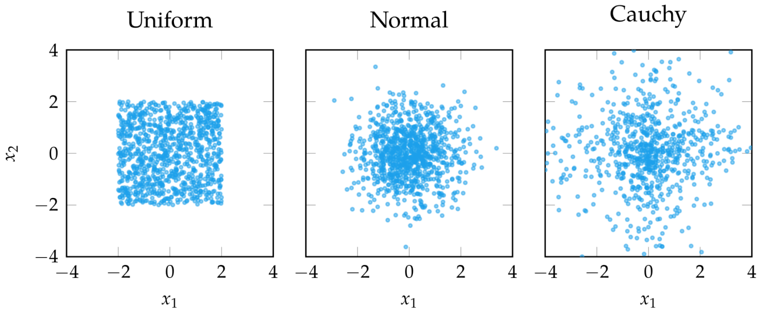
## Outline

## Population Methods

- Instead of optimizing a single design point, population methods optimize a collection of **individuals**
- A large number of individuals prevents algorithm from being stuck in a local minimum
- Useful information can be shared between individuals
- Stochastic in nature
- Easy to parallelize

## Initialization

- Population methods begin with an initial population
- Common initializations are uniform, normal distribution, and Cauchy distribution
- But also space filling designs (later)

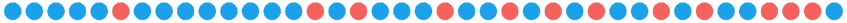


## Genetic Algorithms

- Inspired by biological evolution where the fittest individuals pass their genetic information to the next generation
- Individuals are interpreted as **chromosomes**
- The fittest individuals are determined by **selection**
- The next generation is formed by selecting the fittest individuals and performing **crossover** and **mutation**

## Genetic Algorithms: Chromosomes

- Simplest representation is the binary string chromosome



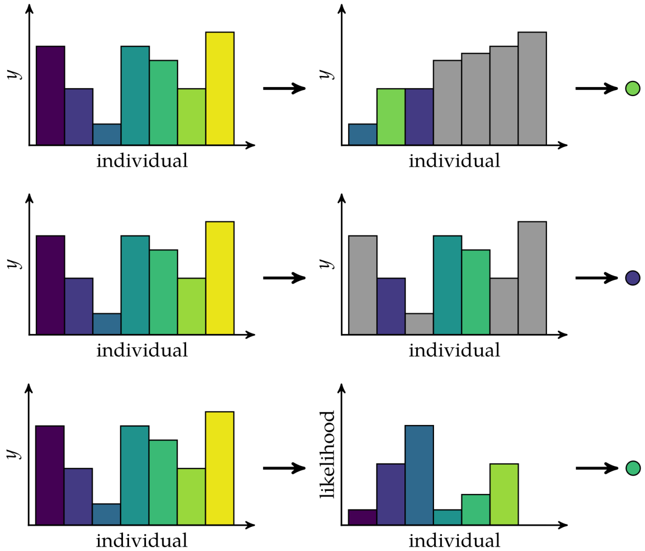
- Chromosomes are more commonly represented as real-valued chromosomes which are simply real-valued vectors
- Typically initialized randomly

## Genetic Algorithms: Selection

- Determining which individuals pass their genetic information on to the next generation choosing chromosomes to use as parents for the next generation
- Truncation selection: truncate the lowest performers
- Tournament selection: selects fittest out of k randomly chosen individuals
- Roulette Wheel selection: individuals are chosen with probability proportional to their fitness

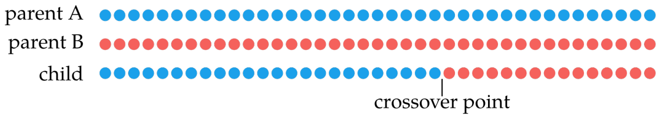
# Genetic Algorithms: Selection

Truncation Selection
Tournament Selection
Roulette Wheel Selection

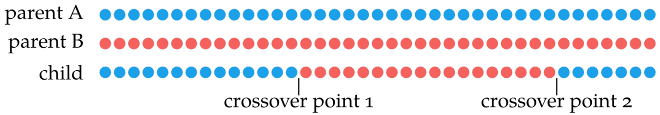


# Genetic Algorithms: Crossover

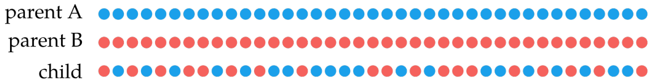
- Combines the chromosomes of the parents to form children
- Single-point crossover: swap occurs after single crossover point



- Two-point crossover: two crossover points



- Uniform crossover: each bit has 50% chance of crossover



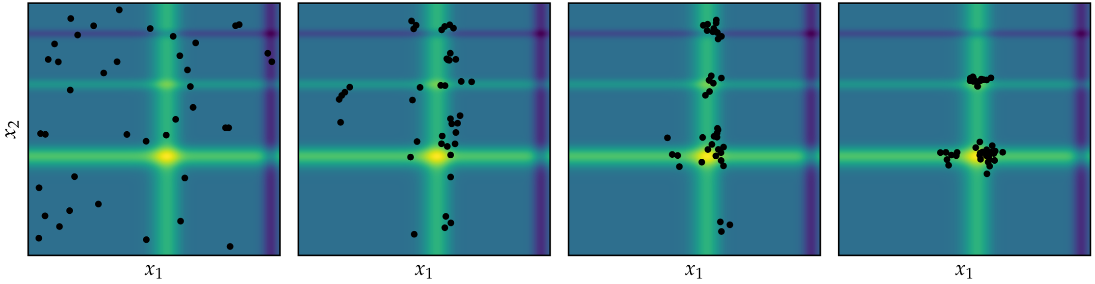
- real values are linearly interpolated between the parents' values $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$: $\boldsymbol{x} \leftarrow (1-\lambda)\boldsymbol{x}_a + \lambda \boldsymbol{x}_b$

