Evolutionary Dataset Optimisation: learning algorithm quality through genetic evolution

Henry Wilde, Vincent Knight, Jonathan Gillard

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

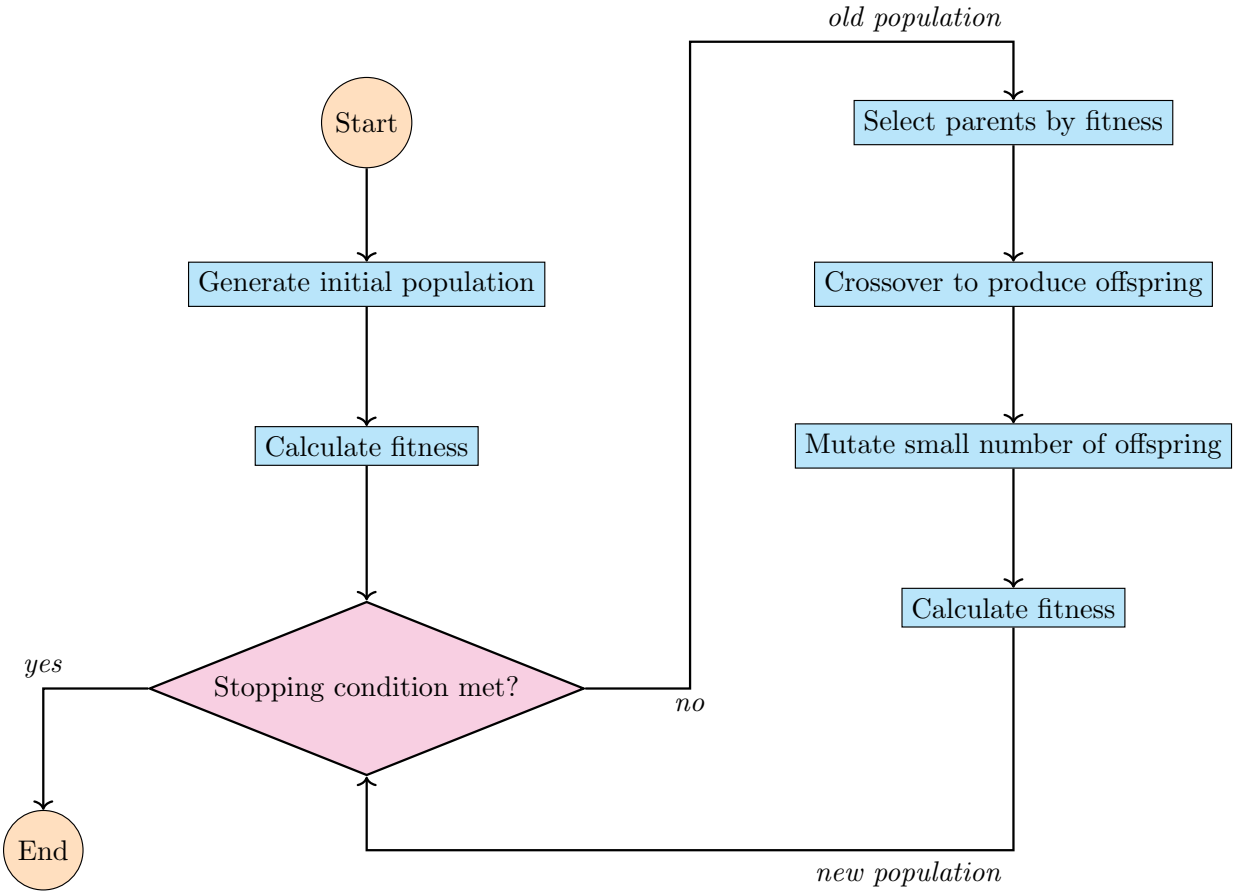

Figure 1: A general schematic for an evolutionary algorithm.

- 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 showing that this is the "reverse" problem

2 The evolutionary algorithm

2.1 Structure

On the surface, the evolutionary algorithm in EDO follows a fairly generic schema; there is a random initial population, individuals are selected through a modified truncation process, and a uniform crossover operator is used.

