Evolutionary Computing
Lecture 3
Evolution Strategies. Memetic Algorithms

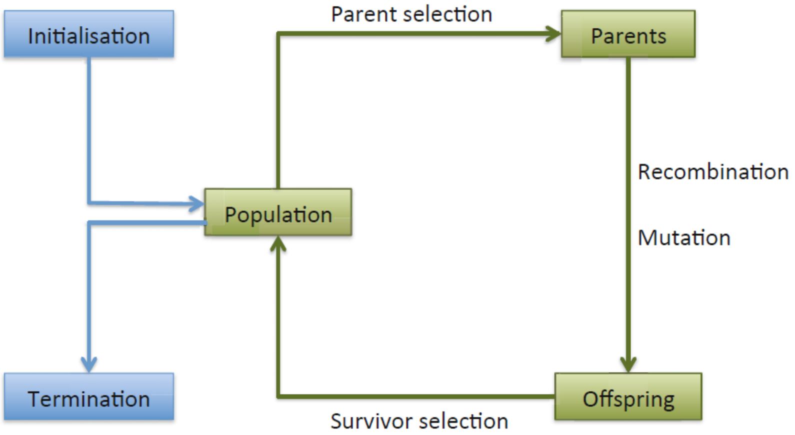
Danylo Tavrov

Outline

Outline

1. Evolution Strategies

2. Memetic Algorithms



Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer-Verlag, Berlin, Heidelberg (2015), Fig. 3.2

- As was mentioned in Lecture 1, there exist several different dialects of evolutionary algorithms, differing mostly in representation of individuals
- *Genetic algorithms*: strings over a finite (typically binary) alphabet
- *Evolution strategies*: real-valued vectors
- *Evolutionary programming*: originally finite state machines, but then real-valued vectors as well
- *Genetic programming*: parse trees
- In previous lectures, we focused on genetic algorithms
- In this lecture, we will discuss evolution strategies

Different Types of Evolutionary Algorithms

- As was mentioned in Lecture 1, there exist several different dialects of evolutionary algorithms, differing mostly in representation of individuals
- *Genetic algorithms*: strings over a finite (typically binary) alphabet
- *Evolution strategies*: real-valued vectors
- *Evolutionary programming*: originally finite state machines, but then real-valued vectors as well
- *Genetic programming*: parse trees
- In previous lectures, we focused on genetic algorithms
- In this lecture, we will discuss evolution strategies

- As was mentioned in Lecture 1, there exist several different dialects of evolutionary algorithms, differing mostly in representation of individuals
- *Genetic algorithms*: strings over a finite (typically binary) alphabet
- *Evolution strategies*: real-valued vectors
- *Evolutionary programming*: originally finite state machines, but then real-valued vectors as well
- *Genetic programming*: parse trees
- In previous lectures, we focused on genetic algorithms
- In this lecture, we will discuss evolution strategies

Different Types of Evolutionary Algorithms

- As was mentioned in Lecture 1, there exist several different dialects of evolutionary algorithms, differing mostly in representation of individuals
- *Genetic algorithms*: strings over a finite (typically binary) alphabet
- *Evolution strategies*: real-valued vectors
- *Evolutionary programming*: originally finite state machines, but then real-valued vectors as well
- *Genetic programming*: parse trees
- In previous lectures, we focused on genetic algorithms
- In this lecture, we will discuss evolution strategies

Different Types of Evolutionary Algorithms

- As was mentioned in Lecture 1, there exist several different dialects of evolutionary algorithms, differing mostly in representation of individuals
- *Genetic algorithms*: strings over a finite (typically binary) alphabet
- *Evolution strategies*: real-valued vectors
- *Evolutionary programming*: originally finite state machines, but then real-valued vectors as well
- *Genetic programming*: parse trees
- In previous lectures, we focused on genetic algorithms
- In this lecture, we will discuss evolution strategies

Different Types of Evolutionary Algorithms

- As was mentioned in Lecture 1, there exist several different dialects of evolutionary algorithms, differing mostly in representation of individuals
- *Genetic algorithms*: strings over a finite (typically binary) alphabet
- *Evolution strategies*: real-valued vectors
- *Evolutionary programming*: originally finite state machines, but then real-valued vectors as well
- *Genetic programming*: parse trees
- In previous lectures, we focused on genetic algorithms
- In this lecture, we will discuss evolution strategies

Different Types of Evolutionary Algorithms

- As was mentioned in Lecture 1, there exist several different dialects of evolutionary algorithms, differing mostly in representation of individuals
- *Genetic algorithms*: strings over a finite (typically binary) alphabet
- *Evolution strategies*: real-valued vectors
- *Evolutionary programming*: originally finite state machines, but then real-valued vectors as well
- *Genetic programming*: parse trees
- In previous lectures, we focused on genetic algorithms
- In this lecture, we will discuss evolution strategies

Motivation

- If the functional relation between the variables and the objective function is unknown, one is forced to experiment[1]
- This is sometimes called **experimental**, or **blind**, **optimization**
- Systematic investigation of all possible states of the system is costly
- On the other hand, random sampling of various combinations is too unreliable
- A search procedure must be *significantly more effective* if it systematically exploits information about preceding attempts
- Experiments are characterized by unavoidable effect of (stochastic) disturbance on the measured results

1. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- If the functional relation between the variables and the objective function is unknown, one is forced to experiment[1]
- This is sometimes called **experimental**, or **blind**, **optimization**
- Systematic investigation of all possible states of the system is costly
- On the other hand, random sampling of various combinations is too unreliable
- A search procedure must be *significantly more effective* if it systematically exploits information about preceding attempts
- Experiments are characterized by unavoidable effect of (stochastic) disturbance on the measured results

1. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- If the functional relation between the variables and the objective function is unknown, one is forced to experiment[1]
- This is sometimes called **experimental**, or **blind**, **optimization**
- Systematic investigation of all possible states of the system is costly
- On the other hand, random sampling of various combinations is too unreliable
- A search procedure must be *significantly more effective* if it systematically exploits information about preceding attempts
- Experiments are characterized by unavoidable effect of (stochastic) disturbance on the measured results

1. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- If the functional relation between the variables and the objective function is unknown, one is forced to experiment[1]
- This is sometimes called **experimental**, or **blind**, **optimization**
- Systematic investigation of all possible states of the system is costly
- On the other hand, random sampling of various combinations is too unreliable
- A search procedure must be *significantly more effective* if it systematically exploits information about preceding attempts
- Experiments are characterized by unavoidable effect of (stochastic) disturbance on the measured results

1. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- If the functional relation between the variables and the objective function is unknown, one is forced to experiment[1]
- This is sometimes called **experimental**, or **blind**, **optimization**
- Systematic investigation of all possible states of the system is costly
- On the other hand, random sampling of various combinations is too unreliable
- A search procedure must be *significantly more effective* if it systematically exploits information about preceding attempts
- Experiments are characterized by unavoidable effect of (stochastic) disturbance on the measured results

1. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- If the functional relation between the variables and the objective function is unknown, one is forced to experiment[1]
- This is sometimes called **experimental**, or **blind**, **optimization**
- Systematic investigation of all possible states of the system is costly
- On the other hand, random sampling of various combinations is too unreliable
- A search procedure must be *significantly more effective* if it systematically exploits information about preceding attempts
- Experiments are characterized by unavoidable effect of (stochastic) disturbance on the measured results

1. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- **Evolution strategies** (ES) take this all into account
- Originally ES were invented in the early 1960s by Ingo Rechenberg and Hans-Paul Schwefel
- To be precise, they invented the so-called *two-membered ES* (discussed below)
- Rechenberg hypothesis was as follows:

- **Evolution strategies** (ES) take this all into account
- Originally ES were invented in the early 1960s by Ingo Rechenberg and Hans-Paul Schwefel
- To be precise, they invented the so-called *two-membered ES* (discussed below)
- Rechenberg hypothesis was as follows:
  The method of organic evolution represents an optimal strategy for the adaptation of living things to their environment [...] it should therefore be worthwhile to take over the principles of biological evolution for the optimization of technical systems.

- **Evolution strategies** (ES) take this all into account
- Originally ES were invented in the early 1960s by Ingo Rechenberg and Hans-Paul Schwefel
- To be precise, they invented the so-called *two-membered ES* (discussed below)
- Rechenberg hypothesis was as follows:
  The method of organic evolution represents an optimal strategy for the adaptation of living things to their environment [...] it should therefore be worthwhile to take over the principles of biological evolution for the optimization of technical systems.

- **Evolution strategies** (ES) take this all into account
- Originally ES were invented in the early 1960s by Ingo Rechenberg and Hans-Paul Schwefel
- To be precise, they invented the so-called *two-membered ES* (discussed below)
- Rechenberg hypothesis was as follows:
  The method of organic evolution represents an optimal strategy for the adaptation of living things to their environment [...] it should therefore be worthwhile to take over the principles of biological evolution for the optimization of technical systems.

- **Evolution strategies** (ES) take this all into account
- Originally ES were invented in the early 1960s by Ingo Rechenberg and Hans-Paul Schwefel
- To be precise, they invented the so-called *two-membered ES* (discussed below)
- Rechenberg hypothesis was as follows:
  The method of organic evolution represents an optimal strategy for the adaptation of living things to their environment [...] it should therefore be worthwhile to take over the principles of biological evolution for the optimization of technical systems.

Original Two-Membered ES

1. **Initialization**: $t = 0$, create a random vector $\mathbf{x}^t = \left(x_1^t, \ldots, x_n^t\right)^\top \in \mathbb{R}^n$
2. Test **termination condition**. If it holds, stop, continue otherwise
3. **Mutation**: draw $z_i \overset{\text{i.i.d.}}{\sim} N(0, \sigma)$ and create $y_i^t = x_i^t + z_i$, $i = 1, \ldots, n$
4. **Selection**[2]: if $f(\mathbf{x}^t) \leq f(\mathbf{y}^t)$, choose $\mathbf{x}^{t+1} = \mathbf{x}^t$, otherwise choose $\mathbf{x}^{t+1} = \mathbf{y}^t$
5. Set $t = t + 1$ and go to step 2

2. For minimization problems

Original Two-Membered ES

1. **Initialization**: $t = 0$, create a random vector $\mathbf{x}^t = \left(x_1^t, \ldots, x_n^t\right)^\top \in \mathbb{R}^n$
2. Test **termination condition**. If it holds, stop, continue otherwise
3. **Mutation**: draw $z_i \overset{\text{i.i.d.}}{\sim} N(0, \sigma)$ and create $y_i^t = x_i^t + z_i$, $i = 1, \ldots, n$
4. **Selection**[2]: if $f(\mathbf{x}^t) \leq f(\mathbf{y}^t)$, choose $\mathbf{x}^{t+1} = \mathbf{x}^t$, otherwise choose $\mathbf{x}^{t+1} = \mathbf{y}^t$
5. Set $t = t + 1$ and go to step 2

2. For minimization problems

Original Two-Membered ES

1. **Initialization**: $t = 0$, create a random vector $\mathbf{x}^t = \left(x_1^t, \ldots, x_n^t\right)^\top \in \mathbb{R}^n$
2. Test **termination condition**. If it holds, stop, continue otherwise
3. **Mutation**: draw $z_i \overset{\text{i.i.d.}}{\sim} N(0, \sigma)$ and create $y_i^t = x_i^t + z_i$, $i = 1, \ldots, n$
4. **Selection**[2]: if $f(\mathbf{x}^t) \leq f(\mathbf{y}^t)$, choose $\mathbf{x}^{t+1} = \mathbf{x}^t$, otherwise choose $\mathbf{x}^{t+1} = \mathbf{y}^t$
5. Set $t = t + 1$ and go to step 2

2. For minimization problems

Original Two-Membered ES

1. **Initialization**: $t = 0$, create a random vector $\mathbf{x}^t = \left(x_1^t, \ldots, x_n^t\right)^\top \in \mathbb{R}^n$
2. Test **termination condition**. If it holds, stop, continue otherwise
3. **Mutation**: draw $z_i \overset{\text{i.i.d.}}{\sim} N(0, \sigma)$ and create $y_i^t = x_i^t + z_i$, $i = 1, \ldots, n$
4. **Selection**[2]: if $f(\mathbf{x}^t) \leq f(\mathbf{y}^t)$, choose $\mathbf{x}^{t+1} = \mathbf{x}^t$, otherwise choose $\mathbf{x}^{t+1} = \mathbf{y}^t$
5. Set $t = t + 1$ and go to step 2

2. For minimization problems

1. **Initialization**: $t = 0$, create a random vector $\mathbf{x}^t = \left(x_1^t, \ldots, x_n^t\right)^\top \in \mathbb{R}^n$
2. Test **termination condition**. If it holds, stop, continue otherwise
3. **Mutation**: draw $z_i \overset{\text{i.i.d.}}{\sim} N(0, \sigma)$ and create $y_i^t = x_i^t + z_i$, $i = 1, \ldots, n$
4. **Selection**[2]: if $f(\mathbf{x}^t) \leq f(\mathbf{y}^t)$, choose $\mathbf{x}^{t+1} = \mathbf{x}^t$, otherwise choose $\mathbf{x}^{t+1} = \mathbf{y}^t$
5. Set $t = t + 1$ and go to step 2

2. For minimization problems

Some Analysis

- Let us discuss how mutation works in this ES[3]
- Normal distribution is chosen to make small mutations more likely than large ones
- Mean zero is chosen also to make mutations neutral on average
- If each $z_i \sim N(0, \sigma_i)$ and all are independent, we can write the probability density function of a (column) vector $\mathbf{z}$ as follows:

$$p(z_1, \ldots, z_n) = \prod_{i=1}^{n} \phi(z_i) = \frac{1}{(2\pi)^{\frac{n}{2}} \prod_{i=1}^{n} \sigma_i} \exp\left(-\frac{1}{2}\sum_{i=1}^{n}\left(\frac{z_i}{\sigma_i}\right)^2\right)$$

- When all standard deviations are the same, we get

$$p(z_1, \ldots, z_n) = \frac{1}{(2\pi)^{\frac{n}{2}} \sigma^n} \exp\left(-\frac{\mathbf{z}^\top \mathbf{z}}{2\sigma^2}\right)$$

- The **length** of $\mathbf{z}$ is given by $S = \sqrt{\mathbf{z}^\top \mathbf{z}}$, and we know from statistics that $S^2 \sim \sigma\chi_n^2$ (χ^2-distribution with n degrees of freedom)
- It is not hard to derive the distribution of S as the square root of S^2 (it is the so called χ-distribution with n degrees of freedom)

3. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- Let us discuss how mutation works in this ES[3]
- Normal distribution is chosen to make small mutations more likely than large ones
- Mean zero is chosen also to make mutations neutral on average
- If each $z_i \sim N(0, \sigma_i)$ and all are independent, we can write the probability density function of a (column) vector $\mathbf{z}$ as follows:

$$p(z_1, \ldots, z_n) = \prod_{i=1}^{n} \phi(z_i) = \frac{1}{(2\pi)^{\frac{n}{2}} \prod_{i=1}^{n} \sigma_i} \exp\left(-\frac{1}{2} \sum_{i=1}^{n} \left(\frac{z_i}{\sigma_i} \right)^2 \right)$$

- When all standard deviations are the same, we get

$$p(z_1, \ldots, z_n) = \frac{1}{(2\pi)^{\frac{n}{2}} \sigma^n} \exp\left(-\frac{\mathbf{z}^\top \mathbf{z}}{2\sigma^2} \right)$$

- The **length** of $\mathbf{z}$ is given by $S = \sqrt{\mathbf{z}^\top \mathbf{z}}$, and we know from statistics that $S^2 \sim \sigma \chi_n^2$ (χ^2-distribution with n degrees of freedom)
- It is not hard to derive the distribution of S as the square root of S^2 (it is the so called χ-distribution with n degrees of freedom)

3. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- Let us discuss how mutation works in this ES[3]
- Normal distribution is chosen to make small mutations more likely than large ones
- Mean zero is chosen also to make mutations neutral on average
- If each $z_i \sim N(0, \sigma_i)$ and all are independent, we can write the probability density function of a (column) vector $\mathbf{z}$ as follows:

$$p(z_1, \ldots, z_n) = \prod_{i=1}^{n} \phi(z_i) = \frac{1}{(2\pi)^{\frac{n}{2}} \prod_{i=1}^{n} \sigma_i} \exp\left(-\frac{1}{2} \sum_{i=1}^{n} \left(\frac{z_i}{\sigma_i}\right)^2\right)$$

- When all standard deviations are the same, we get

$$p(z_1, \ldots, z_n) = \frac{1}{(2\pi)^{\frac{n}{2}} \sigma^n} \exp\left(-\frac{\mathbf{z}^\top \mathbf{z}}{2\sigma^2}\right)$$

- The **length** of $\mathbf{z}$ is given by $S = \sqrt{\mathbf{z}^\top \mathbf{z}}$, and we know from statistics that $S^2 \sim \sigma\chi_n^2$ (χ^2-distribution with n degrees of freedom)
- It is not hard to derive the distribution of S as the square root of S^2 (it is the so called χ-distribution with n degrees of freedom)

3. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- Let us discuss how mutation works in this ES[3]
- Normal distribution is chosen to make small mutations more likely than large ones
- Mean zero is chosen also to make mutations neutral on average
- If each $z_i \sim N(0, \sigma_i)$ and all are independent, we can write the probability density function of a (column) vector $\mathbf{z}$ as follows:

$$p(z_1, \ldots, z_n) = \prod_{i=1}^{n} \phi(z_i) = \frac{1}{(2\pi)^{\frac{n}{2}} \prod_{i=1}^{n} \sigma_i} \exp\left(-\frac{1}{2} \sum_{i=1}^{n} \left(\frac{z_i}{\sigma_i}\right)^2\right)$$

- When all standard deviations are the same, we get

$$p(z_1, \ldots, z_n) = \frac{1}{(2\pi)^{\frac{n}{2}} \sigma^n} \exp\left(-\frac{\mathbf{z}^\top \mathbf{z}}{2\sigma^2}\right)$$

- The **length** of $\mathbf{z}$ is given by $S = \sqrt{\mathbf{z}^\top \mathbf{z}}$, and we know from statistics that $S^2 \sim \sigma \chi_n^2$ (χ^2-distribution with n degrees of freedom)
- It is not hard to derive the distribution of S as the square root of S^2 (it is the so called χ-distribution with n degrees of freedom)

3. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- Let us discuss how mutation works in this ES[3]
- Normal distribution is chosen to make small mutations more likely than large ones
- Mean zero is chosen also to make mutations neutral on average
- If each $z_i \sim N(0, \sigma_i)$ and all are independent, we can write the probability density function of a (column) vector $\mathbf{z}$ as follows:

$$p(z_1, \dots, z_n) = \prod_{i=1}^{n} \phi(z_i) = \frac{1}{(2\pi)^{\frac{n}{2}} \prod_{i=1}^{n} \sigma_i} \exp\left(-\frac{1}{2} \sum_{i=1}^{n} \left(\frac{z_i}{\sigma_i} \right)^2 \right)$$

- When all standard deviations are the same, we get

$$p(z_1, \dots, z_n) = \frac{1}{(2\pi)^{\frac{n}{2}} \sigma^n} \exp\left(-\frac{\mathbf{z}^\top \mathbf{z}}{2\sigma^2} \right)$$

- The **length** of $\mathbf{z}$ is given by $S = \sqrt{\mathbf{z}^\top \mathbf{z}}$, and we know from statistics that $S^2 \sim \sigma \chi_n^2$ (χ^2-distribution with n degrees of freedom)
- It is not hard to derive the distribution of S as the square root of S^2 (it is the so called χ-distribution with n degrees of freedom)

3. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- Let us discuss how mutation works in this ES[3]
- Normal distribution is chosen to make small mutations more likely than large ones
- Mean zero is chosen also to make mutations neutral on average
- If each $z_i \sim N(0, \sigma_i)$ and all are independent, we can write the probability density function of a (column) vector $\mathbf{z}$ as follows:

$$p(z_1, \ldots, z_n) = \prod_{i=1}^{n} \phi(z_i) = \frac{1}{(2\pi)^{\frac{n}{2}} \prod_{i=1}^{n} \sigma_i} \exp\left(-\frac{1}{2}\sum_{i=1}^{n}\left(\frac{z_i}{\sigma_i}\right)^2\right)$$

- When all standard deviations are the same, we get

$$p(z_1, \ldots, z_n) = \frac{1}{(2\pi)^{\frac{n}{2}} \sigma^n} \exp\left(-\frac{\mathbf{z}^\top \mathbf{z}}{2\sigma^2}\right)$$

- The **length** of $\mathbf{z}$ is given by $S = \sqrt{\mathbf{z}^\top \mathbf{z}}$, and we know from statistics that $S^2 \sim \sigma\chi_n^2$ (χ^2-distribution with n degrees of freedom)
- It is not hard to derive the distribution of S as the square root of S^2 (it is the so called χ-distribution with n degrees of freedom)

3. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- Let us discuss how mutation works in this ES[3]
- Normal distribution is chosen to make small mutations more likely than large ones
- Mean zero is chosen also to make mutations neutral on average
- If each $z_i \sim N(0, \sigma_i)$ and all are independent, we can write the probability density function of a (column) vector $\mathbf{z}$ as follows:

$$p(z_1, \ldots, z_n) = \prod_{i=1}^{n} \phi(z_i) = \frac{1}{(2\pi)^{\frac{n}{2}} \prod_{i=1}^{n} \sigma_i} \exp\left(-\frac{1}{2} \sum_{i=1}^{n} \left(\frac{z_i}{\sigma_i} \right)^2 \right)$$

- When all standard deviations are the same, we get

$$p(z_1, \ldots, z_n) = \frac{1}{(2\pi)^{\frac{n}{2}} \sigma^n} \exp\left(-\frac{\mathbf{z}^\top \mathbf{z}}{2\sigma^2} \right)$$

- The **length** of $\mathbf{z}$ is given by $S = \sqrt{\mathbf{z}^\top \mathbf{z}}$, and we know from statistics that $S^2 \sim \sigma \chi_n^2$ (χ^2-distribution with n degrees of freedom)
- It is not hard to derive the distribution of S as the square root of S^2 (it is the so called χ-distribution with n degrees of freedom)

3. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- From the central limit theorem, as n becomes "large," we know that a random variable with χ_n^2 distribution has approximate normal distribution $N(n, 2n)$
- Using the so-called delta method, we can show that χ_n has approximate normal distribution $N(\sqrt{n}, 0.5)$
- Finally, we see that as n is "large," $\mathbb{E}[S] = \sigma\sqrt{n}$, $\mathrm{Var}(S) = \frac{\sigma^2}{2}$, and the coefficient of variation is

$$\frac{\sqrt{\mathrm{Var}(S)}}{\mathbb{E}[S]} = \frac{1}{\sqrt{2n}}$$

- In practice, this means that the most probable value for $\|\mathbf{z}\|$ at constant σ increases as $\sqrt{n}$
- And the relative width of variation decreases as $\frac{1}{\sqrt{2n}}$

- From the central limit theorem, as n becomes "large," we know that a random variable with χ_n^2 distribution has approximate normal distribution $N(n, 2n)$
- Using the so-called delta method, we can show that χ_n has approximate normal distribution $N(\sqrt{n}, 0.5)$
- Finally, we see that as n is "large," $\mathbb{E}[S] = \sigma\sqrt{n}$, $\mathrm{Var}(S) = \frac{\sigma^2}{2}$, and the coefficient of variation is

$$\frac{\sqrt{\mathrm{Var}(S)}}{\mathbb{E}[S]} = \frac{1}{\sqrt{2n}}$$

- In practice, this means that the most probable value for $\|\mathbf{z}\|$ at constant σ increases as $\sqrt{n}$
- And the relative width of variation decreases as $\frac{1}{\sqrt{2n}}$

- From the central limit theorem, as n becomes "large," we know that a random variable with χ_n^2 distribution has approximate normal distribution $N(n, 2n)$
- Using the so-called delta method, we can show that χ_n has approximate normal distribution $N(\sqrt{n}, 0.5)$
- Finally, we see that as n is "large," $\mathbb{E}[S] = \sigma\sqrt{n}$, $\mathrm{Var}(S) = \frac{\sigma^2}{2}$, and the coefficient of variation is
$$\frac{\sqrt{\mathrm{Var}(S)}}{\mathbb{E}[S]} = \frac{1}{\sqrt{2n}}$$
- In practice, this means that the most probable value for $\|\mathbf{z}\|$ at constant σ increases as $\sqrt{n}$
- And the relative width of variation decreases as $\frac{1}{\sqrt{2n}}$

- From the central limit theorem, as n becomes "large," we know that a random variable with χ_n^2 distribution has approximate normal distribution $N(n, 2n)$
- Using the so-called delta method, we can show that χ_n has approximate normal distribution $N(\sqrt{n}, 0.5)$
- Finally, we see that as n is "large," $\mathbb{E}[S] = \sigma\sqrt{n}$, $\mathrm{Var}(S) = \frac{\sigma^2}{2}$, and the coefficient of variation is

$$\frac{\sqrt{\mathrm{Var}(S)}}{\mathbb{E}[S]} = \frac{1}{\sqrt{2n}}$$

- In practice, this means that the most probable value for $\|\mathbf{z}\|$ at constant σ increases as $\sqrt{n}$
- And the relative width of variation decreases as $\frac{1}{\sqrt{2n}}$

- From the central limit theorem, as n becomes "large," we know that a random variable with χ_n^2 distribution has approximate normal distribution $N(n, 2n)$
- Using the so-called delta method, we can show that χ_n has approximate normal distribution $N(\sqrt{n}, 0.5)$
- Finally, we see that as n is "large," $\mathbb{E}[S] = \sigma\sqrt{n}$, $\operatorname{Var}(S) = \frac{\sigma^2}{2}$, and the coefficient of variation is

$$\frac{\sqrt{\operatorname{Var}(S)}}{\mathbb{E}[S]} = \frac{1}{\sqrt{2n}}$$

- In practice, this means that the most probable value for $\|\mathbf{z}\|$ at constant σ increases as $\sqrt{n}$
- And the relative width of variation decreases as $\frac{1}{\sqrt{2n}}$

- The geometric locus of equally likely changes in variation is an n-dimensional hyperellipsoid with the equation

$$\sum_{i=1}^{n} \left(\frac{z_i^t}{\sigma_i} \right)^2 = \text{const}$$

with the center at $\mathbf{x}^t$

- Illustration for all equal σs

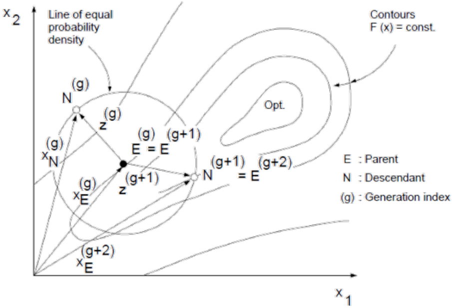
- The geometric locus of equally likely changes in variation is an n-dimensional hyperellipsoid with the equation
$$\sum_{i=1}^{n}\left(\frac{z_i^t}{\sigma_i}\right)^2 = \text{const}$$
with the center at $\mathbf{x}^t$
- Illustration for all equal σs

- The geometric locus of equally likely changes in variation is an n-dimensional hyperellipsoid with the equation

$$\sum_{i=1}^{n} \left(\frac{z_i^t}{\sigma_i} \right)^2 = \text{const}$$

with the center at $\mathbf{x}^t$

- Illustration for all equal σs



Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993), Fig. 5.1

- Evolutionary algorithms operate locally, by generating offspring in a certain neighborhood of their parents
- Therefore, evolution takes place only within a very narrow band of mutation step size, which is sometimes called **evolution window**
- The only parameter that influences ES performance is the mutation step size σ: it determines the extent, to which given gene alleles are perturbed by the mutation operator
- In original ES, this parameter was changed with time according to the 1/5 success rule[4]:

$$\sigma' = \begin{cases} \frac{\sigma}{c}\,, & p_s > 0.2 \\ \sigma \cdot c\,, & p_s < 0.2 \\ \sigma\,, & p_s = 0.2 \end{cases}$$

- Here p_s is the relative frequency of successful mutations over a number of generations
- c was typically chosen as 0.85
- In **modern ES**, however, this rule is not used
- Instead, mutation step size is *incorporated* into the genotype

4. Rechenberg, I.: Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog Verlag (1973)

Adaptation in Original ES

- Evolutionary algorithms operate locally, by generating offspring in a certain neighborhood of their parents
- Therefore, evolution takes place only within a very narrow band of mutation step size, which is sometimes called **evolution window**
- The only parameter that influences ES performance is the mutation step size σ: it determines the extent, to which given gene alleles are perturbed by the mutation operator
- In original ES, this parameter was changed with time according to the 1/5 success rule[4]:

$$\sigma' = \begin{cases} \frac{\sigma}{c}\,, & p_s > 0.2 \\ \sigma \cdot c\,, & p_s < 0.2 \\ \sigma\,, & p_s = 0.2 \end{cases}$$

- Here p_s is the relative frequency of successful mutations over a number of generations
- c was typically chosen as 0.85
- In **modern ES**, however, this rule is not used
- Instead, mutation step size is *incorporated* into the genotype

