Genetics Algorithm

Javad Salimi

Fall 2019

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

The simple GA

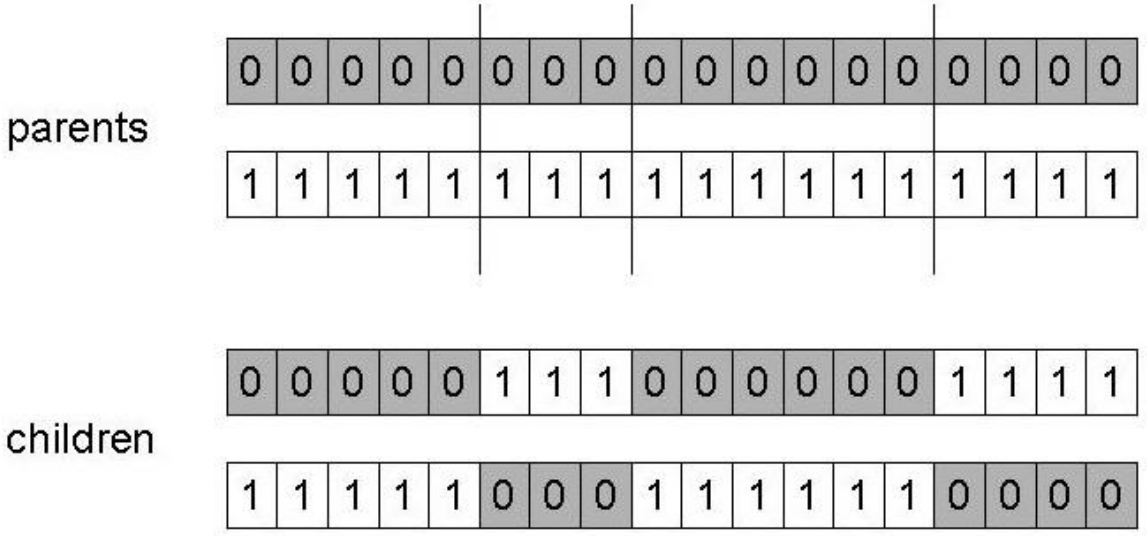
- Has been subject of many (early) studies
  - still often used as benchmark for novel GAs
- Shows many shortcomings, e.g.
  - Representation is too restrictive
  - Mutation & crossovers only applicable for bit-string & integer representations
  - Selection mechanism sensitive for converging populations with close fitness values

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

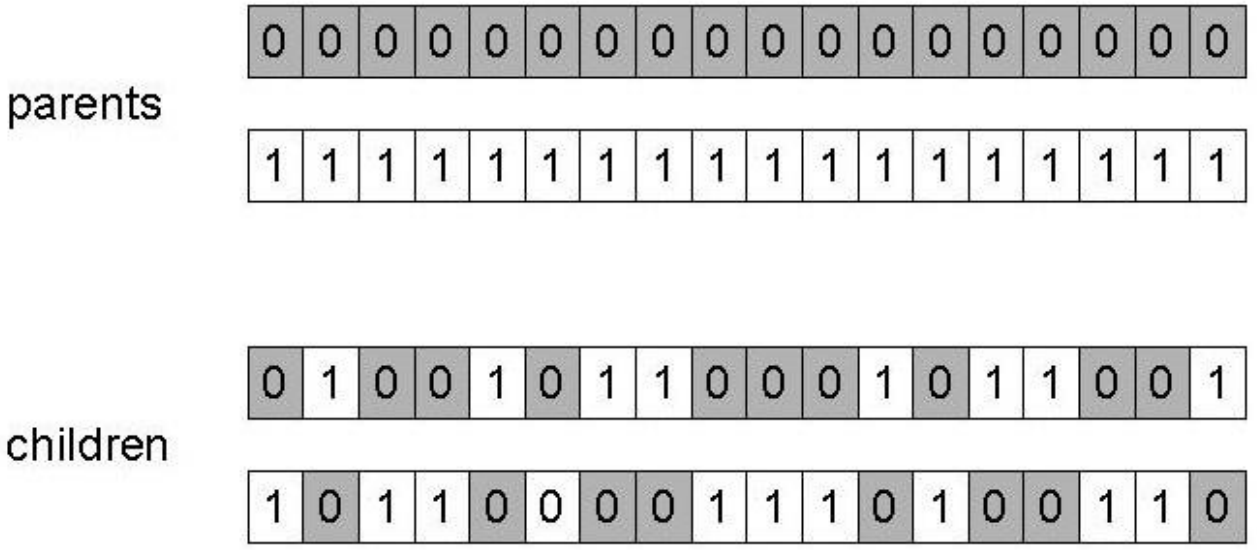
n-point crossover

- 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?

