An in-depth approach to fitting probability distributions using Evolutionary Algorithms

Eduard M. Constantinescu
e.constantinescu@student.utwente.nl

ABSTRACT

The practice of estimating probability density functions is well-known in the field of statistics and probability theory, with commonly used techniques including histograms, kernel estimators, orthogonal series and nearest-neighbor methods. Furthermore, goodness-of-fit tests are being used for evaluating the accuracy of a statistical model, based on a provided set of observations. While these practices have been widely used so far, they heavily rely on human assumptions and deductions and. This causes them to be susceptible to errors and bias. Techniques using evolutionary algorithms - a metaheuristic optimization type of algorithms - have had optimistic results in estimating the parameters of an assumed distribution function and, as later works have shown, determining the most probable distribution type with its respective parameters. However, recent work scarcely provide any discussions about the importance of the design choices behind important aspects of the evolutionary algorithm. With these facts in mind, the goals of this paper are to explore the alternative of using an evolutionary algorithm to not only estimate the parameters of a probability distribution but to also choose which type of distribution shape would fit the observed data best and to provide an in-depth discussion which will allow for easier replicability.

Keywords

evolutionary algorithm, genetic algorithm, probability distributions, probability density functions, metaheuristics

1. INTRODUCTION

Complex real-world systems usually generate large quantities of data points through multiple methods and require carefully chosen metrics which help evaluate performance and reliability. To exemplify, system component failures could be measured as one time series and, in order to investigate the likelihood of the overall system failure, a likelihood of failure per system component is required. Hence, the task at hand is to model each individual component's likelihood of error as a probability distribution. The distribution of these random variables must be chosen carefully since they influence the accuracy of the results and hence their trustworthiness [1].

33rd Twente Student Conference on IT July 3rd, 2020, Enschede, The Netherlands.

The time-honored histogram is widely used among scientists and others to represent, at least approximately, the shape of the probability density function [2]. However, while these methods have proven to be effective in producing accurate estimates, it is imperative to explore different alternatives, with the hope of uncovering innovative and improved techniques which could enhance, or even replace, already existing ones.

One such alternative proposed in this paper is the usage of *evolutionary algorithms*, which are often viewed as function optimizers, although the range of problems to which these algorithms have been applied is quite broad [3]. More than that, they are metaheuristic algorithms, which attempt to combine basic heuristic methods to efficiently and effectively explore the solution space [4]. Being characterized as "approximate algorithms", they guide the search process in a non-deterministic manner, allowing for near-optimal solutions to be found in rapid time. In short, they sacrifice the guarantee of uncovering optimal solutions, while focusing on producing good solutions in a significantly reduced amount of time [4].

Thus, the proposed goal is to design, develop and fine-tune an evolutionary algorithm which can be reliably used in estimating probability distributions and, more than that, encourage scholars to engage in researching and practicing such algorithms in the future. Furthermore, one key element that this paper emphasizes is the replicability of the propsed approach, which will hopefully make it more accessible for others to improve this work. Finally, the paper attempts to answer the following research question:

By building on top of previous works, how should an evolutionary algorihm for fitting probability distributions be designed and to what extent can it be made accessible and easy to replicate?

In order to fully explore this research question, attention needs to be paid to the following underlying questions, which help in shaping the overall design of the proposed algorithm:

- Q1 How should the input data be encoded in order to build the genetic profile of a chromosome?
- Q2 How should an accurate and reliable evaluation function be designed?
- Q3 How should good partial solutions (chromosomes) be selected for reproduction?
- Q4 How should crossover and mutation rules be designed in order to ensure genetic diversity and reliable convergence?
- Q5 What are the criterias that must be met in order to terminate the algorithm?

Q6 How can accessibility and replicabilty be achieved?

Q7 What are the issues when designing such an algorithm and what possible solutions are there for them?

The paper is structured as follows: Section 2 provides some information on regular fitting techniques and briefly discusses the current state of the art for the evolutionary algorithm approach; Section 3 describes the design choices and the implementation of the proposed evolutionary algorithm; Section 4 evaluates the performance of the proposed algorithm when tested against random data and a series of real-life data; Section 5 provides some concluding remarks and discusses possible future improvements of this work.

2. BACKGROUND & RELATED WORK

The task of estimating a probability density function can be represented as follows: given a sequence of independent identically distributed random variables
$X_1, X_2, ..., X_n$ with common probability density function $f(x)$, how can one estimate $f(x)$ [5]? In most traditional approaches, this involves a three-step procedure: in the first step, a specific family of distributions is hypothesized [1], which usually includes well-known distributions, such as the normal, exponential or Weibull distributions. The second and third steps involve estimating the parameters of the assumed distribution and performing goodness-of-fit tests to determine the accuracy of results.

Nonparametric density estimation concerns the problem of estimating $f(x)$ when no formal parametric structure is specified [6]. Hence, the term "nonparametric" is ideally a procedure which is valid irrespective of the type of distribution from which the sample is taken [2]. The most notable example of estimating f would be the histogram, which was shortly followed by kernel estimators, orthogonal series and nearest-neighbor methods [6]. However, it must be mentioned that such procedures are valid only for members of some fixed family of distributions and it is the size of this family which distinguishes parametric from nonparametric methods [2].

