Optimization Methods-2

Lecture 2

III. What is a Metaheuristic Method?

- Meta : in an upper level
- Heuristic : to find

A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions. [Osman and Laporte 1996].

Fundamental Properties of Metaheuristics [Blum and Roli 2003]

- Metaheuristics are strategies that “guide” the search process.
- The goal is to efficiently explore the search space in order to find (near-)optimal solutions.
- Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes.
- Metaheuristic algorithms are approximate and usually non-deterministic.

Fundamental Properties of Metaheuristics (cont.)

- They may incorporate mechanisms to avoid getting trapped in confined areas of the search space.
- The basic concepts of metaheuristics permit an abstract level description.
- Metaheuristics are not problem-specific.
- Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.
- Today's more advanced metaheuristics use search experience (embodied in some form of memory) to guide the search.

Genetic Algorithms (GA) Quick Overview

- Developed: USA in the 1970's
- Early names: J. Holland, K. DeJong, D. Goldberg
- Typically applied to:
  - discrete optimization
- Attributed features:
  - not too fast
  - good heuristic for combinatorial problems
- Special Features:
  - Traditionally emphasizes combining information from good parents (crossover)
  - many variants, e.g., reproduction models, operators

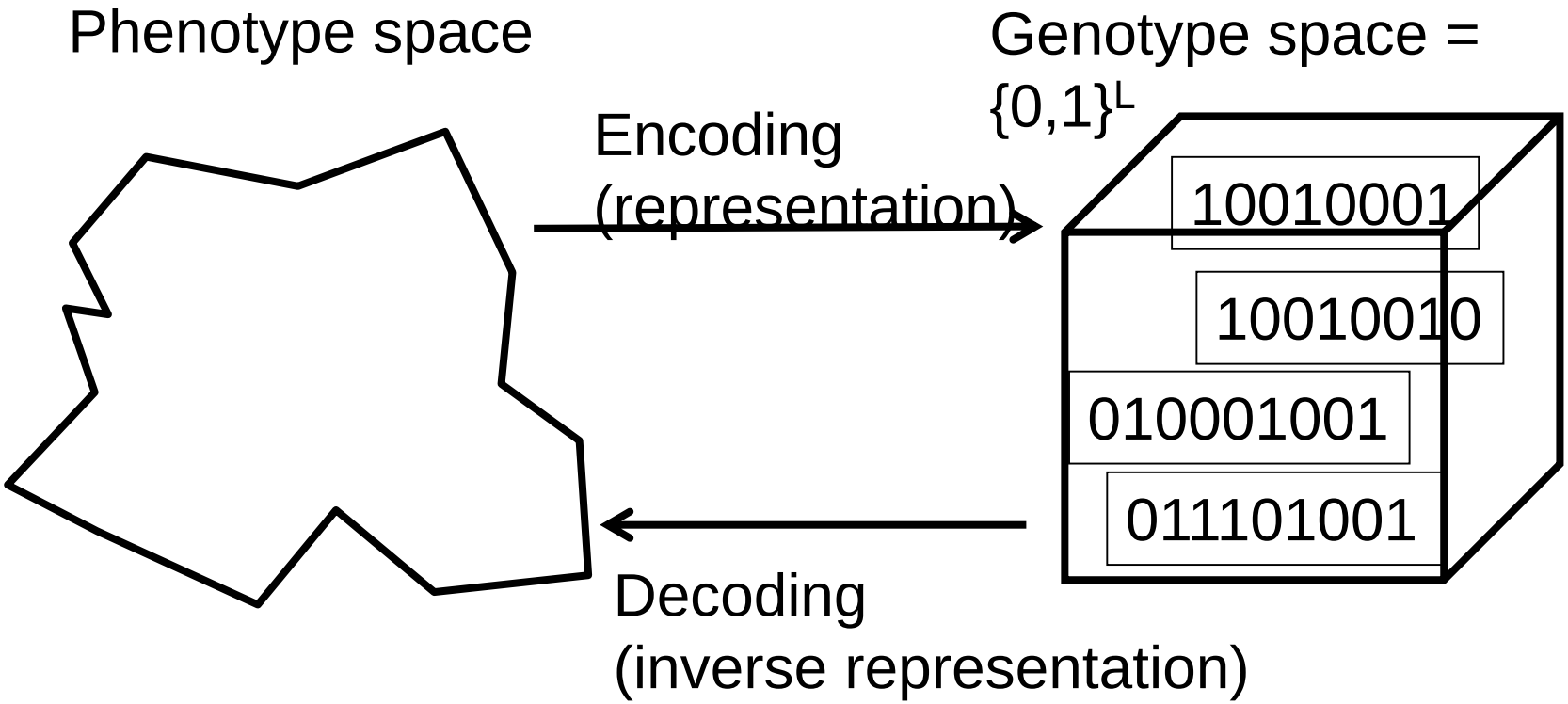
Genetic algorithms

- Holland's original GA is now known as the simple genetic algorithm (SGA)
- Other GAs use different:
  - Representations
  - Mutations
  - Crossovers
  - Selection mechanisms

SGA technical summary tableau

Representation	Binary strings
Recombination	N-point or uniform
Mutation	Bitwise bit-flipping with fixed probability
Parent selection	Fitness-Proportionate
Survivor selection	All children replace parents
Speciality	Emphasis on crossover

Representation



The Metaphor

Genetic Algorithm	Nature
Optimization problem	Environment
Feasible solutions	Individuals living in that environment
Solutions quality (fitness function)	Individual's degree of adaptation to its surrounding environment

The Metaphor (cont)

Genetic Algorithm	Nature
A set of feasible solutions	A population of organisms (species)
Stochastic operators	Selection, recombination and mutation in nature's evolutionary process
Iteratively applying a set of stochastic operators on a set of feasible solutions	Evolution of populations to suit their environment

The Metaphor (cont)

The computer model introduces simplifications (relative to the real biological mechanisms),

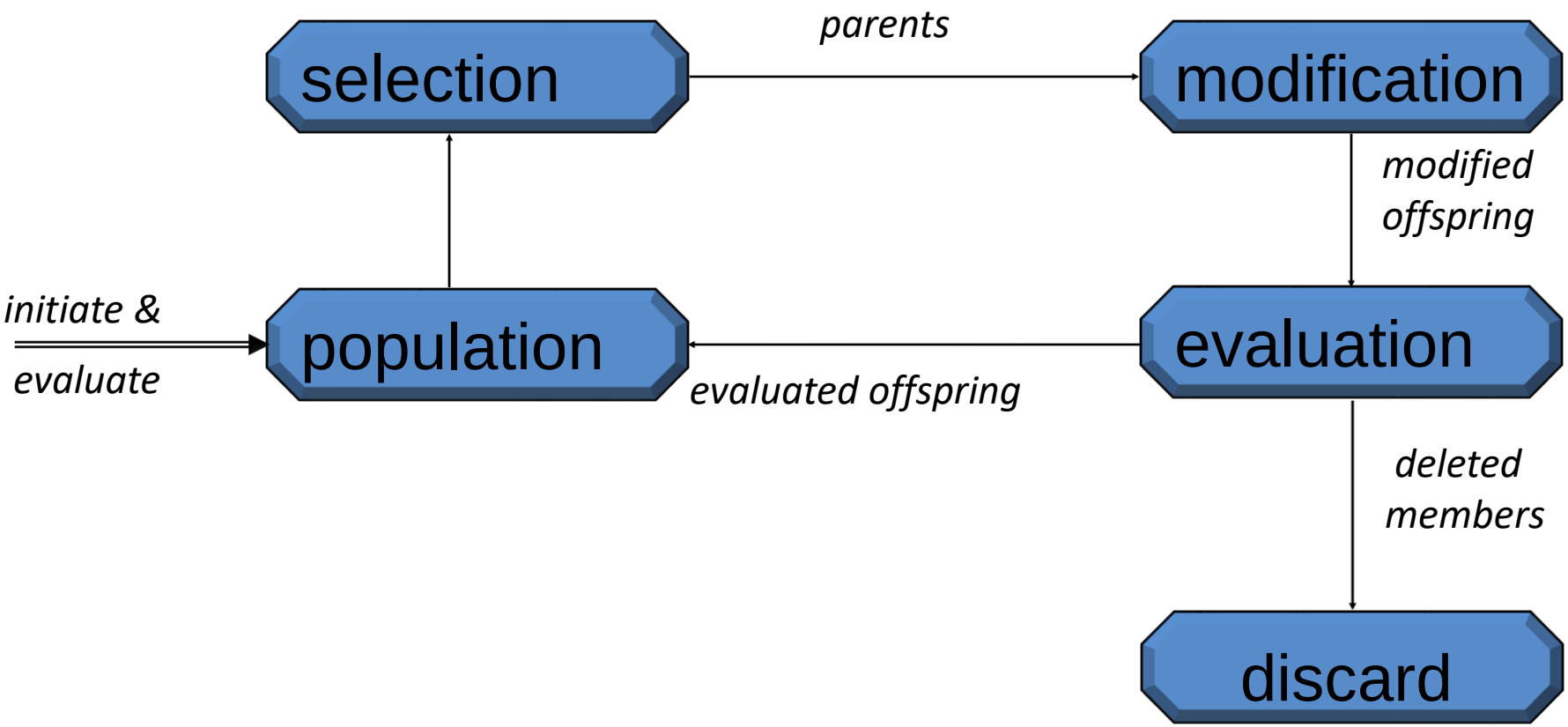
BUT

surprisingly complex and interesting structures have emerged out of evolutionary algorithms

SGA reproduction cycle

1. Select parents for the mating pool
   (size of mating pool = population size)
2. Shuffle the mating pool
3. For each consecutive pair apply crossover with probability p_c , otherwise copy parents
4. For each offspring apply mutation (bit-flip with probability p_m independently for each bit)
5. Replace the whole population with the resulting offspring

