Last class

- Evolutionary algorithms: Origins
- Evolutionary algorithms: Components
- Evolutionary algorithms: Applications

Heuristic Search and Evolutionary Algorithms

Lecture 6: Evolutionary Algorithms – Representation, Mutation and Recombination

Chao Qian （钱超）

Associate Professor, Nanjing University, China

Email: qianc@nju.edu.cn
Homepage: http://www.lamda.nju.edu.cn/qianc/

Representation and variation operators

- The first stage of applying evolutionary algorithms is to decide a right representation for the problem
  - Binary representation
  - Integer representation
  - Real-valued representation
  - Permutation representation
  - Tree representation

- Variation operators depend on the chosen representation
  - Mutation
  - Recombination

Binary representation

- Genotype space: $\{0,1\}^n$

Knapsack problem with n items

$\Rightarrow$ $x \in \{0,1\}^n$, where $x_i = 1$ denotes that the i-th item is included

$\arg max_{x \in \{0,1,\ldots,15\}} \; x^2$

$\Rightarrow$ $x \in \{0,1\}^4$, and the corresponding integer is $\sum_{i=1}^{4} x_i 2^{4-i}$

Binary representation

- Genotype space: $\{0,1\}^n$

$\arg max_{x\in\{0,1,\ldots,15\}} x^2$ ⇒ $x \in \{0,1\}^4$, and the corresponding integer is $\sum_{i=1}^{4} x_i 2^{4-i}$

Standard binary coding

7 is represented by 0111
8 is represented by 1000
9 is represented by 1001

The Hamming distance between consecutive integers can be very large

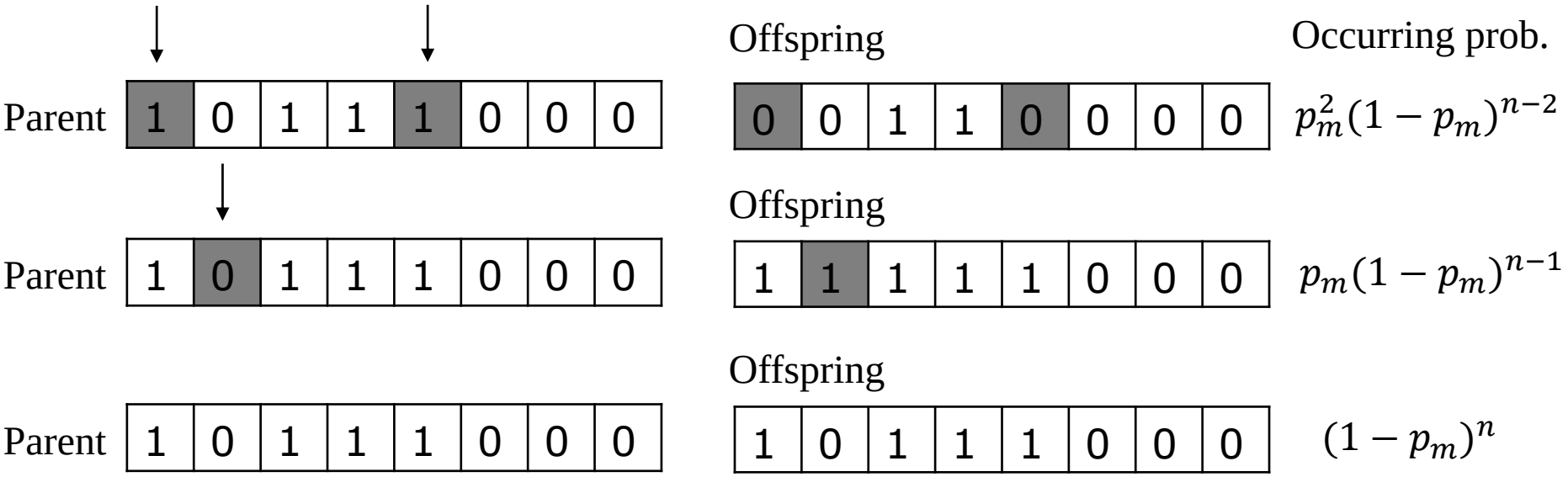
Gray coding

0	0000	4	0110	8	1100	12	1010
1	0001	5	0111	9	1101	13	1011
2	0011	6	0101	10	1111	14	1001
3	0010	7	0100	11	1110	15	1000

The Hamming distance between consecutive integers is always 1

Binary representation: Mutation

- Bit-wise mutation: flip each bit independently with prob. p_m



The common setting of p_m is $1/n$, where n is the length of the binary string

- One-bit mutation: flip a randomly chosen bit

For the above mutation behaviors, the occurring prob. are 0, $1/n$ and 0

Binary representation: Mutation

- Bit-wise mutation: flip each bit independently with prob. p_m

  The number of flipped bits is a random variable, denoted by X

  $$\mathrm{P}(X = k) = \binom{n}{k} p_m^k (1 - p_m)^{n-k}$$

  X satisfies the binomial distribution $B(n, p_m)$

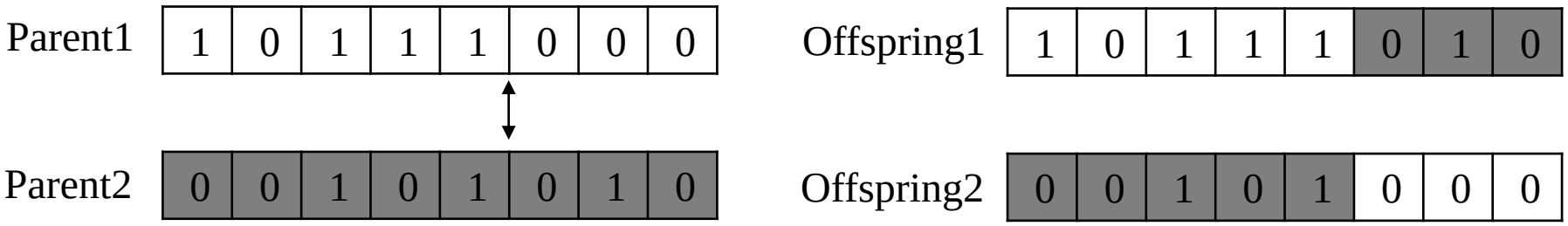
  The expected number of flipped bits is $\mathrm{E}[X] = np_m$, which is 1 for $p_m = 1/n$

- One-bit mutation: flip a randomly chosen bit

  The number of flipped bits is 1

Binary representation: Recombination

- One-point crossover
  - Choose a random number $r \in \{1,2,\ldots,n-1\}$
  - Split both parents at this point
  - Create two offspring by exchanging the tails

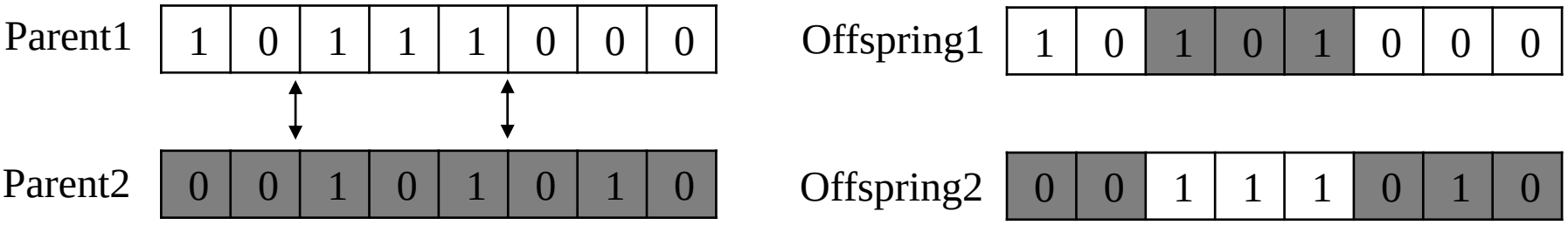


The occurring probability is $1/(n-1)$

Binary representation: Recombination

- m-point crossover
  - Choose m crossover points from $\{1,2,\ldots,n-1\}$
  - Split both parents along these points
  - Create two offspring by taking alternative segments