4. Rechenberg, I.: Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog Verlag (1973)

Adaptation in Original ES

- Evolutionary algorithms operate locally, by generating offspring in a certain neighborhood of their parents
- Therefore, evolution takes place only within a very narrow band of mutation step size, which is sometimes called **evolution window**
- The only parameter that influences ES performance is the mutation step size σ: it determines the extent, to which given gene alleles are perturbed by the mutation operator
- In original ES, this parameter was changed with time according to the 1/5 success rule[4]:

$$\sigma' = \begin{cases} \frac{\sigma}{c}\,, & p_s > 0.2 \\ \sigma \cdot c\,, & p_s < 0.2 \\ \sigma\,, & p_s = 0.2 \end{cases}$$

- Here p_s is the relative frequency of successful mutations over a number of generations
- c was typically chosen as 0.85
- In **modern ES**, however, this rule is not used
- Instead, mutation step size is *incorporated* into the genotype

4. Rechenberg, I.: Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog Verlag (1973)

Adaptation in Original ES

- Evolutionary algorithms operate locally, by generating offspring in a certain neighborhood of their parents
- Therefore, evolution takes place only within a very narrow band of mutation step size, which is sometimes called **evolution window**
- The only parameter that influences ES performance is the mutation step size σ: it determines the extent, to which given gene alleles are perturbed by the mutation operator
- In original ES, this parameter was changed with time according to the 1/5 success rule[4]:

$$\sigma' = \begin{cases} \frac{\sigma}{c}\,, & p_s > 0.2 \\ \sigma \cdot c\,, & p_s < 0.2 \\ \sigma\,, & p_s = 0.2 \end{cases}$$

- Here p_s is the relative frequency of successful mutations over a number of generations
- c was typically chosen as 0.85
- In **modern ES**, however, this rule is not used
- Instead, mutation step size is *incorporated* into the genotype

4. Rechenberg, I.: Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog Verlag (1973)

- Evolutionary algorithms operate locally, by generating offspring in a certain neighborhood of their parents
- Therefore, evolution takes place only within a very narrow band of mutation step size, which is sometimes called **evolution window**
- The only parameter that influences ES performance is the mutation step size σ: it determines the extent, to which given gene alleles are perturbed by the mutation operator
- In original ES, this parameter was changed with time according to the 1/5 success rule[4]:

$$\sigma' = \begin{cases} \frac{\sigma}{c}\,, & p_s > 0.2 \\ \sigma \cdot c\,, & p_s < 0.2 \\ \sigma\,, & p_s = 0.2 \end{cases}$$

- Here p_s is the relative frequency of successful mutations over a number of generations
- c was typically chosen as 0.85
- In **modern ES**, however, this rule is not used
- Instead, mutation step size is *incorporated* into the genotype

4. Rechenberg, I.: Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog Verlag (1973)

- Evolutionary algorithms operate locally, by generating offspring in a certain neighborhood of their parents
- Therefore, evolution takes place only within a very narrow band of mutation step size, which is sometimes called **evolution window**
- The only parameter that influences ES performance is the mutation step size σ: it determines the extent, to which given gene alleles are perturbed by the mutation operator
- In original ES, this parameter was changed with time according to the 1/5 success rule[4]:

$$\sigma' = \begin{cases} \frac{\sigma}{c}\,, & p_s > 0.2 \\ \sigma \cdot c\,, & p_s < 0.2 \\ \sigma\,, & p_s = 0.2 \end{cases}$$

- Here p_s is the relative frequency of successful mutations over a number of generations
- c was typically chosen as 0.85
- In **modern ES**, however, this rule is not used
- Instead, mutation step size is *incorporated* into the genotype

4. Rechenberg, I.: Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog Verlag (1973)

- Evolutionary algorithms operate locally, by generating offspring in a certain neighborhood of their parents
- Therefore, evolution takes place only within a very narrow band of mutation step size, which is sometimes called **evolution window**
- The only parameter that influences ES performance is the mutation step size σ: it determines the extent, to which given gene alleles are perturbed by the mutation operator
- In original ES, this parameter was changed with time according to the 1/5 success rule[4]:

$$\sigma' = \begin{cases} \frac{\sigma}{c}\,, & p_s > 0.2 \\ \sigma \cdot c\,, & p_s < 0.2 \\ \sigma\,, & p_s = 0.2 \end{cases}$$

- Here p_s is the relative frequency of successful mutations over a number of generations
- c was typically chosen as 0.85
- In **modern ES**, however, this rule is not used
- Instead, mutation step size is *incorporated* into the genotype

4. Rechenberg, I.: Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog Verlag (1973)

- Evolutionary algorithms operate locally, by generating offspring in a certain neighborhood of their parents
- Therefore, evolution takes place only within a very narrow band of mutation step size, which is sometimes called **evolution window**
- The only parameter that influences ES performance is the mutation step size σ: it determines the extent, to which given gene alleles are perturbed by the mutation operator
- In original ES, this parameter was changed with time according to the 1/5 success rule[4]:

$$\sigma' = \begin{cases} \frac{\sigma}{c}\,, & p_s > 0.2 \\ \sigma \cdot c\,, & p_s < 0.2 \\ \sigma\,, & p_s = 0.2 \end{cases}$$

- Here p_s is the relative frequency of successful mutations over a number of generations
- c was typically chosen as 0.85
- In **modern ES**, however, this rule is not used
- Instead, mutation step size is *incorporated* into the genotype

4. Rechenberg, I.: Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog Verlag (1973)

- In general, an individual in ES is of the form

$$(x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n)$$

- Here $x_1, \ldots, x_n$ are **object variables**
- Other values are **strategy parameters**
- σ_i determine the mutation step size in each direction
- This representation enables **self-adaptation** within the algorithm

- In general, an individual in ES is of the form

$$(x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n)$$

- Here $x_1, \ldots, x_n$ are **object variables**
- Other values are **strategy parameters**
- σ_i determine the mutation step size in each direction
- This representation enables **self-adaptation** within the algorithm

- In general, an individual in ES is of the form

$$(x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n)$$

- Here $x_1, \ldots, x_n$ are **object variables**
- Other values are **strategy parameters**
- σ_i determine the mutation step size in each direction
- This representation enables **self-adaptation** within the algorithm

Representation in Modern ES

- In general, an individual in ES is of the form

$$(x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n)$$

- Here $x_1, \ldots, x_n$ are **object variables**
- Other values are **strategy parameters**
- σ_i determine the mutation step size in each direction
- This representation enables **self-adaptation** within the algorithm

Representation in Modern ES

- In general, an individual in ES is of the form

$$(x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n)$$

- Here $x_1, \ldots, x_n$ are **object variables**
- Other values are **strategy parameters**
- σ_i determine the mutation step size in each direction
- This representation enables **self-adaptation** within the algorithm

1. **Initialization**: $t = 0$, create initial population P_0 of size μ, and calculate fitness values for all individuals
2. Test **termination condition**. If it holds, stop, continue otherwise
3. Creation of P_{t+1}:
   - For $l = 1, \ldots, \lambda$ do:

     Calculate fitness of each child
     Selection:
4. Set $t = t + 1$ and go to step 2

Modern ES

1. **Initialization**: $t = 0$, create initial population P_0 of size μ, and calculate fitness values for all individuals
2. Test **termination condition**. If it holds, stop, continue otherwise
3. Creation of P_{t+1}:
   - For $l = 1, \ldots, \lambda$ do:
4. Set $t = t + 1$ and go to step 2

Modern ES

1. **Initialization**: $t = 0$, create initial population P_0 of size μ, and calculate fitness values for all individuals
2. Test **termination condition**. If it holds, stop, continue otherwise
3. Creation of P_{t+1}:
    - For $l = 1, \dots, \lambda$ do:
        - **Recombination**: Pick ρ individuals and recombine them (with probability 1)
        - **Mutation**: mutate each offspring (with probability 1)
    - Calculate fitness of each child
    - **Selection:**
4. Set $t = t + 1$ and go to step 2

1. **Initialization**: $t = 0$, create initial population P_0 of size μ, and calculate fitness values for all individuals
2. Test **termination condition**. If it holds, stop, continue otherwise
3. Creation of P_{t+1}:
   - For $l = 1, \ldots, \lambda$ do:
     - **Recombination**: Pick ρ individuals and recombine them (with probability 1)
     - **Mutation**: mutate each offspring (with probability 1)
   - Calculate fitness of each child
   - **Selection:**
4. Set $t = t + 1$ and go to step 2

1. **Initialization**: $t = 0$, create initial population P_0 of size μ, and calculate fitness values for all individuals
2. Test **termination condition**. If it holds, stop, continue otherwise
3. Creation of P_{t+1}:
   - For $l = 1, \ldots, \lambda$ do:
     - **Recombination**: Pick ρ individuals and recombine them (with probability 1)
     - **Mutation**: mutate each offspring (with probability 1)
   - Calculate fitness of each child
   - **Selection:**
4. Set $t = t + 1$ and go to step 2

Modern ES

1. **Initialization**: $t = 0$, create initial population P_0 of size μ, and calculate fitness values for all individuals
2. Test **termination condition**. If it holds, stop, continue otherwise
3. Creation of P_{t+1}:
    - For $l = 1, \ldots, \lambda$ do:
        - **Recombination**: Pick ρ individuals and recombine them (with probability 1)
        - **Mutation**: mutate each offspring (with probability 1)
    - Calculate fitness of each child
    - **Selection:**
        - If rule of type (μ, λ), select μ fittest individuals among λ children
          If rule of type $(\mu + \lambda)$, select μ fittest individuals among μ parents and λ children
4. Set $t = t + 1$ and go to step 2

1. **Initialization**: $t = 0$, create initial population P_0 of size μ, and calculate fitness values for all individuals
2. Test **termination condition**. If it holds, stop, continue otherwise
3. Creation of P_{t+1}:
    - For $l = 1, \ldots, \lambda$ do:
        - **Recombination**: Pick ρ individuals and recombine them (with probability 1)
        - **Mutation**: mutate each offspring (with probability 1)
    - Calculate fitness of each child
    - **Selection:**
        - If it is of type (μ, λ), select μ fittest individuals among λ children
        - If it is of type $(\mu + \lambda)$, select μ fittest individuals among μ parents and λ children
4. Set $t = t + 1$ and go to step 2

1. **Initialization**: $t = 0$, create initial population P_0 of size μ, and calculate fitness values for all individuals
2. Test **termination condition**. If it holds, stop, continue otherwise
3. Creation of P_{t+1}:
   - For $l = 1, \ldots, \lambda$ do:
     - **Recombination**: Pick ρ individuals and recombine them (with probability 1)
     - **Mutation**: mutate each offspring (with probability 1)
   - Calculate fitness of each child
   - **Selection:**
     - If it is of type (μ, λ), select μ fittest individuals **among λ children**
     - If it is of type $(\mu + \lambda)$, select μ fittest individuals **among μ parents and λ children**
4. Set $t = t + 1$ and go to step 2

Modern ES

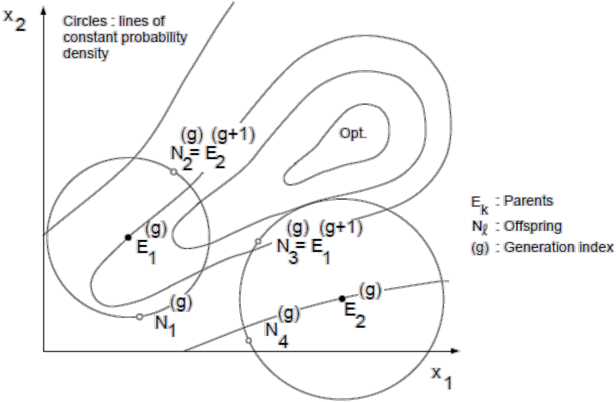
1. **Initialization**: $t = 0$, create initial population P_0 of size μ, and calculate fitness values for all individuals
2. Test **termination condition**. If it holds, stop, continue otherwise
3. Creation of P_{t+1}:
   - For $l = 1, \ldots, \lambda$ do:
     - **Recombination**: Pick ρ individuals and recombine them (with probability 1)
     - **Mutation**: mutate each offspring (with probability 1)
   - Calculate fitness of each child
   - **Selection:**
     - If it is of type (μ, λ), select μ fittest individuals **among $\boldsymbol{\lambda}$ children**
     - If it is of type $(\mu + \lambda)$, select μ fittest individuals **among $\boldsymbol{\mu}$ parents and $\boldsymbol{\lambda}$ children**
4. Set $t = t + 1$ and go to step 2

1. **Initialization**: $t = 0$, create initial population P_0 of size μ, and calculate fitness values for all individuals
2. Test **termination condition**. If it holds, stop, continue otherwise
3. Creation of P_{t+1}:
   - For $l = 1, \ldots, \lambda$ do:
     - **Recombination**: Pick ρ individuals and recombine them (with probability 1)
     - **Mutation**: mutate each offspring (with probability 1)
   - Calculate fitness of each child
   - **Selection:**
     - If it is of type (μ, λ), select μ fittest individuals **among $\boldsymbol{\lambda}$ children**
     - If it is of type $(\mu + \lambda)$, select μ fittest individuals **among $\boldsymbol{\mu}$ parents and $\boldsymbol{\lambda}$ children**
4. Set $t = t + 1$ and go to step 2

Modern ES

1. **Initialization**: $t = 0$, create initial population P_0 of size μ, and calculate fitness values for all individuals
2. Test **termination condition**. If it holds, stop, continue otherwise
3. Creation of P_{t+1}:
    - For $l = 1, \ldots, \lambda$ do:
        - **Recombination**: Pick ρ individuals and recombine them (with probability 1)
        - **Mutation**: mutate each offspring (with probability 1)
    - Calculate fitness of each child
    - **Selection:**
        - If it is of type (μ, λ), select μ fittest individuals **among $\boldsymbol{\lambda}$ children**
        - If it is of type $(\mu + \lambda)$, select μ fittest individuals **among $\boldsymbol{\mu}$ parents and $\boldsymbol{\lambda}$ children**
4. Set $t = t + 1$ and go to step 2

Illustration for a $(2, 4)$ ES



Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993), Fig. 5.4

- **First**, the strategy parameters are mutated
- We get $\boldsymbol{\sigma}' = (\sigma'_1, \ldots, \sigma'_n)$
- **Then**, object variables are changed:

$$\mathbf{x}' = \mathbf{x} + \Delta\mathbf{x}$$

- Here the probability density function of $\Delta\mathbf{x}$ depends on strategy parameters: $p(\Delta\mathbf{x}) = p(\Delta\mathbf{x}, \boldsymbol{\sigma}')$
- This sequence is **obligatory**, otherwise we will not have self-adaptation