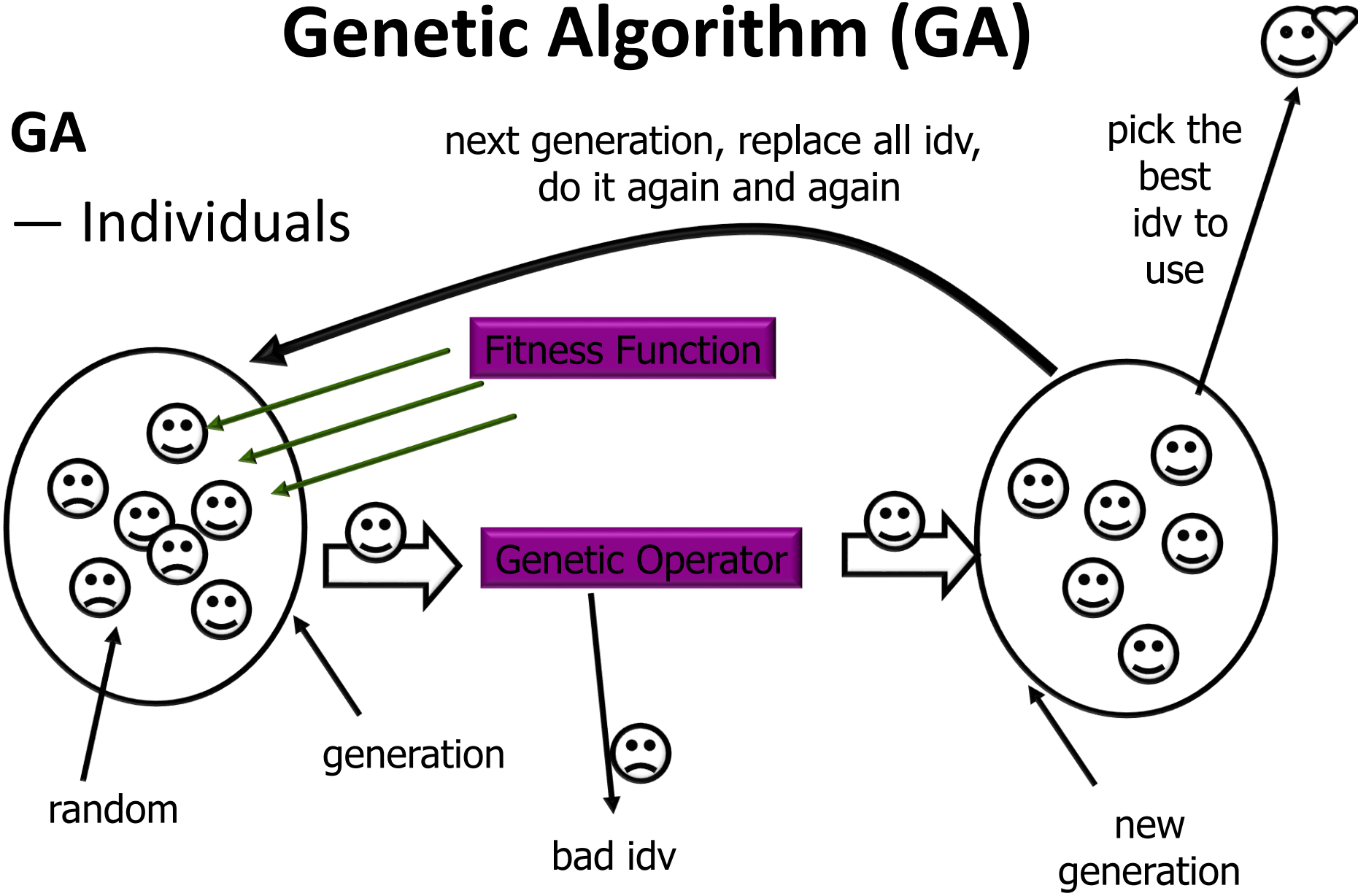
The Evolutionary Cycle

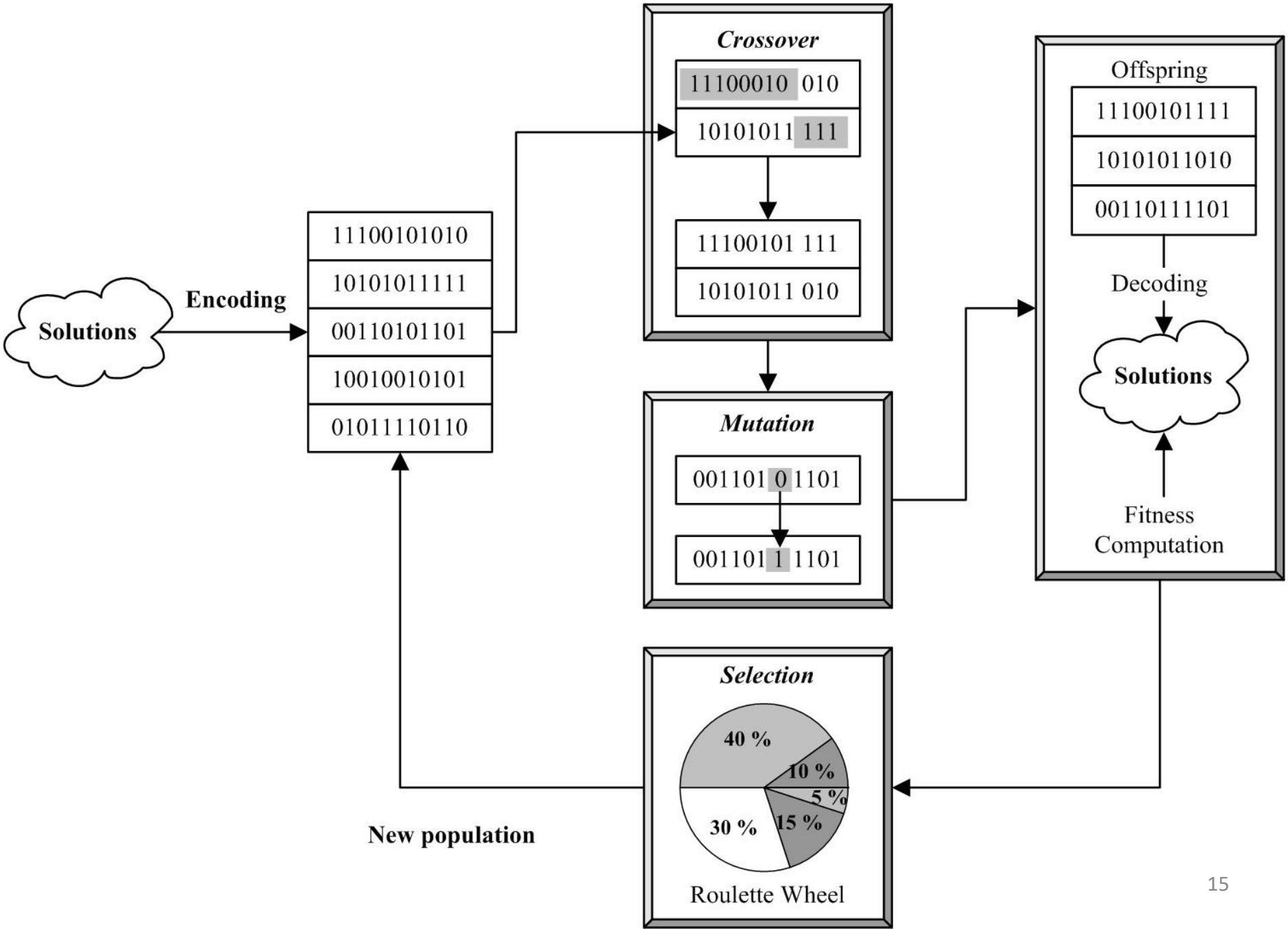


Genetic Algorithm (GA)

GA

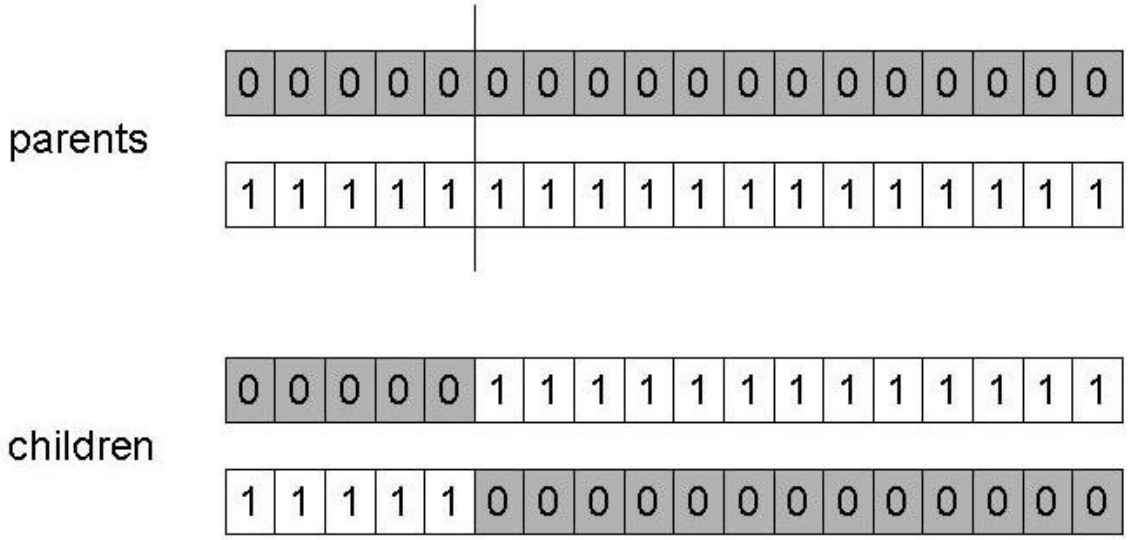
— Individuals




Solutions
Encoding
11100101010
10101011111
00110101101
10010010101
01011110110
Crossover
11100010 010
10101011 111
11100101 111
10101011 010
Mutation
001101 0 1101
001101 1 1101
Offspring
11100101111
10101011010
00110111101
Decoding
Solutions
Fitness Computation
Selection
40 %
10 %
5 %
15 %
30 %
Roulette Wheel
New population

SGA operators: 1-point crossover

- Choose a random point on the two parents
- Split parents at this crossover point
- Create children by exchanging tails
- P_c typically in range (0.6, 0.9)



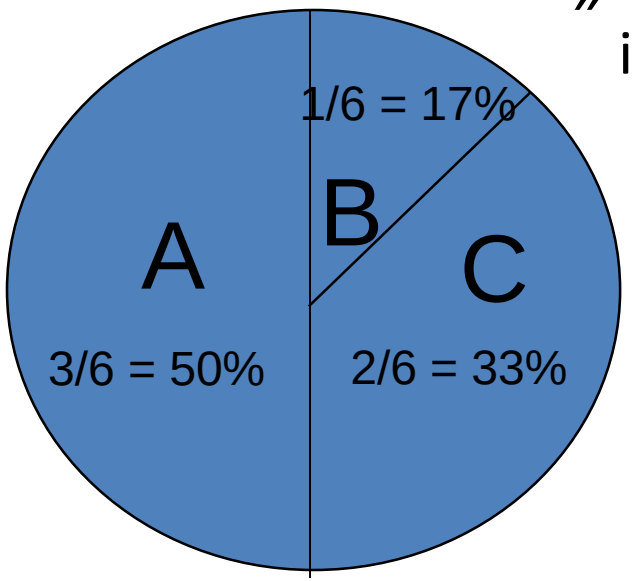
SGA operators: mutation

- Alter each gene independently with a probability p_m
- p_m is called the mutation rate
  - Typically between 1/pop_size and 1/chromosome_length

parent	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
child	0	1	0	0	1	0	1	1	0	0	0	1	0	1	1	0	0	1

SGA operators: Selection

- Main idea: better individuals get higher chance
  - Chances proportional to fitness
  - Implementation: roulette wheel technique
    - » Assign to each individual a part of the roulette wheel
    - » Spin the wheel n times to select n individuals



fitness(A) = 3

fitness(B) = 1

fitness(C) = 2

An example after Goldberg '89 (1)

- Simple problem: max x^2 over {0,1,…,31}
- GA approach:
  - Representation: binary code, e.g. 01101 $\leftrightarrow$ 13
  - Population size: 4
  - 1-point xover, bitwise mutation
  - Roulette wheel selection
  - Random initialisation
- We show one generational cycle done by hand

x2 example: selection

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

X2 example: crossover

String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 | 1	4	0 1 1 0 0	12	144
2	1 1 0 0 | 0	4	1 1 0 0 1	25	625
2	1 1 | 0 0 0	2	1 1 0 1 1	27	729
4	1 0 | 0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

X^2 example: mutation

String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	1 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	18	324
Sum				2354
Average				588.5
Max				729

