Mutation

- depends on the search space
- generates one offspring from one parent
- makes only small changes with high probability

Mutation operators for $\{0,1\}^n$

- Standard Bit Mutation (parameter mutation-probability p_m)
  Copy x to y and invert every bit of y independently with probability p_m.
  - expected number of inverted bits $= p_m \cdot n$
  - $p_m \in (0;1/2]$ to favor small changes
  - most often used mutation probability $p_m = 1/n$
- b-Bit Mutation (parameter b)
  Copy x to y, choose randomly uniformly
  b different positions in y,
  and invert the bits of y at these positions
  - b often very small, most often $b = 1$
  - easier to analyze than standard-bit-mutation
  - Behavior can vary greatly from standard-bit-mutation

Mutation operators for $\mathbb{R}^n$

most often add random vector m on x to generate y

almost always $\mathsf{E}(m) = 0^n$

often $m = (m'_1, m'_2, \cdots, m'_n)$ where all $m' \in \mathbb{R}$

random choice of $m' \in \mathbb{R}$ where $\mathsf{E}(m') = 0$

- restricted $m' \in [a; b]$, most often $m' \in [-a; a]$ uniformly
- unrestricted often normally distributed with probability density $\frac{1}{\sqrt{2\pi\sigma}} e^{-r^2/(2\sigma^2)}$ $\rightsquigarrow$ $\mathsf{E}(m') = 0$, $\mathsf{Var}(m') = \sigma^2$
  sometimes $\sigma = 1$ fixed and use $s \cdot m'$ instead of m'

How to choose s

- most often s is not fixed
- Idea choose large s, when far away from the optimum
- Idea choose small s, when close to the optimum

Mutation operators for S_n

- we don't discuss problem-specific mutation operators
- Exchange

  Copy parent, choose $i \neq j \in \{1, 2, \dots, n\}$ uniformly randomly. Swap elements with positions i and j.
- Jump

  Copy parent, choose $i \neq j \in \{1, 2, \dots, n\}$ uniformly randomly. Delete element at position i and insert it at position j (elements in-between are moved).
- Combination of exchange and jump

  Choose $k \in \mathbb{N}_0$ randomly according to Poisson-distribution with parameter 1, i. e. Prob $(k = r) = \frac{1}{e \cdot r!}$.

  Do $k + 1$ iterations, where randomly uniformly either an exchange or a jump is done.

Crossover

- depends on the search space
- usually generates one offspring from at least two parents
- usually generates offspring, that are 'similar' to the parents
- sometimes two parents generate exactly two offspring

Crossover operators for $\{0,1\}^n$ (1)

- k-point crossover (parameter k)

  Generate offspring y from parents x_1 and x_2.

  Choose k different crossover points

  $p_1, \ldots, p_k \in \{1, 2, \ldots, n-1\}$

  with $p_1 < p_2 < p_3 \cdots < p_k$.

  $y = x_1[1]x_1[2] \cdots x_1[p_1]x_2[p_1+1] \cdots x_2[p_2]x_1[p_2+1] \cdots$.

  - most often k very small, usually $k = 2$ or even $k = 1$

- uniform crossover

  Generate offspring y from parents x_1 and x_2.

  For every position $i \in \{1, \ldots, n\}$ choose y_i randomly uniformly from either $x_1[i]$ or $x_2[i]$.

  - $\forall i\colon x_1[i] = x_2[i] \Rightarrow y[i] = x_1[i]$
  - generates uniformly one of the possible offspring of x_1 and x_2
  - in general there are much more possible offspring than using k-point crossover

Crossover operators for $\{0,1\}^n$ (2)

- Genepool crossover

  Generate offspring y from parents $x_1, x_2, \ldots, x_\mu$.

  Choose $y[i] = 1$ with probability $\sum_{j=1}^{\mu} x_j[i]/\mu$.

  - often the whole population acts as parents
  - theoretically motivated

Crossover operators for $\mathbb{R}^n$

- k-point crossover
  works as in $\{0,1\}^n$
- uniform crossover
  works as in $\{0,1\}^n$
- arithmetic crossover
  Generate offspring y from parents $x_1, x_2, \ldots, x_\mu$.
  Generate $y = \sum_{i=1}^{\mu} \alpha_i \cdot x_i$ with parameters $\alpha_1, \ldots, \alpha_\mu$
  where $\sum_{i=1}^{\mu} \alpha_i = 1$.
  - often $\alpha_i = 1/\mu$ for all i
  - for $\alpha_i = 1/\mu$, y is centroid of parents
  - for $\alpha_i = 1/\mu$, it's also called intermediate crossover
  - arithmetic crossover is the only deterministic variation operator

Crossover operators for S_n

- most often offspring y is generated from two parents x_1, x_2
- frequent idea Choose two positions in x_1, and sort elements in this interval according to their order in x_2 (concrete examples: PMX, CX).
- many problem-specific crossover operators (e. g. edge recombination or inver-over for TSP)
- no crossover operators successful over a wide range of applications