合肥學院
HEFEI UNIVERSITY

Metaheuristic Optimization
10. Genetic Algorithms

Thomas Weise · 汤卫思

tweise@hfuu.edu.cn · http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Faculty of Computer Science and Technology
Institute of Applied Optimization
230601 Shushan District, Hefei, Anhui, China
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99

合肥学院 南艳湖校区/南2区
计算机科学与技术系
应用优化研究所
中国 安徽省 合肥市 蜀山区 230601
经济技术开发区 锦绣大道99号

Outline

website

Section Outline

Introduction

- Darwin's Idea of Evolution (1859): "On the Origin of Species" [1]

Introduction

- Darwin's Idea of Evolution (1859): "On the Origin of Species" [1]
  1. individuals possess great fertility and produce more offspring than can grow into adulthood

Introduction

- Darwin's Idea of Evolution (1859): "On the Origin of Species" [1]
  1. individuals possess great fertility and produce more offspring than can grow into adulthood
  2. under the absence of external influences, the population size of a species roughly remains constant

- Darwin's Idea of Evolution (1859): "On the Origin of Species" [1]
  1. individuals possess great fertility and produce more offspring than can grow into adulthood
  2. under the absence of external influences, the population size of a species roughly remains constant
  3. if no external influences occur, the food resources are limited but stable

- Darwin's Idea of Evolution (1859): "On the Origin of Species" [1]
  1. individuals possess great fertility and produce more offspring than can grow into adulthood
  2. under the absence of external influences, the population size of a species roughly remains constant
  3. if no external influences occur, the food resources are limited but stable
  4. individuals compete for these limited resources

- Darwin's Idea of Evolution (1859): "On the Origin of Species" [1]
  1. individuals possess great fertility and produce more offspring than can grow into adulthood
  2. under the absence of external influences, the population size of a species roughly remains constant
  3. if no external influences occur, the food resources are limited but stable
  4. individuals compete for these limited resources
  5. in sexual reproducing species, no two individuals are equal

Introduction

- Darwin's Idea of Evolution (1859): "On the Origin of Species" [1]
  1. individuals possess great fertility and produce more offspring than can grow into adulthood
  2. under the absence of external influences, the population size of a species roughly remains constant
  3. if no external influences occur, the food resources are limited but stable
  4. individuals compete for these limited resources
  5. in sexual reproducing species, no two individuals are equal
  6. some of the variations between the individuals will affect their fitness and hence, their ability to survive

Introduction

- Darwin's Idea of Evolution (1859): "On the Origin of Species" [1]
  1. individuals possess great fertility and produce more offspring than can grow into adulthood
  2. under the absence of external influences, the population size of a species roughly remains constant
  3. if no external influences occur, the food resources are limited but stable
  4. individuals compete for these limited resources
  5. in sexual reproducing species, no two individuals are equal
  6. some of the variations between the individuals will affect their fitness and hence, their ability to survive
  7. a fraction of these variations are inheritable

- Darwin's Idea of Evolution (1859): "On the Origin of Species" [1]
  1. individuals possess great fertility and produce more offspring than can grow into adulthood
  2. under the absence of external influences, the population size of a species roughly remains constant
  3. if no external influences occur, the food resources are limited but stable
  4. individuals compete for these limited resources
  5. in sexual reproducing species, no two individuals are equal
  6. some of the variations between the individuals will affect their fitness and hence, their ability to survive
  7. a fraction of these variations are inheritable
  8. individuals less fit are less likely to reproduce, whereas the fittest individuals will survive and produce offspring more probably

- Darwin's Idea of Evolution (1859): "On the Origin of Species" [1]
  1. individuals possess great fertility and produce more offspring than can grow into adulthood
  2. under the absence of external influences, the population size of a species roughly remains constant
  3. if no external influences occur, the food resources are limited but stable
  4. individuals compete for these limited resources
  5. in sexual reproducing species, no two individuals are equal
  6. some of the variations between the individuals will affect their fitness and hence, their ability to survive
  7. a fraction of these variations are inheritable
  8. individuals less fit are less likely to reproduce, whereas the fittest individuals will survive and produce offspring more probably
  9. individuals that reproduce will likely pass on their traits to their offspring

Introduction

- Darwin's Idea of Evolution (1859): "On the Origin of Species" [1]
  1. individuals possess great fertility and produce more offspring than can grow into adulthood
  2. under the absence of external influences, the population size of a species roughly remains constant
  3. if no external influences occur, the food resources are limited but stable
  4. individuals compete for these limited resources
  5. in sexual reproducing species, no two individuals are equal
  6. some of the variations between the individuals will affect their fitness and hence, their ability to survive
  7. a fraction of these variations are inheritable
  8. individuals less fit are less likely to reproduce, whereas the fittest individuals will survive and produce offspring more probably
  9. individuals that reproduce will likely pass on their traits to their offspring
  10. hence, a species will slowly change and adapt to a given environment

- Individuals of a species exist within a population (not alone)

- Individuals of a species exist within a population (not alone)
- Here, we consider generations: there is a parent generation, followed by a child generation (the offspring)

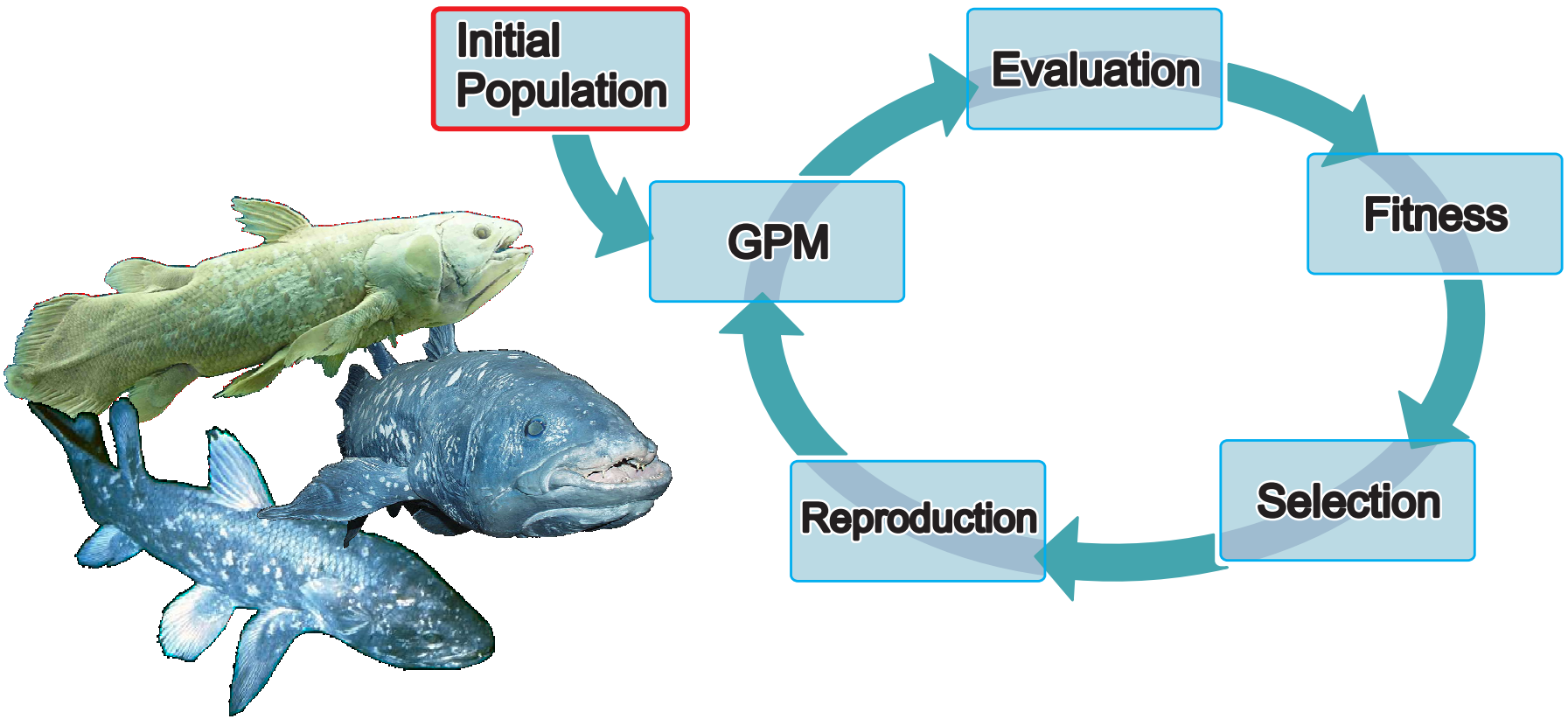
- Individuals of a species exist within a population (not alone)
- Here, we consider generations: there is a parent generation, followed by a child generation (the offspring)
- The fitness of each individual determines its probability of survival during selection

- Individuals of a species exist within a population (not alone)
- Here, we consider generations: there is a parent generation, followed by a child generation (the offspring)
- The fitness of each individual determines its probability of survival during selection
- Individuals which survive become the parents of the next generation

Section Outline

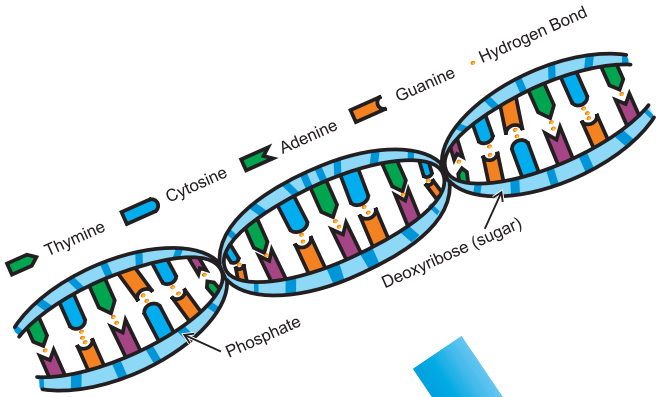
Evolution: First Generation

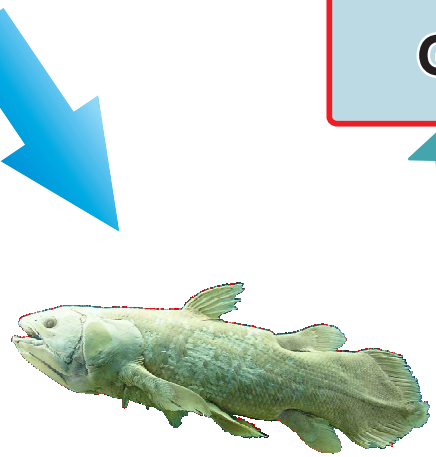
- Start with a random set of individuals (or, their genetic codes)

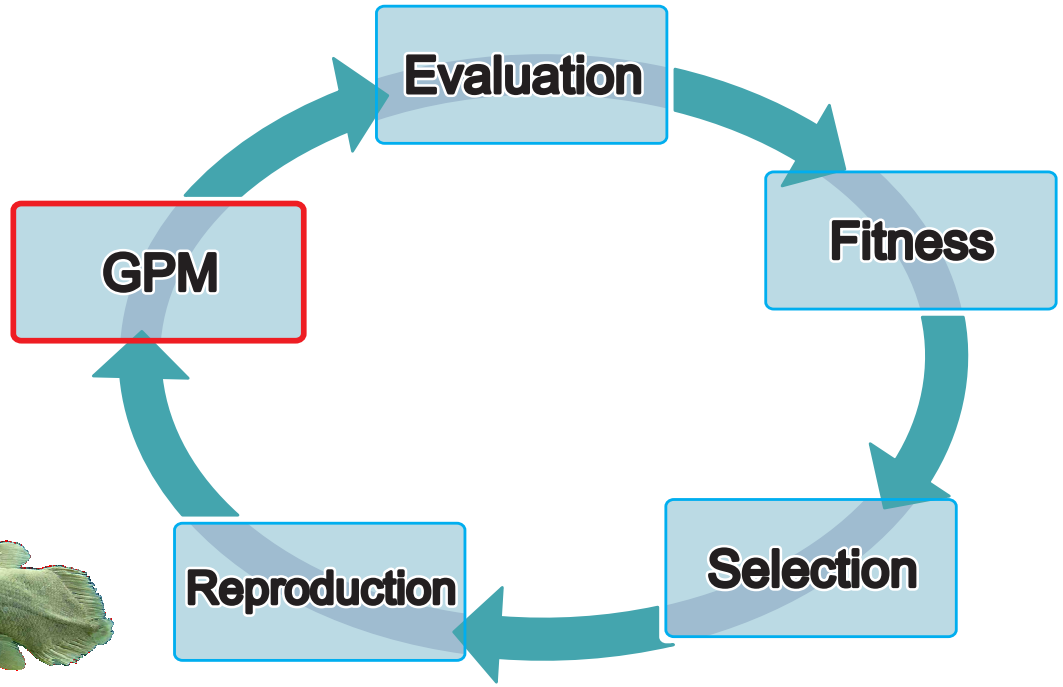


Evolution: Genotype-Phenotype Mapping

- Live begins with the development from genotype to phenotype

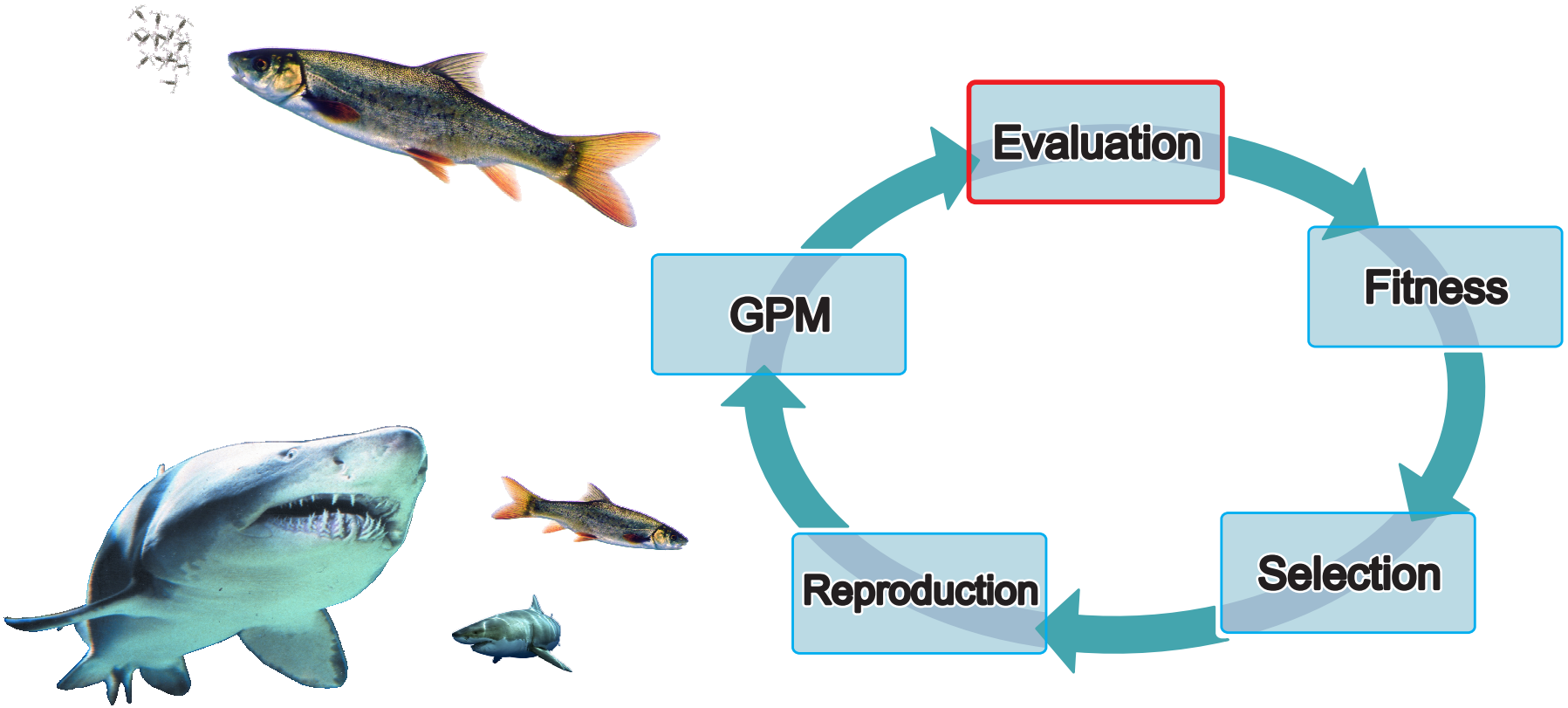






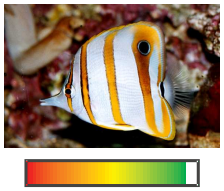
Evolution: Evaluation

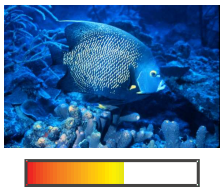
- Test the features of each individual

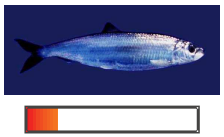


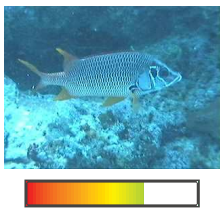
Evolution: Fitness

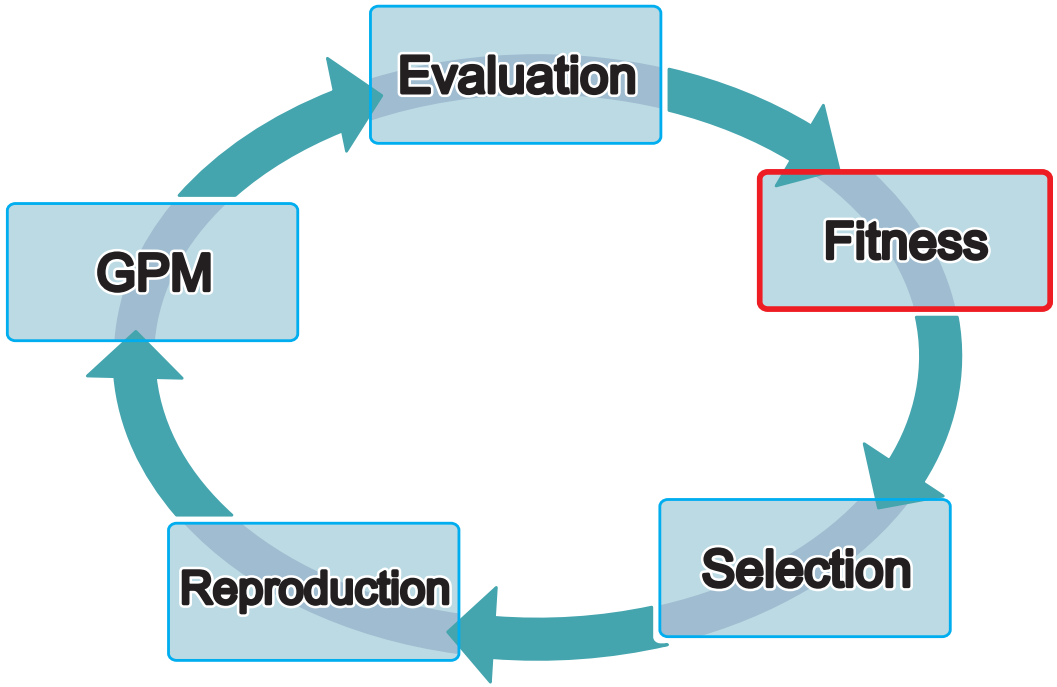
- Fitness is relative, determined/defined as number of offspring





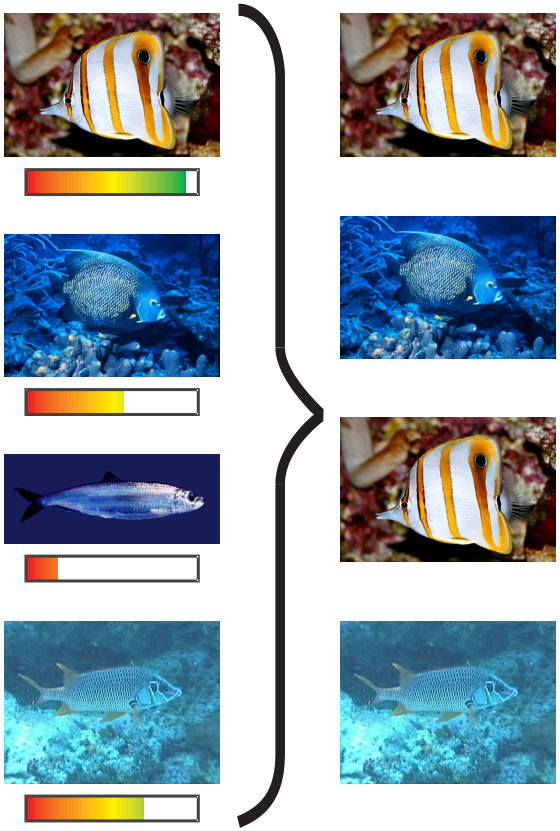


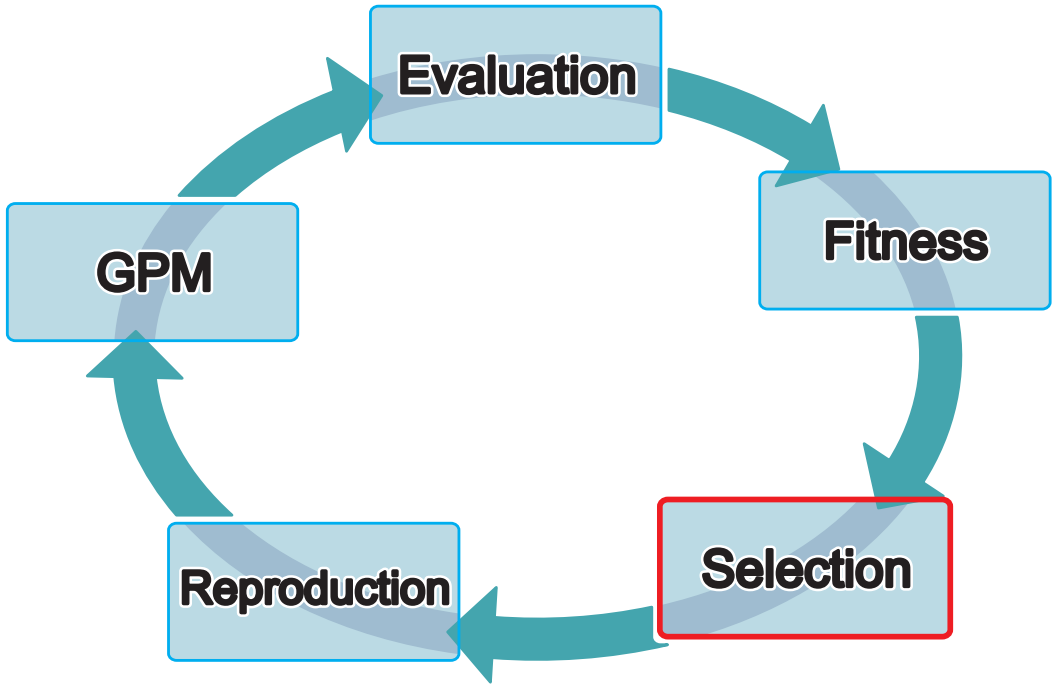




Evolution: Selection

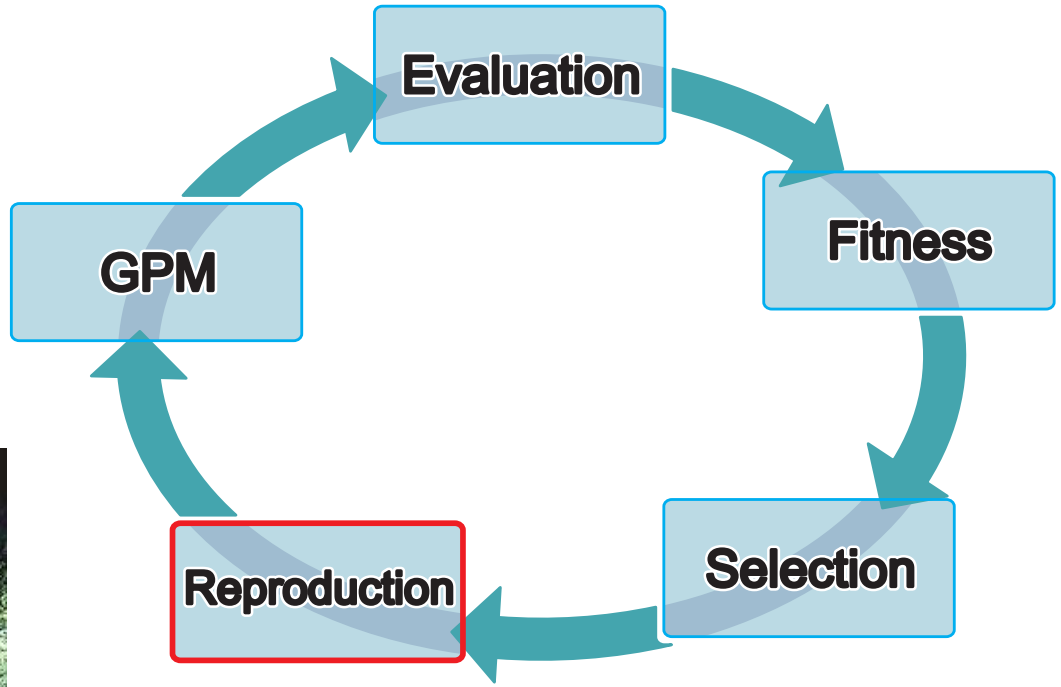
- Fitter individuals usually survive selection, have more offspring