The simple GA

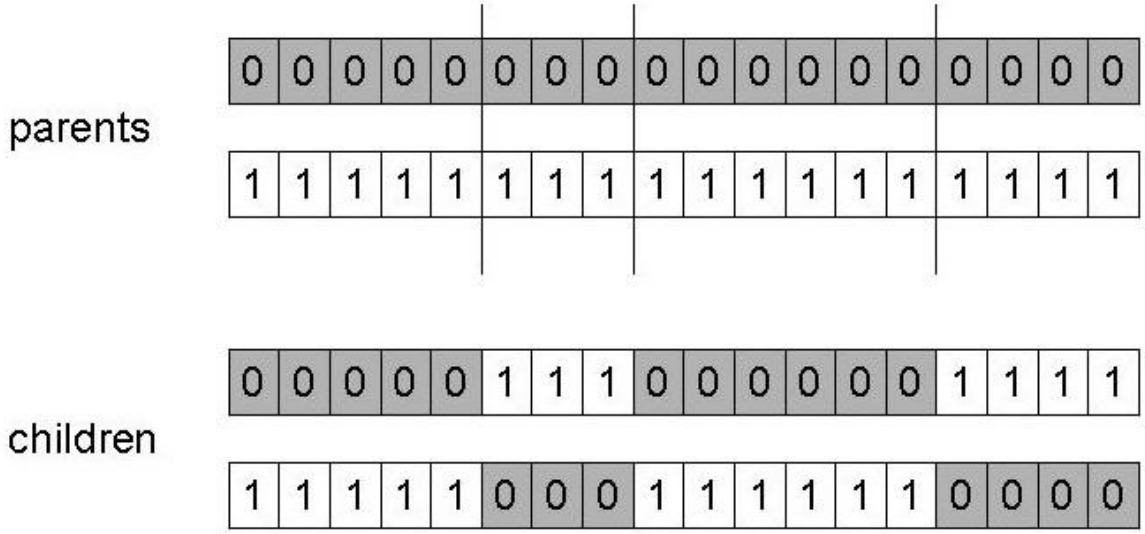
- Has been subject of many (early) studies
  - still often used as benchmark for novel GAs
- Shows many shortcomings, e.g.
  - Representation is too restrictive
  - Mutation & crossovers only applicable for bit-string & integer representations
  - Selection mechanism sensitive for converging populations with close fitness values
  - Generational population model (step 5 in SGA repr. cycle) can be improved with explicit survivor selection

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
  - Can never keep together genes from opposite ends of string
  - This is known as *Positional Bias*
  - Can be exploited if we know about the structure of our problem, but this is not usually the case

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)



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

parents

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

children

0	1	0	0	1	0	1	1	0	0	0	1	0	1	1	0	0	1
1	0	1	1	0	0	0	0	1	1	1	0	1	0	0	1	1	0

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, xover-only-EA would not work

Crossover OR mutation? (cont'd)

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

Crossover OR mutation? (cont'd)

- Only crossover can combine information from two parents
- Only mutation can introduce new information (alleles)
- Crossover does not change the allele frequencies of the population (thought experiment: 50% 0's on first bit in the population, ?% after performing *n* crossovers)
- To hit the optimum you often need a 'lucky' mutation

Other representations

- Gray coding of integers (still binary chromosomes)
  - Gray coding is a mapping that means that small changes in the genotype cause small changes in the phenotype (unlike binary coding). “Smoother” genotype-phenotype mapping makes life easier for the GA

Nowadays it is generally accepted that it is better to encode numerical variables directly as

- Integers
- Floating point variables

Integer representations

- Some problems naturally have integer variables, e.g. image processing parameters
- Others take *categorical* values from a fixed set e.g. {blue, green, yellow, pink}
- N-point / uniform crossover operators work
- Extend bit-flipping mutation to make
  - “creep” i.e. more likely to move to similar value
  - Random choice (esp. categorical variables)
  - For ordinal problems, it is hard to know correct range for creep, so often use two mutation operators in tandem

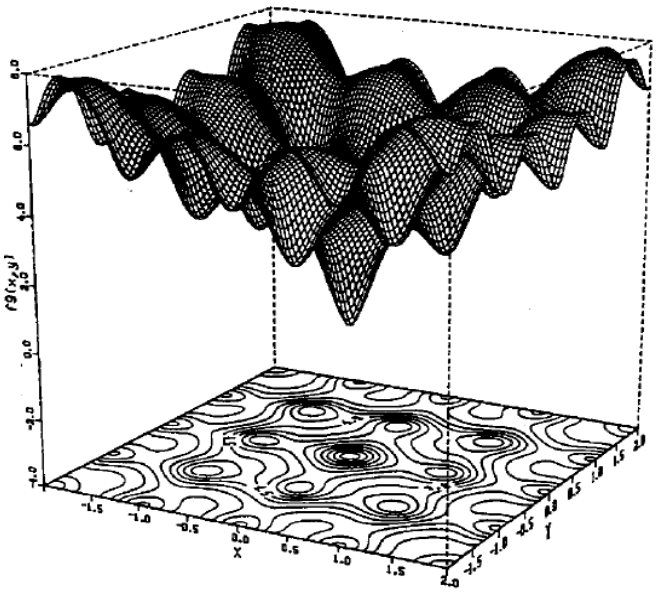
Real valued problems

- Many problems occur as real valued problems, e.g. continuous parameter optimisation $f : \Re^n \rightarrow \Re$
- Illustration: Ackley's function (often used in EC)

$$f(\overline{x}) = -c_1 \cdot exp\left(-c_2 \cdot \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right)$$

$$-exp\left(\frac{1}{n} \cdot \sum_{i=1}^{n} cos(c_3 \cdot x_i)\right) + c_1 + 1$$

$$c_1 = 20,\ c_2 = 0.2,\ c_3 = 2\pi$$



Mapping real values on bit strings

$z \in [x,y] \subseteq \mathcal{R}$ represented by $\{a_1,\ldots,a_L\} \in \{0,1\}^L$

- $[x,y] \rightarrow \{0,1\}^L$ must be invertible (one phenotype per genotype)
- $\Gamma: \{0,1\}^L \rightarrow [x,y]$ defines the representation

$$\Gamma(a_1,\ldots,a_L) = x + \frac{y-x}{2^L - 1} \cdot \left(\sum_{j=0}^{L-1} a_{L-j} \cdot 2^j\right) \in [x,y]$$

- Only 2^L values out of infinite are represented
- L determines possible maximum precision of solution
- High precision → long chromosomes (slow evolution)

Floating point mutations 1

General scheme of floating point mutations

$$\bar{x} = \langle x_1, \ \ldots, x_l \rangle \rightarrow \bar{x}' = \langle x_1', \ldots, x_l' \rangle$$
$$x_i, x_i' \in [LB_i, UB_i]$$

- Uniform mutation:

$$x_i' \text{ drawn randomly (uniform) from } [LB_i, UB_i]$$

- Analogous to bit-flipping (binary) or random resetting (integers)

Floating point mutations 2

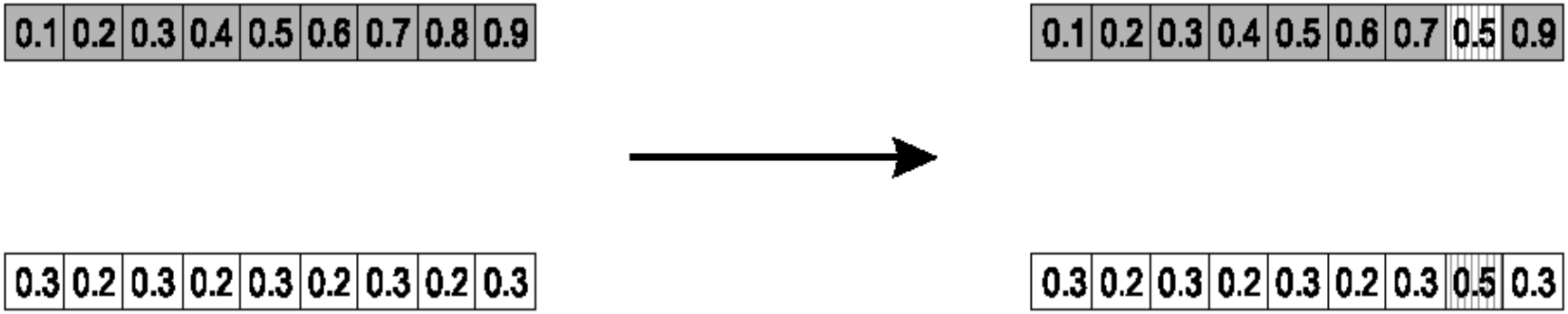
- Non-uniform mutations:
  - Many methods proposed,such as time-varying range of change etc.
  - Most schemes are probabilistic but usually only make a small change to value
  - Most common method is to add random deviate to each variable separately, taken from N(0, σ) Gaussian distribution and then curtail to range
  - Standard deviation σ controls amount of change (2/3 of deviations will lie in range (- σ to + σ)

Crossover operators for real valued GAs

- Discrete:
  - each allele value in offspring *z* comes from one of its parents *(x,y)* with equal probability: $z_i = x_i$ or y_i
  - Could use n-point or uniform
- Intermediate
  - exploits idea of creating children "between" parents (hence a.k.a. *arithmetic* recombination)
  - $z_i = \alpha x_i + (1 - \alpha) y_i$ where α : $0 \leq \alpha \leq 1$.
  - The parameter α can be:
    - constant: uniform arithmetical crossover
    - variable (e.g. depend on the age of the population)
    - picked at random every time

Single arithmetic crossover

- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- Pick a single gene (*k*) at random,
- $child_1$ is: $\left\langle x_1, \ldots, x_k, \alpha \cdot y_k + (1-\alpha) \cdot x_k, \ldots, x_n \right\rangle$
- reverse for other child. e.g. with $\alpha = 0.5$

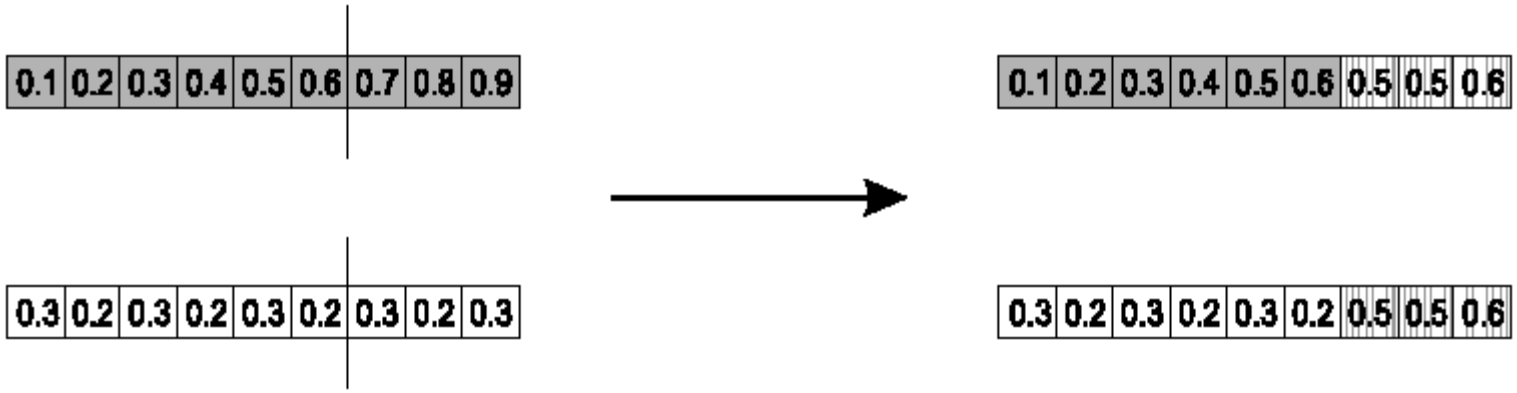


Simple arithmetic crossover

- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- Pick random gene *(k)* after this point mix values
- $child_1$ is:

$$\left\langle x_1,\ldots,x_k,\alpha\cdot y_{k+1}+(1-\alpha)\cdot x_{k+1},\ldots,\alpha\cdot \mathrm{y}_{\mathrm{n}}+(1-\alpha)\cdot x_n \right\rangle$$

