Evolutionary Computation

What is Evolutionary Computation (EC)?

The analogy:

Nature		Problem Solving
Environments	⟷	Problems
Individuals	⟷	Solutions
Fitness	⟷	Quality

- Fitness: How fit the individuals are.
- Quality: How good the solutions are.

Inspirations from Darwinian Evolution

- Survival of the fittest
  - Natural selection: Finite / limited resources
  - Compete efficiently → Chances to reproduce
- Phenotypic variations/diversity
  - Phenotypic traits
    - Behavioral or physical features
    - Inheritance, development, and random changes
    - Determine the fitness of individuals
  - "Good" traits tend to increase in future
  - Recombination

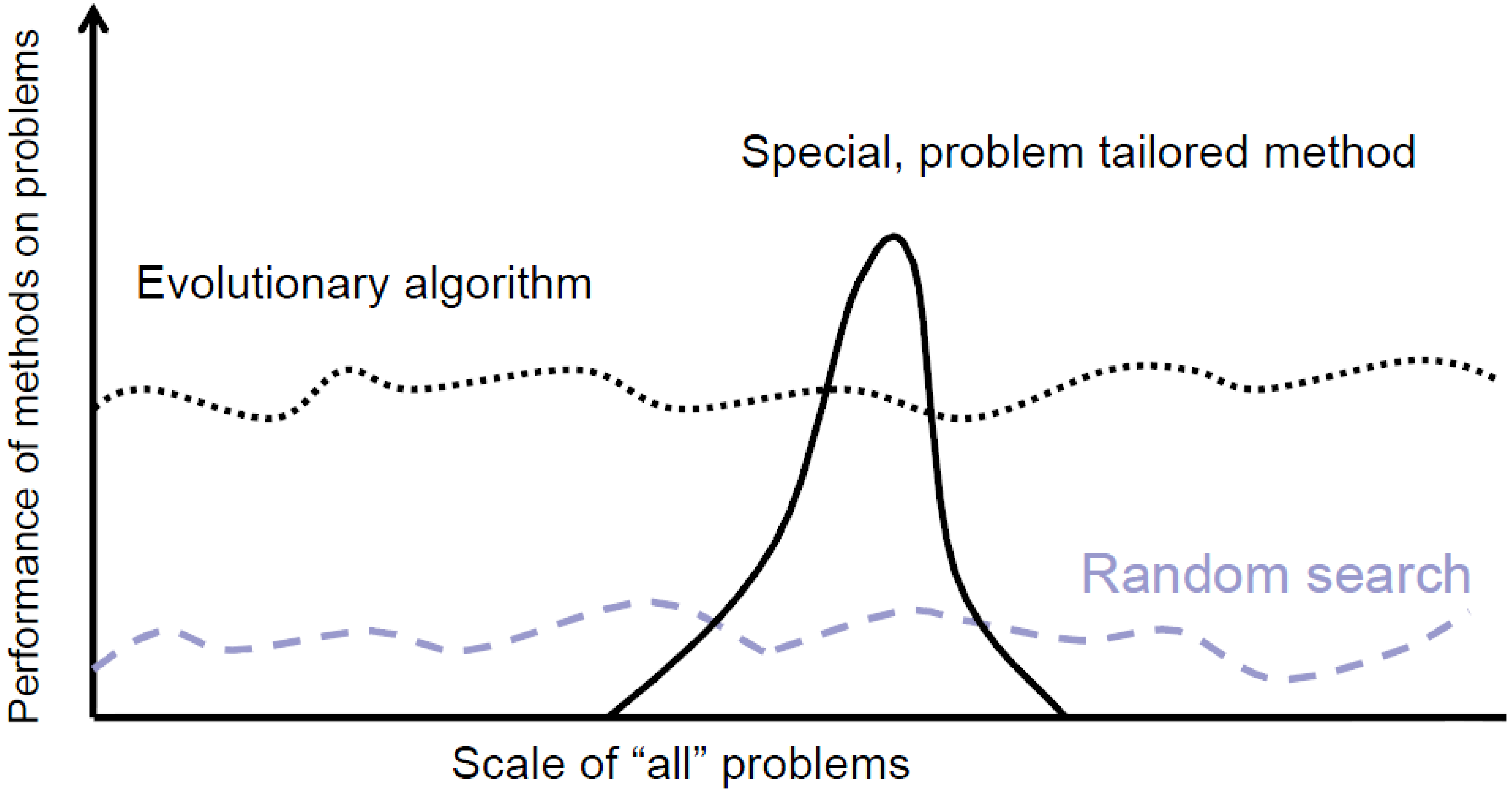
Inspirations from Genetics

- DNA: The code of life
  - Genotype → Inside, structure