However, there are some additional features and the processes within are designed with the objective of artificial data generation in mind. With that, there are a number of parameters passed to the algorithm, including several that are not typical:

- A real-valued fitness function, $f : X \to \mathbb{R}$, which acts on a dataset to return a single fitness score.
- A population size, $N \in \mathbb{N}$.
- Limits on the number of rows a dataset can have, $R \in \big\{(l, u) \in \mathbb{N}^2 \mid l \leq u\big\}$.
- Limits on the number of columns a dataset can have, $C \in \big\{(l, u) \in \mathbb{N}^2 \mid l \leq u\big\}$. Note that it is possible to use a different form for C so as to specify a minimum or maximum number of columns a dataset should have for each distribution passed in $\mathcal{P}$.
- A pool of probability distribution families, $\mathcal{P}$. Each family in this pool has a set of parameter limits which form a part of the overall search space. For instance, the normal distribution family, denoted by $N(\mu, \sigma^2)$, would have limits on reasonable values for the mean, μ, and the standard deviation, σ.
- A probability vector to sample distributions from $\mathcal{P}$, $w = \big(w_1, \ldots, w_{|\mathcal{P}|}\big)$ such that $\sum_{i=1}^{|\mathcal{P}|} w_i = 1$ and $w_i \in [0, 1] \; \forall \; i = 1, \ldots, |\mathcal{P}|$.
- A maximum number of iterations, $M \in \mathbb{N}$.
- Two selection parameters: one to indicate the proportion of the fittest individuals to carry forward, $b \in [0, 1]$, and to allow for a small proportion of "lucky" individuals in the next generation, $l \in [0, 1]$.
- A mutation probability, $p_m \in [0, 1]$, used to indicate the intensity and likelihood of the mutation a new individual might undergo.

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

Algorithm 1: The evolutionary dataset optimisation algorithm

```
Data: f, N, R, C, P, w, M, b, l, μ, s
Result: 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