- reverse for other child. e.g. with α = 0.5

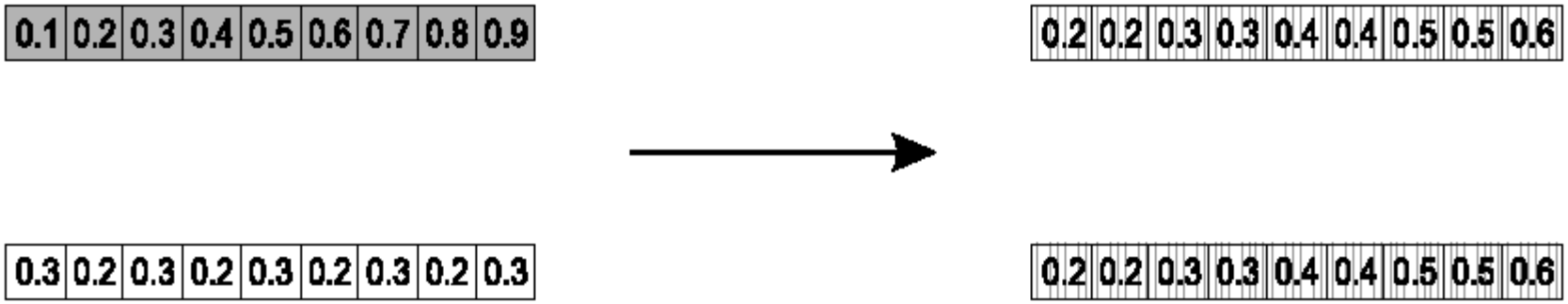


Whole arithmetic crossover

- Most commonly used
- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- $child_1$ is:

$$a \cdot \bar{x} + (1-a) \cdot \bar{y}$$

- reverse for other child. e.g. with $\alpha = 0.5$



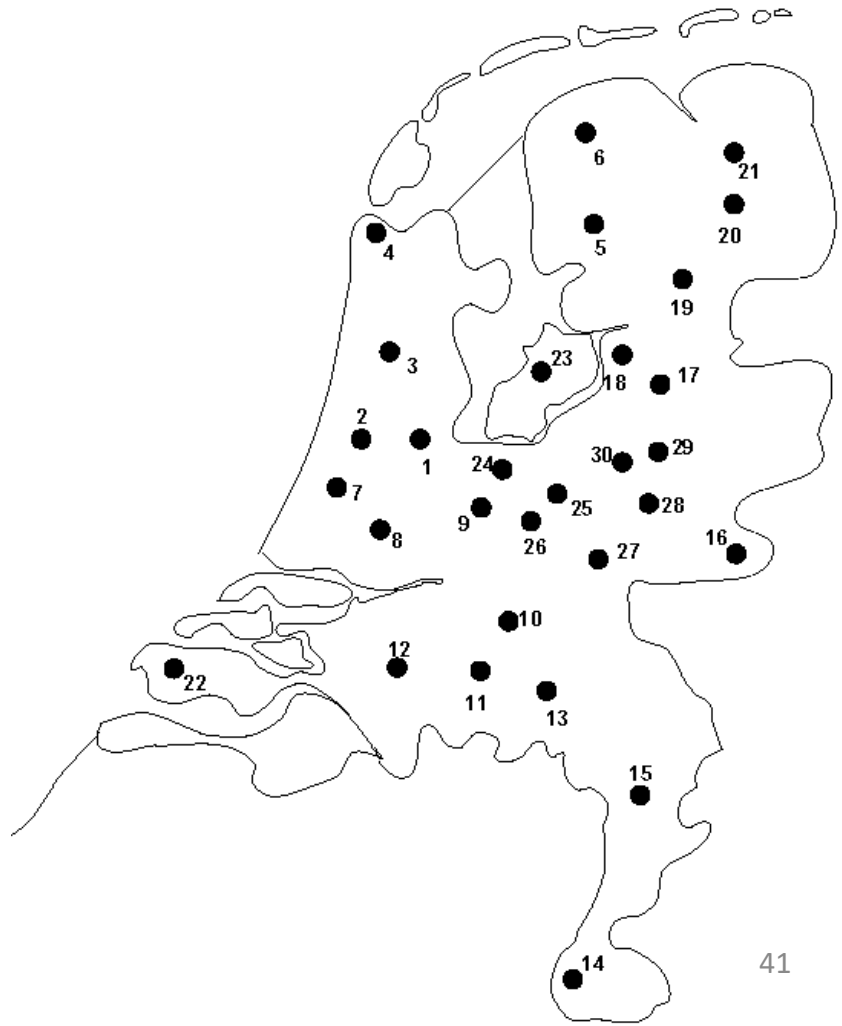
Permutation Representations

- Ordering/sequencing problems form a special type
- Task is (or can be solved by) arranging some objects in a certain order
  - Example: sort algorithm: important thing is which elements occur before others (order)
  - Example: Travelling Salesman Problem (TSP) : important thing is which elements occur next to each other (adjacency)
- These problems are generally expressed as a permutation:
  - if there are *n* variables then the representation is as a list of *n* integers, each of which occurs exactly once

Permutation representation: TSP example

- Problem:
  - Given n cities
  - Find a complete tour with minimal length
- Encoding:
  - Label the cities 1, 2, … , *n*
  - One complete tour is one permutation (e.g. for n =4 [1,2,3,4], [3,4,2,1] are OK)
- Search space is BIG:

  for 30 cities there are 30! ≈ 10^{32} possible tours



Mutation operators for permutations

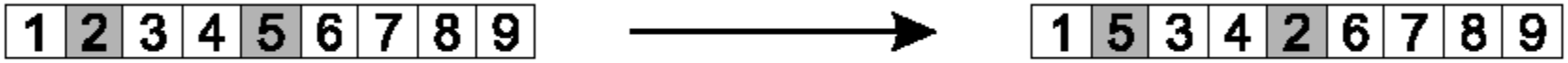
- Normal mutation operators lead to inadmissible solutions
  - e.g. bit-wise mutation : let gene *i* have value *j*
  - changing to some other value *k* would mean that *k* occurred twice and *j* no longer occurred
- Therefore must change at least two values
- Mutation parameter now reflects the probability that some operator is applied once to the whole string, rather than individually in each position

Insert Mutation for permutations

- Pick two allele values at random
- Move the second to follow the first, shifting the rest along to accommodate
- Note that this preserves most of the order and the adjacency information

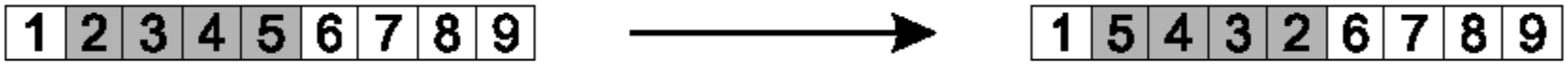
Swap mutation for permutations

- Pick two alleles at random and swap their positions
- Preserves most of adjacency information (4 links broken), disrupts order more



Inversion mutation for permutations

- Pick two alleles at random and then invert the substring between them.
- Preserves most adjacency information (only breaks two links) but disruptive of order information



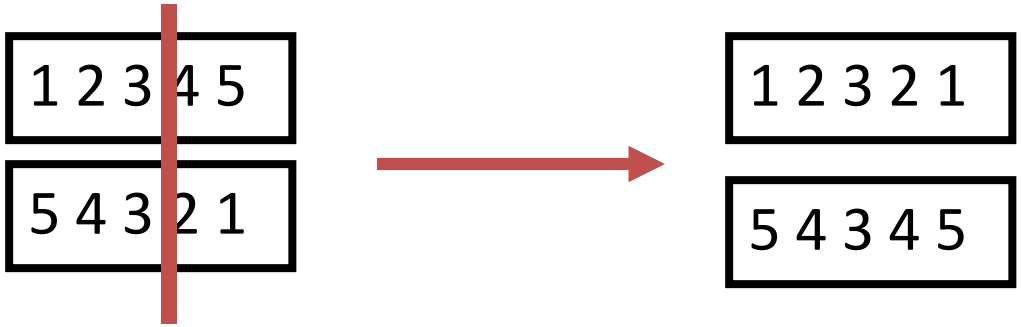
Scramble mutation for permutations

- Pick a subset of genes at random
- Randomly rearrange the alleles in those positions

(note subset does not have to be contiguous)

Crossover operators for permutations

- "Normal" crossover operators will often lead to inadmissible solutions



- Many specialised operators have been devised which focus on combining order or adjacency information from the two parents

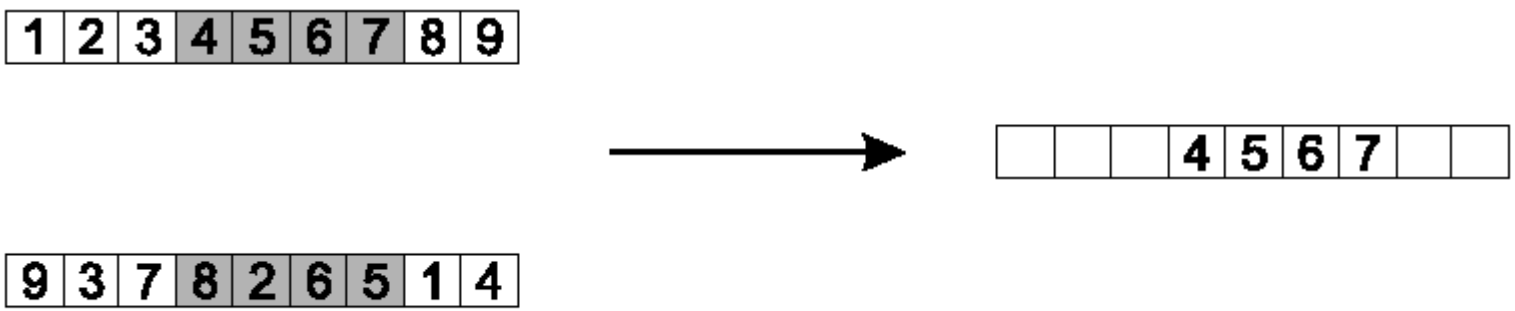
Order 1 crossover

- Idea is to preserve relative order that elements occur
- Informal procedure:

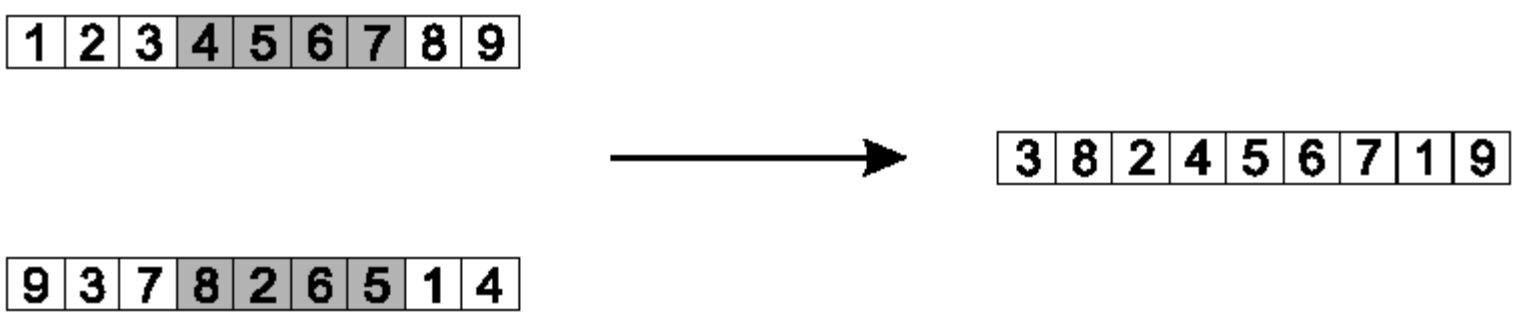
1. Choose an arbitrary part from the first parent
2. Copy this part to the first child
3. Copy the numbers that are not in the first part, to the first child:
   - starting right from cut point of the copied part,
   - using the **order** of the second parent
   - and wrapping around at the end
4. Analogous for the second child, with parent roles reversed

Order 1 crossover example

- Copy randomly selected set from first parent



- Copy rest from second parent in order 1,9,3,8,2



Partially Mapped Crossover (PMX)

Informal procedure for parents P1 and P2:

1. Choose random segment and copy it from P1
2. Starting from the first crossover point look for elements in that segment of P2 that have not been copied
3. For each of these *i* look in the offspring to see what element *j* has been copied in its place from P1
4. Place *i* into the position occupied *j* in P2, since we know that we will not be putting *j* there (as is already in offspring)
5. If the place occupied by *j* in P2 has already been filled in the offspring *k*, put *i* in the position occupied by *k* in P2
6. Having dealt with the elements from the crossover segment, the rest of the offspring can be filled from P2.

