Genetic Algorithms

Note: today and tomorrow, we will go through lots and lots of examples of components of Evolutionary Algorithms

(I will give you lots of building blocks to experiment with later…)

GA Quick Overview

- Developed: USA in the 1970's
- Early names: J. Holland and K. DeJong (1975)
- Typically applied to:
  - Discrete & continuous 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

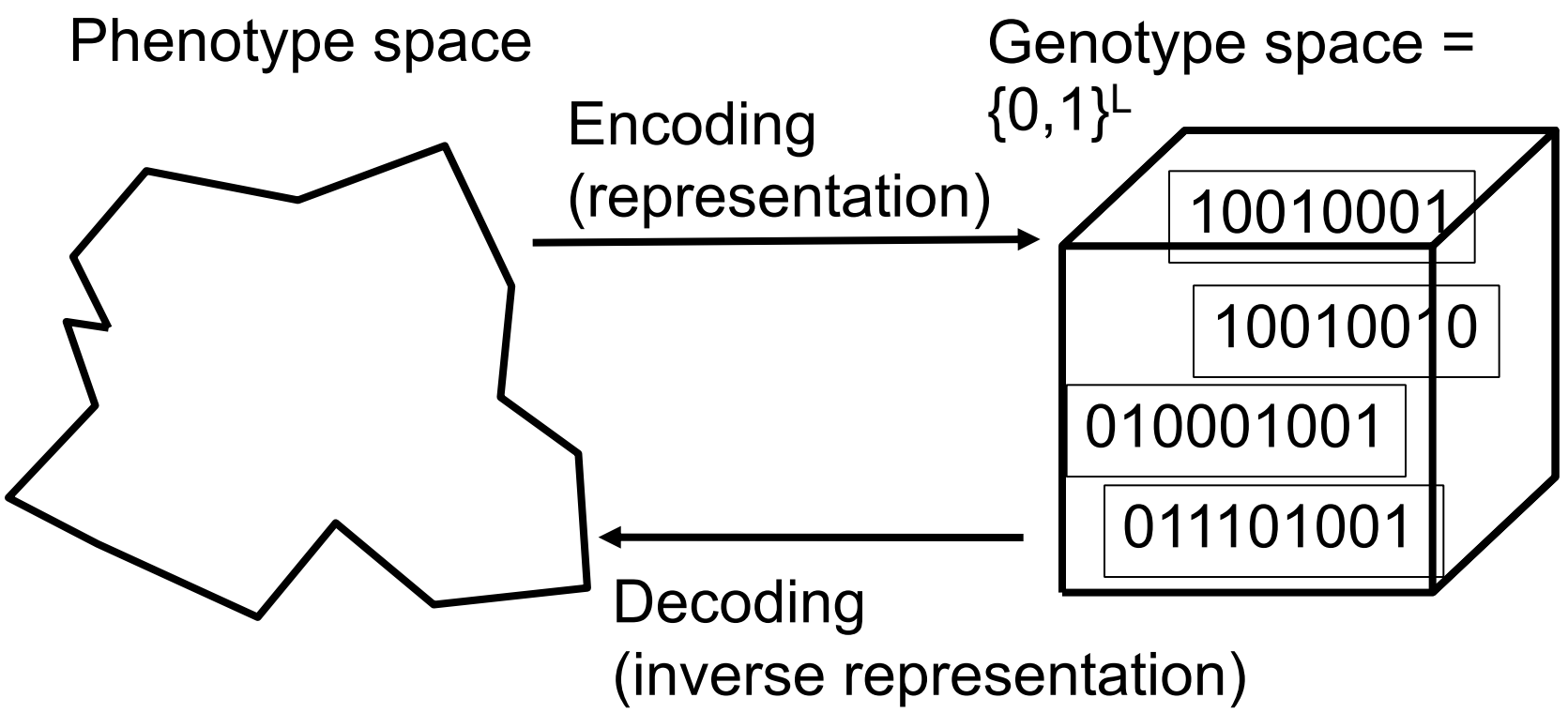
Genetic algorithms

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

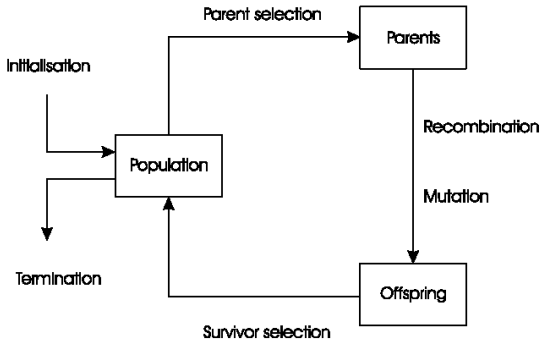
SGA technical summary

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



SGA reproduction cycle



1. Select parents for the mating pool

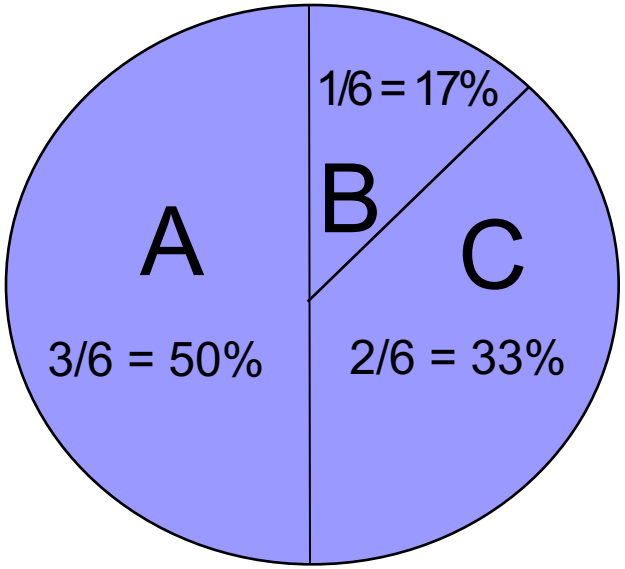
   (size of mating pool = population size)
2. For each consecutive pair apply crossover with probability p_c , otherwise copy parents
3. For each offspring apply mutation (bit-flip with probability p_m independently for each bit)
4. Replace the whole population with the resulting offspring