```
Data: parents, N, R, C, P, w, μ
Result: 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
```

The statement of the EDO algorithm is deliberately vague here. This is, in part, because of the customisable nature of its build but also to lay out its general structure from a high level perspective. Lower level discussion is provided in Section 2.2 where additional algorithms for the individual creation, evolutionary operator and shrinkage processes are given.

Note that there are no defined processes for how to stop the algorithm or adjust the mutation probability,

p_m. These form two key areas of potential customisation since such conditions can be specific to the problem domain. With that, as is detailed in the Python implementation, a user may define their stopping condition or p_m-adjustment to be any reasonable function. Some examples include:

- These should be real-world examples used in other EA with references.
- Stopping when the difference in the variance of the current and last population fitnesses is below some tolerance.
- Halving the mutation probability every 100 iterations.

2.2 Internal mechanisms

Below are detailed instructions and discussion around the internal mechanisms of EDO; each component has an affiliated algorithm and diagram (where appropriate). In addition to the traditional biological operators, there are particular mechanisms for representing individuals as well as shrinking the mutation space.

2.2.1 Individuals

Evolutionary algorithms operate on succeeding populations of individuals often coined as "generations". Typically, an individual would be encoded as a bit string of a fixed length and treated as a chromosome-like object to be manipulated. In EDO, as the objective is to generate datasets, there is no encoding process. Instead, the datasets are manipulated directly so that the biological operators can behave and be interpreted in a more meaningful way.

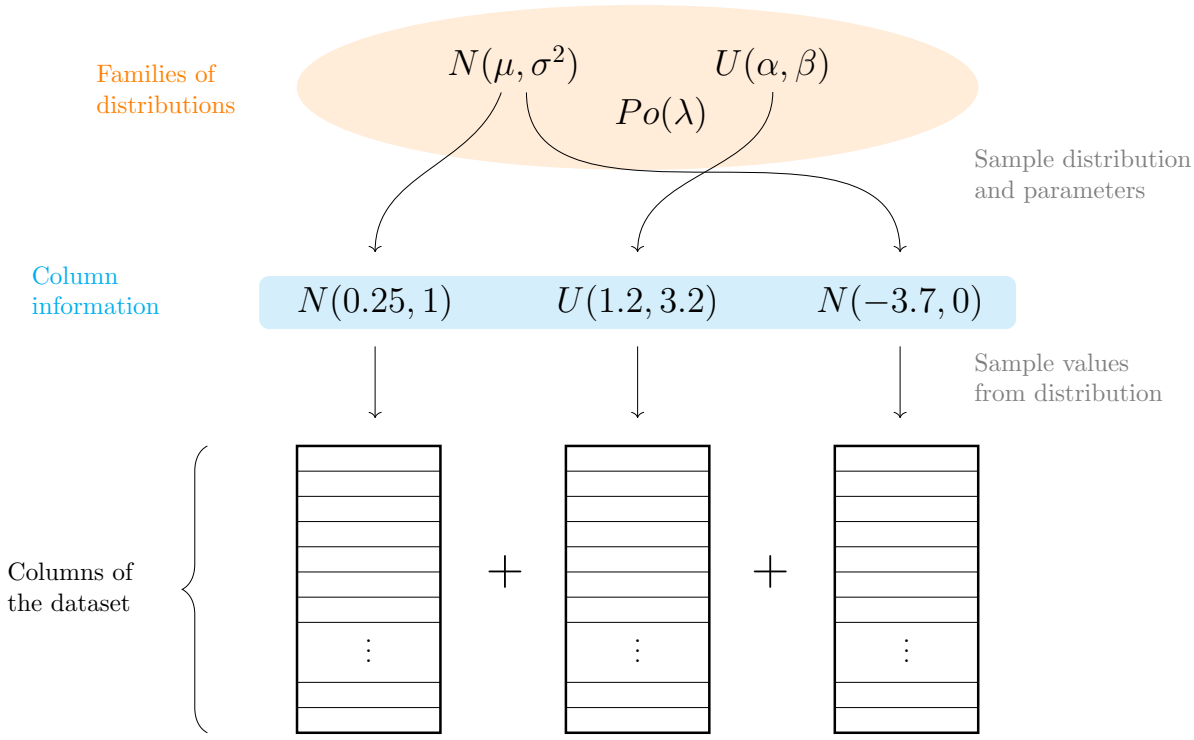

Figure 2: An example of how an individual is first created.

In a sense, a dataset is treated similarly to a classical bit string as the primary components (loci) of the dataset are considered to be its columns. As is seen in Figure 2, an individual's creation is defined by the random generation of its columns. A set of instructions on how to sample new values for that column are recorded in the form of a probability distribution. These distributions are sampled from a pool of distribution families which is passed to the evolutionary algorithm along with the other parameters.

Obviously, users and interpreters of EDO should not be so quick to assume that these pairs of distributions and columns are typical of their partner. That is, that the columns are a reliable representative of the distribution associated with them, or vice versa. A caveat to this statement: 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. In the case of the latter, the column metadata can become more useful for casually analysing the data which is generated. In any case, however, more direct and sophisticated methods should be employed to understand the structure and characteristics of the data before formal conclusions are made.

Algorithm 3: Creating an individual

