AEEM 5117/6117: Intelligent Robotics

An Introduction To Genetic Algorithms

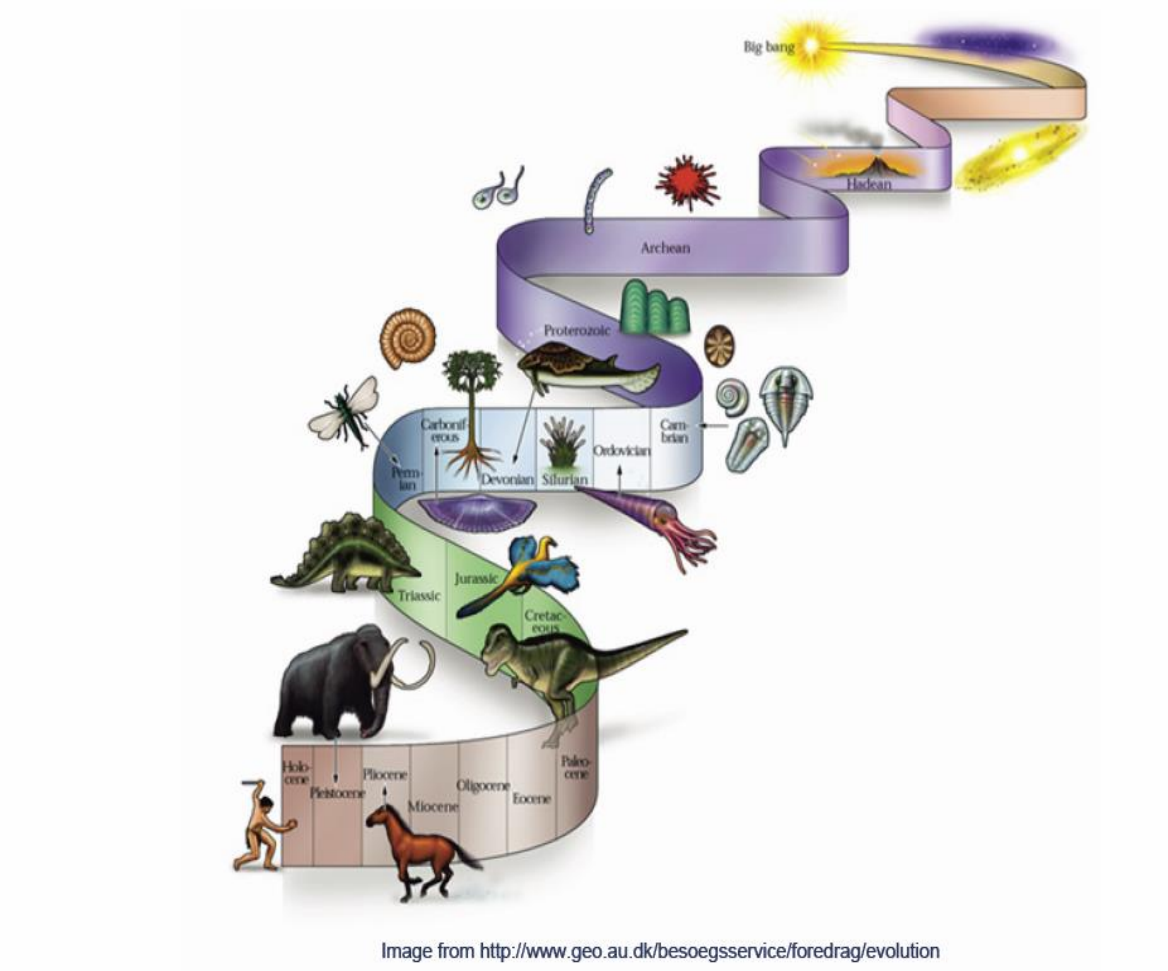
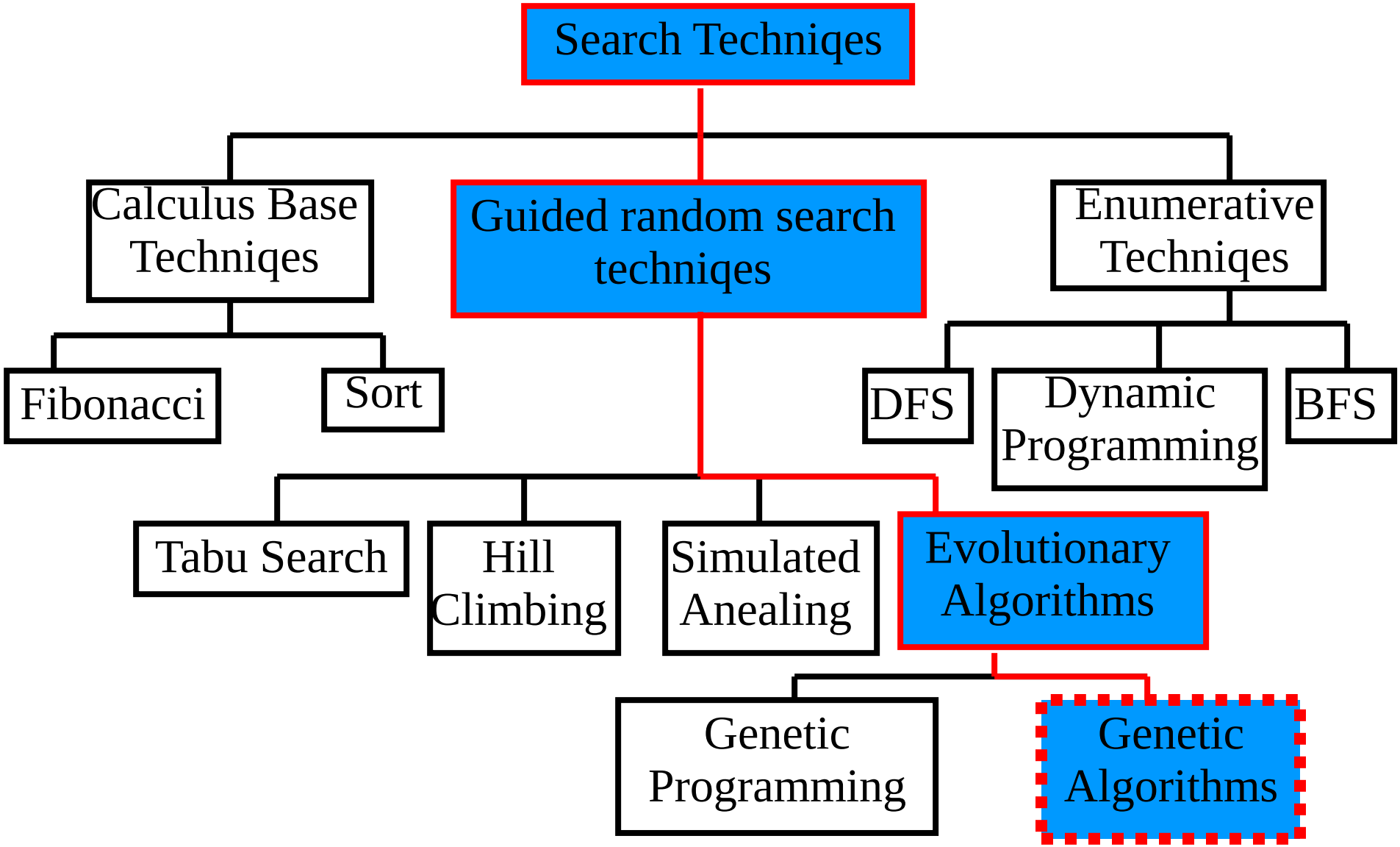