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

18 red, 18 black, 1 green
➔ 37 numbers in total



fitness(A) = 3

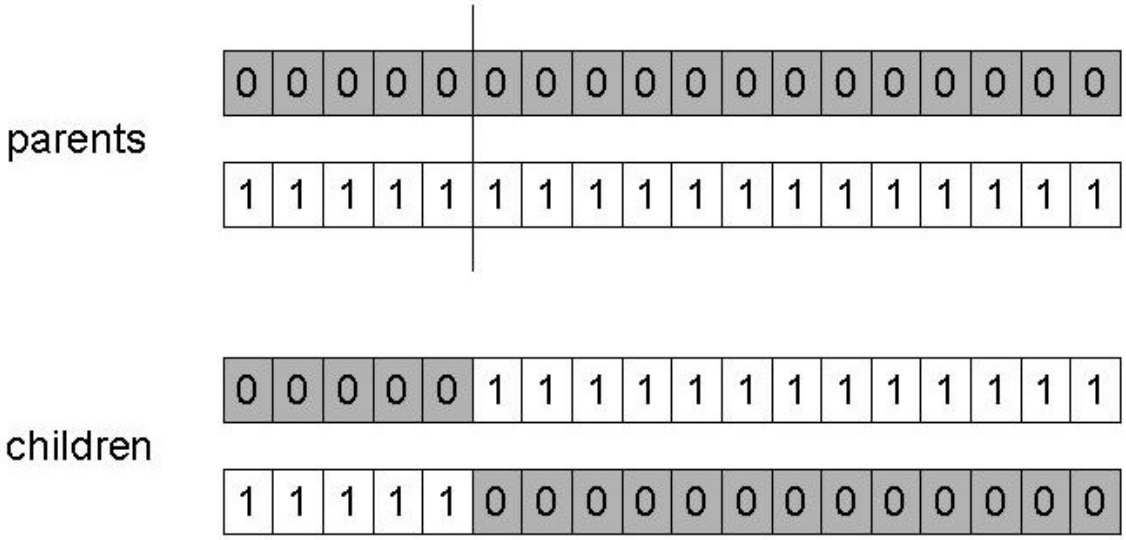
fitness(B) = 1

fitness(C) = 2

http://i.ytimg.com/vi/0Zj_9ypBnzg/maxresdefault.jpg

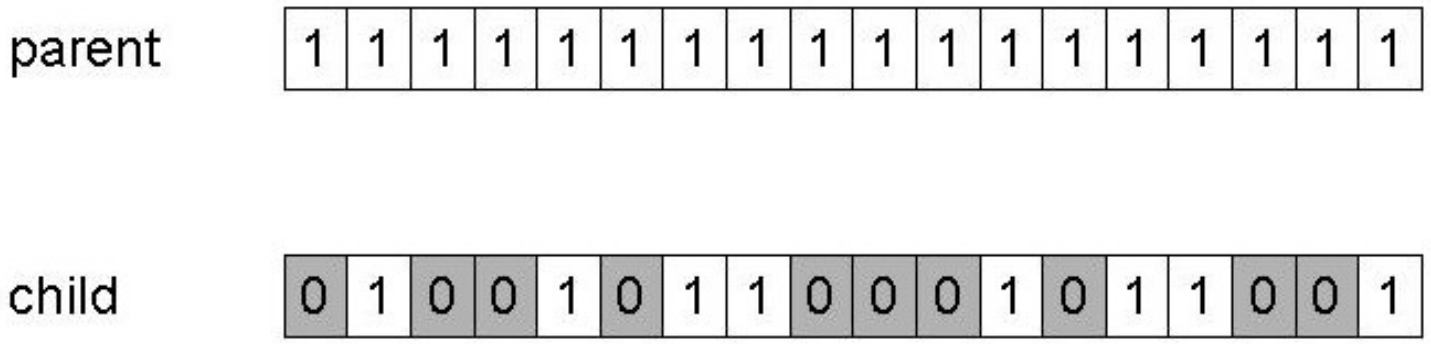
SGA operators: 1-point crossover

- 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

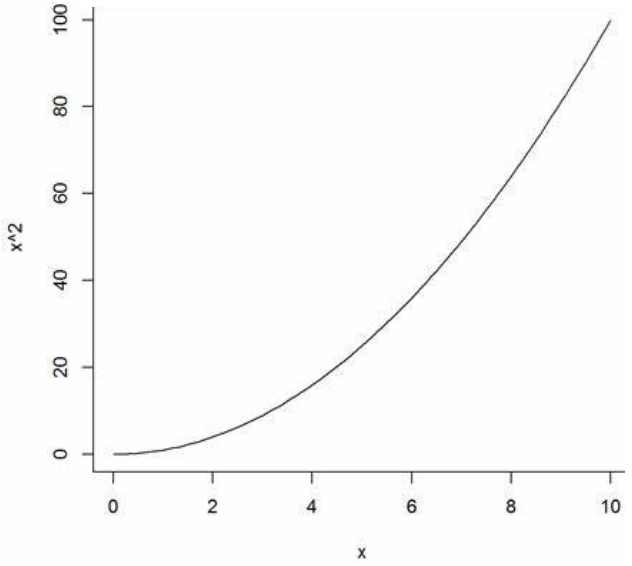
- Alter each gene independently with a probability p_m
- p_m is called the mutation rate
  - Typically between 1/pop_size and 1/chromosome_length



