Genetic Algorithms

commonly used selection, replacement, and variation operators

Fernando Lobo

University of Algarve

Outline

- Selection methods
- Replacement methods
- Variation operators

Selection Methods

Components of a GA

- Initialization
  - usually at random, but can use prior knowledge.
- Selection
  - give preference to better solutions.
- Variation
  - create new solutions through crossover and mutation.
- Replacement
  - combine original population with newly created solutions.

Selection

- Various methods can be used.
- Main idea is to make copies of solutions that perform better at the expense of solutions that perform worse.
- Two major classes of methods:
  - Proportionate
  - Ordinal

Proportionate-based methods

- Probability of selecting a solution is proportional to its fitness value.
- Probability $p_{select}(X)$ of selecting X from population P(t) is

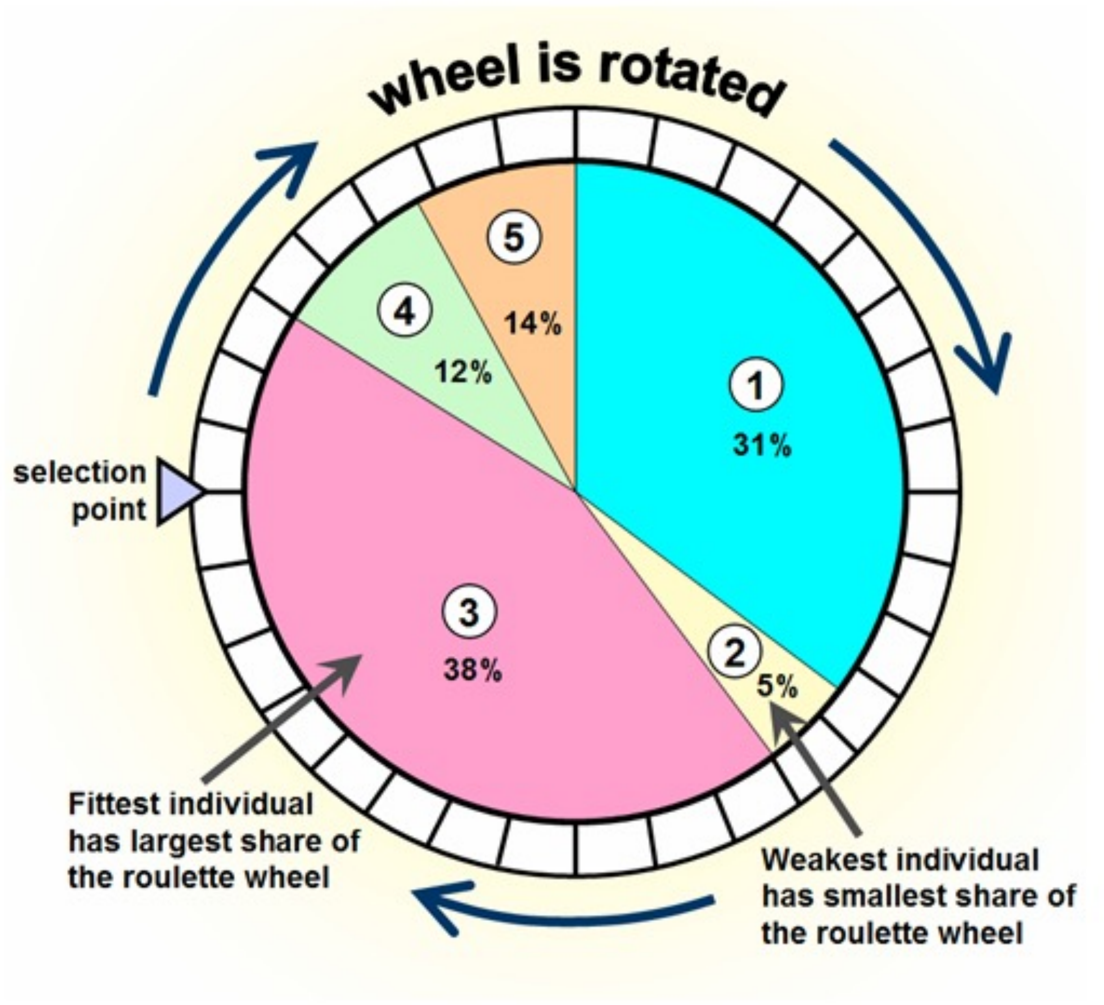
$$p_{select}(X) = \frac{f(X)}{\sum_{Y \in P(t)} f(Y)}$$

- Fitness is assumed to be positive.

Proportionate-based methods

- We will look at two methods:
  - Roulette Wheel
  - Stochastic Universal Sampling (SUS)

Roulette Wheel



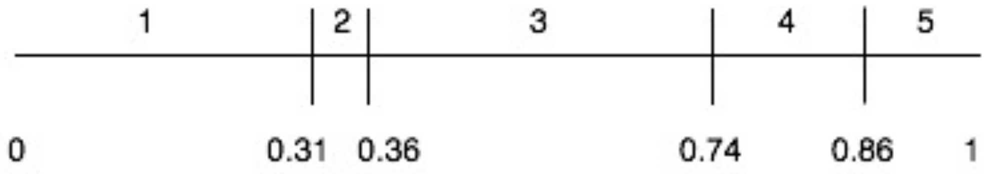
wheel is rotated
5
14%
4
12%
1
31%
selection point
3
38%
2
5%
Fittest individual has largest share of the roulette wheel
Weakest individual has smallest share of the roulette wheel

Roulette Wheel

- Fitness of an individual corresponds to a slice of the roulette.
- Implementation: Fitnesses are mapped to contiguous segments of a line.
- Each individual's segment is equal in size to its fitness.
- A uniform random number is generated and the individual whose segment spans the random number is selected.
- Spin the roulette N times in order to select N individuals.