```
Data: R, C, P, w
Result: 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 P
        create an instance of the distribution
        fill in the column by sampling from this instance
        record the instance in the metadata
    end
end
```

2.2.2 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.

In EDO, a modified truncation selection method is used. Standard 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. The modification is an optional stage after the best individuals have been chosen. By passing some small l to the evolutionary algorithm, a number of random individuals can be selected to be carried forward. This number is given by $n_l = \lceil lN \rceil$. It should be noted that even with this modification, no individual may be selected more than once in a single iteration.

The capability to include random individuals is included to encourage diversity. This has a place in more complex optimisation scenarios - or for larger populations where a greater loss of diversity is seen in the selection process simply due to the nature of truncation selection [3] - and should be used sparingly so as not to dominate the selection process with unwanted randomness.

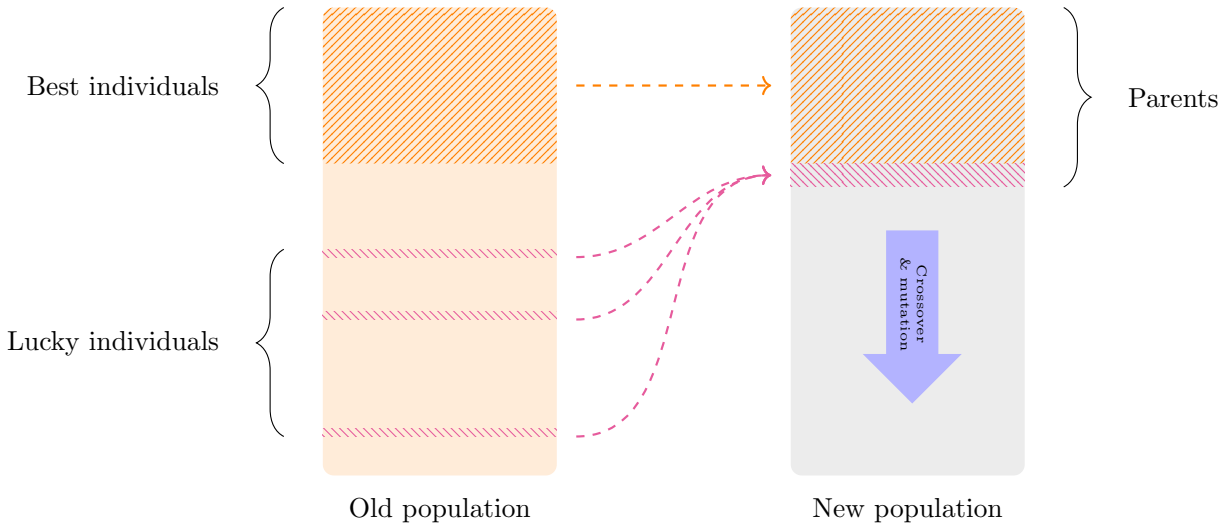

Figure 3: The selection process with the inclusion of some lucky individuals.

Algorithm 4: The selection process

Data: population, population fitness, b, l
Result: 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

2.2.3 Crossover

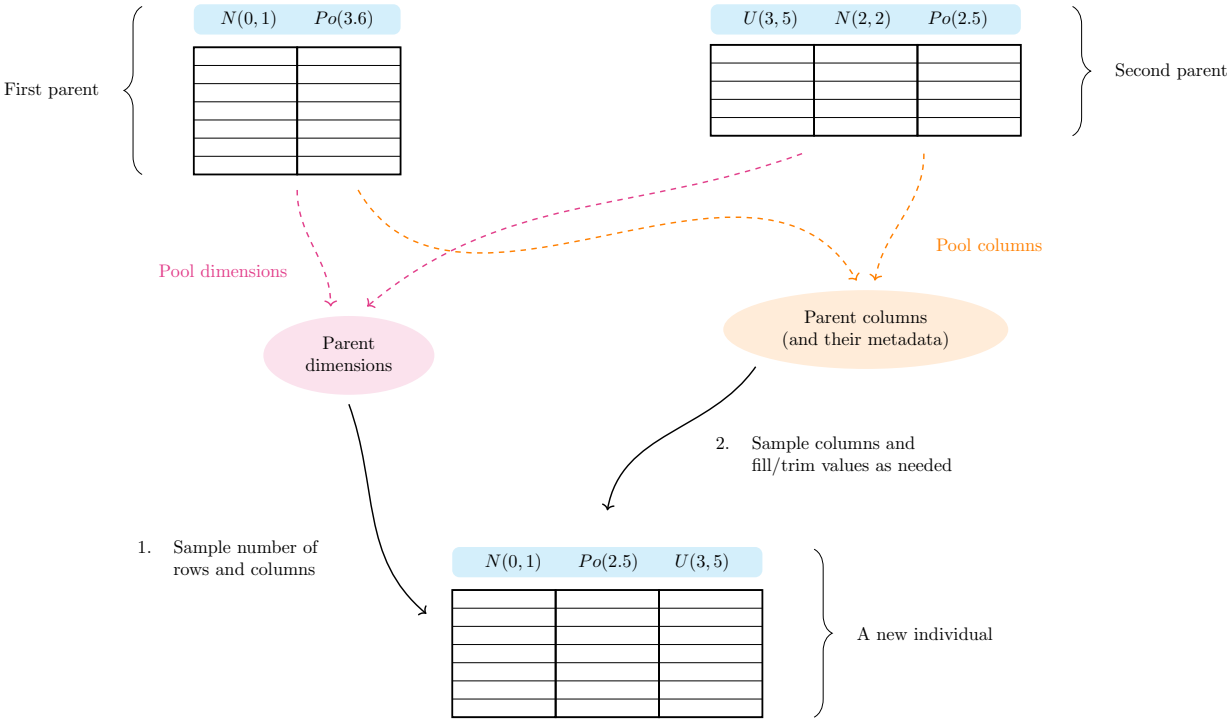

Figure 4: The crossover process.

2.2.4 Mutation

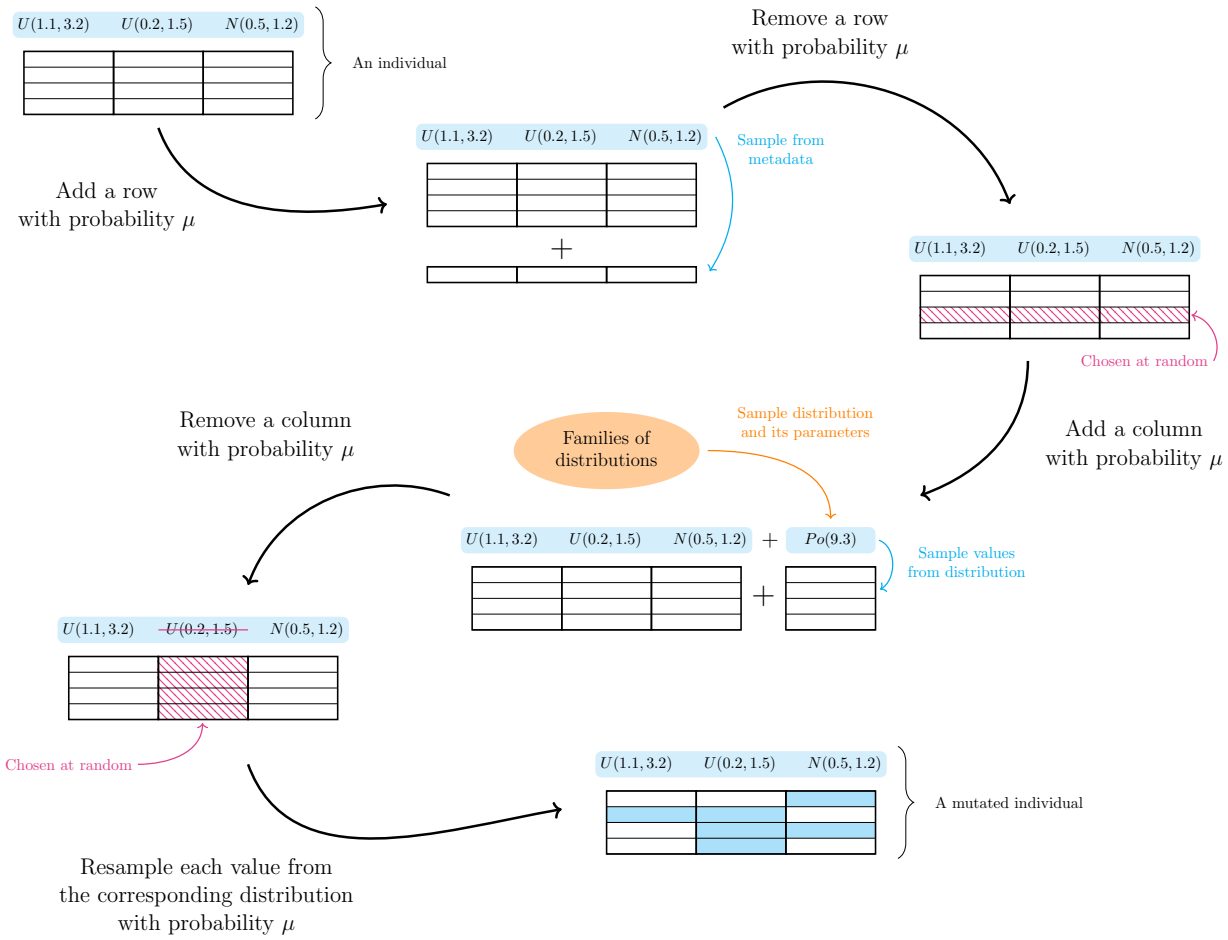

Figure 5: The mutation process.

2.2.5 Shrinking