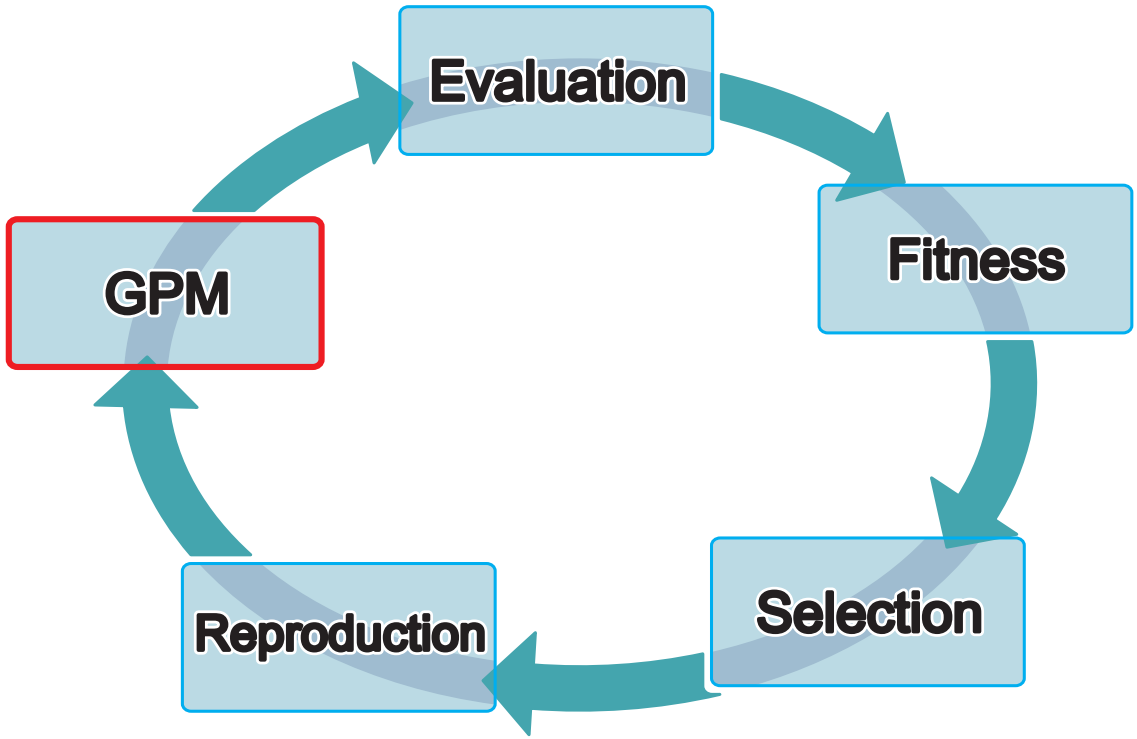
Evolution: Reproduction

- Asexual and sexual reproduction



Evolution: Next Generation

- Cycle starts again with the next "generation"



Section Outline

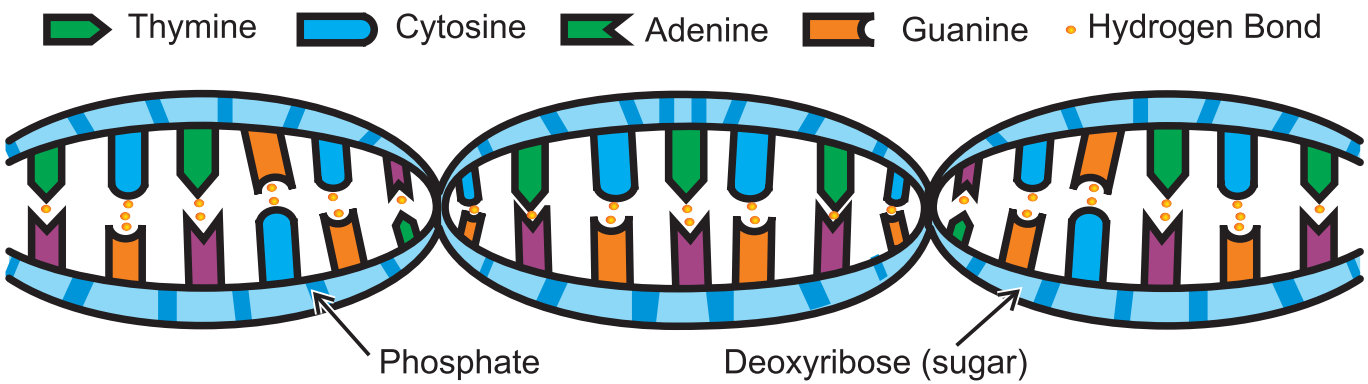
- In 1950s, biologists like Barricelli [2–5] and the computer scientist Fraser [6–8] begin to use computers to simulate evolution

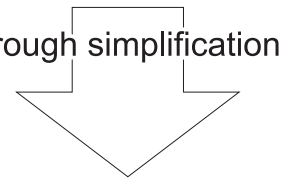
- In 1950s, biologists like Barricelli [2–5] and the computer scientist Fraser [6–8] begin to use computers to simulate evolution
- In the early 1960s, Bremermann [9] and Bledsoe [10–13] use simulated evolution to solve some optimization problems

- In 1950s, biologists like Barricelli [2–5] and the computer scientist Fraser [6–8] begin to use computers to simulate evolution
- In the early 1960s, Bremermann [9] and Bledsoe [10–13] use simulated evolution to solve some optimization problems
- In the late 1960s/early 1970s, Holland [14–17] formalizes Genetic Algorithms

- In 1950s, biologists like Barricelli [2–5] and the computer scientist Fraser [6–8] begin to use computers to simulate evolution
- In the early 1960s, Bremermann [9] and Bledsoe [10–13] use simulated evolution to solve some optimization problems
- In the late 1960s/early 1970s, Holland [14–17] formalizes Genetic Algorithms
- De Jong [18] uses them for function optimization

- Idea: Try to emulate the natural process of evolution on a very simple search space $\mathbb{G}$: the bit strings of length n, i.e., $\{\texttt{false}, \texttt{true}\}^n$



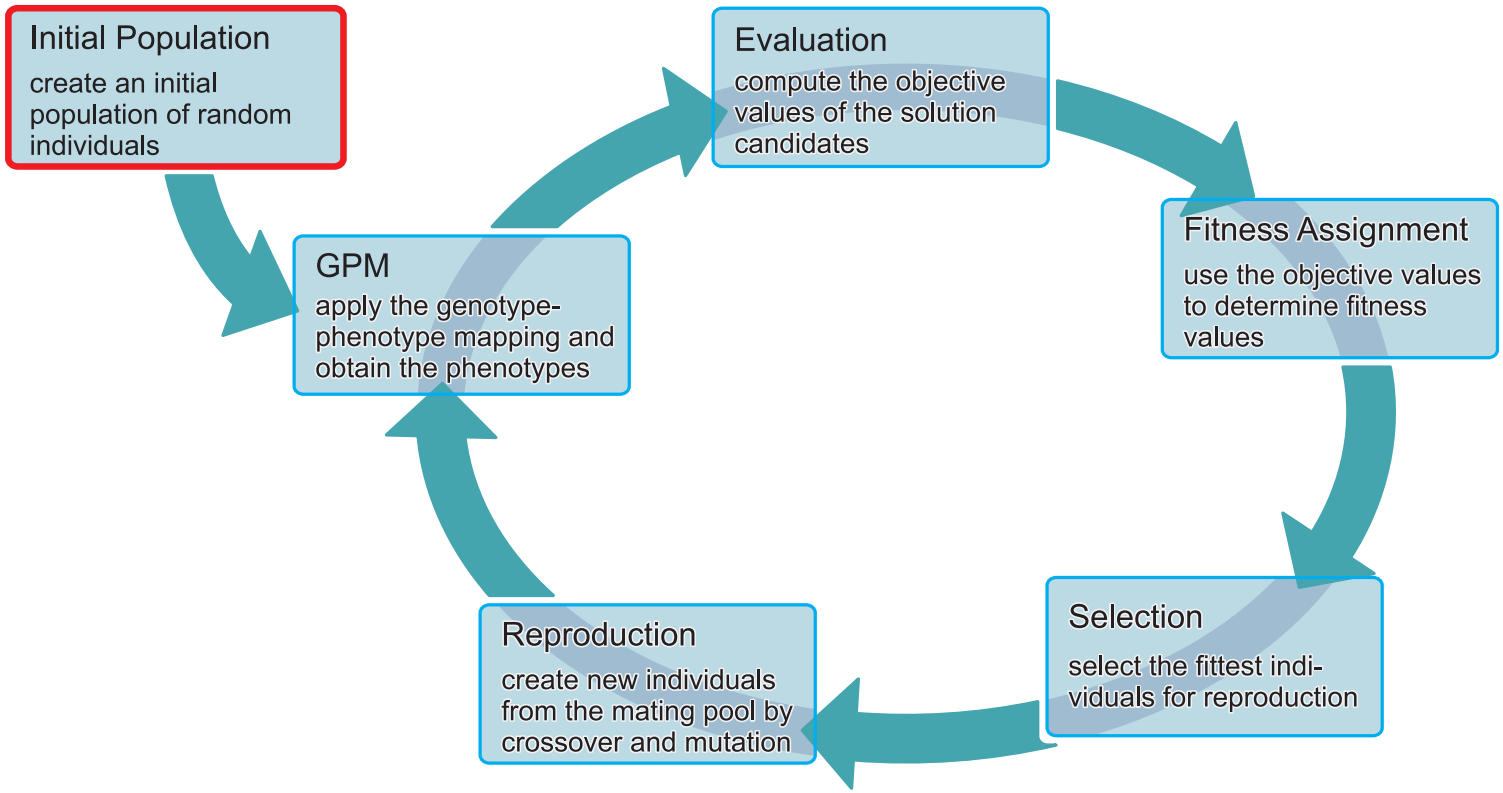


$g \in \mathbb{G}$ | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |

Search Space $\mathbb{G}$: all bit strings of a given length
Genotypes g: specific bit strings

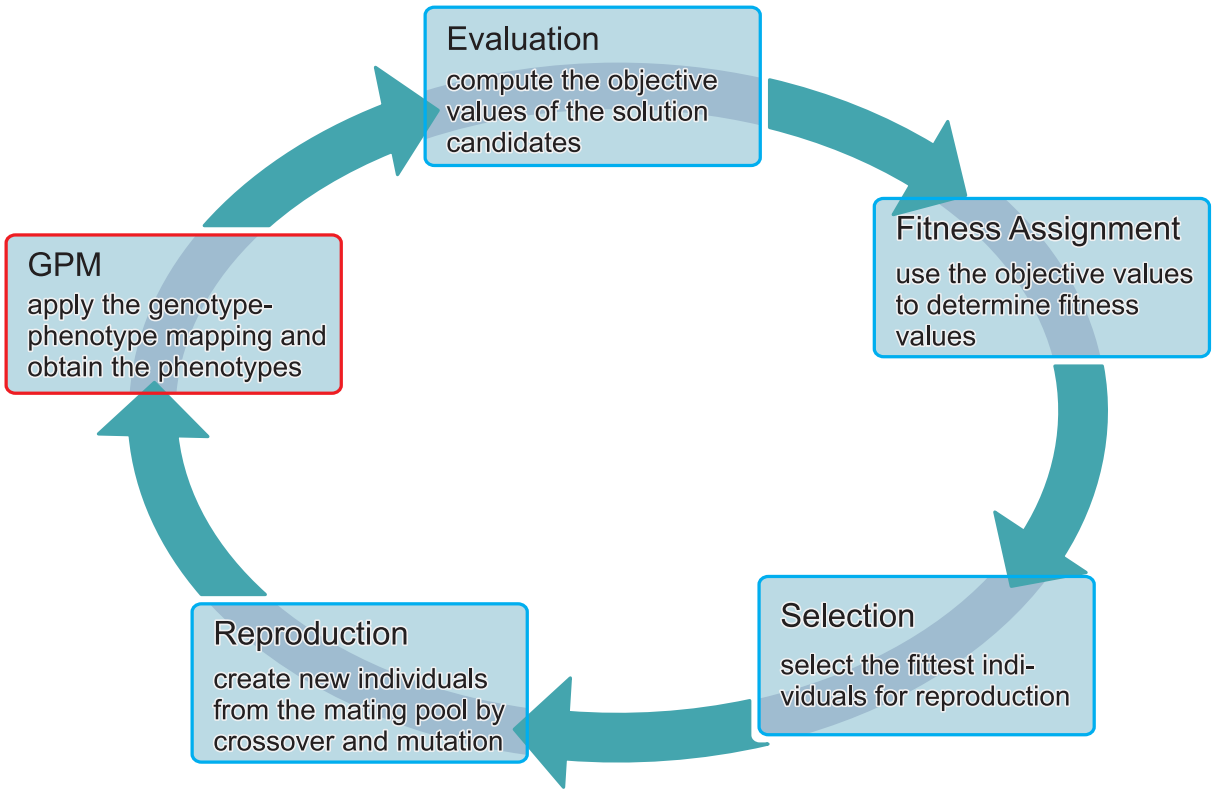
Genetic Algorithms: First Generation

- Each genotype is a bit string of length n
- Nullary search operation to create initial individuals: create a population of random bit strings



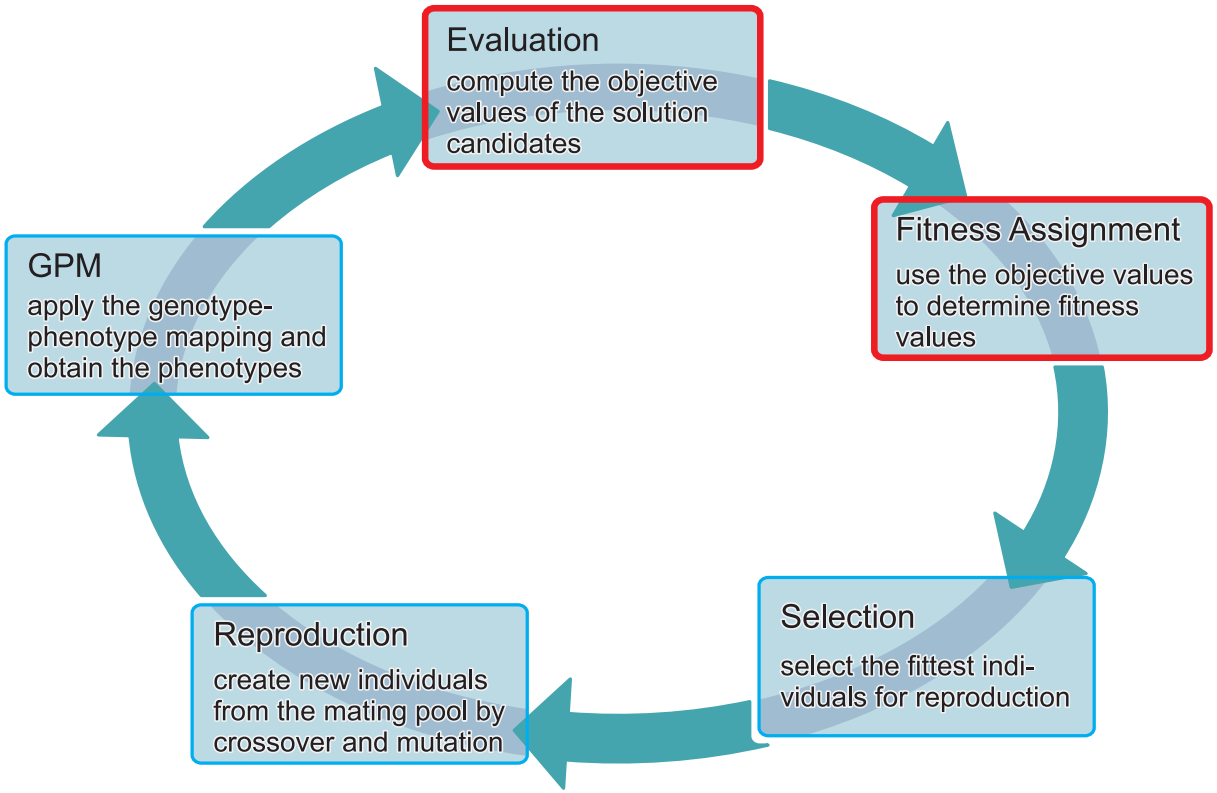
Genetic Algorithms: GPM

- Map the genotypes to phenotypes
- The GPM is usually problem-dependent



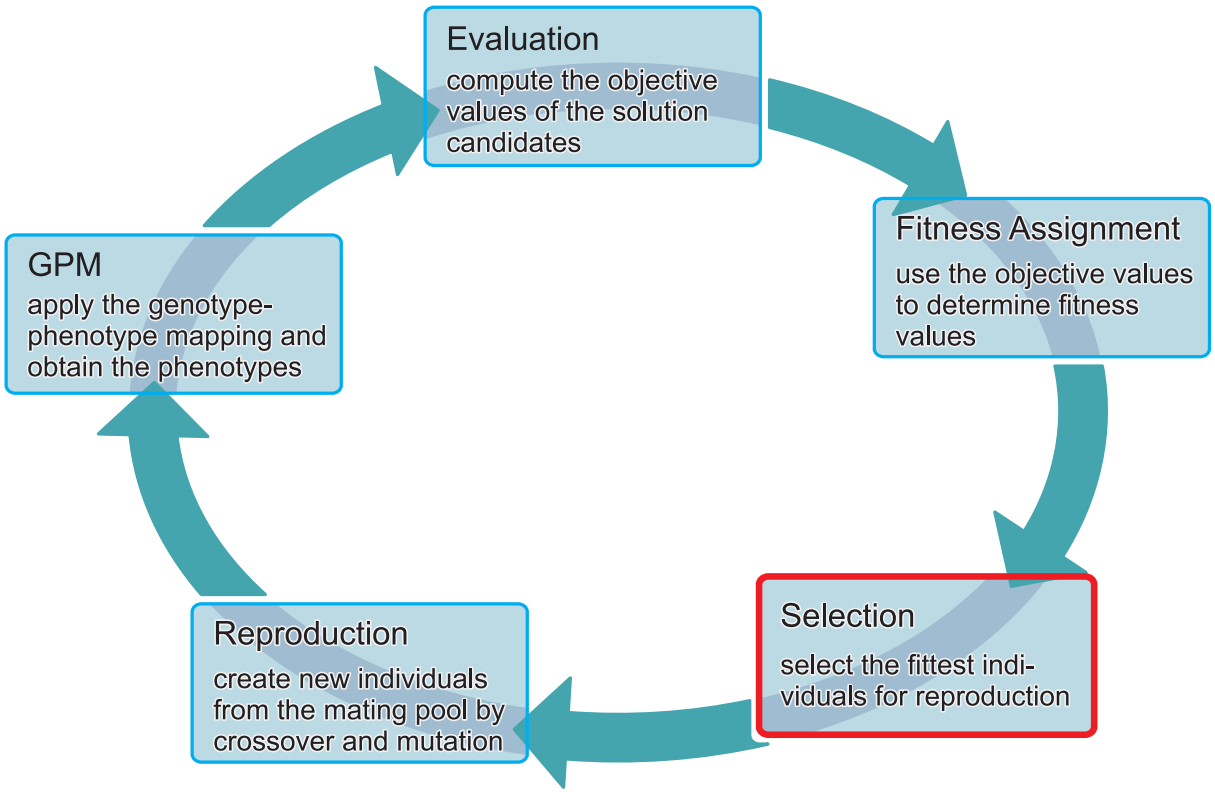
Genetic Algorithms: Evaluation

- Evaluate the objective function(s)
- In the original GA, fitness = objective values. In Multi-Objective Evolutionary Algorithms, this is not the case

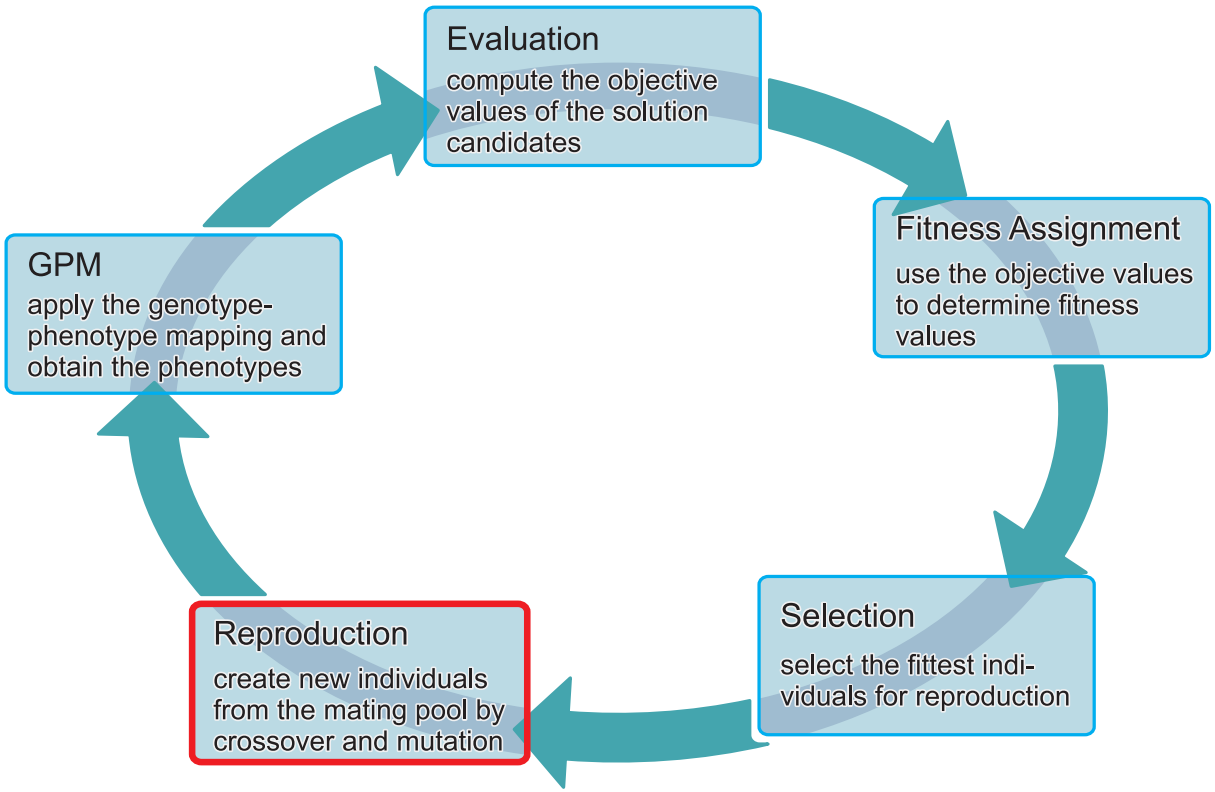


Genetic Algorithms: Selection

- Select the best individuals with highest probability
- Many different selection algorithms exist: Roulette-Wheel Selection, Tournament Selection, etc.