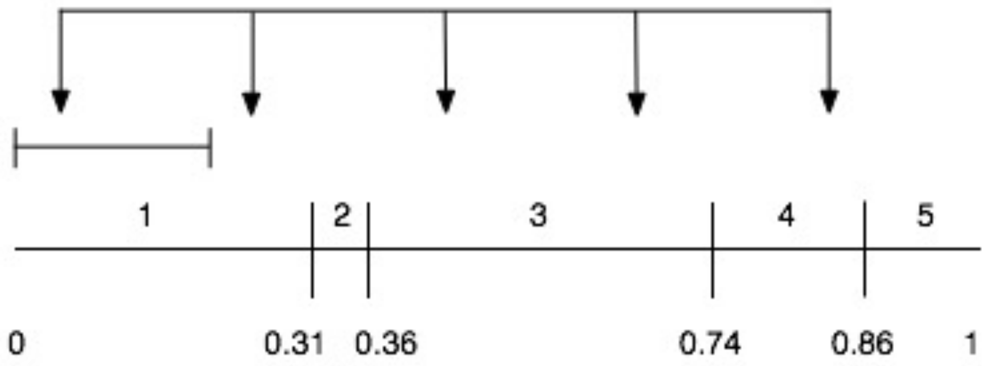
Roulette Wheel: an example

No.	chromosome	fitness	fraction of total
1	0100010001	6.82	0.31
2	1100101001	1.11	0.05
3	1100111001	8.48	0.38
4	0101011111	2.57	0.12
5	1100100100	3.08	0.14
	Totals:	22.05	1.00



Stochastic Universal Sampling (SUS)

- Make N equally spaced marks.
- Generate a uniform random number in [0,1/N]. That number is the position of the 1st mark. (equivalent to spinning the equally spaced marks on top of the roulette. Just need to spin it once.)



Roulette Wheel vs SUS

- What's the difference?
- SUS has less variance.
- SUS is computationally more efficient.

Ordinal-base methods

- Probability of selecting an individual depends on its relative order (or ranking) compared with other members.
- Two commonly used methods:
  - tournament selection
  - truncation selection

Tournament selection

- Pick *s* individuals and keep the best one.
- *s* is a parameter.
- We used *s*=2 in our example by hand.
- Two variations:
  - with replacement
  - without replacement

With and without replacement

- Imagine that we are picking individuals from a bag.
- With replacement, after picking s individuals, we put them back in the bag.
- Without replacement, we do not put them back in the bag.
- What’s the difference?
  - without replacement has less variance (just like SUS has less variance compared to the roulette wheel).

Truncation selection

- Selects a given percentage τ of the best individuals in the population.
- τ is a parameter.
- Ex: $\tau=0.25$ means selecting the top 25% individuals.
- Make multiple copies of individuals, as needed.
  - with $\tau=0.25$ we would make 4 copies of each, in order to select N individuals.

Proportionate vs ordinal-based methods

- Fitness proportionate-based methods have 2 major drawbacks:
  - super individual can take over the population too quickly.
  - search tends to stall when all the individuals have about the same fitness.

- Ordinal-based methods are often preferred.
  - Enable a sustained pressure towards better solutions.
  - Can control selection pressure easily.

Replacement Methods

Components of a GA

- Initialization
  - usually at random, but can use prior knowledge.
- Selection
  - give preference to better solutions.
- Variation
  - create new solutions through crossover and mutation.
- Replacement
  - combine original population with newly created solutions.

Replacement

- Newly created solutions replace some (or all) of the current solutions.
- Delete-all or full replacement
  - all previous solutions are deleted.
- Steady-state replacement
  - Some of the old solutions become part of the new population.

Steady-state replacement

- Different strategies can be used to choose solutions to be replaced
  - choose randomly
  - choose the worst
- Whenever there is the guarantee that the best solution never dies, GAs are said to be *elitist*.
- Percentage of replaced individuals is called the *generation gap*.

Variation Operators

Components of a GA

- Initialization
  - usually at random, but can use prior knowledge.
- Selection
  - give preference to better solutions.
- Variation
  - create new solutions through crossover and mutation.
- Replacement
  - combine original population with newly created solutions.

Variation operators in GAs

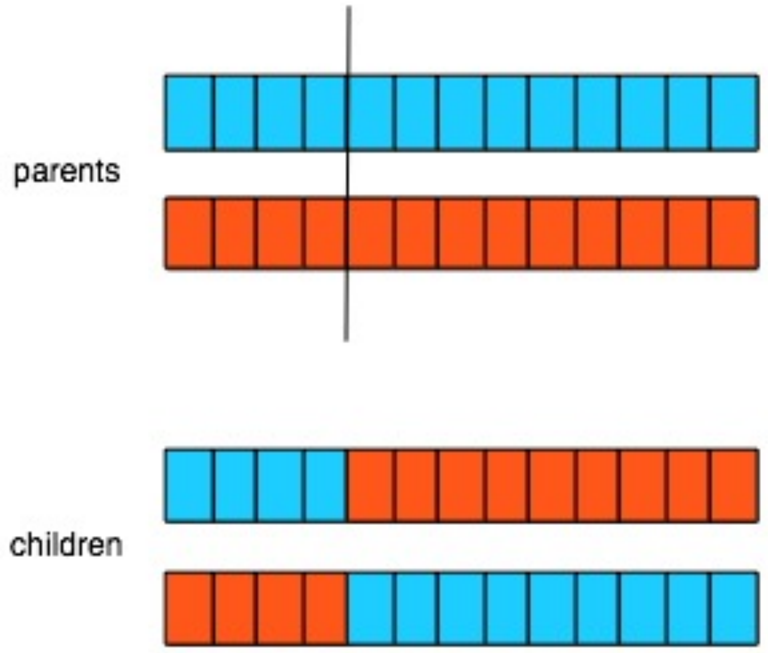
- Let's look at some of the commonly used variation operators for GAs.
  - for binary strings.
  - for real-valued vectors and permutations.

Commonly used crossover operators for binary strings.