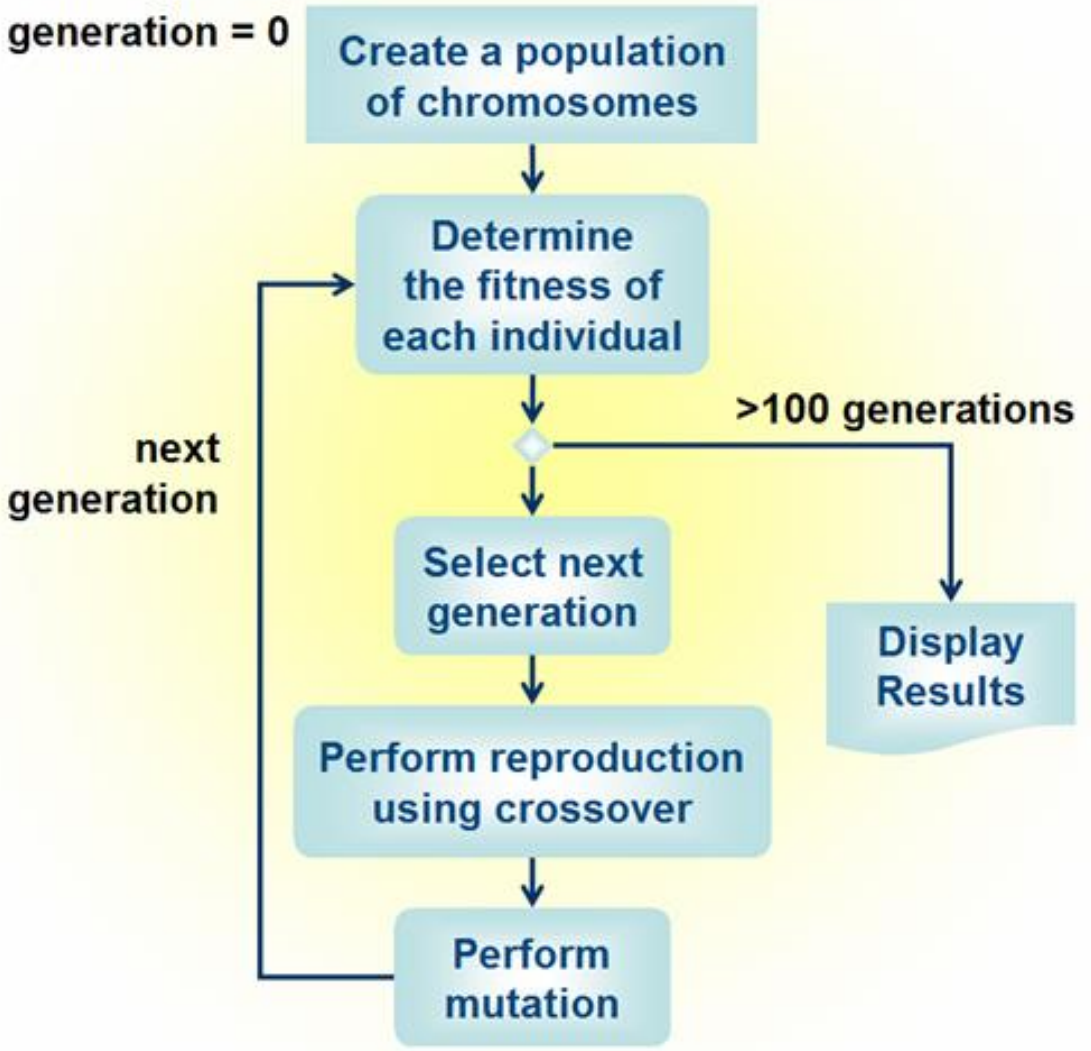
Image from http://www.geo.au.dk/besoegsservice/foredrag/evolution

Classes of Search Techniques



Introduction Genetic Algorithms

A genetic algorithm maintains a population of candidate solutions for the problem at hand, and makes it evolve by iteratively applying a set of stochastic operators



The Genetic Algorithms (GA)

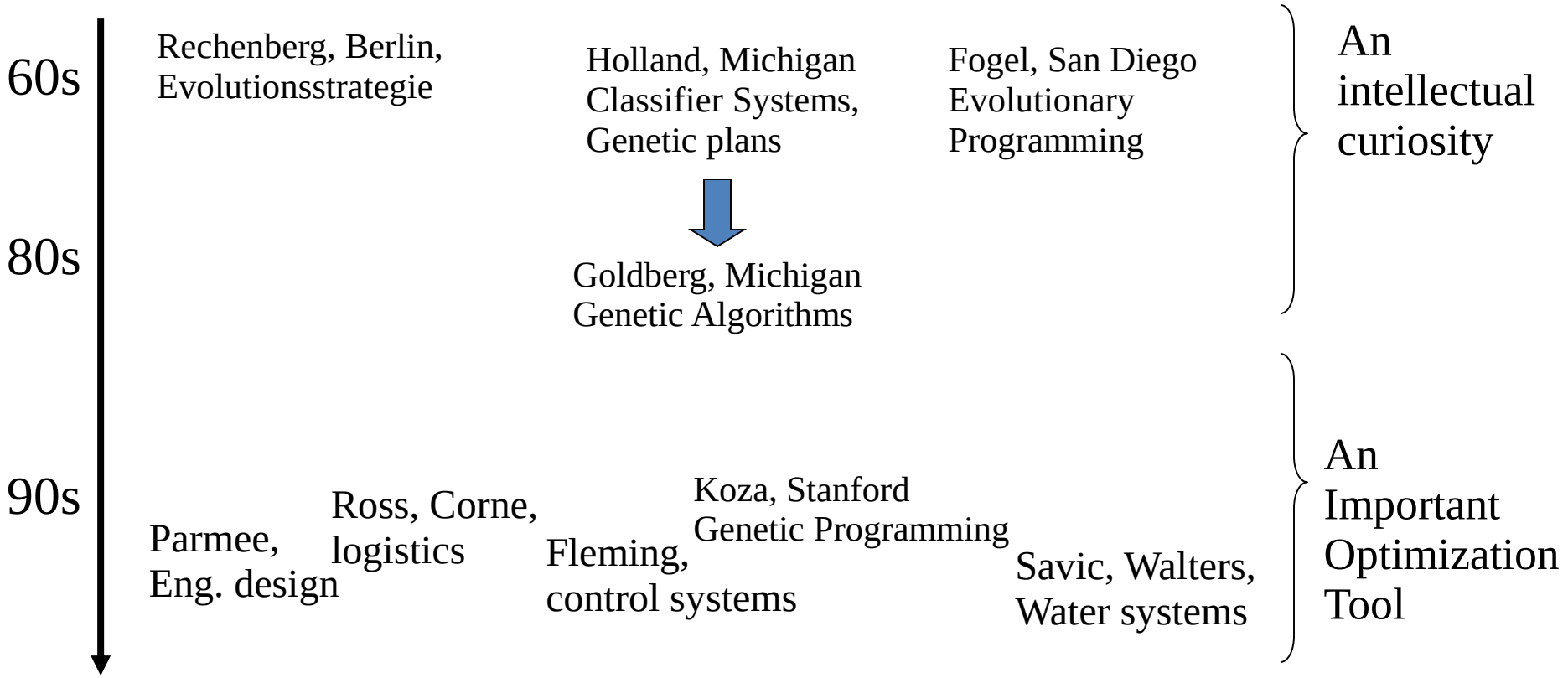
- Based on the mechanics of biological evolution
- Initially developed by John Holland, University of Michigan (1970's)
  - To understand processes in natural systems
  - To design artificial systems retaining the robustness and adaptation properties of natural systems

- Holland's original GA is known as the simple genetic algorithm (SGA)
- Provide efficient techniques for optimization and machine learning applications
- Widely used in business, science and engineering

Image adapted from http://today.mun.ca/news.php?news_id=2376

The Evolutionary Computation Fossil Record

The first published ideas using evolution in optimisation came in the 50s. But the lineage of current algorithms is like this:



Genetic Algorithms Techniques

- GAs are a particular class of evolutionary algorithms. The techniques common to all GAs are:
  - Inheritance
  - Mutation
  - Selection
  - Crossover (also called recombination)
- GAs are best used when the objective function is:
  - Discontinuous
  - Highly nonlinear
  - Stochastic
  - Has unreliable or undefined derivatives