2-point crossover

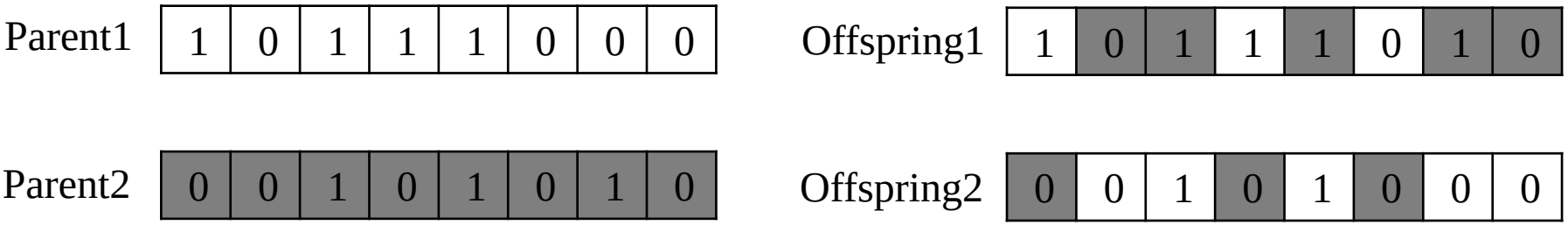


The occurring probability is $2/((n-1)(n-2))$

Binary representation: Recombination

- Uniform crossover
  - For the first offspring, each gene is inherited from the first parent with probability p independently; otherwise from the second parent
  - The second offspring is created using the inverse mapping

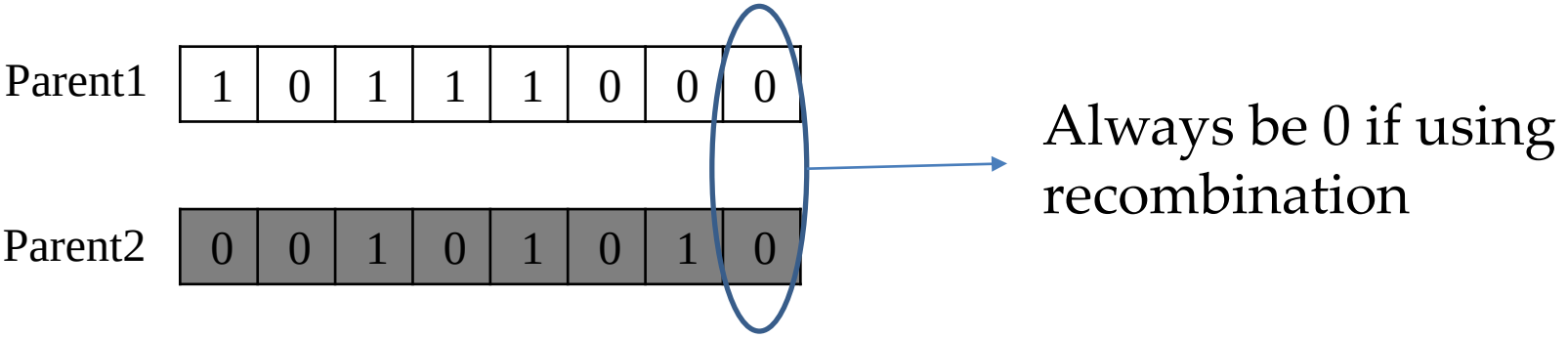
The common setting of p is $1/2$



The occurring probability is $p^3(1-p)^{n-3}$

Binary representation: Mutation and recombination

- Only mutation can introduce new information



- Only recombination can combine information from two parents
  - One-point and *m*-point crossover: more likely to keep together genes that are near each other
  - Uniform crossover: no positional bias

Integer representation

- Genotype space: N^n, where N denotes the integer

- Mutation: mutate each gene independently with probability p_m
  - Random resetting: choose a new value at random → Cardinal attributes
  - Creep mutation: add a small value, which is sampled from a distribution → Ordinal attributes

- Recombination: same as for binary representation

Real-valued representation: Mutation

- Genotype space: R^n, where R denotes the real-number
- Mutation

$$\boldsymbol{x} = (x_1, \ldots, x_n) \rightarrow \boldsymbol{x}' = (x_1', \ldots, x_n'), \text{ where } x_i, x_i' \in [lb_i, ub_i]$$

  - Uniform mutation: for each x_i, with prob. p_m, change it to a value drawn uniformly randomly from $[lb_i, ub_i]$

$$x_i' = \begin{cases} x_i & with\ prob.\ 1 - p_m \\ U(lb_i, ub_i) & otherwise \end{cases}$$

  - Nonuniform mutation: for each x_i, add a value drawn randomly from a Gaussian distribution $N(0, \sigma^2)$

$$x_i' = x_i + \delta, \text{ where } \delta \sim N(0, \sigma^2)$$

σ: mutation step size

Real-valued representation: Nonuniform mutation

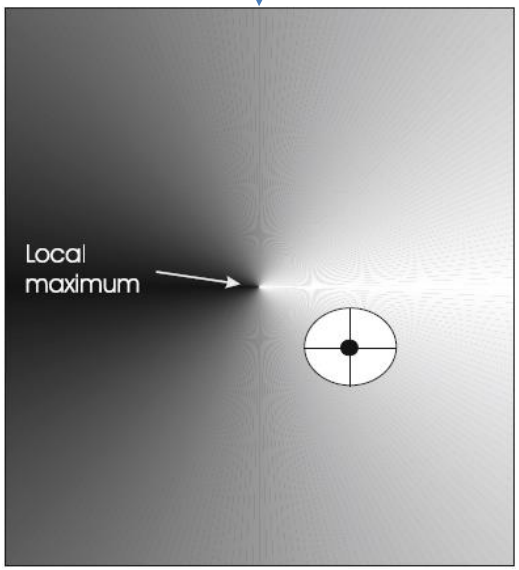
- Uncorrelated mutation with one step size σ:

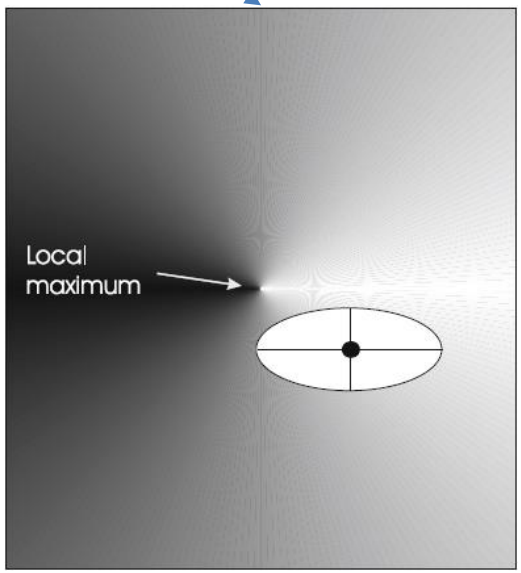
$$x_i' = x_i + \sigma \cdot N_i(0,1)$$

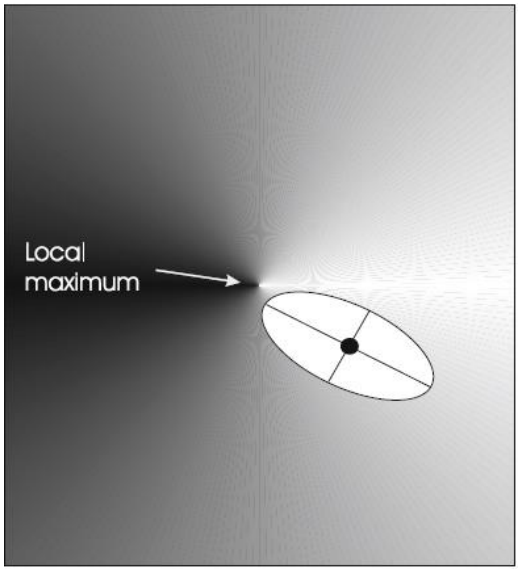
- Uncorrelated mutation with n step sizes:

$$x_i' = x_i + \sigma_i \cdot N_i(0,1)$$

?







Real-valued representation: Nonuniform mutation

- Correlated mutation:

$$\boldsymbol{\delta} \sim N(\mathbf{0}, \mathbf{I}) \xrightarrow{\text{scaling}} \mathbf{S}\boldsymbol{\delta} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \boldsymbol{\delta} \xrightarrow{\text{rotation}} \mathbf{RS}\boldsymbol{\delta} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix} \mathbf{S}\boldsymbol{\delta}$$

bivariate standard normal distribution

$\mathbf{RS}\boldsymbol{\delta}$: $\boldsymbol{\delta}'$

$$\mathrm{Cov}(\boldsymbol{\delta}') = \mathbf{RS}\,\mathrm{Cov}(\boldsymbol{\delta})\,(\mathbf{RS})^{\mathrm{T}} = \begin{bmatrix} a^2\cos^2(\alpha) + b^2\sin^2(\alpha) & (a^2 - b^2)\sin(2\alpha)/2 \\ (a^2 - b^2)\sin(2\alpha)/2 & a^2\sin^2(\alpha) + b^2\cos^2(\alpha) \end{bmatrix}$$

$\mathrm{Cov}(\boldsymbol{\delta}')$: $\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$