One of the most popular evolutionary algorithms, namely the genetic algorithm, refers to a model introduced and investigated by John Holland and his students in 1975 [3]. This algorithm is a stochastic global search method that mimics the behaviour of natural biological evolution [1]. In the common implementations, it generates a population of (usually random) chromosomes and then evaluates these structures and allocates reproductive opportunities in such a way that those chromosomes which represent a better solution are given more chances to 'reproduce' [3]. This approach was used by Strelen [1] to generate distribution parameters which fit random samples of Weibull, Two-Mode Weibull and Gamma distributions with remarkable accuracy: Z-scores (number of standard deviations above the expected results) of $0.0293, 0.024$ and 0.031 respectively. Other similar techniques, such as using a Breeder Genetic Algorithm [7] or using continuous evolutionary algorithms [8] were used for the same purpose, producing promising results. Finally, a 1995 article showcased an implementation of a genetic algorithm for Weibull parameter estimation which proved to be superior to the usually used Newton-Raphson Algorithm [9].

Furthermore, Strelen used his genetic algorithm in an experiment for differentiating between a Gamma and a Weibull distribution [1]. His approach included a Weibull distribution as input and attempted to fit the data to mixed Weibull and Gamma distribution,

$$F(x) = p \cdot F_1(x) + (1 - p) \cdot F_2(x)$$

where F_1 is Weibull and F_2 is Gamma [1]. The algorithm outputed a value of $p = 0.998$, resulting in a considerably high accuracy (Z-score of 0.047). Albeit successful, his approach does not attempt to learn the "structure" of the data, as it merely compares partially generated solutions against a human-made assumption - in other words, it excludes the possibility of the result being anything else other than a Weibull or Gamma distribution. Moreover, one may wonder how scalable this approach is when considering a larger family of available distributions.

Finally, a 2010 work by Colla et al. [10] presented a genetic algorithm which was capable of fitting six different continuous probability distribution functions and estimate their parameters. A couple of years laters, the authors published a work which used a modified version of their original algorithm, tasked to fit discrete distributions [11]. While their studies produced positive results when compared to the popular 'Maximum likelihood estimation' for parameter estimation, the respective papers seem to lack a thorough explanation of the design choices behind important aspects of the algorithm. Combined with a lack of pseudocode, their papers do not allow for accessible replicability. With these facts in mind, the following paper will attempt to build upon (and, if possible, improve) previous work in the field of distribution fitting using evolutionary algorithms, with the goal of providing in-depth information about the design aspects to be considered when developing such a tool.

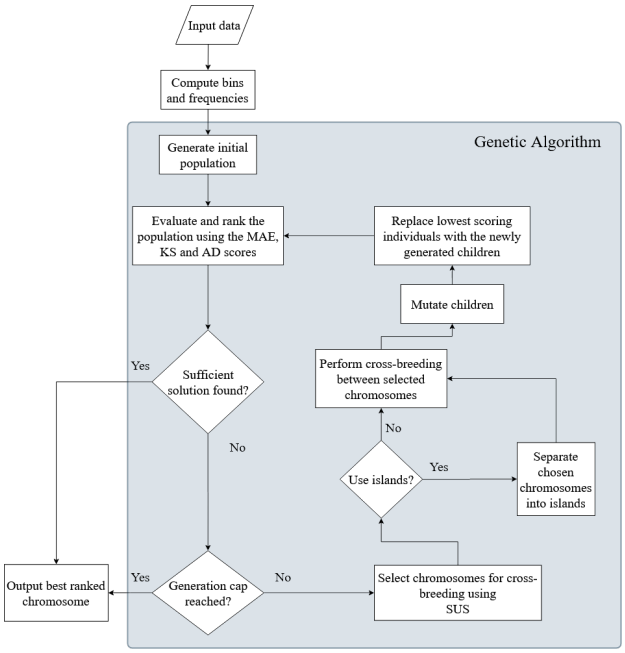

Figure 1. Flow chart of the proposed algorithm

3. THE GENETIC ALGORITHM

In this paper, a genetic algorithm approach for distribution fitting is presented, which has been built by following the work of Colla et al. [10]. Although having a similar goal, this paper presents new considerations towards the *evaluation function* (see Section 3.3), *selection procedure* (see Section 3.4) and *reproduction rules* (see Section 3.5). Likewise, the aim is to design, build and evaluate a tech-

Gene #1	Gene #2	Gene #3	Gene #4	Gene #5
Gaussian	Mean	Standard deviation	None	None
Cauchy	Shape	Scale > 0	None	None
Frechet	Shape > 0	Scale > 0	Location	None
Log-logistic	Shape > 0	Scale > 0	Location	None
Burr12	Shape 1 > 0	Shape 2 > 0	Scale	Location
Dagum	Shape 1 > 0	Shape 2 > 0	Scale > 0	Location

Table 1. Encoding of chromosomes

Gene #1	Estimator #2	Estimator #3	Estimator #4	Estimator #5
Gaussian	Mean = Sample mean	Standard deviation = Sample std	None	None
Cauchy	Shape = Sample mean	Scale = $0.5 \cdot IQR$	None	None
Frechet	-	Scale = (Sample max - Sample min)	Location = (Sample Mean - Scale)	None
Log-logistic	-	Scale = IQR	Location = (Sample Mean - Scale)	None
Burr12	-	-	Scale = (Sample max - Sample min)	Location = (Sample Mean - Scale)
Dagum	-	-	Scale = (Sample max - Sample min)	Location = (Sample Mean - Scale)

Table 2. Estimators for distribution parameters

nique based on an evolutionary algorithm which accepts as input a series of data points and can select a distribution type and its respective parameters that best fit the observed input.

