=1,B12, ”0”)	=IF(C12=2, B12, ”0”)	=IF(C12=3, B12, ”0”)
13	Mary	73	3	=IF(C13=1,B13, ”0”)	=IF(C13=2, B13, ”0”)	=IF(C13=3, B13, ”0”)
14	Number			=COUNTIF(D2:D13,”>0”)	=COUNTIF(E2:E13,”>0”)	=COUNTIF(F2:F13,”>0”)
15	Average			=AVERAGE(D2:D13)	=AVERAGE(E2:E13)	=AVERAGE(F2:F13)
16	Std Dev			=STDEV(D2:D13)	=STDEV(E2:E13)	=STDEV(F2:F13)
17						
18				=SUM(D16:F16)		

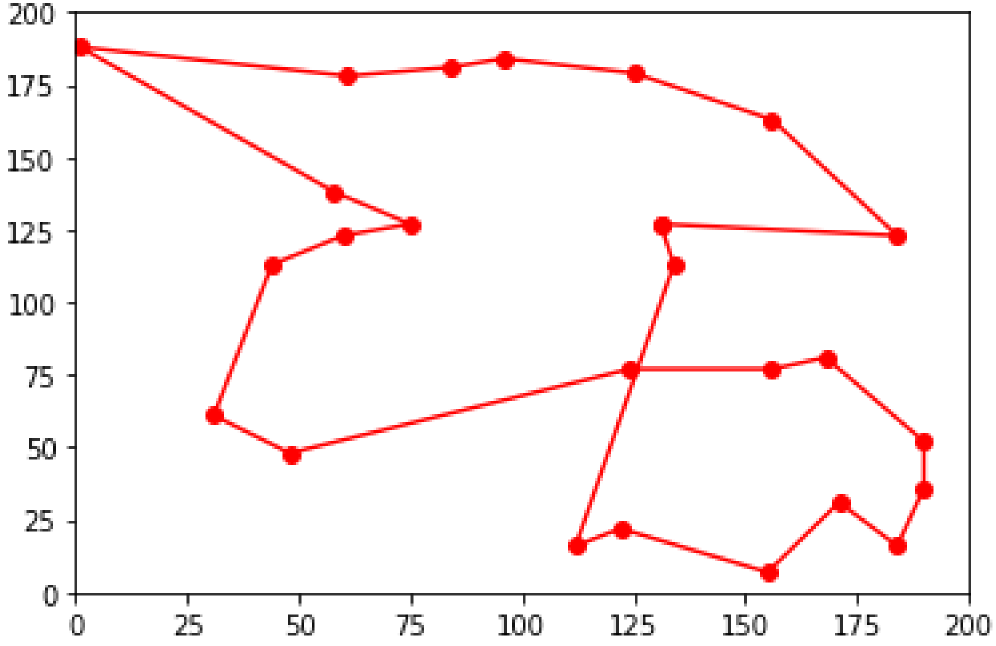
GA Workshop: Solving Travelling Salesman Problem

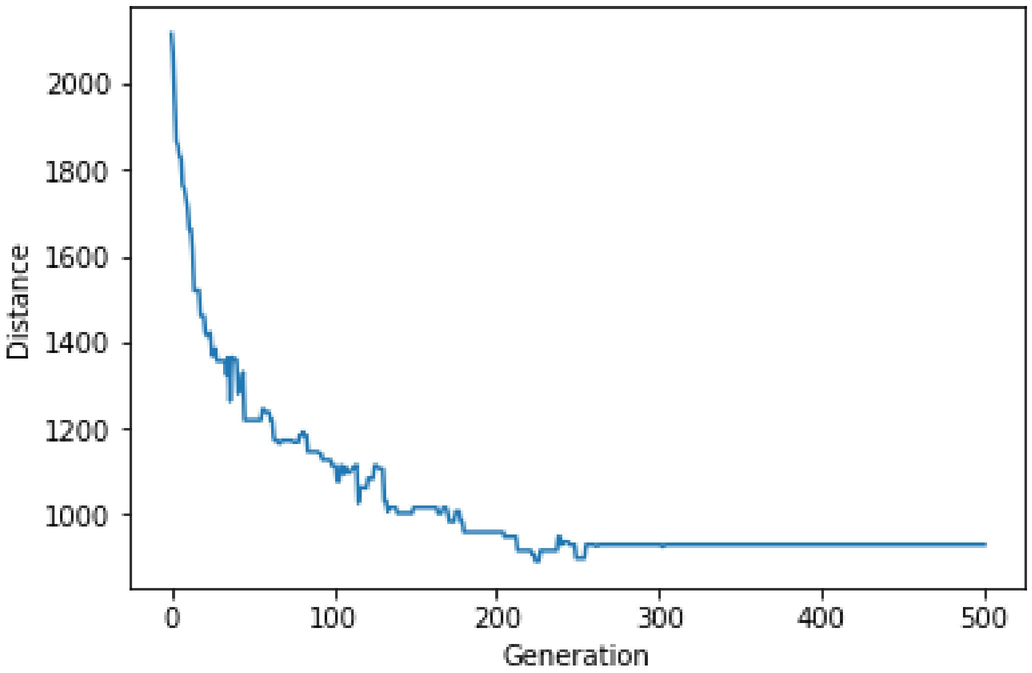
Using Python

Problem Statement: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?"

TSP - Python

- Run the Python code provided
- Understand the GA concepts through the code
- Experiment with different parameter settings on popSize, mutationRate, etc.





References

1) Forgel, L.J., Owens, A.F., & Walsh, M.J., “Artificial Intelligence through Simulated Evolution” New York: John Wiley (1966).
2) Holland, J.H., “Adaptation in Natural and Artificial Systems” , Ann Arbor, MI: The University of Michigan Press (1975).
3) Goldberg, D.E., “Genetic Algorithms in Search, Optimization & Machine Learning”, Addison-Wesley (1989).
4) Davis, L., “Handbook of Genetic Algorithms”, Van Nostrand Reinhold, New York (1991).
5) Michalewicz, Z., “Genetic Algorithms + Data Structures = Evolution Programs”, Springer-Varlag, New York (1992).
6) Koza, J.R., “Genetic Programming: on the programming of computers by means of natural selection”, Cambridge, MA:MIT press (1992).

References

7) Genetic Algorithms & Grouping Problems, Emanuel Falkenauer, Wiley, 1998.

8) Mitchell, M. “An Introduction to Genetic Algorithms”, MIT Press, Cambridge, MA, 1996.

9) Koza, J. “Genetic Programming II”, MIT Press, Cambridge, MA, 1994.

10) Baeck, Th. “Evolutionary Algorithms in Theory and Practice”, Oxford University Press, New York, 1996.

11) Eiben, A.E., Smith, J.E., Introduction to Evolutionary Computing, Springer, 2010.

12) De Jong, K. A., Evolutionary computation: a unified approach. MIT Press, Cambridge MA, 2006.

13) Frances Buontempo, Genetic Algorithms and Machine Learning for Programmers: Create AI Models and Evolve Solutions, Pragmatic Bookshelf, 2019.

14) Dan Simon, Evolutionary Optimization Algorithms, Wiley; 1 edition, 2013.

15) Clinton Sheppard, Genetic Algorithms with Python, CreateSpace Independent Publishing Platform, 2016.