

# Understanding algorithm quality through genetic evolution

Henry Wilde and supervisors

## Abstract

When faced with a problem involving data, it is almost certainly the case that the data is fixed and in order to do something with that data, a researcher must select an algorithm that is appropriate for the problem domain whilst performing well on their data. The value prescribed to an algorithm is often found through a process of surveying the current literature to create a shortlist, then running various trials with the shortlisted algorithms. The winning algorithm is then chosen based on some common objective value. The issue with this process is that it does not necessarily allow (or require) the researcher to consider why certain algorithms perform better on particular datasets and not others, and which characteristics make data “good” for their chosen algorithm.

This paper introduces a novel method for generating artificial data through genetic evolution, the purpose of which is to create populations of datasets for which a particular algorithm performs well. This is done by passing an algorithm’s objective function to an evolutionary algorithm. Therein, each individual is a particular dataset defined by its dimensions, entries, and the approximate statistical shape of each of its attributes. In this way, detailed information about each individual is retained throughout the algorithm. Hence, they may be manipulated in a meaningful way during the run, and studied once the algorithm has terminated.

Following this, a number of examples are given to show the performance of the method. These examples are created using a Python implementation of the process which is built to be highly customisable, interpretable and reproducible.

# 1 Introduction

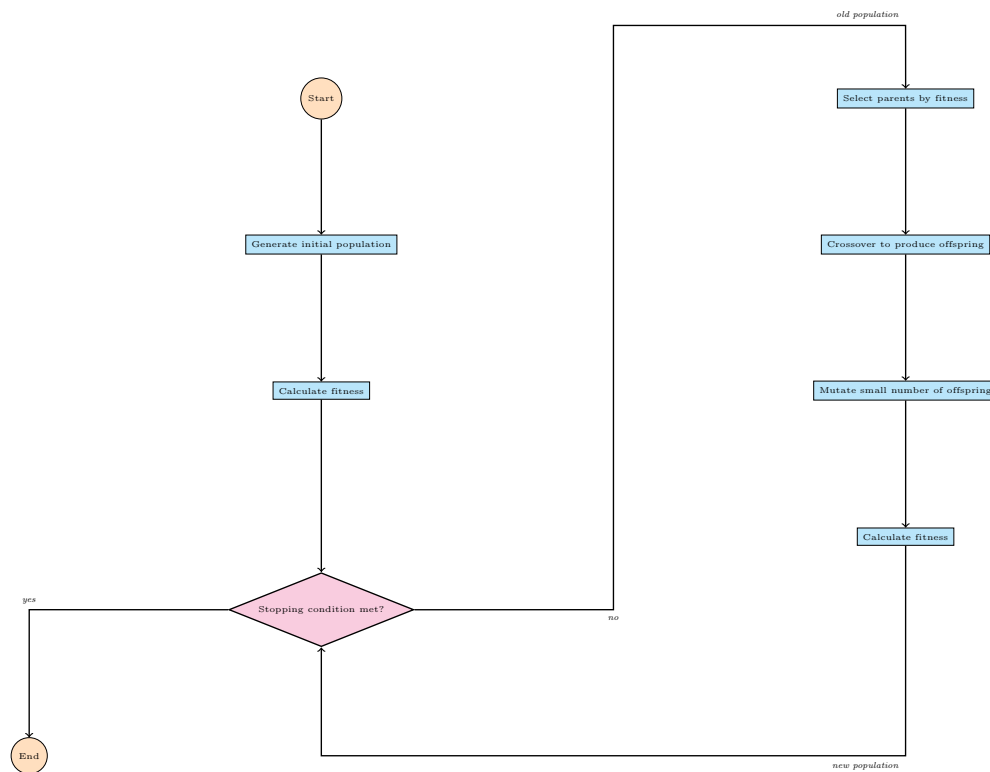


Figure 1: A general schematic for an evolutionary algorithm.

- What is the motivation?
- What is the problem?
- What is the solution?