The following subsections will discuss the design choices behind each core aspect of the proposed evolutionary algorithm. Furthermore, figure 1 presents a flow chart of the algorithm.

3.1 Problem encoding

The genetic algorithm requires a robust encoding of the candidate solutions, represented as chromosomes with distinctive genetic profiles. Following the work of Colla et al. [10, 11], the proposed algorithm uses a mixed encoding operator: the first gene determines the distribution type and the next four genes represent the parameter values, for the respective distribution. Furthermore, the genetic profile has a fixed length of 5 genes in order to solve the problem caused by different distributions having a distinct number of parameters. To exemplify, the Cauchy distribution has 2 parameters (”shape” and ”scale”), while the Dagum distribution has 4 parameters (”shape 1”, ”shape 2”, ”scale” and ”location”). Table 1 shows the encoding of all the supported distributions.

3.2 Initial population

The genetic algorithm requires an initial population of individuals which are, usually, randomly generated - this is the case for previous similar studies [1, 10, 11]. However, to achieve a faster convergence time, it is desirable to balance the distribution of chromosomes over the possible solution space, such that the initial population is not biased in any way.

The proposed algorithm achieves this via ‘seeding’ the first generation, a process which involves making several simple parameter estimations based on the provided data set, for all the supported distributions types. It is possible to compute unbiased estimators for the Gaussian distribution, using the sample mean and the sample standard deviation, and for the Cauchy distribution, using the sample mean and half of the interquartile range. However, for the remaining four probability distributions, the proposed estimators have been chosen through experimental trials and, hence, only help in reducing the possible solution to space to a smaller one. More than that, there are several parameters for which no estimation can be made, namely the first paramater of the Frechet and Log-logistic distributions and the first two paramaters for the Burr12 and the Dagum distributions. Finally, after computing the sample estimators for each type of parameter, the initial individuals' parameters are randomly chosen from an interval centered around the sample estimator's value. Table 2 showcases the chosen estimators for each type of distribution, and its respective parameters.

3.3 Evaluation function

Once the problem is encoded, every generation of the population (and its respective individuals) will be evaluated with the help of the evaluation function. As such, this function has to be determined in such a way that the genetic algorithm was able to rank chromosomes based on how well they fit the provided input. To achieve this, previous works relied on using a modified version of a Maximul Absolute Error (MAE) score [1, 10, 11] and a Kolmogorov-Smirnov (KS) test statistic [10, 11]. The proposed algorithm builds on these previous approaches and adds a third score function, namely the Anderson-Darling (AD) goodness-of-fit test statistic. The results of each score function were linearly combined in order to compute a single score for each individual. More specifically:

$$\mathit{fitness}(x) = w_1 \cdot MAE + w_2 \cdot KS + (1 - w_1 - w_2) \cdot AD, \quad (1)$$

$$0 \leq w_1 + w_2 \leq 1 \quad (2)$$

where x is the chromosome to be tested and w_1, w_2 are weighting factors, which values can be changed in order to assign more relevance/importance to the different scoring functions.

3.3.1 Maximum Absolute Error (MAE)

The MAE tests how well a candidate solution fits the observed data. In more detail, it averages the distance between the predicted values of a chromosome and a normalized histogram generated from the provided data set. The number of bins for the generated histogram was chosen as

$$k = \lceil \sqrt{n} \rceil \quad (3)$$

where k is the number of bins and n is the number of points in the data set. This number of bins provides sufficient accuracy in computing the MAE score while, at the same time, reducing the total time complexity of the score computation. Following this, the algorithm computes a set MID consisting of x-coordinates for the middle of each bin.

Finally, the formula for computing the *MAE* score for a given chromosome is:

$$MAE = \frac{1}{k} \cdot \sum_{i=1}^{k} | \, h(MID(i)) - f(MID(i)) \, | \qquad (4)$$

where $h(x)$ is the value of the histogram in point x and $f(x)$ is the value of the candidate probability distribution function in point x.

3.3.2 *Kolmogorov-Smirnov goodness-of-fit test*

The KS test [12] has been designed to test the hypothesis that a given data set has been sampled from a given distribution. It operates by making comparisons between the empirical cumulative distribution function, computed from the provided data points, and the cumulative distribution function of the assumed distribution. The result of the KS test statistic is the maximum absolute distance found between the two CDFs. This test statistic value can range between 0, indicating that the data set was most likely sampled from the tested distribution, and 1, in the complete opposite scenario.

3.3.3 *Anderson-Darling goodness-of-fit test*

The AD test [13] is a modified version of the KS test, which allows for a better analysis of a distribution's tails. Since, for some distributions, the KS test is rather unsensitive to values found outside the body of the distribution, the AD test statistic is used to determine the relevance of the KS score. Unlike the KS test statistic, the AD score ranges between 0 and ∞. Hence, distributions which fit the data better will have scores closer to 0 and, in the opposite scenario, the AD score will linearly increase when the tested distribution does not fit the data - albeit mostly the worst solutions will have score larger than 1. In the scenario where the KS test wrongly ranks one chromosome as a good solution, a large value of the AD test will be able to more accurately determine the respective chromosome's validty.