- One-point, two-point, k-point crossover
- Uniform crossover

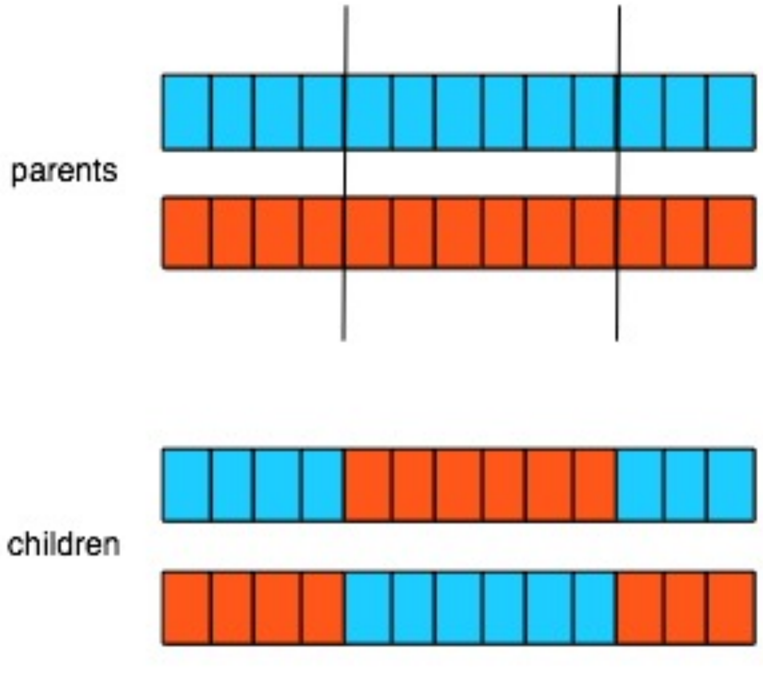
One-point crossover

- A crossing position is chosen at random and the two parents exchange all their bits after that position (like we did in our example by hand).



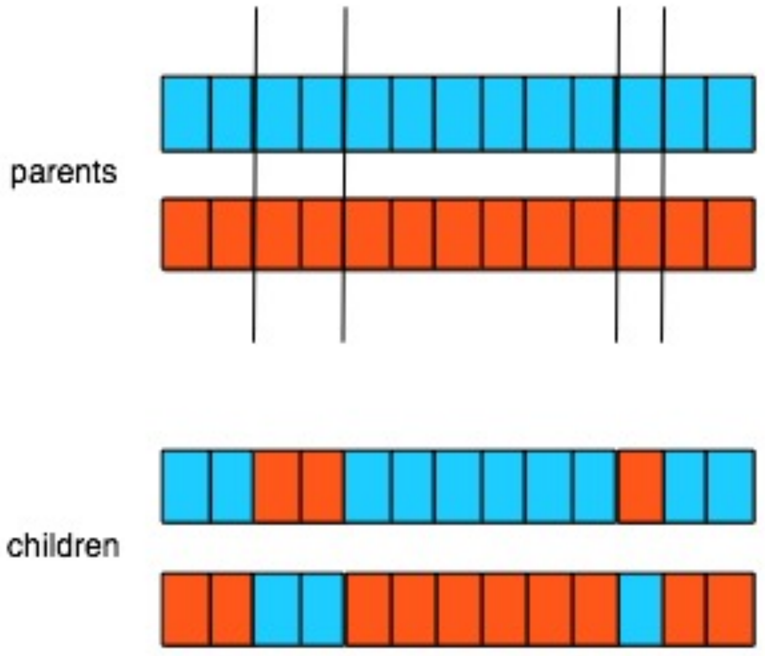
Two-point crossover

- Two crossing positions are chosen at random and the two parents exchange all their bits between the two locations.



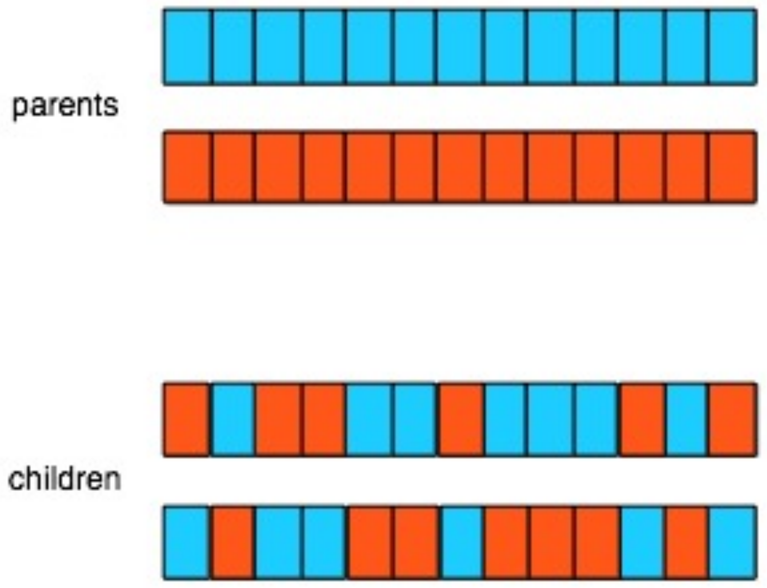
k-point crossover

- Can generalize method to k crossing positions. Here is an example with k=4.



Uniform crossover

- For each position, exchange alleles with a given probability (typically 0.5). Here’s an example with ‘1011001000101’ (1=exchange, 0=don’t exchange)



k-point vs uniform crossover

- 1-point, 2-point, …, k-point crossover establish an implicit linkage among genes (string positions).
  - Positions close to each other are likely to be treated together (either exchanged or not).

- Uniform crossover treats every position independently.
  - It is independent of the ordering of the genes.

Other representations

- Problem solutions are not always represented as binary strings.

- Can also be represented over strings of variables, each defined over a finite alphabet, vectors of real values, permutations, sets, among others.

