Evolutionary Dataset Optimisation: learning algorithm quality through evolution

Henry Wilde, Vincent Knight, Jonathan Gillard

Abstract

In this paper we propose a new method for learning how algorithms perform. Classically, algorithms are compared on a finite number of existing (or newly simulated) benchmark data sets based on some fixed metric. The algorithm(s) with the smallest value of this metric are chosen to be the 'best performing'.

We offer a new approach to flip this paradigm. We instead aim to gain a richer picture of the performance of an algorithm by generating artificial data through genetic evolution, the purpose of which is to create populations of datasets for which a particular algorithm performs well. These data sets can be studied to learn as to what attributes lead to a particular progress of a given algorithm.

Following a detailed description of the algorithm as well as a brief description of an open source implementation, a number of numeric experiments are presented to show the performance of the method.

1 Introduction

This work presents a novel approach to learning the quality and performance of an algorithm through the use of evolution. When an algorithm is developed to solve a given problem, the designer is presented with questions about the performance of their proposed method, and its relative performance against existing methods. This is an inherently difficult task. However, under the current paradigm, the standard response to this situation is to use a known fixed set of datasets and a common metric amongst the proposed method and its competitors. The algorithm is then assessed based on this metric with minimal consideration for both the appropriateness or reliability of the datasets being used, and the robustness of the method in question.

This notion is not so easily observed when travelling in the opposite direction. Suppose that, instead, the benchmark was a dataset of particular interest and a preferable algorithm was to be determined for some task. There exist a number of methods employed across disciplines to complete this task that take into account the characteristics of the data and the context of the research problem. These methods include the use of diagnostic tests. For instance, in the case of clustering, if the data displayed an indeterminate number of non-convex blobs, then one could recommend that an appropriate clustering algorithm would be DBSCAN [5]. Otherwise, for scalability, k-means may be chosen.

The approach presented in this work aims to flip the paradigm described here by allowing the data itself to be unfixed. This fluidity in the data is achieved by generating data for which the algorithm performs well (or better than some other) through the use of an evolutionary algorithm. The purpose of doing so is not to simply create a bank of useful datasets but rather to allow for the subsequent studying of these datasets. In doing so, the attributes and characteristics which lead to the success (or failure) of the algorithm may be described, giving a broader understanding of the algorithm on the whole.

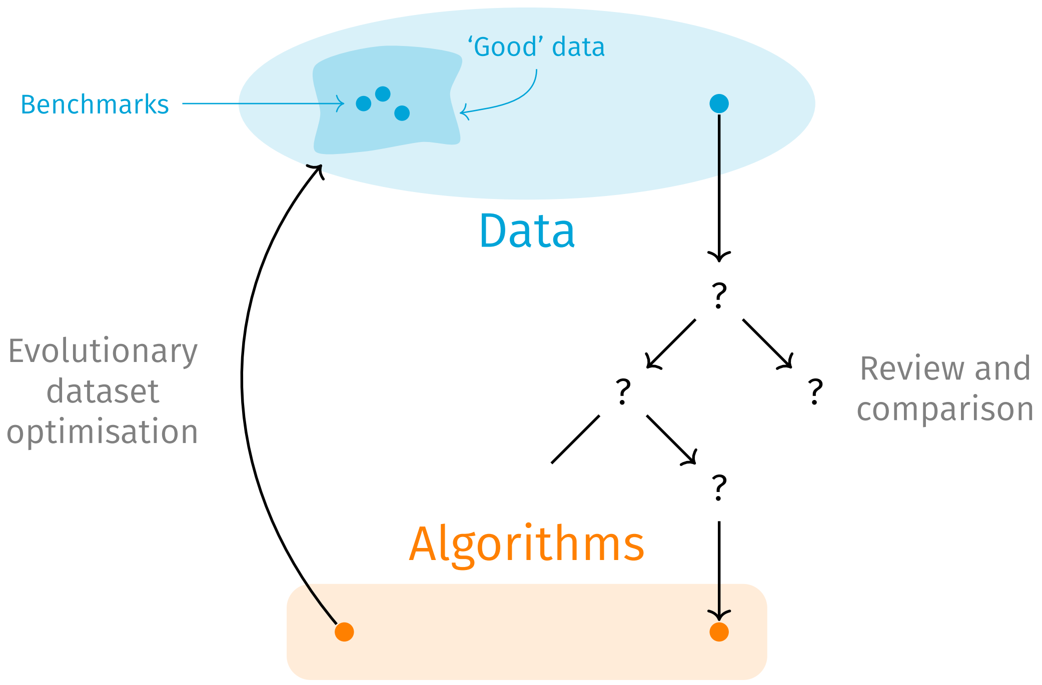


Figure 1: On the right: the current path for selecting some algorithm(s) based on their validity and performance for a given dataset. On the left: the proposed flip to better understand the space in which 'good' datasets exist for an algorithm.

This proposed flip has a number of motivations, and below is a non-exhaustive list of some of the problems that are presented by the established evaluation paradigm:

1. How are these benchmark examples selected? There is no true measure of their reliability other than their frequent use. In some domains and disciplines there are well-established benchmarks so those found through literature may well be reliable, but in others less so.

2. Sometimes, when there is a lack of benchmark examples, a 'new' dataset is simulated to assess the algorithm. This begs the question as to how and why that simulation is created. Not only this, but the origins of existing benchmarks is often a matter of convenience rather than their merit.

3. In disciplines where there are established benchmarks, there may still be underlying problems around the true performance of an algorithm:

(i) As an example, work by Torralba and Efros [22] showed that image classifiers trained and evaluated on a particular dataset, or datasets, did not perform reliably when evaluated using other benchmark datasets that were determined to be similar. Thus leading to a model which lacks robustness.

(ii) The amount of learning one can gain as to the characteristics of data which lead to good (or bad) performance of an algorithm is constrained to the finite set of attributes present in the benchmark data chosen in the first place.

Evolutionary algorithms (EAs) have been applied successfully to solve a wide array of problems - particularly where the complexity of the problem or its domain are significant. These methods are highly adaptive and their population-based construction (displayed in Figure 2) allows for the efficient solving of problems that are otherwise beyond the scope of traditional search and optimisation methods.