The potential benefits of adapting the search space of an EA has been well-discussed in the domain of complex optimisation problems. During the development of EDO, two methods were considered to do this. Each of these methods relied on a law relating a successive generation's search space with its predecessor. It should be noted that in both cases the methods were adapted to conform with the choice to represent individuals as they are rather than as bit strings.

The first was developed to be a two-part process based on a linear law with two parameters: a shrink factor, $s \in (0, 1)$, and the maximum number of iterations $M \in \mathbb{N}$ [2]. The adapted method would be as follows. For all iterations, $t \geq sM$, no shrinking would take place. However, for all previous iterations, the

suggested method would take the parents found during selection and act on each component of the search space. That is, for each iteration, $t < sM$, every component would have its lower and upper limits, denoted by l_t and u_t respectively, adjusted so that they are centred about the mean parent value, μ, and be such that:

$$u_{t+1} - l_{t+1} = (u_t - l_t)\left(1 - \frac{t}{sM}\right)$$

More specifically, the adjusted limits would be calculated as follows:

$$l_{t+1} = \max\left\{l_t,\ \mu - \frac{1}{2}(u_t - l_t)\left(1 - \frac{t}{sM}\right)\right\}, \qquad u_{t+1} = \min\left\{u_t,\ \mu + \frac{1}{2}(u_t - l_t)\left(1 - \frac{t}{sM}\right)\right\}$$

The second method was for use in a genetic algorithm which mapped weighted bit strings to some predefined interval with lower and upper bounds [1]. The proposed method would rely on a power law with a single parameter: some shrink factor, $s \in [0, 1]$. Again, at each iteration, the parents would be taken and every component's limits would be adjusted so that they were centred about the mean parent value, μ, such that:

$$u_{t+1} - l_{t+1} = (u_t - l_t)s^t$$

The process by which the values of l_{t+1} and u_{t+1} would be found are equivalent to the above but with a different shift term:

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

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

From these brief definitions alone, these methods appear to be largely indistinguishable. However, following a wider discussion around how EDO would work for the general user, it was decided that the first method be rejected in favour of the second. It was found that the second method having fewer parameters was a particularly redeeming feature; this removed any hidden or otherwise unwanted interactions between a parameter dedicated to the shrink process, s, and the maximum number of iterations, M, which is often used as a fallback stopping criterion in complex problem domains.

2.3 Implementation

- Documentation: `https://edo.readthedocs.io`

Algorithm 5: Shrinking the mutation space

```
Data: parents, current iteration, 𝒫, M, s
Result: A new mutation space focussed around the parents
begin
    for each distribution in 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
```

- 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.

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] Adil Amirjanov and Fahreddin Sadikoglu. "Linear adjustment of a search space in genetic algorithm". In: *Procedia Computer Science* 120 (2017), pp. 953 –960. DOI: https://doi.org/10.1016/j.procs.2017.11.331.

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