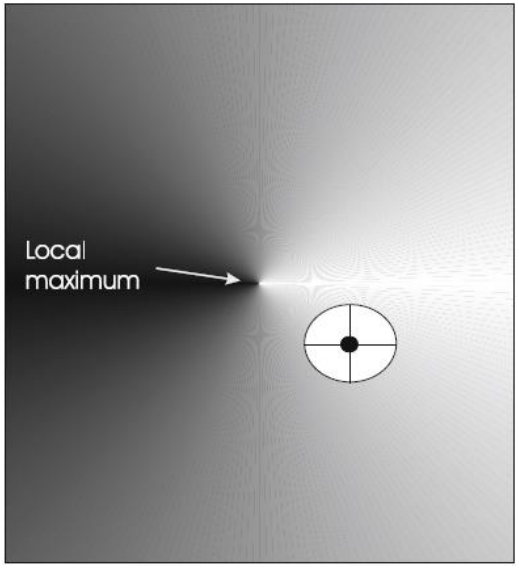
$$\tan(2\alpha) = 2\,c_{12}/(\,c_{11} - c_{22})$$

Generalization: $c_{ii} = \sigma_i^2,\ c_{ij,i\neq j} = \left(\sigma_i^2 - \sigma_j^2\right)\tan(2\alpha_{ij})/2$

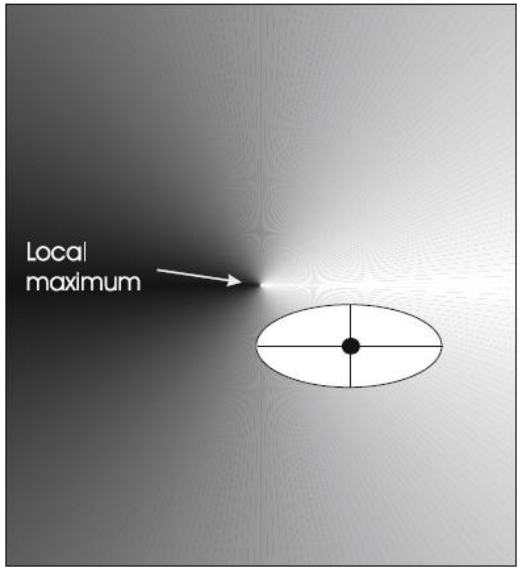
α_{ij}: rotation angle for the i-th and j-th dimension

Real-valued representation: Nonuniform mutation

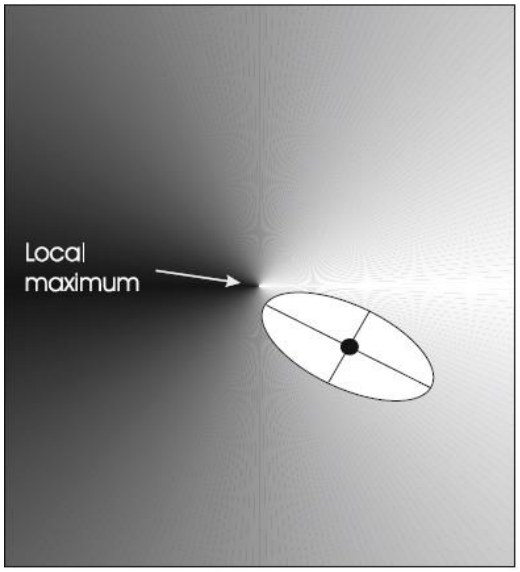
Uncorrelated mutation with one step size σ



Uncorrelated mutation with n step sizes



Correlated mutation



$$\boldsymbol{x}' = \boldsymbol{x} + N(\mathbf{0}, \mathbf{C})$$

$$c_{ii} = \sigma_i^2, \qquad c_{ij,i\neq j} = \left(\sigma_i^2 - \sigma_j^2\right)\tan(2\alpha_{ij})\,/2$$

Real-valued representation: Self-adaptive mutation

- Self-adaptive mutation: Mutation step size σ is not set by user but coevolves with solution

- For example, the genotype is now $(x_1, \ldots, x_n, \sigma)$

  Self-adaptation $\sigma \rightarrow \sigma'$

  $$x_i' = x_i + \sigma' \cdot N_i(0,1)$$

- The fitness of $\boldsymbol{x}'$ can be used to measure the goodness of both the offspring $\boldsymbol{x}'$ and the mutation step size σ'

- Why? Under different circumstances, different step sizes will behave differently

Real-valued representation: Self-adaptive mutation

- Uncorrelated mutation with one step size σ:

$$(x_1, \ldots, x_n, \sigma) \rightarrow (x_1', \ldots, x_n', \sigma')$$

Self-adaptation $\sigma' = \sigma \cdot e^{\tau \cdot N(0,1)}$

$$x_i' = x_i + \sigma' \cdot N_i(0,1)$$

Typically $\tau \propto 1/\sqrt{n}$

$\sigma' < \epsilon_0 \Rightarrow \sigma' = \epsilon_0$

Satisfy these requirements:

- Smaller modifications occur more often than larger ones
- Greater than 0
- The median is 1
- Neural on average: equal prob. of drawing a value and its reciprocal

Real-valued representation: Self-adaptive mutation

- Uncorrelated mutation with n step sizes:

$$(x_1, \dots, x_n, \sigma_1, \dots, \sigma_n) \rightarrow (x_1', \dots, x_n', \sigma_1', \dots, \sigma_n')$$

Self-adaptation

$$\sigma_i' = \sigma_i \cdot e^{\tau' \cdot N(0,1) + \tau \cdot N_i(0,1)}$$

$$x_i' = x_i + \sigma_i' \cdot N_i(0,1)$$

$$\tau' \propto \frac{1}{\sqrt{2n}}, \tau \propto \frac{1}{\sqrt{2\sqrt{n}}}$$

$$\sigma_i' < \epsilon_0 \Rightarrow \sigma_i' = \epsilon_0$$

- $e^{\tau' \cdot N(0,1)}$: overall change; $e^{\tau \cdot N_i(0,1)}$: coordinate-wise change
- The distribution is still lognormal, satisfying those requirements

Real-valued representation: Self-adaptive mutation

- Correlated mutation:

$$\left(x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n, \alpha_1, \ldots, \alpha_{n(n-1)/2}\right)$$

$$\rightarrow \left(x_1', \ldots, x_n', \sigma_1', \ldots, \sigma_n', \alpha_1', \ldots, \alpha_{n(n-1)/2}'\right)$$

Self-adaptation

$$\sigma_i' = \sigma_i \cdot e^{\tau' \cdot N(0,1) + \tau \cdot N_i(0,1)}$$

$$\alpha_j' = \alpha_j + \beta \cdot N_j(0,1)$$

$$\boldsymbol{x}' = \boldsymbol{x} + N(\mathbf{0}, \mathbf{C}')$$

$$\tau' \propto \frac{1}{\sqrt{2n}}, \tau \propto \frac{1}{\sqrt{2\sqrt{n}}}$$

$$\beta \approx 5^o$$

$$\sigma_i' < \epsilon_0 \Rightarrow \sigma_i' = \epsilon_0$$

$$\left|\alpha_j'\right| > \pi \Rightarrow \alpha_j' = \alpha_j' - 2\pi \cdot sign(\alpha_j')$$

Real-valued representation: Recombination

- Discrete recombination: Same as for binary representation, e.g., m-point crossover and uniform crossover

- Arithmetic recombination: Create offspring "between" parents

$$z_i = (1-\alpha)x_i + \alpha y_i, \text{ where } \alpha \in [0,1]$$

- Blend recombination: Create offspring in a larger region

$$z_i = (1-\gamma)x_i + \gamma y_i, \text{ where } \gamma = (1+2\alpha)u - \alpha, u \in [0,1]$$

Real-valued representation: Arithmetic recombination

- Single arithmetic recombination:

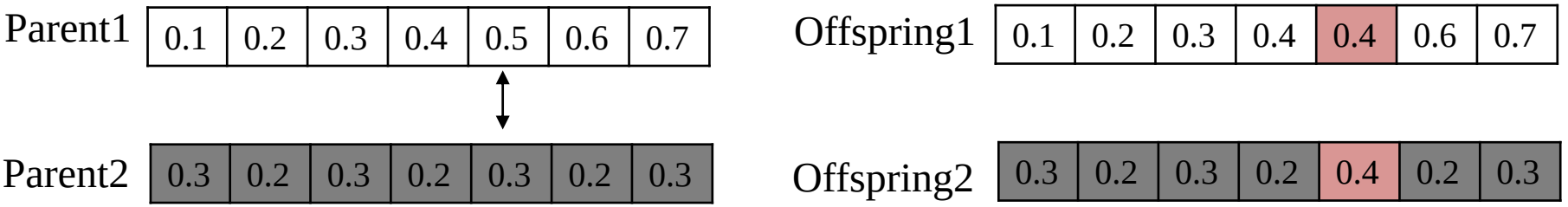
$$(x_1, \dots, x_n) \qquad (y_1, \dots, y_n)$$