Second child is created analogously

PMX example

- Step 1

1	2	3	4	5	6	7	8	9

			4	5	6	7		

9	3	7	8	2	6	5	1	4

- Step 2

1	2	3	4	5	6	7	8	9

		2	4	5	6	7		8

9	3	7	8	2	6	5	1	4

- Step 3

1	2	3	4	5	6	7	8	9

9	3	2	4	5	6	7	1	8

9	3	7	8	2	6	5	1	4

Cycle crossover

Basic idea:

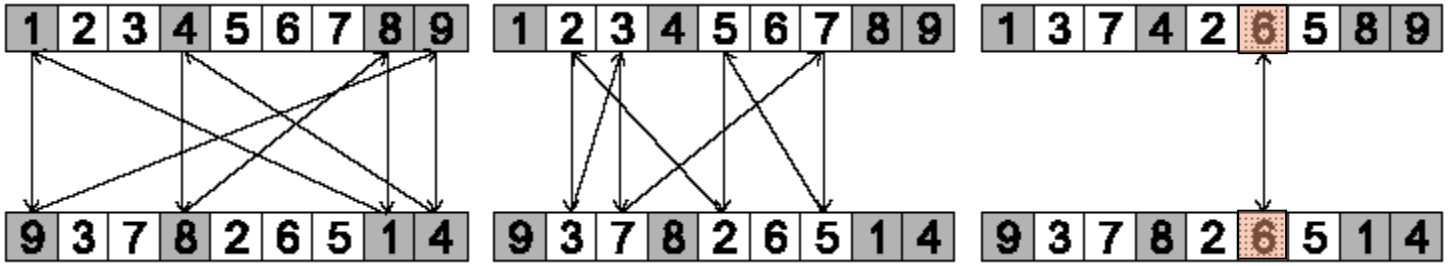
Each allele comes from one parent *together with its position*.

Informal procedure:

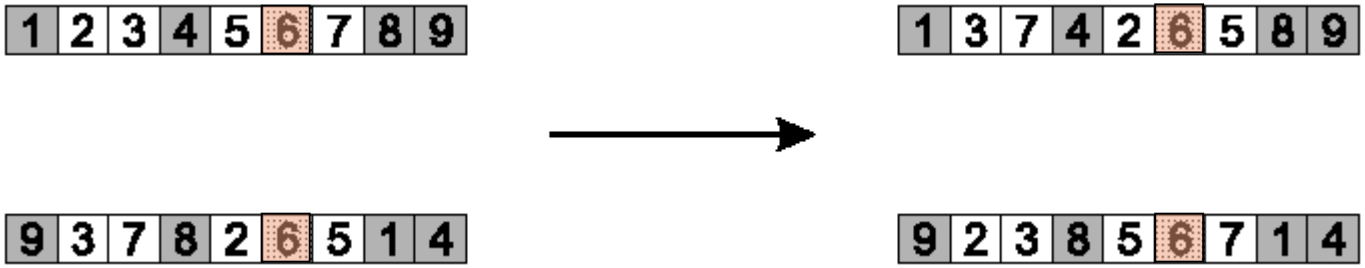
1. Make a cycle of alleles from P1 in the following way.
   (a) Start with the first allele of P1.
   (b) Look at the allele at the *same position* in P2.
   (c) Go to the position with the *same allele* in P1.
   (d) Add this allele to the cycle.
   (e) Repeat step b through d until you arrive at the first allele of P1.
2. Put the alleles of the cycle in the first child on the positions they have in the first parent.
3. Take next cycle from second parent

Cycle crossover example

- Step 1: identify cycles



- Step 2: copy alternate cycles into offspring



Multiparent recombination

- Recall that we are not constricted by the practicalities of nature
- Noting that mutation uses 1 parent, and “traditional” crossover 2, the extension to a>2 is natural to examine
- Been around since 1960s, still rare but studies indicate useful
- Three main types:
  - Based on allele frequencies, e.g., p-sexual voting generalising uniform crossover
  - Based on segmentation and recombination of the parents, e.g., diagonal crossover generalising n-point crossover
  - Based on numerical operations on real-valued alleles, e.g., center of mass crossover, generalising arithmetic recombination operators

Population Models

- SGA uses a Generational model:
  - each individual survives for exactly one generation
  - the entire set of parents is replaced by the offspring
- At the other end of the scale are Steady-State models:
  - one offspring is generated per generation,
  - one member of population replaced,
- Generation Gap
  - the proportion of the population replaced
  - 1.0 for GGA, 1/pop_size for SSGA

Fitness Based Competition

- Selection can occur in two places:
  - Selection from current generation to take part in mating (parent selection)
  - Selection from parents + offspring to go into next generation (survivor selection)
- Selection operators work on whole individual
  - i.e. they are representation-independent
- Distinction between selection
  - operators: define selection probabilities
  - algorithms: define how probabilities are implemented

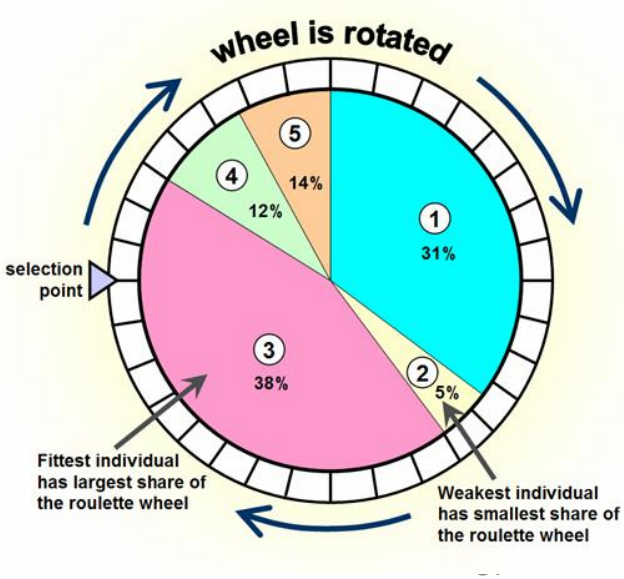
Implementation example: SGA

- Expected number of copies of an individual *i*

$$E(n_i) = \mu \cdot f(i) / \langle f \rangle$$

(μ = pop.size, f(i) = fitness of i, ⟨f⟩ avg. fitness in pop.)

- Roulette wheel algorithm:
  - Given a probability distribution, spin a 1-armed wheel *n* times to make *n* selections
  - No guarantees on actual value of n_i