Mutations in Modern ES

- **First**, the strategy parameters are mutated
- We get $\boldsymbol{\sigma}' = (\sigma'_1, \ldots, \sigma'_n)$
- **Then**, object variables are changed:

$$\mathbf{x}' = \mathbf{x} + \Delta\mathbf{x}$$

- Here the probability density function of $\Delta\mathbf{x}$ depends on strategy parameters: $p(\Delta\mathbf{x}) = p(\Delta\mathbf{x}, \boldsymbol{\sigma}')$
- This sequence is **obligatory**, otherwise we will not have self-adaptation

- **First**, the strategy parameters are mutated
- We get $\boldsymbol{\sigma}' = (\sigma'_1, \ldots, \sigma'_n)$
- **Then**, object variables are changed:

$$\mathbf{x}' = \mathbf{x} + \Delta\mathbf{x}$$

- Here the probability density function of $\Delta\mathbf{x}$ depends on strategy parameters: $p(\Delta\mathbf{x}) = p(\Delta\mathbf{x}, \boldsymbol{\sigma}')$
- This sequence is **obligatory**, otherwise we will not have self-adaptation

- **First**, the strategy parameters are mutated
- We get $\boldsymbol{\sigma}' = (\sigma'_1, \ldots, \sigma'_n)$
- **Then**, object variables are changed:

$$\mathbf{x}' = \mathbf{x} + \Delta\mathbf{x}$$

- Here the probability density function of $\Delta\mathbf{x}$ depends on strategy parameters: $p(\Delta\mathbf{x}) = p(\Delta\mathbf{x}, \boldsymbol{\sigma}')$
- This sequence is **obligatory**, otherwise we will not have self-adaptation

Mutations in Modern ES

- **First**, the strategy parameters are mutated
- We get $\boldsymbol{\sigma}' = (\sigma_1', \ldots, \sigma_n')$
- **Then**, object variables are changed:

$$\mathbf{x}' = \mathbf{x} + \Delta\mathbf{x}$$

- Here the probability density function of $\Delta\mathbf{x}$ depends on strategy parameters: $p(\Delta\mathbf{x}) = p(\Delta\mathbf{x}, \boldsymbol{\sigma}')$
- This sequence is **obligatory**, otherwise we will not have self-adaptation

- Typically, the strategy parameters $\sigma_1, \ldots, \sigma_n$ are mutated according to the following general scheme[5]:

$$\sigma_i' = \sigma_i \cdot W$$

- Random variable W should possess the following properties
- The *median* should be 1 (no deterministic drift)
- *Increase* and *decrease* in step length should occur with the same frequency
- Small changes should occur more often than large ones
- All these requirements can be satisfied by the **log-normal** distribution

5. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- Typically, the strategy parameters $\sigma_1, \ldots, \sigma_n$ are mutated according to the following general scheme[5]:

$$\sigma_i' = \sigma_i \cdot W$$

- Random variable W should possess the following properties
- The *median* should be 1 (no deterministic drift)
- *Increase* and *decrease* in step length should occur with the same frequency
- Small changes should occur more often than large ones
- All these requirements can be satisfied by the **log-normal** distribution

5. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- Typically, the strategy parameters $\sigma_1, \ldots, \sigma_n$ are mutated according to the following general scheme[5]:
$$\sigma_i' = \sigma_i \cdot W$$
- Random variable W should possess the following properties
- The *median* should be 1 (no deterministic drift)
- *Increase* and *decrease* in step length should occur with the same frequency
- Small changes should occur more often than large ones
- All these requirements can be satisfied by the **log-normal** distribution

5. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- Typically, the strategy parameters $\sigma_1, \ldots, \sigma_n$ are mutated according to the following general scheme[5]:
$$\sigma_i' = \sigma_i \cdot W$$
- Random variable W should possess the following properties
- The *median* should be 1 (no deterministic drift)
- *Increase* and *decrease* in step length should occur with the same frequency
- Small changes should occur more often than large ones
- All these requirements can be satisfied by the **log-normal** distribution

5. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- Typically, the strategy parameters $\sigma_1, \ldots, \sigma_n$ are mutated according to the following general scheme[5]:
$$\sigma_i' = \sigma_i \cdot W$$
- Random variable W should possess the following properties
- The *median* should be 1 (no deterministic drift)
- *Increase* and *decrease* in step length should occur with the same frequency
- Small changes should occur more often than large ones
- All these requirements can be satisfied by the **log-normal** distribution

5. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- Typically, the strategy parameters $\sigma_1, \ldots, \sigma_n$ are mutated according to the following general scheme[5]:

$$\sigma_i' = \sigma_i \cdot W$$

- Random variable W should possess the following properties
- The *median* should be 1 (no deterministic drift)
- *Increase* and *decrease* in step length should occur with the same frequency
- Small changes should occur more often than large ones
- All these requirements can be satisfied by the **log-normal** distribution

5. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

Uncorrelated Mutation With One Step Size

- The same distribution is used for each object variable: $\sigma_1 = \ldots = \sigma_n = \sigma$
- This σ is mutated as $\sigma' = \sigma \cdot W$, where $\ln W \sim N(0, \tau^2)$
- Here τ is the learning rate, which is proportional to $\frac{1}{\sqrt{n}}$
- The constant of proportionality a depends on λ/μ (basically selection pressure)
- For a $(10, 100)$ ES, it should be set to 1[6]
- For higher λ, we can set a sublinearly according to $\lambda \sim \sqrt{a}e^{a}$
- To prevent too small mutations, we set σ' to ε_0 if $\sigma' < \varepsilon_0$, for some user-specified constant ε_0
- Finally, the object variables are then mutated as

$$\mathbf{x}' = \mathbf{x} + \mathbf{z} \; ,$$

where $\mathbf{z}$ is the multivariate normal distribution with zero mean and covariance matrix $(\sigma')^2 \cdot \mathbf{I}$

6. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- The same distribution is used for each object variable: $\sigma_1 = \ldots = \sigma_n = \sigma$
- This σ is mutated as $\sigma' = \sigma \cdot W$, where $\ln W \sim N(0, \tau^2)$
- Here τ is the learning rate, which is proportional to $\frac{1}{\sqrt{n}}$
- The constant of proportionality a depends on λ/μ (basically selection pressure)
- For a $(10, 100)$ ES, it should be set to 1[6]
- For higher λ, we can set a sublinearly according to $\lambda \sim \sqrt{a}e^{a}$
- To prevent too small mutations, we set σ' to ε_0 if $\sigma' < \varepsilon_0$, for some user-specified constant ε_0
- Finally, the object variables are then mutated as

$$\mathbf{x}' = \mathbf{x} + \mathbf{z} \ ,$$

where $\mathbf{z}$ is the multivariate normal distribution with zero mean and covariance matrix $(\sigma')^2 \cdot \mathbf{I}$

6. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- The same distribution is used for each object variable: $\sigma_1 = \ldots = \sigma_n = \sigma$
- This σ is mutated as $\sigma' = \sigma \cdot W$, where $\ln W \sim N(0, \tau^2)$
- Here τ is the learning rate, which is proportional to $\frac{1}{\sqrt{n}}$
- The constant of proportionality a depends on λ/μ (basically selection pressure)
- For a $(10, 100)$ ES, it should be set to 1[6]
- For higher λ, we can set a sublinearly according to $\lambda \sim \sqrt{a}e^a$
- To prevent too small mutations, we set σ' to ε_0 if $\sigma' < \varepsilon_0$, for some user-specified constant ε_0
- Finally, the object variables are then mutated as

$$\mathbf{x}' = \mathbf{x} + \mathbf{z} \ ,$$

where $\mathbf{z}$ is the multivariate normal distribution with zero mean and covariance matrix $(\sigma')^2 \cdot \mathbf{I}$

6. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

- The same distribution is used for each object variable: $\sigma_1 = \ldots = \sigma_n = \sigma$
- This σ is mutated as $\sigma' = \sigma \cdot W$, where $\ln W \sim N(0, \tau^2)$
- Here τ is the learning rate, which is proportional to $\frac{1}{\sqrt{n}}$
- The constant of proportionality a depends on λ/μ (basically selection pressure)
- For a $(10, 100)$ ES, it should be set to 1[6]
- For higher λ, we can set a sublinearly according to $\lambda \sim \sqrt{a}e^a$
- To prevent too small mutations, we set σ' to ε_0 if $\sigma' < \varepsilon_0$, for some user-specified constant ε_0
- Finally, the object variables are then mutated as

$$\mathbf{x}' = \mathbf{x} + \mathbf{z} \ ,$$

where $\mathbf{z}$ is the multivariate normal distribution with zero mean and covariance matrix $(\sigma')^2 \cdot \mathbf{I}$

6. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

Uncorrelated Mutation With One Step Size

- The same distribution is used for each object variable: $\sigma_1 = \ldots = \sigma_n = \sigma$
- This σ is mutated as $\sigma' = \sigma \cdot W$, where $\ln W \sim N(0, \tau^2)$
- Here τ is the learning rate, which is proportional to $\frac{1}{\sqrt{n}}$
- The constant of proportionality a depends on λ/μ (basically selection pressure)
- For a $(10, 100)$ ES, it should be set to 1[6]
- For higher λ, we can set a sublinearly according to $\lambda \sim \sqrt{a}e^a$
- To prevent too small mutations, we set σ' to ε_0 if $\sigma' < \varepsilon_0$, for some user-specified constant ε_0
- Finally, the object variables are then mutated as

$$\mathbf{x}' = \mathbf{x} + \mathbf{z} \,,$$

  where $\mathbf{z}$ is the multivariate normal distribution with zero mean and covariance matrix $(\sigma')^2 \cdot \mathbf{I}$

6. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

Uncorrelated Mutation With One Step Size

- The same distribution is used for each object variable: $\sigma_1 = \ldots = \sigma_n = \sigma$
- This σ is mutated as $\sigma' = \sigma \cdot W$, where $\ln W \sim N(0, \tau^2)$
- Here τ is the learning rate, which is proportional to $\frac{1}{\sqrt{n}}$
- The constant of proportionality a depends on λ/μ (basically selection pressure)
- For a $(10, 100)$ ES, it should be set to 1[6]
- For higher λ, we can set a sublinearly according to $\lambda \sim \sqrt{a}e^a$
- To prevent too small mutations, we set σ' to ε_0 if $\sigma' < \varepsilon_0$, for some user-specified constant ε_0
- Finally, the object variables are then mutated as

$$\mathbf{x}' = \mathbf{x} + \mathbf{z} ,$$

where $\mathbf{z}$ is the multivariate normal distribution with zero mean and covariance matrix $(\sigma')^2 \cdot \mathbf{I}$

6. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

Uncorrelated Mutation With One Step Size

- The same distribution is used for each object variable: $\sigma_1 = \ldots = \sigma_n = \sigma$
- This σ is mutated as $\sigma' = \sigma \cdot W$, where $\ln W \sim N(0, \tau^2)$
- Here τ is the learning rate, which is proportional to $\frac{1}{\sqrt{n}}$
- The constant of proportionality a depends on λ/μ (basically selection pressure)
- For a $(10, 100)$ ES, it should be set to 1[6]
- For higher λ, we can set a sublinearly according to $\lambda \sim \sqrt{a}e^a$
- To prevent too small mutations, we set σ' to ε_0 if $\sigma' < \varepsilon_0$, for some user-specified constant ε_0
- Finally, the object variables are then mutated as

$$\mathbf{x}' = \mathbf{x} + \mathbf{z}\ ,$$

where $\mathbf{z}$ is the multivariate normal distribution with zero mean and covariance matrix $(\sigma')^2 \cdot \mathbf{I}$

6. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

Uncorrelated Mutation With One Step Size

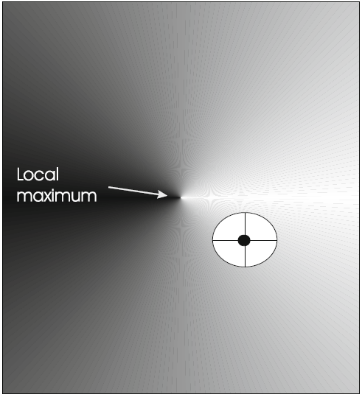
- The same distribution is used for each object variable: $\sigma_1 = \ldots = \sigma_n = \sigma$
- This σ is mutated as $\sigma' = \sigma \cdot W$, where $\ln W \sim N(0, \tau^2)$
- Here τ is the learning rate, which is proportional to $\frac{1}{\sqrt{n}}$
- The constant of proportionality a depends on λ/μ (basically selection pressure)
- For a $(10, 100)$ ES, it should be set to 1[6]
- For higher λ, we can set a sublinearly according to $\lambda \sim \sqrt{a}e^a$
- To prevent too small mutations, we set σ' to ε_0 if $\sigma' < \varepsilon_0$, for some user-specified constant ε_0
- Finally, the object variables are then mutated as

$$\mathbf{x}' = \mathbf{x} + \mathbf{z} \ ,$$

  where $\mathbf{z}$ is the multivariate normal distribution with zero mean and covariance matrix $(\sigma')^2 \cdot \mathbf{I}$

6. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, Inc., New York (1993)

Illustration



Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer-Verlag, Berlin, Heidelberg (2015), Fig. 4.4

- The motivation is to treat dimensions differently
- Each σ_i is mutated as
$$\sigma_i' = \sigma_i(W + V_i) \ ,$$
where $\ln W \sim N\left(0, (\tau')^2\right)$, $\ln V_i \sim N(0, \tau^2)$
- Here τ' is proportional to $\frac{1}{\sqrt{2n}}$, and τ is proportional to $\frac{1}{\sqrt{2\sqrt{n}}}$
- The first part allows for an overall change of the mutability (the preservation of all degrees of freedom)
- The coordinate-specific second part provides the flexibility to use different mutation strategies in different directions
- To prevent too small mutations, we again set σ_i' to ε_0 if $\sigma_i' < \varepsilon_0$, for some user-specified constant ε_0
- Finally, the object variables are then mutated as
$$\mathbf{x}' = \mathbf{x} + \mathbf{z} \ ,$$
where $\mathbf{z}$ is the multivariate normal distribution with zero mean and covariance matrix
$$\begin{pmatrix} \sigma_1^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_n^2 \end{pmatrix}$$

Uncorrelated Mutation With n Step Sizes

- The motivation is to treat dimensions differently
- Each σ_i is mutated as

  $$\sigma_i' = \sigma_i(W + V_i) \ ,$$

  where $\ln W \sim N\left(0, (\tau')^2\right)$, $\ln V_i \sim N(0, \tau^2)$
- Here τ' is proportional to $\frac{1}{\sqrt{2n}}$, and τ is proportional to $\frac{1}{\sqrt{2\sqrt{n}}}$
- The first part allows for an overall change of the mutability (the preservation of all degrees of freedom)
- The coordinate-specific second part provides the flexibility to use different mutation strategies in different directions
- To prevent too small mutations, we again set σ_i' to ε_0 if $\sigma_i' < \varepsilon_0$, for some user-specified constant ε_0
- Finally, the object variables are then mutated as

  $$\mathbf{x}' = \mathbf{x} + \mathbf{z} \ ,$$

  where $\mathbf{z}$ is the multivariate normal distribution with zero mean and covariance matrix

  $$\begin{pmatrix} \sigma_1^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_n^2 \end{pmatrix}$$