Here: 18 bits

An example [Goldberg ’89]

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



x^2 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					
2	1 1 0 0 0					
3	0 1 0 0 0					
4	1 0 0 1 1					
Sum						
Average						
Max						

x^2 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

x^2 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			
2	1 1 0 0 | 0	4			
2	1 1 | 0 0 0	2			
4	1 0 | 0 1 1	2			
Sum					1170
Average					293
Max					576

Reminder: at the beginning of the generation

x^2 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

Reminder: at the beginning of the generation

1170
293
576

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			
2	1 1 0 0 1			
2	1 1 0 1 1			
4	1 0 0 0 0			
Sum				
Average				
Max				

Reminder:
at the beginning
of the generation

1170
293
576

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	28	784
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				2462
Average				615.5
Max				784

Reminder:
at the beginning
of the generation

1170
293
576

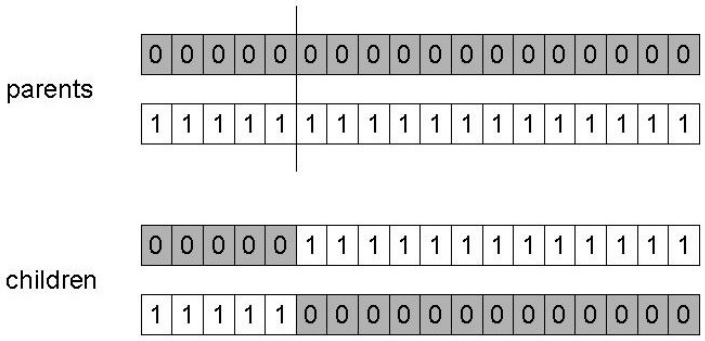
The simple GA

- Has been subject of many (early) studies
  - For a very long time often used as benchmark for novel GAs (not nowadays anymore…)
- Shows many shortcomings, e.g.,
  - Representation is too restrictive
  - Mutation & crossovers only applicable for bit-string & integer representations
  - Generational population model (step 6 in SGA repr. cycle) can be improved with explicit survivor selection

Alternative Crossover Operators

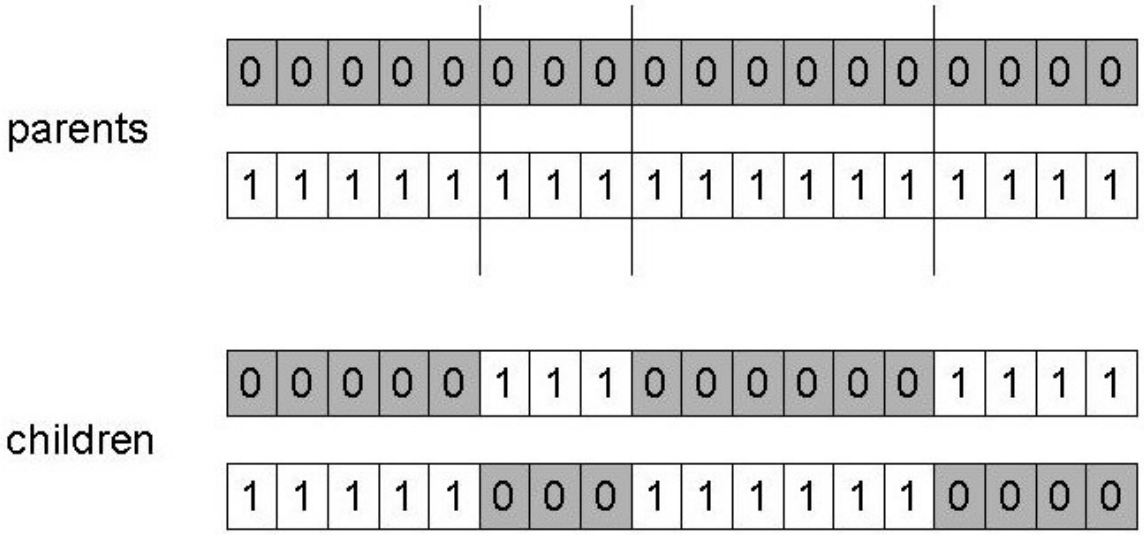
Performance with 1 Point Crossover depends on the order that variables occur in the representation