Other representations: two approaches

1. Map original representation to binary strings.
   - May be far from trivial.

2. Modify variation operators.
   - Manipulate the new representation directly.
   - (Nowadays, this is the approach followed most often.)

Map original representation to binary strings

- May be difficult to design a one-to-one mapping.
  - Not all admissible solutions considered.

- May be far from trivial in some cases.

- But may introduce new opportunities to discover problem regularities (more on this later in the course).

Map original representation to binary strings. Example: Real → Binary

- Solutions to many real-world problems can be encoded as vectors of real-valued numbers.

- Many ways of mapping real numbers to binary strings of fixed length.
  - what if mapping is not one-to-one?
  - what about accuracy?

Map original representation to binary strings. Example: Real → Binary

- Assume real-valued variable can obtain values in $[a,b]$ with $a < b$.
- We would like to encode a vector of such variables in binary.
- We can use k bits to represent each variable.
- A vector of n real variables would require $n*k$ bits.

Map original representation to binary strings. Example: Real → Binary

- The *k* bits of each variable X_i are interpreted as the binary representation of an integer k_i, from *0* to 2^k-1.

- The value of X_i would then be given by

$$X_i = a + (b-a)\frac{k_i}{2^k-1}$$

- Precision is $(b - a) / (2^k - 1)$

Map original representation to binary strings. Example: Real → Binary

- Example: *X* in [2.0,5.0] represented with *k*=10 bits.
- 0000000000 maps to 2.0, 1111111111 maps to 5.0
- Precision = (5.0 - 2.0) / 1023 ≈ 0.00293
- 0001011011 maps to 2.0 + (5.0 - 2.0) 91 / 1023 ≈ 2.26686

Map original representation to binary strings. Example: Real → Binary

- The more bits to represent a real value, the better the accuracy of the representation.
- But with more bits the GA takes longer.
- Appropriate value of *k* depends on the desired accuracy of the solution, and on the ruggedness of the fitness landscape.

Map original representation to binary strings. Example: Real → Binary

- Assuming equidistance values can be very inefficient.
- After a few generations of the GA, most variables will have their values concentrated in one or several smaller regions of [a,b].
  - Can use an adaptive discretization
  - Idea: use more bits to subintervals that appear in the population most frequently.

Map original representation to binary strings. Example: Real → Binary

- Problems:
  - Hamming cliffs.
    - small difference in decoded value may need a big difference in the encoded value (ex: 7 -> 8, 0111 -> 1000)
    - cause difficulties to a gradual search in a continuous space.
    - Gray coding can attenuate that.
  - Difficulty to achieve any arbitrary precision.
  - Variable bounds may not be known a priori.

Modifying variation operators

- It is often better to work directly with the original representation.
- We must modify the standard crossover and mutation operators.

Some extensions are trivial

- Solutions represented by strings over a finite (non-binary) alphabet.
- Standard crossover operators don’t need to be modified
- Mutation has to be changed
  - change the variable value to any other admissible value (makes sense for nominal values).
  - change the value to another value in such a way that small changes have a higher probability than large changes (makes sense for ordinal values).

Variation operators for real-valued vectors

- Let's look at 1 mutation and 3 popular crossover operators for real-valued vectors.

- Mutation
  - Gaussian mutation

- Crossover
  - Discrete crossover
  - Arithmetic crossover
  - Simulated binary crossover (SBX)

Gaussian mutation

- Each variable is modified by adding a random number sampled from a Normal distribution with zero mean.

- The variance of the Normal distribution gives the strength of the mutation.
  - Typically significantly smaller than the range of values of the variable.

Discrete crossover

- It’s just like the standard crossover operators for binary strings.

- Pick two individuals and exchange some of their variable values.

Arithmetic crossover

- Select a location *i* at random and generate a random number α in [0,1]. Let X_i and Y_i be the variables in the two parents at position *i*. These values are mapped to X'_i and Y'_i as follows

$$X'_i = (1-\alpha)X_i + \alpha Y_i,$$

$$Y'_i = (1-\alpha)Y_i + \alpha X_i.$$

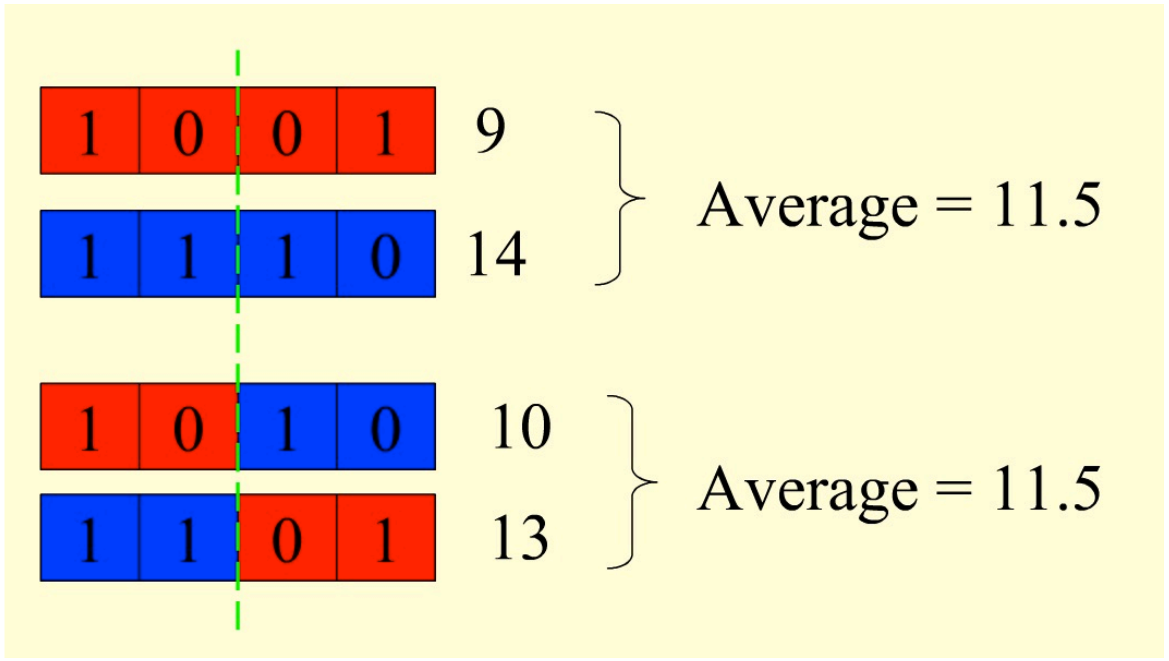
- Can be applied to several variables.