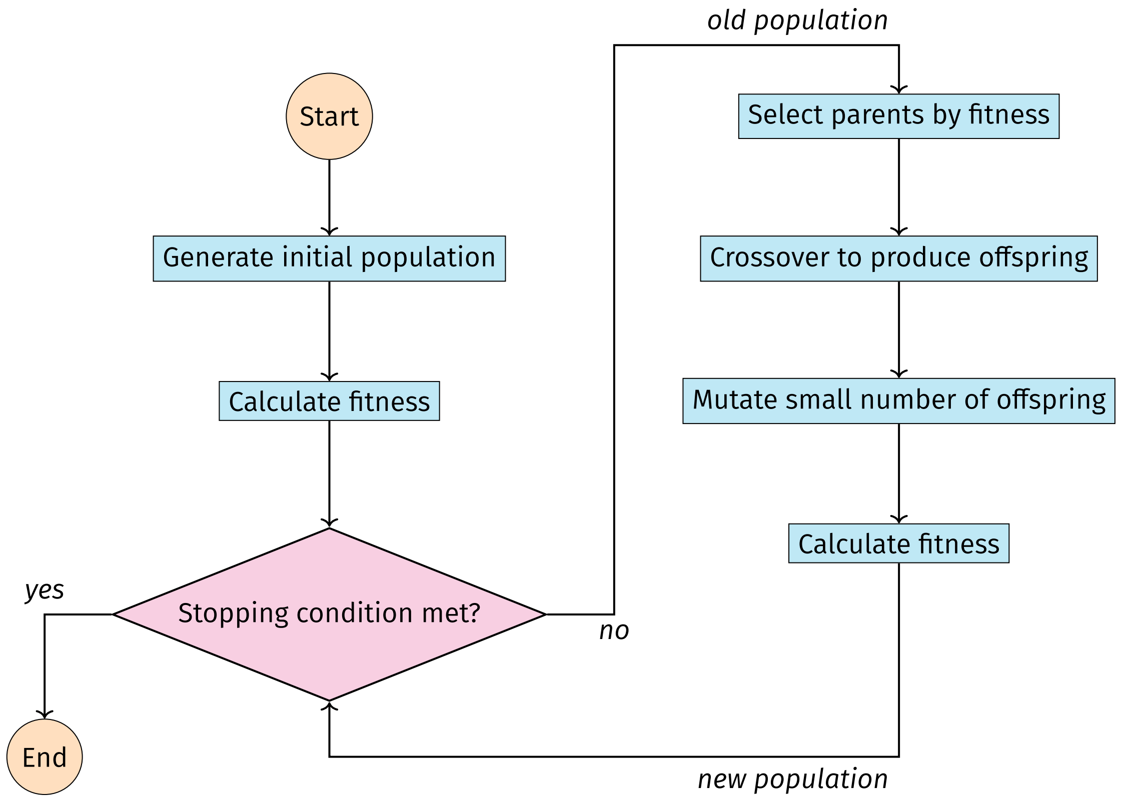

old population
Start
Select parents by fitness
Generate initial population
Crossover to produce offspring
Calculate fitness
Mutate small number of offspring
Calculate fitness
yes
Stopping condition met?
no
End
new population

Figure 2: A general schematic for an evolutionary algorithm.

The use of EAs to generate artificial data is not a new concept. Its applications in data generation have included developing methods for the automated testing of software [9, 15, 19] and the synthesis of existing or confidential data [3]. Such methods also have a long history in the parameter optimisation of algorithms, and recently in the automated design of convolutional neural network (CNN) architecture [20, 21].

Other methods for the generation or synthesis of artificial data include simulated annealing [13] and generative adversarial networks (GANs) [6]. The unconstrained learning style of methods such as CNNs and GANs aligns with that proposed in this work. By allowing the EA to explore and learn about the search space in an organic way, less-prejudiced insight can be established that is not necessarily reliant on any particular framework or agenda.

Note that the proposed methodology is not simply to use an EA to optimise an algorithm over a search space with fixed dimension or datatype such as those set out in [3]. The size and sample space itself is considered as a property that can be traversed through the algorithm.

2 The evolutionary algorithm

2.1 Structure

In this section, the details of an algorithm that generates data for which a given function or, equivalently, an algorithm which is well suited, is described. This algorithm is to be referred to as "Evolutionary Dataset Optimisation" (EDO).

The EDO method is built as an evolutionary algorithm which follows a traditional (generic) schema with some additional features that keep the objective of artificial data generation in mind. With that, there are a number of parameters that are passed to EDO; the typical parameters of an evolutionary algorithm are a fitness function, f, which maps from an individual to a real number, as well as a population size, N, a maximum number of iterations, M, a selection parameter, b, and a mutation probability, p_m. In addition to these, EDO takes the following parameters:

- Limits on the number of rows an individual dataset can have:

$$R \in \left\{(r_{\min}, r_{\max}) \in \mathbb{N}^2 \mid r_{\min} \le r_{\max}\right\}$$

- Limits on the number of columns a dataset can have:

$$C := \left(C_1, \ldots, C_{|\mathcal{P}|}\right) \text{ where } C_j \in \left\{(c_{\min}, c_{\max}) \in (\mathbb{N} \cup \{\infty\})^2 \mid c_{\min} \le c_{\max}\right\}$$

  for each $j = 1, \ldots, |\mathcal{P}|$. That is, C defines the minimum and maximum number of columns a dataset may have from each distribution in $\mathcal{P}$.

- A set of probability distribution families, $\mathcal{P}$. Each family in this set has some parameter limits which form a part of the overall search space. For instance, the family of normal distributions, denoted by $N(\mu, \sigma^2)$, would have limits on values for the mean, μ, and the standard deviation, σ.

- A probability vector to sample distributions from $\mathcal{P}$, $w = \left(w_1, \ldots, w_{|\mathcal{P}|}\right)$.

- A second selection parameter, $l \in [0, 1]$, to allow for a small proportion of "lucky" individuals to be carried forward.

- A shrink factor, $s \in [0, 1]$, defining the relative size of a component of the search space to be retained after adjustment.

The concepts discussed in this section form the mechanisms of the evolutionary dataset optimisation algorithm. To use the algorithm practically, these components have been implemented in Python as a library built on the scientific Python stack [14, 17]. The library is fully tested and documented (at `https://edo.readthedocs.io`) and is freely available online under the MIT license [4]. The EDO implementation was developed to be consistent with the best practices of open source software development.