  - Phenotype → Outside, property
- Gene: Functional unit
- Genome: Complete genetic material
  - Chromosome: DNA Strands
- Crossover
- Mutation

What's Special about EC?

- Uniform approach for different problems
  - No need to design or develop specific algorithms
- Black-box optimization
  - No need to analyze or understand the problem
  - Interactive, subjective, or even no function
- Population of individuals (solutions)
  - Non-deterministic in nature
  - Providing alternative solutions
- Multi-objective capability
  - Several objectives can be handled naturally
  - Providing solutions that cannot be compared

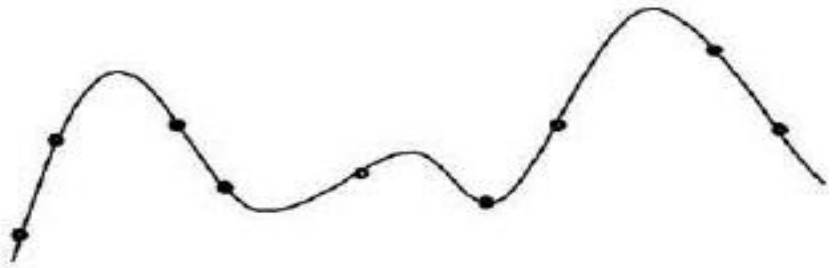
Applicability of Evolutionary Algorithms



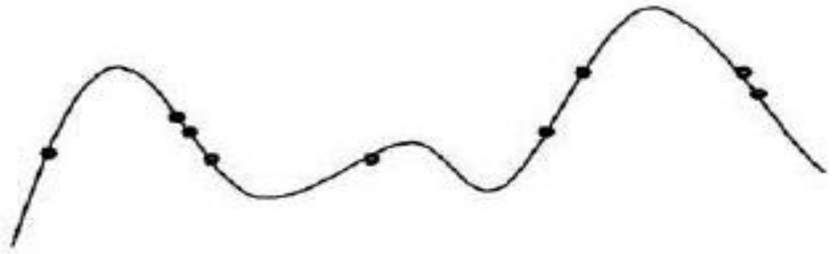
Progress in Evolutionary Algorithms

So, what actually happens?