3.4 Selection procedure

After the chromosomes have been ranked, they must undergo a selection procedure based on their fitness scores. The chosen chromosomes will become 'parents', with the goal of breeding new chromosomes via recombination. Previous works use a straight-forward fitness-based proportional selection procedure [1] or seem to omit to discuss their selection procedure designing [10, 11].

In order to reduce bias and give weaker chromosomes a chance to be chosen, the proposed algorithm uses the "stochastic universal sampling" (SUS) [14] technique, introduced by James Baker. It operates by using a "roulette wheel selection" with multiple pointers, as it can be seen in Figure 2. Whenever a pointer falls onto a chromosome, then it will be selected for reproduction. Finally, the number of chromosomes selected for reproduction is determined by the crossover rate parameter.

3.5 Reproduction

Furthermore, since this approach is generation dependent, there are several other considerations to be held in mind. After several generations, selection may drive most of the individuals to a similar state. If this happens without the genetic algorithm converging to a satisfactory solution, then the algorithm has prematurely converged [3]. Techniques such as crossover and mutation are to be used in order to preserve genetic diversity among the population while minimizing their disruptive effects. These techniques are commonly applied to individuals which have been highly ranked by the evaluating function. A new generation will be created from these highly ranked individuals, by combining the genetic profile of their parents, with the additional option of making random small changes to certain genes.

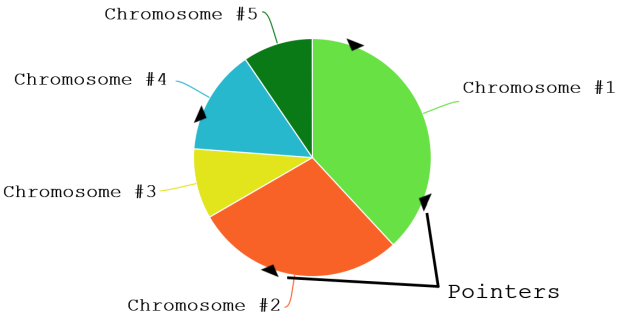

Figure 2. Stochastic Universal Sampling procedure

3.5.1 *Cross-breeding*

Chromosomes selected via SUS (see Section 3.4) will undergo the cross-breeding operation, generating a new set of chromosomes for the following generation. The proposed algorithm randomly groups the 'parents' into pairs of two and generates two new children for each such pair. Initially, the children are exact copies of the parents. In order to cross-breeed, the algorithm uses a uniform cross-breeding operator, which makes use of a randomly generated crossover mask to determine which genes will be swapped between the children.

The mask is an array with a fixed size equal to the number of genes in the chromosome encoding, which elements can have a value of 1 or 0. A value of 1 indicates that the respective gene will be swapped, while a value of 0 will keep the gene in place. The algorithm does not take into account the value of the first element since the first gene determines the distribution type and, hence, should not be considered for crossover. Finally, the newly generated chromosomes will replace the lowest scoring individuals in the current population. Figure 3 depicts an example of the cross-breeding procedure between two chromosomes.

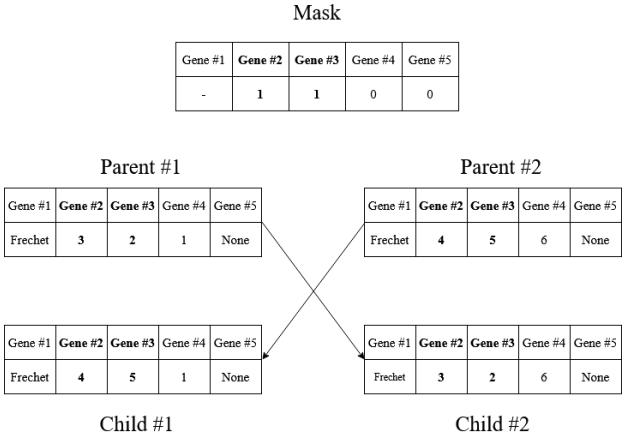

Figure 3. Example of cross-breeding between two 'Frechet' chromosomes; genes #2 and #3 have been swapped according to the generated crossover mask

Distribution type	Parameter #1	Parameter #2	Parameter # 3	Parameter #4
Gaussian	$\mu = 4$	$\sigma = 2$	None	None
Cauchy	$x_0 = 3$	$\gamma = 2$	None	None
Frechet	$\alpha = 10$	$s = 1.5$	$location = 4$	None
Log-logistic	$\beta = 3$	$\alpha = 1$	$location = 2$	None
Burr12	$c = 10$	$k = 4$	$scale = 1$	$location = 0$
Dagum	$p = 7$	$a = 3$	$b = 1$	$location = 2$

Table 3. Choice of distributions for random input generation.

3.5.2 Mutation

Building on the approach of previous works [1, 10, 11], the mutation process is randomly applied to newly generated chromosomes, with a starting chance of 0.01 per individual gene – with the exception of the first gene, which will not be chosen for mutation, as to not ruin the genetic integrity of a chromosome. However, in order to preserve genetic diversity and to allow for more accurate convergence, the proposed algorithm makes use of an adaptive mutation operator and mutation rate.