The statement of the EDO algorithm is presented here to lay out its general structure from a high level perspective. Lower level discussion is provided below where additional algorithms

Algorithm 1: The evolutionary dataset optimisation algorithm

```
Input: $f, N, R, C, \mathcal{P}, w, M, b, l, p_m, s$
Output: A full history of the populations and their fitnesses.
begin
    create initial population of individuals
    find fitness of each individual
    record population and its fitness
    while current iteration less than the maximum and stopping condition not met do
        select parents based on fitness and selection proportions
        use parents to create new population through crossover and mutation
        find fitness of each individual
        update population and fitness histories
        if adjusting the mutation probability then
            update mutation probability
        end
        if using a shrink factor then
            shrink the mutation space based on parents
        end
    end
end
```

Algorithm 2: Creating a new population

```
Input: parents, $N, R, C, \mathcal{P}, w, p_m$
Output: A new population of size $N$
begin
    add parents to the new population
    while the size of the new population is less than $N$ do
        sample two parents at random
        create an offspring by crossing over the two parents
        mutate the offspring according to the mutation probability
        add the mutated offspring to the population
    end
end
```

for the individual creation, evolutionary operator and shrinkage processes are given along with diagrams (where appropriate).

Note that there are no defined processes for how to stop the algorithm or adjust the mutation probability, p_m. This is down to their relevance to a particular use case. Some examples include:

- Stopping when no improvement in the best fitness is found within some K consecutive iterations [11].
- Utilising global behaviours in fitness to indicate a stopping point [12].
- Regular decreasing in mutation probability across the available attributes [10].

2.2 Individuals

Evolutionary algorithms operate in an iterative process on populations of individuals that each represent a solution to the problem in question. In a genetic algorithm, an individual is a solution encoded as a bit string of, typically, fixed length and treated as a chromosome-like object to be manipulated. In EDO, as the objective is to generate datasets and explore the space in which datasets exist, there is no encoding. As such the distinction is made that EDO is an evolutionary algorithm.

As is seen in Figure 3, an individual's creation is defined by the generation of its columns. A set of instructions on how to sample new values (in mutation, for instance, Section 2.5) for that column are recorded in the form of a probability distribution. These distributions are sampled and created from the families passed in $\mathcal{P}$. In EDO, the produced datasets and their metadata are manipulated directly so that the biological operators can be designed and be interpreted in a more meaningful way as will be seen later in this section.

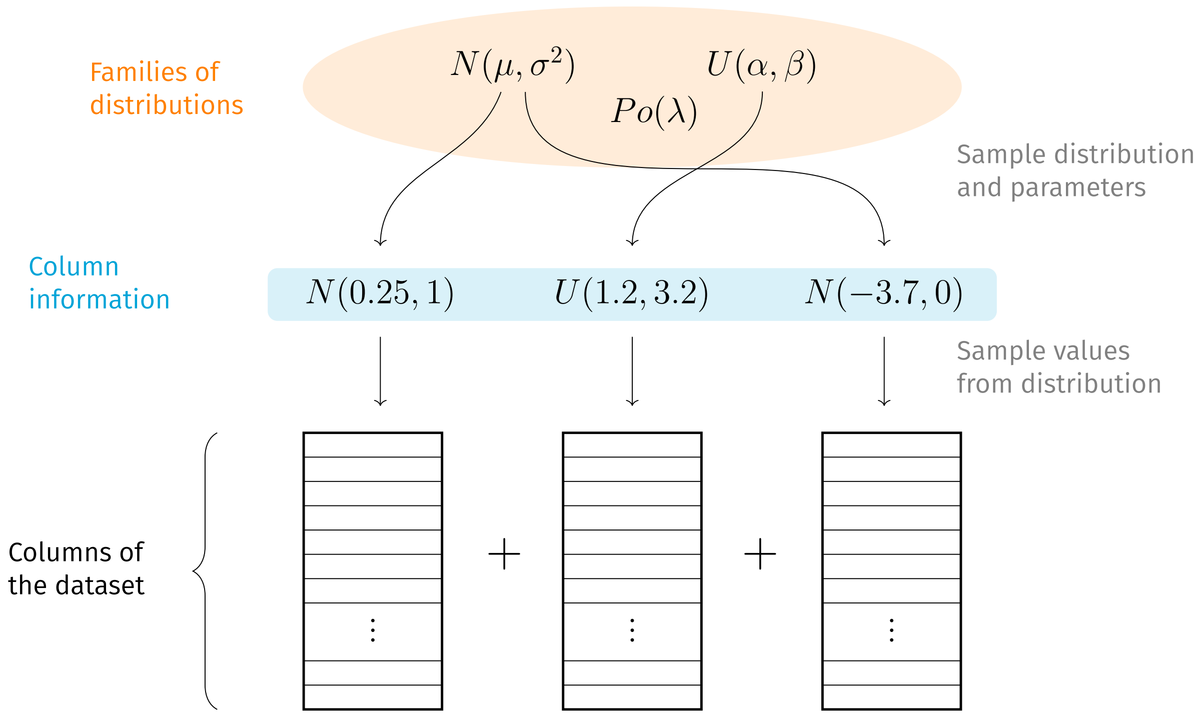


Figure 3: An example of how an individual is first created.

Algorithm 3: Creating an individual

```
Input: $R, C, \mathcal{P}, w$
Output: An individual defined by a dataset and some metadata
begin
    sample a number of rows and columns
    create an empty dataset
    for each column in the dataset do
        sample a distribution from $\mathcal{P}$
        create an instance of the distribution
        fill in the column by sampling from this instance
        record the instance in the metadata
    end
end
```

2.3 Selection

The selection operator describes the process by which individuals are chosen from the current population to generate the next. Almost always, the likelihood of an individual being selected is determined by their fitness. This is because the purpose of selection is to preserve favourable qualities and encourage some homogeneity within future generations [2].