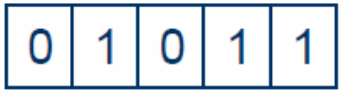
Performance

- GAs can provide solutions for highly complex search spaces
- GAs perform well approximating solutions to all types of problems because they do not make any assumption about the underlying fitness landscape (the shape of the fitness function, or objective function)
- However, GAs can be outperformed by more field-specific algorithms

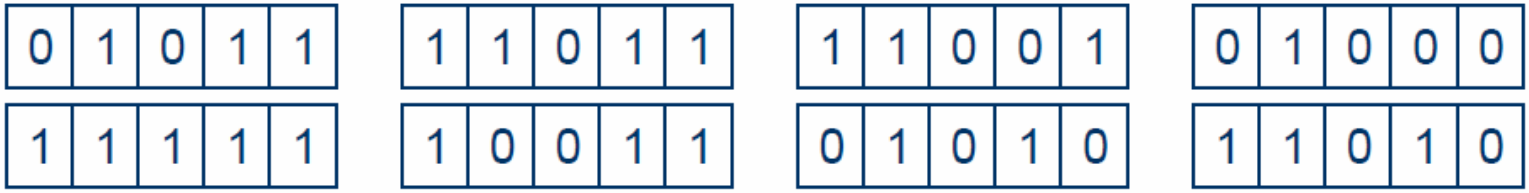
Biological Terminology

- Gene – a single encoding of part of the solution space, i.e. either single bits or short blocks of adjacent bits that encode an element of the candidate solution

- Chromosome – a string of genes that represents a solution

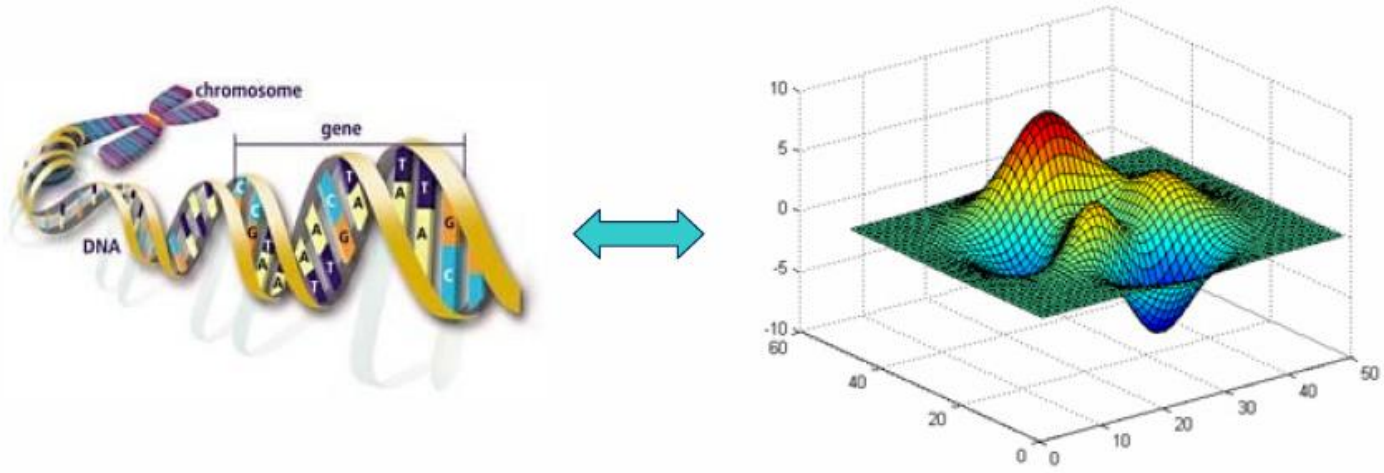


- Population – the number of chromosomes available to test



Biology vs. Optimization

- Candidate solutions to the optimization problem play the role of individuals in a population (or chromosomes)
- Cost/fitness/objective function determines the environment within which the solutions "live"



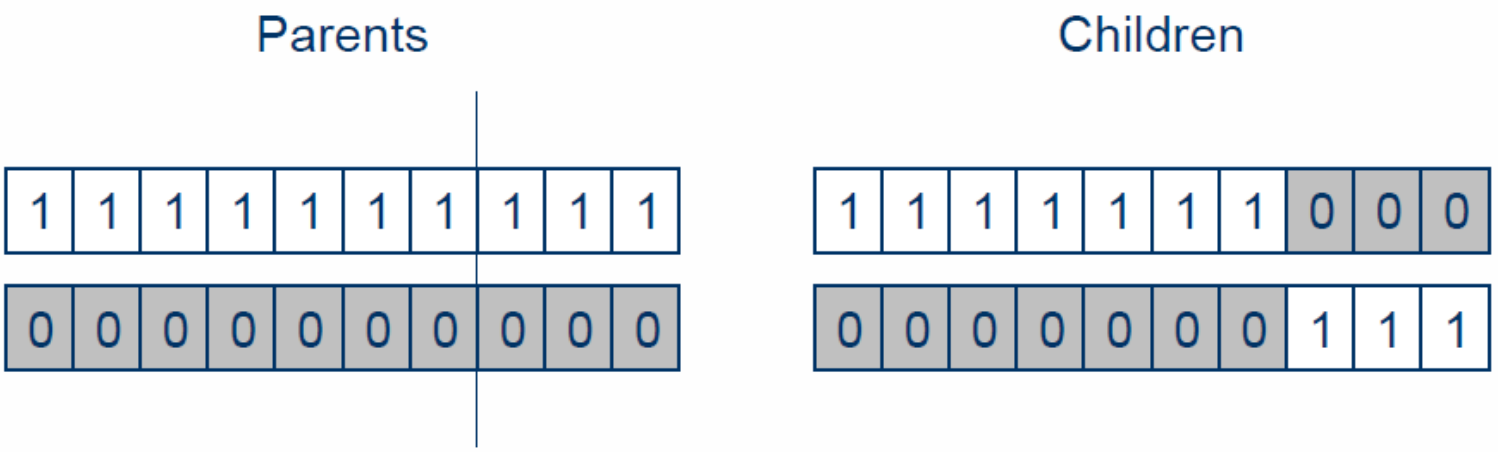
DNA image from http://static.howstuffworks.com/gif/cell-dna.jpg

Features of Genetic Algorithms

- Not too fast but cover large search space
  - Capable of quickly finding promising regions of the search space but may take a relatively long time to reach the optimal solution. Solution: hybrid algorithms
- Good heuristics for combinatorial problems
- Usually emphasize combining information from good parents (crossover)
- Different GAs use different
  - Representations
  - Mutations
  - Crossovers
  - Selection mechanisms

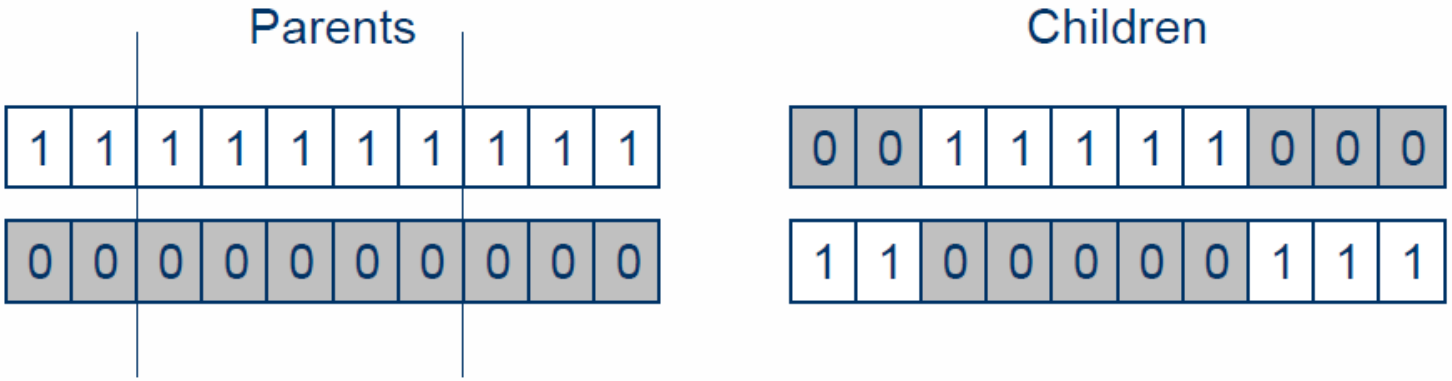
1-Point Crossover

- Choose a random point
- Split parents at this crossover point
- Create children by exchanging tails
- Probability of crossover is typically in range (0.6, 0.9)