- 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



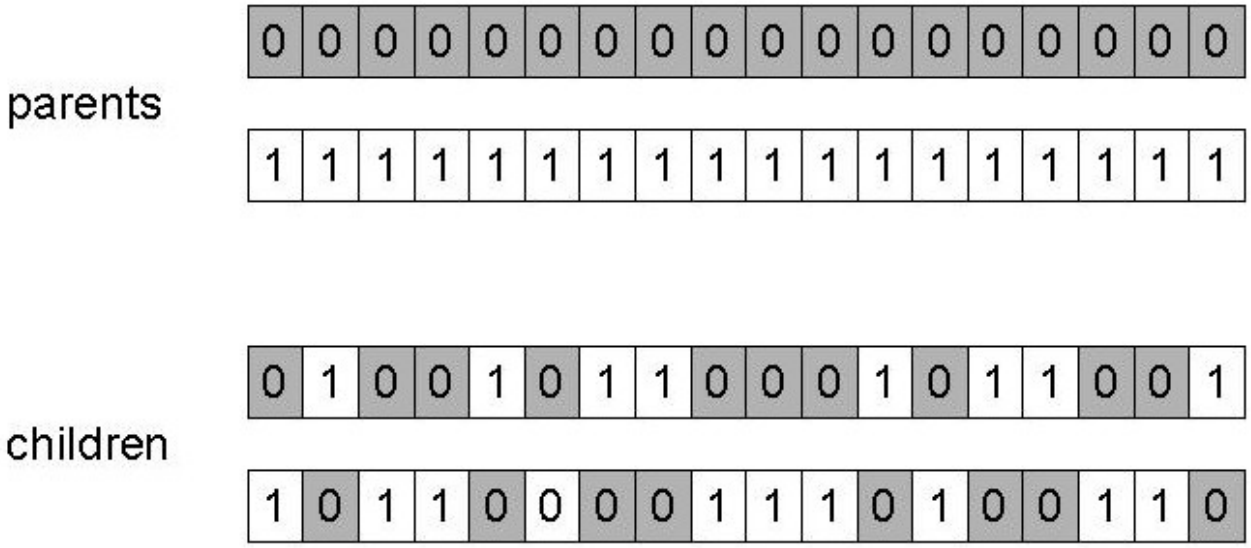
n-point crossover

- 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



Crossover OR mutation?

- A long debate: which one is better / necessary / main vs. background

Onlineclicker.org: ZZZZ
Think by yourself (no talking!):
What is best?
1. mutation
2. crossover

Crossover OR mutation?

- A long debate: which one is better / necessary / main vs. background

Onlineclicker.org: 8818
Think by yourself (no talking!):
What is best?
1. mutation
2. crossover

Crossover OR mutation?

- A long debate: which one is better / necessary / main vs. background

Discuss with your neighbours / in the Zoom Breakout Room!
What is best?
1. mutation
2. crossover

Crossover OR mutation?

- A long debate: which one is better / necessary / main vs. background

Onlineclicker.org: 8819
What is best?
1. mutation
2. crossover

Crossover OR mutation?

- A long debate: which one is better / necessary / main vs. background

- Answer (at least, rather wide agreement):
  - it depends on the problem, but
  - in general, it is good to have both
  - both have different roles
  - mutation-only-EA is possible, xover-only-EA generally 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 existing information.

- 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
  (note: in different stages of the optimisation, the operators can have different effects) [note: not always true]

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

Permutation Representations

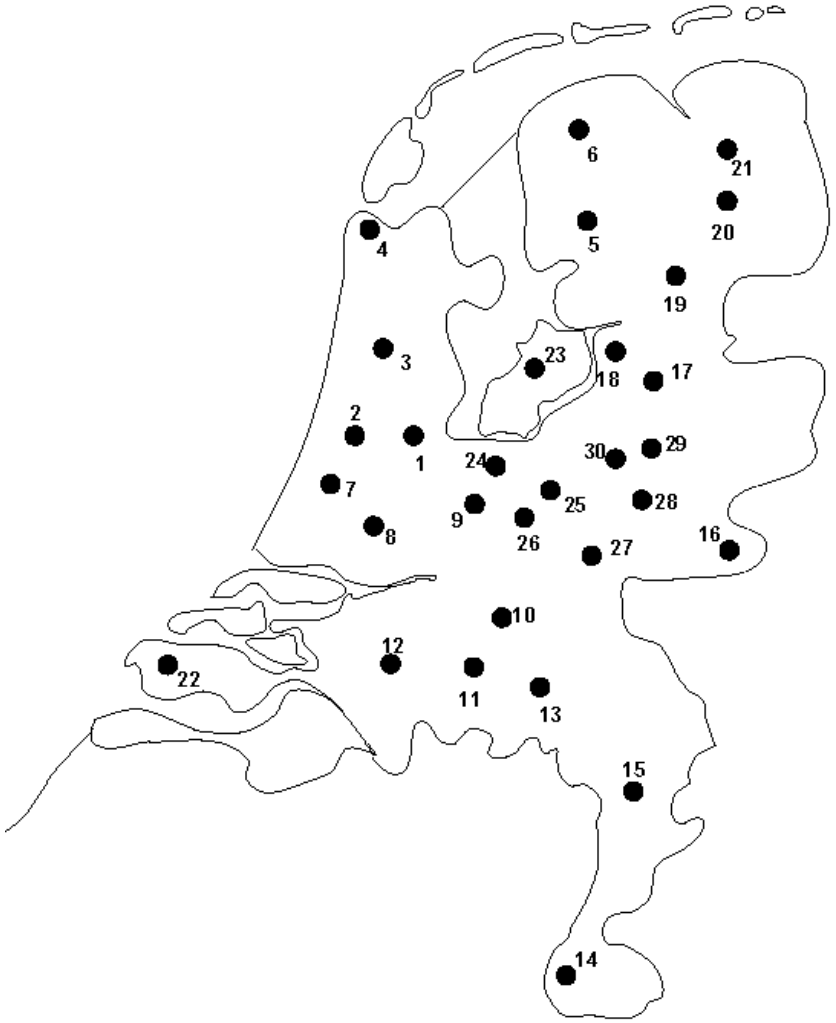
- Ordering/sequencing problems form a special type
- Task is (or can be solved by) arranging some objects in a certain order
  - Example: sort algorithm: important thing is which elements occur before others (order)
  - Example: Travelling Salesman Problem (TSP): important thing is which elements occur next to each other (adjacency)
- Solutions to these problems are generally expressed as permutations:
  - 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