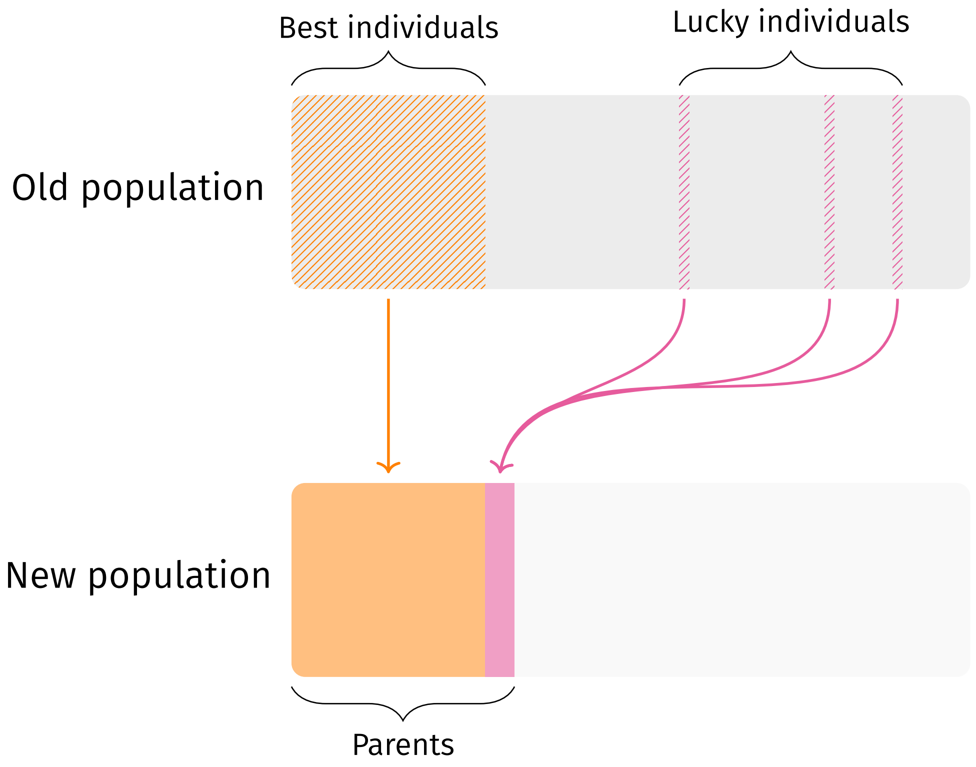


Figure 4: The selection process with the inclusion of some lucky individuals.

In EDO, a modified truncation selection method is used [8], as can be seen in Figure 4. Truncation selection takes a fixed number, $n_b = \lceil bN \rceil$, of the fittest individuals in a population and makes them the "parents" of the next. It has been observed that, despite its efficiency as a selection operator, truncation selection can lead to premature convergence at local optima [8, 16]. The modification for EDO is an optional stage after the best individuals have been chosen: with some small l, a number, $n_l = \lceil lN \rceil$, of the remaining individuals can be selected at random to be carried forward. Hence, allowing for a small number of randomly selected individuals may

Algorithm 4: The selection process

```
Input: population, population fitness, $b$, $l$
Output: A set of parent individuals
begin
    calculate $n_b$ and $n_l$
    sort the population by the fitness of its individuals
    take the first $n_b$ individuals and make them parents
    if there are any individuals left then
        take the next $n_l$ individuals and make them parents
    end
end
```

encourage diversity and further exploration throughout the run of the algorithm. It should be noted that regardless of this step, an individual could potentially be present throughout the entirety of the algorithm.

A final component to the selection process is an adjustment to the search space made up of the distribution parameter limits in $\mathcal{P}$. This adjustment gives the ability to "shrink" the search space about the region observed in a given population. This method is based on a power law described in [1] that relies on a shrink factor, s. At each iteration, t, every distribution which is present in the parents has its parameter's limits, (l_t, u_t), adjusted. This adjustment is such that the new limits, (l_{t+1}, u_{t+1}) are centred about the mean observed value, μ, for that parameter:

$$l_{t+1} = \max\left\{l_t,\ \mu - \frac{1}{2}(u_t - l_t)s^t\right\} \tag{1}$$

$$u_{t+1} = \max\left\{u_t,\ \mu + \frac{1}{2}(u_t - l_t)s^t\right\} \tag{2}$$

The shrinking process is given explicitly in Algorithm 5. Note that the behaviour of this process can produce reductive results for some use cases.

2.4 Crossover

Crossover is the operation of combining two individuals in order to create at least one offspring. In genetic algorithms, the term "crossover" can be taken literally: two bit strings are crossed at

Algorithm 5: Shrinking the mutation space

```
Input: parents, current iteration, $\mathcal{P}, M, s$
Output: A new mutation space focussed around the parents
begin
    for each distribution in $\mathcal{P}$ do
        for each parameter of the distribution do
            get the current values for parameter over all parent columns
            find the mean of the current values
            find the new lower (1) and upper (2) bounds around the mean
            set the parameter limits
        end
    end
end
```

a point to create two new bit strings. Another popular method is uniform crossover, which has been favoured for its efficiency and efficacy in combining individuals [18]. For EDO, this method is adapted to support dataset manipulation: a new individual is created by uniformly sampling each of its components (dimensions and then columns) from a set of two "parent" individuals, as shown in Figure 5.

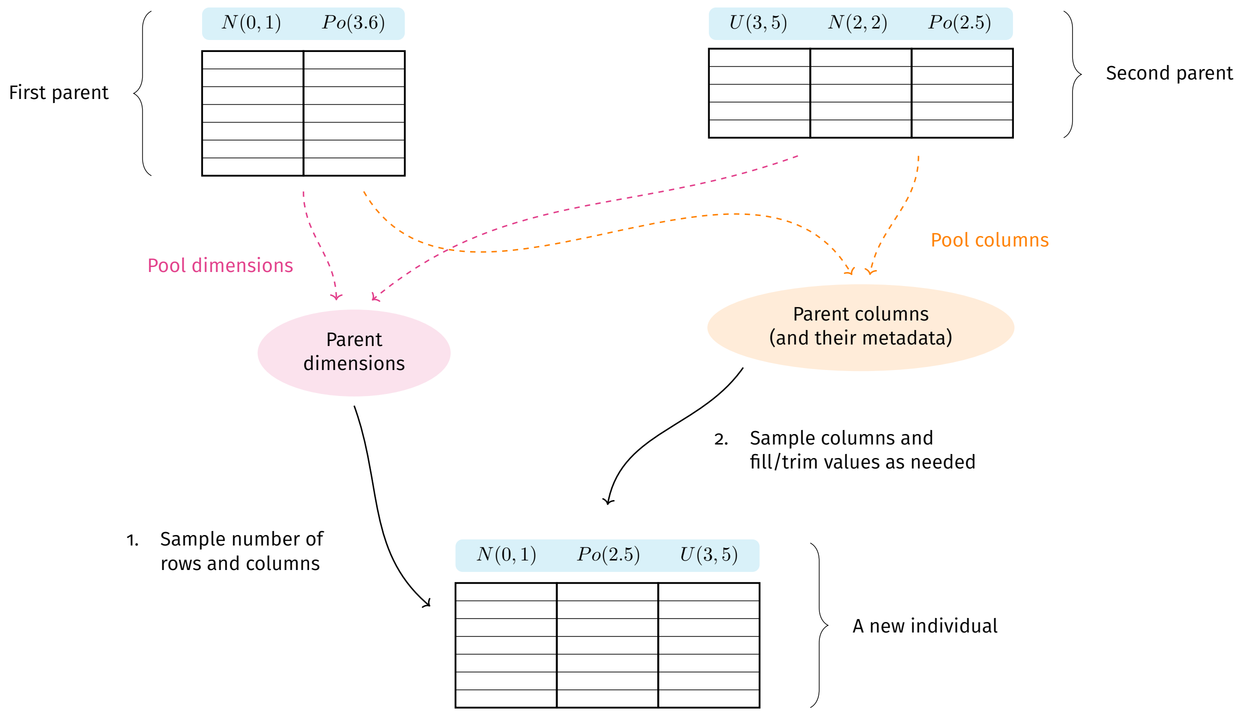

$N(0,1)$ $Po(3.6)$
First parent
$U(3,5)$ $N(2,2)$ $Po(2.5)$
Second parent
Pool dimensions
Pool columns
Parent dimensions
Parent columns (and their metadata)
2. Sample columns and fill/trim values as needed
1. Sample number of rows and columns
$N(0,1)$ $Po(2.5)$ $U(3,5)$
A new individual