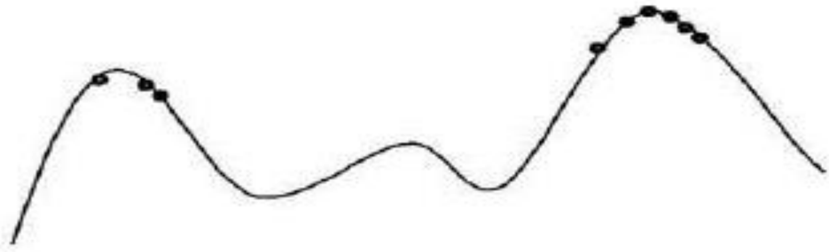
- Early phase:
  - Randomly distributed



- Mid-phase:
  - Around or on hills

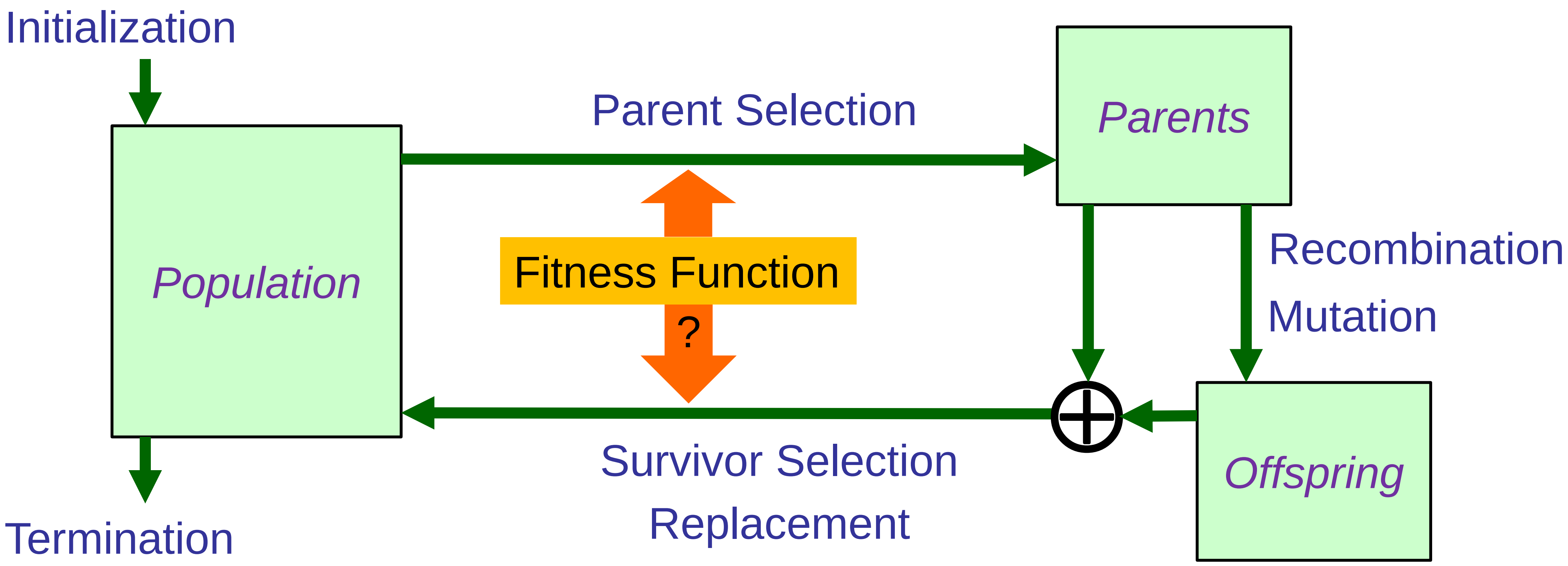


- Late phase:
  - Concentrated on hills



How Evolutionary Algorithms Work

A general scheme:



Population

- Unit of evolution
- Multiple chromosomes/individuals
- Fixed size or variable size
- Diversity
  - Many indicators/many ways to measure
- Initialization
  - Usually totally random to ensure enough raw genetic materials
  - Good initial points can be employed
  - Problem domain knowledge can be used

Selection and Replacement

- Selection: Select the parents to mate
  - Depends on the fitness in some way
  - Usually probabilistic
    - High fitness → High probability to reproduce
- Replacement: Select victims to be replaced
  - Usually deterministic
    - Fitness-based
    - Age-based (e.g., FIFO)
  - Elitism: The best individuals never die