- 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)

- 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

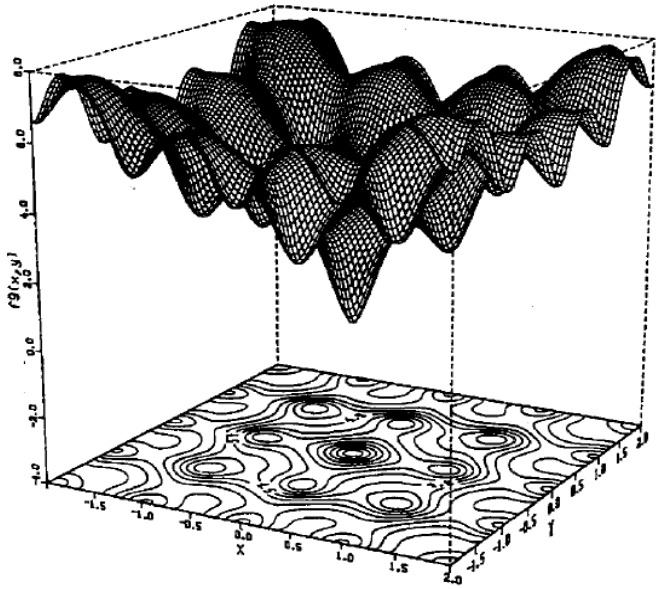
Real valued problems

- Many problems occur as real valued problems, e.g. continuous parameter optimization $f : \mathcal{R}^n \rightarrow \mathcal{R}$
- 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

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

- Uniform mutation:

x_i' drawn randomly (uniform) from $[LB_i, UB_i]$

Floating point mutations 2

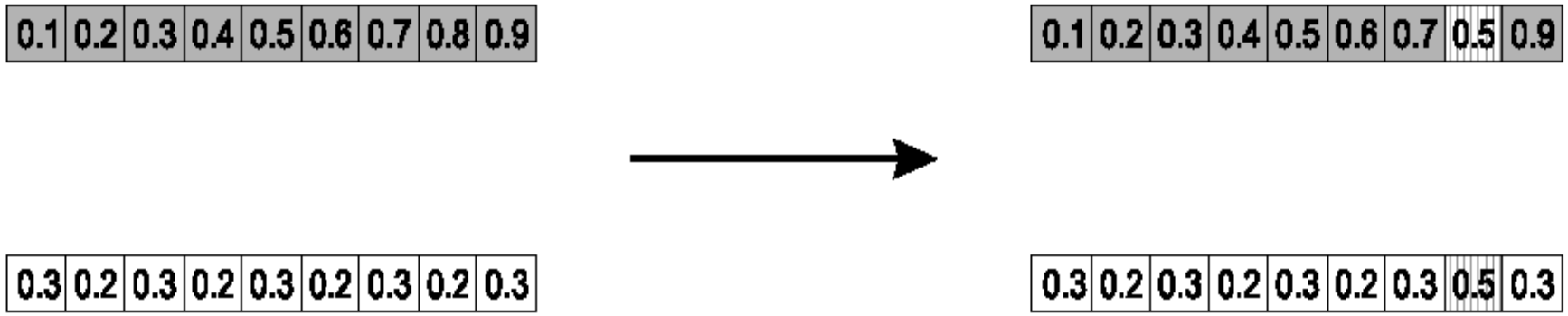
- Non-uniform mutations:
  - Many methods proposed,such as time-varying range of change etc.
  - Most schemes are probabilistic but usually only make a small change to value
  - Most common method is to add random deviate to each variable separately, taken from N(0, σ) Gaussian distribution and then 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 $\alpha : 0 \leq \alpha \leq 1$.
  - 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: $\langle x_1, ..., x_k, \alpha \cdot y_k + (1-\alpha) \cdot x_k, ..., x_n \rangle$
- reverse for other child. e.g. with $\alpha = 0.5$