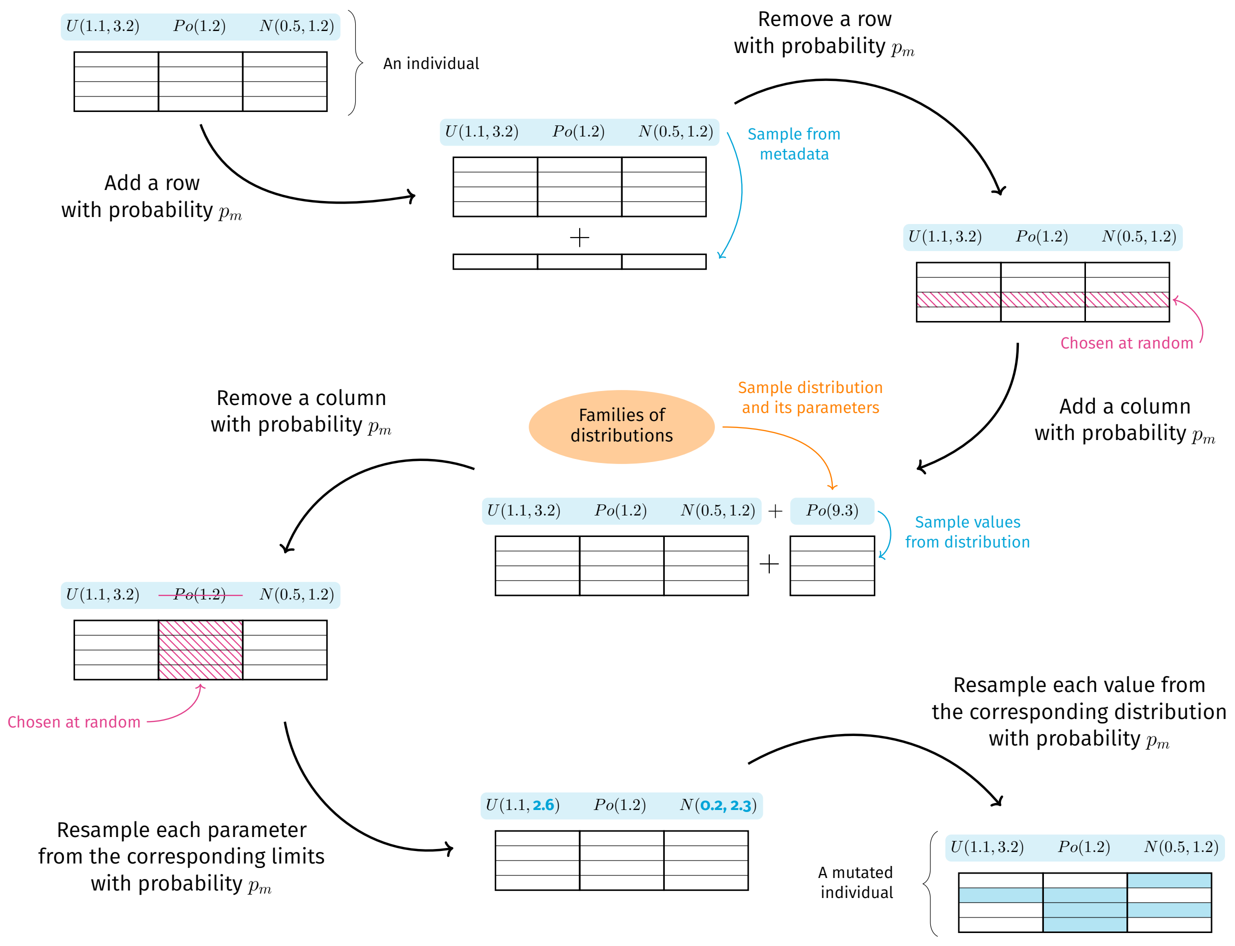
Figure 5: The crossover process between two individuals with different dimensions.

Algorithm 6: The crossover process

```
Input: Two parents
Output: An offspring made from the parents ready for mutation
begin
    collate the columns and metadata from each parent in a pool
    sample each dimension from between the parents uniformly
    form an empty dataset with these dimensions
    for each column in the dataset do
        sample a column (and its corresponding metadata) from the pool
        if this column is longer than required then
            randomly select entries and delete them as needed
        end
        if this column is shorter than required then
            sample new values from the metadata and append them to the column as
              needed
        end
        add this column to the dataset and record its metadata
    end
end
```

2.5 Mutation

Mutation is used in evolutionary algorithms to encourage a broader exploration of the search space at each generation. Under this framework, the mutation process manipulates the phenotype of an individual where numerous things need to be modified including an individual's dimensions, column metadata and the entries themselves. This process is described in Figure 6.



$U(1.1, 3.2)$ $Po(1.2)$ $N(0.5, 1.2)$
An individual
Add a row with probability p_m
Sample from metadata
Remove a row with probability p_m
Chosen at random
Add a column with probability p_m
Families of distributions
Sample distribution and its parameters
$Po(9.3)$
Sample values from distribution
Remove a column with probability p_m
~~$Po(1.2)$~~
Chosen at random
Resample each parameter from the corresponding limits with probability p_m
$U(1.1, 2.6)$ $Po(1.2)$ $N(0.2, 2.3)$
Resample each value from the corresponding distribution with probability p_m
A mutated individual

Figure 6: The mutation process.

As shown in Figure 6, each of the potential mutations occur with the same probability, p_m. However, the way in which columns are maintained assure that (assuming appropriate choices for f and $\mathcal{P}$) many mutations in the metadata and the dataset itself will only result in some incremental change in the individual's fitness relative to, say, a completely new individual.