In more detail, the mutation rate will increase by 0.001 for each successive generation. Likewise, the mutation operator will allow for larger mutations at the beginning and, as the generation counter increases, will limit the size of the mutations in order to allow for small refinements of the candidate solutions. The proposed algorithm starts with mutation changes of ±50% to the gene value and linearly decrease the size of change down to ±10% of the gene value. Hence, the size of the change is computed at each generation i, by using the following formula:

$$size\_of\_change(i) = -\frac{0.4}{generation\_cap} \cdot i + 0.5 \qquad (5)$$

3.5.3 Issues with regular cross-breeding techniques

As seen in Section 3.1, different types of distributions support between 2 and 4 parameters, thus rendering ineffective any simple cross-breeding techniques. To exemplify, a possible scenario can involve 2 chromosomes which have a different number of parameters, such as a Gaussian distribution and a Frechet one. By simply copying genes from their parents using the crossover mask, the resulting children can be 'corrupted' if, in this example, a child Frechet distribution will have less than 3 parameters. Figure 4 depicts such a scenario.

It is clear that, in such a scenario, the cross-breeding operator will make no progress towards generating better candidate solutions. Since previous works which use similar chromosome encodings [10, 11] seem to overlook these issues, the proposed algorithm allows for 2 different solutions that help in mitigating these problems:

(a) **Special rules & integrity checker**
The cross-breeding operator will be adapted to not copy any genes that might affect the validity of a chromosome's genetic profile. Applying this to the previous example, a 'None' type gene will not be copied to any of the first three gene slots of the Frechet chromosome. Lastly, an integrity checker evaluates each child's genetic integrity, by verifying that all the genes are in their corresponding value range (see Table 1). While this solution ensures that newly generated chromosomes are valid, it also increases the time complexity of the proposed algorithm by quite a margin.

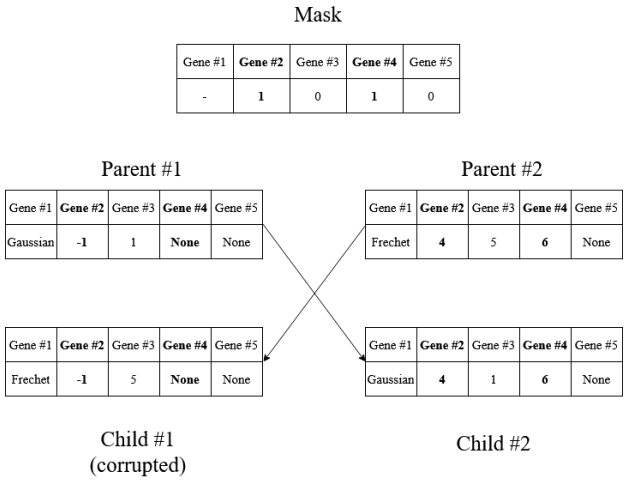

Figure 4. Example of cross-breeding between two chromosomes with different number of parameters; genes #2 and #4 have been swapped according to the generated crossover mask, resulting in a 'corrupted' child

(b) **Chromosome islands**
As a slightly more efficient solution, the proposed algorithm can separate parent chromosomes based on their first gene, which indicates the assumed distribution type. Thus, several distinct groups of chromosomes will be created, which can be thought of as 'chromosome islands' in a given population. Finally, cross-breeding will only be allowed between chromosomes from the same island, thus removing any possibility of a potential genetic corruption. While this solution is less complex than the previous one, it reduces the genetic diversity of the children chromosomes.

3.6 Stopping conditions

Similar to most genetic algorithms, the proposed algorithm specifies two distinct stopping conditions: terminate when the total number of generation has reached the generation cap or, if a sufficiently good solution has been found, terminate at generation $i < generation\_cap$. Initially, the algorithm was tested without any early stopping conditions in order to determine a favorable stopping threshold by observing the results after full convergence. Hence, a most favorable threshold determined through experimental observation is a maximum aggregate fitness score of 0.05, for the best evaluated chromosome in the current generation.

4. RESULTS

The proposed algorithm has been developed and tested using the Python programming language, more specifically the stable version 3.8. Python offers a wide varieties of pre-existing modules for generating random data, deploying

statistical tests and plotting of results - which all fall under the scope of this research. Furthermore, it is appealing due to its accessibility, portability and extensibility.

Access to the proposed algorithm's implementation can be found on the following public repository: `https://git.snt.utwente.nl/s1922629/GA-PDF-fitting`.

4.1 Testing with random input

The algorithm's behavior was initially tested with randomly generated data sets. Between 1,000 and 10,000 data points were sampled from a list of predefined distributions. This section showcases one example of fitting each type of supported probability distribution. The choice of distributions and their respective parameters can be seen in Table 3. A sample list of more tests can be found in Appendix A.

4.1.1 Fitting a Gaussian distribution

The input data consists of 1000 data points sampled from a Gaussian distribution with mean $\mu = 4$ and standard deviation $\sigma = 2$. Weights assigned for the evaluation function are $w_1 = \frac{1}{3}, w_2 = \frac{1}{3}$. The algorithm ran with a generation cap of 100 and a crossover rate of 0.6. The best ranked chromosome was a Gaussian distribution with mean $\mu = 4.010$ and standard deviation $\sigma = 1.760$.