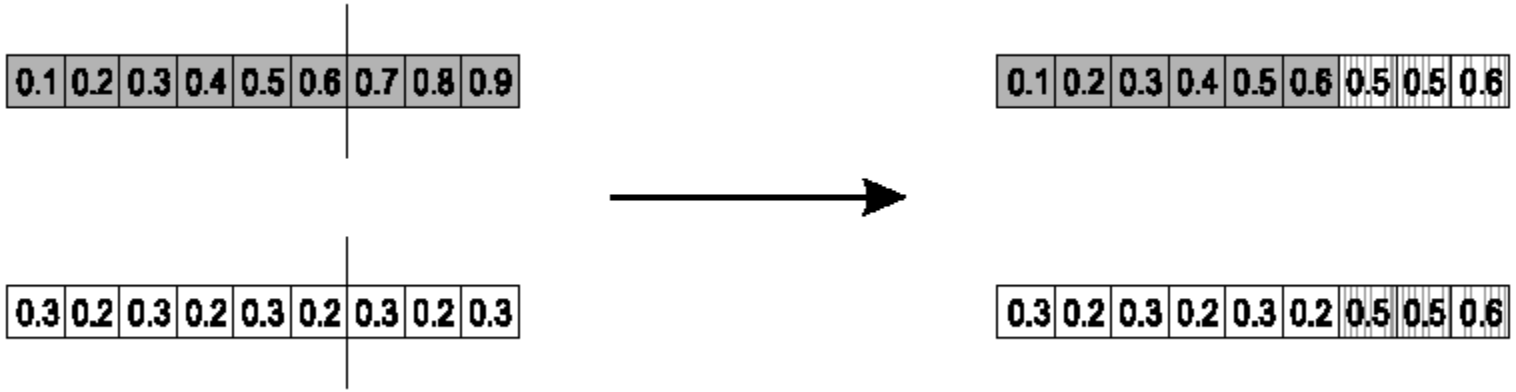


Simple arithmetic crossover

- 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:

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

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

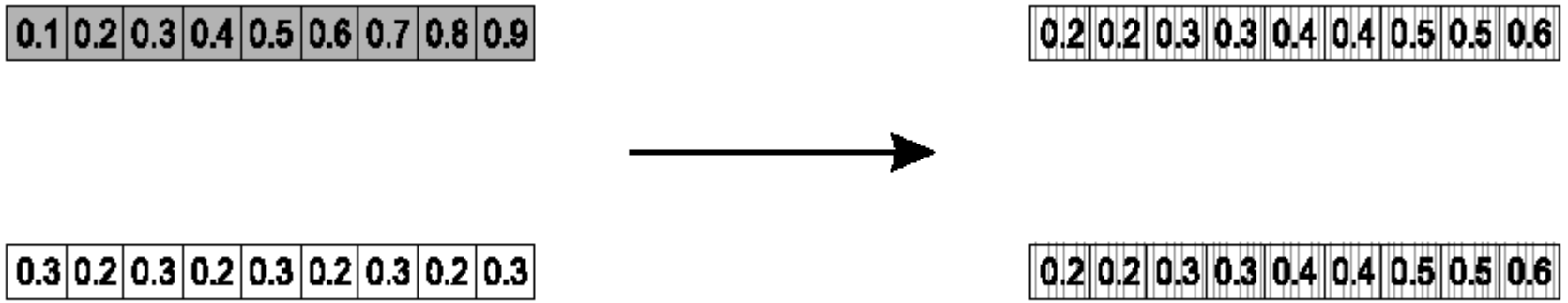


Whole arithmetic crossover

- 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 $\alpha = 0.5$



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

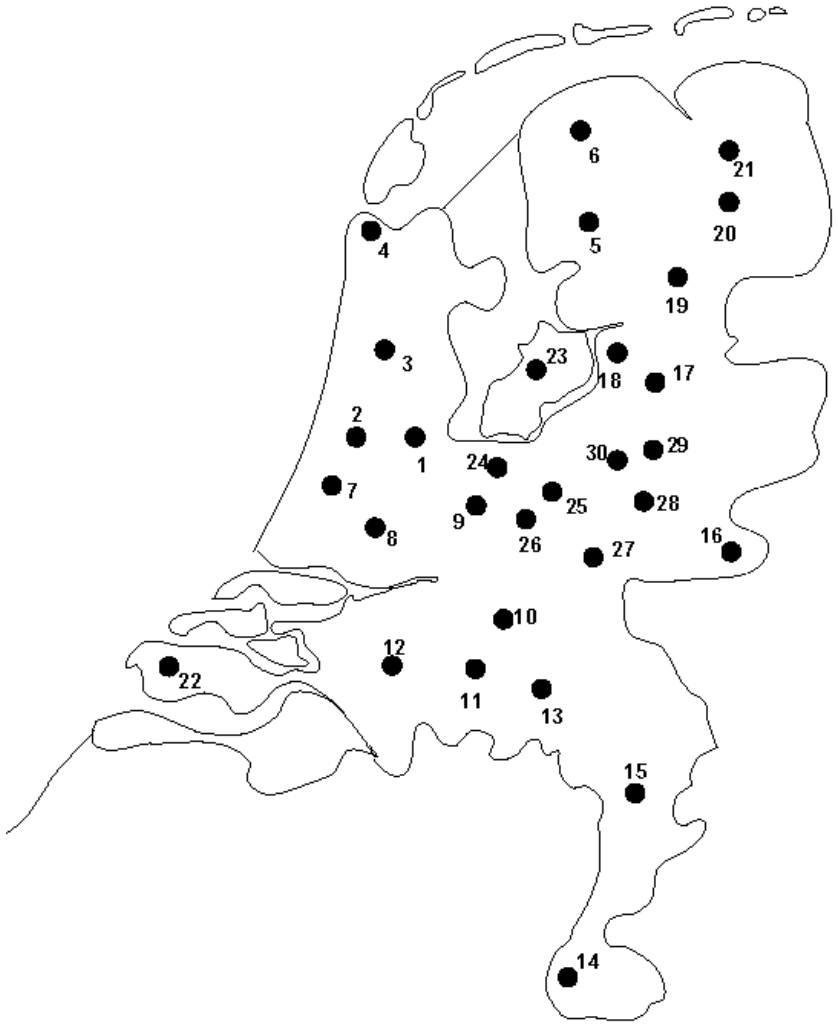
Permutation Representations

- 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

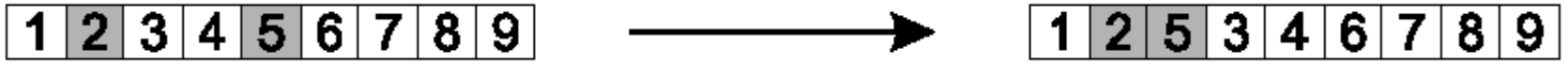


Mutation operators for permutations

- 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

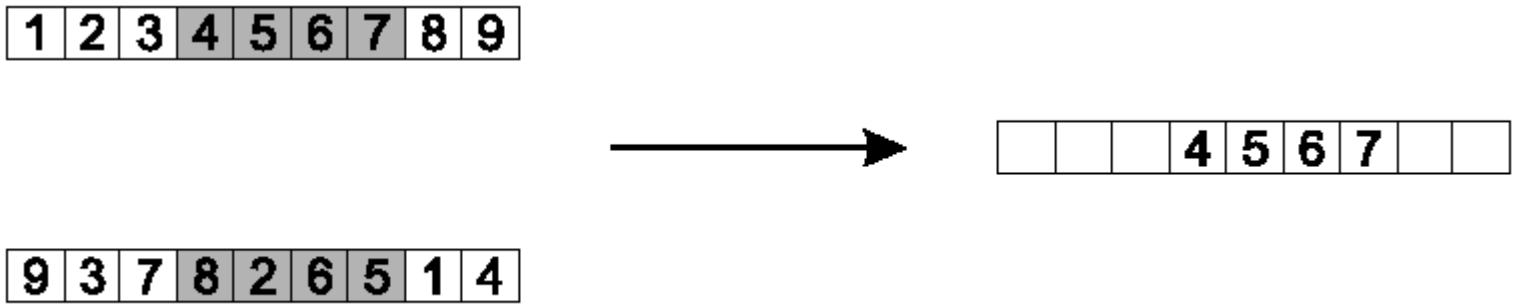
- Many specialised operators have been devised which focus on combining order or adjacency information from the two parents