## 1.1 Literature review

- How is artificial data made?
- Why hasn't this been done before?
- Genetic algorithms used to train algorithms for data
- Diagram

## 2 The evolutionary algorithm

### 2.1 Structure

---

**Algorithm 1:** The evolutionary dataset optimisation algorithm

---

Fitness function,  $f : X \rightarrow \mathbb{R}$ ; population size,  $N$ ; row limits,  $R = [r_l, r_u]$ ; column limits,  $C = [c_l, c_u]$ ;

**Data:** Column distributions,  $P$ ; relative weights for  $P$ ,  $w \in [0, 1]^{|P|}$  s.t.  $\sum w = 1$ ; maximum iterations,  $M$ ;  
Best proportion,  $\delta$ ; lucky proportion,  $\epsilon$ ; mutation probability,  $\mu$ ; compaction ratio,  $s$ .

**Result:** A full history of the population and their fitnesses.

**begin**

- initialisation
- $PopulationHistory \leftarrow \emptyset$
- $Population \leftarrow CreateInitialPopulation(N, R, C, P, w)$
- $Fitness \leftarrow GetPopulationFitness(Population, f)$
- $PopulationHistory \leftarrow (Population, Fitness)$
- $iteration \leftarrow 0$
- while**  $iteration < M$  **and not** **STOP** **do**
  - select parent individuals
  - $Parents \leftarrow Selection(Population, Fitness, \delta, \epsilon)$
  - create new population
  - $NewPopulation \leftarrow Parents$
  - while**  $|NewPopulation| < N$  **do**
    - Sample two parents,  $ParentOne, ParentTwo$ , from  $Parents$
    - $Offspring \leftarrow Crossover(ParentOne, ParentTwo, C, P, )$
    - $Mutant \leftarrow Mutation(Offspring, \mu, R, C, P, w)$
    - $NewPopulation \leftarrow NewPopulation \cup \{Mutant\}$
  - $Population \leftarrow NewPopulation$
  - $Fitness \leftarrow GetPopulationFitness(Population, f)$
  - $PopulationHistory \leftarrow (Population, Fitness)$
  - adjust parameters
  - if** **DWINDLE** **then**
    - $\mu \leftarrow DWINDLE(\mu, m)$
  - $ReduceMutationSpace(Parents, P, m, M, s)$
  - $iteration \leftarrow iteration + 1$

---



---

**Algorithm 2:**  $CreateInitialPopulation$

---

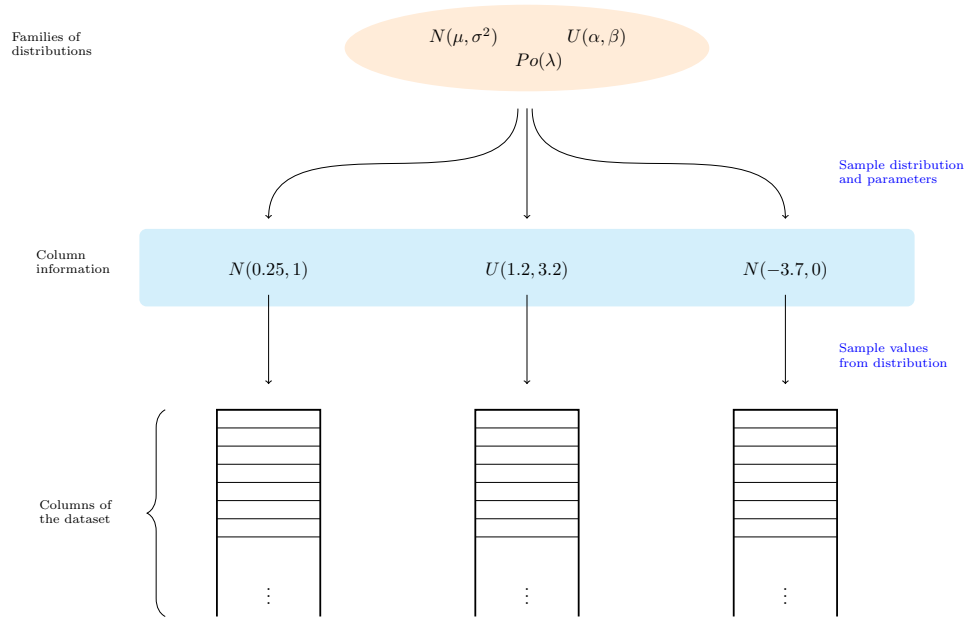


Figure 2: An example of how an individual is first created.