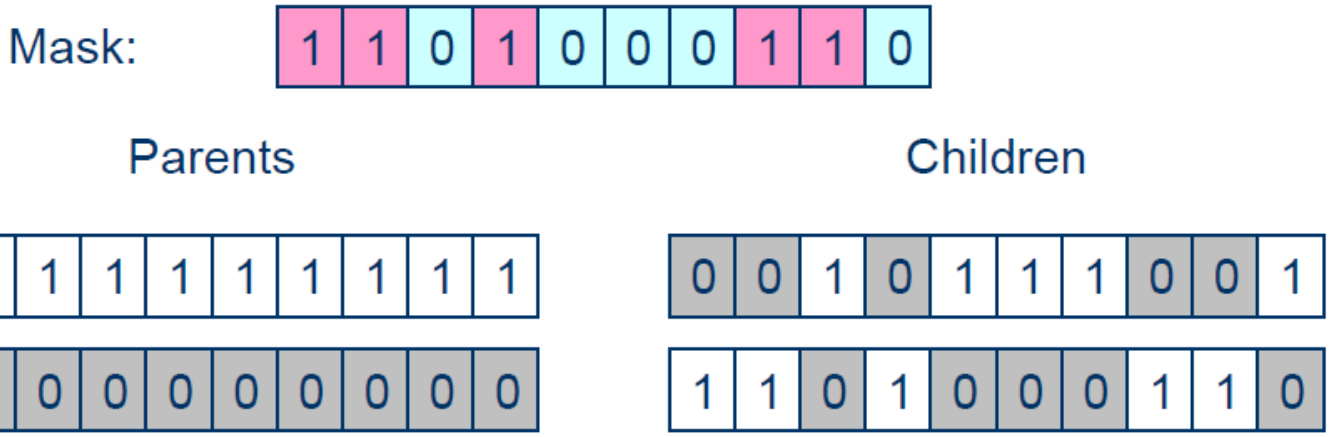


Other Crossover Types

- Two-point crossover

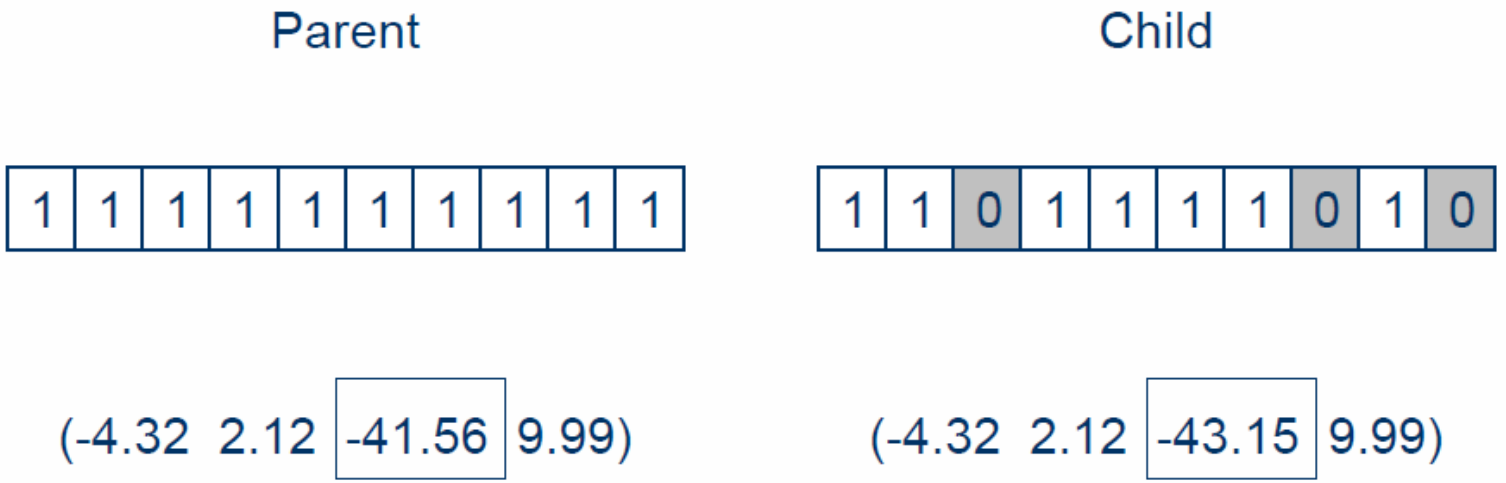


- Uniform crossover: randomly generated mask



Mutation

- Alter each gene independently
- Mutation probability is typically in range (1/population_size, 1/chromosome_length)

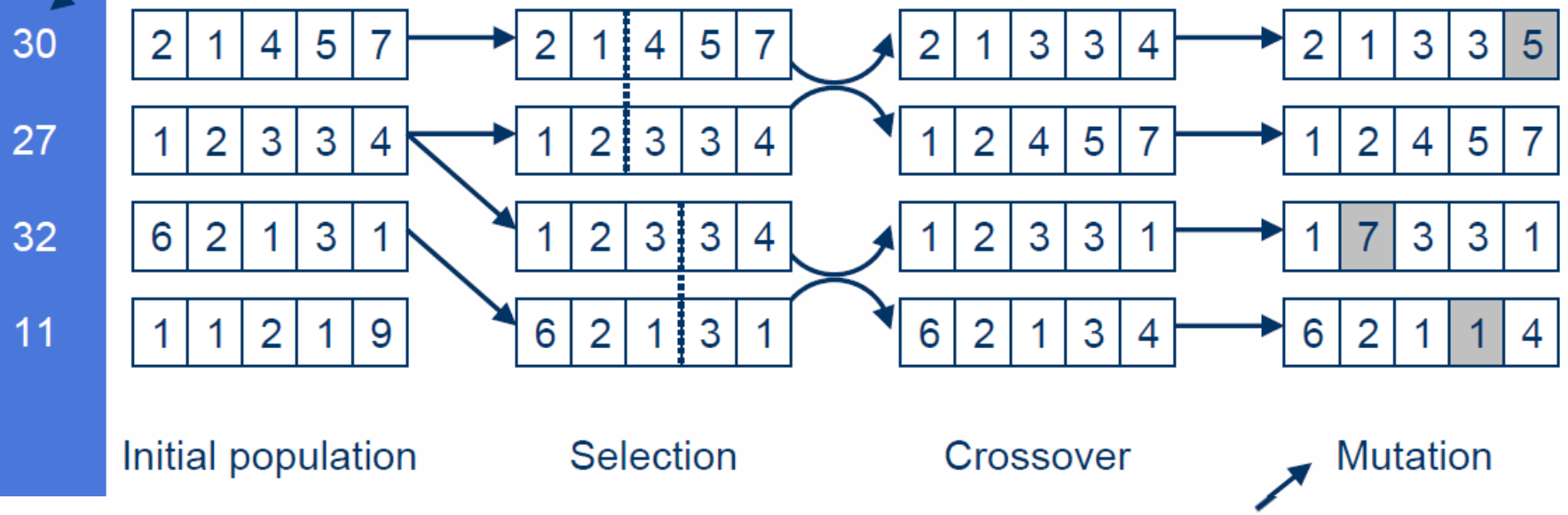


- Choose mutation with the best fit

Selection

- Parents with better fitness have better chances to produce offspring

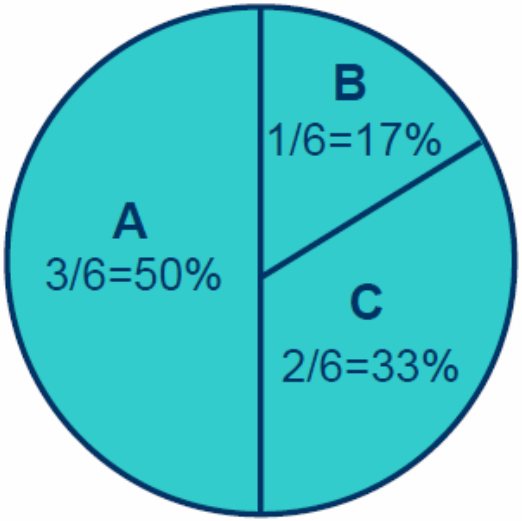
Fitness function – measure of goodness of the candidate solution