Uncorrelated Mutation With n Step Sizes

- The motivation is to treat dimensions differently
- Each σ_i is mutated as

$$\sigma_i' = \sigma_i (W + V_i) \ ,$$

where $\ln W \sim N\left(0, (\tau')^2\right)$, $\ln V_i \sim N(0, \tau^2)$

- Here τ' is proportional to $\frac{1}{\sqrt{2n}}$, and τ is proportional to $\frac{1}{\sqrt{2\sqrt{n}}}$
- The first part allows for an overall change of the mutability (the preservation of all degrees of freedom)
- The coordinate-specific second part provides the flexibility to use different mutation strategies in different directions
- To prevent too small mutations, we again set σ_i' to ε_0 if $\sigma_i' < \varepsilon_0$, for some user-specified constant ε_0
- Finally, the object variables are then mutated as

$$\mathbf{x}' = \mathbf{x} + \mathbf{z} \ ,$$

where $\mathbf{z}$ is the multivariate normal distribution with zero mean and covariance matrix

$$\begin{pmatrix} \sigma_1^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_n^2 \end{pmatrix}$$

Uncorrelated Mutation With n Step Sizes

- The motivation is to treat dimensions differently
- Each σ_i is mutated as

$$\sigma_i' = \sigma_i (W + V_i) ,$$

where $\ln W \sim N\left(0, (\tau')^2\right)$, $\ln V_i \sim N(0, \tau^2)$

- Here τ' is proportional to $\frac{1}{\sqrt{2n}}$, and τ is proportional to $\frac{1}{\sqrt{2\sqrt{n}}}$
- The first part allows for an overall change of the mutability (the preservation of all degrees of freedom)
- The coordinate-specific second part provides the flexibility to use different mutation strategies in different directions
- To prevent too small mutations, we again set σ_i' to ε_0 if $\sigma_i' < \varepsilon_0$, for some user-specified constant ε_0
- Finally, the object variables are then mutated as

$$\mathbf{x}' = \mathbf{x} + \mathbf{z} ,$$

where $\mathbf{z}$ is the multivariate normal distribution with zero mean and covariance matrix

$$\begin{pmatrix} \sigma_1^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_n^2 \end{pmatrix}$$

Uncorrelated Mutation With n Step Sizes

- The motivation is to treat dimensions differently
- Each σ_i is mutated as

$$\sigma_i' = \sigma_i(W + V_i) \ ,$$

  where $\ln W \sim N\left(0, (\tau')^2\right)$, $\ln V_i \sim N(0, \tau^2)$
- Here τ' is proportional to $\frac{1}{\sqrt{2n}}$, and τ is proportional to $\frac{1}{\sqrt{2\sqrt{n}}}$
- The first part allows for an overall change of the mutability (the preservation of all degrees of freedom)
- The coordinate-specific second part provides the flexibility to use different mutation strategies in different directions
- To prevent too small mutations, we again set σ_i' to ε_0 if $\sigma_i' < \varepsilon_0$, for some user-specified constant ε_0
- Finally, the object variables are then mutated as

$$\mathbf{x}' = \mathbf{x} + \mathbf{z} \ ,$$

  where $\mathbf{z}$ is the multivariate normal distribution with zero mean and covariance matrix

$$\begin{pmatrix} \sigma_1^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_n^2 \end{pmatrix}$$

Uncorrelated Mutation With n Step Sizes

- The motivation is to treat dimensions differently
- Each σ_i is mutated as

$$\sigma_i' = \sigma_i(W + V_i)\ ,$$

where $\ln W \sim N\left(0, (\tau')^2\right)$, $\ln V_i \sim N(0, \tau^2)$

- Here τ' is proportional to $\frac{1}{\sqrt{2n}}$, and τ is proportional to $\frac{1}{\sqrt{2\sqrt{n}}}$
- The first part allows for an overall change of the mutability (the preservation of all degrees of freedom)
- The coordinate-specific second part provides the flexibility to use different mutation strategies in different directions
- To prevent too small mutations, we again set σ_i' to ε_0 if $\sigma_i' < \varepsilon_0$, for some user-specified constant ε_0
- Finally, the object variables are then mutated as

$$\mathbf{x}' = \mathbf{x} + \mathbf{z}\ ,$$

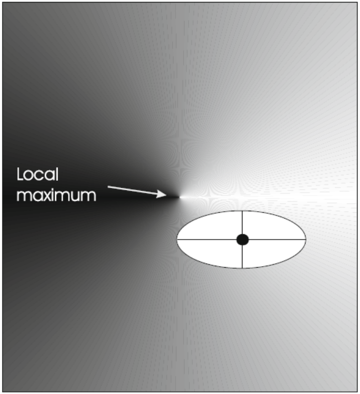
where $\mathbf{z}$ is the multivariate normal distribution with zero mean and covariance matrix

$$\begin{pmatrix} \sigma_1^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_n^2 \end{pmatrix}$$

Uncorrelated Mutation With n Step Sizes

- The motivation is to treat dimensions differently
- Each σ_i is mutated as
$$\sigma_i' = \sigma_i(W + V_i) \ ,$$
where $\ln W \sim N\left(0, (\tau')^2\right)$, $\ln V_i \sim N(0, \tau^2)$
- Here τ' is proportional to $\frac{1}{\sqrt{2n}}$, and τ is proportional to $\frac{1}{\sqrt{2\sqrt{n}}}$
- The first part allows for an overall change of the mutability (the preservation of all degrees of freedom)
- The coordinate-specific second part provides the flexibility to use different mutation strategies in different directions
- To prevent too small mutations, we again set σ_i' to ε_0 if $\sigma_i' < \varepsilon_0$, for some user-specified constant ε_0
- Finally, the object variables are then mutated as
$$\mathbf{x}' = \mathbf{x} + \mathbf{z} \ ,$$
where $\mathbf{z}$ is the multivariate normal distribution with zero mean and covariance matrix
$$\begin{pmatrix} \sigma_1^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_n^2 \end{pmatrix}$$

Illustration



Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer-Verlag, Berlin, Heidelberg (2015), Fig. 4.5

Recombination in Modern ES

- In ES, recombination can be **panmictic** ($\rho = \mu$) or **sexual** ($\rho = 2$)
- In the latter case, all individuals can be selected as parents with uniform probabilities
- Recombination can be:
  - **Discrete (dominant)**: coordinate-wise random selection from the ρ corresponding coordinate values of the parent family
  - **Intermediate**: arithmetic mean of the ρ parents (in practice, used more often)

Recombination in Modern ES

- In ES, recombination can be **panmictic** ($\rho = \mu$) or **sexual** ($\rho = 2$)
- In the latter case, all individuals can be selected as parents with uniform probabilities
- Recombination can be:
  - **Discrete (dominant):** coordinate-wise random selection from the ρ corresponding coordinate values of the parent family
  - **Intermediate:** arithmetic mean of the ρ parents (in practice, used more often)

Recombination in Modern ES

- In ES, recombination can be **panmictic** ($\rho = \mu$) or **sexual** ($\rho = 2$)
- In the latter case, all individuals can be selected as parents with uniform probabilities
- Recombination can be:
  - **Discrete (dominant)**: coordinate-wise random selection from the ρ corresponding coordinate values of the parent family
  - **Intermediate**: arithmetic mean of the ρ parents (in practice, used more often)

Recombination in Modern ES

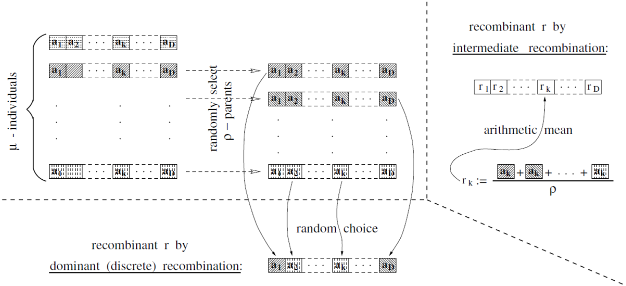
- In ES, recombination can be **panmictic** ($\rho = \mu$) or **sexual** ($\rho = 2$)
- In the latter case, all individuals can be selected as parents with uniform probabilities
- Recombination can be:
  - **Discrete (dominant)**: coordinate-wise random selection from the ρ corresponding coordinate values of the parent family
  - **Intermediate**: arithmetic mean of the ρ parents (in practice, used more often)

Recombination in Modern ES

- In ES, recombination can be **panmictic** ($\rho = \mu$) or **sexual** ($\rho = 2$)
- In the latter case, all individuals can be selected as parents with uniform probabilities
- Recombination can be:
  - **Discrete (dominant)**: coordinate-wise random selection from the ρ corresponding coordinate values of the parent family
  - **Intermediate**: arithmetic mean of the ρ parents (in practice, used more often)

Recombination in Modern ES

- In ES, recombination can be **panmictic** ($\rho = \mu$) or **sexual** ($\rho = 2$)
- In the latter case, all individuals can be selected as parents with uniform probabilities
- Recombination can be:
  - **Discrete (dominant)**: coordinate-wise random selection from the ρ corresponding coordinate values of the parent family
  - **Intermediate**: arithmetic mean of the ρ parents (in practice, used more often)



Beyer, H.-G., Schwefel, H.-P.: Evolution Strategies. A Comprehensive Introduction. Natural Computing, 1, 3–52 (2002), Fig. 6

- The best μ individuals for the next generation are always chosen **deterministically**
- In the (μ, λ) selection, μ fittest individuals are selected among λ children
- Since all the parents are discarded, it is possible to leave local optima
- The suggested ratio[7] is $7\mu = \lambda$
- In the $(\mu + \lambda)$ selection, μ fittest individuals are selected among μ parents and λ offspring
- This selection guarantees the survival of the best individual found so far
- However, it also can keep outdated individuals
- In general, it seems to be better to use the (μ, λ) selection for unbounded search spaces

7. Baeck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press (1996)

- The best μ individuals for the next generation are always chosen **deterministically**
- In the (μ, λ) selection, μ fittest individuals are selected among λ children
- Since all the parents are discarded, it is possible to leave local optima
- The suggested ratio[7] is $7\mu = \lambda$
- In the $(\mu + \lambda)$ selection, μ fittest individuals are selected among μ parents and λ offspring
- This selection guarantees the survival of the best individual found so far
- However, it also can keep outdated individuals
- In general, it seems to be better to use the (μ, λ) selection for unbounded search spaces

7. Baeck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press (1996)

Survival Selection

- The best μ individuals for the next generation are always chosen **deterministically**
- In the (μ, λ) selection, μ fittest individuals are selected among λ children
- Since all the parents are discarded, it is possible to leave local optima
- The suggested ratio[7] is $7\mu = \lambda$
- In the $(\mu + \lambda)$ selection, μ fittest individuals are selected among μ parents and λ offspring
- This selection guarantees the survival of the best individual found so far
- However, it also can keep outdated individuals
- In general, it seems to be better to use the (μ, λ) selection for unbounded search spaces

7. Baeck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press (1996)

- The best μ individuals for the next generation are always chosen **deterministically**
- In the (μ, λ) selection, μ fittest individuals are selected among λ children
- Since all the parents are discarded, it is possible to leave local optima
- The suggested ratio[7] is $7\mu = \lambda$
- In the $(\mu + \lambda)$ selection, μ fittest individuals are selected among μ parents and λ offspring
- This selection guarantees the survival of the best individual found so far
- However, it also can keep outdated individuals
- In general, it seems to be better to use the (μ, λ) selection for unbounded search spaces

7. Baeck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press (1996)

- The best μ individuals for the next generation are always chosen **deterministically**
- In the (μ, λ) selection, μ fittest individuals are selected among λ children
- Since all the parents are discarded, it is possible to leave local optima
- The suggested ratio[7] is $7\mu = \lambda$
- In the $(\mu + \lambda)$ selection, μ fittest individuals are selected among μ parents and λ offspring
- This selection guarantees the survival of the best individual found so far
- However, it also can keep outdated individuals
- In general, it seems to be better to use the (μ, λ) selection for unbounded search spaces

7. Baeck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press (1996)

- The best μ individuals for the next generation are always chosen **deterministically**
- In the (μ, λ) selection, μ fittest individuals are selected among λ children
- Since all the parents are discarded, it is possible to leave local optima
- The suggested ratio[7] is $7\mu = \lambda$
- In the $(\mu + \lambda)$ selection, μ fittest individuals are selected among μ parents and λ offspring
- This selection guarantees the survival of the best individual found so far
- However, it also can keep outdated individuals
- In general, it seems to be better to use the (μ, λ) selection for unbounded search spaces

7. Baeck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press (1996)

- The best μ individuals for the next generation are always chosen **deterministically**
- In the (μ, λ) selection, μ fittest individuals are selected among λ children
- Since all the parents are discarded, it is possible to leave local optima
- The suggested ratio[7] is $7\mu = \lambda$
- In the $(\mu + \lambda)$ selection, μ fittest individuals are selected among μ parents and λ offspring
- This selection guarantees the survival of the best individual found so far
- However, it also can keep outdated individuals
- In general, it seems to be better to use the (μ, λ) selection for unbounded search spaces

7. Baeck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press (1996)

- The best μ individuals for the next generation are always chosen **deterministically**
- In the (μ, λ) selection, μ fittest individuals are selected among λ children
- Since all the parents are discarded, it is possible to leave local optima
- The suggested ratio[7] is $7\mu = \lambda$
- In the $(\mu + \lambda)$ selection, μ fittest individuals are selected among μ parents and λ offspring
- This selection guarantees the survival of the best individual found so far
- However, it also can keep outdated individuals
- In general, it seems to be better to use the (μ, λ) selection for unbounded search spaces

7. Baeck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press (1996)

Initialization, Termination, Constraint Handling

- Initial population is obtained by mutating the starting point μ times
- Initial strategy parameters can be selected all equal to 1
- Termination criteria can be:
  - Maximum number of generations elapsed
  - Distance between best and worst fitness values is below a certain threshold
- Constraint handling is very intuitive: after mutation, discard individuals that violate constraints, and generate new ones

Initialization, Termination, Constraint Handling

- Initial population is obtained by mutating the starting point μ times
- Initial strategy parameters can be selected all equal to 1
- Termination criteria can be:
  - Maximum number of generations elapsed
  - Distance between best and worst fitness values is below a certain threshold
- Constraint handling is very intuitive: after mutation, discard individuals that violate constraints, and generate new ones