## Genetic Algorithms: Mutation

- Mutation supports exploration of new areas of design space
- Each bit or real-valued element has a probability of being flipped or modified by noise
- The probability of an element mutating is called **mutation rate**

# Genetic Algorithms

Genetic algorithm with truncation selection, single point crossover, and Gaussian mutation applied to Michalewicz function



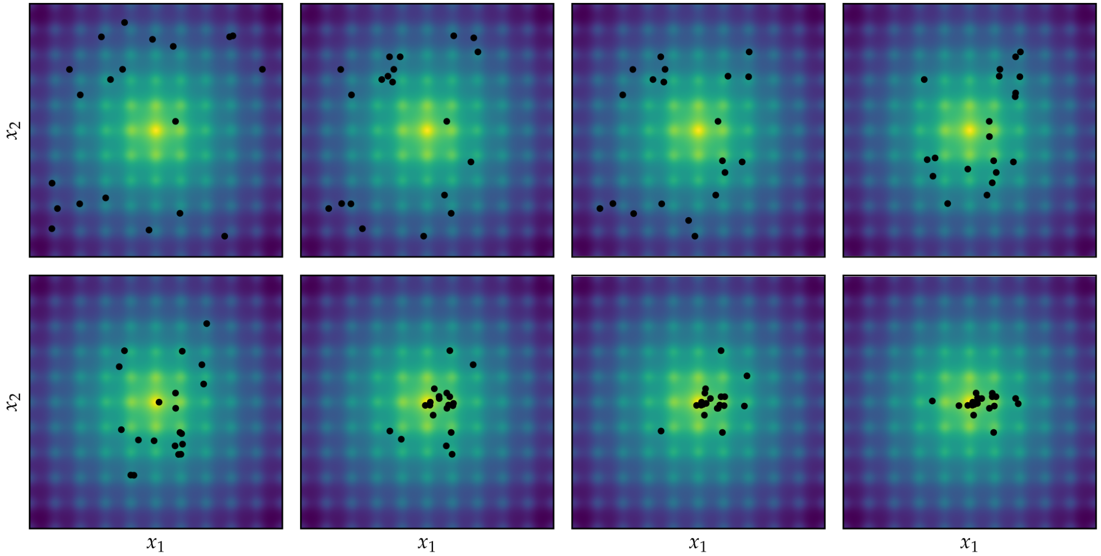
## Differential Evolution

Improves each individual $\boldsymbol{x}$ by recombining other individuals according to a simple formula

1. Choose three random, distinct individuals $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$
2. Construct interim design $\boldsymbol{z} = \boldsymbol{a} + w(\boldsymbol{b} - \boldsymbol{c})$
3. Choose a random dimension to optimize in
4. Construct candidate $\boldsymbol{x}'$ via binary crossover of $\boldsymbol{x}'$ and $\boldsymbol{z}$

$$x_i' = \begin{cases} z_i & \text{if } i = j \text{ or with probability } p \\ x_i & \text{otherwise} \end{cases}$$