Mutation can be performed in a way that maximizes fitness function

Selection: Roulette Wheel

- Better solutions get higher chance to become parents for next generation solutions
- Roulette wheel technique:
  - Assign each individual part of the wheel
  - Spin wheel N times to select N individuals



fitness(A) = 3

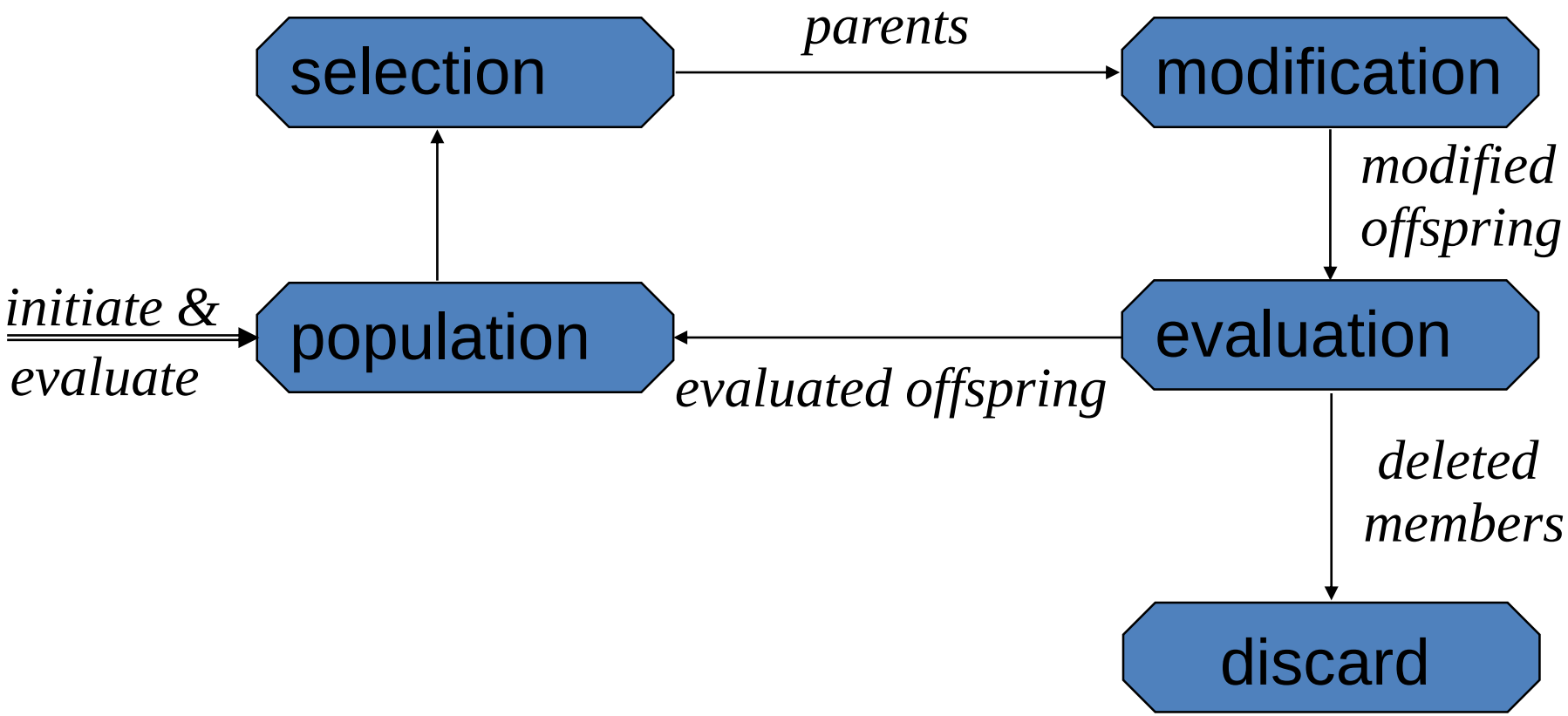
fitness(B) = 1

fitness(C) = 2

The Basic Genetic Algorithm

```
{   % Generate random population of chromosomes
    Initialize population;
    % Evaluate the fitness of each chromosome in the population
    Evaluate population;                                [Fitness]
    % Create, accept, and test a new population:
    while Termination_Criteria_Not_Satisfied
    {       % Select according to fitness
            Select parents for reproduction;            [Selection]
            % With a crossover probability perform crossover or copy parents
            Perform crossover;                          [Crossover]
            % With a mutation probability mutate offspring at each position in chromosome
            Perform mutation;                           [Mutation]
            Accept new generation;
            Evaluate population;                        [Fitness]
    }
}
```

The Evolutionary Cycle



A Generic Evolutionary Algorithm

Suppose you have to find a solution to some problem or other, and suppose, given any candidate solution *s* you have a function *f*(*s*) which measures how good *s* is as a solution to your problem.

Generate an initial population *P* of randomly generated solutions (this is typically 100 or 500 or so). Evaluate the fitness of each. Then:

Repeat until a termination condition is reached:

1. Selection: Choose some of *P* to be *parents*

2. Variation: Apply genetic operators to the parents to produce some *children*, and then evaluate the fitness of the children.

3. Population update: Update the population *P* by retaining some of the children and removing some of the incumbents.

Basic Varieties of Evolutionary Algorithm

1. Selection: Choose some of *P* to be *parents*

There are many different ways to select – e.g. choose top 10% of the population; choose with probability proportionate to fitness; choose randomly from top 20%, etc …

2. Variation: Apply genetic operators …

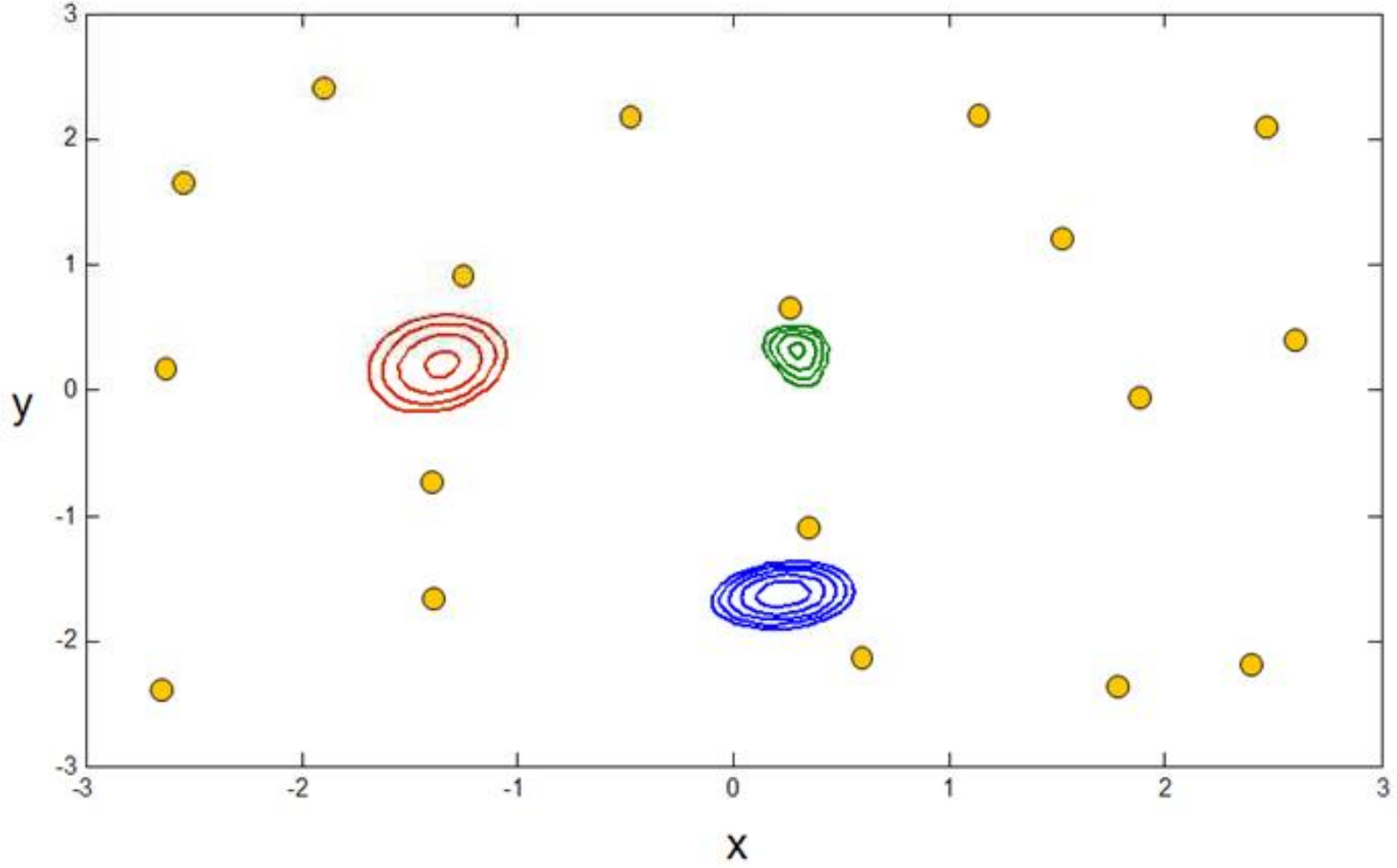
There are many different ways to do this, and it depends much on the encoding (see next slide). We will learn certain standard ways.