Simulated binary crossover (SBX)

- Proposed by Deb and Agrawal (1995)
- Idea is to manipulate real variables as if they were encoded in binary strings.
  - but without the problem of Hamming cliffs, fixed precision, and bounded variables.
- Parts of the next slides taken from:
  - http://www.slideshare.net/paskorn/simulated-binary-crossover-presentation

SBX (cont.)
An important property of crossover

- The average of the decoded parameter values is the same before and after the crossover operation.



SBX (cont.)

- If the decoded parameter values become closer to each other (as in the previous example), the crossover is said to be a *contracting* crossover.

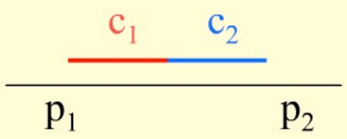
- If they move away from each other, the crossover is said to be an *expanding* crossover.

SBX (cont.)
Spread factor β

- Spread factor β is defined as the ratio of the distance between the children to that of the parents.

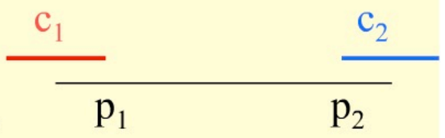
$$\beta = \left| \frac{c_1 - c_2}{p_1 - p_2} \right|$$

- **Contracting Crossover** $\beta < 1$



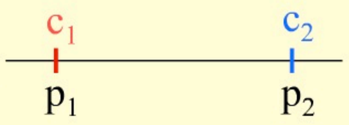
The offspring points are enclosed by the parent points.

- **Expanding Crossover** $\beta > 1$



The offspring points enclose the parent points.

- **Stationary Crossover** $\beta = 1$



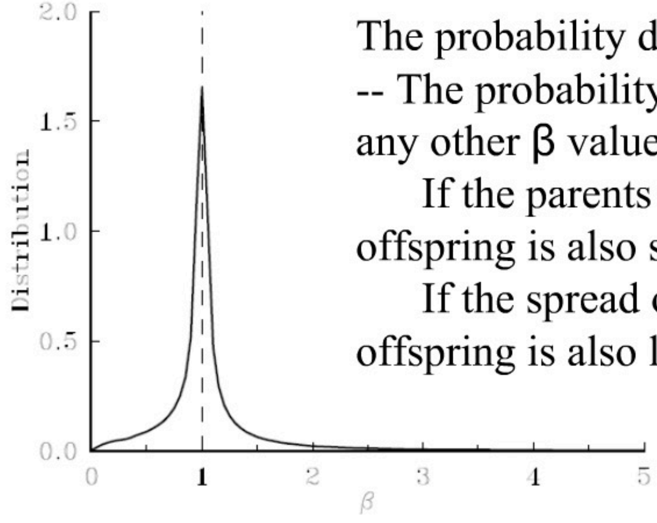
The offspring points are the same as parent points

SBX (cont.)

- Assume parents strings are two strings:

  p1= $(a_1a_2a_3\ldots a_{15})$

  p2 = $(b_1b_2b_3\ldots b_{15})$.
- The crossover point is varied from (0,15)



The probability distribution:
-- The probability of occurrence of $\beta \approx 1$ is more likely than any other β value.

If the parents are closer, the spread of the two likely offspring is also smaller.

If the spread of parents is more, the spread of likely offspring is also large.

Figure 4: Probability distributions of contracting and expanding crossovers on all pairs of random binary strings of length 15 are shown.

SBX (cont.)

- SBX was designed to simulate the behaviour of a binary coded GA with one-point crossover.

- SBX obeys to,
  - **Average Property**: the average of the decoded parameter values is conserved.
  - **Spread Factor Property**: most crossover events correspond to a spread factor of $\beta \approx 1$. (i.e. children tend to be close to their parents).

SBX (cont.): creating the offspring

To reserve the average property, the offspring values are calculated as:

Offspring:

$$c_1 = \bar{x} - \tfrac{1}{2}\beta(p_2 - p_1)$$

$$c_2 = \bar{x} + \tfrac{1}{2}\beta(p_2 - p_1)$$

By,

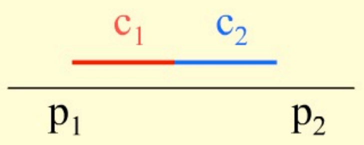
$$\bar{x} = \tfrac{1}{2}(p_1 + p_2)$$

$$p_2 > p_1$$

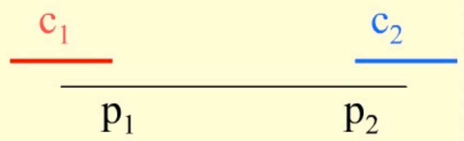
So,

$$\bar{c} = \bar{x}$$

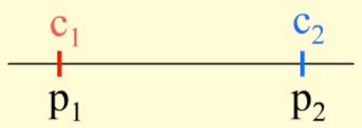
$\beta < 1$



$\beta > 1$



$\beta = 1$

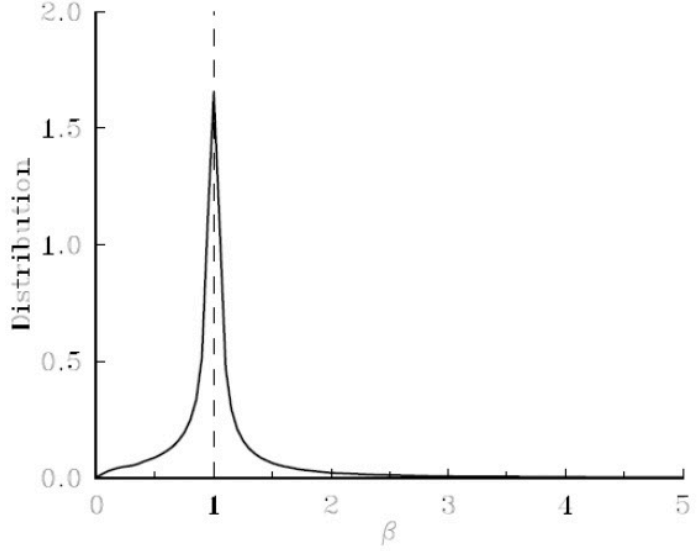


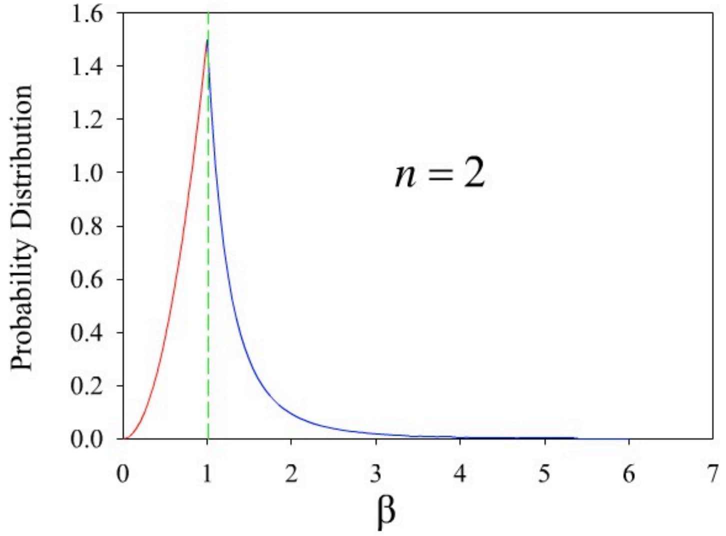
SBX (cont.): How to obtain β ?

- Use a similar probability distribution. Deb and Agrawal proposed this one:

$$c(\beta) = 0.5(n+1)\beta^n, \beta \le 1$$

$$c(\beta) = 0.5(n+1)\frac{1}{\beta^{n+2}}, \beta > 1$$





SBX (cont.): effect of n

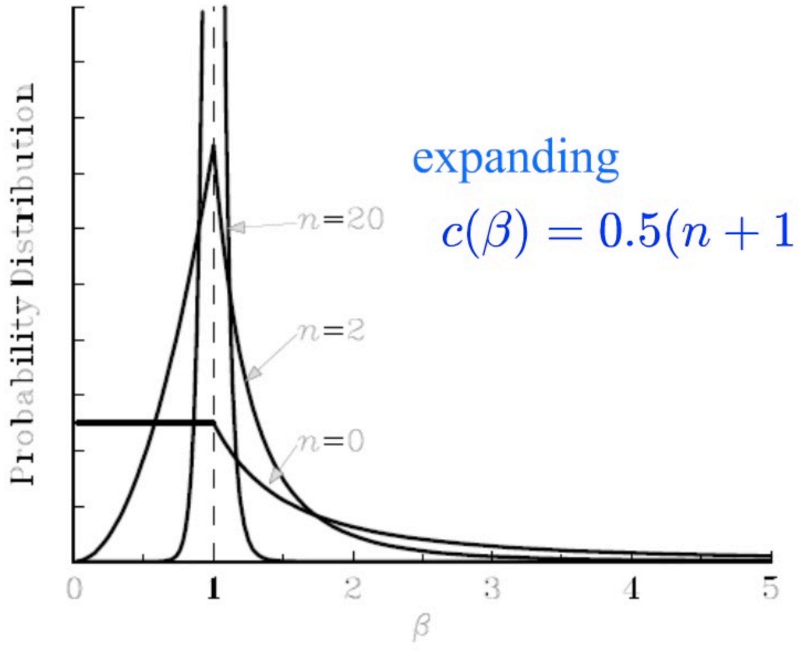
A large value of n gives a higher probability for creating ‘near-parent’ solutions.

contracting

$$c(\beta) = 0.5(n+1)\beta^n, \beta \leq 1$$

expanding

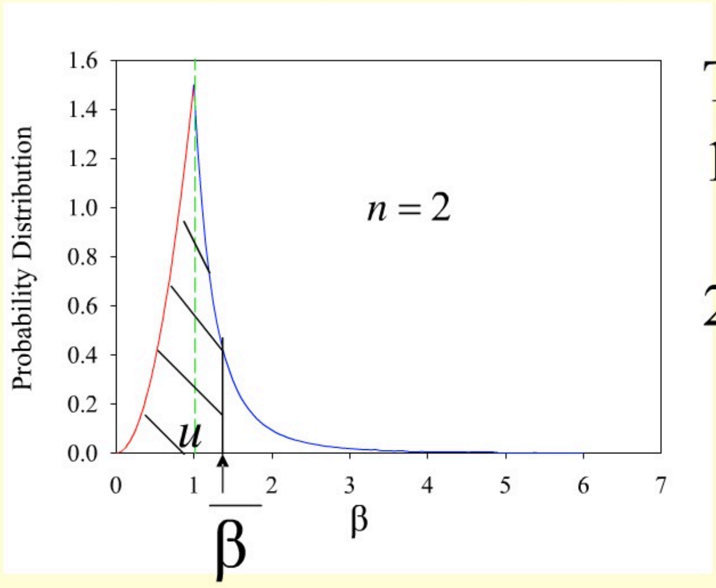
$$c(\beta) = 0.5(n+1)\frac{1}{\beta^{n+2}}, \beta > 1$$



Mostly, n=2 to 5

SBX (cont.): sampling a β value

- $\overline{\beta}$ is a random number that follows the proposed probability distribution function:



To get a $\overline{\beta}$ value,

1) A unified random number u is generated [0,1]
2) Get β value that makes the area under the curve = u

Offspring:

$$c_1 = \bar{x} - \tfrac{1}{2}\overline{\beta}(p_2 - p_1)$$

$$c_2 = \bar{x} + \tfrac{1}{2}\overline{\beta}(p_2 - p_1)$$

SBX (cont.)