Variation Operators

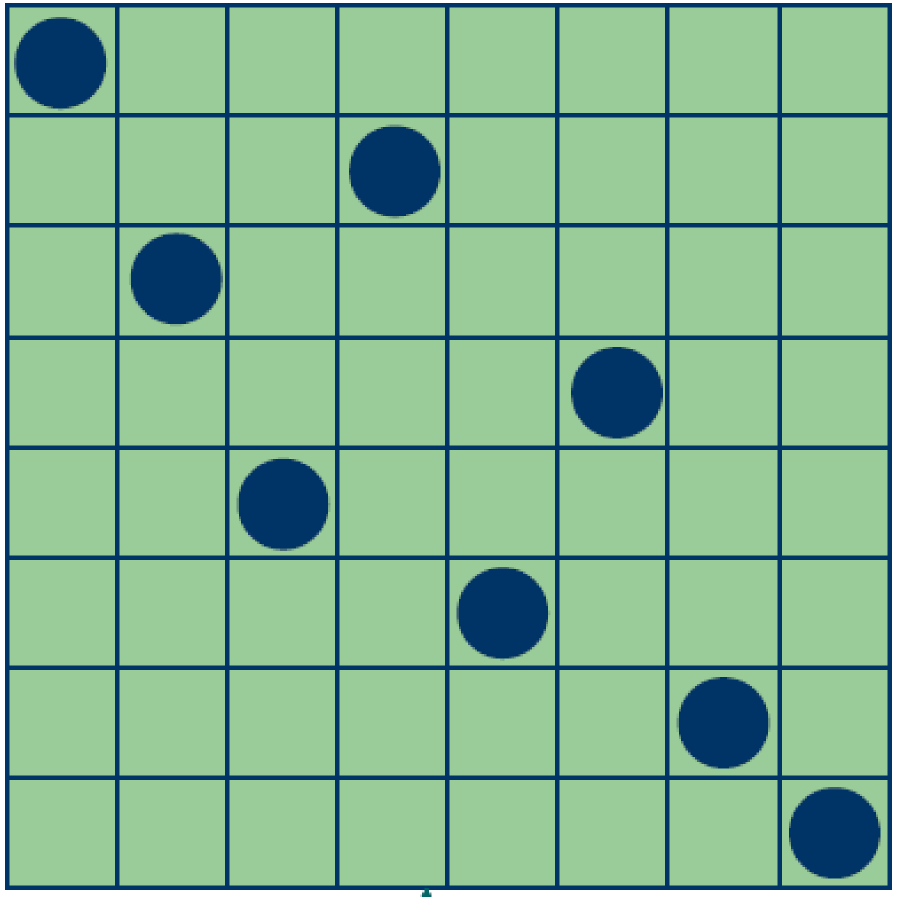
- Mutation: more of exploitation
  - Unary operation: One parent
  - Random changes → New stuff from nowhere
- Crossover: more of exploration
  - N-ary operation: N parents (usually N=2)
  - Non-deterministic
  - Explore combinations of traits/genes
- Mutation vs. Crossover
  - Relative importance?
  - Depends on representation
  - Use both in practice

Termination

- Common termination criteria:
  - Number of generations: Maximum allowed
  - Certain fitness/objective values
  - Estimator of diversity (check of convergence)

Example: 8-Queens

Representation:



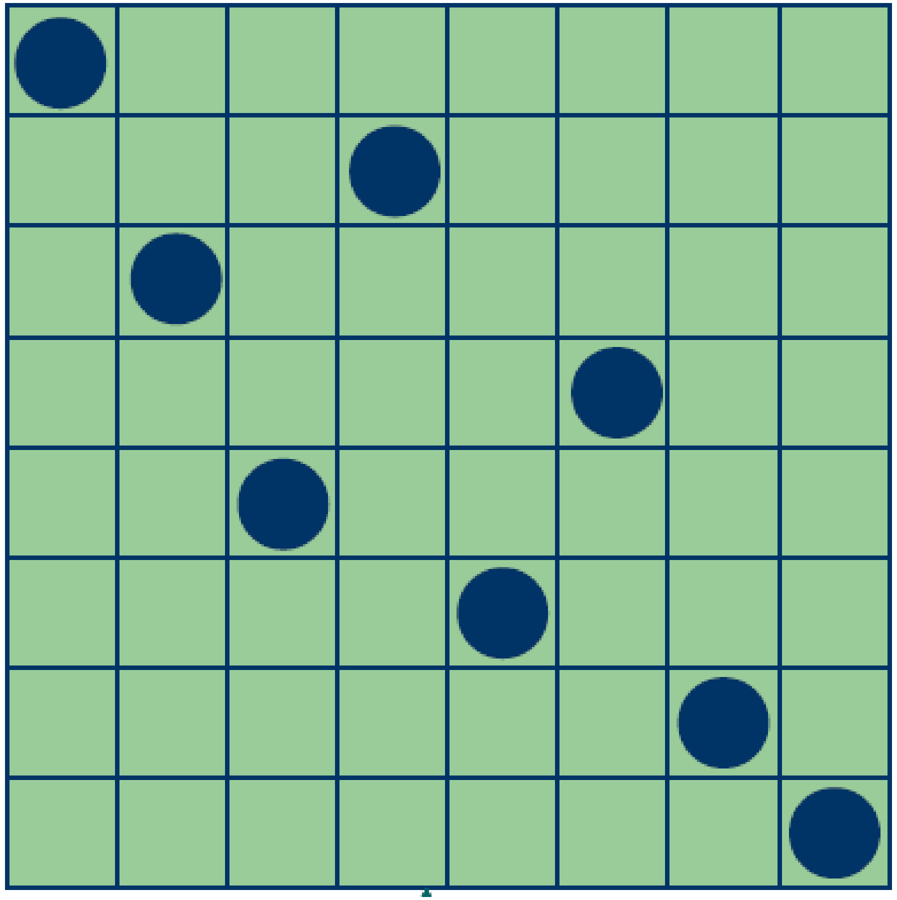
- **Phenotype**: A board configuration
- **Genotype**: A permutation of integers 1 … 8

1	3	5	2	6	4	7	8

Example: 8-Queens

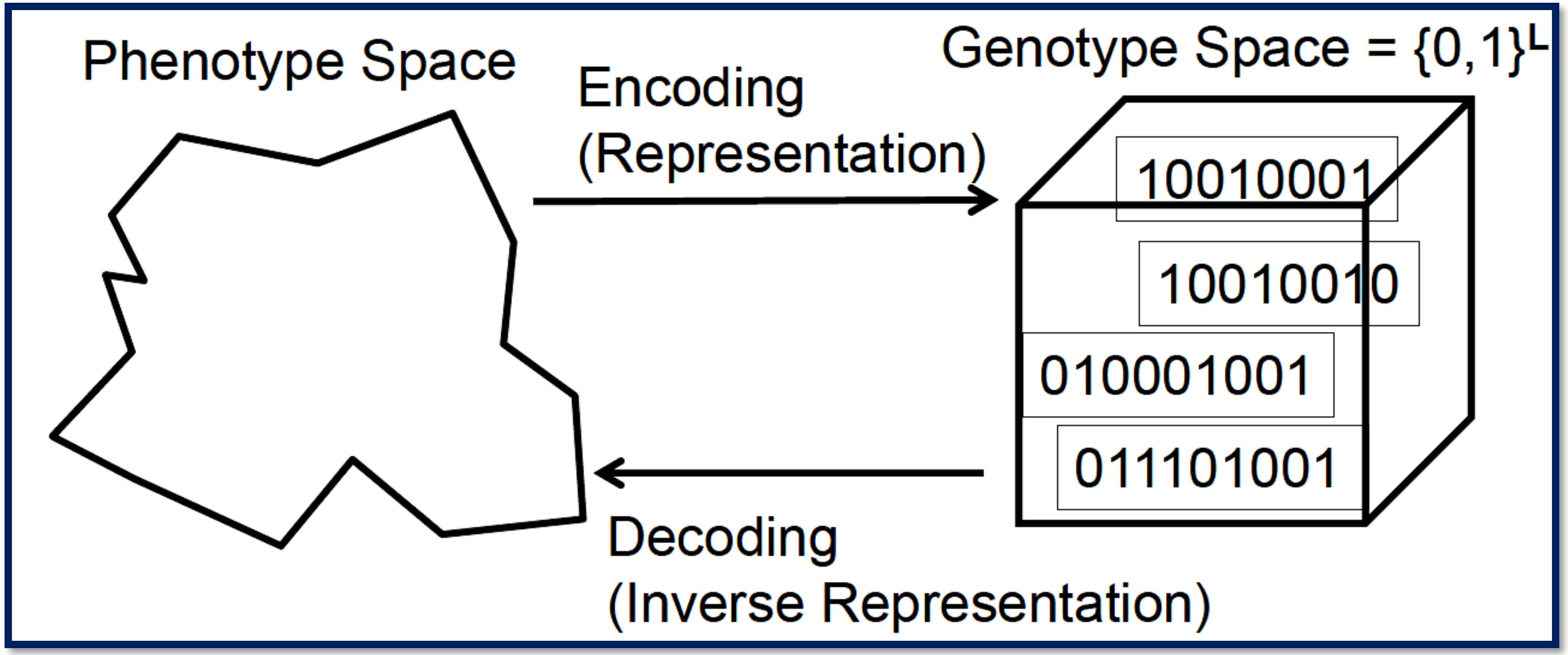
Fitness Function:

- Penalty:
  - Number of mutually attacking pairs
  - To be minimized
- Fitness:
  - Reverse the penalty in some way
  - To be maximized



Bit-String Representation

- Representation: Binary (bit) strings ➔ All kinds of information can be coded.



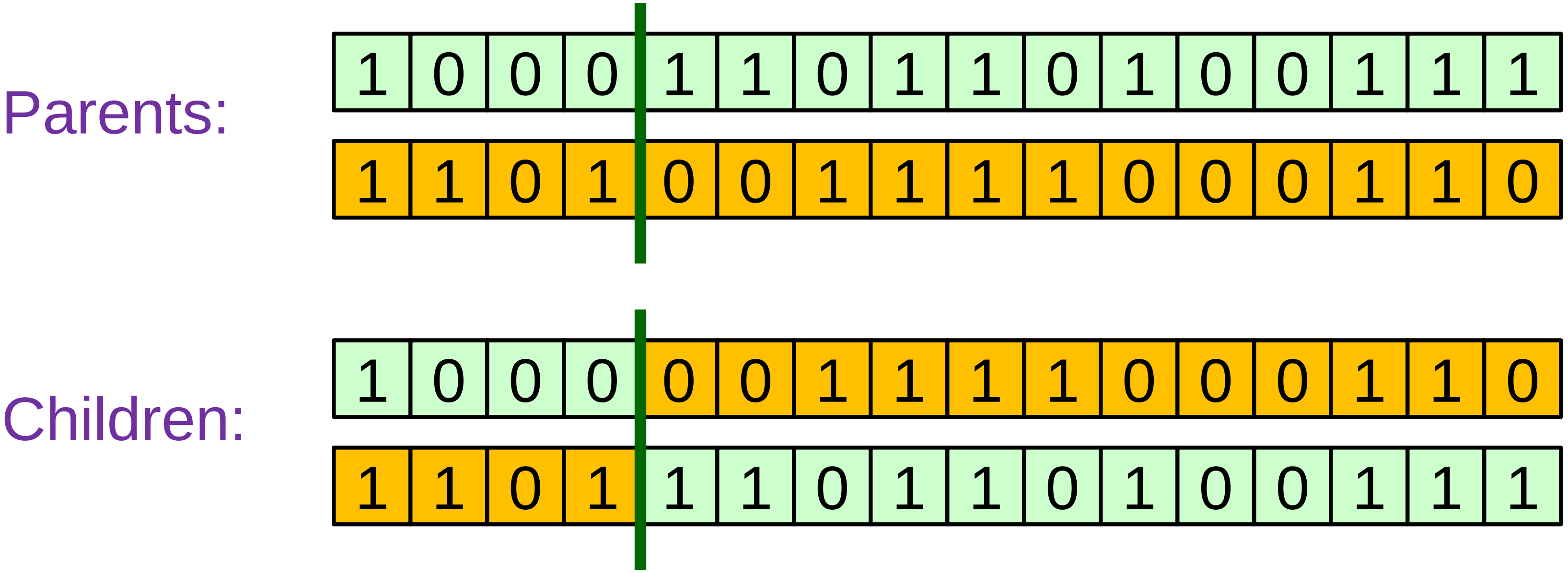
Bit-String Representation: Mutation

- Alter each gene independently with probability p_m.
- p_m typically is 1/n (n = number of bits)