3. Population update: Update the population *P* by …

There are many several ways to do this, e.g. replace entire population with the new children; choose best |P| from *P* and the new ones, etc.

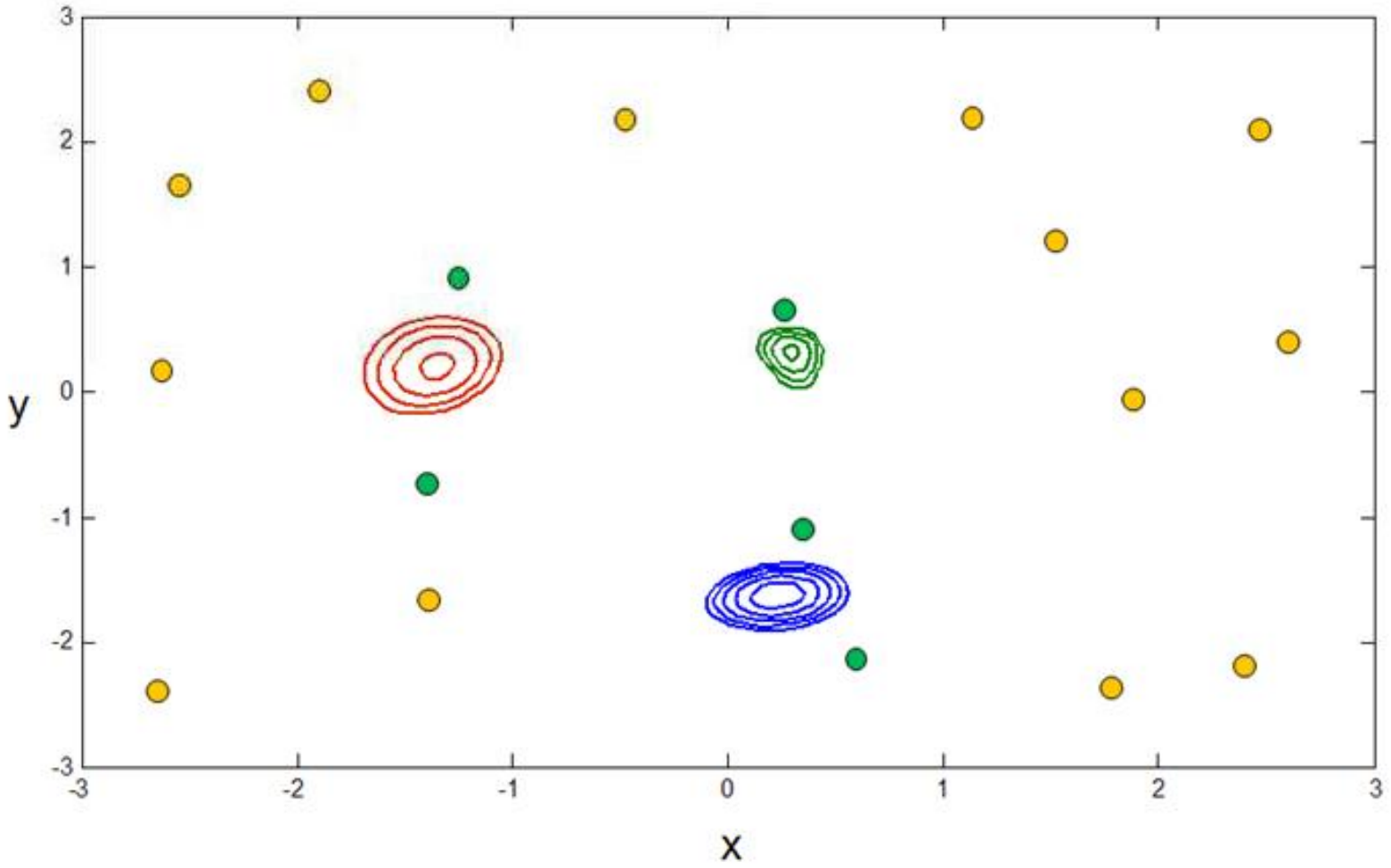
Genetic Algorithm – Iteration 1

Evaluate initial population



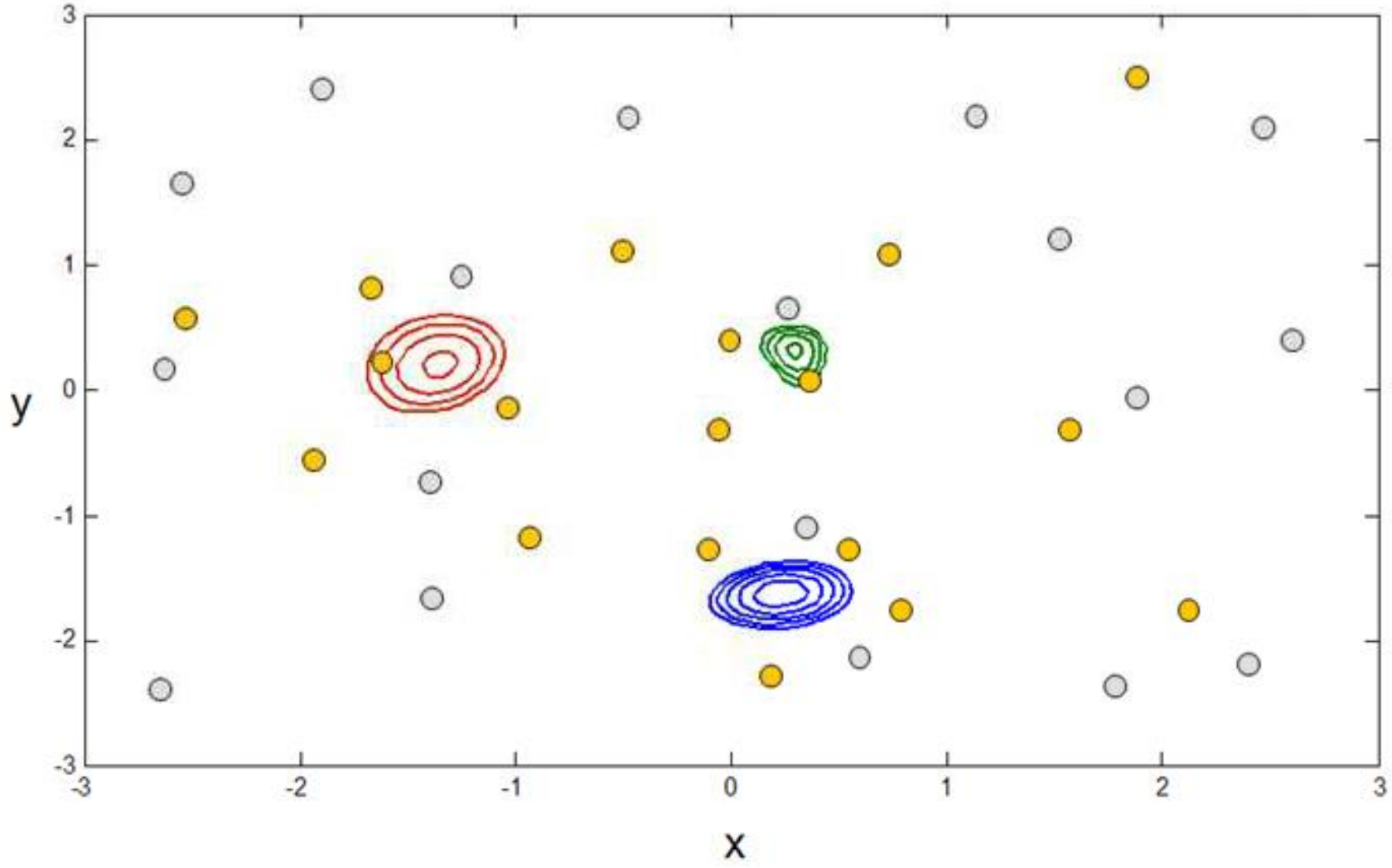
Genetic Algorithm – Iteration 1

Select a few good solutions for reproduction



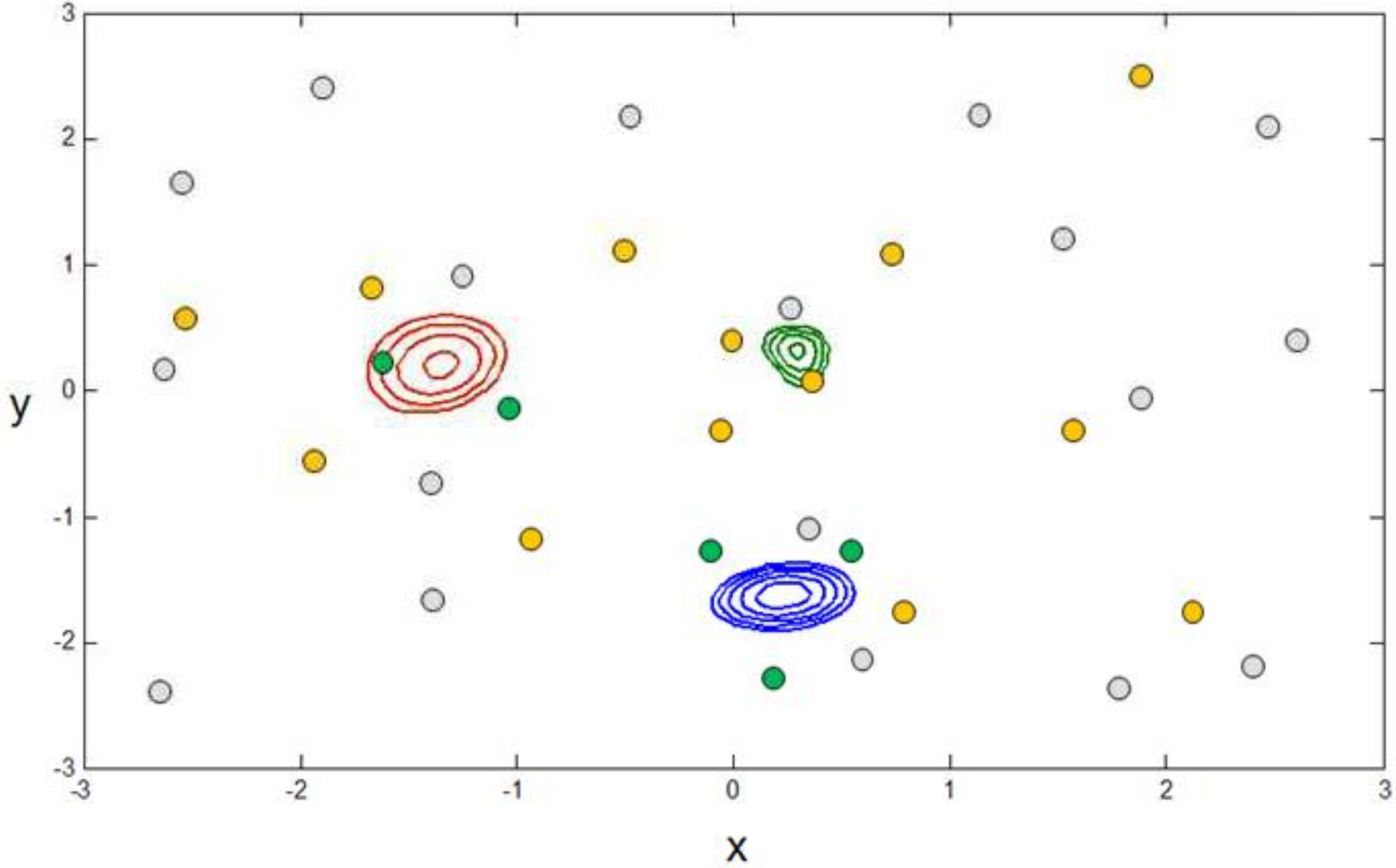
Genetic Algorithm – Iteration 2

Generate new population and evaluate



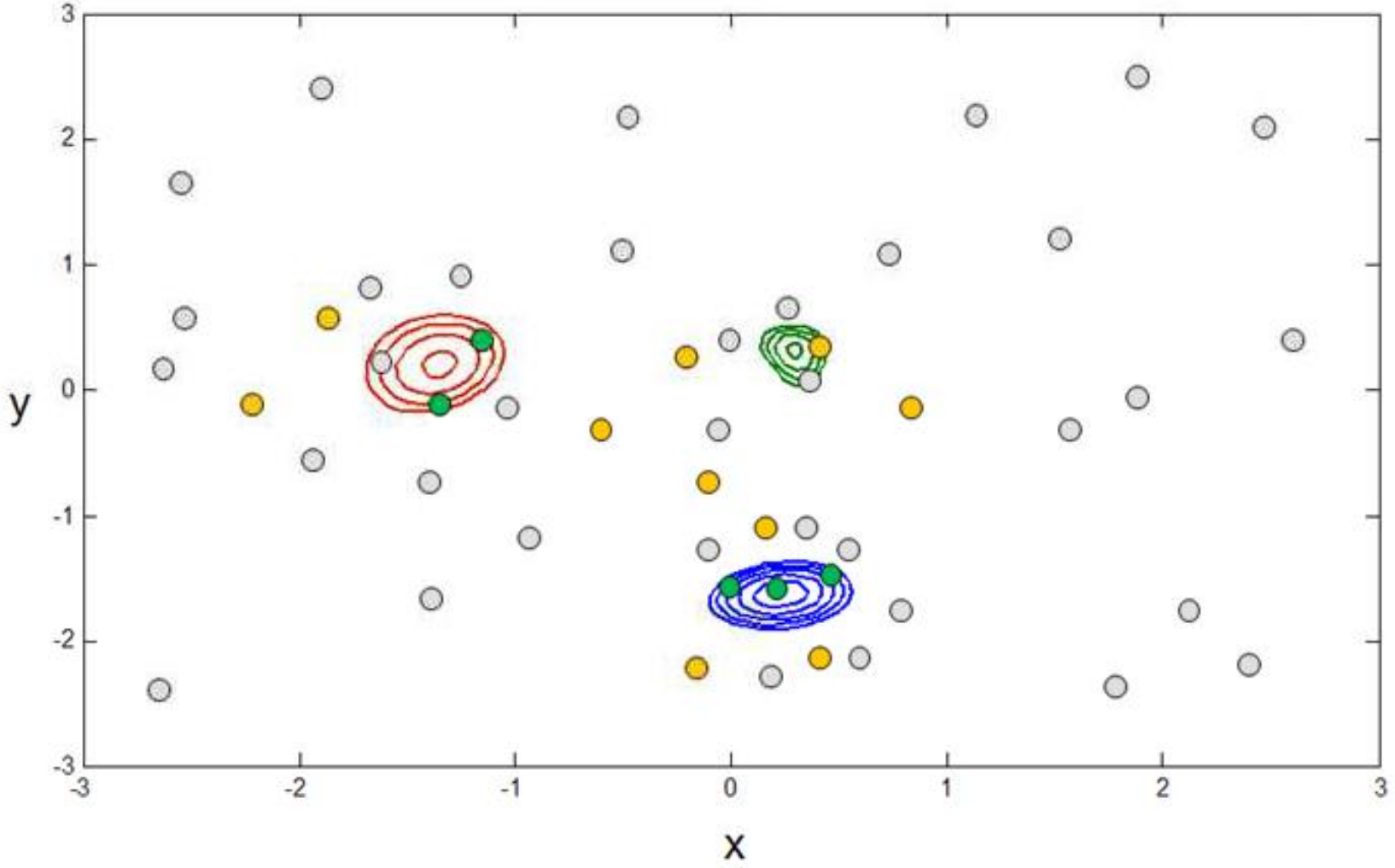
Genetic Algorithm – Iteration 2

Select a few good solutions for reproduction



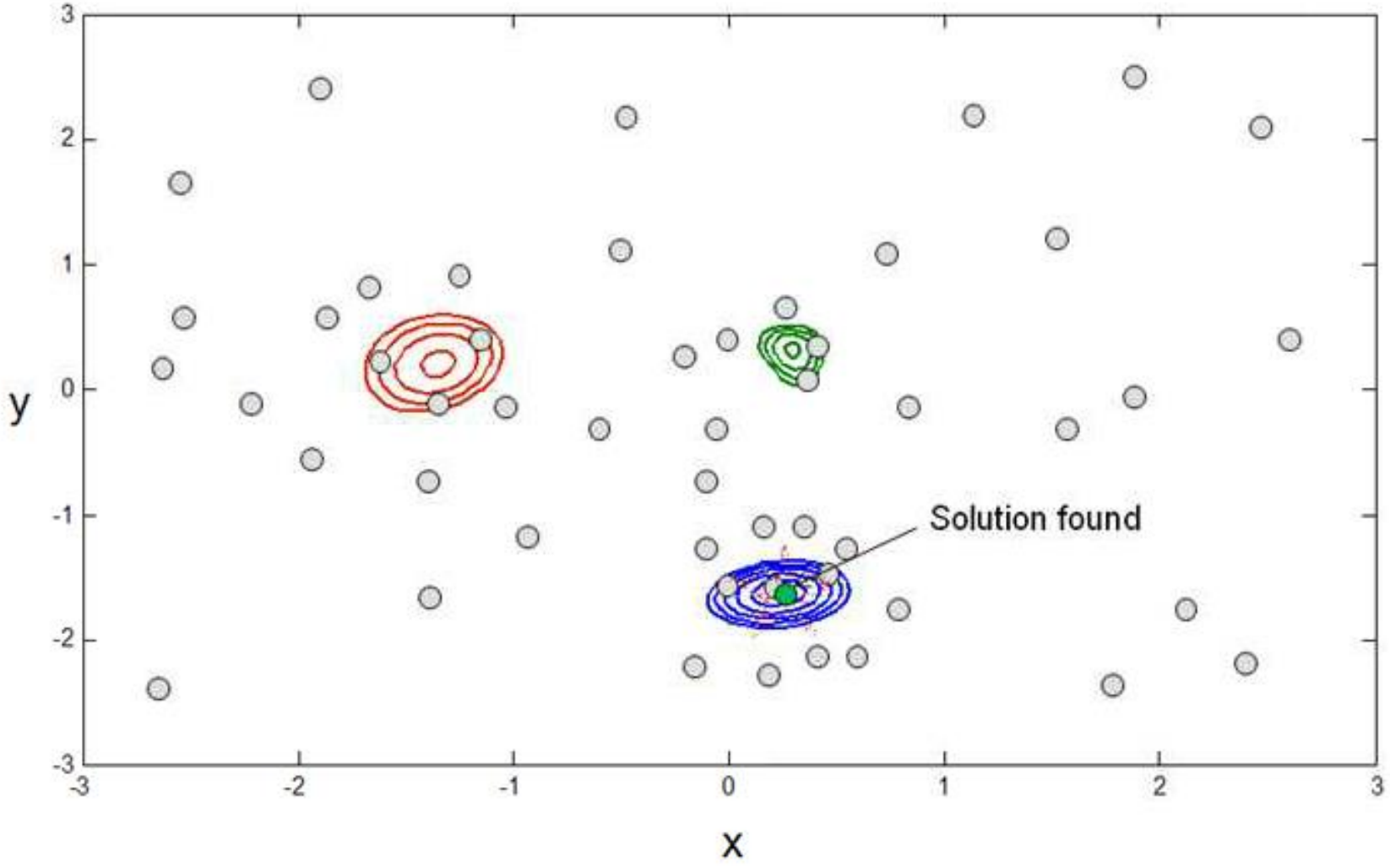
Genetic Algorithm – Iteration 3

Select a few good solutions for reproduction



Genetic Algorithm – Iteration N

Continue process until stopping criteria are met



What are they good for ?

Suppose we want the best possible schedule for a **university lecture timetable**.

- Or the best possible **pipe network** design for a ship's engine room
- Or the best possible design for an **antenna** with given requirements
- Or a **formula** that fits a curve better than any others