Select a random k

$$(x_1, \dots, x_{k-1}, \alpha y_k + (1-\alpha)x_k, x_{k+1}, \dots, x_n)$$

$$(y_1, \dots, y_{k-1}, \alpha x_k + (1-\alpha)y_k, y_{k+1}, \dots, y_n)$$

Example with $\alpha = 0.5$



Real-valued representation: Arithmetic recombination

- Simple arithmetic recombination:

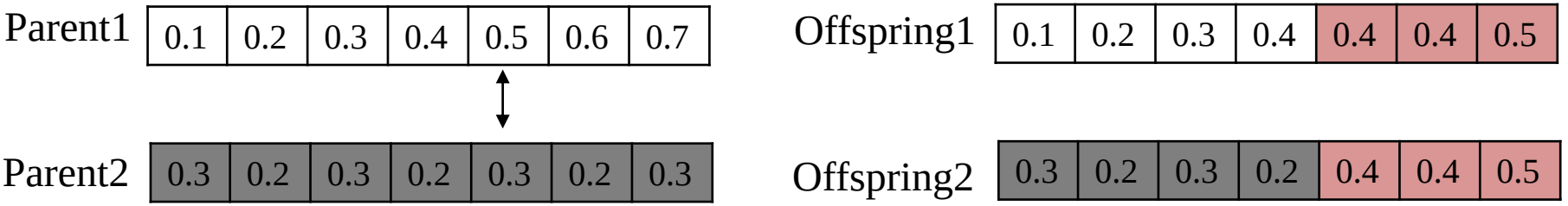
$(x_1, \ldots, x_n)$ $\quad$ $(y_1, \ldots, y_n)$

Select a random k

$(x_1, \ldots, x_{k-1}, \alpha y_k + (1-\alpha)x_k, \ldots, \alpha y_n + (1-\alpha)x_n)$

$(y_1, \ldots, y_{k-1}, \alpha x_k + (1-\alpha)y_k, \ldots, \alpha x_n + (1-\alpha)y_n)$

Example with $\alpha = 0.5$



Real-valued representation: Arithmetic recombination

- Whole arithmetic recombination:

$$(x_1, \ldots, x_n) \qquad (y_1, \ldots, y_n)$$

$$(\alpha y_1 + (1-\alpha)x_1, \ldots, \alpha y_n + (1-\alpha)x_n)$$
$$(\alpha x_1 + (1-\alpha)y_1, \ldots, \alpha x_n + (1-\alpha)y_n)$$

Example with $\alpha = 0.5$

Parent1	0.1	0.2	0.3	0.4	0.5	0.6	0.7
Parent2	0.3	0.2	0.3	0.2	0.3	0.2	0.3

Offspring1	0.2	0.2	0.3	0.3	0.4	0.4	0.5
Offspring2	0.2	0.2	0.3	0.3	0.4	0.4	0.5

Real-valued representation: Blend recombination

- Blend recombination: Create offspring in a larger region

$$z_i = (1-\gamma)x_i + \gamma y_i, \text{ where } \gamma = (1+2\alpha)u - \alpha, u \in [0,1]$$

$$z_i = x_i + (y_i - x_i)(1+2\alpha)u - \alpha(y_i - x_i)$$

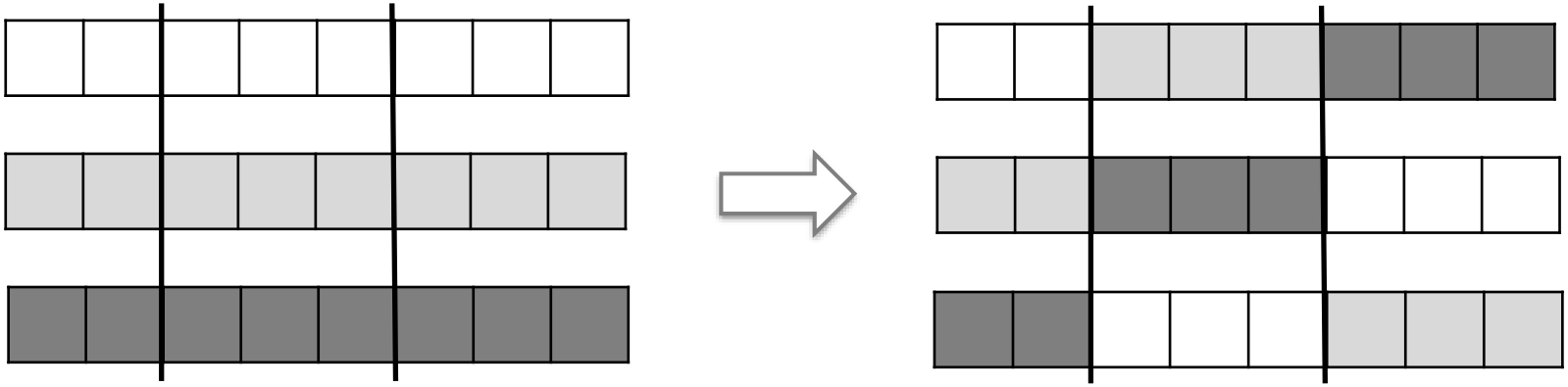
$d_i > 0$

$$u \sim U(0,1) \quad \Rightarrow \quad z_i \sim U(x_i - \alpha d_i, y_i + \alpha d_i)$$

How about $z_i = (1-\gamma)y_i + \gamma x_i$?

Multi-parent recombination

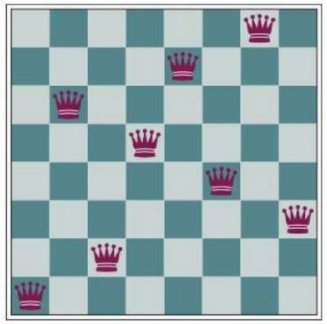
- For example, diagonal crossover for m parents:
  - Choose $m - 1$ crossover points randomly
  - Compose m offspring from the segments of the parents in along a "diagonal", wrapping around



Permutation representation

- Genotype space: a permutation of a fixed set of values

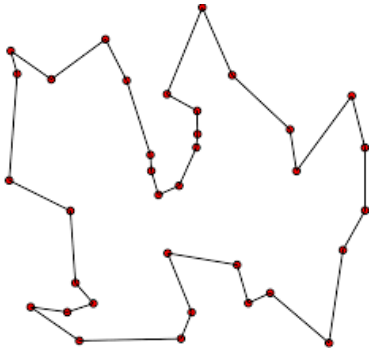
8-queens problem



position of the queen on each column

1	6	2	5	7	4	8	3

Traveling salesman problem



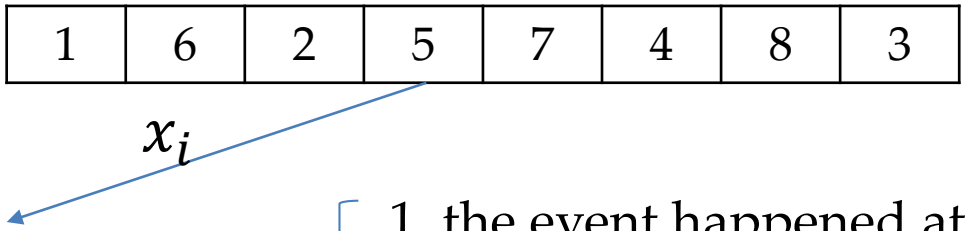
the order of visiting cities

1	6	2	5	7	4	8	3

6	2	5	7	4	8	3	1

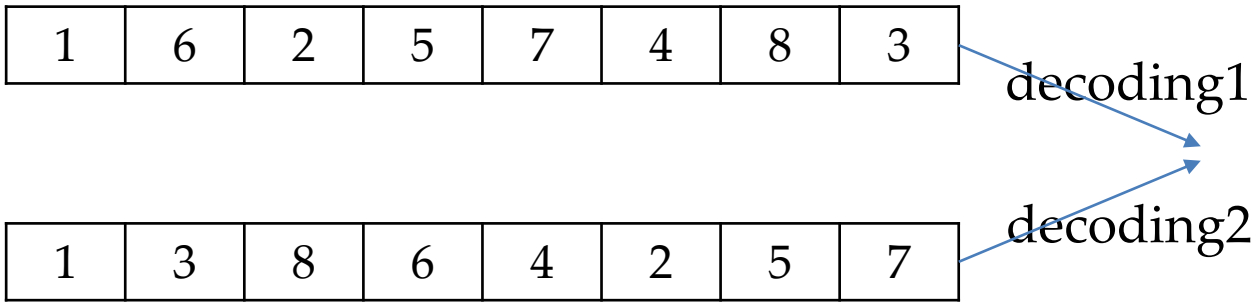
Permutation representation

- Genotype space: a permutation of a fixed set of values



Two ways of decoding:

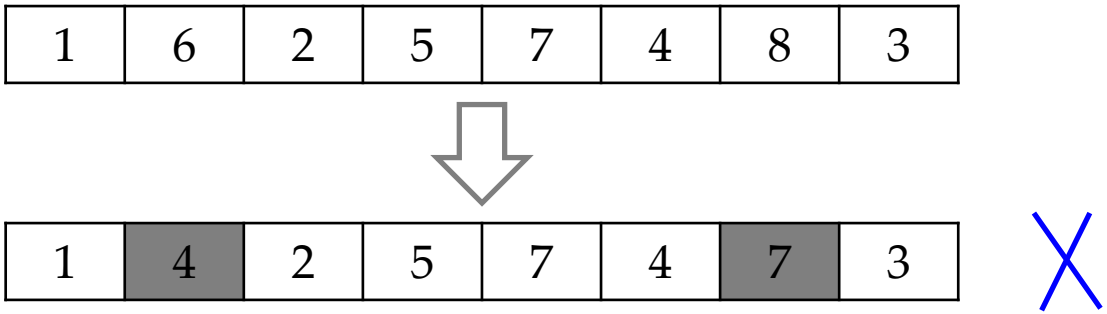
1. the event happened at the i-th position
2. the position where the i-th event happens



the same order in which a sequence of events occur

Permutation representation: Mutation

- Mutation operators for binary, integer and real-valued representation will lead to inadmissible solutions

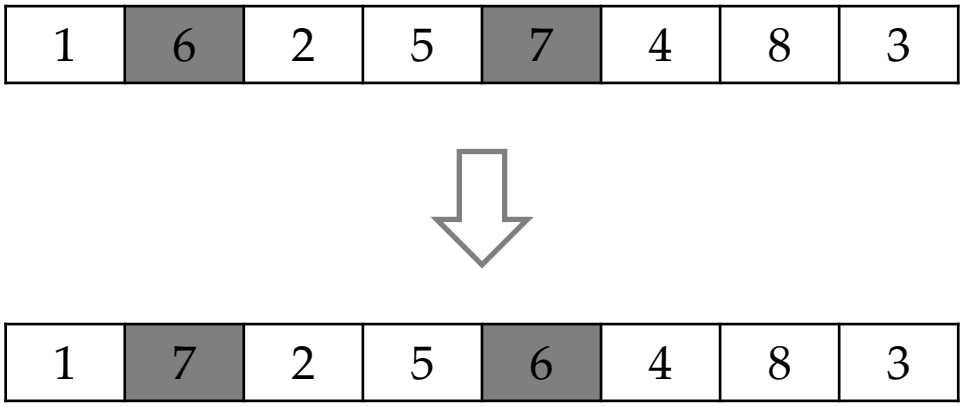


- Common mutation for permutation representation
  - Swap mutation
  - Insert mutation
  - Scramble mutation
  - Inversion mutation

The mutation probability now reflects the probability of applying mutation, rather than altering a single gene

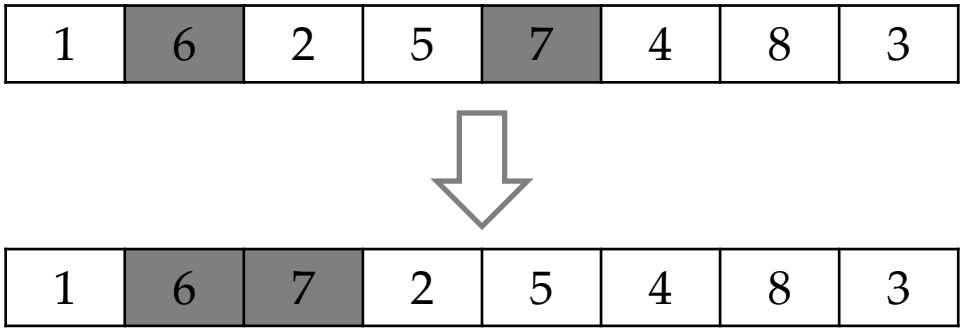
Permutation representation: Swap mutation

- Swap mutation:
  - Select two positions randomly
  - Swap their values



Permutation representation: Insert mutation

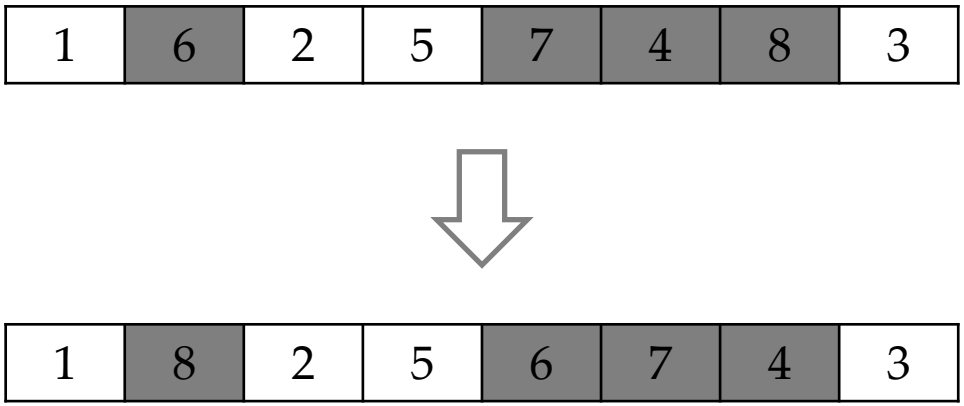
- Insert mutation:
  - Select two positions randomly
  - Move the second next to the first
  - Shift the rest along to accommodate



Preserve most of the order and the adjacency information

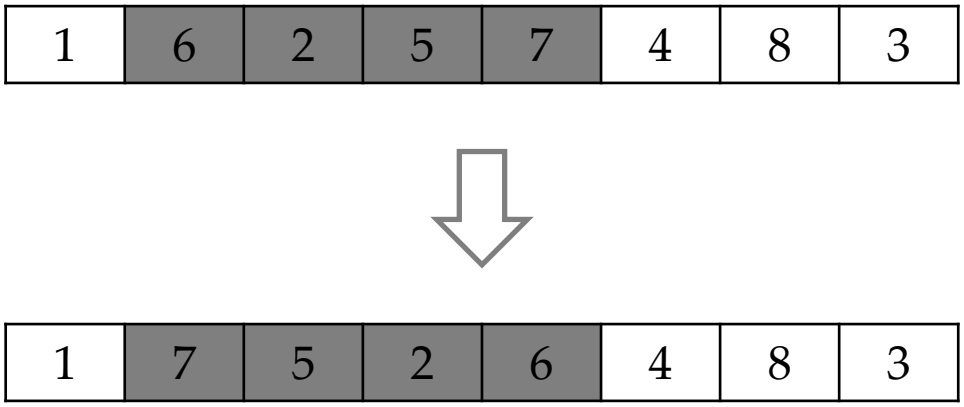
Permutation representation: Scramble mutation

- Scramble mutation:
  - Select a subset of positions randomly
  - Rearrange their values randomly



Permutation representation: Inversion mutation

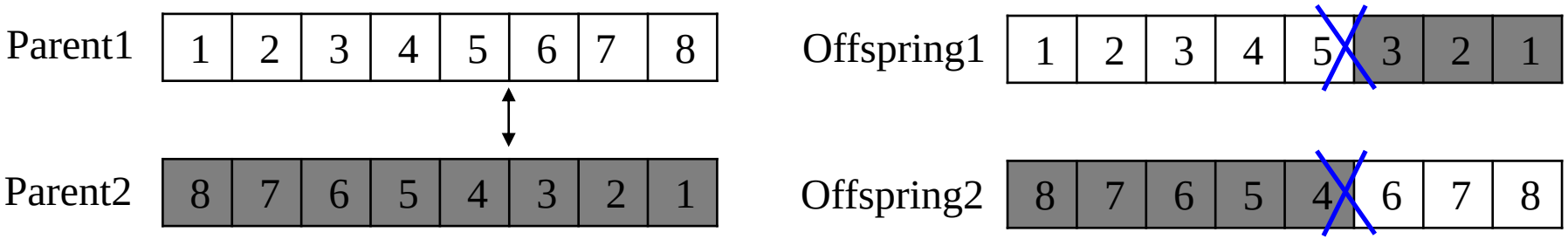
- Inversion mutation:
  - Select two positions randomly
  - Invert the values between them



Preserve most adjacency information (only breaks two links), but disruptive of order information

Permutation representation: Recombination

- Recombination operators for binary, integer and real-valued representation will lead to inadmissible solutions