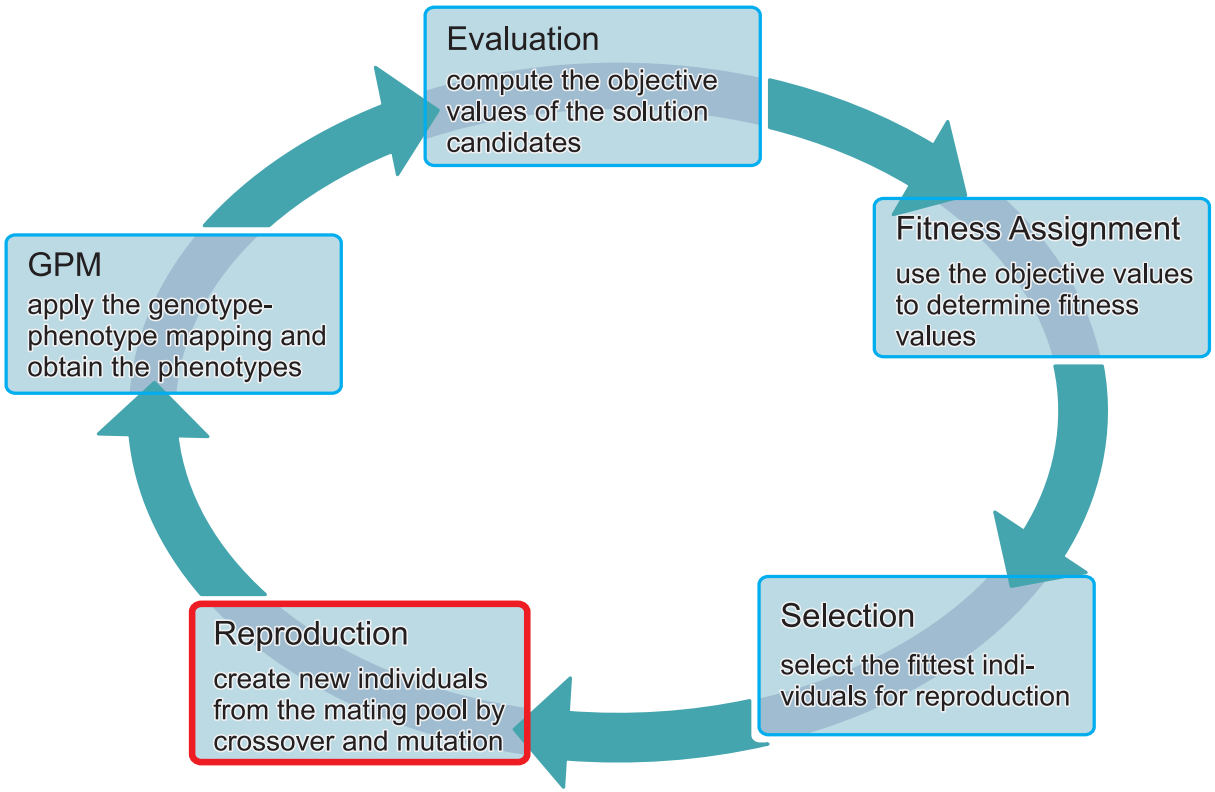


- Recombine genotypes: two genotypes are combined to create a new one (binary search operation); crossover rate cr
- Building Block Hypothesis: Good genes will aggregate [16, 19, 20]



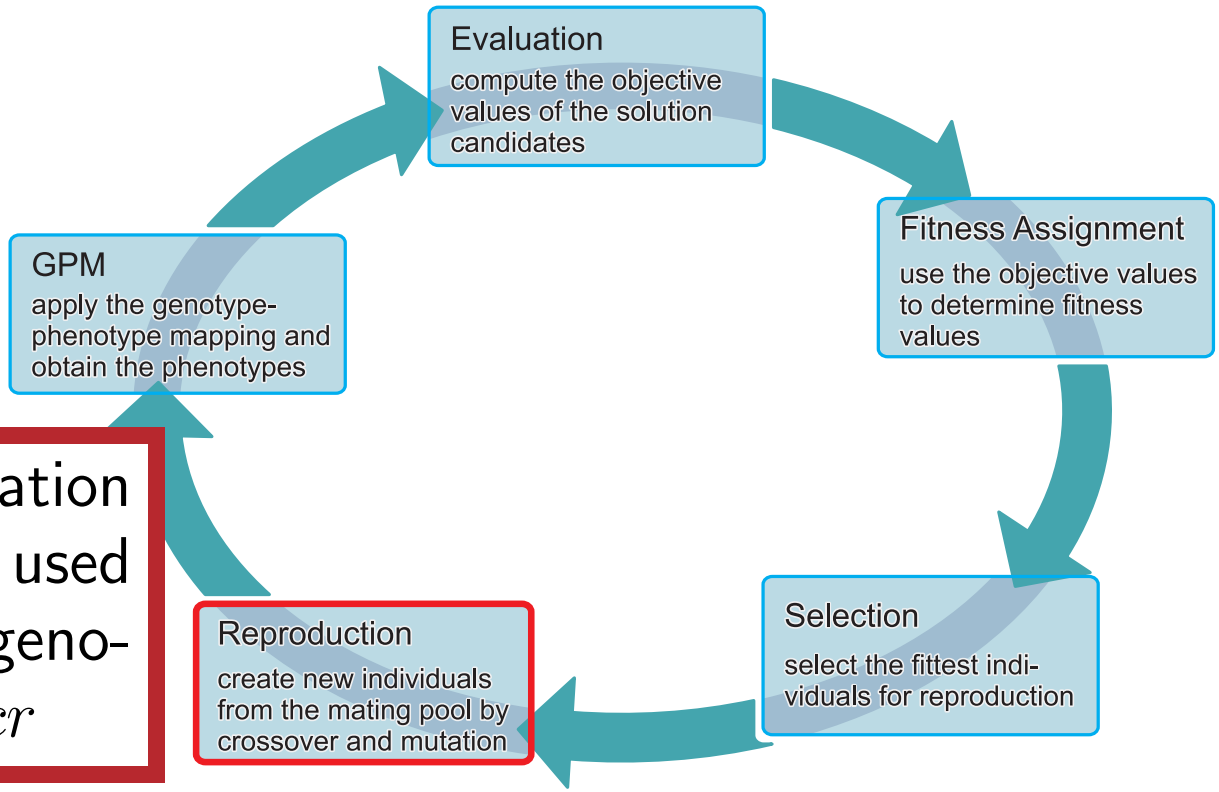
Genetic Algorithms: Mutation

- Perform mutation (unary search operation) with probability mr
- Slight perturbations to increase diversity in population



Genetic Algorithms: Mutation

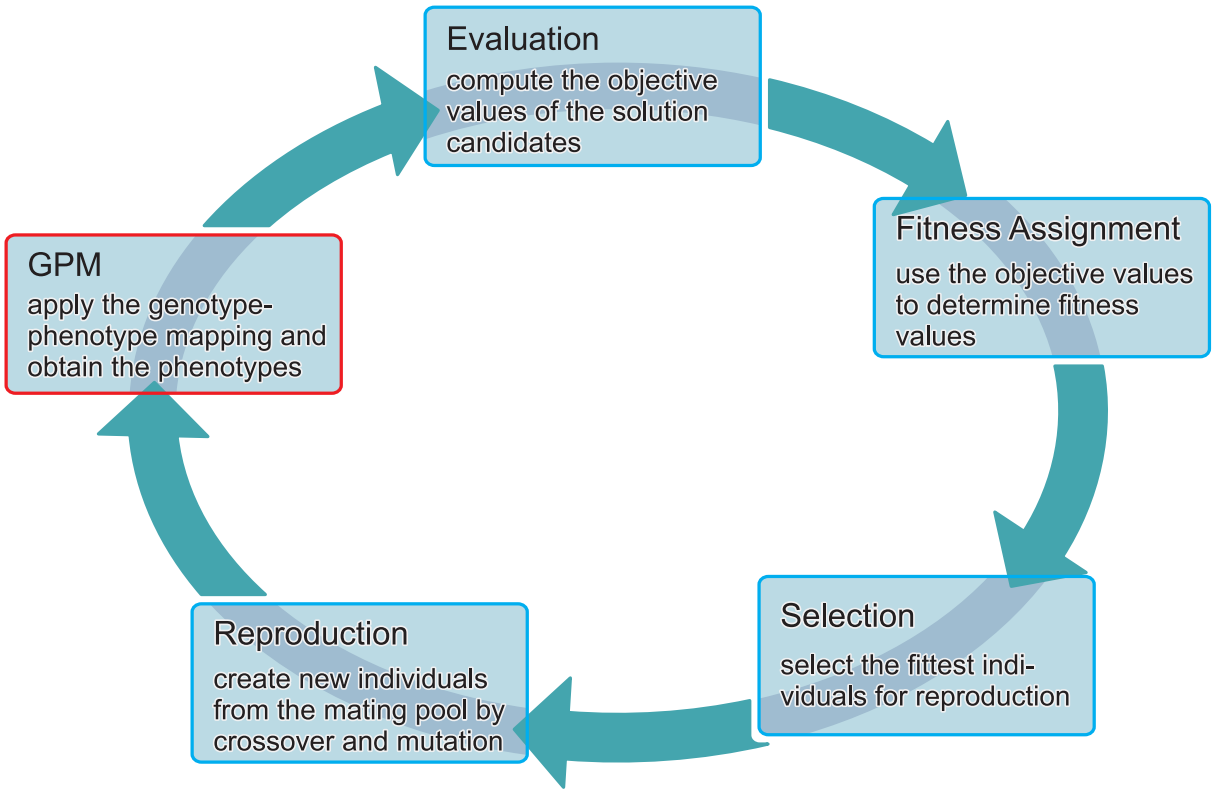
- Perform mutation (unary search operation) with probability mr
- Slight perturbations to increase diversity in population



Often either mutation or crossover is used to create a new genotype: $1 = mr + cr$

Genetic Algorithms: New Generation

- Start with new population in next generation.



Genetic Algorithms: Putting it Together

1. In the first generation ($t = 1$), a population pop of ps individuals p is created with the nullary search operation

Genetic Algorithms: Putting it Together

1. In the first generation ($t = 1$), a population pop of ps individuals p is created with the nullary search operation
2. the genotypes $p.g$ are translated to phenotypes $p.x$

Genetic Algorithms: Putting it Together

1. In the first generation ($t = 1$), a population pop of ps individuals p is created with the nullary search operation
2. the genotypes $p.g$ are translated to phenotypes $p.x$
3. compute objective value $f(p.x)$ of each candidate solution $p.x$ in pop

Genetic Algorithms: Putting it Together

1. In the first generation ($t = 1$), a population pop of ps individuals p is created with the nullary search operation
2. the genotypes $p.g$ are translated to phenotypes $p.x$
3. compute objective value $f(p.x)$ of each candidate solution $p.x$ in pop
4. perform selection: put mps individuals into the mating pool $\mathrm{matePool}$ with selection algorithm

Genetic Algorithms: Putting it Together

1. In the first generation ($t = 1$), a population pop of ps individuals p is created with the nullary search operation
2. the genotypes $p.g$ are translated to phenotypes $p.x$
3. compute objective value $f(p.x)$ of each candidate solution $p.x$ in pop
4. perform selection: put mps individuals into the mating pool $\mathrm{matePool}$ with selection algorithm
5. reproduce the individuals by using crossover and mutation, according to crossover rate cr

Genetic Algorithms: Putting it Together

1. In the first generation ($t = 1$), a population pop of ps individuals p is created with the nullary search operation
2. the genotypes $p.g$ are translated to phenotypes $p.x$
3. compute objective value $f(p.x)$ of each candidate solution $p.x$ in pop
4. perform selection: put mps individuals into the mating pool $\mathrm{matePool}$ with selection algorithm
5. reproduce the individuals by using crossover and mutation, according to crossover rate cr
6. Check the termination criterion (usually done after every objective function evaluation)

Listing: A simple Evolutionary Algorithm

```
public class EA<G, X> extends OptimizationAlgorithm<G, X> {
  public ISelectionAlgorithm selection;
  public int ps;
  public int mps;
  public double cr;
  public IBinarySearchOperation<G> binary;
  public EA() {
    super();
    this.cr = 0.3d;
    this.ps = 128;
    this.mps = 64;
    this.selection = TruncationSelection.INSTANCE;
  }
  public Individual<G, X> solve(final IObjectiveFunction<X> f) {
    Individual<G, X> pbest, pcur;
    Individual<G, X>[] pop, mate;
    int i;

    pbest = new Individual<>();
    pop = new Individual[this.ps];
    mate = new Individual[this.mps];

    for (i = pop.length; (--i) >= 0;) {
      pop[i] = pcur = new Individual<>();
      pcur.g = this.nullary.create(this.random);
    }

    for (;;) {
      for (i = pop.length; (--i) >= 0;) {
        pcur = pop[i];
        pcur.x = this.gpm.gpm(pcur.g);
        pcur.v = f.compute(pcur.x);
        if (pcur.v < pbest.v) {
          pbest.assign(pcur);
        }
        if (this.termination.shouldTerminate()) {
          return pbest;
        }
      }

      this.selection.select(pop, mate, this.random);

      for (i = pop.length; (--i) >= 0;) {
        pop[i] = pcur = new Individual<>();
        if (this.random.nextDouble() < this.cr) {
          pcur.g = this.binary.recombine(mate[i % mate.length].g,
              mate[this.random.nextInt(mate.length)].g, this.random);
        } else {
          pcur.g = this.unary.mutate(mate[i % mate.length].g, this.random);
        }
      }
    }
  }
}
```

Section Outline

Selection

- One of the unclear things so far: What are population, mating pool, and selection?

Selection

- One of the unclear things so far: What are population, mating pool, and selection?
- The population pop is the set of the ps solutions currently tested

Selection

- One of the unclear things so far: What are population, mating pool, and selection?
- The population pop is the set of the ps solutions currently tested
- We want that some of the best ones can reproduce

Selection

- One of the unclear things so far: What are population, mating pool, and selection?
- The population pop is the set of the ps solutions currently tested
- We want that some of the best ones can reproduce
- The mps individuals that can reproduce are placed into the mating pool $\mathrm{matePool}$

Selection

- One of the unclear things so far: What are population, mating pool, and selection?
- The population pop is the set of the ps solutions currently tested
- We want that some of the best ones can reproduce
- The mps individuals that can reproduce are placed into the mating pool $\mathrm{matePool}$
- *Selection* is the process of choosing which individuals from pop can enter$\mathrm{matePool}$

Selection

- One of the unclear things so far: What are population, mating pool, and selection?
- The population pop is the set of the ps solutions currently tested
- We want that some of the best ones can reproduce
- The mps individuals that can reproduce are placed into the mating pool $\mathrm{matePool}$
- *Selection* is the process of choosing which individuals from pop can enter$\mathrm{matePool}$
- Those individuals not selected are discarted

Selection

- One of the unclear things so far: What are population, mating pool, and selection?
- The population pop is the set of the ps solutions currently tested
- We want that some of the best ones can reproduce
- The mps individuals that can reproduce are placed into the mating pool $\mathrm{matePool}$
- *Selection* is the process of choosing which individuals from pop can enter$\mathrm{matePool}$
- Those individuals not selected are discarted
- Two common, basic approaches to choose mps, independent from how selection is done:

Selection

- One of the unclear things so far: What are population, mating pool, and selection?
- The population pop is the set of the ps solutions currently tested
- We want that some of the best ones can reproduce
- The mps individuals that can reproduce are placed into the mating pool $\mathrm{matePool}$
- *Selection* is the process of choosing which individuals from pop can enter$\mathrm{matePool}$
- Those individuals not selected are discarted
- Two common, basic approaches to choose mps, independent from how selection is done:
  1. $ps > mps$: only a few individuals are selected and these have multiple offspring

Selection

- One of the unclear things so far: What are population, mating pool, and selection?
- The population pop is the set of the ps solutions currently tested
- We want that some of the best ones can reproduce
- The mps individuals that can reproduce are placed into the mating pool $\mathrm{matePool}$
- *Selection* is the process of choosing which individuals from pop can enter$\mathrm{matePool}$
- Those individuals not selected are discarted
- Two common, basic approaches to choose mps, independent from how selection is done:
  1. $ps > mps$: only a few individuals are selected and these have multiple offspring
  2. $ps = mps$: each selected individual has one offspring, an individual can be selected multiple times

Selection

- population pop has ps individuals

Selection

- population pop has ps individuals
- We want to keep and modify only some of them

Selection

- population pop has ps individuals
- We want to keep and modify only some of them
- We will select mps individuals and put them into the mating pool $\mathrm{matePool}$

Selection

- population pop has ps individuals
- We want to keep and modify only some of them
- We will select mps individuals and put them into the mating pool $\mathrm{matePool}$
- There exist multiple selection algorithms which can be used for this purpose

Selection

- population pop has ps individuals
- We want to keep and modify only some of them
- We will select mps individuals and put them into the mating pool $\mathrm{matePool}$
- There exist multiple selection algorithms which can be used for this purpose
- General basis for selection: Fitness ν.

Selection

- population pop has ps individuals
- We want to keep and modify only some of them
- We will select mps individuals and put them into the mating pool $\mathrm{matePool}$
- There exist multiple selection algorithms which can be used for this purpose
- General basis for selection: Fitness ν.
- In the most simple, single-objective case (only one objective function, i.e., what we did so far):

Selection

- population pop has ps individuals
- We want to keep and modify only some of them
- We will select mps individuals and put them into the mating pool $\mathrm{matePool}$
- There exist multiple selection algorithms which can be used for this purpose
- General basis for selection: Fitness ν.
- In the most simple, single-objective case (only one objective function, i.e., what we did so far):

$$\nu(p) = f(p.x) \quad \forall \text{individuals} \tag{1}$$

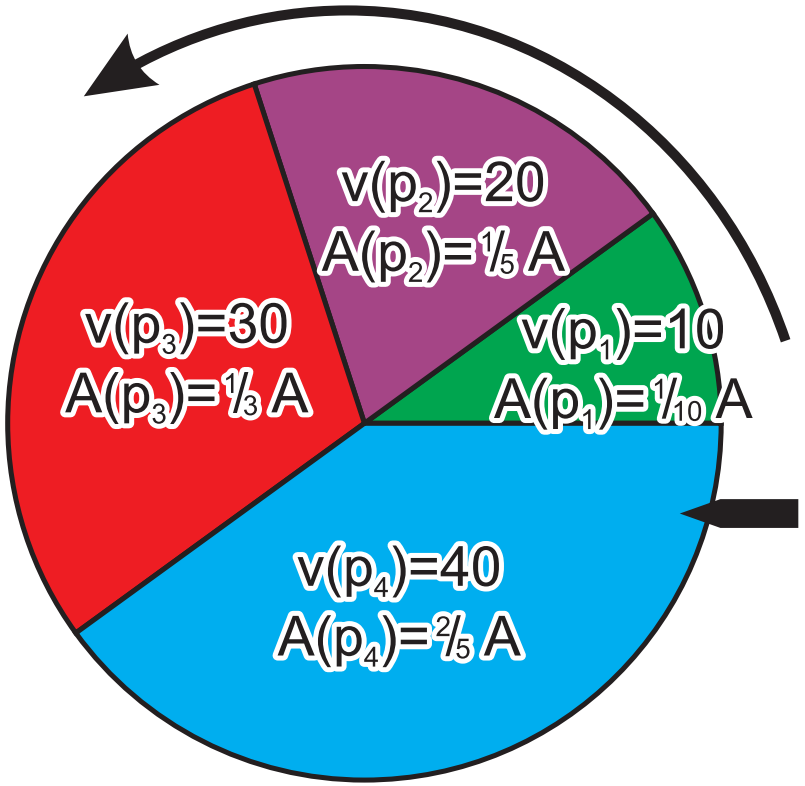
Listing: The Selection Algorithm: Programmer's Perspective

```
package metaheuristicOptimization.algorithms.ea;

import java.util.Random;

import metaheuristicOptimization.Individual;

/** the interface for selection algorithms */
public interface ISelectionAlgorithm {
  /**
   * Fill the mating pool with selected individuals from the population
   *
   * @param pop
   *          the population of the current individuals
   * @param mate
   *          the mating pool to be filled with individuals
   * @param r
   *          the random number generator
   */
  public abstract void select(final Individual<?, ?>[] pop, final
      Individual<?, ?>[] mate,
      final Random r);
}
```

Traditional: Roulette-Wheel Selection

- Traditional method: *Roulette-Wheel Selection* – Number of offspring is proportional to fitness (fitness is maximized!) [16, 18, 21–25]

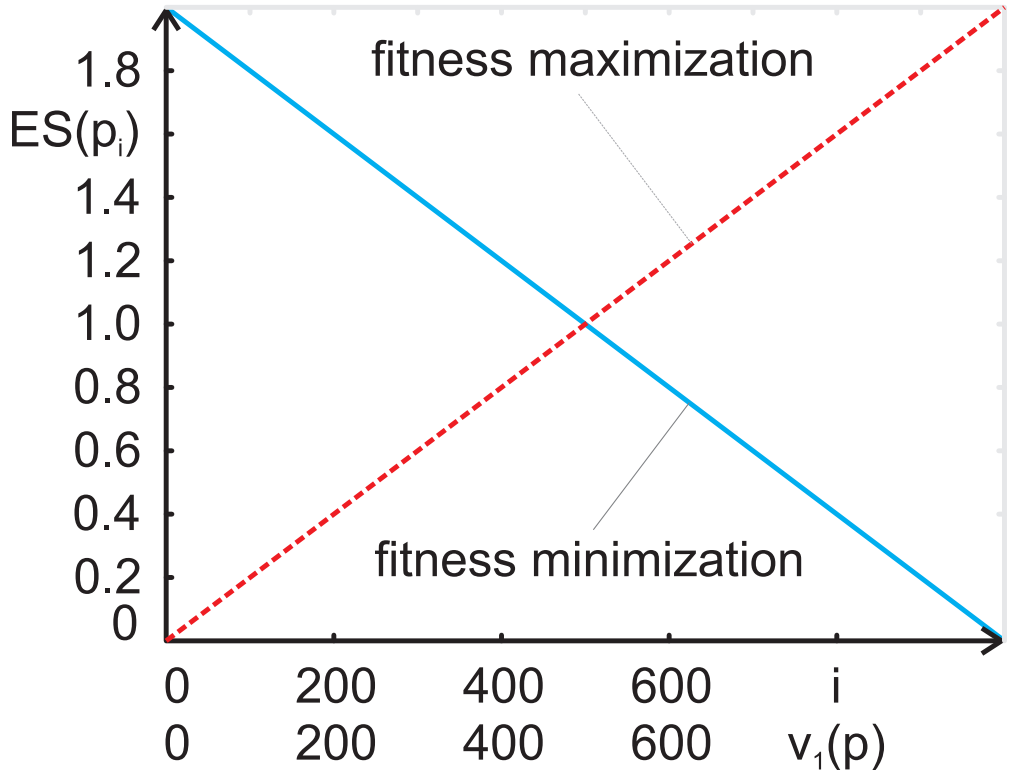
$$P(\text{select}(p)) = \frac{f(p.x)}{\sum_{\forall p' \in \text{pop}} f(p'.x)} \quad (2)$$



Traditional: Roulette-Wheel Selection

- Traditional selection algorithm.