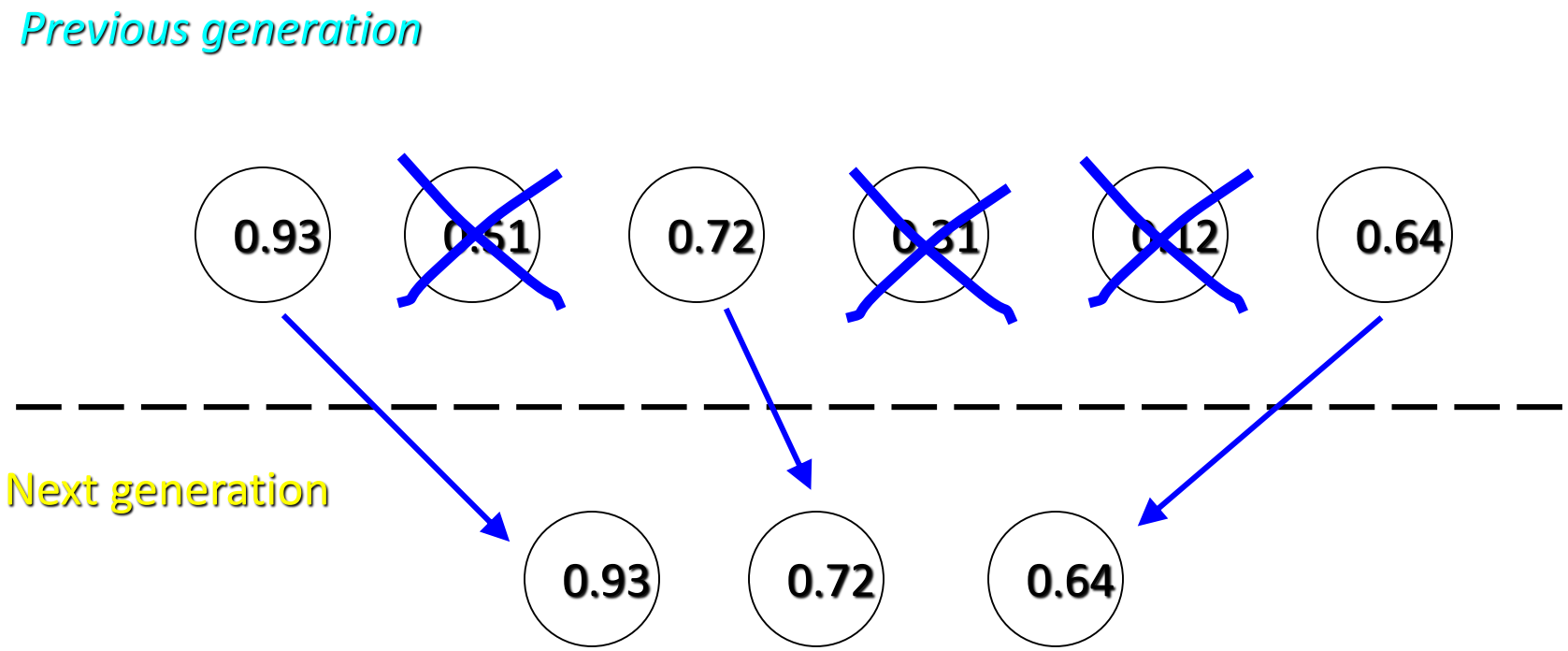


Fitness-Proportionate Selection

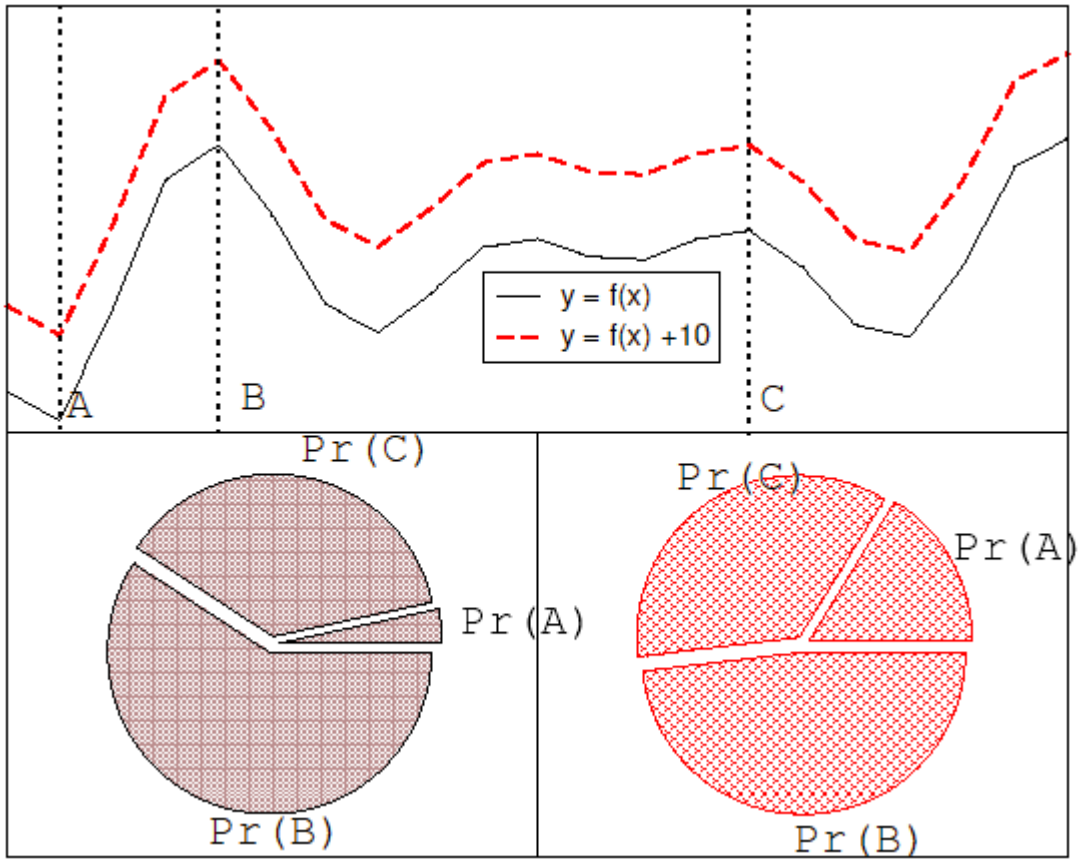
- Problems include
  - One highly fit member can rapidly take over if rest of population is much less fit: Premature Convergence
  - At end of runs when fitnesses are similar, lose selection pressure
  - Highly susceptible to function transposition
- Scaling can fix last two problems
  - Windowing: $f'(i) = f(i) - \beta^t$
    - where β is worst fitness in this (last n) generations
  - Sigma Scaling: $f'(i) = max(f(i) - (\langle f \rangle - c \cdot \sigma_f), 0.0)$
    - where c is a constant, usually 2.0

Selection

Survival of the Strongest



Function transposition for FPS



Rank – Based Selection

- Attempt to remove problems of FPS by basing selection probabilities on *relative* rather than *absolute* fitness
- Rank population according to fitness and then base selection probabilities on rank where fittest has rank μ and worst rank 1
- This imposes a sorting overhead on the algorithm, but this is usually negligible compared to the fitness evaluation time

Linear Ranking

$$P_{lin-rank}(i) = \frac{(2-s)}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}$$

- Parameterised by factor *s:* $1.0 < s \leq 2.0$
  - measures advantage of best individual
  - in GGA this is the number of children allotted to it
- Simple 3 member example

	Fitness	Rank	P_{selFP}	P_{selLR} $(s=2)$	P_{selLR} $(s=1.5)$
A	1	1	0.1	0	0.167
B	5	2	0.5	0.67	0.5
C	4	2	0.4	0.33	0.33
Sum	10		1.0	1.0	1.0

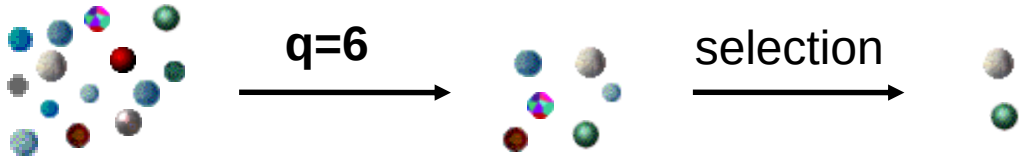
Exponential Ranking

$$P_{exp-rank}(i) = \frac{1 - e^{-i}}{c}.$$

- Linear Ranking is limited to selection pressure
- Exponential Ranking can allocate more than 2 copies to fittest individual
- Normalise constant factor *c* according to population size

Tournament Selection

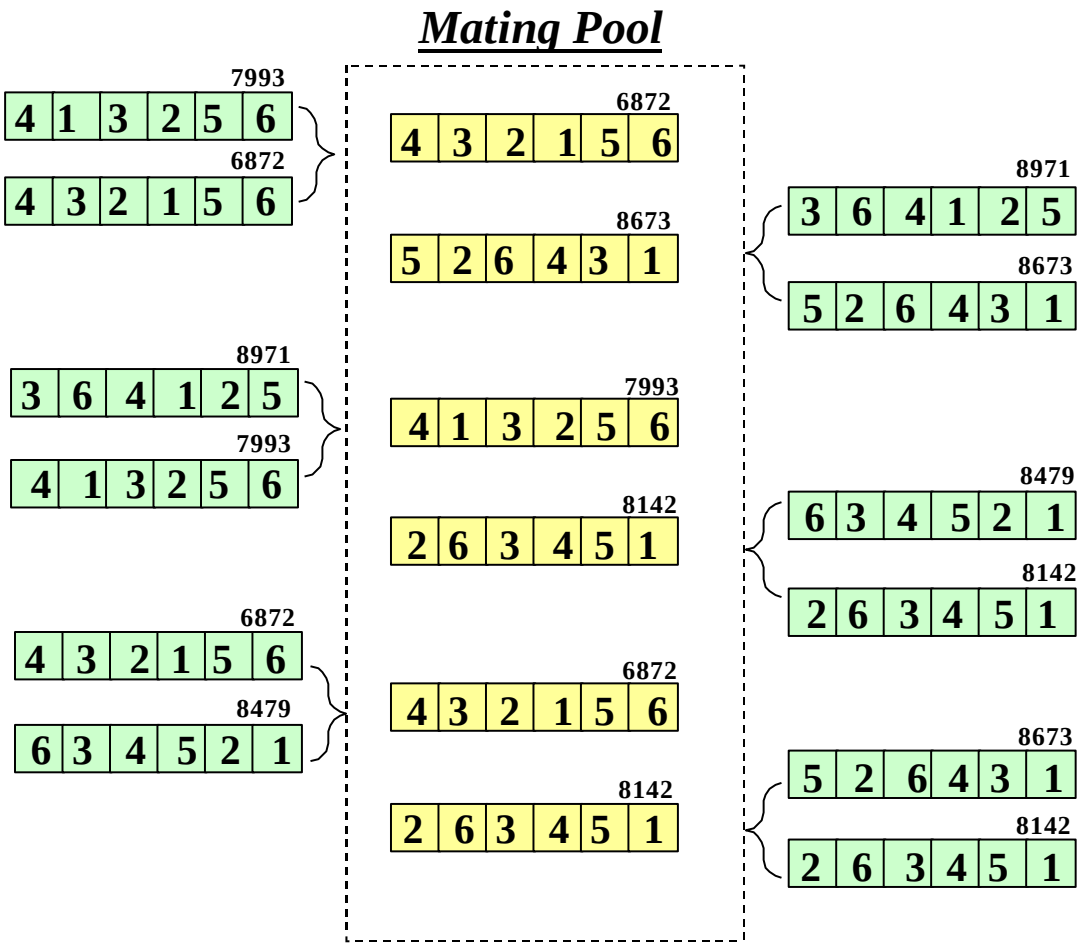
- All methods above rely on global population statistics
  - Could be a bottleneck esp. on parallel machines
  - Relies on presence of external fitness function which might not exist: e.g. evolving game players
- Informal Procedure:
  - Pick *k* members at random then select the best of these
  - Repeat to select more individuals



Tournament Selection 2

- Probability of selecting *i* will depend on:
  - Rank of *i*
  - Size of sample *k*
    - higher *k* increases selection pressure
  - Whether contestants are picked with replacement
    - Picking without replacement increases selection pressure
  - Whether fittest contestant always wins (deterministic) or this happens with probability *p*
- For $k = 2$, time for fittest individual to take over population is the same as linear ranking with $s = 2 \cdot p$

Tournament Selection



Survivor Selection

- Most of methods above used for parent selection
- Survivor selection can be divided into two approaches:
  - Age-Based Selection
    - e.g. SGA
    - In SSGA can implement as “delete-random” (not recommended) or as first-in-first-out (a.k.a. delete-oldest)
  - Fitness-Based Selection
    - Using one of the methods above or

Two Special Cases

- Elitism
  - Widely used in both population models (GGA, SSGA)
  - Always keep at least one copy of the fittest solution so far
- GENITOR: a.k.a. “delete-worst”
  - From Whitley’s original Steady-State algorithm (he also used linear ranking for parent selection)
  - Rapid takeover : use with large populations or “no duplicates” policy

Examples

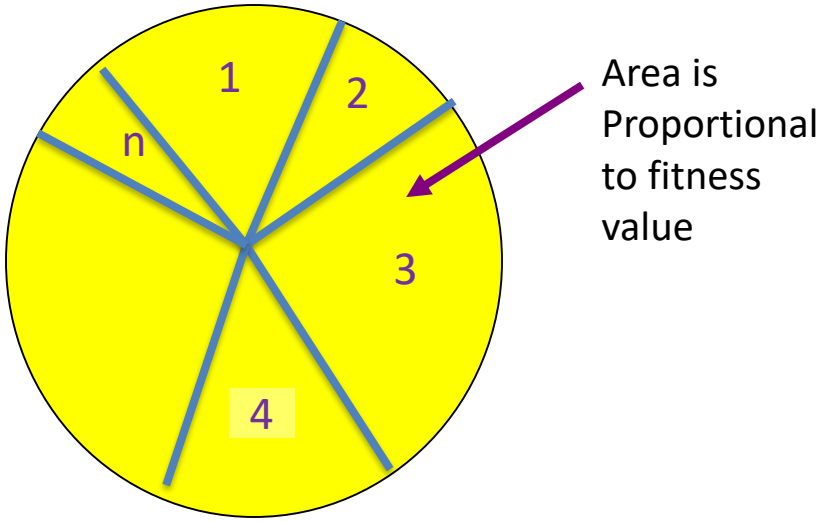
- **Maxone**
- Travelling Salesperson
- Diophantine Equation
- Knapsack
- Arithmetic

Example (selection1)

Next we apply fitness proportionate selection with the roulette wheel method:

Individual *i* will have a probability to be chosen $\frac{f(i)}{\sum_i f(i)}$

We repeat the extraction as many times as the number of individuals we need to have the same parent population size (6 in our case)



Example (selection2)

Suppose that, after performing selection, we get the following population:

s_1` = 1111010101 (s_1)

s_2` = 1110110101 (s_3)

s_3` = 1110111101 (s_5)

s_4` = 0111000101 (s_2)

s_5` = 0100010011 (s_4)

s_6` = 1110111101 (s_5)

Example (crossover1)

Next we mate strings for crossover. For each couple we decide according to crossover probability (for instance 0.6) whether to actually perform crossover or not