- We can find the area by integrating the probability density function. Then equate it to *u* and solve for β.
- Result:

$$\beta = \begin{cases} (2u)^{\frac{1}{n+1}} & \text{if } u \le 0.5 \\ \left(\frac{1}{2-2u}\right)^{\frac{1}{n+1}} & \text{if } u > 0.5 \end{cases}$$

More about SBX

- Read the paper by Deb and Agrawal.

Modifying variation operators: permutations

- In many problems, solutions are represented as permutations. Ex: Travelling Salesman Problem (TSP).
- Standard crossover operators don’t work
  - children may not result in permutations.
- One approach is to repair the resulting solution.
- Another approach is to design an operator that only produces permutations.

Variation operators for permutations

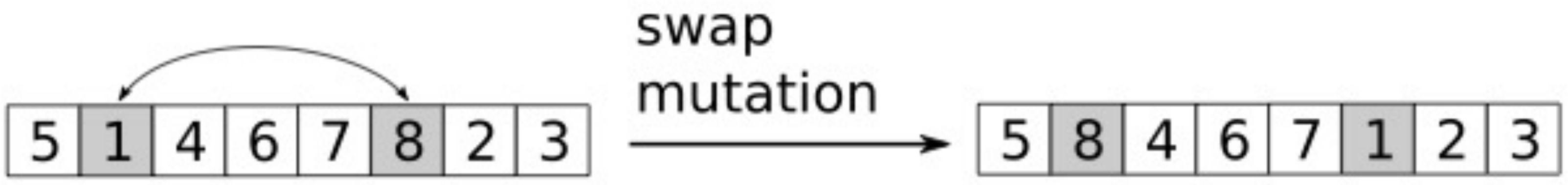
- Let’s look at some popular crossover and mutation operators for permutations.

- Mutation
  - Swap
  - Inversion
  - Scramble

- Crossover
  - Partially matched crossover (PMX)
  - Uniform order-based crossover
  - Edge recombination

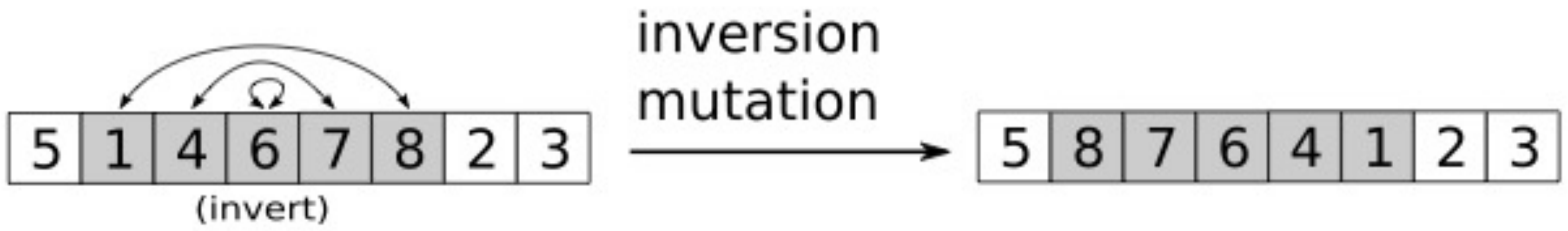
Swap mutation

- Exchange the elements in the selected locations.



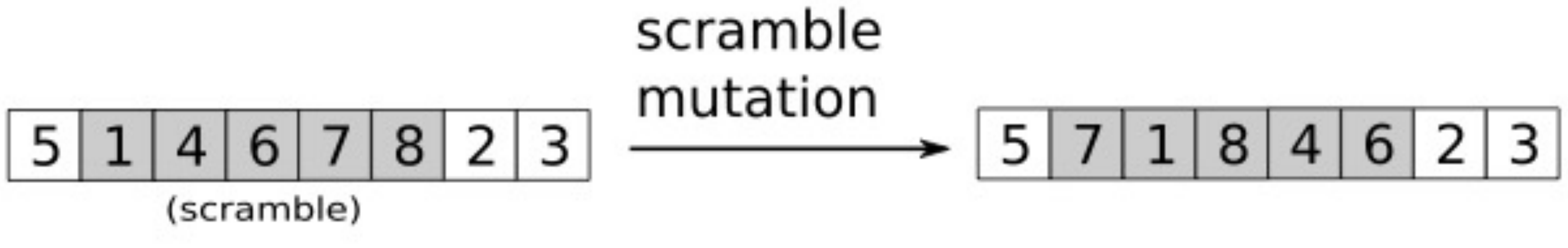
Inversion mutation

- Invert the order of the elements between the selected locations.