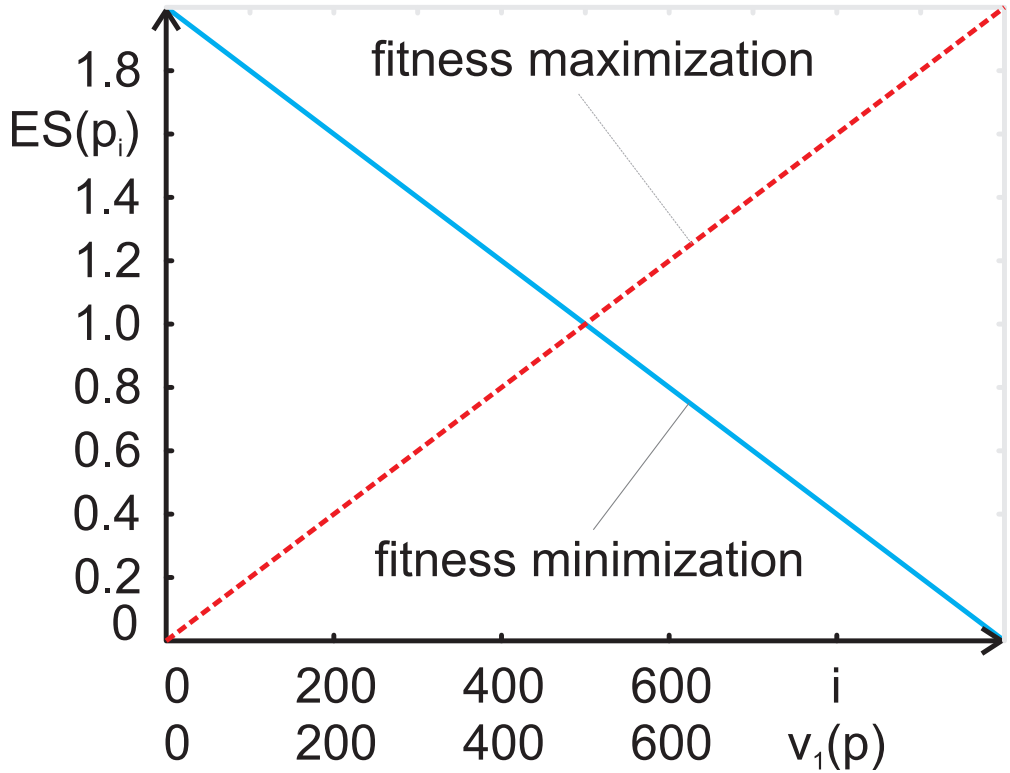
If we had 1000 individuals p_i with fitness i, the expected number $ES(p_i)$ of times individual p_i enters the mating pool is. . .



Traditional: Roulette-Wheel Selection

- Traditional selection algorithm.
- Roulette-Wheel Selection: Number of offspring is proportional to fitness (fitness is maximized!) [16, 18, 21–25]

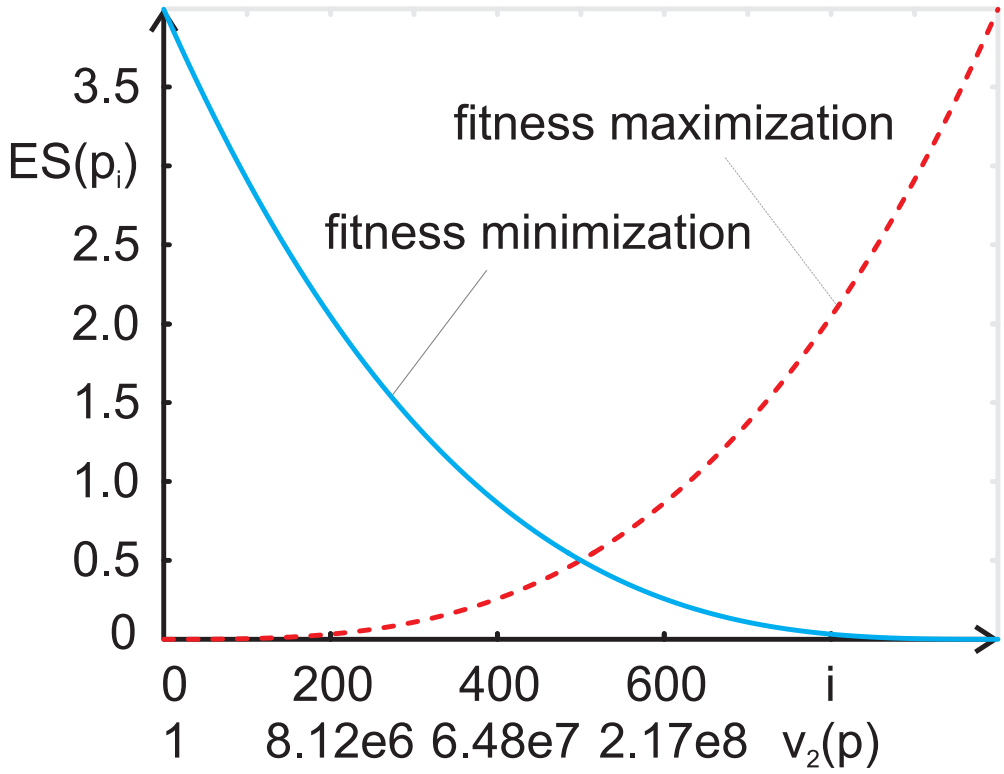
If we had 1000 individuals p_i with fitness i, the expected number $ES(p_i)$ of times individual p_i enters the mating pool is. . .



Traditional: Roulette-Wheel Selection

- Roulette-Wheel Selection: Number of offspring is proportional to fitness (fitness is maximized!) [16, 18, 21–25]

If we had 1000 individuals p_i with fitness $(i+1)^3$, the expected number $ES(p_i)$ of times individual p_i enters the mating pool is...

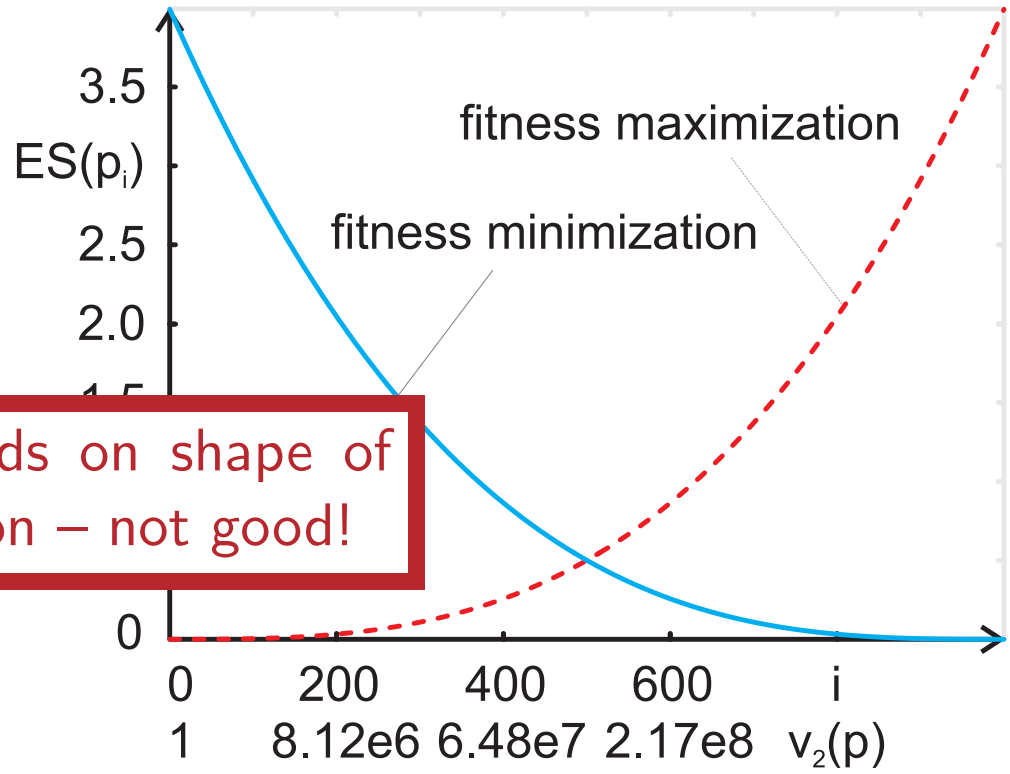


Traditional: Roulette-Wheel Selection

- Roulette-Wheel Selection: Number of offspring is proportional to fitness (fitness is maximized!) [16, 18, 21–25]

If we had 1000 individuals p_i with fitness $(i+1)^3$, the expected number $ES(p_i)$ of times individual p_i ente pool is...

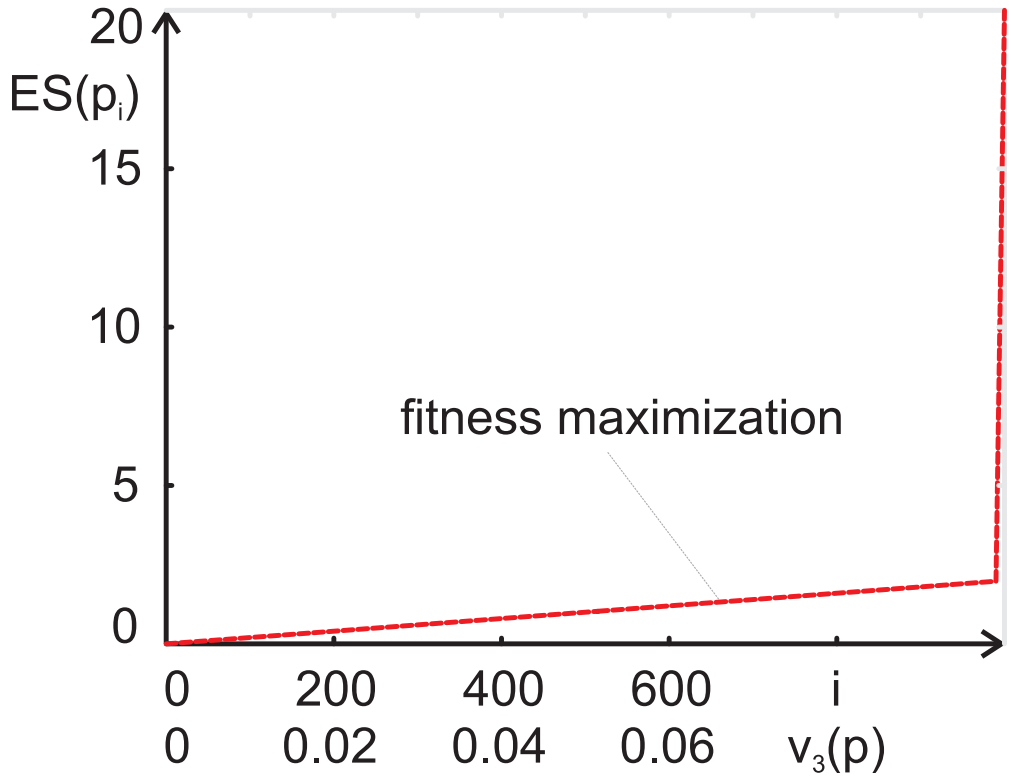
Selection depends on shape of objective function – not good!



Traditional: Roulette-Wheel Selection

- Roulette-Wheel Selection: Number of offspring is proportional to fitness (fitness is maximized!) [16, 18, 21–25]

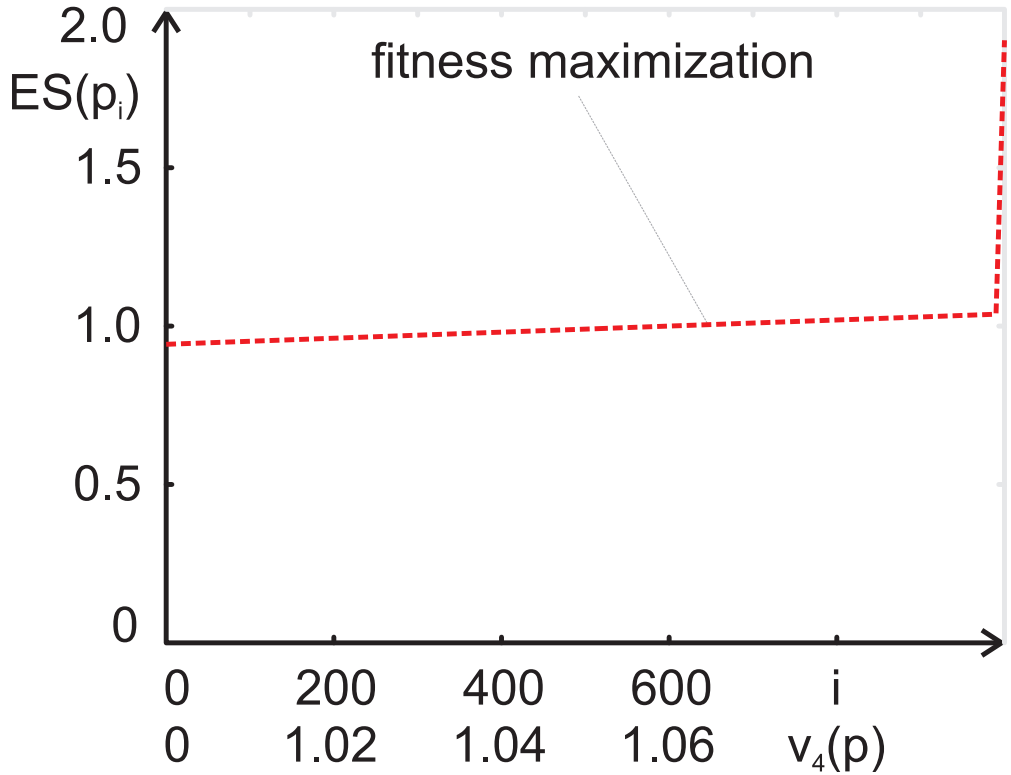
If we had 1000 individuals p_i with fitness $0.0001i$ for $i \in 0..998$ and 1 for $i = 1000$, the expected number $ES(p_i)$ of times individual p_i enters the mating pool is...



Traditional: Roulette-Wheel Selection

- Roulette-Wheel Selection: Number of offspring is proportional to fitness (fitness is maximized!) [16, 18, 21–25]

If we had 1000 individuals p_i with fitness $1 + 0.0001i$ for $i \in 0..998$ and 2 for $i = 1000$, the expected number $ES(p_i)$ of times individual p_i enters the mating pool is completely different.

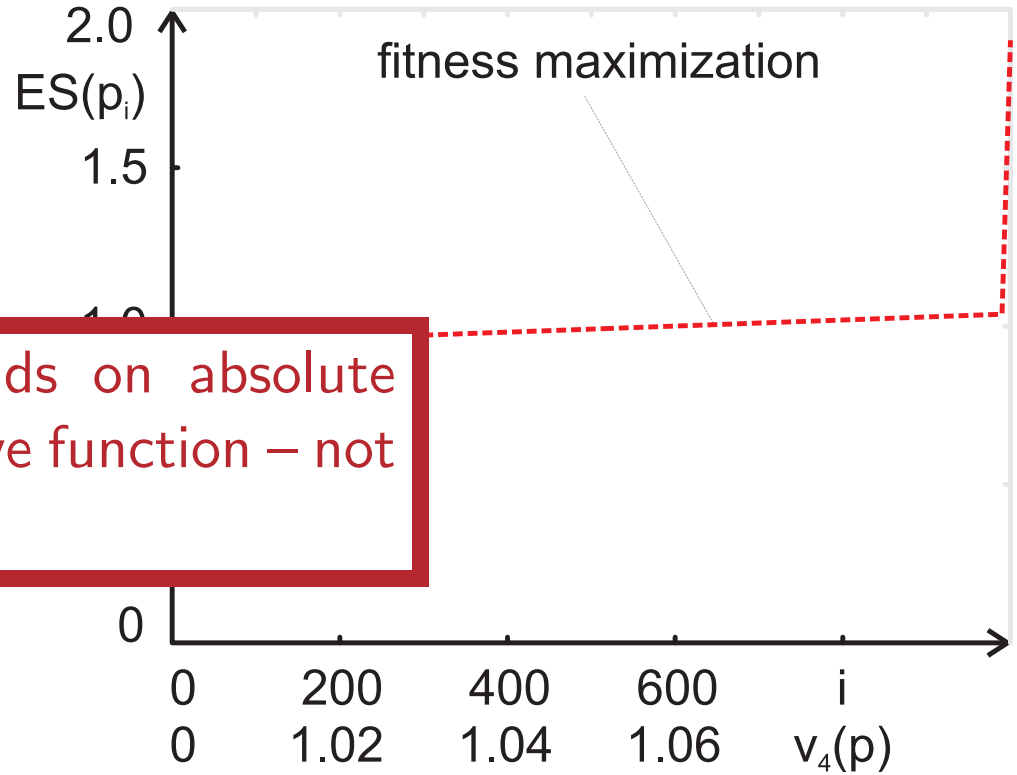


Traditional: Roulette-Wheel Selection

- Roulette-Wheel Selection: Number of offspring is proportional to fitness (fitness is maximized!) [16, 18, 21–25]

If we had 1000 individuals p_i with fitness $1 + 0.0001i$ for $i \in 0..998$ and 2 fo the expected num times individual mating pool is completely different.

Selection depends on absolute offset of objective function – not good!



Traditional: Roulette-Wheel Selection

- Roulette-Wheel Selection: Fitness-proportionate selection
- Traditional selection method

Traditional: Roulette-Wheel Selection

- Roulette-Wheel Selection: Fitness-proportionate selection
- Traditional selection method
- Not robust! Influenced on

Traditional: Roulette-Wheel Selection

- Roulette-Wheel Selection: Fitness-proportionate selection
- Traditional selection method
- Not robust! Influenced on:
  1. shape of objective function

Traditional: Roulette-Wheel Selection

- Roulette-Wheel Selection: Fitness-proportionate selection
- Traditional selection method
- Not robust! Influenced on:
  1. shape of objective function
  2. big-$\mathcal{O}$ class of objective function

Traditional: Roulette-Wheel Selection

- Roulette-Wheel Selection: Fitness-proportionate selection
- Traditional selection method
- Not robust! Influenced on:
  1. shape of objective function
  2. big-$\mathcal{O}$ class of objective function
  3. absolute offset of function

Traditional: Roulette-Wheel Selection

- Roulette-Wheel Selection: Fitness-proportionate selection
- Traditional selection method
- Not robust! Influenced on:
  1. shape of objective function
  2. big-$\mathcal{O}$ class of objective function
  3. absolute offset of function
- An optimization algorithm should work good regardless whether we add an offset to an objective function or use the squared functions...

Traditional: Roulette-Wheel Selection

- Roulette-Wheel Selection: Fitness-proportionate selection
- Traditional selection method
- Not robust! Influenced on:
  1. shape of objective function
  2. big-$\mathcal{O}$ class of objective function
  3. absolute offset of function
- An optimization algorithm should work good regardless whether we add an offset to an objective function or use the squared functions...
- ...but Roulette-Wheel Selection gives different results in these cases!

Traditional: Roulette-Wheel Selection

- Roulette-Wheel Selection: Fitness-proportionate selection
- Traditional selection method
- Not robust! Influenced on:
  1. shape of objective function
  2. big-$\mathcal{O}$ class of objective function
  3. absolute offset of function
- An optimization algorithm should work good regardless whether we add an offset to an objective function or use the squared functions...
- ...but Roulette-Wheel Selection gives different results in these cases!
- Avoid to use this directly

Traditional: Roulette-Wheel Selection

- Roulette-Wheel Selection: Fitness-proportionate selection
- Traditional selection method
- Not robust! Influenced on:
  1. shape of objective function
  2. big-$\mathcal{O}$ class of objective function
  3. absolute offset of function
- An optimization algorithm should work good regardless whether we add an offset to an objective function or use the squared functions...
- ...but Roulette-Wheel Selection gives different results in these cases!
- Avoid to use this directly
- Good only if fitness "*fits*" to this method, as e.g., in Pareto ranking [19, 26, 27]

matePool $\longleftarrow$ rouletteWheelSelection(pop)mps

Input: pop: the list of individuals to select from
Input: [implicit] ps: the population size
Input: mps: the number of individuals to be placed into the mating pool matePool
Output: matePool: the mating pool

begin
- $A \longleftarrow$ create empty list
- $max \longleftarrow -\infty$
- `// Initialize fitness array and find extreme fitnesses`
- **for** $i \longleftarrow 0$ **up to** $ps - 1$ **do**
  - $a \longleftarrow \text{pop}[i].y$
  - $A[i] \longleftarrow a$
  - **if** $a > max$ **then** $max \longleftarrow a$
- $sum \longleftarrow 0$
- **for** $i \longleftarrow 0$ **up to** $ps - 1$ **do**
  - $sum \longleftarrow sum + (max - A[i])$
  - $A[i] \longleftarrow sum$
- **for** $i \longleftarrow 0$ **up to** $mps - 1$ **do**
  - $a \longleftarrow$ {randomly from $[0, sum]$}
  - append $\text{pop}[\max\{i : A[i] \leq a\}]$ to matePool
- **return** $matePool$

Listing: The Roulette-Wheel Selection Algorithm

```
public class RouletteWheelSelection implements ISelectionAlgorithm {
  public void select(final Individual<?, ?>[] pop, final Individual<?, ?>[] mate, final Random r) {
    double[] t;
    double max, last;
    int i, j;

    t = this.temp;
    if ((t == null) || (t.length < pop.length)) {
      this.temp = t = new double[pop.length];
    }

    max = Double.NEGATIVE_INFINITY;
    for (Individual<?, ?> indi : pop) {
      max = Math.max(indi.v, max);
    }

    max = Math.nextUp(max);
    last = 0d;
    for (i = 0; i < t.length; i++) {
      last += (max - pop[i].v);
      t[i] = last;
    }
    t[t.length - 1] = Double.POSITIVE_INFINITY;

    for (i = 0; i < mate.length; i++) {
      j = Arrays.binarySearch(t, last * r.nextDouble());
      if (j < 0) {
        j = ((-j) - 1);
      }
      mate[i] = pop[j];
    }
  }
}
```

Tournament Selection

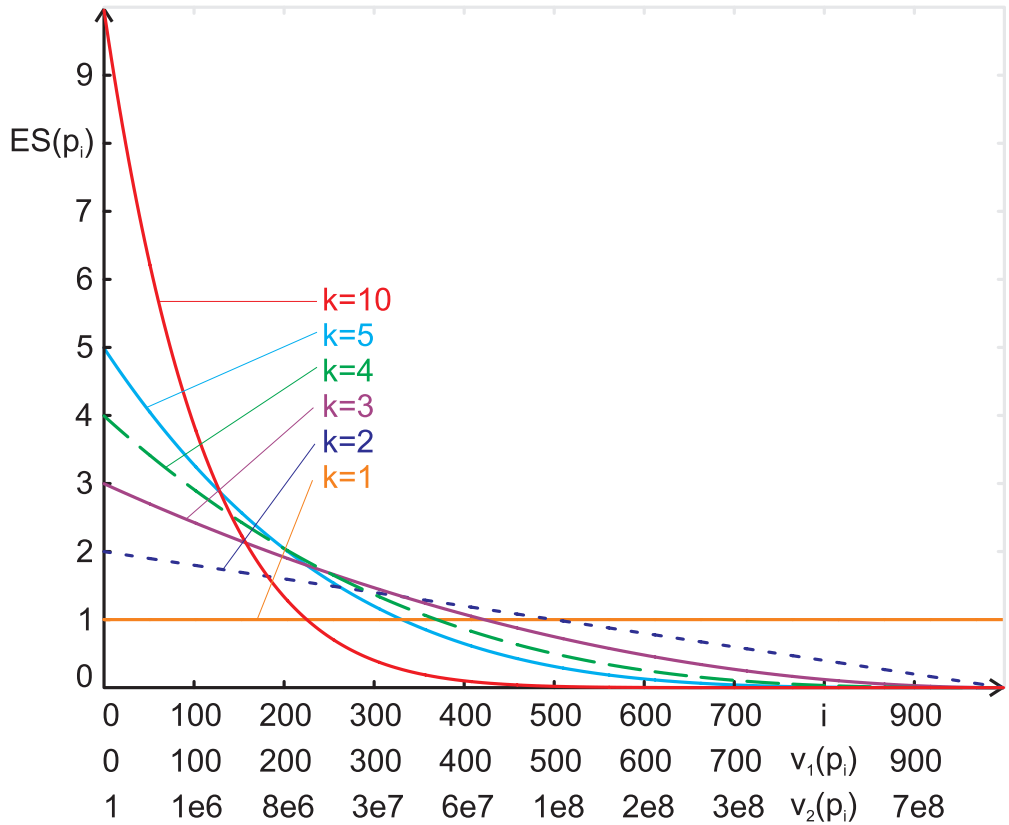
- *Tournament Selection*: k individuals compete to produce 1 offspring, the best wins!