Initialization, Termination, Constraint Handling

- Initial population is obtained by mutating the starting point μ times
- Initial strategy parameters can be selected all equal to 1
- Termination criteria can be:
  - Maximum number of generations elapsed
  - Distance between best and worst fitness values is below a certain threshold
- Constraint handling is very intuitive: after mutation, discard individuals that violate constraints, and generate new ones

Initialization, Termination, Constraint Handling

- Initial population is obtained by mutating the starting point μ times
- Initial strategy parameters can be selected all equal to 1
- Termination criteria can be:
  - Maximum number of generations elapsed
  - Distance between best and worst fitness values is below a certain threshold
- Constraint handling is very intuitive: after mutation, discard individuals that violate constraints, and generate new ones

Initialization, Termination, Constraint Handling

- Initial population is obtained by mutating the starting point μ times
- Initial strategy parameters can be selected all equal to 1
- Termination criteria can be:
  - Maximum number of generations elapsed
  - Distance between best and worst fitness values is below a certain threshold
- Constraint handling is very intuitive: after mutation, discard individuals that violate constraints, and generate new ones

Initialization, Termination, Constraint Handling

- Initial population is obtained by mutating the starting point μ times
- Initial strategy parameters can be selected all equal to 1
- Termination criteria can be:
  - Maximum number of generations elapsed
  - Distance between best and worst fitness values is below a certain threshold
- Constraint handling is very intuitive: after mutation, discard individuals that violate constraints, and generate new ones

Evolution Strategy Jupyter Notebook

Covariance Matrix Adaptation ES

- There is another version of ES, in which not only mutation step sizes are self-adapted, but rotation angles as well
- It is called the **covariance matrix adaptation ES** (CMA-ES)[8]
- The idea is to enable mutation of the whole covariance matrix (not only its diagonal elements σ_i)

8. Hansen, N., Ostermeier, A.: Completely Derandomized Self-Adaptation in Evolution Strategies, *Evolutionary Computation* 9(2), 159–195 (2001)

Covariance Matrix Adaptation ES

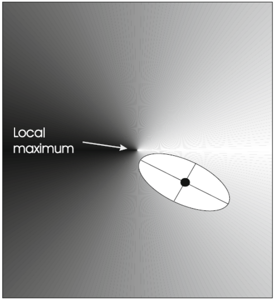
- There is another version of ES, in which not only mutation step sizes are self-adapted, but rotation angles as well
- It is called the **covariance matrix adaptation ES** (CMA-ES)[8]
- The idea is to enable mutation of the whole covariance matrix (not only its diagonal elements σ_i)

8. Hansen, N., Ostermeier, A.: Completely Derandomized Self-Adaptation in Evolution Strategies, *Evolutionary Computation* 9(2), 159–195 (2001)

- There is another version of ES, in which not only mutation step sizes are self-adapted, but rotation angles as well
- It is called the **covariance matrix adaptation ES** (CMA-ES)[8]
- The idea is to enable mutation of the whole covariance matrix (not only its diagonal elements σ_i)

8. Hansen, N., Ostermeier, A.: Completely Derandomized Self-Adaptation in Evolution Strategies, *Evolutionary Computation* 9(2), 159–195 (2001)

- There is another version of ES, in which not only mutation step sizes are self-adapted, but rotation angles as well
- It is called the **covariance matrix adaptation ES** (CMA-ES)[8]
- The idea is to enable mutation of the whole covariance matrix (not only its diagonal elements σ_i)



Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer-Verlag, Berlin, Heidelberg (2015), Fig. 4.6

8. Hansen, N., Ostermeier, A.: Completely Derandomized Self-Adaptation in Evolution Strategies, *Evolutionary Computation* 9(2), 159–195 (2001)

Covariance Matrix Adaptation ES

- We will not discuss it in much detail, because it is rather involved
- Aside from adapting covariance matrix, CMA-ES also uses *cumulative* adaptation, by adapting strategy parameters in a weighted manner from generation to generation
- Please refer to appropriate papers in the *Literature* folder and some implementations posted online:
  - pycma (implementation by the author of CMA-ES)
  - cmaes (a lightweight implementation)
- It is considered very successful and is widely used in practice

Covariance Matrix Adaptation ES

- We will not discuss it in much detail, because it is rather involved
- Aside from adapting covariance matrix, CMA-ES also uses *cumulative* adaptation, by adapting strategy parameters in a weighted manner from generation to generation
- Please refer to appropriate papers in the *Literature* folder and some implementations posted online:
    - pycma (implementation by the author of CMA-ES)
    - cmaes (a lightweight implementation)
- It is considered very successful and is widely used in practice

Covariance Matrix Adaptation ES

- We will not discuss it in much detail, because it is rather involved
- Aside from adapting covariance matrix, CMA-ES also uses *cumulative* adaptation, by adapting strategy parameters in a weighted manner from generation to generation
- Please refer to appropriate papers in the *Literature* folder and some implementations posted online:
  - pycma (implementation by the author of CMA-ES)
  - cmaes (a lightweight implementation)
- It is considered very successful and is widely used in practice

- We will not discuss it in much detail, because it is rather involved
- Aside from adapting covariance matrix, CMA-ES also uses *cumulative* adaptation, by adapting strategy parameters in a weighted manner from generation to generation
- Please refer to appropriate papers in the *Literature* folder and some implementations posted online:
  - pycma (implementation by the author of CMA-ES)
  - cmaes (a lightweight implementation)
- It is considered very successful and is widely used in practice

Covariance Matrix Adaptation ES

- We will not discuss it in much detail, because it is rather involved
- Aside from adapting covariance matrix, CMA-ES also uses *cumulative* adaptation, by adapting strategy parameters in a weighted manner from generation to generation
- Please refer to appropriate papers in the *Literature* folder and some implementations posted online:
  - pycma (implementation by the author of CMA-ES)
  - cmaes (a lightweight implementation)
- It is considered very successful and is widely used in practice

- We will not discuss it in much detail, because it is rather involved
- Aside from adapting covariance matrix, CMA-ES also uses *cumulative* adaptation, by adapting strategy parameters in a weighted manner from generation to generation
- Please refer to appropriate papers in the *Literature* folder and some implementations posted online:
    - pycma (implementation by the author of CMA-ES)
    - cmaes (a lightweight implementation)
- It is considered very successful and is widely used in practice

Outline

- As we have seen, EAs are very good at *exploration*, but are less good at fine-tuning of solutions (*exploitation*)
- It seems to be a good idea to incorporate problem-specific knowledge in EAs
- The concept of memes was introduced by Richard Dawkins in 1976 in *The Selfish Gene*
- According to Dawkins, memes are units of cultural transmission, just like genes are units of biological transmission

- As we have seen, EAs are very good at *exploration*, but are less good at fine-tuning of solutions (*exploitation*)
- It seems to be a good idea to incorporate problem-specific knowledge in EAs
- The concept of memes was introduced by Richard Dawkins in 1976 in *The Selfish Gene*
- According to Dawkins, memes are units of cultural transmission, just like genes are units of biological transmission

    "Examples of memes are tunes, ideas, catch-phrases, clothes fashions, ways of making pots or of building arches. Just as genes propagate themselves in the gene pool by leaping from body to body via sperm or eggs, so memes propagate themselves in the meme pool by leaping from brain to brain via a process which, in the broad sense, can be called imitation."

- As we have seen, EAs are very good at *exploration*, but are less good at fine-tuning of solutions (*exploitation*)
- It seems to be a good idea to incorporate problem-specific knowledge in EAs
- The concept of memes was introduced by Richard Dawkins in 1976 in *The Selfish Gene*
- According to Dawkins, memes are units of cultural transmission, just like genes are units of biological transmission

> *"Examples of memes are tunes, ideas, catch-phrases, clothes fashions, ways of making pots or of building arches. Just as genes propagate themselves in the gene pool by leaping from body to body via sperm or eggs, so memes propagate themselves in the meme pool by leaping from brain to brain via a process which, in the broad sense, can be called imitation."*

Motivation

- As we have seen, EAs are very good at *exploration*, but are less good at fine-tuning of solutions (*exploitation*)
- It seems to be a good idea to incorporate problem-specific knowledge in EAs
- The concept of memes was introduced by Richard Dawkins in 1976 in *The Selfish Gene*
- According to Dawkins, memes are units of cultural transmission, just like genes are units of biological transmission

> *"Examples of memes are tunes, ideas, catch-phrases, clothes fashions, ways of making pots or of building arches. Just as genes propagate themselves in the gene pool by leaping from body to body via sperm or eggs, so memes propagate themselves in the meme pool by leaping from brain to brain via a process which, in the broad sense, can be called imitation."*

- As we have seen, EAs are very good at *exploration*, but are less good at fine-tuning of solutions (*exploitation*)
- It seems to be a good idea to incorporate problem-specific knowledge in EAs
- The concept of memes was introduced by Richard Dawkins in 1976 in *The Selfish Gene*
- According to Dawkins, memes are units of cultural transmission, just like genes are units of biological transmission

  > *"Examples of memes are tunes, ideas, catch-phrases, clothes fashions, ways of making pots or of building arches. Just as genes propagate themselves in the gene pool by leaping from body to body via sperm or eggs, so memes propagate themselves in the meme pool by leaping from brain to brain via a process which, in the broad sense, can be called imitation."*

Genes vs. Memes

- Genes and memes differ strongly in the copying fidelity: memes mutate with a much higher rate
- Memes allow improvement: the individual adapts the meme as it sees best; memes can be improved by the individual holding it
- This idea can be used in genetic algorithms to improve individuals
- *Local heuristics* can be applied so that the space of possible solutions is reduced to the *subspace of local optima*

- Genes and memes differ strongly in the copying fidelity: memes mutate with a much higher rate
- Memes allow improvement: the individual adapts the meme as it sees best; memes can be improved by the individual holding it
- This idea can be used in genetic algorithms to improve individuals
- *Local heuristics* can be applied so that the space of possible solutions is reduced to the *subspace of local optima*

- Genes and memes differ strongly in the copying fidelity: memes mutate with a much higher rate
- Memes allow improvement: the individual adapts the meme as it sees best; memes can be improved by the individual holding it
- This idea can be used in genetic algorithms to improve individuals
- *Local heuristics* can be applied so that the space of possible solutions is reduced to the *subspace of local optima*

- Genes and memes differ strongly in the copying fidelity: memes mutate with a much higher rate
- Memes allow improvement: the individual adapts the meme as it sees best; memes can be improved by the individual holding it
- This idea can be used in genetic algorithms to improve individuals
- *Local heuristics* can be applied so that the space of possible solutions is reduced to the *subspace of local optima*

- The term “memetic algorithm” was coined by Pablo Moscato in 1989
- Memetic algorithms (MAs) are “carefully orchestrated interplay between (stochastic) global search and (stochastic) local search algorithms”
- These algorithms are also frequently called hybrid genetic algorithms
- Memetic algorithms are generally much faster and more accurate than EAs on some problems

Introducing Memetic Algorithms

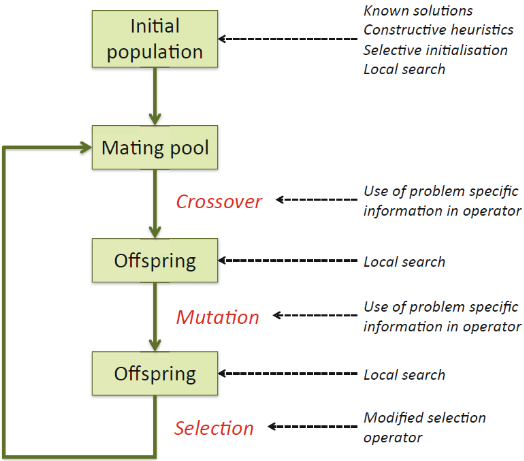
- The term "memetic algorithm" was coined by Pablo Moscato in 1989
- Memetic algorithms (MAs) are "carefully orchestrated interplay between (stochastic) global search and (stochastic) local search algorithms"
- These algorithms are also frequently called hybrid genetic algorithms
- Memetic algorithms are generally much faster and more accurate than EAs on some problems

Introducing Memetic Algorithms

- The term "memetic algorithm" was coined by Pablo Moscato in 1989
- Memetic algorithms (MAs) are "carefully orchestrated interplay between (stochastic) global search and (stochastic) local search algorithms"
- These algorithms are also frequently called hybrid genetic algorithms
- Memetic algorithms are generally much faster and more accurate than EAs on some problems

- The term "memetic algorithm" was coined by Pablo Moscato in 1989
- Memetic algorithms (MAs) are "carefully orchestrated interplay between (stochastic) global search and (stochastic) local search algorithms"
- These algorithms are also frequently called hybrid genetic algorithms
- Memetic algorithms are generally much faster and more accurate than EAs on some problems

Places to Apply Hybridization



Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer-Verlag, Berlin, Heidelberg (2015), Fig. 10.3

- Local search can be integrated within the evolutionary cycle in two ways[9]
- The first one is "lifetime learning": application of the local search to a candidate solution
- In this case the metaphor is cultural development of individuals, which is then transmitted to other solutions over subsequent generations
- The second one is application of the local search during solution **generation**, i.e. generation of a perfect child
- This class of memetic implementations aims at selecting the most convenient offspring among potential offspring solutions
- E.g., heuristic selection of the crossover point or the best bit to mutate

9. de Oca, M.A., Cotta, C., Neri, F.: Local Search. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 29–41. Springer-Verlag (2012)

- Local search can be integrated within the evolutionary cycle in two ways[9]
- The first one is "lifetime learning": application of the local search to a candidate solution
- In this case the metaphor is cultural development of individuals, which is then transmitted to other solutions over subsequent generations
- The second one is application of the local search during solution **generation**, i.e. generation of a perfect child
- This class of memetic implementations aims at selecting the most convenient offspring among potential offspring solutions
- E.g., heuristic selection of the crossover point or the best bit to mutate

9. de Oca, M.A., Cotta, C., Neri, F.: Local Search. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 29–41. Springer-Verlag (2012)

- Local search can be integrated within the evolutionary cycle in two ways[9]
- The first one is "lifetime learning": application of the local search to a candidate solution
- In this case the metaphor is cultural development of individuals, which is then transmitted to other solutions over subsequent generations
- The second one is application of the local search during solution **generation**, i.e. generation of a perfect child
- This class of memetic implementations aims at selecting the most convenient offspring among potential offspring solutions
- E.g., heuristic selection of the crossover point or the best bit to mutate

9. de Oca, M.A., Cotta, C., Neri, F.: Local Search. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 29–41. Springer-Verlag (2012)