Algorithm 7: The mutation process

```
Input: An individual, $p_m$, $R$, $C$, $\mathcal{P}$, $w$
Output: A mutated individual
begin
    sample a random number $r \in [0, 1]$
    if $r < p_m$ and adding a row would not violate $R$ then
        sample a value from each distribution in the metadata
        append these values as a row to the end of the dataset
    end
    sample a new $r \in [0, 1]$
    if $r < p_m$ and removing a row would not violate $R$ then
        remove a row at random from the dataset
    end
    sample a new $r \in [0, 1]$
    if $r < p_m$ and adding a new column would not violate $C$ then
        create a new column using $\mathcal{P}$ and $w$
        append this column to the end of the dataset
    end
    sample a new $r \in [0, 1]$
    if $r < p_m$ and removing a column would not violate $C$ then
        remove a column (and its associated metadata) at random from the dataset
    end
    for each distribution in the metadata do
        for each parameter of the distribution do
            sample a random number $r \in [0, 1]$
            if $r < p_m$ then
                sample a new value from within the distribution parameter limits
                update the parameter value with this new value
            end
        end
    end
    for each entry in the dataset do
        sample a random number $r \in [0, 1]$
        if $r < p_m$ then
            sample a new value from the associated column distribution
            update the entry with this new value
        end
    end
end
```

3 Examples

The following examples act as a form of validation for EDO, and also highlight some of the nuances in its use. The examples will be focused around the clustering of data and, in particular, the k-means (Lloyd's) algorithm. Clustering was chosen as it is a well-understood problem that is easily accessible - especially when restricted to two dimensions. The k-means algorithm is an iterative, centroid-based method that aims to minimise the "inertia" of the current partition, $Z = \{Z_1, \ldots, Z_k\}$, of some dataset X:

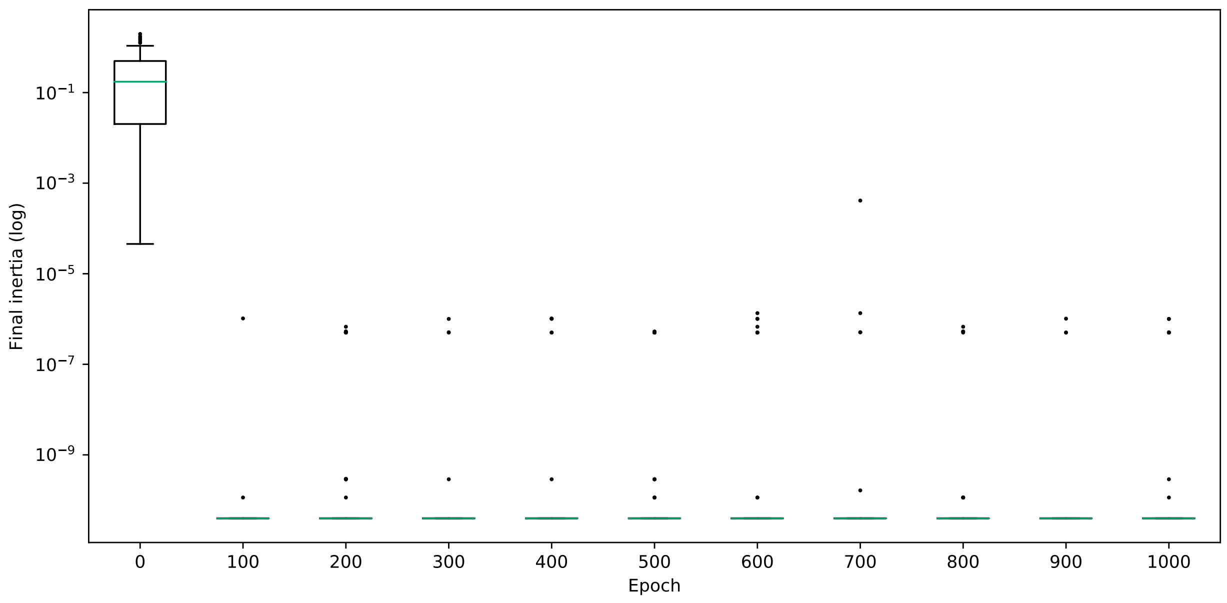
$$I(Z, X) := \frac{1}{|X|} \sum_{j=1}^{k} \sum_{x \in Z_j} d(x, z_j)^2$$

A full statement of the algorithm is given in A.1. However, it should be clear that I may take any non-negative value.

This inertia function is often taken as the objective of the k-means algorithm, and is used for evaluating the final clustering. This is particularly true when the algorithm is not being considered an unsupervised classifier where accuracy may be used [7]. With that, the first example is to use this inertia as the fitness function in EDO. That is, to find datasets which minimise I.

In this example, EDO is restricted to only two-dimensional datasets, i.e. $C = ((2, 2))$. In addition to this, all columns are formed from the uniform distribution restricted to the unit interval, $\mathcal{U} := \{U(a, b) \mid a, b \in [0, 1]\}$. The remaining parameters are as follows: $N = 100$, $R = (3, 100)$, $M = 1000$, $b = 0.2$, $l = 0$, $p_m = 0.01$, and shrinkage excluded. Figure 7 shows an example of the fitness (left) and dimension (right) progression of the evolutionary algorithm under these conditions.

It is clear that there is a steep learning curve here; within the first 100 generations an individual is found with a fitness of roughly 10^{-10} which cannot be improved on for the remaining 900 epochs. In fact, this individual was found in the 44^{th} epoch as can be seen in Figure 8. The same quick convergence