Suppose that we decide to actually perform crossover only for couples (s_1`, $s_2$`) and (s_5`, $s_6$`). For each couple, we randomly extract a crossover point, for instance 2 for the first and 5 for the second

Example (crossover2)

Before crossover:

s_1` = 1111010101

s_2` = 1110110101

s_5` = 0100010011

s_6` = 1110111101

After crossover:

s_1`` = 1110110101

s_2`` = 1111010101

s_5`` = 0100011101

s_6`` = 1110110011

Example (mutation1)

The final step is to apply random mutation: for each bit that we are to copy to the new population we allow a small probability of error (for instance 0.1)

Before applying mutation:

s_1`` = 1110110101

s_2`` = 1111010101

s_3`` = 1110111101

s_4`` = 0111000101

s_5`` = 0100011101

s_6`` = 1110110011

Example (mutation2)

After applying mutation:

$s_1^{```} = 1110100101 \quad f(s_1^{```}) = 6$

$s_2^{```} = 1111110100 \quad f(s_2^{```}) = 7$

$s_3^{```} = 1110101111 \quad f(s_3^{```}) = 8$

$s_4^{```} = 0111000101 \quad f(s_4^{```}) = 5$

$s_5^{```} = 0100011101 \quad f(s_5^{```}) = 5$

$s_6^{```} = 1110110001 \quad f(s_6^{```}) = 6$

Example (end)

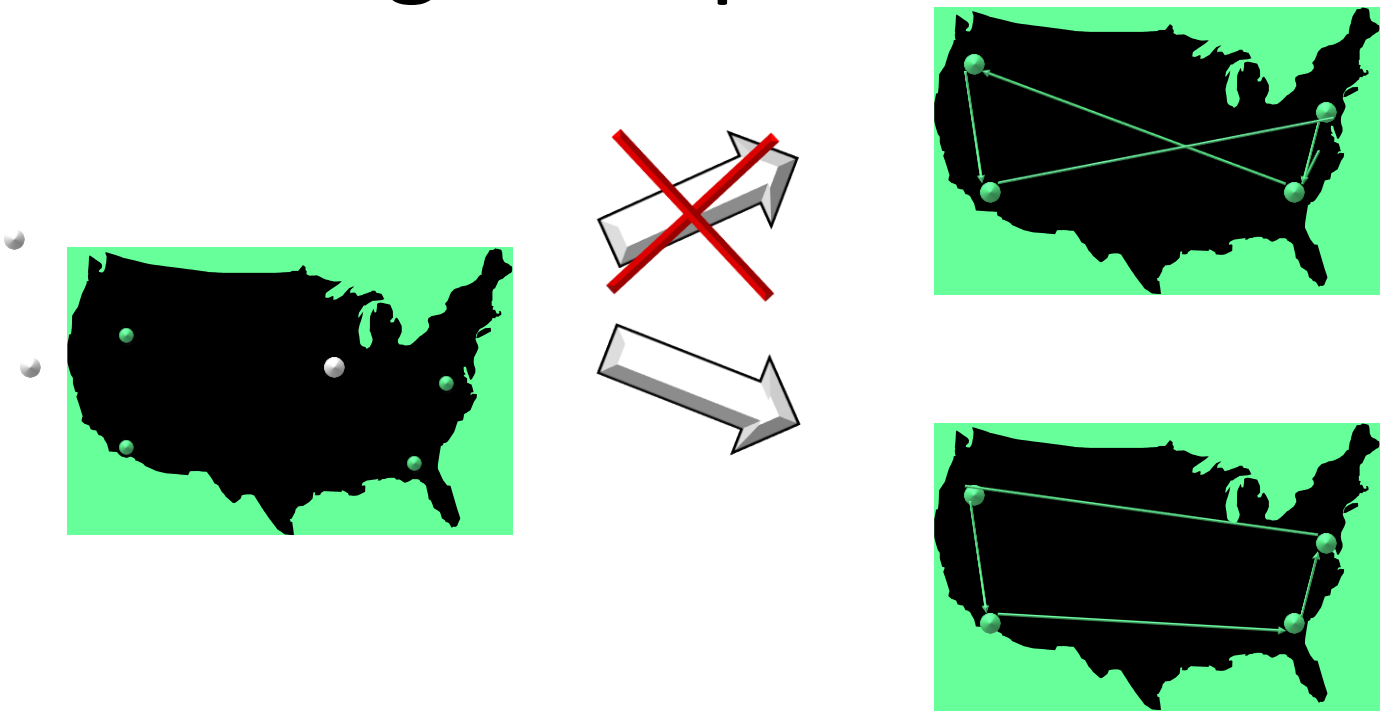
In one generation, the total population fitness changed from 34 to 37, thus improved by ~9%

At this point, we go through the same process all over again, until a stopping criterion is met

Examples

- Maxone
- **Travelling Salesperson**
- Diophantine Equation
- Knapsack

Traveling Salesperson Problem



Find a tour of a given set of cities so that

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

Encoding

- Represent every city with an integer .

- Consider 6 Indian cities –

  Mumbai, Nagpur , Calcutta, Delhi, Bangalore and Pune
  assign a number to each.

Mumbai → 1
Nagpur → 2
Calcutta → 3
Delhi → 4
Bangalore → 5
Pune → 6

Encoding

- Thus a path would be represented as a sequence of integers from 1 to 6.

- The path [1 2 3 4 5 6] represents a path from

Mumbai to Nagpur - Nagpur to Calcutta - Calcutta to Delhi - Delhi to Bangalore - Bangalore to Pune and Pune to Mumbai.

Mumbai	1
Nagpur	2
Calcutta	3
Delhi	4
Pune	6

Fitness Function

- The fitness function will be the total cost of the tour represented by each chromosome.
- This can be calculated as the sum of the distances traversed in each travel segment.

The Lesser The Sum, The Fitter The Solution Represented By That Chromosome.

Distance/Cost Matrix For TSP

	1	2	3	4	5	6
1	0	863	1987	1407	998	163
2	863	0	1124	1012	1049	620
3	1987	1124	0	1461	1881	1844
4	1407	1012	1461	0	2061	1437
5	998	1049	1881	2061	0	841
6	163	620	1844	1437	841	0

Fitness Function (contd.)

- So, for a chromosome [4 1 3 2 5 6], the total cost of travel or fitness will be calculated as shown below

- Fitness $= 1407 + 1987 + 1124 + 1049 + 841$

  $= 6408$ *kms.*

- Since our objective is to Minimize the distance, the lesser the total distance, the fitter the solution.

Initial Population for TSP

(5,3,4,6,2)	(2,4,6,3,5)	(4,3,6,5,2)
(2,3,4,6,5)	(4,3,6,2,5)	(3,4,5,2,6)
(3,5,4,6,2)	(4,5,3,6,2)	(5,4,2,3,6)
(4,6,3,2,5)	(3,4,2,6,5)	(3,6,5,1,4)

Select Parents

(5,3,4,6,2)	(2,4,6,3,5)	(4,3,6,5,2)
(2,3,4,6,5)	(4,3,6,2,5)	(3,4,5,2,6)
(3,5,4,6,2)	(4,5,3,6,2)	(5,4,2,3,6)
(4,6,3,2,5)	**(3,4,2,6,5)**	(3,6,5,1,4)

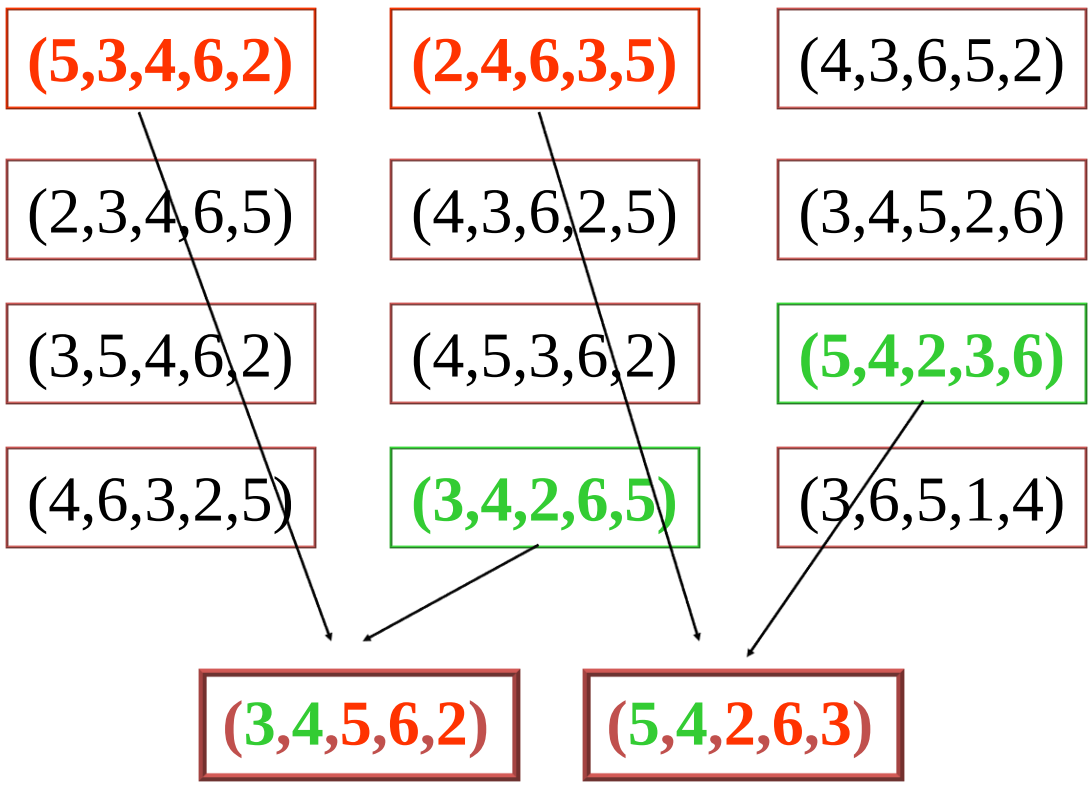
Try to pick the better ones.

Create Off-Spring – 1 point

(5,3,4,6,2)	(2,4,6,3,5)	(4,3,6,5,2)
(2,3,4,6,5)	(4,3,6,2,5)	(3,4,5,2,6)
(3,5,4,6,2)	(4,5,3,6,2)	(5,4,2,3,6)
(4,6,3,2,5)	**(3,4,2,6,5)**	(3,6,5,1,4)

(3,4,5,6,2)

Create More Offspring



Mutate

(5,3,4,6,2)	**(2,4,6,3,5)**	(4,3,6,5,2)
(2,3,4,6,5)	**(4,3,6,2,5)**	(3,4,5,2,6)
(3,5,4,6,2)	(4,5,3,6,2)	**(5,4,2,3,6)**
(4,6,3,2,5)	**(3,4,2,6,5)**	(3,6,5,1,4)

(3,4,5,6,2) **(5,4,2,6,3)**

Mutate

(5,3,4,6,2)	**(2,4,6,3,5)**	(4,3,6,5,2)
(2,3,4,6,5)	**(2,3,6,4,5)**	(3,4,5,2,6)
(3,5,4,6,2)	(4,5,3,6,2)	**(5,4,2,3,6)**
(4,6,3,2,5)	**(3,4,2,6,5)**	(3,6,5,1,4)

(3,4,5,6,2) **(5,4,2,6,3)**

Eliminate

(5,3,4,6,2)	**(2,4,6,3,5)**	~~(4,3,6,5,2)~~
~~(2,3,4,6,5)~~	**(2,3,6,4,5)**	(3,4,5,2,6)
(3,5,4,6,2)	(4,5,3,6,2)	**(5,4,2,3,6)**
(4,6,3,2,5)	**(3,4,2,6,5)**	(3,6,5,1,4)

(3,4,5,6,2) **(5,4,2,6,3)**

Tend to kill off the worst ones.