Tournament Selection

- *Tournament Selection*: k individuals compete to produce 1 offspring, the best wins!
- k randomly picked contestants compete for each slot in the mating pool – the best gets it [22, 28–34]

- *Tournament Selection*: k individuals compete to produce 1 offspring, the best wins!
- k randomly picked contestants compete for each slot in the mating pool – the best gets it [22, 28–34]

If we had 1000 individuals p_i with fitness i; $(i+1)^3$; $0.0001i$ for $i \in 0..998$ and 1 for $i = 1000$; or $1 + 0.0001i$ for $i \in 0..998$ and 2 for $i = 1000$, the expected number $ES(p_i)$ of times individual p_i enters the mating pool is always the same!

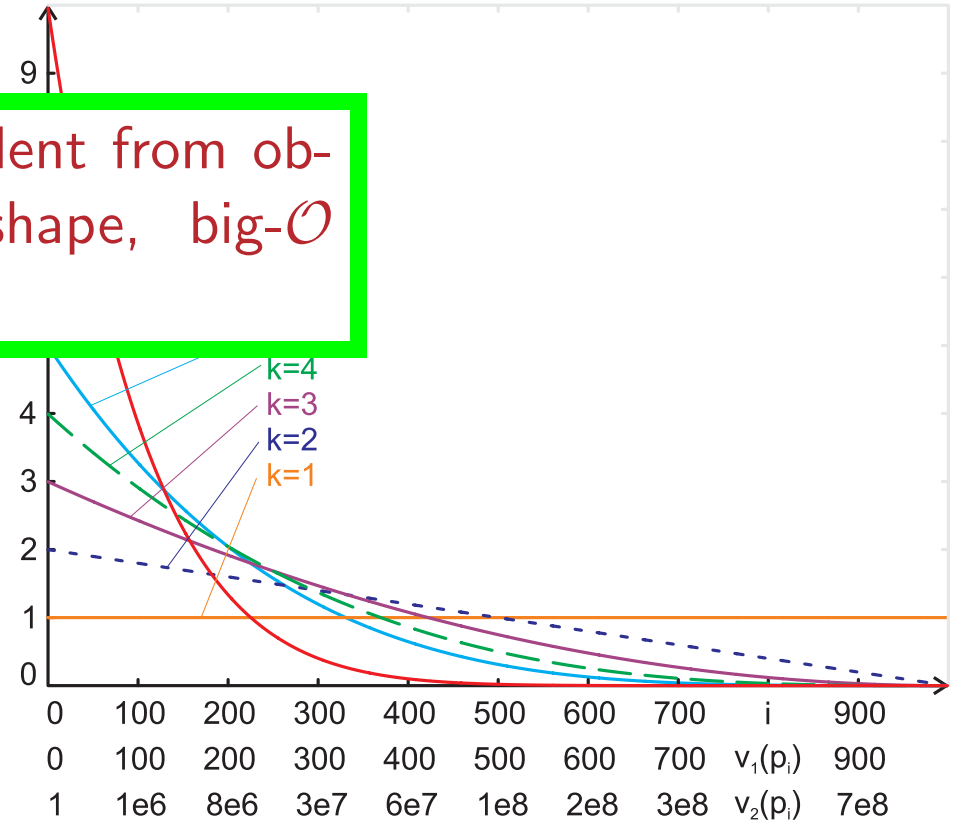


Tournament Selection

- *Tournament Selection*: k individuals compete to produce 1 offspring, the best wins!
- k randomly picked contestants compete for each slot in the mating pool – the best gets it [22, 28–34]

If we had 1000 with fitness i; $(i$ for $i \in 0..998$ and 1 for $i = 1000$; or $1 + 0.0001i$ for $i \in 0..998$ and 2 for $i = 1000$, the expected number $ES(p_i)$ of times individual p_i enters the mating pool is always the same!

Selection independent from objective function shape, big-$\mathcal{O}$ class, or offset



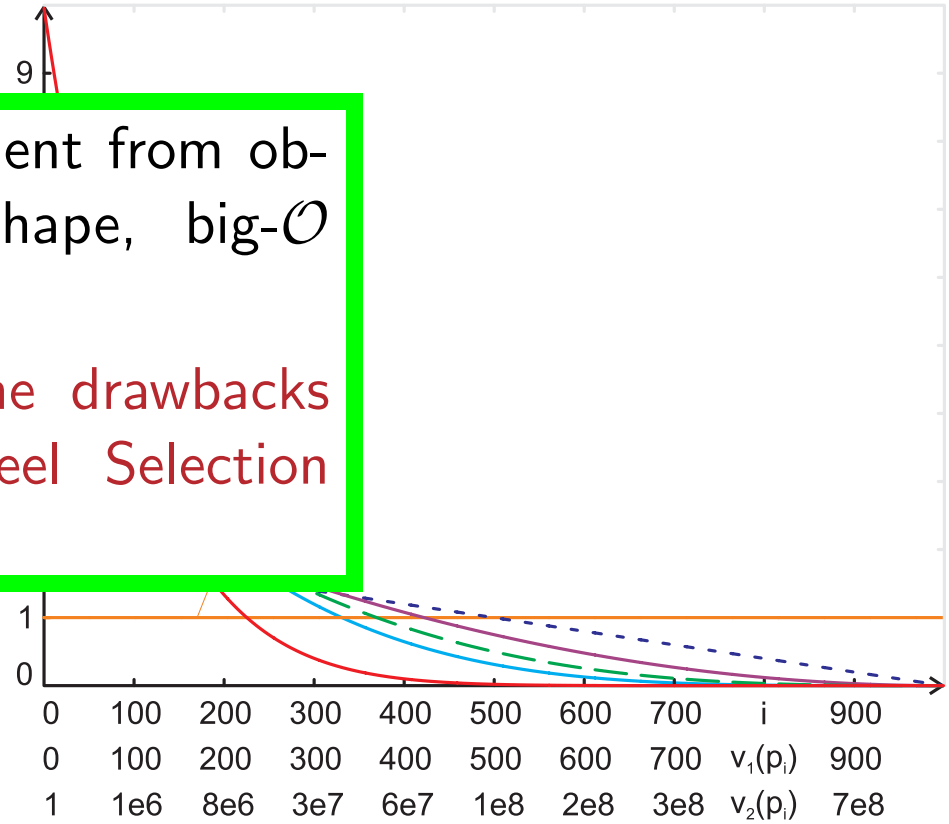
Tournament Selection

- *Tournament Selection*: k individuals compete to produce 1 offspring, the best wins!
- k randomly picked contestants compete for each slot in the mating pool – the best gets it [22, 28–34]

If we had 1000 with fitness i; $(i$ for $i \in 0..998$ a 1000; or $1 + 0.($ $0..998$ and 2 fo the expected number $ES(p_i)$ of times individual p_i enters the mating pool is always the same!

Selection independent from objective function shape, big-$\mathcal{O}$ class, or offset

Does not have the drawbacks that Roulette-Wheel Selection has!



Tournament Selection

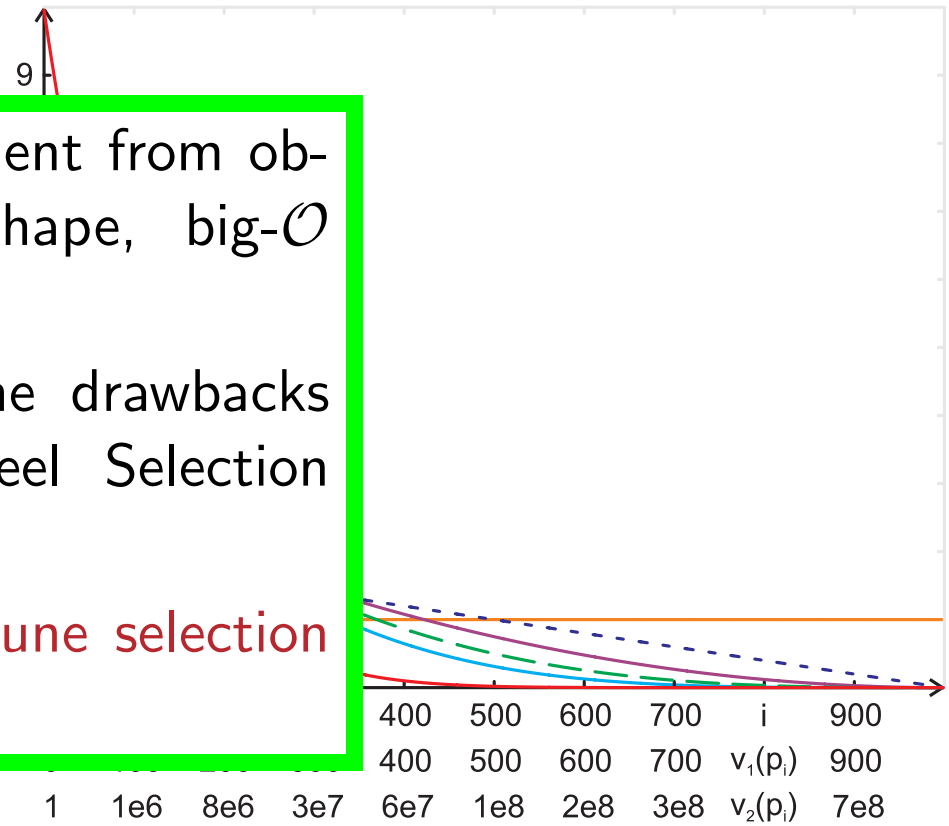
- *Tournament Selection*: k individuals compete to produce 1 offspring, the best wins!
- k randomly picked contestants compete for each slot in the mating pool – the best gets it [22, 28–34]

If we had 1000 with fitness i; $(i$ for $i \in 0..998$ a 1000; or $1 + 0.($ $0..998$ and 2 fo the expected num times individual mating pool is alv

Selection independent from objective function shape, big-$\mathcal{O}$ class, or offset

Does not have the drawbacks that Roulette-Wheel Selection has!

Plus we can fine-tune selection pressure via k



matePool ⟵ tournamentSelection(k, pop, mps)

Input: pop: the list of individuals to select from
Input: [implicit] ps: the population size
Input: mps: the number of individuals to be placed into the mating pool matePool
Input: [implicit] k: the tournament size
Output: matePool: the winners of the tournaments which now form the mating pool

begin
- matePool ⟵ create empty list
- **for** $i \longleftarrow 1$ **up to** mps **do**
  - $a \longleftarrow$ {randomly from $0..sum-1$}
  - **for** $j \longleftarrow 1$ **up to** $k-1$ **do**
    - $b \longleftarrow$ {randomly from $0..sum-1$}
    - **if** $pop[b].y < pop[a].y$ **then** $a \longleftarrow b$
  - append pop[a] to matePool
- **return** *matePool*

Listing: The Tournament Selection Algorithm

```
public class TournamentSelection implements ISelectionAlgorithm {
  public void select(final Individual<?, ?>[] pop, final Individual<?, ?>[]
      mate, final Random r) {
    int i, j;
    Individual<?, ?> x, y;

    for (i = 0; i < mate.length; i++) {
      x = pop[r.nextInt(pop.length)];
      for (j = 1; j < this.k; j++) {
        y = pop[r.nextInt(pop.length)];
        if (y.v < x.v) {
          x = y;
        }
      }
      mate[i] = x;
    }
  }
}
```

Truncation Selection

- Choose the mps best individuals from the population pop into the mating pool $\mathrm{matePool}$

Truncation Selection

- Choose the mps best individuals from the population pop into the mating pool $\mathrm{matePool}$
- Here, the mating pool $\mathrm{matePool}$ is always smaller than the population pop ($mps < ps$)

Truncation Selection

- Choose the mps best individuals from the population pop into the mating pool $\mathrm{matePool}$
- Here, the mating pool $\mathrm{matePool}$ is always smaller than the population pop ($mps < ps$)
- Extremely simple to implement

Truncation Selection

- Choose the mps best individuals from the population pop into the mating pool $\mathrm{matePool}$
- Here, the mating pool $\mathrm{matePool}$ is always smaller than the population pop ($mps < ps$)
- Extremely simple to implement
- Also used in Evolution Strategies [35–41] (see Lesson 12: *Evolution Strategies*)

Truncation Selection

- Choose the mps best individuals from the population pop into the mating pool $\mathrm{matePool}$
- Here, the mating pool $\mathrm{matePool}$ is always smaller than the population pop ($mps < ps$)
- Extremely simple to implement
- Also used in Evolution Strategies [35–41] (see Lesson 12: *Evolution Strategies*)
- Lässig et al. [42, 43] show that this selection strategy is optimal!

Truncation Selection

- Choose the mps best individuals from the population pop into the mating pool $\mathrm{matePool}$
- Here, the mating pool $\mathrm{matePool}$ is always smaller than the population pop ($mps < ps$)
- Extremely simple to implement
- Also used in Evolution Strategies [35–41] (see Lesson 12: *Evolution Strategies*)
- Lässig et al. [42, 43] show that this selection strategy is optimal!
- However, the right mating pool size mps is not known...

Truncation Selection

matePool $\longleftarrow$ truncationSelection(μ, pop)

Input: pop: the list of individuals to select from (length λ or $\mu + \lambda$)
Input: μ: the number of individuals to be placed into the mating pool matePool
Output: matePool: the survivors of the truncation which now form the mating pool

begin
    sort the pop according to fitness (best first)
    return first μ individuals from pop

Truncation Selection

Listing: The Truncation Selection Algorithm

```
public class TruncationSelection implements ISelectionAlgorithm {
  public void select(final Individual<?, ?>[] pop, final Individual<?, ?>[]
      mate, final Random r) {
    Arrays.sort(pop);
    System.arraycopy(pop, 0, mate, 0, mate.length);
  }
}
```

Population Treatment

- Three different ways of population treatment

Population Treatment

- Three different ways of population treatment
  1. Generational: only the offspring of a generation survive

Population Treatment

- Three different ways of population treatment
  1. Generational: only the offspring of a generation survive
  2. steady-state: offspring and older generations compete

Population Treatment

- Three different ways of population treatment
  1. Generational: only the offspring of a generation survive
  2. steady-state: offspring and older generations compete
  3. Elitism: Either steady-state OR generationals PLUS the best solutions survive always

Section Outline

Crossover

- Genetic Algorithms: crossover is a binary search operation

Crossover

- Genetic Algorithms: crossover is a binary search operation
- Idea:
  - two individuals have been selected (as parents)

Crossover

- Genetic Algorithms: crossover is a binary search operation
- Idea:
  - two individuals have been selected (as parents)
  - thus, we can assume that both have good features

Crossover

- Genetic Algorithms: crossover is a binary search operation
- Idea:
  - two individuals have been selected (as parents)
  - thus, we can assume that both have good features
  - if the population is *diverse*, then the two selected individuals probably have different features

Crossover

- Genetic Algorithms: crossover is a binary search operation
- Idea:
  - two individuals have been selected (as parents)
  - thus, we can assume that both have good features
  - if the population is *diverse*, then the two selected individuals probably have different features
- Goal:

Crossover

- Genetic Algorithms: crossover is a binary search operation
- Idea:
  - two individuals have been selected (as parents)
  - thus, we can assume that both have good features
  - if the population is *diverse*, then the two selected individuals probably have different features
- Goal:
  - Combine these different (good) features...

Crossover

- Genetic Algorithms: crossover is a binary search operation
- Idea:
  - two individuals have been selected (as parents)
  - thus, we can assume that both have good features
  - if the population is *diverse*, then the two selected individuals probably have different features
- Goal:
  - Combine these different (good) features...
  - ...and obtain a new, possible better candidate solution

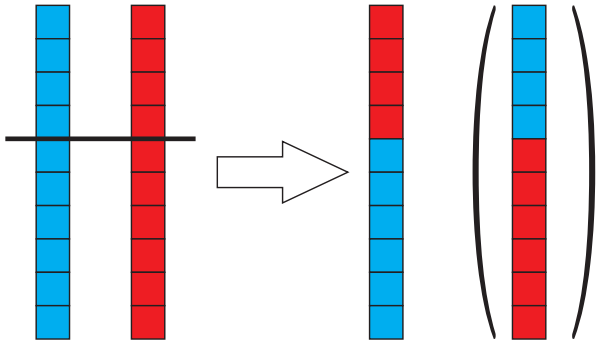
Crossover

- Genetic Algorithms: crossover is a binary search operation
- Idea:
  - two individuals have been selected (as parents)
  - thus, we can assume that both have good features
  - if the population is *diverse*, then the two selected individuals probably have different features
- Goal:
  - Combine these different (good) features...
  - ...and obtain a new, possible better candidate solution
- There exist different crossover operators

Crossover

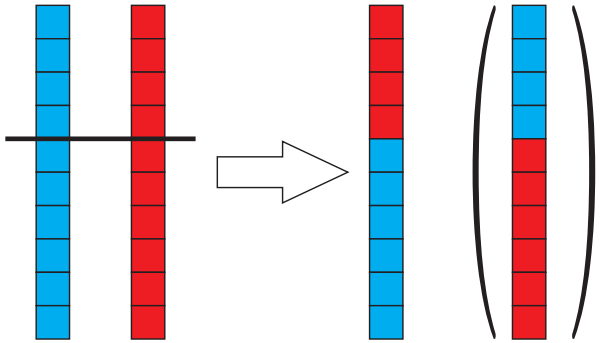
- Genetic Algorithms: crossover is a binary search operation
- Idea:
  - two individuals have been selected (as parents)
  - thus, we can assume that both have good features
  - if the population is *diverse*, then the two selected individuals probably have different features
- Goal:
  - Combine these different (good) features...
  - ...and obtain a new, possible better candidate solution
- There exist different crossover operators
- Building Block Hypothesis: Good genes/features will aggregate [16, 19, 20]

Crossover: SPX

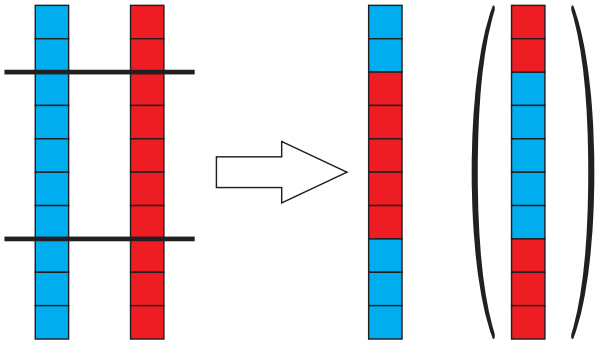


Single-Point (SPX)

Crossover: TPX

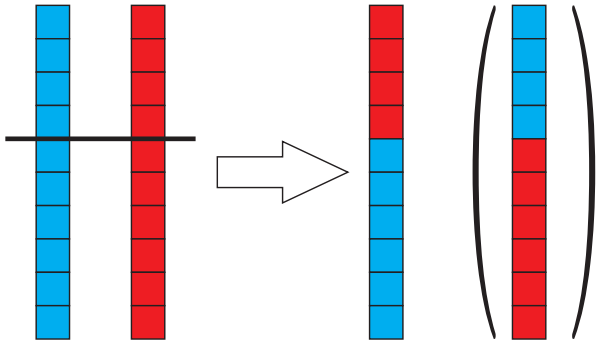


Single-Point (SPX)

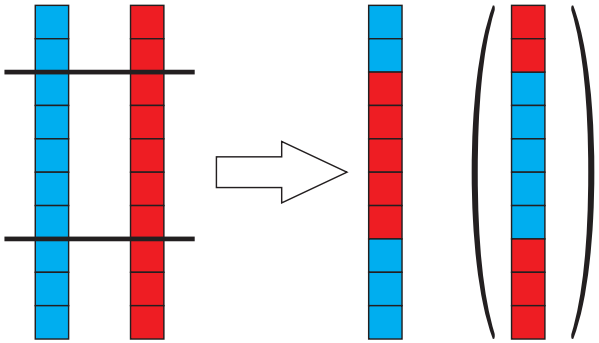


Two-Point (TPX)

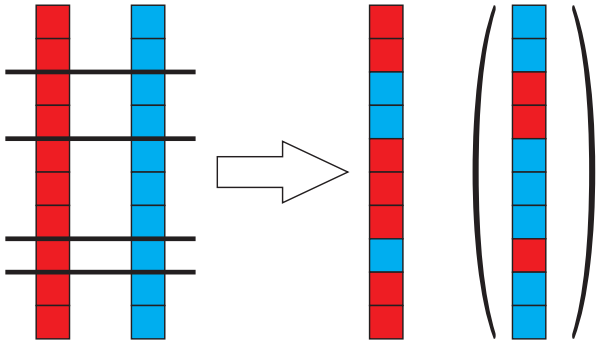
Crossover: MPX



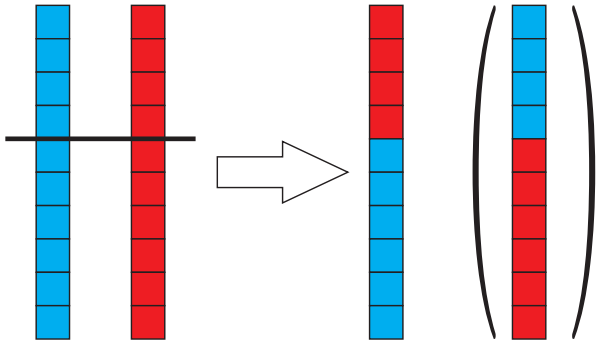
Single-Point (SPX)



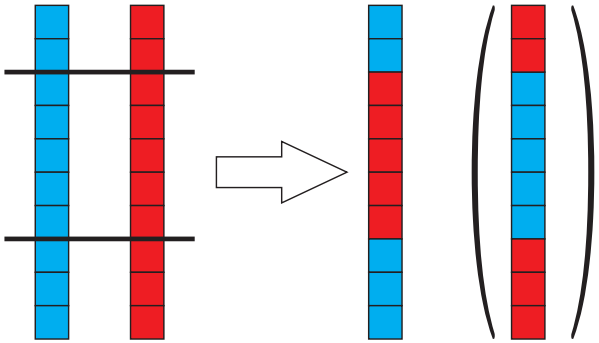
Two-Point (TPX)



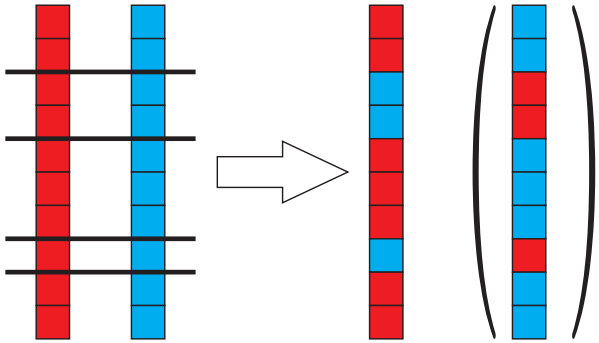
Multi-Point (MPX)



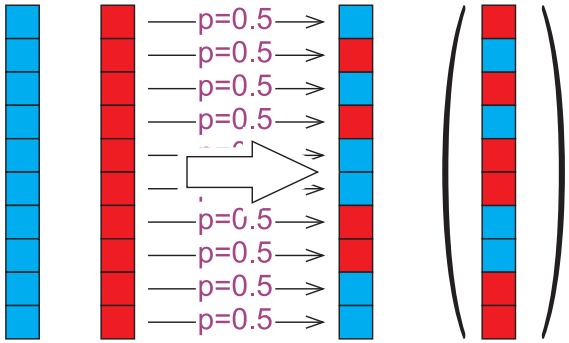
Single-Point (SPX)



Two-Point (TPX)



Multi-Point (MPX)



Uniform (UX)

Single-Point Crossover

Listing: Single-Point Crossover

```
public class BitsBinarySPX implements IBinarySearchOperation<boolean[]> {
  public boolean[] recombine(final boolean[] p1, final boolean[] p2, final
      Random r) {
    final boolean[] g;
    final int x;

    g = new boolean[p1.length]; // create empty bit string
    x = (1 + r.nextInt(g.length - 1)); // select crossover point
    System.arraycopy(p1, 0, g, 0, x); // copy from first parent
    System.arraycopy(p2, x, g, x, g.length - x);// copy from second parent

    return g; // return new bit string
  }
}
```