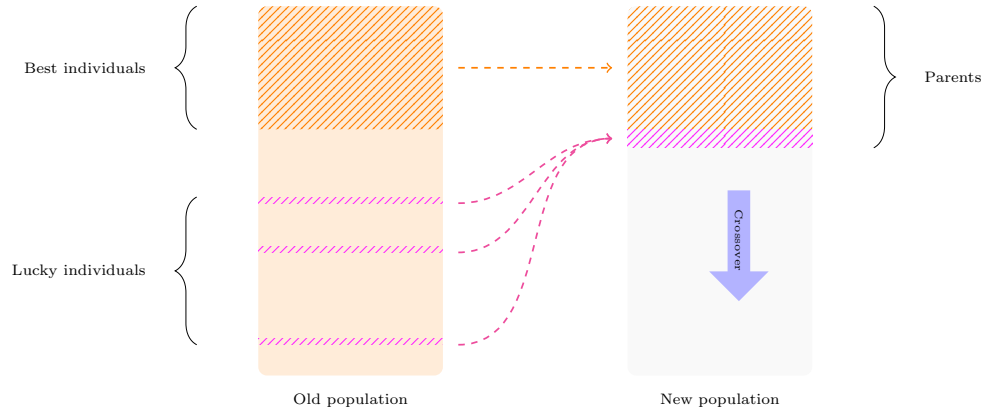


Figure 3: An example of the selection process with the inclusion of some lucky individuals.

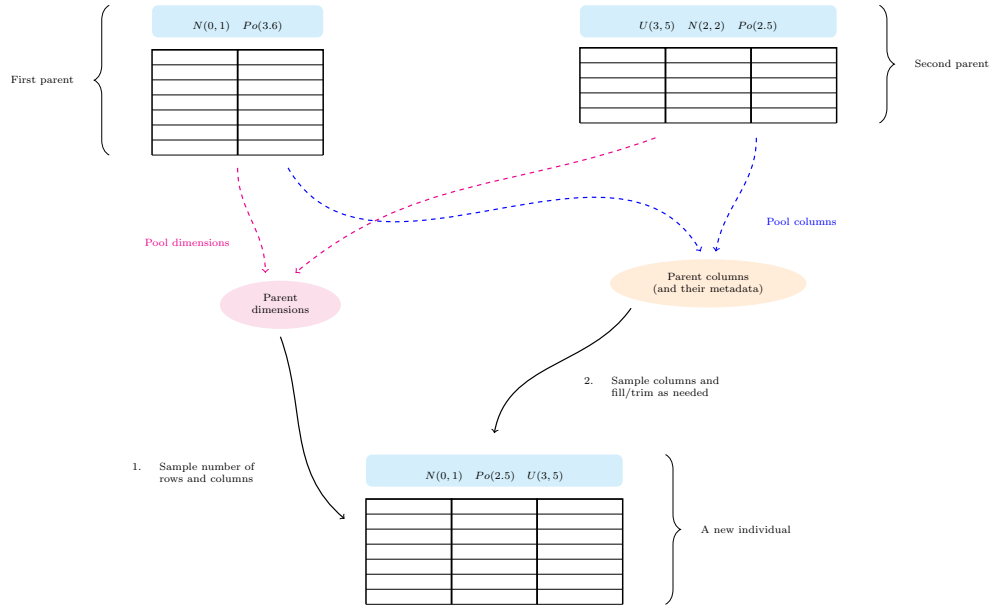


Figure 4: An example of the crossover process.

## 2.2 Internal mechanisms

### 2.2.1 Individuals

### 2.2.2 Selection

### 2.2.3 Crossover

### 2.2.4 Mutation

## 2.3 Implementation

- Documentation: <https://edo.readthedocs.io>
- Repo: <https://github.com/daffidwilde/edo>

## 3 Numerical examples

- The  $x^2$  example from the docs is a nice easy one to illustrate things
- Something stochastic
- Make use of the moving parts (linear focus of the mutation space, dwindling mutation probability, stopping conditions)

- If not the  $k$ -modes initialisation example, then another clustering one. Maybe  $k$ -means versus DBSCAN to show DBSCAN works better on non-convex clusters.