- Or the best design for a **comms network** in terms of reliability for given cost
- Or the best **strategy for flying a fighter aircraft**
- Or the best **factory production schedule** we can get,
- Or the most accurate **neural network** for a data mining or control problem,
- Or the best **treatment plan** (beam shapes and angles) for radiotherapy cancer treatment
- And so on and so on ….!
- The applications cover all of optimisation and machine learning.

GA's: WHY?

- GA's broad applicability is only one of many strengths.

•Compared to more normal optimization and search processes, GA's:

1. Are far more robust.
2. Make few (if any) assumptions about the problem
3. Require very little information (no characteristic functions, pmf's, cdf's, diff eq's, etc.)
4. In fact need almost no knowledge of math beyond + - * /

GA's: WHY NOT?

•While GA's have many desirable qualities, they aren't without downsides, as they:

1. Can be unnecessarily costly computationally for simple problems.
2. Are occasionally prone to local minima (though less so than many other methods).
3. Cannot solve certain problems (variant problems) with no clear fitness function.
4. As mentioned before, are stochastic. If 100% predictability is required, GA's should not be utilized by themselves.

GA's: HOW?

•These strengths and weaknesses come from four qualities described by Goldberg:

1. GA's work with a coding of the parameter set, not the parameters themselves.
2. GA's search from a population of points, not a single point.
3. GA's use payoff (objective function) information, not derivatives or other auxiliary knowledge.
4. GA's use probabilistic transition rules, not deterministic rules.

GA's: HOW?

•Structure of GA's can vary widely, most follow:

- Create/obtain initial population of answers
- Evaluate fitness of each answer
- Selectively (randomly with bias) pick answers to breed
- Selected answers produce offspring and make up a new population of answers related to them
- These have their fitness evaluated
- …
- Process continues until termination

Acknowledgement

Introduction to Genetic Algorithms

Assaf Zaritsky

Ben-Gurion University, Israel

www.cs.bgu.ac.il/~assafza

*www.macs.hw.ac.uk/~dwcorne/Teaching/gaprimer.**ppt***

Global Optimization
Genetic Algorithms

Olesya Peshko

http://www.cas.mcmaster.ca/~cs777/presentations/3_GO_Olesya_Genetic_Algorithms.pdf