  TSP is NP-hard
- 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

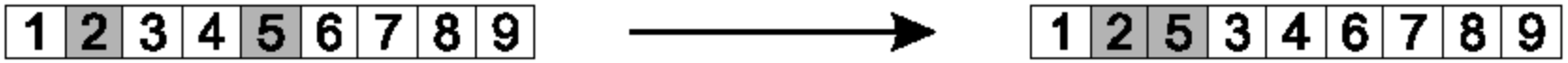


Mutation operators for permutations

- 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

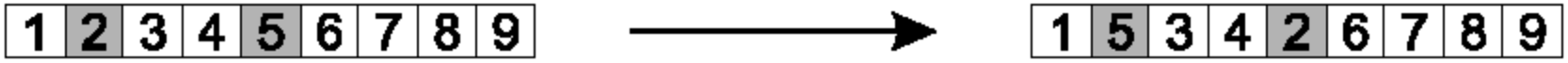
Insert Mutation for permutations

- 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



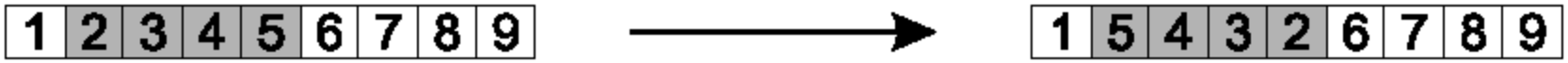
Swap mutation for permutations

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



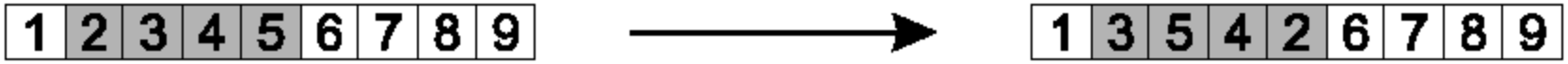
Inversion mutation for permutations

- 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

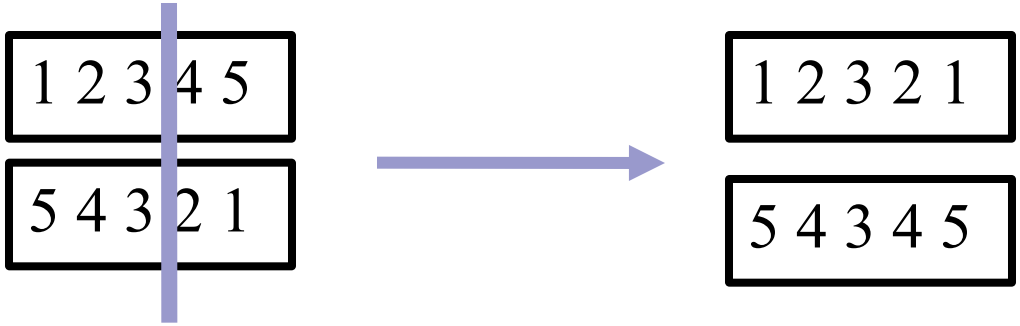
- Pick a subset of genes at random
- Randomly rearrange the alleles in those positions



(note: subset does not have to be contiguous)

Crossover operators for permutations

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



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

Order crossover (version 1)

The 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 crossover (version 2)

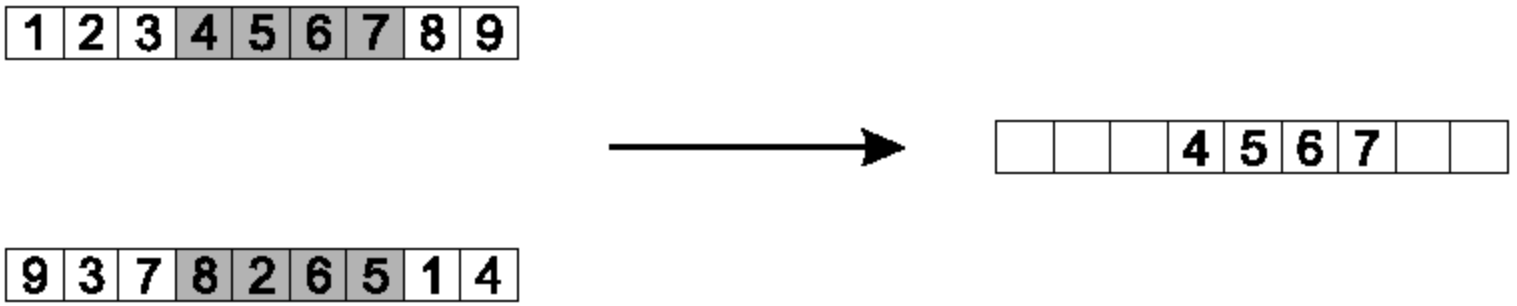
The idea is to preserve relative order that elements occur.