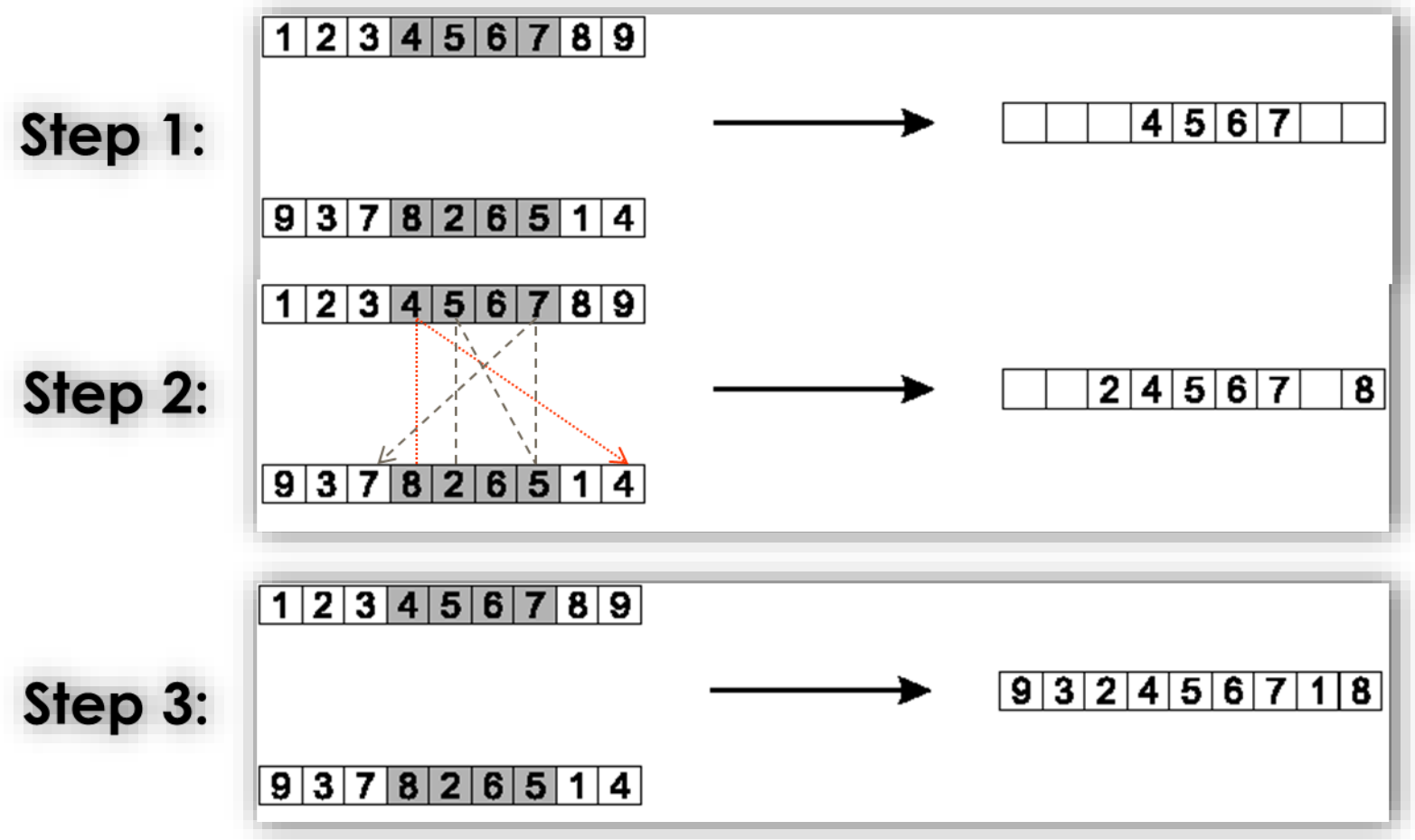
- Common recombination for permutation representation
  - Partially mapped crossover
  - Edge crossover
  - Order crossover
  - Cycle crossover

combine order or adjacency information from the two parents

Permutation representation: Partially mapped crossover

- Partially mapped crossover:
  1. Choose two crossover points randomly, and copy the segment between them from parent P1 into the first offspring
  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 by *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 by *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 first offspring can be filled from P2
  7. Create the second offspring analogously with parental roles reversed

Permutation representation: Partially mapped crossover



Permutation representation: Edge crossover

- First step, construct a table listing which edges are present in the two parents; if an edge is common, mark with a +
- For example,

Parent1	1	2	3	4	5	6	7	8	9
Parent2	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+		

Permutation representation: Edge crossover

- Edge crossover: After constructing the edge table,
  1. Pick an initial element, entry, 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 the 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: a new element is chosen at random

Permutation representation: Edge crossover

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+		

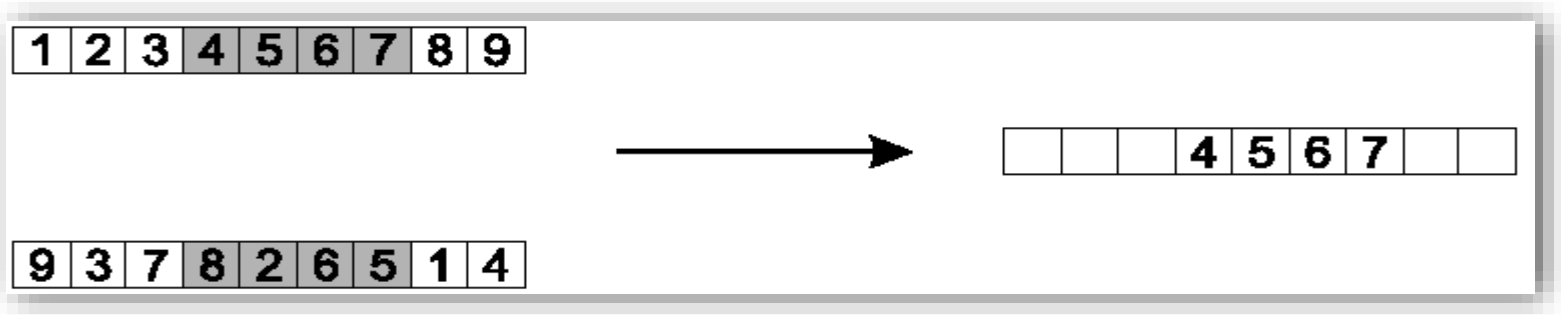
Choices	Element selected	Reason	Partial result
All	1	Random	[*1*]
2,5,4,9	5	Shortest list	[*1 5*]
4,6	6	Common edge	[*1 5 6*]
2,7	2	Random choice (both have two items in list)	[*1 5 6 2*]
3,8	8	Shortest list	[*1 5 6 2 8*]
7,9	7	Common edge	[*1 5 6 2 8 7*]
3	3	Only item in list	[*1 5 6 2 8 7 3*]
4,9	9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

Permutation representation: Order crossover

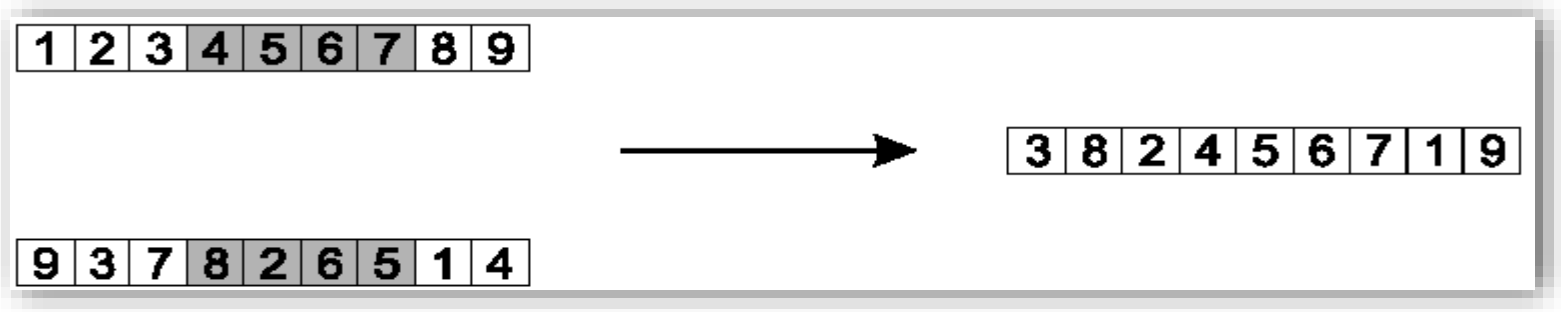
- Order crossover:
  1. Choose two crossover points randomly
  2. Copy the segment between them from the first parent to the first offspring
  3. Copy the numbers that are not in the segment, to the first offspring:
     - starting right from the second crossover point
     - using the order of the second parent
     - and wrapping around at the end
  4. Create the second offspring in an analogous manner, with the parent roles inversed