Integrate

(5,3,4,6,2)	**(2,4,6,3,5)**	**(5,4,2,6,3)**
(3,4,5,6,2)	**(2,3,6,4,5)**	(3,4,5,2,6)
(3,5,4,6,2)	(4,5,3,6,2)	**(5,4,2,3,6)**
(4,6,3,2,5)	**(3,4,2,6,5)**	(3,6,5,1,4)

Restart

(5,3,4,6,2)	(2,4,6,3,5)	(5,4,2,6,3)
(3,4,5,6,2)	(2,3,6,4,5)	(3,4,5,2,6)
(3,5,4,6,2)	(4,5,3,6,2)	(5,4,2,3,6)
(4,6,3,2,5)	(3,4,2,6,5)	(3,6,5,1,4)

Examples

- Maxone
- Travelling Salesperson
- **Diophantine Equation**
- Knapsack

Diophantine Equation

- Diophantine (only integer solutions) equation: $a+2b+3c+4d=30$, where a, b, c, d are positive integers.
- Using a genetic algorithm to solve above equation.
- why not just use a brute force method ?

Solution

- First choose 5 random initial solution sets, with constraints 1 =< a, b, c, d =< 30.

Chromosome	(a, b, c, d)
1	(1,28,15,3)
2	(14,9,2,4)
3	(13,5,7,3)
4	(23,8,16,19)
5	(9,13,5,2)

Solution cont..

- Then calculate the fitness values for above solution

Chromosome	Fitness Value
1	|114-30|=84
2	|54-30|=24
3	|56-30|=26
4	|163-30|=133
5	|58-30|=28

Solution cont..

- Calculate the likelihood.
- Fitness values that are lower are closer to the desired answer.

Chromosome	Likelihood
1	(1/84)/0.135266 = 8.80%
2	(1/24)/0.135266 = 30.8%
3	(1/26)/0.135266 = 28.4%
4	(1/133)/0.135266 = 5.56%
5	(1/28)/0.135266 = 26.4%

Solution cont..

- Select parents according to likelihoods calculated above

Father	Mother
3	1
5	2
3	5
2	5
5	3

Solution cont..

- The offspring of each of these parents contains the genetic information of both father and mother.

Father	Mother	Offspring
a_1 | b_1,c_1,d_1	a_2 | b_2,c_2,d_2	a_1,b_2,c_2,d_2 or a_2,b_1,c_1,d_1
a_1,b_1 | c_1,d_1	a_2,b_2 | c_2,d_2	a_1,b_1,c_2,d_2 or a_2,b_2,c_1,d_1
a_1,b_1,c_1 | d_1	a_2,b_2,c_2 | d_2	a_1,b_1,c_1,d_2 or a_2,b_2,c_2,d_1

Solution cont..

- There are many ways in which parents can trade genetic information to create an offspring, crossing over is just one way.

Solution cont..

- Simulated Crossovers from Parent Chromosomes

Father Chromosome	Mother Chromosome	Offspring Chromosome
(13 | 5,7,3)	(1 | 28,15,3)	(13,28,15,3)
(9,13 | 5,2)	(14,9 | 2,4)	(9,13,2,4)
(13,5,7 | 3)	(9,13,5 | 2)	(13,5,7,2)
(14 | 9,2,4)	(9 | 13,5,2)	(14,13,5,2)
(13,5 | 7, 3)	(9,13 | 5, 2)	(13,5,5,2)

Solution cont..

- Again we calculate fitness values for new generation of offsprings.

Offspring Chromosome	Fitness Value
(13,28,15,3)	|126-30|=96
(9,13,2,4)	|57-30|=27
(13,5,7,2)	|57-30|=22
(14,13,5,2)	|63-30|=33
(13,5,5,2)	|46-30|=16

Solution cont..

- The average fitness value for the offspring chromosomes were 38.8, while the average fitness value for the parent chromosomes were 59.4.
- Progressing at this rate, one chromosome should eventually reach a fitness level of 0 eventually, that is when a solution is found.

Examples

- Maxone
- Travelling Salesperson
- Diophantine Equation
- **Knapsack**

Example: The Knapsack Problem

- You are going on an overnight hike and have a number of items that you could take along. Each item has a weight (in pounds) and a benefit or value to you on the hike(for measurements sake let's say, in US dollars), and you can take one of each item at most. There is a capacity limit on the weight you can carry (constraint). This problem only illustrates one constraint, but in reality there could be many constraints including volume, time, etc.

GA Example: The Knapsack Problem

- **Item: 1 2 3 4 5 6 7**
- **Benefit: 5 8 3 2 7 9 4**
- **Weight: 7 8 4 10 4 6 4**
- **Knapsack holds a maximum of 22 pounds**
- **Fill it to get the maximum benefit**
- **Solutions take the form of a string of 1's and 0's**
- **Solutions: Also known as strings of genes called Chromosomes**
  - 1. 0101010
  - 2. 1100100
  - 3. 0100111

Example: The Knapsack Problem

- We represent a solution as a string of seven 1s and 0s and the fitness function as the total benefit, which is the sum of the gene values in a string solution times their representative benefit coefficient.
- The method generates a set of random solutions (initial parents), uses total benefit as the fitness function and selects the parents randomly to create generations of offspring by crossover and mutation.

Knapsack Example

- Typically, a string of 1s and 0s can represent a solution.
- Possible solutions generated by the system using *Reproduction, Crossover*, or *Mutations*
  - 1. 0101010
  - 2. 1100100
  - 3. 0100111

Knapsack Example
Solution 1

Item	1	2	3	4	5	6	7
Solution	0	1	0	1	0	1	0
Benefit	5	8	3	2	7	9	4
Weight	7	8	4	10	4	6	4

- Benefit 8 + 2 + 9 = 19
- Weight 8 + 10 + 6 = 24

Knapsack Example
Solution 2

Item	1	2	3	4	5	6	7
Solution	1	1	0	0	1	0	0
Benefit	5	8	3	2	7	9	4
Weight	7	8	4	10	4	6	4

- Benefit 5 + 8 + 7 = 20
- Weight 7 + 8 + 4 = 19

Knapsack Example
Solution 3

Item	1	2	3	4	5	6	7
Solution	0	1	0	0	1	1	1
Benefit	5	8	3	2	7	9	4
Weight	7	8	4	10	4	6	4

- Benefit 8 + 7 + 9 + 4 = 28
- Weight 8 + 4 + 6 + 4 = 22

Knapsack Example

- Solution 3 is clearly the best solution and has met our conditions, therefore, item number 2, 5, 6, and 7 will be taken on the hiking trip. We will be able to get the most benefit out of these items while still having weight equal to 22 pounds.
- This is a simple example illustrating a genetic algorithm approach.

Knapsack Problem

To understand GA must work with the following problem:

(Knap Sack Problem)

- Thief wants to steal gold store.
- Thief has a bag(the bag can hold a specific weight).
- Every piece of gold has a specific weight and price.
- Thief wants to steal gold with high price but the weight must equal or less than the weight that bag can hold it.
- If we gave every gold piece a specific number 1,2,3,…,n(suggest n=8 in this example).

1-Encoding (*representation*)(gene,chromosome)

Chromosome could be:

- Bit strings (101101010100).
- Real numbers (43.1,45.2,66.3,11.0).
- Permutation of elements (E11 E3 E7 … E1 E15).
- Integer Numbers (11,12,54,98,625,1).
- Any data structures.
- In knap sack problem can represent any solution as chromosome by using bit string of length 8.

Ex:- 1 1 0 1 0 0 0 1

8 7 6 5 4 3 2 1

1(gene):this piece taken, 0(gene):this piece untaken.

2-Initialize population

- Implementers specify population size .
- To initialize population create chromosomes randomly and store them in list of length the population size.
- In our problem lets take population size 6 chromosomes.
- We can initialize population a following:

3-Evaluation of population.

- Giving every chromosome in population **fitness value** by using **fitness function**.
- **Fitness functions** will differ according to the problem and encoding technique.
- **Fitness function** returns a single numerical(**fitness**) which reflex the **utility** or the **ability** of the individual which that chromosome represents.
- Fitness function can calculate : strength, weight, width, maximum load,cost,construction time or combination of all these.
- Fitness value well be stored with chromosome.

In our example we will make fitness function as the sum of price of all gold pieces.

-To complete our example must apply fitness function on all chromosomes.

	Chromosome	fitness value

	1	2	3	4	5	6	7	8
weight	5	3	10	6	5	5	4	4
price	50	50	75	50	150	250	30	100

4-Selection of new parents(*reproduction*)

- Individuals are selected from population randomly or by using any selection method to improve the population itself.
- Good individuals will probably be selected several times in a generation ,poor ones may not be at all.

- ***Methods of selection***

Random ,Best, Tournament, Roulette wheel, Truncation, Rank, Exponential, Boltzman, Steady state, Interactive and binary tournament selection.

- In our example we will use **Truncation** selection with parameter 3.
- search best 3 chromosomes and then select 6 chromosomes randomly from these three chromosomes and store them as new population to be used in the next step.

Old population

Search best 3 chromosomes

New population

5-Crossover

- Crossover is performed with probability ***Pcross*** (crossover probability or crossover rate) between two selected individuals, called ***parents***, by exchanging parts of their genes to form two new individuals called ***offsprings.***
- The simplest method know as ***single point crossover***.
- Single point crossover take 2 individual and cut their chromosome strings at randomly chosen position, to produce 2 ***head*** segments and 2 ***tail*** segments .The tail segments are then swapped over to produce 2 new full length chromosomes.
- ***Pcross*** (crossover probability or crossover rate) is typically between 0.6 and 1.0
- There is also **Multi-Point, Uniform, Bit Simulated, Problem Centered** and **specialized crossover** techniques.

we will do it just for first two parents.

Pcross=0.6

Select two parent(1,2)

Generate random number between 0.0-1.0(0.3)

0.3<=0.6(yes) apply crossover

generate random number between 1-8(3)

old 0 1 0 1 0 1 0 1 0 0 0 0 1 1 1 1

new 0 1 0 1 0 1 1 1 0 0 0 0 1 1 0 1 swap tails

Do this for each pair in population.

6-Mutation

- Applied to each child individually after crossover .
- It alters some of genes in chromosome with small probability .
- Must specify **Pmut**(mutation probability that is relatively small) therefore a few number of chromosomes will be mutated.
- In our example:

suppose **Pmut** =0.2

generate number between 0-1 (0.01)

0.01<=0.2(yes) apply mutation.

Generate number between 1-8(6)

0 1 **0** 1 0 1 0 1 => 0 1 **1** 1 0 1 0 1

Do this for each chromosome in population.

Termination Criteria

There exist three termination condition type:

1-*Time*:in seconds, in minutes and may be in hours according to the problem that you have it.

2-*Number of generations*: in hundreds, in thousands may be in millions according to the problem you have it.

3-*convergence*: when 95% of populations have the same fitness value we can say the convergence started to appear and the user can stop its genetic program to take the result.

The Knapsack Problem

- The knapsack problem, though simple, has many important applications including determining what items to take on a space ship mission.

Summary

Given an exponential problem to solve, ...

- Encoding technique (*gene, chromosome*)
- Initialization procedure *(creation)*
- Evaluation function *(environment)*
- Selection of parents *(reproduction)*
- Genetic operators *(mutation, recombination)*
- Parameter settings *(practice and art)*