Crossover: MPX

$g \longleftarrow \text{recombinationMPX}(g_{p1}, g_{p2}, k)$

begin
- // find k crossover points from 0 to $n-1$
- $CP \longleftarrow$ new empty list
- **for** $i \longleftarrow 1$ **up to** k **do**
  - **repeat**
    - $cp \longleftarrow \{\text{randomly from } 0..n-1\} + 1$
  - **until** $cp \notin CP$
  - append cp to CP
- sort the list CP
- // perform the crossover by copying sub-strings
- $g \longleftarrow$ empty list
- $s \longleftarrow 0$
- $b \longleftarrow$ true
- $g_u \longleftarrow g_{p1}$
- **for** $i \longleftarrow 0$ **up to** k **do**
  - $e \longleftarrow CP[i]$
  - **if** b **then** $g_u \longleftarrow g_{p1}$
  - **else** $g_u \longleftarrow g_{p2}$
  - append sub-range $s..e-1$ of g_u to g
  - $b \longleftarrow \neg b$
  - $s \longleftarrow e$
- **return** g

Crossover: Uniform Crossover (UX)

$g \longleftarrow \text{recombinationUXX}(g_{p1}, g_{p2}, k)$

Input: $g_{p1}, g_{p2} \in \mathbb{G}$: the parental genotypes (n bits)
Data: i: a counter variable
Output: $g \in \mathbb{G}$: a new genotype (n bits)

begin
  $g \longleftarrow g_{p1}$
  for $i \longleftarrow 0$ **up to** $n-1$ **do**
    if $\{$randomly from $[0,1]\} < 0.5$ **then** $g[i] \longleftarrow g_{p2}[i]$
  return g

Uniform Crossover

Listing: Uniform Crossover

```
public class BitsBinaryUX implements IBinarySearchOperation<boolean[]> {
  public boolean[] recombine(final boolean[] p1, final boolean[] p2, final
      Random r) {
    final boolean[] g;

    g = p1.clone(); // copy first parent string
    for (int i = g.length; (--i) >= 0;) { // for all bits...
      if (r.nextBoolean()) {
        g[i] = p2[i];
      } // take value from p2 with probability 0.5
    }

    return g; // return new bit string
  }
}
```

Section Outline

Mutation

- Mutation corresponds to the unary search operators we already know

Mutation

- Mutation corresponds to the unary search operators we already know
- There exist different mutation operators

Mutation

- Mutation corresponds to the unary search operators we already know
- There exist different mutation operators
- Crossover reduces the diversity in the population:
  - different genotypes are combined

Mutation

- Mutation corresponds to the unary search operators we already know
- There exist different mutation operators
- Crossover reduces the diversity in the population:
  - different genotypes are combined
  - all genes of the new individual already existed in one of its parents

Mutation

- Mutation corresponds to the unary search operators we already know
- There exist different mutation operators
- Crossover reduces the diversity in the population:
  - different genotypes are combined
  - all genes of the new individual already existed in one of its parents
  - no now genetic material is created

Mutation

- Mutation corresponds to the unary search operators we already know
- There exist different mutation operators
- Crossover reduces the diversity in the population:
  - different genotypes are combined
  - all genes of the new individual already existed in one of its parents
  - no now genetic material is created
  - crossover and selection alone may step by step make the population converge to one single genotype

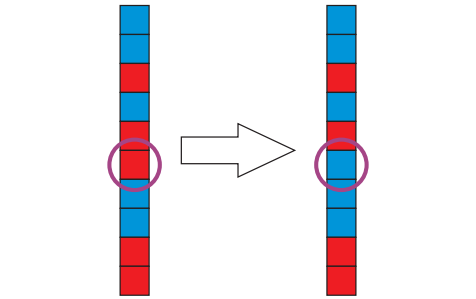
Mutation

- Mutation corresponds to the unary search operators we already know
- There exist different mutation operators
- Crossover reduces the diversity in the population:
  - different genotypes are combined
  - all genes of the new individual already existed in one of its parents
  - no now genetic material is created
  - crossover and selection alone may step by step make the population converge to one single genotype
- Mutation: introduce some randomnes (in from of new genetic material) into the population

Mutation

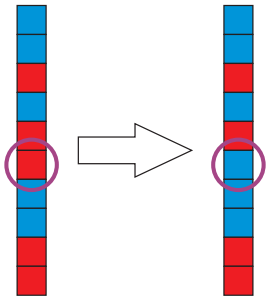
- Mutation corresponds to the unary search operators we already know
- There exist different mutation operators
- Crossover reduces the diversity in the population:
  - different genotypes are combined
  - all genes of the new individual already existed in one of its parents
  - no now genetic material is created
  - crossover and selection alone may step by step make the population converge to one single genotype
- Mutation: introduce some randomnes (in from of new genetic material) into the population
- But: not too much, systems with too much and too powerful mutation cannot converge or exploit local optima sufficiently

Mutation: Single Bit

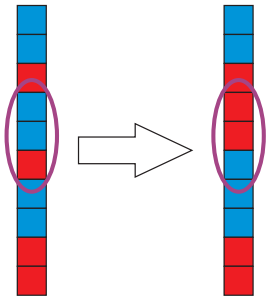


Single Bit Flip Mutation

Mutation: Multi Bit 1

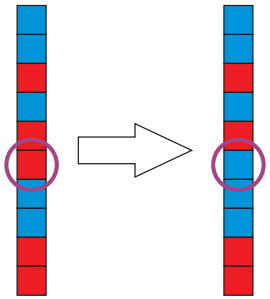


Single Bit Flip Mutation

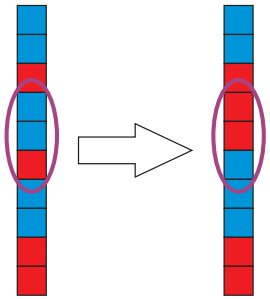


Multi Bit Flip Mutation
(consecutive)

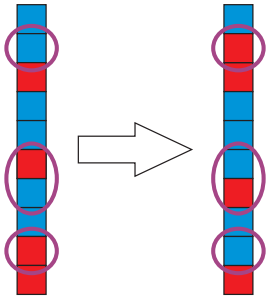
Mutation: Multi Bit 2



Single Bit Flip Mutation

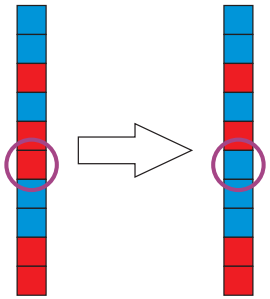


Multi Bit Flip Mutation
(consecutive)

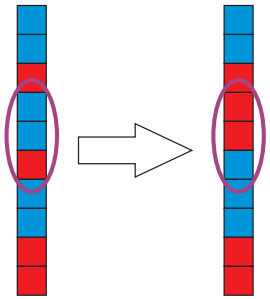


Multi Bit Flip Mutation
(random location)

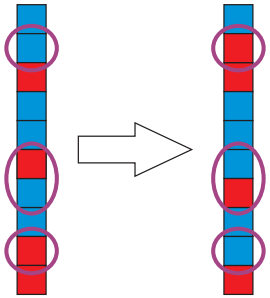
Mutation: Complete



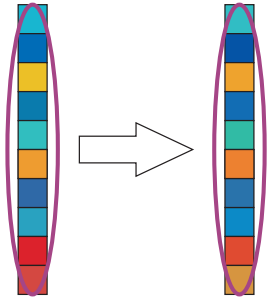
Single Bit Flip Mutation



Multi Bit Flip Mutation
(consecutive)



Multi Bit Flip Mutation
(random location)



Complete Mutation
($\mathbb{G} \subseteq \mathbb{R}$)

Multi Bit Mutation

$g \longleftarrow \text{multiBitFlip}(g_p, \eta)$

Input: g_p: the parent individual
Output: g: the new random permutation of the numbers $0 \ldots n-1$

begin
  $g \longleftarrow g_p$
  repeat
    $i \longleftarrow \{\text{randomly from } 0..n-1\}$
    $g[i] \longleftarrow \neg g[i]$
  until $\{\text{randomly from } [0,1]\} < \eta$
  return g

Single-Bit Flip Mutation

Listing: The Single-Bit Flip Mutation

```
public class BitsUnarySingleFlip implements IUnarySearchOperation<boolean[]>
      {
  public boolean[] mutate(final boolean[] p, final Random r) {
    final boolean[] g;

    g = p.clone(); // copy parent string
    g[r.nextInt(g.length)] ^= true; // flip the bit
    return g; // return new bit string
  }
}
```

Multi-Bit Flip Mutation

Listing: The Multi-Bit Flip Mutation

```
public class BitsUnaryFlip implements IUnarySearchOperation<boolean[]> {
  public boolean[] mutate(final boolean[] p, final Random r) {
    final boolean[] g;

    g = p.clone();                          // copy parent string
    do {                                    // at least once, but maybe more often
      g[r.nextInt(g.length)] ^= true;       // flip the bit
    } while (r.nextBoolean());              // maybe repeat (small chance for
        large changes
    return g;                               // return new bit string
  }
}
```

Flip a Fraction of the Bits

Listing: Flip a Fraction of the Bits

```
public class BitsUnaryFractionFlip implements
    IUnarySearchOperation<boolean[]> {
  /** the fraction to flip */
  public final double frac;
  public boolean[] mutate(final boolean[] p, final Random r) {
    final boolean[] g;
    int f;

    g = p.clone(); // copy parent string
    f = Math.max(1, Math.min(p.length, ((int) (p.length * this.frac))));
    for (; (--f) >= 0;) {// go through the bits
      g[r.nextInt(p.length)] ^= true; // flip
    }

    return g; // return new bit string
  }
}
```

Section Outline

Schema Theorem: Introduction

- Schema Theorem makes a statement about how a Genetic Algorithm progresses and how the fitness of its population may improve

Schema Theorem: Introduction

- Schema Theorem makes a statement about how a Genetic Algorithm progresses and how the fitness of its population may improve
- It was first stated by Holland back in 1975 [16, 18, 44]

Schema Theorem: Introduction

- Schema Theorem makes a statement about how a Genetic Algorithm progresses and how the fitness of its population may improve
- It was first stated by Holland back in 1975 [16, 18, 44]
- One of the ideas behind GAs and its binary search operator crossover was

Schema Theorem: Introduction

- Schema Theorem makes a statement about how a Genetic Algorithm progresses and how the fitness of its population may improve
- It was first stated by Holland back in 1975 [16, 18, 44]
- One of the ideas behind GAs and its binary search operator crossover was:
  - good features of the phenotype

Schema Theorem: Introduction

- Schema Theorem makes a statement about how a Genetic Algorithm progresses and how the fitness of its population may improve
- It was first stated by Holland back in 1975 [16, 18, 44]
- One of the ideas behind GAs and its binary search operator crossover was:
  - good features of the phenotype
  - are represented by specific values of specific genes (bits at specific positions in the genotypes)

Schema Theorem: Introduction

- Schema Theorem makes a statement about how a Genetic Algorithm progresses and how the fitness of its population may improve
- It was first stated by Holland back in 1975 [16, 18, 44]
- One of the ideas behind GAs and its binary search operator crossover was:
  - good features of the phenotype
  - are represented by specific values of specific genes (bits at specific positions in the genotypes)
  - there may be different good features, encoded by different parts of the genotypes

Schema Theorem: Introduction

- Schema Theorem makes a statement about how a Genetic Algorithm progresses and how the fitness of its population may improve
- It was first stated by Holland back in 1975 [16, 18, 44]
- One of the ideas behind GAs and its binary search operator crossover was:
  - good features of the phenotype
  - are represented by specific values of specific genes (bits at specific positions in the genotypes)
  - there may be different good features, encoded by different parts of the genotypes
  - these different good parts may be in different individuals

Schema Theorem: Introduction

- Schema Theorem makes a statement about how a Genetic Algorithm progresses and how the fitness of its population may improve
- It was first stated by Holland back in 1975 [16, 18, 44]
- One of the ideas behind GAs and its binary search operator crossover was:
  - good features of the phenotype
  - are represented by specific values of specific genes (bits at specific positions in the genotypes)
  - there may be different good features, encoded by different parts of the genotypes
  - these different good parts may be in different individuals
  - but may be combined later by crossover (Building Block Hypothesis, see later)

Schema Theorem: Introduction

- Schema Theorem makes a statement about how a Genetic Algorithm progresses and how the fitness of its population may improve
- It was first stated by Holland back in 1975 [16, 18, 44]
- One of the ideas behind GAs and its binary search operator crossover was:
  - good features of the phenotype
  - there may be different good features, encoded by different parts of the genotypes
  - these different good parts may be in different individuals
- Interesting "parts" of a genotype are described with blueprints (masks, schemas)

Schema Theorem: Introduction

- Schema Theorem makes a statement about how a Genetic Algorithm progresses and how the fitness of its population may improve
- It was first stated by Holland back in 1975 [16, 18, 44]
- One of the ideas behind GAs and its binary search operator crossover was:
  - good features of the phenotype
  - there may be different good features, encoded by different parts of the genotypes
  - these different good parts may be in different individuals
- Interesting "parts" of a genotype are described with blueprints (masks, schemas)
- the Schema Theorem roughly estimates how such schemas multiply, depending on their average fitness

Schema Theorem: Introduction

- Schema Theorem makes a statement about how a Genetic Algorithm progresses and how the fitness of its population may improve
- It was first stated by Holland back in 1975 [16, 18, 44]
- One of the ideas behind GAs and its binary search operator crossover was:
  - good features of the phenotype
  - there may be different good features, encoded by different parts of the genotypes
  - these different good parts may be in different individuals
- Interesting "parts" of a genotype are described with blueprints (masks, schemas)
- the Schema Theorem roughly estimates how such schemas multiply, depending on their average fitness
- It tries to answer the question: "How and why does a GA work?"

Schema Theorem: Some Definitions

- For a search space $\mathbb{G} \subseteq \{\texttt{false}, \texttt{true}\}^n$ of dimension n...

Schema Theorem: Some Definitions

- For a search space $\mathbb{G} \subseteq \{\texttt{false}, \texttt{true}\}^n$ of dimension n...
- Mask m is an element of the power set $\mathcal{P}(0 \ldots n-1)$ and defines a set of loci

Schema Theorem: Some Definitions

- For a search space $\mathbb{G} \subseteq \{\texttt{false}, \texttt{true}\}^n$ of dimension n...
- Mask m is an element of the power set $\mathcal{P}(0 \ldots n-1)$ and defines a set of loci
- The order of a mask m is the number of loci defined by it: $\text{order}(m) = |m|$

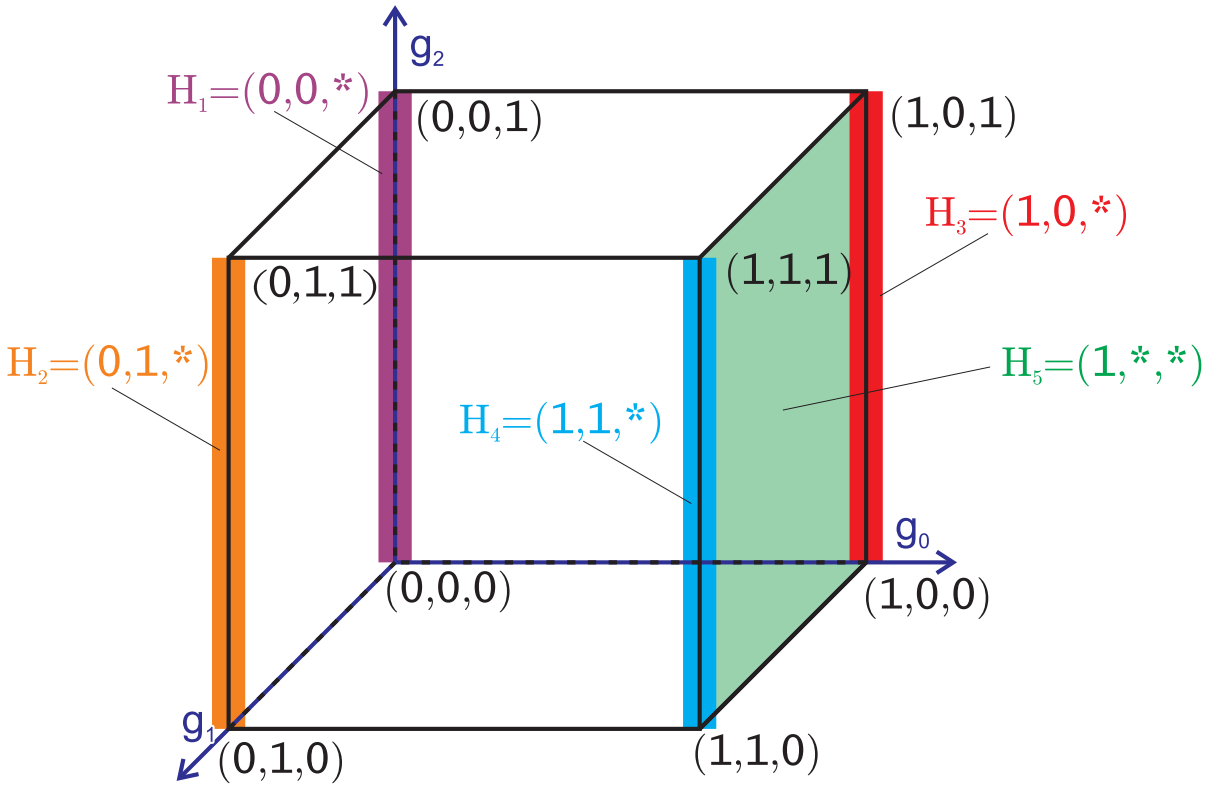
Schema Theorem: Some Definitions

- For a search space $\mathbb{G} \subseteq \{\texttt{false}, \texttt{true}\}^n$ of dimension n...
- Mask m is an element of the power set $\mathcal{P}(0 \ldots n-1)$ and defines a set of loci
- The order of a mask m is the number of loci defined by it: $\text{order}(m) = |m|$
- The defined length $\delta(m)$ is the maximum distance between two loci in a mask $\delta(m) = \max\{|j-k| \, \forall j,k \in m\}$

Schema Theorem: Some Definitions

- For a search space $\mathbb{G} \subseteq \{\texttt{false}, \texttt{true}\}^n$ of dimension n...
- Mask m is an element of the power set $\mathcal{P}(0 \ldots n-1)$ and defines a set of loci
- The order of a mask m is the number of loci defined by it: $\text{order}(m) = |m|$
- The defined length $\delta(m)$ is the maximum distance between two loci in a mask $\delta(m) = \max\{|j-k| \, \forall j,k \in m\}$
- A Schema H is an equivalence class concerning the values at specific loci (i.e., values of genotypes according to a mask): $H(j) \in \{0,1,*\} \forall j \in 0 \ldots n-1,\ H(j) = * \forall j \notin m$

- A Schema H is an equivalence class concerning the values at specific loci (i.e., values of genotypes according to a mask):
  $H(j) \in \{0, 1, *\} \forall j \in 0 \ldots n-1,\ H(j) = * \forall j \notin m$



Schema Theorem: Selection Probability

- Holland uses Roulette Wheel Selection in his GA

Schema Theorem: Selection Probability

- Holland uses Roulette Wheel Selection in his GA
- For each open slot in the mating pool, the chance of an individual $p \in \mathrm{pop}$ from the population pop to be copied into this slot is

Schema Theorem: Selection Probability

- Holland uses Roulette Wheel Selection in his GA
- For each open slot in the mating pool, the chance of an individual $p \in \mathrm{pop}$ from the population pop to be copied into this slot is:

$$P(\mathrm{select}(p)) = \frac{f(p.x)}{\sum_{\forall p' \in \mathrm{pop}} f(p'.x)} \tag{3}$$

- But we do not look at a single individual $p \in (\mathrm{pop} \cap H)$, we look at *all the individuals* in population pop that fit to schema H

Schema Theorem: Selection Probability

- Holland uses Roulette Wheel Selection in his GA
- For each open slot in the mating pool, the chance of an individual $p \in \text{pop}$ from the population pop to be copied into this slot is:

$$P(\text{select}(p)) = \frac{f(p.x)}{\sum_{\forall p' \in \text{pop}} f(p'.x)} \tag{3}$$

- We look at *all the individuals* in population pop that fit to schema H
- The chance $P(\text{select}(H))$ that one of them is chosen into the slot is[1]:

$$P(\text{select}(H)) = \sum_{\forall p \in (\text{pop} \cap H)} \left[\frac{f(p.x)}{\sum_{\forall p' \in \text{pop}} f(p'.x)} \right] \tag{4}$$

[1] $A \cap B$ is the intersection operator, returning a set with only the elements that are in both A and B

Schema Theorem: Selection Probability

- Holland uses Roulette Wheel Selection in his GA
- For each open slot in the mating pool, the chance of an individual $p \in \mathrm{pop}$ from the population pop to be copied into this slot is:

$$P(\mathrm{select}(p)) = \frac{f(p.x)}{\sum_{\forall p' \in \mathrm{pop}} f(p'.x)} \tag{3}$$

- We look at *all the individuals* in population pop that fit to schema H
- The chance $P(\mathrm{select}(H))$ that one of them is chosen into the slot is[1]:

$$P(\mathrm{select}(H)) = \frac{1}{\sum_{\forall p' \in \mathrm{pop}} f(p'.x)} \sum_{\forall p \in (\mathrm{pop} \cap H)} f(p.x) \tag{4}$$

[1] $A \cap B$ is the intersection operator, returning a set with only the elements that are in both A and B

Schema Theorem: Selection Probability

- Holland uses Roulette Wheel Selection in his GA
- For each open slot in the mating pool, the chance of an individual $p \in \mathrm{pop}$ from the population pop to be copied into this slot is:

$$P(\mathrm{select}(p)) = \frac{f(p.x)}{\sum_{\forall p' \in \mathrm{pop}} f(p'.x)} \tag{3}$$

- We look at *all the individuals* in population pop that fit to schema H
- The chance $P(\mathrm{select}(H))$ that one of them is chosen into the slot is[1]:

$$P(\mathrm{select}(H)) = \frac{\sum_{\forall p \in (\mathrm{pop} \cap H)} f(p.x)}{\sum_{\forall p' \in \mathrm{pop}} f(p'.x)} \tag{4}$$

[1] $A \cap B$ is the intersection operator, returning a set with only the elements that are in both A and B

Schema Theorem: Selection + Mean Fitness

$$P(\text{select}(H)) = \frac{\sum_{\forall p \in (\text{pop} \cap H)} f(p.x)}{\sum_{\forall p' \in \text{pop}} f(p'.x)} \tag{5}$$

$$P(\text{select}(H)) = \frac{\sum_{\forall p \in (\text{pop} \cap H)} f(p.x)}{\sum_{\forall p' \in \text{pop}} f(p'.x)} \tag{5}$$

- Consider the *mean fitness* $\overline{f}(\text{pop})$ of all individuals in the population and the mean fitness $\overline{f}(\text{pop} \cap H)$ of all instances of H in the population[2]:

$$\overline{f}(\text{pop}) = \frac{1}{ps} \sum_{\forall p' \in \text{pop}} f(p'.x) \qquad \overline{f}(\text{pop} \cap H) = \frac{1}{|\text{pop} \cap H|} \sum_{\forall p \in (\text{pop} \cap H)} f(p.x) \tag{6}$$

[2] $|A|$ is the set size of A, the number of elements in A.

$$P(\text{select}(H)) = \frac{\sum_{\forall p \in (\text{pop} \cap H)} f(p.x)}{\sum_{\forall p' \in \text{pop}} f(p'.x)} \tag{5}$$

- Consider the *mean fitness* $\overline{f}(\text{pop})$ of all individuals in the population and the mean fitness $\overline{f}(\text{pop} \cap H)$ of all instances of H in the population[2]:

$$ps * \overline{f}(\text{pop}) = \sum_{\forall p' \in \text{pop}} f(p'.x) \qquad |\text{pop} \cap H| * \overline{f}(\text{pop} \cap H) = \sum_{\forall p \in (\text{pop} \cap H)} f(p.x) \tag{6}$$

[2] $|A|$ is the set size of A, the number of elements in A.