Order 1 crossover

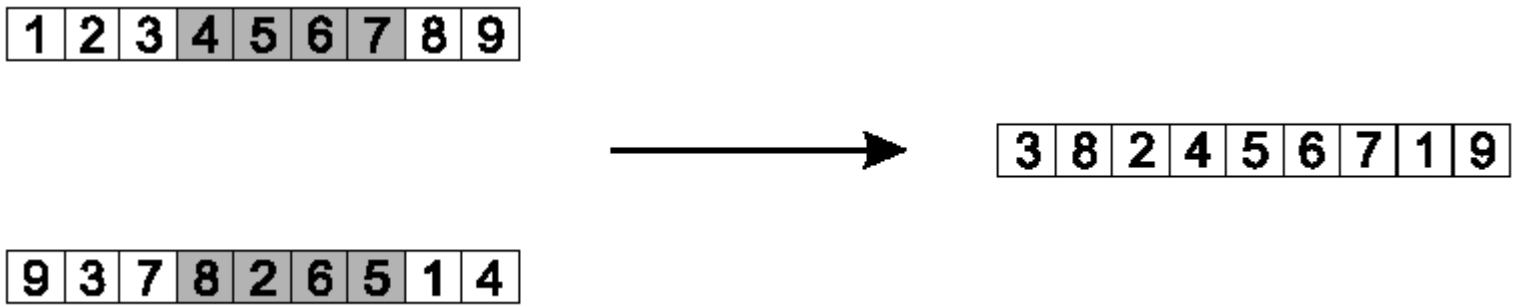
- Idea is to preserve relative order that elements occur
- Informal procedure:
  1. Choose an arbitrary part from the first parent
  2. Copy this part to the first child
  3. Copy the numbers that are not in the first part, to the first child:
     - starting right from cut point of the copied part,
     - 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)

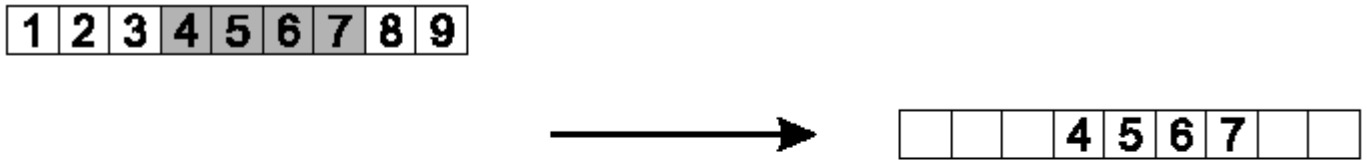
Informal procedure for parents P1 and P2:

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.