Informal procedure:

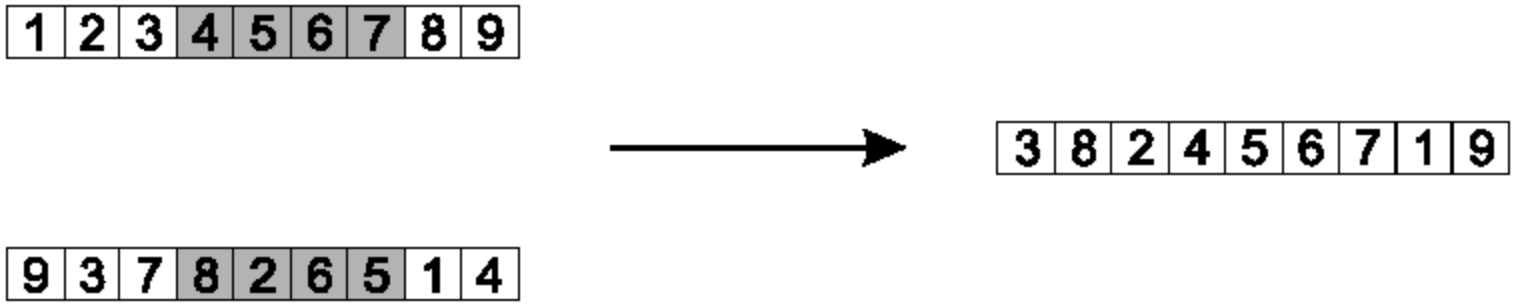
1. Choose an arbitrary leftmost part from the first parent
2. Copy this part to the first child
3. Copy the numbers that are not in the first part, to the first child:
   - starting ~~right from cut point of the copied part~~ from the left,
   - using the **order** of the second parent
   - ~~and wrapping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

Order crossover example (version 1)

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:

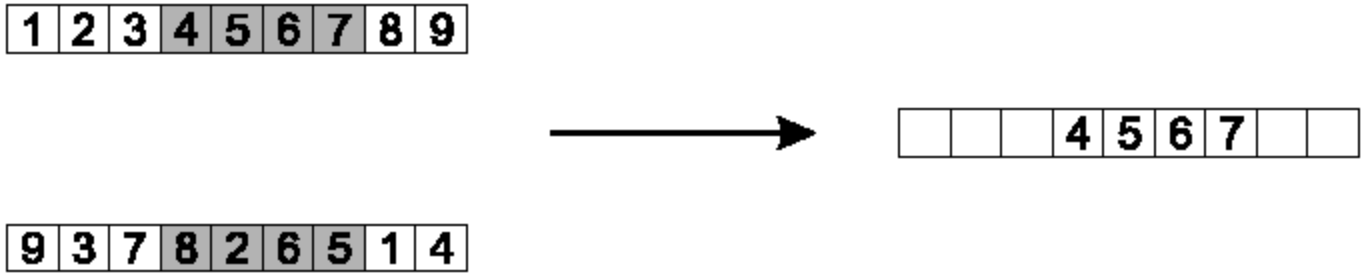
1. Choose random segment and copy it from P1
2. Starting from the first crossover point look for elements in that segment of P2 that have not been copied
3. For each of these *i* look in the offspring to see what element *j* has been copied in its place from P1
4. Place *i* into the position occupied *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

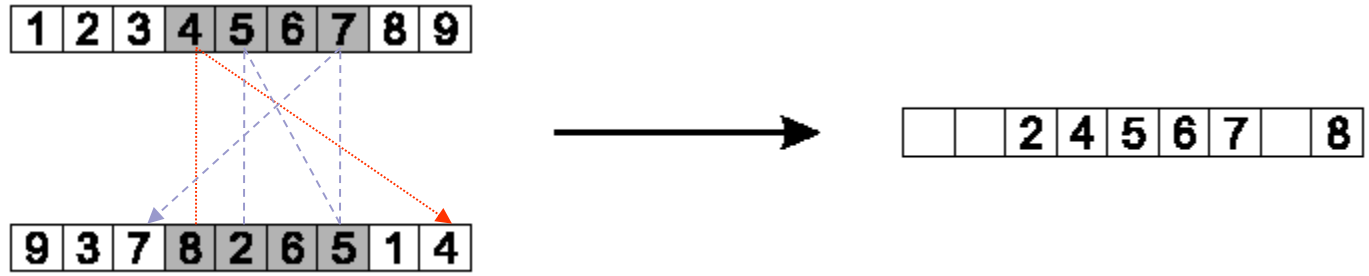
Step 1

copy segment over



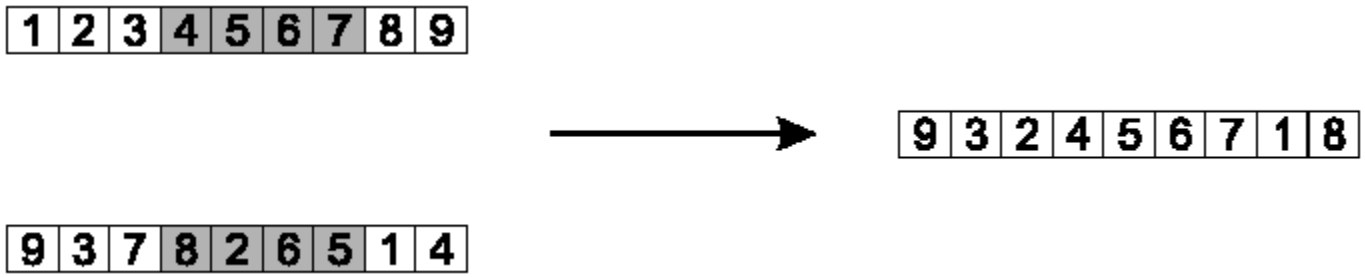
Step 2

elements in the segment



Step 3

copy remaining elements over



Note: {5-6} and {7-8} are in the parents, but only {5-6} is in the child.

Cycle crossover

Basic idea:

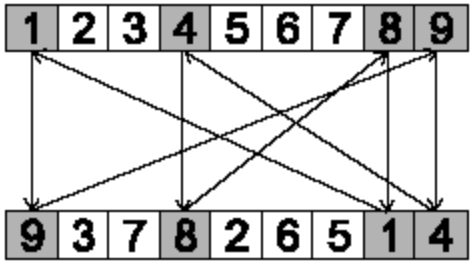
Each allele comes from one parent *together with its position*.