Schema Theorem: Selection + Mean Fitness

$$P(\text{select}(H)) = \frac{\sum_{\forall p \in (\text{pop} \cap H)} f(p.x)}{\sum_{\forall p' \in \text{pop}} f(p'.x)} \tag{5}$$

- Consider the *mean fitness* $\overline{f}(\text{pop})$ of all individuals in the population and the mean fitness $\overline{f}(\text{pop} \cap H)$ of all instances of H in the population:

$$ps * \overline{f}(\text{pop}) = \sum_{\forall p' \in \text{pop}} f(p'.x) \qquad |\text{pop} \cap H| * \overline{f}(\text{pop} \cap H) = \sum_{\forall p \in (\text{pop} \cap H)} f(p.x) \tag{6}$$

- Now we put this into the first equation:

$$P(\text{select}(H)) = \frac{|\text{pop} \cap H| * \overline{f}(\text{pop} \cap H)}{ps * \overline{f}(\text{pop})} \tag{7}$$

Schema Theorem: Expected Selections

- Under Roulette-Wheel Selection, the chance $P(\text{select}(H))$ to select an instance of schema H from the population pop_t (at generation t) for one slot in the mating pool is

$$P(\text{select}(H)) = \frac{|\text{pop}_t \cap H| * \overline{f}(\text{pop}_t \cap H)}{ps * \overline{f}(\text{pop}_t)} \tag{8}$$

Schema Theorem: Expected Selections

- Under Roulette-Wheel Selection, the chance $P(\text{select}(H))$ to select an instance of schema H from the population pop_t (at generation t) for one slot in the mating pool is

$$P(\text{select}(H)) = \frac{|\text{pop}_t \cap H| * \overline{f}(\text{pop}_t \cap H)}{ps * \overline{f}(\text{pop}_t)} \tag{8}$$

- Holland assumes that the size mps of the mating pool is the same as the size ps of the population, so this is applied ps times

Schema Theorem: Expected Selections

- Under Roulette-Wheel Selection, the chance $P(\text{select}(H))$ to select an instance of schema H from the population pop_t (at generation t) for one slot in the mating pool is

$$P(\text{select}(H)) = \frac{|\text{pop}_t \cap H| * \overline{f}(\text{pop}_t \cap H)}{ps * \overline{f}(\text{pop}_t)} \tag{8}$$

- Holland assumes that the size mps of the mating pool is the same as the size ps of the population, so this is applied ps times
- The expected number of offspring of the schema's instances, i.e., its expected occurences $E(|\text{matePool}_t \cap H|)$ in the mating pool, are:

$$E(|\text{matePool}_t \cap H|) = \frac{|\text{pop}_t \cap H| * \overline{f}(\text{pop}_t \cap H)}{\overline{f}(\text{pop}_t)} \tag{9}$$

Schema Theorem: Expected Selections

- Under Roulette-Wheel Selection, the chance $P(\text{select}(H))$ to select an instance of schema H from the population pop_t (at generation t) for one slot in the mating pool is

$$P(\text{select}(H)) = \frac{|\text{pop}_t \cap H| * \overline{f}(\text{pop}_t \cap H)}{ps * \overline{f}(\text{pop}_t)} \tag{8}$$

- Holland assumes that the size mps of the mating pool is the same as the size ps of the population, so this is applied ps times
- The expected number of offspring of the schema's instances, i.e., its expected occurences $E(|\text{matePool}_t \cap H|)$ in the mating pool

$$\frac{E(|\text{matePool}_t \cap H|)}{|\text{pop}_t \cap H|} = \frac{\overline{f}(\text{pop}_t \cap H)}{\overline{f}(\text{pop}_t)} \tag{9}$$

Schema Theorem: Expected Selections

- Under Roulette-Wheel Selection, the chance $P(\text{select}(H))$ to select an instance of schema H from the population pop_t (at generation t) for one slot in the mating pool is

$$P(\text{select}(H)) = \frac{|\text{pop}_t \cap H| * \overline{f}(\text{pop}_t \cap H)}{ps * \overline{f}(\text{pop}_t)} \tag{8}$$

- The expected number of offspring of the schema's instances, i.e., its expected occurences $E(|\text{matePool}_t \cap H|)$ in the mating pool

$$\frac{E(|\text{matePool}_t \cap H|)}{|\text{pop}_t \cap H|} = \frac{\overline{f}(\text{pop}_t \cap H)}{\overline{f}(\text{pop}_t)} \tag{9}$$

- If $\overline{f}(\text{pop}_t \cap H) > \overline{f}(\text{pop}_t)$, then $E(|\text{matePool}_t \cap H|) > |\text{pop}_t \cap H|$

Schema Theorem: Expected Selections

- Under Roulette-Wheel Selection, the chance $P(\text{select}(H))$ to select an instance of schema H from the population pop_t (at generation t) for one slot in the mating pool is

$$P(\text{select}(H)) = \frac{|\text{pop}_t \cap H| * \overline{f}(\text{pop}_t \cap H)}{ps * \overline{f}(\text{pop}_t)} \tag{8}$$

- The expected number of offspring of the schema's instances, i.e., its expected occurences $E(|\text{matePool}_t \cap H|)$ in the mating pool

$$\frac{E(|\text{matePool}_t \cap H|)}{|\text{pop}_t \cap H|} = \frac{\overline{f}(\text{pop}_t \cap H)}{\overline{f}(\text{pop}_t)} \tag{9}$$

- The number of offsprings of a schema with above-average fitness will be higher than the current number of its instances in the population

Schema Theorem: Expected Selections

- Under Roulette-Wheel Selection, the chance $P(\text{select}(H))$ to select an instance of schema H from the population pop_t (at generation t) for one slot in the mating pool is

$$P(\text{select}(H)) = \frac{|\text{pop}_t \cap H| * \overline{f}(\text{pop}_t \cap H)}{ps * \overline{f}(\text{pop}_t)} \tag{8}$$

- The expected number of offspring of the schema's instances, i.e., its expected occurences $E(|\text{matePool}_t \cap H|)$ in the mating pool

$$\frac{E(|\text{matePool}_t \cap H|)}{|\text{pop}_t \cap H|} = \frac{\overline{f}(\text{pop}_t \cap H)}{\overline{f}(\text{pop}_t)} \tag{9}$$

- The number of instances of a schema with above average fitness will increase

Schema Theorem: Probability of Destruction

- The number of instances of a schema with above average fitness will increase

Schema Theorem: Probability of Destruction

- The number of instances of a schema with above average fitness will increase...
- ...but: We only considered the number of instances in the mating pool

Schema Theorem: Probability of Destruction

- The number of instances of a schema with above average fitness will increase...
- ...but: We only considered the number of instances in the mating pool...
- ...without including the influence of crossover and mutation!

Schema Theorem: Probability of Destruction

- The number of instances of a schema with above average fitness will increase...
- ...but: We only considered the number of instances in the mating pool...
- ...without including the influence of crossover and mutation!
- Both crossover and mutation may change the genetic material

Schema Theorem: Probability of Destruction

- The number of instances of a schema with above average fitness will increase...
- ...but: We only considered the number of instances in the mating pool...
- ...without including the influence of crossover and mutation!
- Both crossover and mutation may change the genetic material
- If the genetic material is changed, an offspring of a schema instance may not be a schema instance!

Schema Theorem: Probability of Destruction

- The number of instances of a schema with above average fitness will increase...
- ...but: We only considered the number of instances in the mating pool...
- ...without including the influence of crossover and mutation!
- Both crossover and mutation may change the genetic material
- If the genetic material is changed, an offspring of a schema instance may not be a schema instance!
- ξ be the probability that an instance of H is destroyed during reproduction

Schema Theorem: Probability of Destruction

- The number of instances of a schema with above average fitness will increase...
- ...but: We only considered the number of instances in the mating pool...
- ...without including the influence of crossover and mutation!
- Both crossover and mutation may change the genetic material
- If the genetic material is changed, an offspring of a schema instance may not be a schema instance!
- ξ be the probability that an instance of H is destroyed during reproduction, so we get:

$$E(|\text{pop}_{t+1} \cap H|) = (1-\xi)E(|\text{matePool}_t \cap H|) = \frac{|\text{pop}_t \cap H| * \overline{f}(\text{pop}_t \cap H)}{\overline{f}(\text{pop}_t)}(1-\xi) \tag{10}$$

Schema Theorem: Schema Destruction

- Single-Point Crossover:

$$\xi_c = \frac{\delta(m)}{n-1} \tag{11}$$

Schema Theorem: Schema Destruction

- Single-Point Crossover:

$$\xi_c = \frac{\delta(m)}{n-1} \quad \Rightarrow \text{short schemas are favored} \tag{11}$$

Schema Theorem: Schema Destruction

- Single-Point Crossover:

$$\xi_c = \frac{\delta(m)}{n-1} \quad \Rightarrow \text{short schemas are favored} \tag{11}$$

- Single Bit Flip Mutation:

$$\xi_m = \frac{\text{order}(m)}{n} \tag{12}$$

Schema Theorem: Schema Destruction

- Single-Point Crossover:

$$\xi_c = \frac{\delta(m)}{n-1} \quad \Rightarrow \text{short schemas are favored} \tag{11}$$

- Single Bit Flip Mutation:

$$\xi_m = \frac{\text{order}(m)}{n} \quad \Rightarrow \text{ schemas with many don't cares are favored} \tag{12}$$

Schema Theorem: Schema Destruction

- Single-Point Crossover:

$$\xi_c = \frac{\delta(m)}{n-1} \quad \Rightarrow \text{short schemas are favored} \tag{11}$$

- Single Bit Flip Mutation:

$$\xi_m = \frac{\text{order}(m)}{n} \quad \Rightarrow \text{ schemas with many don't cares are favored} \tag{12}$$

- For either-mutation-or-crossover-GAs, we get:

$$\xi \leq \xi_c * cr + \xi_m * mr = cr\frac{\delta(m)}{n-1} + mr\frac{\text{order}(m)}{n} \tag{13}$$

- ξ be the probability that an instance of H is destroyed during reproduction, so we get:

$$E(|\text{pop}_{t+1} \cap H|) = \frac{|\text{pop}_t \cap H| * \overline{f}(\text{pop}_t \cap H)}{\overline{f}(\text{pop}_t)}(1 - \xi)$$

- For either-mutation-or-crossover-GAs, we get:

$$\xi \leq \xi_c * cr + \xi_m * mr = cr\frac{\delta(m)}{n-1} + mr\frac{\text{order}(m)}{n}$$

- ξ be the probability that an instance of H is destroyed during reproduction, so we get:

$$E(|\text{pop}_{t+1} \cap H|) = \frac{|\text{pop}_t \cap H| * \overline{f}(\text{pop}_t \cap H)}{\overline{f}(\text{pop}_t)}(1 - \xi)$$

- For either-mutation-or-crossover-GAs, we get:

$$\xi \leq \xi_c * cr + \xi_m * mr = cr\frac{\delta(m)}{n-1} + mr\frac{\text{order}(m)}{n}$$

- Hence:

$$E(|\text{pop}_{t+1} \cap H|) = \frac{|\text{pop}_t \cap H| * \overline{f}(\text{pop}_t \cap H)}{f_t(\text{pop}_t)}\left(1 - cr\frac{\delta(m)}{n-1} + mr\frac{\text{order}(m)}{n}\right) \quad (14)$$

- Let's go back to:

$$E(|\text{pop}_{t+1} \cap H|) = \frac{|\text{pop}_t \cap H| * \overline{f}(\text{pop}_t \cap H)}{\overline{f}(\text{pop}_t)}(1 - \xi)$$

- Let's go back to:

$$E(|\text{pop}_{t+1} \cap H|) = \frac{|\text{pop}_t \cap H| * \overline{f}(\text{pop}_t \cap H)}{\overline{f}(\text{pop}_t)}(1 - \xi)$$

- We can expect an increase of instances if:

$$\overline{f}(\text{pop}_t \cap H) > \frac{\overline{f}(\text{pop}_t)}{1 - \xi} \tag{15}$$

- Let's go back to:

$$E(|\text{pop}_{t+1} \cap H|) = \frac{|\text{pop}_t \cap H| * \overline{f}(\text{pop}_t \cap H)}{\overline{f}(\text{pop}_t)}(1 - \xi)$$

- We can expect an increase of instances if:

$$\overline{f}(\text{pop}_t \cap H) > \frac{\overline{f}(\text{pop}_t)}{1 - \xi} \tag{15}$$

- If the relation between $\overline{f}(\text{pop}_t \cap H)$ and $\overline{f}(\text{pop}_t)$ remains constant and positive over multiple generations, we have exponential growth of number of schema instances

- If the relation between $\overline{f}(\text{pop}_t \cap H)$ and $\overline{f}(\text{pop}_t)$ remains constant and positive over multiple generations, we have exponential growth of number of schema instances

- If the relation between $\overline{f}(\text{pop}_t \cap H)$ and $\overline{f}(\text{pop}_t)$ remains constant and positive over multiple generations, we have exponential growth of number of schema instances, but
  1. if number of instances of H increase, $\overline{f}(\text{pop})$ increases too

- If the relation between $\overline{f}(\text{pop}_t \cap H)$ and $\overline{f}(\text{pop}_t)$ remains constant and positive over multiple generations, we have exponential growth of number of schema instances, but
  1. if number of instances of H increase, $\overline{f}(\text{pop})$ increases too
  2. population is finite

- If the relation between $\overline{f}(\text{pop}_t \cap H)$ and $\overline{f}(\text{pop}_t)$ remains constant and positive over multiple generations, we have exponential growth of number of schema instances, but
  1. if number of instances of H increase, $\overline{f}(\text{pop})$ increases too
  2. population is finite
  3. actually, only $\overline{f}(\text{pop}_t \cap H)$ is not known (samples of H), not $\overline{f}(H)$
     $\Longrightarrow$ new instances of H may actually be very bad

Building Block Hypothesis

- Building Block Hypothesis: Good genes will aggregate [16, 19, 20]

Building Block Hypothesis

- Building Block Hypothesis: Good genes will aggregate [16, 19, 20]
- If there exist some low-order, low-defining length schemata with above-average fitness...

Building Block Hypothesis

- Building Block Hypothesis: Good genes will aggregate [16, 19, 20]
- If there exist some low-order, low-defining length schemata with above-average fitness...
- ...these schemata are combined step by step by the Genetic Algorithm in order to form larger and better strings.

Building Block Hypothesis

- Building Block Hypothesis: Good genes will aggregate [16, 19, 20]
- If there exist some low-order, low-defining length schemata with above-average fitness...
- ...these schemata are combined step by step by the Genetic Algorithm in order to form larger and better strings.
- No proof for this exists so far!

Building Block Hypothesis

- Building Block Hypothesis: Good genes will aggregate [16, 19, 20]
- If there exist some low-order, low-defining length schemata with above-average fitness...
- ...these schemata are combined step by step by the Genetic Algorithm in order to form larger and better strings.
- No proof for this exists so far!
- Consider the criticism of Schema Theorem

Alternative Hypothesis

- Alternative Hypothesis [45, 46]: Genetic Repair & Similarity Extraction by Beyer [46]

Alternative Hypothesis

- Alternative Hypothesis [45, 46]: Genetic Repair & Similarity Extraction by Beyer [46]
- Assume uniform crossover (UX)

Alternative Hypothesis

- Alternative Hypothesis [45, 46]: Genetic Repair & Similarity Extraction by Beyer [46]
- Assume uniform crossover (UX)
  1. If a gene has the same allele in both parents, it will be inherited

Alternative Hypothesis

- Alternative Hypothesis [45, 46]: Genetic Repair & Similarity Extraction by Beyer [46]
- Assume uniform crossover (UX)
  1. If a gene has the same allele in both parents, it will be inherited
  2. Otherwise, one parent has allele 1 and the other has 0 in the gene, ...

Alternative Hypothesis

- Alternative Hypothesis [45, 46]: Genetic Repair & Similarity Extraction by Beyer [46]
- Assume uniform crossover (UX)
  1. If a gene has the same allele in both parents, it will be inherited
  2. Otherwise, one parent has allele 1 and the other has 0 in the gene, ...
  3. ...i.e., , the allele in the child will be 0 with 50% chance or 1 with 50% chance,

Alternative Hypothesis

- Alternative Hypothesis [45, 46]: Genetic Repair & Similarity Extraction by Beyer [46]
- Assume uniform crossover (UX)
  1. If a gene has the same allele in both parents, it will be inherited
  2. Otherwise, one parent has allele 1 and the other has 0 in the gene, ...
  3. ...i.e., , the allele in the child will be 0 with 50% chance or 1 with 50% chance,
  4. i.e., effectively be "randomized"

Alternative Hypothesis

- Alternative Hypothesis [45, 46]: Genetic Repair & Similarity Extraction by Beyer [46]
- Assume uniform crossover (UX)
  1. If a gene has the same allele in both parents, it will be inherited
  2. Otherwise, one parent has allele 1 and the other has 0 in the gene, ...
  3. ...i.e., , the allele in the child will be 0 with 50% chance or 1 with 50% chance,
  4. i.e., effectively be "randomized"
- genes with similar alleles remain, different alleles are "randomized"

Alternative Hypothesis

- Alternative Hypothesis [45, 46]: Genetic Repair & Similarity Extraction by Beyer [46]
- Assume uniform crossover (UX)
  1. If a gene has the same allele in both parents, it will be inherited
  2. Otherwise, one parent has allele 1 and the other has 0 in the gene, ...
  3. ...i.e., , the allele in the child will be 0 with 50% chance or 1 with 50% chance,
  4. i.e., effectively be "randomized"
- genes with similar alleles remain, different alleles are "randomized"
- useful gene sequences which step-by-step are built by mutation

Section Outline

Extension to Evolutionary Algorithms

- Evolutionary Algorithms (EAs) are the generalization of Genetic Algorithms to different search spaces and problem types [47–55]

Extension to Evolutionary Algorithms

- Evolutionary Algorithms (EAs) are the generalization of Genetic Algorithms to different search spaces and problem types [47–55]
- They are the most successful branch of metaheuristic optimization

Extension to Evolutionary Algorithms

- Evolutionary Algorithms (EAs) are the generalization of Genetic Algorithms to different search spaces and problem types [47–55]
- They are the most successful branch of metaheuristic optimization
- Examples:

Extension to Evolutionary Algorithms

- Evolutionary Algorithms (EAs) are the generalization of Genetic Algorithms to different search spaces and problem types [47–55]
- They are the most successful branch of metaheuristic optimization
- Examples:
  - Strings in GAs do not need to have a fixed length. Why not using variable-length strings? [47]

Extension to Evolutionary Algorithms

- Evolutionary Algorithms (EAs) are the generalization of Genetic Algorithms to different search spaces and problem types [47–55]
- They are the most successful branch of metaheuristic optimization
- Examples:
  - Strings in GAs do not need to have a fixed length. Why not using variable-length strings? [47]
  - Why bit strings? Why not permutations?

Extension to Evolutionary Algorithms

- Evolutionary Algorithms (EAs) are the generalization of Genetic Algorithms to different search spaces and problem types [47–55]
- They are the most successful branch of metaheuristic optimization
- Examples:
  - Strings in GAs do not need to have a fixed length. Why not using variable-length strings? [47]
  - Why bit strings? Why not permutations?
  - Why not vector of real numbers? (ES) [35–41, 47]

Extension to Evolutionary Algorithms

- Evolutionary Algorithms (EAs) are the generalization of Genetic Algorithms to different search spaces and problem types [47–55]
- They are the most successful branch of metaheuristic optimization
- Examples:
  - Strings in GAs do not need to have a fixed length. Why not using variable-length strings? [47]
  - Why bit strings? Why not permutations?
  - Why not vector of real numbers? (ES) [35–41, 47]
  - Search space does not need to be strings. Why not tree data structures? (GP) [47, 56–63]

Extension to Evolutionary Algorithms

- Evolutionary Algorithms (EAs) are the generalization of Genetic Algorithms to different search spaces and problem types [47–55]
- They are the most successful branch of metaheuristic optimization
- Examples:
  - Strings in GAs do not need to have a fixed length. Why not using variable-length strings? [47]
  - Why bit strings? Why not permutations?
  - Why not vector of real numbers? (ES) [35–41, 47]
  - Search space does not need to be strings. Why not tree data structures? (GP) [47, 56–63]
  - Why a single objective function? Why not multiple? (MOEAs) [64–66]

Extension to Evolutionary Algorithms

- Evolutionary Algorithms (EAs) are the generalization of Genetic Algorithms to different search spaces and problem types [47–55]
- They are the most successful branch of metaheuristic optimization
- Examples:
  - Strings in GAs do not need to have a fixed length. Why not using variable-length strings? [47]
  - Why bit strings? Why not permutations?
  - Why not vector of real numbers? (ES) [35–41, 47]
  - Search space does not need to be strings. Why not tree data structures? (GP) [47, 56–63]
  - Why a single objective function? Why not multiple? (MOEAs) [64–66]
  - Why is recombination binary? Why not ternary? (DE) [67, 68]

Extension to Evolutionary Algorithms

- Evolutionary Algorithms (EAs) are the generalization of Genetic Algorithms to different search spaces and problem types [47–55]
- They are the most successful branch of metaheuristic optimization
- Examples:
  - Strings in GAs do not need to have a fixed length. Why not using variable-length strings? [47]
  - Why bit strings? Why not permutations?
  - Why not vector of real numbers? (ES) [35–41, 47]
  - Search space does not need to be strings. Why not tree data structures? (GP) [47, 56–63]
  - Why a single objective function? Why not multiple? (MOEAs) [64–66]
  - Why is recombination binary? Why not ternary? (DE) [67, 68]
  - Why do we need a population? Why can't we use a probabilistic model instead? (EDAs) [69–72]

Extension to Evolutionary Algorithms

- Evolutionary Algorithms (EAs) are the generalization of Genetic Algorithms to different search spaces and problem types [47–55]
- They are the most successful branch of metaheuristic optimization
- Examples:
  - Strings in GAs do not need to have a fixed length. Why not using variable-length strings? [47]
  - Why bit strings? Why not permutations?
  - Why not vector of real numbers? (ES) [35–41, 47]
  - Search space does not need to be strings. Why not tree data structures? (GP) [47, 56–63]
  - Why a single objective function? Why not multiple? (MOEAs) [64–66]
  - Why is recombination binary? Why not ternary? (DE) [67, 68]
  - Why do we need a population? Why can't we use a probabilistic model instead? (EDAs) [69–72]
  - Why not including a local search (such as Hill Climbers) to refine the results? (MAs) [73, 74]

Summary

- Genetic Algorithms are population-based metaheuristics inspired by natural evolution

Summary

- Genetic Algorithms are population-based metaheuristics inspired by natural evolution
- They proceed in a cycle of GPM, evaluation, selection, and reproduction

Summary

- Genetic Algorithms are population-based metaheuristics inspired by natural evolution
- They proceed in a cycle of GPM, evaluation, selection, and reproduction
- Originally: bit-string based search spaces

Summary

- Genetic Algorithms are population-based metaheuristics inspired by natural evolution
- They proceed in a cycle of GPM, evaluation, selection, and reproduction
- Originally: bit-string based search spaces
- Different selection methods: Tournament better than Roulette-Wheel

Summary

- Genetic Algorithms are population-based metaheuristics inspired by natural evolution
- They proceed in a cycle of GPM, evaluation, selection, and reproduction
- Originally: bit-string based search spaces
- Different selection methods: Tournament better than Roulette-Wheel
- Crossover: different algorithms

Summary

- Genetic Algorithms are population-based metaheuristics inspired by natural evolution
- They proceed in a cycle of GPM, evaluation, selection, and reproduction
- Originally: bit-string based search spaces
- Different selection methods: Tournament better than Roulette-Wheel
- Crossover: different algorithms
- Mutation: Multi-point is good

Summary

- Genetic Algorithms are population-based metaheuristics inspired by natural evolution
- They proceed in a cycle of GPM, evaluation, selection, and reproduction
- Originally: bit-string based search spaces
- Different selection methods: Tournament better than Roulette-Wheel
- Crossover: different algorithms
- Mutation: Multi-point is good
- Schema theorem

Summary

- Genetic Algorithms are population-based metaheuristics inspired by natural evolution
- They proceed in a cycle of GPM, evaluation, selection, and reproduction
- Originally: bit-string based search spaces
- Different selection methods: Tournament better than Roulette-Wheel
- Crossover: different algorithms
- Mutation: Multi-point is good
- Schema theorem
- Many extensions

谢谢

Thank you

Thomas Weise [汤卫思]
tweise@hfuu.edu.cn
http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Institute of Applied Optimization
Shushan District, Hefei, Anhui,
China

Caspar David Friedrich, “Der Wanderer über dem Nebelmeer”, 1818
http://en.wikipedia.org/wiki/Wanderer_above_the_Sea_of_Fog

Bibliography

Bibliography I

1. Charles Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. London, UK: John Murray, 6th edition, November 24, 1859. URL `http://www.gutenberg.org/etext/1228`.
2. Nils Aaall Barricelli. Esempi numerici di processi di evoluzione. *Methodos*, 6(21–22):45–68, 1954.
3. Nils Aaall Barricelli. Symbiogenetic evolution processes realized by artificial methods. *Methodos*, 9(35–36):143–182, 1957.
4. Nils Aaall Barricelli. Numerical testing of evolution theories. part i. theroetical introduction and basic tests. *Acta Biotheoretica*, 16(1/2):69–98, March 1962. doi: 10.1007/BF01556771. Received: 27 November 1961.
5. Nils Aaall Barricelli. Numerical testing of evolution theories. part ii. preliminary tests of performance. symbiogenesis and terrestrial life. *Acta Biotheoretica*, 16(3/4):99–126, September 1963. doi: 10.1007/BF01556602. Received: 27 November 1961.
6. Alex S. Fraser. Simulation of genetic systems by automatic digital computers. i. introduction. *Australian Journal of Biological Science (AJBS)*, 10:484–491, 1957.
7. Alex S. Fraser. Simulation of genetic systems by automatic digital computers. ii. effects of linkage or rates of advance under selection. *Australian Journal of Biological Science (AJBS)*, 10:484–491, 1957.
8. David B. Fogel. In memoriam – alex s. fraser. *Evolutionary Computation*, 6(5):429–430, October 2002. doi: 10.1109/TEVC.2002.805212.
9. Hans J. Bremermann. Optimization through evolution and recombination. In Marshall C. Yovits, George T. Jacobi, and Gordon D. Goldstein, editors, *Self-Organizing Systems (Proceedings of the conference sponsored by the Information Systems Branch of the Office of Naval Research and the Armour Research Foundation of the Illinois Institute of Technology.)*, pages 93–103, Chicago, IL, USA, May 22–24, 1962. Washington, DC, USA: Spartan Books. URL `http://holtz.org/Library/Natural%20Science/Physics/`.
10. Woodrow "Woody" Wilson Bledsoe. Lethally dependent genes using instant selection. Technical Report PRI 1, Palo Alto, CA, USA: Panoramic Research, Inc., 1961.
11. Woodrow "Woody" Wilson Bledsoe. The use of biological concepts in the analytical study of systems. Technical Report PRI 2, Palo Alto, CA, USA: Panoramic Research, Inc., 1961. Presented at ORSA-TIMS National Meeting, San Francisco, California, November 10, 1961.
12. Woodrow "Woody" Wilson Bledsoe. An analysis of genetic populations. Technical report, Palo Alto, CA, USA: Panoramic Research, Inc., 1962.
13. Woodrow "Woody" Wilson Bledsoe. The evolutionary method in hill climbing: Convergence rates. Technical report, Palo Alto, CA, USA: Panoramic Research, Inc., 1962.