5. Insert better design between $\boldsymbol{x}$ and $\boldsymbol{x}'$ into next generation


$x_2$
$x_2$
$x_1$
$x_1$
$x_1$
$x_1$

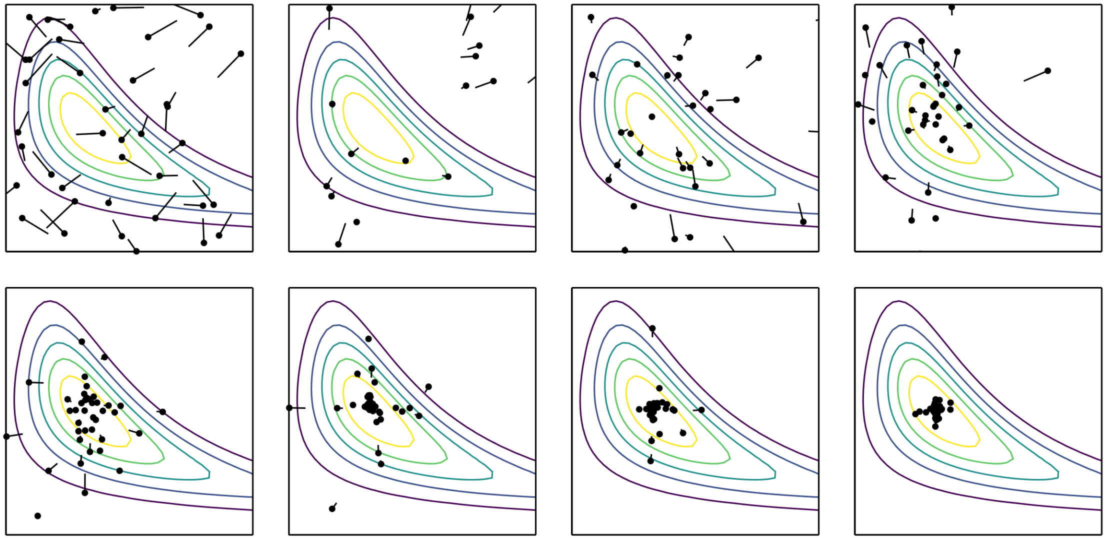
## Particle Swarm Optimization

- Each individual, or particle, tracks the following
    - Current position
    - Current velocity
    - Best position seen so far by the particle
    - Best position seen so far by any particle
- At each iteration, these factors produce force and momentum effects to determine each particle's movement

$$x_i \leftarrow x_i + v_i$$
$$v_i \leftarrow wv_i + c_1 r_1 \left(x_i^{best} - x_i\right) + c_2 r_2 \left(x_i^{best} - x_i\right)$$

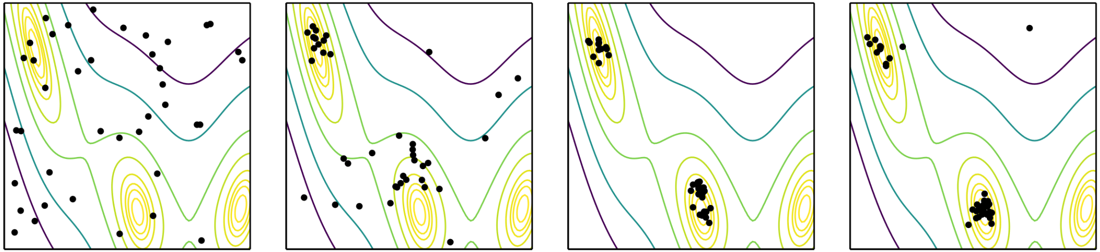
$\boldsymbol{x}_{best}$ is the best location found so far over all particles; $w$, $c_1$, and $c_2$ are parameters; and $r_1$ and $r_2$ are random numbers drawn from $U(0,1)$

# Particle Swarm Optimization



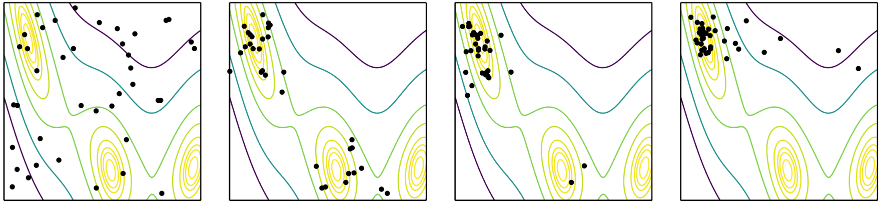
## Firefly Algorithm

- Inspired by the way fireflies flash their lights to attract mates
- Attractiveness is determined by low function value
- At each iteration, fireflies move toward the most attractive lights
- Random noise is added to increase exploration



## Cuckoo Search

- ...



## ...

The Evolutionary Computation Bestiary
http://fcampelo.github.io/EC-Bestiary/

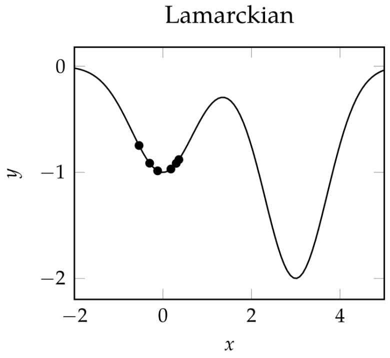
## Hybrid Methods

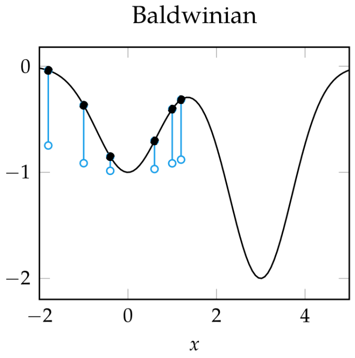
- Generally, population methods are good at finding the best regions in design space, but do not perform as well as descent methods near the minimizer
- Hybrid methods try to leverage the strength of both methods
- Two hybrid approaches
  Lamarckian learning
  Baldwinian learning

## Hybrid Methods

- Lamarckian learning
  Performs regular descent method update on each individual

- Baldwinian learning
  Uses value of descent method update to augment the objective value of each design point





## Summary

- Population methods use a collection of individuals in the design space to guide progression toward an optimum
- Genetic algorithms leverage selection, crossover, and mutations to produce better subsequent generations
- Differential evolution, particle swarm optimization, the firefly algorithm, and cuckoo search include rules and mechanisms for attracting design points to the best individuals in the population while maintaining suitable state space exploration
- Population methods can be extended with local search approaches to improve convergence