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

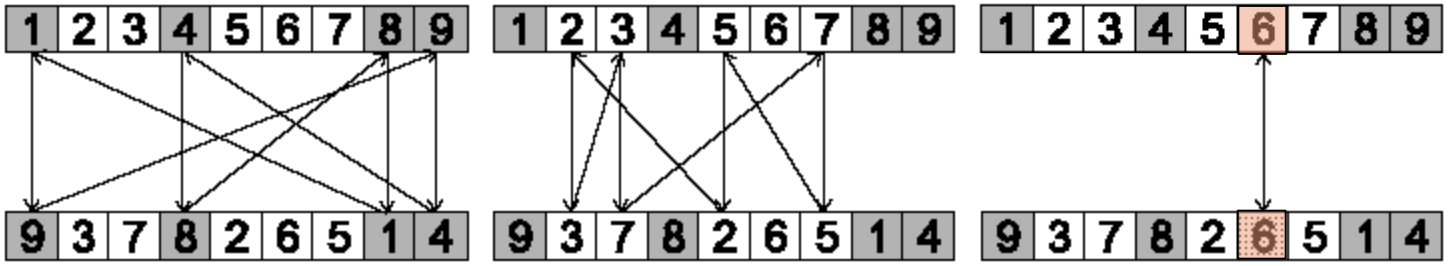
Step 1: identify cycles



(use this operator when positions matter)

Cycle crossover example

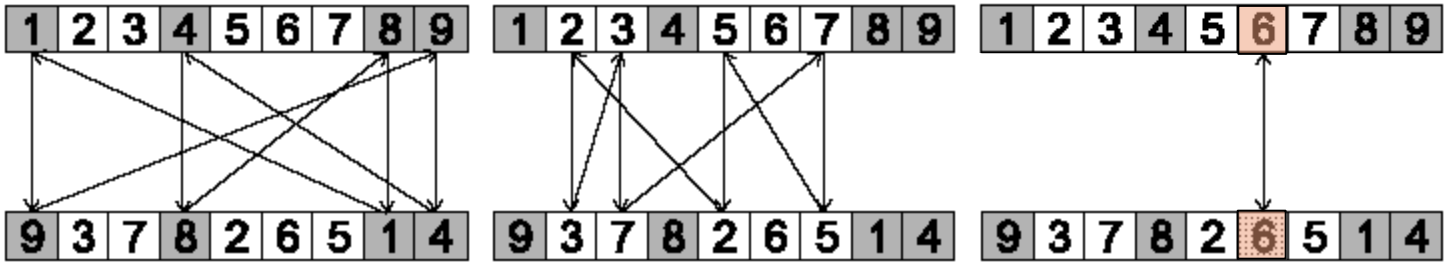
Step 1: identify cycles



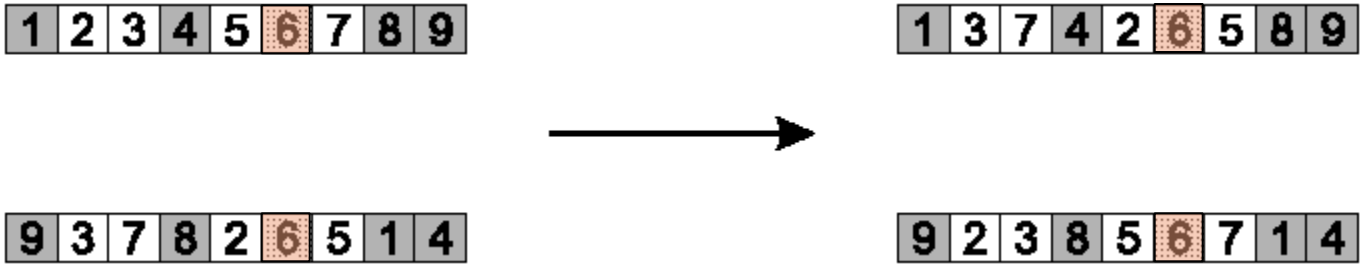
(use this operator when positions matter)

Cycle crossover example

Step 1: identify cycles



Step 2: copy alternate cycles into offspring



(use this operator when positions matter)

Edge Recombination

Works by constructing a table listing which edges are present in the two parents, if an edge is common to both, mark with a +

e.g. [1 2 3 4 5 6 7 8 9] and [9 3 7 8 2 6 5 1 4]

Element	Edges	Element	Edges
1	2,5,4,9	6	2,5+,7
2	1,3,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+, 9
4	1,3,5,9	9	1,3,4,8
5	1,4,6+		

Edge Recombination

Works by constructing a table listing which edges are present in the two parents, if an edge is common to both, mark with a +

e.g. [1 2 3 4 5 6 7 8 9] and [9 3 7 8 2 6 5 1 4]

Element	Edges	Element	Edges
1	2,5,4,9	6	2,5+,7
2	1,3,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+, 9
4	1,3,5,9	9	1,3,4,8
5	1,4,6+		

Edge Recombination 2

Informal procedure once edge table is constructed

1. Pick an initial element at random and put it in the offspring
2. Set the variable current element = entry
3. Remove all references to current element from the table
4. Examine list for current element:
   - If there is a common edge, pick that to be next element
   - Otherwise pick the entry in the list which itself has the shortest list
   - Ties are split at random
5. In the case of reaching an empty list:
   - Examine the other end of the offspring is for extension
   - Otherwise a new element is chosen at random