- Local search can be integrated within the evolutionary cycle in two ways[9]
- The first one is "lifetime learning": application of the local search to a candidate solution
- In this case the metaphor is cultural development of individuals, which is then transmitted to other solutions over subsequent generations
- The second one is application of the local search during solution **generation**, i.e. generation of a perfect child
- This class of memetic implementations aims at selecting the most convenient offspring among potential offspring solutions
- E.g., heuristic selection of the crossover point or the best bit to mutate

9. de Oca, M.A., Cotta, C., Neri, F.: Local Search. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 29–41. Springer-Verlag (2012)

- Local search can be integrated within the evolutionary cycle in two ways[9]
- The first one is "lifetime learning": application of the local search to a candidate solution
- In this case the metaphor is cultural development of individuals, which is then transmitted to other solutions over subsequent generations
- The second one is application of the local search during solution **generation**, i.e. generation of a perfect child
- This class of memetic implementations aims at selecting the most convenient offspring among potential offspring solutions
- E.g., heuristic selection of the crossover point or the best bit to mutate

9. de Oca, M.A., Cotta, C., Neri, F.: Local Search. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 29–41. Springer-Verlag (2012)

- Local search can be integrated within the evolutionary cycle in two ways[9]
- The first one is "lifetime learning": application of the local search to a candidate solution
- In this case the metaphor is cultural development of individuals, which is then transmitted to other solutions over subsequent generations
- The second one is application of the local search during solution **generation**, i.e. generation of a perfect child
- This class of memetic implementations aims at selecting the most convenient offspring among potential offspring solutions
- E.g., heuristic selection of the crossover point or the best bit to mutate

9. de Oca, M.A., Cotta, C., Neri, F.: Local Search. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 29–41. Springer-Verlag (2012)

- To understand local search, the notion of **neighborhood** is essential[10]
- Let S be the search space of the given problem
- A neighborhood N over S is any function that associates to each solution s from S other solutions from $N(s)$
- Any solution s' from $N(s)$ is called a **neighbor** of s
- A solution s^l is a **local optimum** with respect to N if it is the best solution in $N(s^l)$ in terms of some objective function
- The notion of neighborhood can be explained in terms of the **move** operator
- Typically applying a move mv to solution s changes s slightly and leads to a neighbor solution s':

$$s' = s \oplus mv$$

- Then, the neighborhood can be defined as

$$N(s) = \{s \oplus mv \mid mv \in \Gamma(s)\} \ ,$$

where $\Gamma(s)$ is the set of all possible moves that can be applied to s

10. Hao, J.-K.: Memetic Algorithms in Discrete Optimization. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 73–94. Springer-Verlag (2012)

- To understand local search, the notion of **neighborhood** is essential[10]
- Let S be the search space of the given problem
- A neighborhood N over S is any function that associates to each solution s from S other solutions from $N(s)$
- Any solution s' from $N(s)$ is called a **neighbor** of s
- A solution s^l is a **local optimum** with respect to N if it is the best solution in $N(s^l)$ in terms of some objective function
- The notion of neighborhood can be explained in terms of the **move** operator
- Typically applying a move mv to solution s changes s slightly and leads to a neighbor solution s':

$$s' = s \oplus mv$$

- Then, the neighborhood can be defined as

$$N(s) = \{s \oplus mv \mid mv \in \Gamma(s)\} \ ,$$

where $\Gamma(s)$ is the set of all possible moves that can be applied to s

10. Hao, J.-K.: Memetic Algorithms in Discrete Optimization. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 73–94. Springer-Verlag (2012)

Local Search Basics

- To understand local search, the notion of **neighborhood** is essential[10]
- Let S be the search space of the given problem
- A neighborhood N over S is any function that associates to each solution s from S other solutions from $N(s)$
- Any solution s' from $N(s)$ is called a **neighbor** of s
- A solution s^l is a **local optimum** with respect to N if it is the best solution in $N(s^l)$ in terms of some objective function
- The notion of neighborhood can be explained in terms of the **move** operator
- Typically applying a move mv to solution s changes s slightly and leads to a neighbor solution s':

$$s' = s \oplus mv$$

- Then, the neighborhood can be defined as

$$N(s) = \{s \oplus mv \mid mv \in \Gamma(s)\} ,$$

where $\Gamma(s)$ is the set of all possible moves that can be applied to s

10. Hao, J.-K.: Memetic Algorithms in Discrete Optimization. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 73–94. Springer-Verlag (2012)

Local Search Basics

- To understand local search, the notion of **neighborhood** is essential[10]
- Let S be the search space of the given problem
- A neighborhood N over S is any function that associates to each solution s from S other solutions from $N(s)$
- Any solution s' from $N(s)$ is called a **neighbor** of s
- A solution s^l is a **local optimum** with respect to N if it is the best solution in $N(s^l)$ in terms of some objective function
- The notion of neighborhood can be explained in terms of the **move** operator
- Typically applying a move mv to solution s changes s slightly and leads to a neighbor solution s':

$$s' = s \oplus mv$$

- Then, the neighborhood can be defined as

$$N(s) = \{s \oplus mv \mid mv \in \Gamma(s)\} \; ,$$

where $\Gamma(s)$ is the set of all possible moves that can be applied to s

10. Hao, J.-K.: Memetic Algorithms in Discrete Optimization. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 73–94. Springer-Verlag (2012)

Local Search Basics

- To understand local search, the notion of **neighborhood** is essential[10]
- Let S be the search space of the given problem
- A neighborhood N over S is any function that associates to each solution s from S other solutions from $N(s)$
- Any solution s' from $N(s)$ is called a **neighbor** of s
- A solution s^l is a **local optimum** with respect to N if it is the best solution in $N(s^l)$ in terms of some objective function
- The notion of neighborhood can be explained in terms of the **move** operator
- Typically applying a move mv to solution s changes s slightly and leads to a neighbor solution s':

$$s' = s \oplus mv$$

- Then, the neighborhood can be defined as

$$N(s) = \{s \oplus mv \mid mv \in \Gamma(s)\} ,$$

  where $\Gamma(s)$ is the set of all possible moves that can be applied to s

10. Hao, J.-K.: Memetic Algorithms in Discrete Optimization. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 73–94. Springer-Verlag (2012)

- To understand local search, the notion of **neighborhood** is essential[10]
- Let S be the search space of the given problem
- A neighborhood N over S is any function that associates to each solution s from S other solutions from $N(s)$
- Any solution s' from $N(s)$ is called a **neighbor** of s
- A solution s^l is a **local optimum** with respect to N if it is the best solution in $N(s^l)$ in terms of some objective function
- The notion of neighborhood can be explained in terms of the **move** operator
- Typically applying a move mv to solution s changes s slightly and leads to a neighbor solution s':

$$s' = s \oplus mv$$

- Then, the neighborhood can be defined as

$$N(s) = \{s \oplus mv \mid mv \in \Gamma(s)\} \ ,$$

where $\Gamma(s)$ is the set of all possible moves that can be applied to s

10. Hao, J.-K.: Memetic Algorithms in Discrete Optimization. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 73–94. Springer-Verlag (2012)

- To understand local search, the notion of **neighborhood** is essential[10]
- Let S be the search space of the given problem
- A neighborhood N over S is any function that associates to each solution s from S other solutions from $N(s)$
- Any solution s' from $N(s)$ is called a **neighbor** of s
- A solution s^l is a **local optimum** with respect to N if it is the best solution in $N(s^l)$ in terms of some objective function
- The notion of neighborhood can be explained in terms of the **move** operator
- Typically applying a move mv to solution s changes s slightly and leads to a neighbor solution s':
$$s' = s \oplus mv$$
- Then, the neighborhood can be defined as
$$N(s) = \{s \oplus mv \mid mv \in \Gamma(s)\} \ ,$$
where $\Gamma(s)$ is the set of all possible moves that can be applied to s

10. Hao, J.-K.: Memetic Algorithms in Discrete Optimization. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 73–94. Springer-Verlag (2012)

- To understand local search, the notion of **neighborhood** is essential[10]
- Let S be the search space of the given problem
- A neighborhood N over S is any function that associates to each solution s from S other solutions from $N(s)$
- Any solution s' from $N(s)$ is called a **neighbor** of s
- A solution s^l is a **local optimum** with respect to N if it is the best solution in $N(s^l)$ in terms of some objective function
- The notion of neighborhood can be explained in terms of the **move** operator
- Typically applying a move mv to solution s changes s slightly and leads to a neighbor solution s':

$$s' = s \oplus mv$$

- Then, the neighborhood can be defined as

$$N(s) = \{s \oplus mv \mid mv \in \Gamma(s)\} \ ,$$

where $\Gamma(s)$ is the set of all possible moves that can be applied to s

10. Hao, J.-K.: Memetic Algorithms in Discrete Optimization. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 73–94. Springer-Verlag (2012)

- Success of a local search algorithm depends strongly on its neighborhood
- The choice of neighborhood is conditioned by the representation
- *Binary representation*: two basic neighborhoods are defined by the k-**flip** and **Swap** move operators:
- *Permutation representation*: two basic neighborhoods are available using *Swap* and *Inversion* moves
- *Real-valued representation*: in this case, the neighborhood is simply the set of points that are closer than some distance (e.g., Euclidean distance)

Types of Neighborhoods

- Success of a local search algorithm depends strongly on its neighborhood
- The choice of neighborhood is conditioned by the representation
- *Binary representation*: two basic neighborhoods are defined by the k-**flip** and **Swap** move operators:
  - The k-flip move changes the values of k bits (all neighbors have a Hamming distance k)
  - Swap operator exchanges values of two variables that have different values
- *Permutation representation*: two basic neighborhoods are available using *Swap* and *Inversion* moves
- *Real-valued representation*: in this case, the neighborhood is simply the set of points that are closer than some distance (e.g., Euclidean distance)

Types of Neighborhoods

- Success of a local search algorithm depends strongly on its neighborhood
- The choice of neighborhood is conditioned by the representation
- *Binary representation*: two basic neighborhoods are defined by the **k-flip** and **Swap** move operators:
  - The k-flip move changes the values of k bits (all neighbors have a Hamming distance k)
  - Swap operator exchanges values of two variables that have different values
- *Permutation representation*: two basic neighborhoods are available using *Swap* and *Inversion* moves
- *Real-valued representation*: in this case, the neighborhood is simply the set of points that are closer than some distance (e.g., Euclidean distance)

Types of Neighborhoods

- Success of a local search algorithm depends strongly on its neighborhood
- The choice of neighborhood is conditioned by the representation
- *Binary representation*: two basic neighborhoods are defined by the **k-flip** and **Swap** move operators:
  - The k-flip move changes the values of k bits (all neighbors have a Hamming distance k)
  - Swap operator exchanges values of two variables that have different values
- *Permutation representation*: two basic neighborhoods are available using *Swap* and *Inversion* moves
- *Real-valued representation*: in this case, the neighborhood is simply the set of points that are closer than some distance (e.g., Euclidean distance)

Types of Neighborhoods

- Success of a local search algorithm depends strongly on its neighborhood
- The choice of neighborhood is conditioned by the representation
- *Binary representation*: two basic neighborhoods are defined by the **k-flip** and **Swap** move operators:
  - The k-flip move changes the values of k bits (all neighbors have a Hamming distance k)
  - Swap operator exchanges values of two variables that have different values
- *Permutation representation*: two basic neighborhoods are available using *Swap* and *Inversion* moves
- *Real-valued representation*: in this case, the neighborhood is simply the set of points that are closer than some distance (e.g., Euclidean distance)

Types of Neighborhoods

- Success of a local search algorithm depends strongly on its neighborhood
- The choice of neighborhood is conditioned by the representation
- *Binary representation*: two basic neighborhoods are defined by the **k-flip** and **Swap** move operators:
  - The k-flip move changes the values of k bits (all neighbors have a Hamming distance k)
  - Swap operator exchanges values of two variables that have different values
- *Permutation representation*: two basic neighborhoods are available using *Swap* and *Inversion* moves
- *Real-valued representation*: in this case, the neighborhood is simply the set of points that are closer than some distance (e.g., Euclidean distance)

Types of Neighborhoods

- Success of a local search algorithm depends strongly on its neighborhood
- The choice of neighborhood is conditioned by the representation
- *Binary representation*: two basic neighborhoods are defined by the **k-flip** and **Swap** move operators:
  - The k-flip move changes the values of k bits (all neighbors have a Hamming distance k)
  - Swap operator exchanges values of two variables that have different values
- *Permutation representation*: two basic neighborhoods are available using *Swap* and *Inversion* moves
- *Real-valued representation*: in this case, the neighborhood is simply the set of points that are closer than some distance (e.g., Euclidean distance)

- Local search algorithms can be classified from various perspectives[11]
- According to the nature of the search logic:
  - **Stochastic**: the generation of the trial solution occurs in a randomized way
  - **Deterministic**: the generation of the trial solution is deterministic
- According to the number of solutions involved:
- According to the **pivot rule**:

11. de Oca, M.A., Cotta, C., Neri, F.: Local Search. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 29–41. Springer-Verlag (2012)

Classification of Local Search Techniques

- Local search algorithms can be classified from various perspectives[11]
- According to the nature of the search logic:
  - **Stochastic**: the generation of the trial solution occurs in a randomized way
  - **Deterministic**: the generation of the trial solution is deterministic
- According to the number of solutions involved:
- According to the **pivot rule**:

11. de Oca, M.A., Cotta, C., Neri, F.: Local Search. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 29–41. Springer-Verlag (2012)

- Local search algorithms can be classified from various perspectives[11]
- According to the nature of the search logic:
  - **Stochastic:** the generation of the trial solution occurs in a randomized way
  - **Deterministic:** the generation of the trial solution is deterministic
- According to the number of solutions involved:
- According to the **pivot rule**:

11. de Oca, M.A., Cotta, C., Neri, F.: Local Search. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 29–41. Springer-Verlag (2012)

Classification of Local Search Techniques

- Local search algorithms can be classified from various perspectives[11]
- According to the nature of the search logic:
    - **Stochastic**: the generation of the trial solution occurs in a randomized way
    - **Deterministic**: the generation of the trial solution is deterministic
- According to the number of solutions involved:
    - **Single-solution**: the algorithm processes and perturbs only one solution
    - **Multiple-solution**: the algorithm processes more that one solution which are usually employed for interacting and jointly generate trial solutions
- According to the **pivot rule**:

11. de Oca, M.A., Cotta, C., Neri, F.: Local Search. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 29–41. Springer-Verlag (2012)

- Local search algorithms can be classified from various perspectives[11]
- According to the nature of the search logic:
  - **Stochastic**: the generation of the trial solution occurs in a randomized way
  - **Deterministic**: the generation of the trial solution is deterministic
- According to the number of solutions involved:
  - **Single-solution**: the algorithm processes and perturbs only one solution
  - **Multiple-solution**: the algorithm processes more that one solution which are usually employed for interacting and jointly generate trial solutions
- According to the **pivot rule**:

11. de Oca, M.A., Cotta, C., Neri, F.: Local Search. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 29–41. Springer-Verlag (2012)

Classification of Local Search Techniques

- Local search algorithms can be classified from various perspectives[11]
- According to the nature of the search logic:
  - **Stochastic**: the generation of the trial solution occurs in a randomized way
  - **Deterministic**: the generation of the trial solution is deterministic
- According to the number of solutions involved:
  - **Single-solution**: the algorithm processes and perturbs only one solution
  - **Multiple-solution**: the algorithm processes more that one solution which are usually employed for interacting and jointly generate trial solutions
- According to the **pivot rule**:
  - **Steepest Descent**: the algorithm generates a set of solutions and selects the most promising only after having explored all the possibilities
    Impossible for continuous domains unless the objective function is known and simple
  - **Greedy**: the algorithm performs the replacement as soon as detects a solution outperforming the current best and starts over the exploration

11. de Oca, M.A., Cotta, C., Neri, F.: Local Search. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 29–41. Springer-Verlag (2012)

- Local search algorithms can be classified from various perspectives[11]
- According to the nature of the search logic:
  - **Stochastic**: the generation of the trial solution occurs in a randomized way
  - **Deterministic**: the generation of the trial solution is deterministic
- According to the number of solutions involved:
  - **Single-solution**: the algorithm processes and perturbs only one solution
  - **Multiple-solution**: the algorithm processes more that one solution which are usually employed for interacting and jointly generate trial solutions
- According to the **pivot rule**:
  - **Steepest descent**: the algorithm generates a set of solutions and selects the most promising only after having explored all the possibilities
  - Impossible for continuous domains unless the objective function is known and simple

11. de Oca, M.A., Cotta, C., Neri, F.: Local Search. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 29–41. Springer-Verlag (2012)

- Local search algorithms can be classified from various perspectives[11]
- According to the nature of the search logic:
  - **Stochastic**: the generation of the trial solution occurs in a randomized way
  - **Deterministic**: the generation of the trial solution is deterministic
- According to the number of solutions involved:
  - **Single-solution**: the algorithm processes and perturbs only one solution
  - **Multiple-solution**: the algorithm processes more that one solution which are usually employed for interacting and jointly generate trial solutions
- According to the **pivot rule**:
  - **Steepest descent**: the algorithm generates a set of solutions and selects the most promising only after having explored all the possibilities
  - **Impossible for continuous domains** unless the objective function is known and simple
  - **Greedy:** the algorithm performs the replacement as soon as detects a solution outperforming the current best and starts over the exploration

11. de Oca, M.A., Cotta, C., Neri, F.: Local Search. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 29–41. Springer-Verlag (2012)

Classification of Local Search Techniques

- Local search algorithms can be classified from various perspectives[11]
- According to the nature of the search logic:
  - **Stochastic**: the generation of the trial solution occurs in a randomized way
  - **Deterministic**: the generation of the trial solution is deterministic
- According to the number of solutions involved:
  - **Single-solution**: the algorithm processes and perturbs only one solution
  - **Multiple-solution**: the algorithm processes more that one solution which are usually employed for interacting and jointly generate trial solutions
- According to the **pivot rule**:
  - **Steepest descent**: the algorithm generates a set of solutions and selects the most promising only after having explored all the possibilities
  - **Impossible for continuous domains** unless the objective function is known and simple
  - **Greedy**: the algorithm performs the replacement as soon as detects a solution outperforming the current best and starts over the exploration

11. de Oca, M.A., Cotta, C., Neri, F.: Local Search. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 29–41. Springer-Verlag (2012)

Classification of Local Search Techniques

- Local search algorithms can be classified from various perspectives[11]
- According to the nature of the search logic:
  - **Stochastic**: the generation of the trial solution occurs in a randomized way
  - **Deterministic**: the generation of the trial solution is deterministic
- According to the number of solutions involved:
  - **Single-solution**: the algorithm processes and perturbs only one solution
  - **Multiple-solution**: the algorithm processes more that one solution which are usually employed for interacting and jointly generate trial solutions
- According to the **pivot rule**:
  - **Steepest descent**: the algorithm generates a set of solutions and selects the most promising only after having explored all the possibilities
  - **Impossible for continuous domains** unless the objective function is known and simple
  - **Greedy**: the algorithm performs the replacement as soon as detects a solution outperforming the current best and starts over the exploration

11. de Oca, M.A., Cotta, C., Neri, F.: Local Search. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 29–41. Springer-Verlag (2012)

Classification of Local Search Techniques

- Local search algorithms can be classified from various perspectives[11]
- According to the nature of the search logic:
  - **Stochastic**: the generation of the trial solution occurs in a randomized way
  - **Deterministic**: the generation of the trial solution is deterministic
- According to the number of solutions involved:
  - **Single-solution**: the algorithm processes and perturbs only one solution
  - **Multiple-solution**: the algorithm processes more that one solution which are usually employed for interacting and jointly generate trial solutions
- According to the **pivot rule**:
  - **Steepest descent**: the algorithm generates a set of solutions and selects the most promising only after having explored all the possibilities
  - **Impossible for continuous domains** unless the objective function is known and simple
  - **Greedy**: the algorithm performs the replacement as soon as detects a solution outperforming the current best and starts over the exploration

11. de Oca, M.A., Cotta, C., Neri, F.: Local Search. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, pp. 29–41. Springer-Verlag (2012)

- **Hill climbing**: a trial solution s is perturbed to obtain s' and is replaced with s' if the latter one is better
- Variants: steepest-descent (explore the full neighborhood), random (pick a random neighbor and decide)
- **Simulated annealing**: the same as hill climbing, but the replacement is possible if s' is worse, with a probability

$$p = \begin{cases} 1\ , & f(s) - f(s') > 0 \\ e^{-\frac{f(s)-f(s')}{T}}\ , & \text{otherwise} \end{cases} ,$$

  where T is a time-varying *temperature*, decreased using a *cooling schedule* (many exist in the literature)
- **Tabu search**: the best neighboring solution is chosen as the next configuration, even if it is worse than the s
- To avoid cycling, a tabu list of movements is kept (the actual meaning of a move being tabu may vary depending on the problem and designer's choice)

- **Hill climbing**: a trial solution s is perturbed to obtain s' and is replaced with s' if the latter one is better
- Variants: steepest-descent (explore the full neighborhood), random (pick a random neighbor and decide)
- **Simulated annealing**: the same as hill climbing, but the replacement is possible if s' is worse, with a probability

$$p = \begin{cases} 1 \,, & f(s) - f(s') > 0 \\ e^{-\frac{f(s)-f(s')}{T}} \,, & \text{otherwise} \end{cases} ,$$

  where T is a time-varying *temperature*, decreased using a *cooling schedule* (many exist in the literature)
- **Tabu search**: the best neighboring solution is chosen as the next configuration, even if it is worse than the s
- To avoid cycling, a tabu list of movements is kept (the actual meaning of a move being tabu may vary depending on the problem and designer's choice)

- **Hill climbing**: a trial solution s is perturbed to obtain s' and is replaced with s' if the latter one is better
- Variants: steepest-descent (explore the full neighborhood), random (pick a random neighbor and decide)
- **Simulated annealing**: the same as hill climbing, but the replacement is possible if s' is worse, with a probability

$$p = \begin{cases} 1 \ , & f(s) - f(s') > 0 \\ e^{-\frac{f(s)-f(s')}{T}} \ , & \text{otherwise} \end{cases} \ ,$$

  where T is a time-varying *temperature*, decreased using a *cooling schedule* (many exist in the literature)
- **Tabu search**: the best neighboring solution is chosen as the next configuration, even if it is worse than the s
- To avoid cycling, a tabu list of movements is kept (the actual meaning of a move being tabu may vary depending on the problem and designer's choice)

- **Hill climbing**: a trial solution s is perturbed to obtain s' and is replaced with s' if the latter one is better
- Variants: steepest-descent (explore the full neighborhood), random (pick a random neighbor and decide)
- **Simulated annealing**: the same as hill climbing, but the replacement is possible if s' is worse, with a probability

$$p = \begin{cases} 1\ , & f(s) - f(s') > 0 \\ e^{-\frac{f(s)-f(s')}{T}}\ , & \text{otherwise} \end{cases} ,$$

  where T is a time-varying *temperature*, decreased using a *cooling schedule* (many exist in the literature)
- **Tabu search**: the best neighboring solution is chosen as the next configuration, even if it is worse than the s
- To avoid cycling, a tabu list of movements is kept (the actual meaning of a move being tabu may vary depending on the problem and designer's choice)

- **Hill climbing**: a trial solution s is perturbed to obtain s' and is replaced with s' if the latter one is better
- Variants: steepest-descent (explore the full neighborhood), random (pick a random neighbor and decide)
- **Simulated annealing**: the same as hill climbing, but the replacement is possible if s' is worse, with a probability

$$p = \begin{cases} 1\ , & f(s) - f(s') > 0 \\ e^{-\frac{f(s)-f(s')}{T}}\ , & \text{otherwise} \end{cases} ,$$

  where T is a time-varying *temperature*, decreased using a *cooling schedule* (many exist in the literature)
- **Tabu search**: the best neighboring solution is chosen as the next configuration, even if it is worse than the s
- To avoid cycling, a tabu list of movements is kept (the actual meaning of a move being tabu may vary depending on the problem and designer's choice)

Some Local Search Procedures for Continuous Domains

- Any well-known optimization methods can be used
- **Zeroth-order methods** (direct search methods): methods that don't use derivatives (e.g., Nelder-Mead methods)
- **First-order methods:** methods that use derivatives (e.g., Powell's direction set method)
- **Second-order methods**: methods that use derivatives and Hessian matrices or approximations thereof (e.g., Davidon-Fletcher-Powell method)

- Any well-known optimization methods can be used
- **Zeroth-order methods** (direct search methods): methods that don't use derivatives (e.g., Nelder-Mead methods)
- **First-order methods:** methods that use derivatives (e.g., Powell's direction set method)
- **Second-order methods**: methods that use derivatives and Hessian matrices or approximations thereof (e.g., Davidon-Fletcher-Powell method)

Some Local Search Procedures for Continuous Domains

- Any well-known optimization methods can be used
- **Zeroth-order methods** (direct search methods): methods that don't use derivatives (e.g., Nelder-Mead methods)
- **First-order methods:** methods that use derivatives (e.g., Powell's direction set method)
- **Second-order methods**: methods that use derivatives and Hessian matrices or approximations thereof (e.g., Davidon-Fletcher-Powell method)

Some Local Search Procedures for Continuous Domains

- Any well-known optimization methods can be used
- **Zeroth-order methods** (direct search methods): methods that don't use derivatives (e.g., Nelder-Mead methods)
- **First-order methods:** methods that use derivatives (e.g., Powell's direction set method)
- **Second-order methods**: methods that use derivatives and Hessian matrices or approximations thereof (e.g., Davidon-Fletcher-Powell method)

Parametrization of Memetic Algorithms

- When designing memetic algorithms, the following issues must be tackled[12]
- How often should local search be applied?
- The more frequently you apply, the longer it takes, also, there is a risk to get stuck in local optima
- On which solutions should local search be used?
- The choice can be random or fitness-based
- How long should the local search be run?
- The more thorough the search, the longer it takes to compute
- How efficient does a local search need to be?

12. Hart, W.: Adaptive global optimization with local search. PhD thesis, University of California, San Diego, CA (1994)

- When designing memetic algorithms, the following issues must be tackled[12]
- How often should local search be applied?
- The more frequently you apply, the longer it takes, also, there is a risk to get stuck in local optima
- On which solutions should local search be used?
- The choice can be random or fitness-based
- How long should the local search be run?
- The more thorough the search, the longer it takes to compute
- How efficient does a local search need to be?

12. Hart, W.: Adaptive global optimization with local search. PhD thesis, University of California, San Diego, CA (1994)

- When designing memetic algorithms, the following issues must be tackled[12]
- How often should local search be applied?
- The more frequently you apply, the longer it takes, also, there is a risk to get stuck in local optima
- On which solutions should local search be used?
- The choice can be random or fitness-based
- How long should the local search be run?
- The more thorough the search, the longer it takes to compute
- How efficient does a local search need to be?

12. Hart, W.: Adaptive global optimization with local search. PhD thesis, University of California, San Diego, CA (1994)

- When designing memetic algorithms, the following issues must be tackled[12]
- How often should local search be applied?
- The more frequently you apply, the longer it takes, also, there is a risk to get stuck in local optima
- On which solutions should local search be used?
- The choice can be random or fitness-based
- How long should the local search be run?
- The more thorough the search, the longer it takes to compute
- How efficient does a local search need to be?

12. Hart, W.: Adaptive global optimization with local search. PhD thesis, University of California, San Diego, CA (1994)

- When designing memetic algorithms, the following issues must be tackled[12]
- How often should local search be applied?
- The more frequently you apply, the longer it takes, also, there is a risk to get stuck in local optima
- On which solutions should local search be used?
- The choice can be random or fitness-based
- How long should the local search be run?
- The more thorough the search, the longer it takes to compute
- How efficient does a local search need to be?

12. Hart, W.: Adaptive global optimization with local search. PhD thesis, University of California, San Diego, CA (1994)

- When designing memetic algorithms, the following issues must be tackled[12]
- How often should local search be applied?
- The more frequently you apply, the longer it takes, also, there is a risk to get stuck in local optima
- On which solutions should local search be used?
- The choice can be random or fitness-based
- How long should the local search be run?
- The more thorough the search, the longer it takes to compute
- How efficient does a local search need to be?

12. Hart, W.: Adaptive global optimization with local search. PhD thesis, University of California, San Diego, CA (1994)

Parametrization of Memetic Algorithms

- When designing memetic algorithms, the following issues must be tackled[12]
- How often should local search be applied?
- The more frequently you apply, the longer it takes, also, there is a risk to get stuck in local optima
- On which solutions should local search be used?
- The choice can be random or fitness-based
- How long should the local search be run?
- The more thorough the search, the longer it takes to compute
- How efficient does a local search need to be?

12. Hart, W.: Adaptive global optimization with local search. PhD thesis, University of California, San Diego, CA (1994)

- When designing memetic algorithms, the following issues must be tackled[12]
- How often should local search be applied?
- The more frequently you apply, the longer it takes, also, there is a risk to get stuck in local optima
- On which solutions should local search be used?
- The choice can be random or fitness-based
- How long should the local search be run?
- The more thorough the search, the longer it takes to compute
- How efficient does a local search need to be?

12. Hart, W.: Adaptive global optimization with local search. PhD thesis, University of California, San Diego, CA (1994)