Bibliography II

14. John Henry Holland. Outline for a logical theory of adaptive systems. *Journal of the Association for Computing Machinery (JACM)*, 9(3):297–314. doi: 10.1145/321127.321128.
15. John Henry Holland. Adaptive plans optimal for payoff-only environments. In *Proceedings of the Second Hawaii International Conference on System Sciences (HICSS'69)*, pages 917–920, Honolulu, HI, USA: University of Hawaii at Manoa, January 22–24, 1969. Amsterdam, The Netherlands: North-Holland Scientific Publishers Ltd. DTIC Accession Number: AD0688839.
16. John Henry Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Ann Arbor, MI, USA: University of Michigan Press, 1975. ISBN 0-472-08460-7 and 978-0-472-08460-9. URL http://books.google.de/books?id=JE5RAAAAMAAJ.
17. John Henry Holland. Nonlinear environments permitting efficient adaptation. In Julius T. Tou, editor, *Proceedings of the Symposium on Computer and Information Sciences II*, pages 147–164, Columbus, OH, USA, August 22–24, 1966. London, New York: Academic Press.
18. Kenneth Alan De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, Ann Arbor, MI, USA: University of Michigan, August 1975. URL http://cs.gmu.edu/~eclab/kdj_thesis.html.
19. David Edward Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 0-201-15767-5 and 978-0-201-15767-3. URL http://books.google.de/books?id=2IIJAAAACAAJ.
20. Melanie Mitchell, Stephanie Forrest, and John Henry Holland. The royal road for genetic algorithms: Fitness landscapes and ga performance. In Francisco J. Varela and Paul Bourgine, editors, *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life (Actes de la Première Conférence Européenne sur la Vie Artificielle) (ECAL'91)*, Bradford Books, pages 245–254, Paris, France, December 11–13, 1991. Cambridge, MA, USA: MIT Press. URL http://web.cecs.pdx.edu/~mm/ecal92.pdf.
21. David Edward Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In Bruce M. Spatz and Gregory J. E. Rawlins, editors, *Proceedings of the First Workshop on Foundations of Genetic Algorithms (FOGA'90)*, pages 69–93, Bloomington, IN, USA: Indiana University, Bloomington Campus, July 15–18, 1990. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. URL http://www.cse.unr.edu/~sushil/class/gas/papers/Select.pdf.
22. Anne F. Brindle. *Genetic Algorithms for Function Optimization*. PhD thesis, Edmonton, Alberta, Canada: University of Alberta, 1980. Technical Report TR81-2.

Bibliography III

23. Lashon Bernard Booker. *Intelligent Behavior as an Adaptation to the Task Environment*. PhD thesis, Ann Arbor, MI, USA: University of Michigan. URL http://hdl.handle.net/2027.42/3746. Technical Report No. 243.
24. James E. Baker. Reducing bias and inefficiency in the selection algorithm. In John J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms and their Applications (ICGA'87)*, pages 14–21, Cambridge, MA, USA: Massachusetts Institute of Technology (MIT), July 28–31, 1987. Mahwah, NJ, USA: Lawrence Erlbaum Associates, Inc. (LEA).
25. John J. Grefenstette and James E. Baker. How genetic algorithms work: A critical look at implicit parallelism. In James David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms (ICGA'89)*, pages 20–27, Fairfax, VA, USA: George Mason University (GMU), June 4–7, 1989. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
26. Haiming Lu. *State-of-the-Art Multiobjective Evolutionary Algorithms – Pareto Ranking, Density Estimation and Dynamic Population*. PhD thesis, Stillwater, OK, USA: Oklahoma State University, Faculty of the Graduate College, August 2002. URL http://www.lania.mx/~ccoello/EMOO/thesis_lu.pdf.gz.
27. Garrison W. Greenwood, Xiaobo Sharon Hu, and Joseph G. D'Ambrosio. Fitness functions for multiple objective optimization problems: Combining preferences with pareto rankings. In Richard K. Belew and Michael D. Vose, editors, *Proceedings of the 4th Workshop on Foundations of Genetic Algorithms (FOGA'96)*, pages 437–455, San Diego, CA, USA: University of San Diego, August 5, 1996. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
28. A. Wetzel. *Evaluation of the Effectiveness of Genetic Algorithms in Combinatorial Optimization*. Pittsburgh, PA, USA: University of Pittsburgh, 1983. Unpublished manuscript, technical report.
29. Tobias Blickle and Lothar Thiele. A mathematical analysis of tournament selection. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA'95)*, pages 9–16, Pittsburgh, PA, USA: University of Pittsburgh, July 15–19, 1995. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. URL http://www.handshake.de/user/blickle/publications/tournament.ps.
30. Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4(4):361–394, Winter 1996. doi: 10.1162/evco.1996.4.4.361. URL http://www.handshake.de/user/blickle/publications/ECfinal.ps.
31. Brad L. Miller and David Edward Goldberg. Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, 4(2):113–131, Summer 1996. doi: 10.1162/evco.1996.4.2.113. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.3449.

Bibliography IV

32. S. Lee, S. Soak, K. Kim, H. Park, and M. Jeon. Statistical properties analysis of real world tournament selection in genetic algorithms. *Applied Intelligence – The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies*, 28(2):195–205, April 2008. doi: 10.1007/s10489-007-0062-2.
33. Kumara Sastry and David Edward Goldberg. Modeling tournament selection with replacement using apparent added noise. IlliGAL Report 2001014, Urbana-Champaign, IL, USA: University of Illinois at Urbana-Champaign, Department of Computer Science, Department of General Engineering, Illinois Genetic Algorithms Laboratory (IlliGAL), January 2001. URL http://www.illigal.uiuc.edu/pub/papers/IlliGALs/2001014.ps.Z.
34. Christopher K. Oei, David Edward Goldberg, and Shau-Jin Chang. Tournament selection, niching, and the preservation of diversity. IlliGAL Report 91011, Urbana-Champaign, IL, USA: University of Illinois at Urbana-Champaign, Department of Computer Science, Department of General Engineering, Illinois Genetic Algorithms Laboratory (IlliGAL), December 1991. URL http://www.illigal.uiuc.edu/pub/papers/IlliGALs/91011.ps.Z.
35. Hans-Georg Beyer. *The Theory of Evolution Strategies*. Natural Computing Series. New York, NY, USA: Springer New York, May 27, 2001. ISBN 3-540-67297-4 and 978-3-540-67297-5. URL http://books.google.de/books?id=8tbInLufkTMC.
36. Ingo Rechenberg. *Cybernetic Solution Path of an Experimental Problem*. Farnborough, Hampshire, UK: Royal Aircraft Establishment, August 1965. Library Translation 1122.
37. Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, Berlin, Germany: Technische Universität Berlin, 1971. URL http://books.google.de/books?id=QcNNGQAACAAJ.
38. Ingo Rechenberg. *Evolutionsstrategie '94*, volume 1 of *Werkstatt Bionik und Evolutionstechnik*. Bad Cannstadt, Stuttgart, Baden-Württemberg, Germany: Frommann-Holzboog Verlag, 1994. ISBN 3-7728-1642-8 and 978-3-772-81642-0. URL http://books.google.de/books?id=savAAAACAAJ.
39. Hans-Paul Schwefel. Kybernetische evolution als strategie der exprimentellen forschung in der strömungstechnik. Master's thesis, Berlin, Germany: Technische Universität Berlin, 1965.
40. Hans-Paul Schwefel. Experimentelle optimierung einer zweiphasendüse teil i. Technical Report 35, Berlin, Germany: AEG Research Institute, 1968. Project MHD–Staustrahlrohr 11.034/68.
41. Hans-Paul Schwefel. *Evolutionsstrategie und numerische Optimierung*. PhD thesis, Berlin, Germany: Technische Universität Berlin, Institut für Meß- und Regelungstechnik, Institut für Biologie und Anthropologie, 1975.

Bibliography V

42. Jörg Lässig, Karl Heinz Hoffmann, and Mihaela Enăchescu. Threshold selecting: Best possible probability distribution for crossover selection in genetic algorithms. In Maarten Keijzer, Giuliano Antoniol, Clare Bates Congdon, Kalyanmoy Deb, Benjamin Doerr, Nikolaus Hansen, John H. Holmes, Gregory S. Hornby, Daniel Howard, James Kennedy, Sanjeev P. Kumar, Fernando G. Lobo, Julian Francis Miller, Jason H. Moore, Frank Neumann, Martin Pelikan, Jordan B. Pollack, Kumara Sastry, Kenneth Owen Stanley, Adrian Stoica, El-Ghazali Talbi, and Ingo Wegener, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'08)*, pages 2181–2185, Atlanta, GA, USA: Renaissance Atlanta Hotel Downtown, July 12–16, 2008. New York, NY, USA: ACM Press. doi: 10.1145/1388969.1389044.
43. Jörg Lässig and Achim G. Hoffmann. Threshold-selecting strategy for best possible ground state detection with genetic algorithms. *Physical Review E*, 79(4):046702–046702–8, April 2009. doi: 10.1103/PhysRevE.79.046702.
44. John Henry Holland. Genetic algorithms – computer programs that "evolve" in ways that resemble natural selection can solve complex problems even their creators do not fully understand. *Scientific American*, 267(1):44–50, July 1992. URL http://www2.econ.iastate.edu/tesfatsi/holland.gaintro.htm.
45. Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies – a comprehensive introduction. *Natural Computing: An International Journal*, 1(1):3–52, March 2002. doi: 10.1023/A:1015059928466. URL http://www.cs.bham.ac.uk/~pxt/NIL/es.pdf.
46. Hans-Georg Beyer. An alternative explanation for the manner in which genetic algorithms operate. *Biosystems*, 41(1):1–15, January 1997. doi: 10.1016/S0303-2647(96)01657-7. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.307.
47. Thomas Weise. *Global Optimization Algorithms – Theory and Application*. Germany: it-weise.de (self-published), 2009. URL http://www.it-weise.de/projects/book.pdf.
48. Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*. Computational Intelligence Library. New York, NY, USA: Oxford University Press, Inc., Dirac House, Temple Back, Bristol, UK: Institute of Physics Publishing Ltd. (IOP), and Boca Raton, FL, USA: CRC Press, Inc., January 1, 1997. ISBN 0-7503-0392-1, 0-7503-0895-8, 978-0-7503-0392-7, and 978-0-7503-0895-3. URL http://books.google.de/books?id=n5nuiIZvmpAC.
49. Raymond Chiong, Thomas Weise, and Zbigniew Michalewicz, editors. *Variants of Evolutionary Algorithms for Real-World Applications*. Berlin/Heidelberg: Springer-Verlag, 2011. ISBN 978-3-642-23423-1 and 978-3-642-23424-8. doi: 10.1007/978-3-642-23424-8. URL http://books.google.de/books?id=B2ONePP4OMEC.
50. Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors. *Evolutionary Computation 1: Basic Algorithms and Operators*. Dirac House, Temple Back, Bristol, UK: Institute of Physics Publishing Ltd. (IOP), January 2000. ISBN 0750306645 and 9780750306645. URL http://books.google.de/books?id=4HMYCq9US78C.

Bibliography VI

51. Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors. *Evolutionary Computation 2: Advanced Algorithms and Operators*. Dirac House, Temple Back, Bristol, UK: Institute of Physics Publishing Ltd. (IOP), November 2000. ISBN 0750306653 and 9780750306652.
52. Dumitru (Dan) Dumitrescu, Beatrice Lazzerini, Lakhmi C. Jain, and A. Dumitrescu. *Evolutionary Computation*, volume 18 of *International Series on Computational Intelligence*. Boca Raton, FL, USA: CRC Press, Inc., June 2000. ISBN 0-8493-0588-8 and 978-0-8493-0588-7. URL `http://books.google.de/books?id=MSU9ep79JvUC`.
53. Ágoston E. Eiben, editor. *Evolutionary Computation*. Theoretical Computer Science. Amsterdam, The Netherlands: IOS Press, 1999. ISBN 4-274-90269-2, 90-5199-471-0, 978-4-274-90269-7, and 978-90-5199-471-1. URL `http://books.google.de/books?id=8LVAGQAACAAJ`. This is the book edition of the journal Fundamenta Informaticae, Volume 35, Nos. 1-4, 1998.
54. David Wolfe Corne, Marco Dorigo, Fred W. Glover, Dipankar Dasgupta, Pablo Moscato, Riccardo Poli, and Kenneth V. Price, editors. *New Ideas in Optimization*. McGraw-Hill's Advanced Topics In Computer Science Series. Maidenhead, England, UK: McGraw-Hill Ltd., May 1999. ISBN 0-07-709506-5 and 978-0-07-709506-2. URL `http://books.google.de/books?id=nC35AAAACAAJ`.
55. Ashish Ghosh and Shigeyoshi Tsutsui, editors. *Advances in Evolutionary Computing – Theory and Applications*. Natural Computing Series. New York, NY, USA: Springer New York, November 22, 2002. ISBN 3-540-43330-9 and 978-3-540-43330-9. URL `http://books.google.de/books?id=OGMEMC9P3vMC`.
56. Riccardo Poli, William Benjamin Langdon, and Nicholas Freitag McPhee. *A Field Guide to Genetic Programming*. London, UK: Lulu Enterprises UK Ltd, March 2008. ISBN 1-4092-0073-6 and 978-1-4092-0073-4. URL `http://www.lulu.com/items/volume_63/2167000/2167025/2/print/book.pdf`. With contributions by John R. Koza.
57. John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Bradford Books. Cambridge, MA, USA: MIT Press, December 1992. ISBN 0-262-11170-5 and 978-0-262-11170-6. URL `http://books.google.de/books?id=Bhtxo60BV0EC`. 1992 first edition, 1993 second edition.
58. Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming: An Introduction – On the Automatic Evolution of Computer Programs and Its Applications*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. and Heidelberg, Germany: dpunkt.verlag, November 30, 1997. ISBN 1-558-60510-X, 3-920993-58-6, and 978-1-558-60510-7. URL `http://books.google.de/books?id=1697qefFdtIC`.
59. Peter John Angeline and Kenneth E. Kinnear, Jr, editors. *Advances in Genetic Programming II*. Bradford Books. Cambridge, MA, USA: MIT Press, October 26, 1996. ISBN 0-262-01158-1 and 978-0-262-01158-7. URL `http://books.google.de/books?id=c3G7QgAACAAJ`.

Bibliography VII

60. Lee Spector, William Benjamin Langdon, Una-May O'Reilly, and Peter John Angeline, editors. *Advances in Genetic Programming III*. Bradford Books. Cambridge, MA, USA: MIT Press, July 16, 1996. ISBN 0-262-19423-6 and 978-0-262-19423-5. URL http://books.google.de/books?id=5Qwbal3AY6oC.
61. John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*, volume 5 of *Genetic Programming Series*. New York, NY, USA: Springer Science+Business Media, Inc., 2003. ISBN 0-387-25067-0, 0-387-26417-5, 1402074468, 6610611831, 978-0-387-25067-0, 978-0-387-26417-2, 978-1402074462, and 9786610611836. URL http://books.google.de/books?id=YQxWzAEnINIC.
62. Branko Souček and IRIS Group, editors. *Dynamic, Genetic, and Chaotic Programming: The Sixth-Generation*. Sixth Generation Computer Technologies. Chichester, West Sussex, UK: Wiley Interscience, April 1992. ISBN 047155717X and 978-0471557173. URL http://www.amazon.com/gp/reader/047155717X/ref=sib_dp_pt/002-6076954-4198445#reader-link.
63. Markus F. Brameier and Wolfgang Banzhaf. *Linear Genetic Programming*, volume 1 of *Genetic Algorithms and Evolutionary Computation*. Boston, MA, USA: Springer US and Norwell, MA, USA: Kluwer Academic Publishers, December 11, 2006. ISBN 0-387-31029-0, 0-387-31030-4, 978-0-387-31029-9, and 978-0-387-31030-5. doi: 10.1007/978-0-387-31030-5. Series Editor: David L. Goldberg, John R. Koza.
64. Carlos Artemio Coello Coello, Gary B. Lamont, and David A. van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*, volume 5 of *Genetic Algorithms and Evolutionary Computation*. Boston, MA, USA: Springer US and Norwell, MA, USA: Kluwer Academic Publishers, 2nd edition, 2002. ISBN 0306467623, 0387332545, 978-0306467622, 978-0-387-33254-3, and 978-0-387-36797-2. doi: 10.1007/978-0-387-36797-2. URL http://books.google.de/books?id=sgX_Cst_yTsC.
65. Joshua D. Knowles, David Wolfe Corne, and Kalyanmoy Deb. *Multiobjective Problem Solving from Nature – From Concepts to Applications*. Natural Computing Series. New York, NY, USA: Springer New York, 2008. ISBN 3-540-72963-1, 978-3-540-72963-1, and 978-3-540-72964-8. doi: 10.1007/978-3-540-72964-8. URL http://books.google.de/books?id=pzq8t9rCKC8C.
66. Ajith Abraham, Lakhmi C. Jain, and Robert Goldberg, editors. *Evolutionary Multiobjective Optimization – Theoretical Advances and Applications*. Advanced Information and Knowledge Processing. Berlin, Germany: Springer-Verlag GmbH. ISBN 1852337877 and 978-1-85233-787-2. URL http://books.google.de/books?id=Ei7q1YSjiSAC.

Bibliography VIII

67. Kenneth V. Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential Evolution – A Practical Approach to Global Optimization*. Natural Computing Series. Basel, Switzerland: Birkhäuser Verlag, 2005. ISBN 3-540-20950-6, 3-540-31306-0, 978-3-540-20950-8, and 978-3-540-31306-9. URL http://books.google.de/books?id=S67vX-KqVqUC.
68. Vitaliy Feoktistov. *Differential Evolution – In Search of Solutions*, volume 5 of *Springer Optimization and Its Applications*. New York, NY, USA: Springer New York, December 2006. ISBN 0-387-36895-7, 0-387-36896-5, 978-0-387-36895-5, and 978-0-387-36896-2. URL http://books.google.de/books?id=kG7aP_v-SU4C.
69. Martin Pelikan, Kumara Sastry, and Erick Cantú-Paz, editors. *Scalable Optimization via Probabilistic Modeling – From Algorithms to Applications*, volume 33 of *Studies in Computational Intelligence*. Berlin/Heidelberg: Springer-Verlag, 2006. ISBN 3-540-34953-7 and 978-3-540-34953-2. doi: 10.1007/978-3-540-34954-9. URL http://books.google.de/books?id=znzGDXPh6NAC.
70. Pedro Larrañaga and José Antonio Lozano, editors. *Estimation of Distribution Algorithms – A New Tool for Evolutionary Computation*, volume 2 of *Genetic Algorithms and Evolutionary Computation*. Boston, MA, USA: Springer US and Norwell, MA, USA: Kluwer Academic Publishers, 2001. ISBN 0-7923-7466-5 and 978-0-7923-7466-4. URL http://books.google.de/books?id=o0llxS4u93wC.
71. José Antonio Lozano, Pedro Larrañaga, Iñaki Inza, and Endika Bengoetxea, editors. *Towards a New Evolutionary Computation – Advances on Estimation of Distribution Algorithms*, volume 192/2006 of *Studies in Fuzziness and Soft Computing*. Berlin, Germany: Springer-Verlag GmbH, 2006. ISBN 3-540-29006-0 and 978-3-540-29006-3. doi: 10.1007/11007937. URL http://books.google.de/books?id=0dku9OKxl6AC.
72. Martin Pelikan. *Hierarchical Bayesian Optimization Algorithm – Toward a New Generation of Evolutionary Algorithms*, volume 170/2005 of *Studies in Fuzziness and Soft Computing*. Berlin, Germany: Springer-Verlag GmbH, 2005. ISBN 3-540-23774-7 and 978-3-540-23774-7. doi: 10.1007/b10910. URL http://books.google.de/books?id=_R0QHqcaTfIC.
73. William Eugene Hart, Natalio Krasnogor, and James E. Smith, editors. *Recent Advances in Memetic Algorithms*, volume 166/2005 of *Studies in Fuzziness and Soft Computing*. Berlin, Germany: Springer-Verlag GmbH, 2005. ISBN 3-540-22904-3 and 978-3-540-22904-9. doi: 10.1007/3-540-32363-5. URL http://books.google.de/books?id=LYf7YW4DmkUC.
74. Crina Grosan, Ajith Abraham, and Hisao Ishibuchi, editors. *Hybrid Evolutionary Algorithms*, volume 75/2007 of *Studies in Computational Intelligence*. Berlin/Heidelberg: Springer-Verlag, 2007. ISBN 3-540-73296-9 and 978-3-540-73296-9. doi: 10.1007/978-3-540-73297-6. URL http://books.google.de/books?id=II7LCqiGFlEC.