Edge Recombination example

[1 2 3 4 5 6 7 8 9] and
[9 3 7 8 2 6 5 1 4]

Element	Edges	Element	Edges
1	2,5,4,9	6	2,5+,7
2	1,3,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+, 9
4	1,3,5,9	9	1,3,4,8
5	1,4,6+		

Choices	Element selected	Reason	Partial result
All	1	Random	[*1*]
2,5,4,9	5	Shortest list	[*1 5*]
4,6	6	Common edge	[*1 5 6*]
2,7	2	Random choice (both have two items in list)	[*1 5 6 2*]
3,8	8	Shortest list	[*1 5 6 2 8*]
7,9	7	Common edge	[*1 5 6 2 8 7*]
3	3	Only item in list	[*1 5 6 2 8 7 3*]
4,9	9	Random choice	[*1 5 6 2 8 7 3 9*]
4	4	Last element	[*1 5 6 2 8 7 3 9 4*]

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 some usefulness
- 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., centre of mass crossover, generalising arithmetic recombination operators

Population Models

- SGA uses so-called 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 (SSGA):
  - one offspring is generated per generation,
  - one member of population replaced,
- Generation Gap
  - the proportion of the population replaced
  - 1.0 for SGA, 1/pop_size for SSGA

Note: many things are possible in EAs and many things will work more or less well.
Experience is key, which is why there is a programming assignment.

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 individuals
  - i.e., they are representation-independent

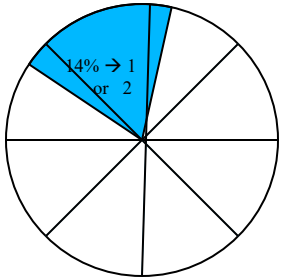
Implementation example: SGA

- 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⟩ sum of all fitnesses in pop.)

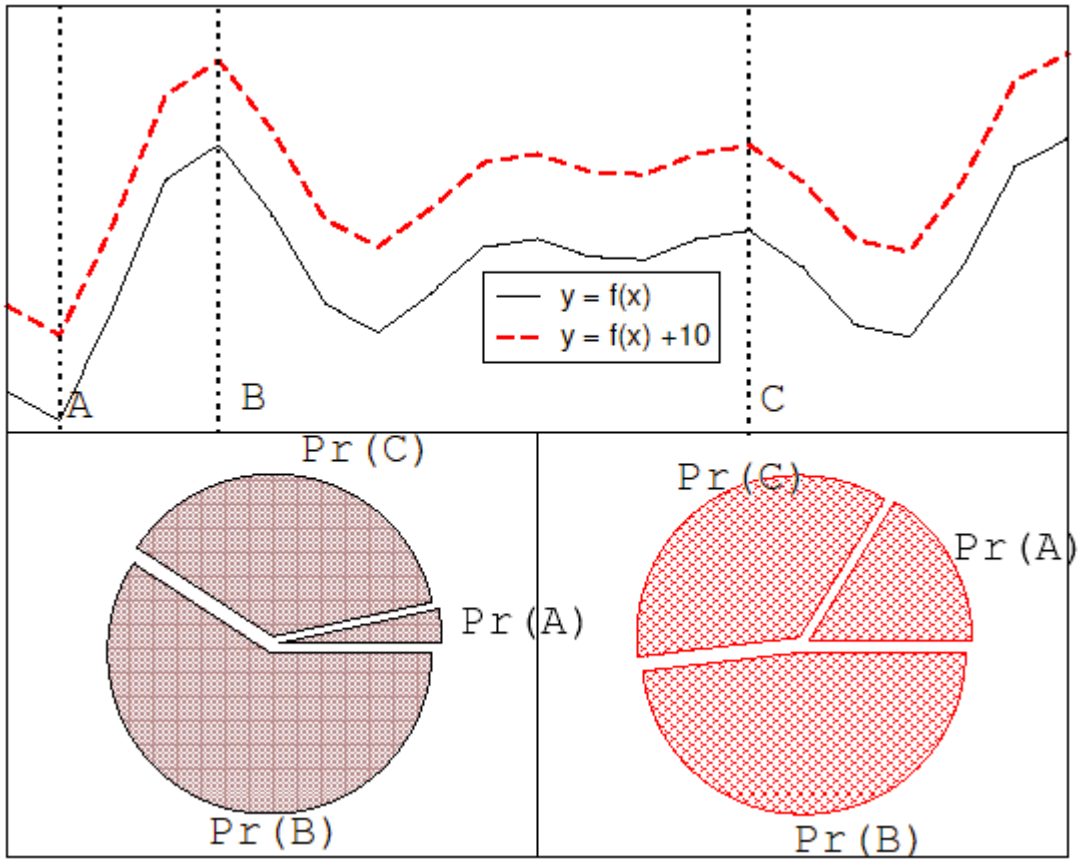
- Roulette wheel algorithm:
  - Given a probability distribution, spin a 1-armed wheel *n* times to make *n* selections
  - No guarantees on actual value of n_i
- Baker's stochastic uniform sampling (SUS) algorithm:
  - *n* evenly spaced arms on wheel and spin once
  - Guarantees $floor(E(n_i)) \leq n_i \leq ceil(E(n_i))$



Fitness-Proportionate Selection (FPS)

- 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 (see next slide)
- 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), σ_f is the standard deviation, and $\langle f \rangle$ is the average fitness

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

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

k=2 tournament

Tournament Selection

- 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*

*** SUMMARY ***

What we have seen today:

- The so-called Simple GA (which focusses on bit-strings) and its components
- Several operators for working with permutations (→ group assignment!)