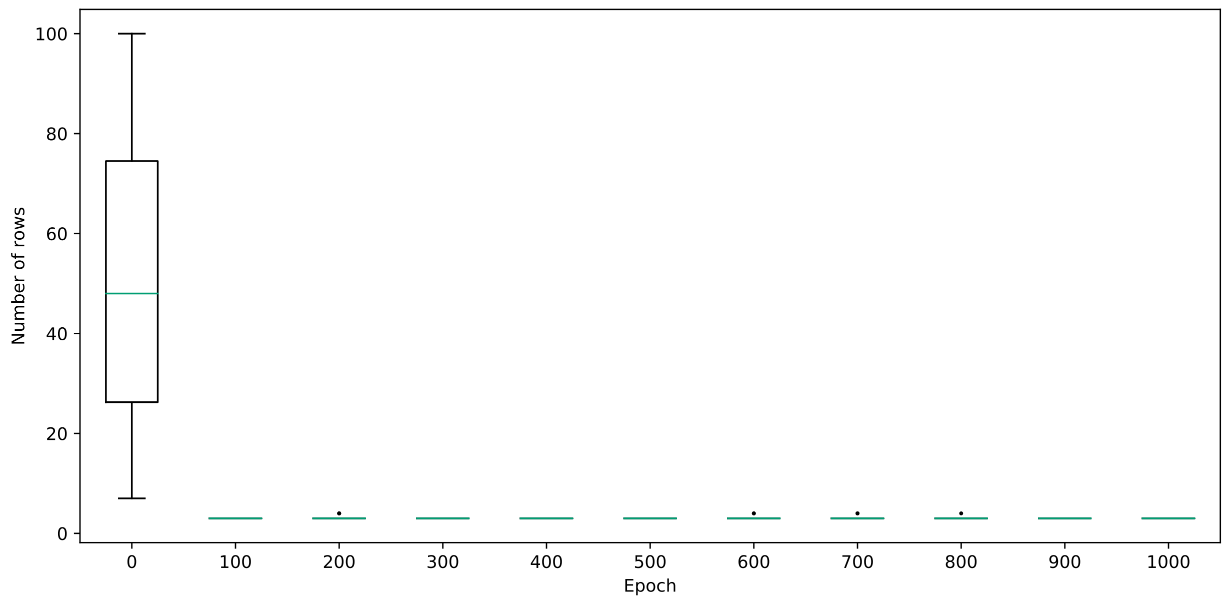
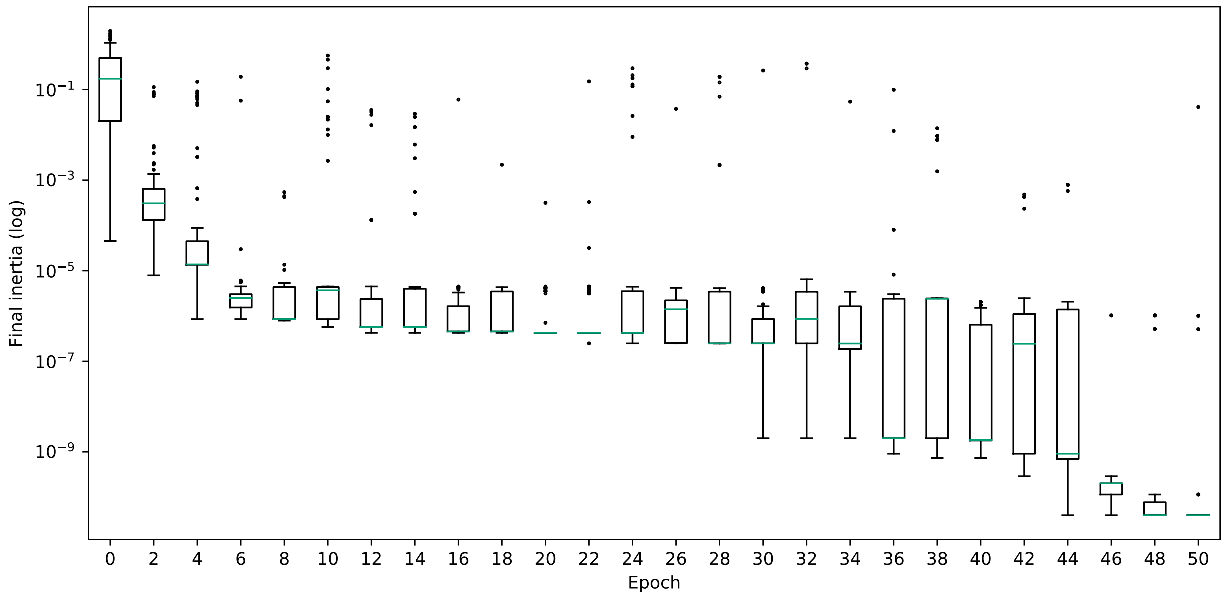


Figure 7: Progressions for fitness and dimension (number of rows) for a run of EDO shown at 100 epoch intervals.



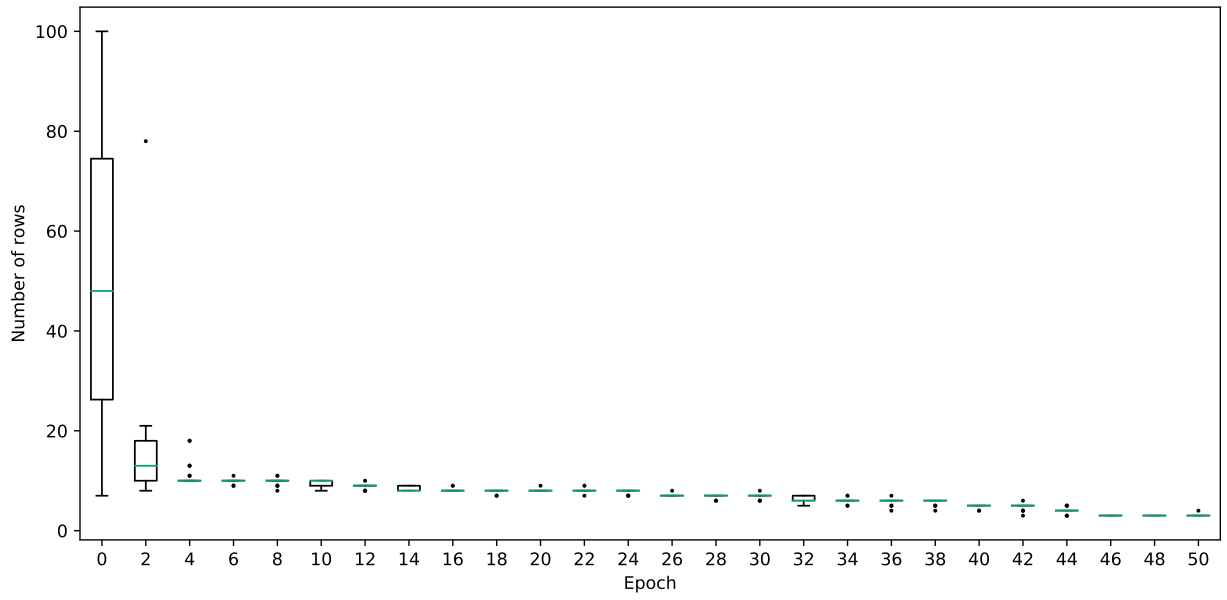


Figure 8: Progressions for fitness and dimension across the first 50 epochs.