4.1.2 Fitting a Cauchy distribution

The input data consists of 1000 data points sampled from a Cauchy distribution with shape $x_0 = 3$ and scale $\gamma = 2$. Weights assigned for the evaluation function are $w_1 = \frac{1}{3}, w_2 = \frac{1}{3}$. The algorithm ran with a generation cap of 100 and a crossover rate of 0.6. The best ranked chromosome was a Cauchy distribution with shape $x_0 = 3.335$ and scale $\gamma = 2.230$.

4.1.3 Fitting a Frechet distribution

The input data consists of 2500 data points sampled from a Frechet distribution with shape $\alpha = 10$, scale $s = 1.5$ and location 4. Weights assigned for the evaluation function are $w_1 = \frac{1}{3}, w_2 = \frac{1}{3}$. The algorithm ran with a generation cap of 100 and a crossover rate of 0.6. The best ranked chromosome was a Frechet distribution with shape $\alpha = 11.912$, scale $s = 1.746$ and location 3.754.

4.1.4 Fitting a Log-logistic distribution

The input data consists of 10000 data points sampled from a Frechet distribution with shape $\beta = 3$, scale $\alpha = 1$ and location 2. Weights assigned for the evaluation function are $w_1 = \frac{1}{3}, w_2 = \frac{1}{3}$. The algorithm ran with a generation cap of 100 and a crossover rate of 0.6. The best ranked chromosome was a Frechet distribution with shape $\beta = 2.663$, scale $\alpha = 0.871$ and location 2.221.

4.1.5 Fitting a Burr type XII distribution

The input data consists of 5000 data points sampled from a Burr type XII distribution with shape parameters $c = 10$ and $k = 4$, scale 1 and location 0. Weights assigned for the evaluation function are $w_1 = \frac{1}{3}, w_2 = \frac{1}{3}$. The algorithm ran with a generation cap of 100 and a crossover rate of 0.6. The best ranked chromosome was a Burr type XII distribution with shape parameters $c = 8.307$ and $k = 2.307$, scale 0.621 and location 0.301

4.1.6 Fitting a Dagum distribution

The input data consists of 5000 data points sampled from a Dagum distribution with shape parameters $p = 7$ and $a = 3$, scale 1 and location 2. Weights assigned for the evaluation function are $w_1 = \frac{1}{3}, w_2 = \frac{1}{3}$. The algorithm ran with a generation cap of 100 and a crossover rate of 0.6. The best ranked chromosome was a Dagum distribution with shape parameters $p = 6.407$ and $a = 2.957$, scale 0.989 and location 1.978