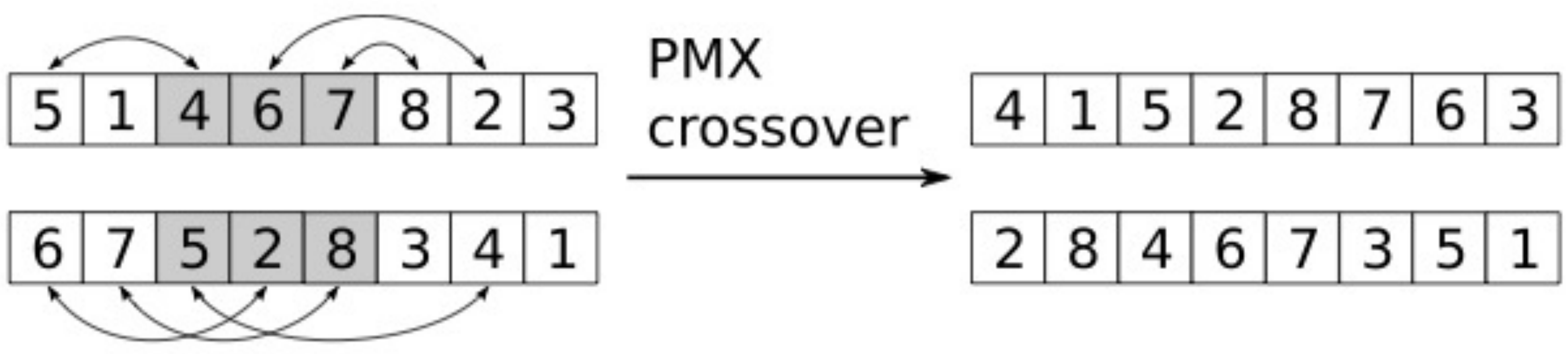
Scramble mutation

- Shuffles the elements between the selected locations.



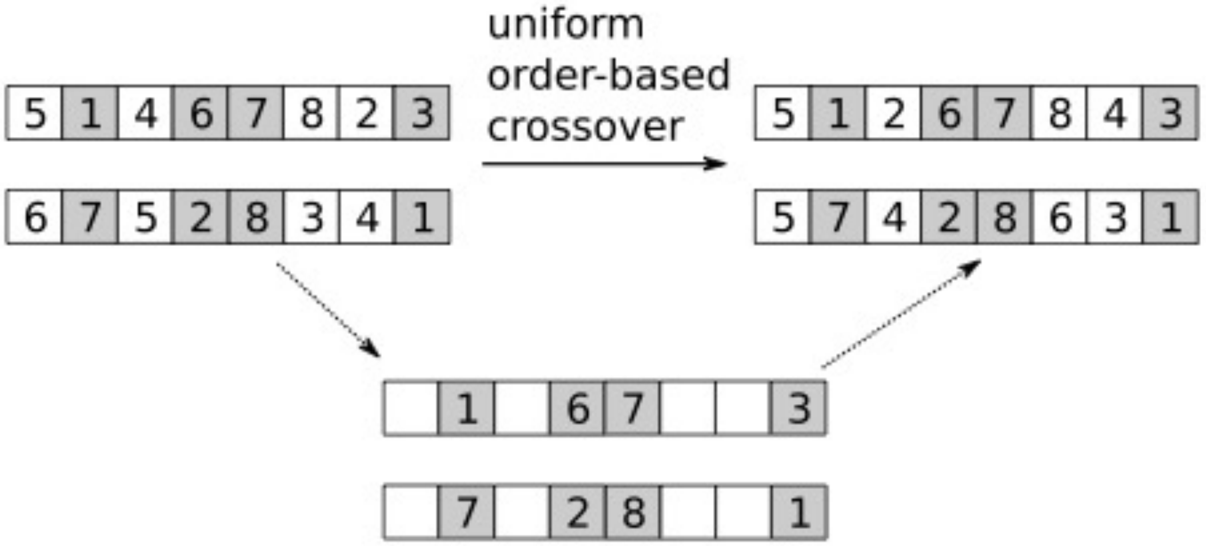
Partially matched crossover (PMX)

- Select 2 positions at random. Then, for each element between the 2 positions, the element is exchanged with the corresponding element in the other parent solution.



Uniform order-based crossover

- Randomly select positions that will not change. The remaining positions are filled with the missing elements copied in the order they appear in the other parent.



Edge recombination

- Specialized operator for the TSP.
- Motivation: preserve adjacency between permutation elements.
- Various versions have been proposed. We will look at one one of them called *edge-3*.
- What follows is an example taken from Darrell Whitley (EC1, 33.3.4, page 278).

Example

- Given 2 parents:
  - parent 1: g d m h b j f i a k e c
  - parent 2: c e k a g b h i j f m d

- An edge list is constructed for each city.
  - Edge list of city x contains all the cities in the two parents that are adjacent to x.
  - If some city is adjacent to x in both parents, we flag it with a + sign.
  - Example for city a: +k, g, i

Example (cont.): Edge table

city	Edge list
a	+k, g, i
b	+h, g, j
c	+e, d, g
d	+m, g, c
e	+k, +c
f	+j, m, i
g	a, b, c, d
h	+b, i, m
i	h, j, a, f
j	+f, i, b
k	+e, +a
m	+d, f, h

Edge recombination procedure

1. Pick a random city as the initial *current city*. Remove all references to this city from the edge table.

2. Look at the adjacency list of the current city.
   - If there is a common edge (flagged with +) go to that city.
   - Otherwise, from the cities on the current adjacency list pick the one whose own adjacency list is shorter. Ties are broken randomly.
   - Once a city is visited, remove references to that city.

Edge recombination procedure (cont.)

3. Repeat step 2 until a tour is complete or a city has been reached that has no entries in its adjacency list.
    - If not all cities have been visited, reverse the partial tour and try to expand it from the other endpoint. If that’s not possible, pick an unvisited city at random to start a new partial tour.

Example

choices	current city	reason	Partial tour
all	a	random	a
+k,g,i	k	common edge	a k
+e	e	only choice	a k e
+c	c	only choice	a k e c
d,g	d	random	a k e c d
+m,g	m	common edge	a k e c d m
f,h	h	random	a k e c d m h
+b,i	b	common edge	a k e c d m h b
g,h	g	shorter list	a k e c d m h b g
	a	reverse partial tour	g b h m d c e k a
i	i	only choice	g b h m d c e k a i
j,f	j	random	g b h m d c e k a i j
f	f	only choice	g b h m d c e k a i j f