4 Conclusion

Rough notes still

A caveat to this: one should not assume that the columns are a reliable representative of the distribution associated with them, or vice versa. This is particularly true of “shorter” datasets with a small number of rows, whereas confidence in the pair could be given more liberally for “longer” datasets with a larger number of rows. In any case, appropriate methods should be employed to understand the structure and characteristics of the data produced before formal conclusions are made.

A Appendix

A.1 Lloyd's algorithm

Algorithm 8: k-means (Lloyd's)

```
Input: a dataset $X$, a number of centroids $k$, a distance metric $d$
Output: a partition of $X$ into $k$ parts, $Z$
begin
    select $k$ initial centroids, $z_1, \dots, z_k \in X$
    while any point changes cluster or some stopping criterion is not met do
        assign each point, $x \in X$, to cluster $Z_{j^*}$ where:
            $$j^* = \operatorname*{arg\,min}_{j=1,\dots,k} \left\{ d\left(x, z_j\right)^2 \right\}$$
        recalculate all centroids by taking the intra-cluster mean:
            $$z_j = \frac{1}{|Z_j|} \sum_{x \in Z_j} x$$
    end
end
```

A.2 Implementation example

Below is an example of how the Python implementation was used to complete the first example, including the definition of the fitness function.

```python
import edo
from edo.pdfs import Uniform
from sklearn.cluster import KMeans


def fitness(dataframe, seed):
    """ Return the final inertia of 2-means on dataframe. """

    km = KMeans(n_clusters=2, random_state=seed).fit(dataframe)
    return km.inertia_
```

```
Uniform.param_limits["bounds"] = [0, 1]
row_limits, col_limits = [3, 100], [2, 2]

edo.run_algorithm(
    fitness,
    size,
    row_limits,
    col_limits,
    families=[Uniform],
    max_iter=1000,
    best_prop=selection,
    mutation_prob=mutation,
    seed=seed,
    root="out",
    fitness_kwargs={"seed": seed},
)
```

References

[1] Adil Amirjanov. “Modeling the Dynamics of a Changing Range Genetic Algorithm”. In: *Procedia Computer Science* 102 (2016), pp. 570 –577. DOI: https://doi.org/10.1016/j.procs.2016.09.444.

[2] Thomas Bäck. “Selective pressure in evolutionary algorithms: a characterization of selection mechanisms”. In: *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*. 1994, pp. 57–62. DOI: 10.1109/ICEC.1994.350042.

[3] Yingrui Chen, Mark Elliot, and Joseph Sakshaug. “A Genetic Algorithm Approach to Synthetic Data Production”. In: *PrAISe@ECAI*. 2016.

[4] The EDO library developers. *EDO: v0.1*. 2018. DOI: 10.5281/zenodo.2557597. URL: http://dx.doi.org/10.5281/zenodo.2557597.

[5] Martin Ester et al. “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. In: *KDD*. 1996.

[6] Ian Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., 2014, pp. 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

[7] Zhexue Huang. “Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values”. In: *Data Mining and Knowledge Discovery* 2.3 (1998), pp. 283–304. DOI: 10.1023/A:1009769707641.

[8] Khalid Jebari. “Selection Methods for Genetic Algorithms”. In: *International Journal of Emerging Sciences* 3 (Dec. 2013), pp. 333–344.

[9] Chahine Koleejan, Bing Xue, and Mengjie Zhang. "Code coverage optimisation in genetic algorithms and particle swarm optimisation for automatic software test data generation". In: *2015 IEEE Congress on Evolutionary Computation (CEC)* (2015), pp. 1204–1211.

[10] Matthias Kuehn, Thomas Severin, and Horst Salzwedel. "Variable Mutation Rate at Genetic Algorithms: Introduction of Chromosome Fitness in Connection with Multi-Chromosome Representation". In: *International Journal of Computer Applications* 72 (July 2013), pp. 31–38. DOI: 10.5120/12636-9343.

[11] Yiu-Wing Leung and Yuping Wang. "An orthogonal genetic algorithm with quantization for global numerical optimization". In: *IEEE Transactions on Evolutionary Computation* 5.1 (2001), pp. 41–53. DOI: 10.1109/4235.910464.

[12] Luis Mart et al. "A stopping criterion for multi-objective optimization evolutionary algorithms". In: *Information Sciences* 367-368 (2016), pp. 700 –718. DOI: https://doi.org/10.1016/j.ins.2016.07.025.

[13] Justin Matejka and George Fitzmaurice. "Same Stats, Different Graphs: Generating Datasets with Varied Appearance and Identical Statistics Through Simulated Annealing". In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI '17. ACM, 2017, pp. 1290–1294. DOI: 10.1145/3025453.3025912.

[14] Wes McKinney. *Data Structures for Statistical Computing in Python*. Proceedings of the 9th Python in Science Conference. [Online; accessed 2019-03-01]. 2010–. URL: https://pandas.pydata.org/.

[15] Christoph C. Michael, Gary McGraw, and Michael Schatz. "Generating Software Test Data by Evolution". In: *IEEE Trans. Software Eng.* 27 (2001), pp. 1085–1110.

[16] Tatsuya Motoki. "Calculating the Expected Loss of Diversity of Selection Schemes". In: *Evolutionary Computation* 10.4 (2002), pp. 397–422. DOI: 10.1162/106365602760972776.

[17] Travis Oliphant. *NumPy: A guide to NumPy*. USA: Trelgol Publishing. [Online; accessed 2019-03-01]. 2006–. URL: http://www.numpy.org/.

[18] Eugene Semenkin and Maria Semenkina. "Self-configuring Genetic Algorithm with Modified Uniform Crossover Operator". In: *Advances in Swarm Intelligence*. 2012, pp. 414–421. ISBN: 978-3-642-30976-2.

[19] Hossein Sharifipour, Mojtaba Shakeri, and Hassan Haghighi. "Structural test data generation using a memetic ant colony optimization based on evolution strategies". In: *Swarm and Evolutionary Computation* 40 (2018), pp. 76 –91. DOI: https://doi.org/10.1016/j.swevo.2017.12.009.

[20] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. "A genetic programming approach to designing convolutional neural network architectures". In: *GECCO*. 2017.

[21] Yanan Sun et al. "Automatically Designing CNN Architectures Using Genetic Algorithm for Image Classification". In: *CoRR* abs/1808.03818 (2018).

[22] A. Torralba and A. A. Efros. *Unbiased Look at Dataset Bias*. Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition. 2011. DOI: 10.1109/CVPR.2011.5995347. URL: http://dx.doi.org/10.1109/CVPR.2011.5995347.