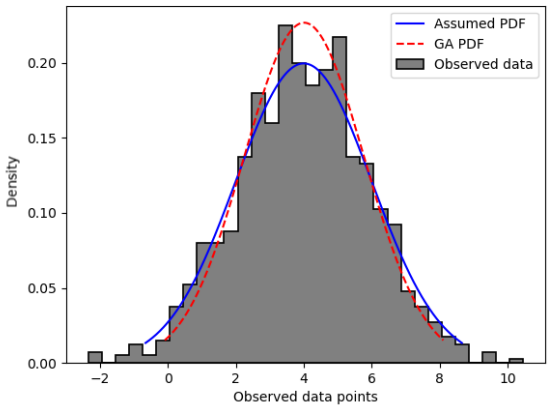

Figure 5. Fitted Gaussian distribution

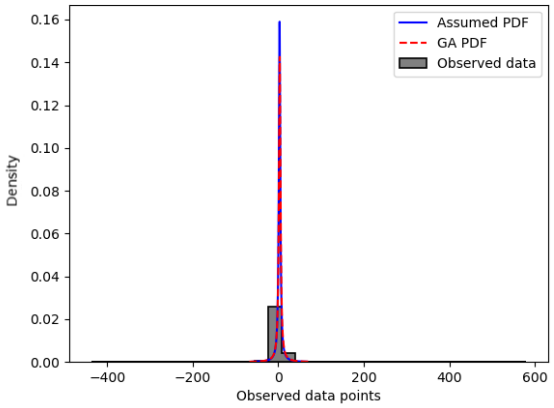

Figure 6. Fitted Cauchy distribution

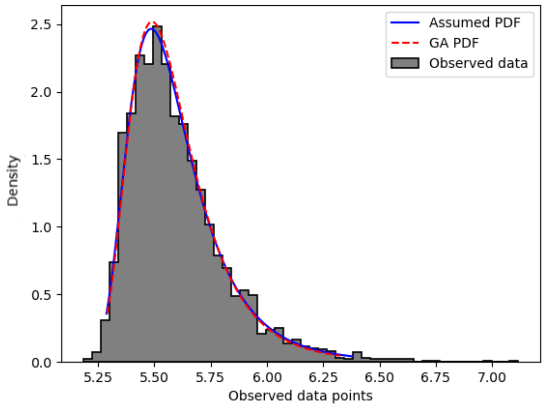

Figure 7. Fitted Frechet distribution

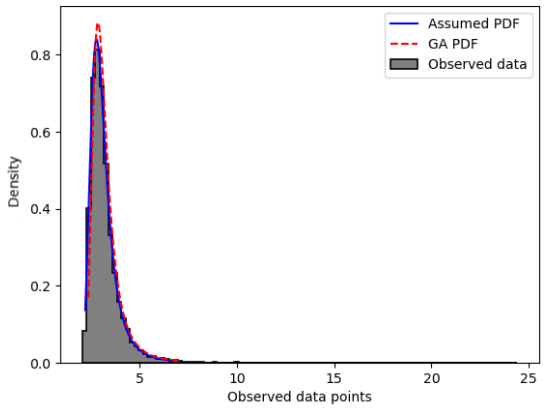

Figure 8. Fitted Log-logistic distribution

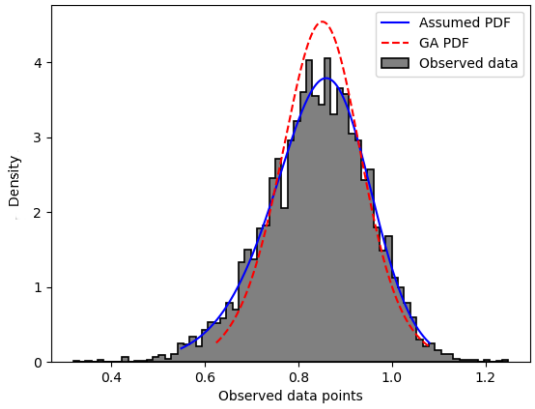

Figure 9. Fitted Burr type XII distribution

4.2 Testing with real-life data

As a case scenario, the proposed algorithm was used on three distinct data sets for the purpose of distribution fitting and parameter estimation. These data sets concern the development of the COVID-19 outbreak in The Netherlands during the 27th of February 2020 - 25th of June 2020 period [15]. More specifically, they track the number of new cases, closed cases and, respectively, hospitalised cases, on a daily basis. Due to its nature, the COVID-19 virus should have seen a large, constant increase in the total number of each type of cases over time. However, the results of the societal measured imposed soon after the sudden spike in number of case can be seen in each of the three data sets. In each of the data sets, the large number of data points near the origin seem to indicate that the number of new daily cases has capped. As a consequence, the tail of each distribution should actually decrease in size as time passes - granted that the societal measures remain honored. Based on these remarks, the assumption is that all of the three data sets are either Log-logistic or Frechet distributed.

Finally, the results of the proposed algorithm can be seen in Figure 11. Each data set was fitted with a Log-logistic distribution, confirming the previous assumption. The best fit found for each data set can be seen in Table 4.

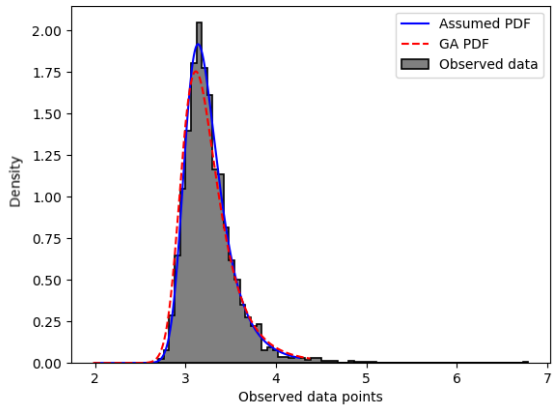

Figure 10. Fitted Dagum distribution

Data set	Gene #1	Gene #2	Gene #3	Gene #4	Gene #5
New cases	Log-logistic	1.511	258.412	-26.322	None
Closed cases	Log-logistic	1.569	35.592	-7.853	None
Hospitalised cases	Log-logistic	1.780	56.647	-20.798	None

Table 4. Results of COVID-19 data set fitting

5. CONCLUSION AND FUTURE WORK

This paper has showcased a proposed in-depth approach for probability distribution fitting and parameter estimation using genetic algorithms. By building on some core concepts of previous similar works, it was possible to experiment with new design choices in a rapid manner. Additionally, by making notes of these design decision, the paper presents and discusses a detailed approach for understanding and replicating the proposed algorithm. As far as the scope of this research is concerned, the hope is that the discussions of Section 3 have successfully presented possible answers for questions Q1-Q5. On a similar note, these discussions should make the reader feel more comfortable in understanding how such an algorithm works and how it can be implemented, by providing a prototype example. Thus, the expectation is that the approach of this paper acts as a good example for answering Q6. However, the proposed algorithm still has some unsolved issues, which should be discussed in order to properly answer question Q7:

I1 As seen in Table 2, the proposed algorithm did not succeed in 'seeding' all types of probability distributions. Future works could include finding new estimators or improving the already existing ones. Otherwise, this algorithm will require a broader coverage of the possible solution space in order to tackle more complex input data sets.

I2 The example implementation's time complexity is heavily affected by the standard implementation of the Anderson-Darling test. In order to achieve faster, more reliable results, attention needs to be paid to improving the computation time of the AD test statistic.

I3 It can be seen in Appendix A that most of the best fitted distributions are Cauchy, which can be explained by how easily it manages to overfit the data. While the proposed evaluation function was designed as to avoid overfiting, by using three different scoring approaches, it seems that future works have to be made in order to improve the current solution.