Permutation representation: Order crossover

1. Copy a randomly selected segment from the first parent



2. Copy the remaining numbers into the offspring in the order that they appear in the second parent

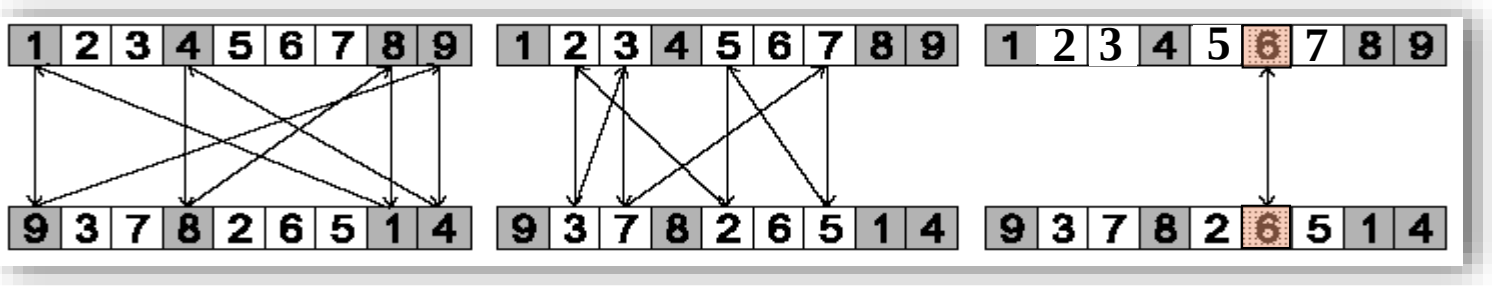


Permutation representation: Cycle crossover

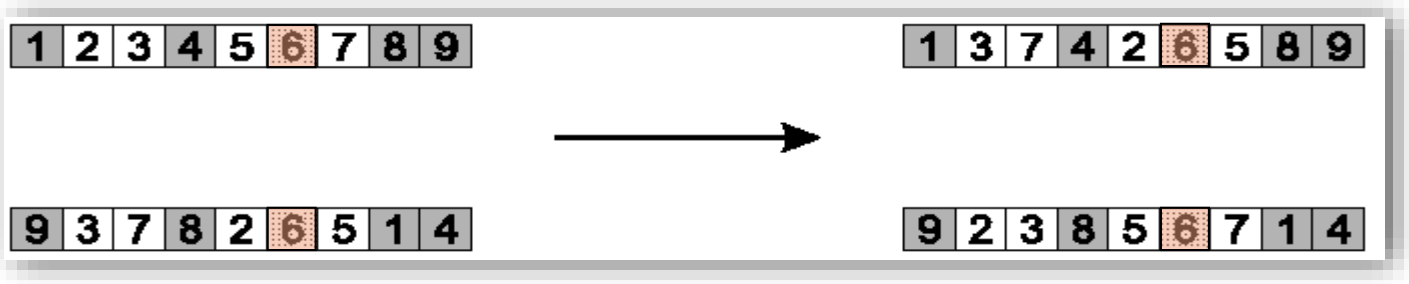
- Cycle crossover:
  1. Divide the alleles into cycles
     - Start with the first unused position and allele of P1
     - Look at the allele in the same position in P2
     - Go to the position with the same allele in P1
     - Add this allele to the cycle
     - Repeat steps 2-4 until arriving at the first allele of P1
  2. Create the offspring by selecting alternate cycles from each parent

Permutation representation: Cycle crossover

1. Identify cycles



2. Copy alternate cycles into offspring

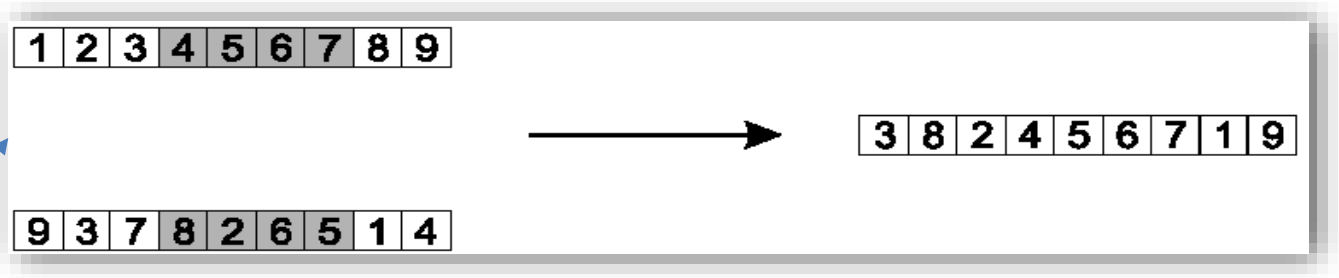


Permutation representation: Recombination

- Partially mapped crossover
- Edge crossover

combine adjacency information from the two parents

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+		



- Order crossover
- Cycle crossover

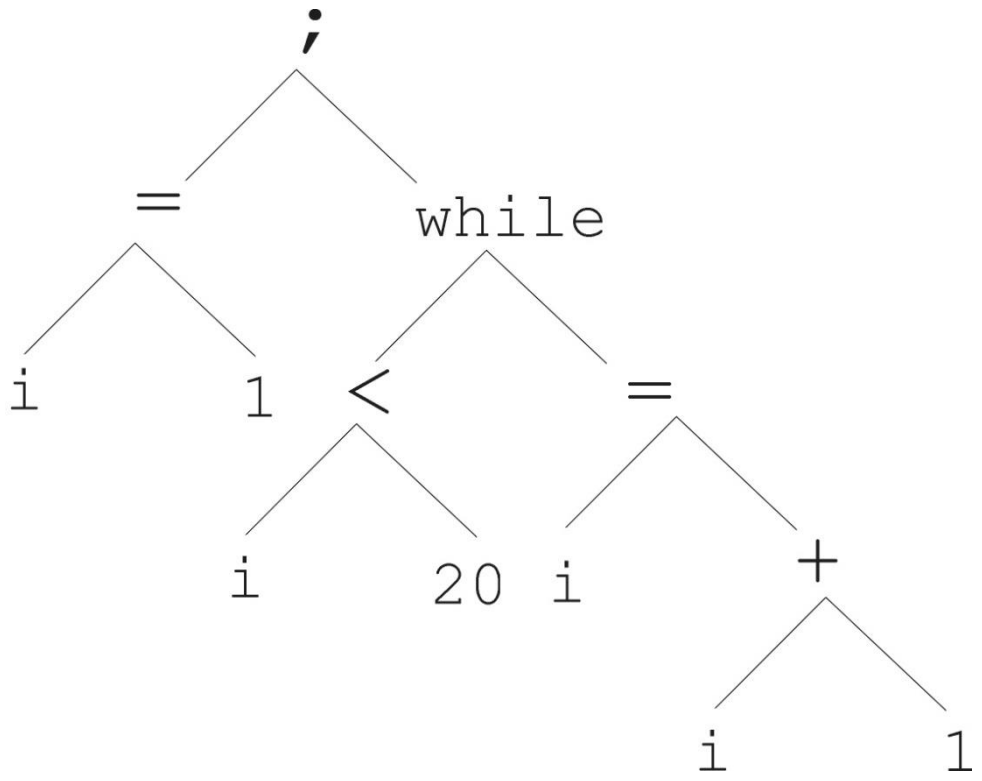
combine order information from the two parents

Tree representation

- Genotype space: a tree

Program:

$i = 1;$
while $(i < 20)$
{
    $i = i + 1;$
}

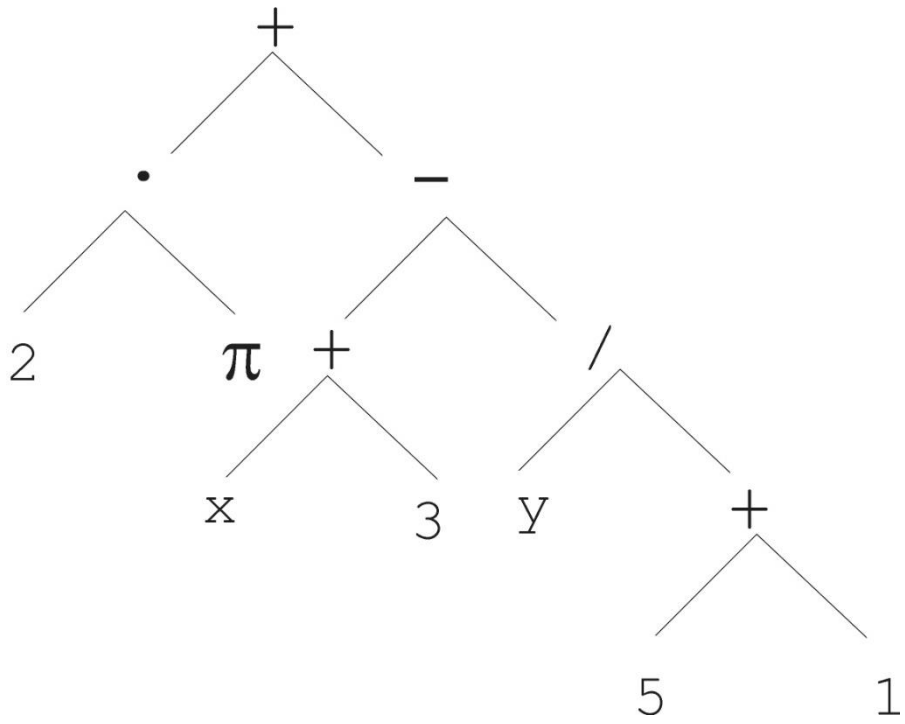


Tree representation

- Genotype space: a tree

Arithmetic formula:

$$2 \cdot \pi + \left((x + 3) - \frac{y}{5 + 1} \right)$$



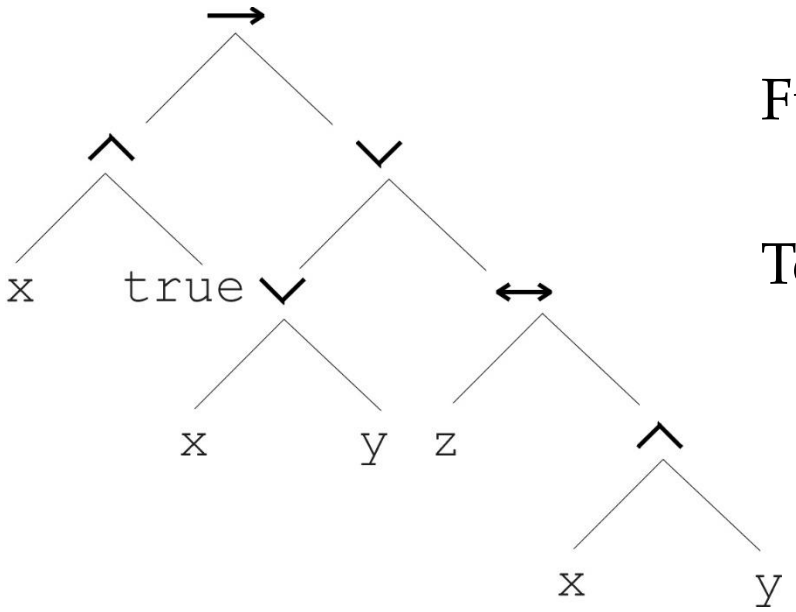
Tree representation

- Genotype space: a tree

Variable structure

Logical formula:

$$(x \wedge true) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$$

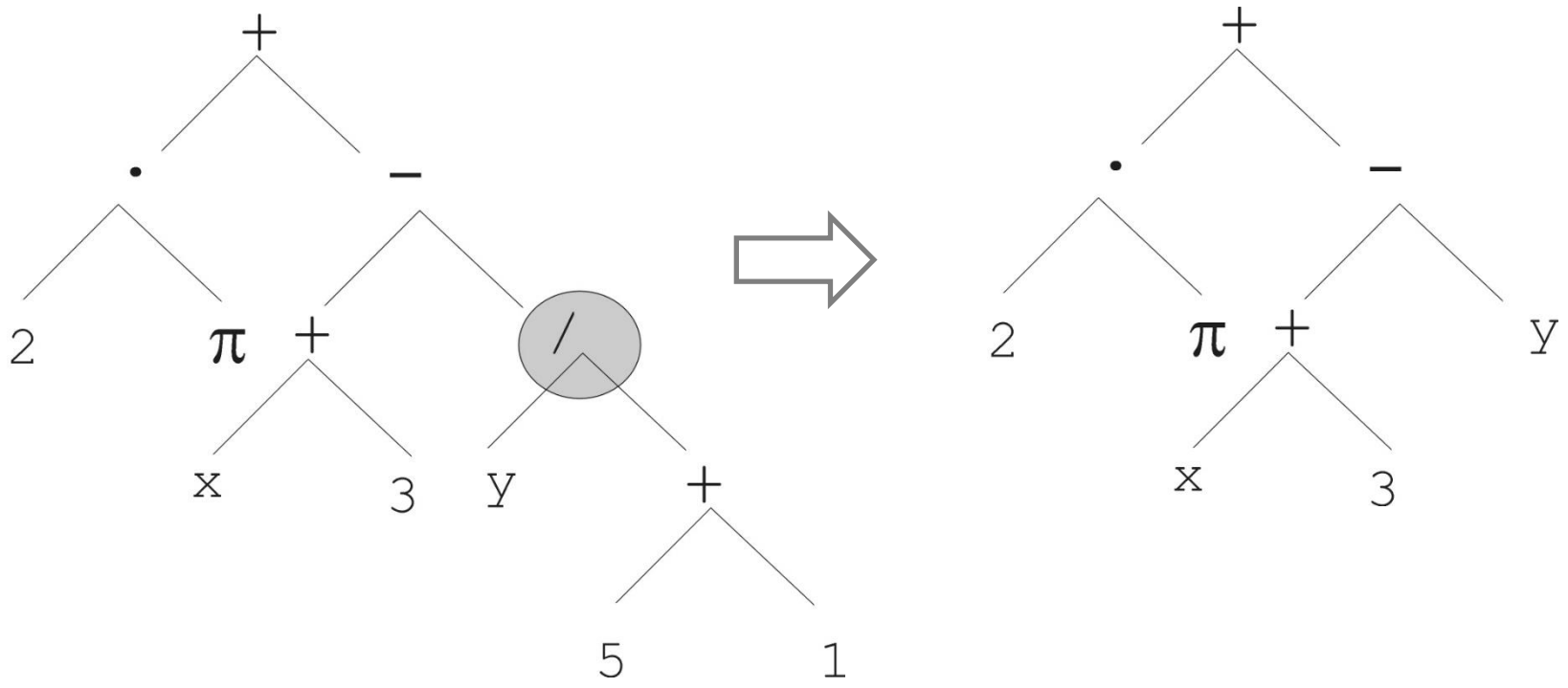


Function set → Internal nodes

Terminal set → Leaves

Tree representation: Mutation

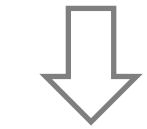
- Mutation: replace randomly chosen subtree by randomly generated tree



Tree representation: Recombination

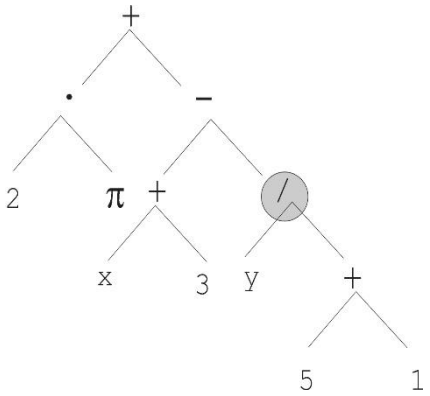
- Recombination: exchange two randomly chosen subtrees among the parents

Assume uniform selection within each parent for exchanging

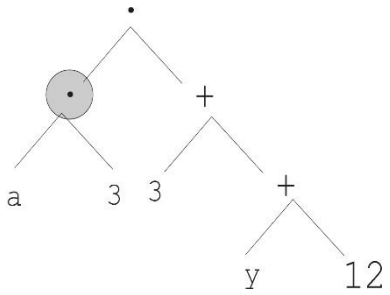


The probability:

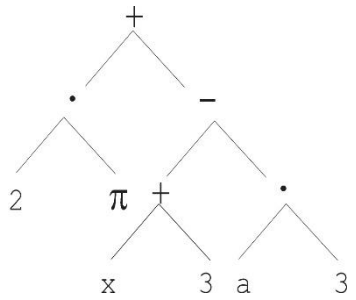
$$\frac{1}{13} \cdot \frac{1}{9}$$



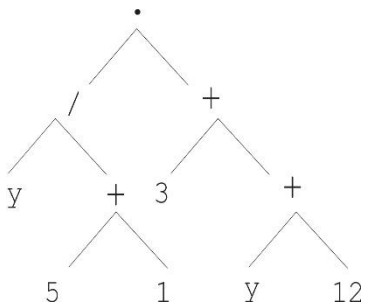
Parent 1



Parent 2



Child 1



Child 2

Mutation or recombination

- Decade long debate: which one is better
  - Evolutionary programming: originally without recombination
  - Genetic programming: originally without mutation
- Recombination is explorative, which can make a big jump to an area somewhere "in between" two (parent) areas
- Mutation is exploitative, which creates random small diversions, thereby staying near (in the area of) the parent
- Now, it is good to have both in general

Summary

- Binary representation
- Integer representation
- Real-valued representation
- Permutation representation
- Tree representation

Representation

Mutation

Recombination

References

- A. E. Eiben and J. E. Smith. Introduction to Evolutionary Computing. Chapter 4.