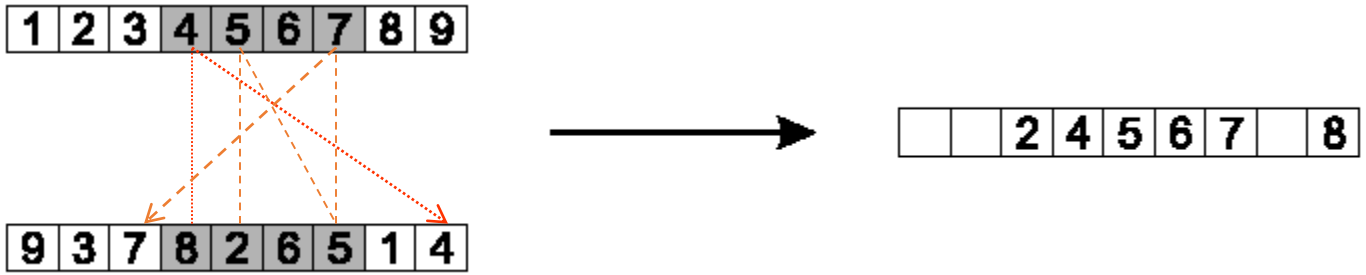
Second child is created analogously

PMX example

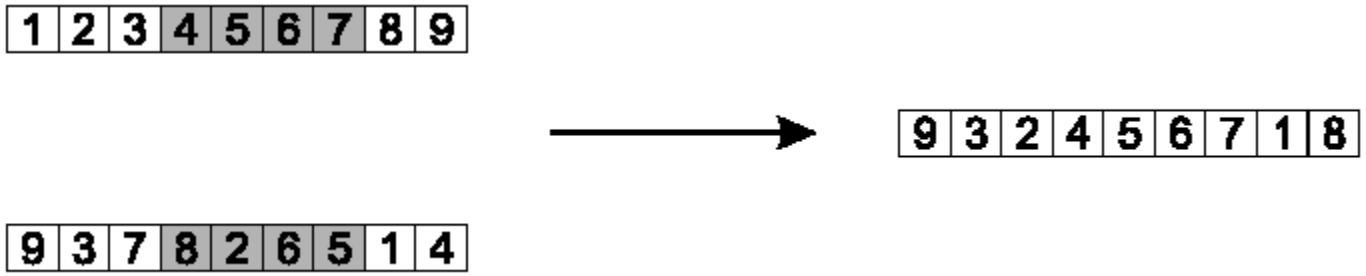
- Step 1



- Step 2



- Step 3



Cycle crossover

Basic idea:

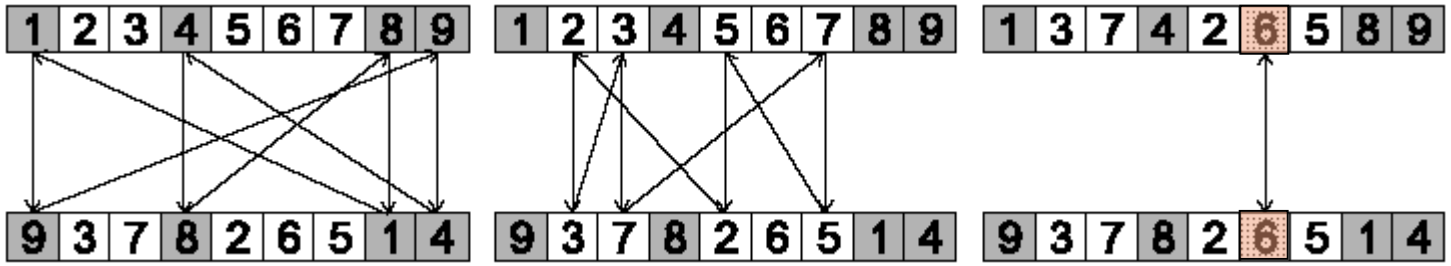
Each allele comes from one parent *together with its position*.

Informal procedure:

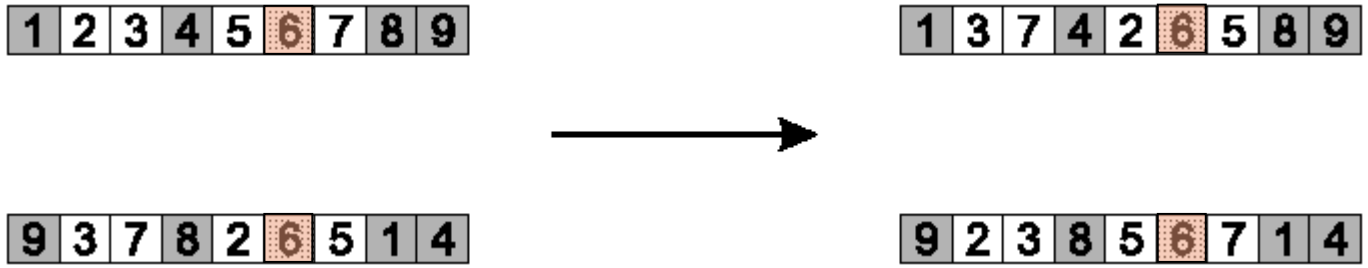
1. Make a cycle of alleles from P1 in the following way.
   (a) Start with the first allele of P1.
   (b) Look at the allele at the *same position* in P2.
   (c) Go to the position with the *same allele* in P1.
   (d) Add this allele to the cycle.
   (e) Repeat step b through d until you arrive at the 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



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

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*]

Modified Edge Recombination

- Edge Recombination - Link based.

- Modified Edge Recombination - Link and order based.

Parent 1	Common links	Offspring 1
(6 7 8 1 2 3 9 4 5 0)	**(- 7 8 1 - - - - 5 0)**	**(3 7 8 1 4 6 2 9 5 0)**

Parent 2	Common links	Offspring 2
(1 8 7 9 5 0 2 6 3 4)	**(1 8 7 - 5 0 - - - -)**	**(1 8 7 3 5 0 6 2 4 9)**

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