Before:	1	0	0	0	1	1	0	1	1	0	1	0	0	1	1	1
After:	1	0	0	1	1	1	0	1	1	0	1	0	0	1	0	1

Bit-String Representation: Crossover

- 1-point crossover
  - Choose a random position on the two parents
  - Create children by exchanging tails
  - p_c typically in the range of [0.6, 0.9]
- Can have n-point crossover as well
- Nearby genes tend to stay together



Bit-String Representation: Crossover

- Uniform crossover: Dependence on loci (gene positions) removed.
  - For each bit, the parents exchange that gene with a probability.

Parents:	1	0	0	0	1	1	0	1	1	0	1	0	0	1	1	1
	1	1	0	1	0	0	1	1	1	1	0	0	0	1	1	0
Children:	1	1	0	0	0	0	0	1	1	0	0	0	0	1	1	1
	1	0	0	1	1	1	1	1	1	1	1	0	0	1	1	0

- With 50% probability, each child has 50% of genes from each of its parents.

Beyond Bit-String Representation

What can be in a gene?

- Integer representations:
  - Discrete variables (e.g., counts of items)
  - Enumeration or category variables
- Permutations (when the objective is to identify good “ordering”, such as the TSP problem)
- Real-number representations (lots of problems)
- Representation using bit strings still work
  - More bits → Longer chromosomes → Higher precision

Integer Representation

- Mutation:
  - Random change by one or some step size
  - Random number from its possible values
- Crossover:
  - N-point or uniform crossover
  - Arithmetic crossover (mean, etc.)

Real-Valued Representation

- Crossover:
  - Gene-wise crossover: just the same as when the genes are bits.
  - Arithmetic recombination (replace some children with weighted means of their respective source)
  - Blend recombination (2 children with opposite weights in arithmetic recombination)
  - Simplex recombination (>2 parents): Each child is a randomly selected point in the simplex formed by parents.

Real-Valued Representation: Mutation

- Standard method: Change the gene value by an amount drawn from a distribution (e.g., Gaussian)
- Some issues
  - Lower bound on the amount of change
  - Upper bound on the amount of change (constraint on gene value)
  - Adjusting the distribution of change during evolution:
    - Exploration: larger changes
    - Exploitation: smaller changes

Permutations

- Nature of problem: Orders, sequences, permutations
- Example problems: N-queens, travelling salesperson (TSP), job scheduling, etc.
- Two main types (different focus):
  - Order: e.g., sorting, job scheduling
  - Adjacency: e.g., TSP
- Different genetic operators have different effects on orders and adjacency.
  - Select the ones that are more suitable for the problem.

Permutations: Representation

- A sequence of integers: 1, 2, …, N
- Two different ways to code a permutation:
  - Example: 5 objects: A, B, C, D, E

    → permutation: C, B, E, A, D
  - Code the object index at each location: 3, 2, 5, 1, 4
  - Code the location of each object: 4, 2, 1, 5, 3

Permutations: Representation

- Permutation representation by **random keys**:
  - Use a set of distinct numbers (integers or real values)
  - Example: 5 objects: A, B, C, D, E

    → random keys: 0.32, 0.79, 0.04, 0.55, 0.73
  - We can get a permutation by sorting the keys of the objects.

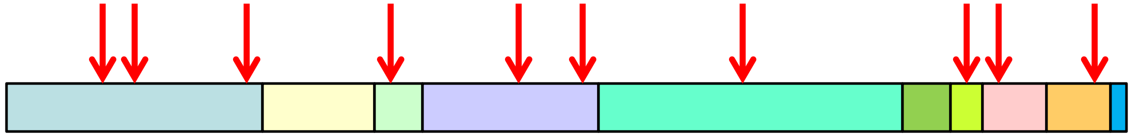
    → permutation: C, A, D, E, B
  - Genetic operators for real-valued genes are applicable here, as the genes are actually the random keys.

Parent Selection

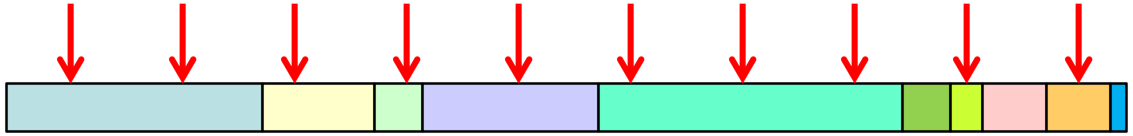
- Fitness based selection:

allocated size $\propto$ fitness value

  - **Roulette wheel selection**:



  - **Stochastic universal sampling**:



- Problem:
  - Superstar: The individual with very high fitness will dominate
    ➔ All individuals become similar
  - Solution: Shift the fitness values (add a constant)

Parent Selection

- **Ranking selection**:
  - Probability of being selected is based on ranking.
  - Example: Linear ranking: $$p(k) = \frac{2-s}{N} + \frac{2k(s-1)}{N(N-1)}$$
    - N: count
    - k: rank of an individual (1:worst; N:best)
    - s ($1<s\leq 2$): selection pressure

- **Truncation selection**:
  - For an integer $m>1$:
    - Keep only the best $1/m$ individuals
    - Duplicate these individuals m times

Parent Selection

- **Tournament selection**:
  - To select one individual:
    - Randomly pick k individuals (k = tournament size)
    - <u>Deterministic tournament</u>: Select the winner
    - <u>Probabilistic tournament</u>: Select from the participants with probabilities based on ranking
      - Example: $k = 2$, p is 0.8 for the winner and 0.2 for the loser.
  - Popular in practice
  - Advantages: no dependence on fitness values, no need to know the distribution of the whole population, easy to parallelize, etc.

Control Flow in EAs

- Generational EAs:
  - The entire parental population replaced in each generation.
- Steady-state EAs:
  - Only one or a few individuals are replaced in each generation.
  - "Survivor selection" / replacement done between generations.

Generation gap: The proportion of the population replaced.

Building Blocks Hypothesis

- Why do genetic algorithms work?
- Difficulty in combinatorial optimization
- First, we need to define a **schema** in GA:
  - A set of binary strings with values at some positions fixed. Example: `0**110**`
    - Order: Number of fixed positions (4 above)
    - Defining length: Distance between the last and the first fixed positions (5 above)

Building Blocks Hypothesis

- Building blocks: low-order, low-defining-length schemata that have above-average fitness.
- Propagation: Schemata are passed from parents to children.
- Crossover leads to the combination of good schemata (hopefully).
- The count (or share) of good building blocks will grow over generations.

The Concept of Metaheuristics

- Here we try to put EC in the larger scope of search / optimization algorithms.
- Heuristics:
  - Information BEYOND the problem definition that helps to guide the search / optimization process, increasing the likelihood and/or efficiency for finding good solutions.
  - Problem-specific
- Metaheuristics:
  - Procedural approach to guide the search / optimization process, also to increase the likelihood and/or efficiency for finding good solutions.
  - General purpose; usually applicable to a wide range of problems.

The Concept of Metaheuristics

- Representative metaheuristics:
  - Hill climbing.
  - Simulated annealing
  - Tabu search
  - Evolutionary computation
  - Swarm algorithms

The Concept of Metaheuristics

- Characteristics:
  - Single-solution (as improved local search) or population based (as approximated global search)
  - Population based methods require some mechanism for information exchange among the individuals.
  - Aims to find good solutions that might not be globally optimal.
  - Always includes some mechanism for tuning between exploration and exploitation.
  - **Memetic search**: Combining population based metaheuristics with local search/learning/refinement techniques.