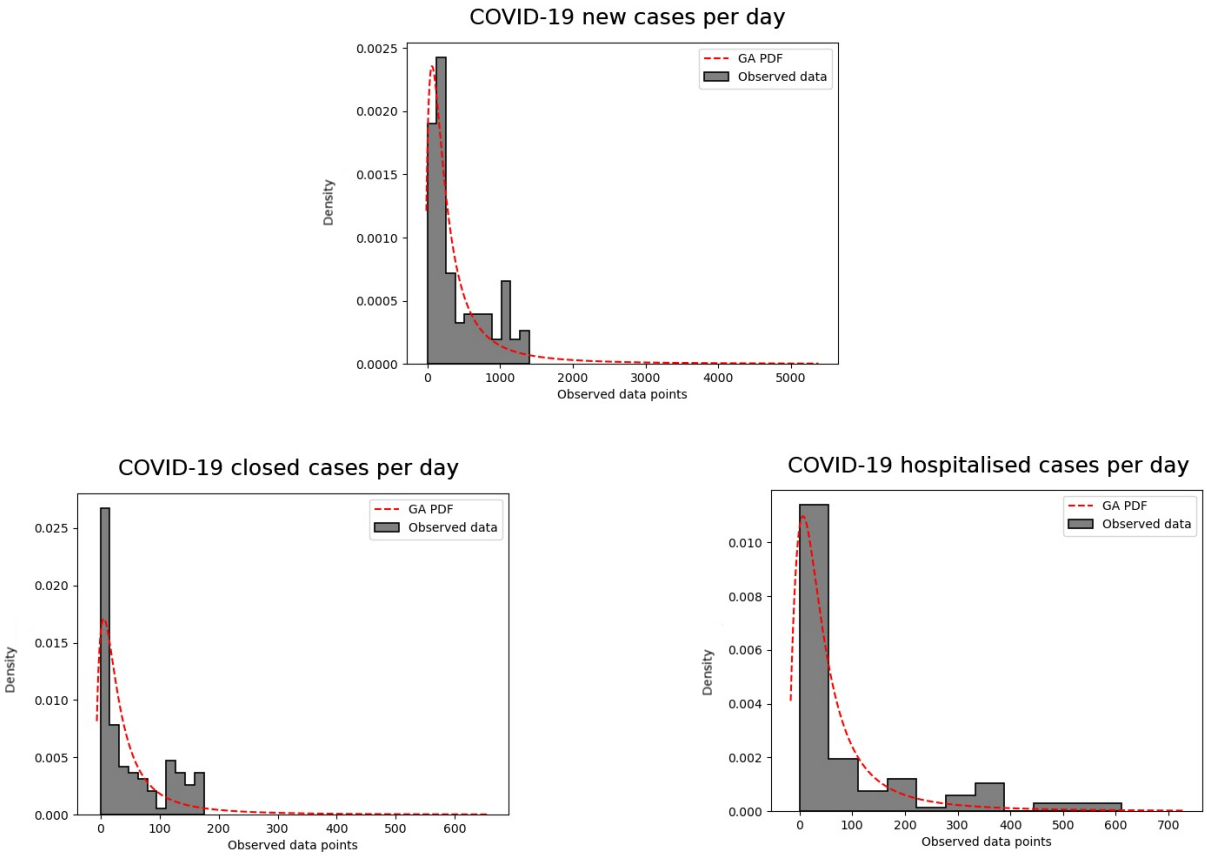

Figure 11. Log-logistic fitting for three distinct COVID-19 data sets

6. REFERENCES

[1] Johann Christoph Strelen. The Genetic Algorithm is Useful to Fitting Input Probability Distributions for Simulation Models. *Proceedings of the Business and Industry Symposium-ASTC 2003*, pages 8–13, 2003.

[2] Edward J. Wegman. Nonparametric Probability Density Estimation: I. A Summary of Available Methods. *Technometrics*, 14(3):533–546, 1972.

[3] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, 1994.

[4] Christian Blum and Andrea Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[5] E Parzen. On Estimation of a Probability Density Functions and Mode. *The Annals of Mathematical Statistics*, 33:1065–1076, 1962.

[6] Alan Julian Izenman. Recent Developments in Nonparametric Density Estimation. *Journal of the American Statistical Association*, 86(413):205, 1991.

[7] H. Mühlenbein, J. Bendisch, and H. M. Voigt. From recombination of genes to the estimation of distributions II. Continuous parameters. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1141:188–197, 1996.

[8] Stefan Kern, Sibylle D. Müller, Nikolaus Hansen, Dirk Büche, Jiri Ocenasek, and Petros Koumoutsakos. Learning Probability Distributions in Continuous Evolutionary Algorithms - a Comparative Review. *Natural Computing*, 3(3):355–356, 2004.

[9] M Thomas, R Gerth, T Velasco, Systems Engineering, and Manufacturing Engineering. Using real-coded genetic algorithms for Weibull parameter estimation. *Computers & Industrial Engineering*, 2(14):377–381, 1995.

[10] Valentina Colla, Gianluca Nastasi, and Nicola Matarese. GADF - Genetic algorithms for distribution fitting. *Proceedings of the 2010 10th International Conference on Intelligent Systems Design and Applications, ISDA'10*, (November):6–11, 2010.

[11] Valentina Colla, Gianluca Nastasi, Silvia Cateni, Marco Vannucci, and Marco Vannocci. Genetic algorithms applied to discrete distribution fitting. *Proceedings - UKSim-AMSS 7th European Modelling Symposium on Computer Modelling and Simulation, EMS 2013*, pages 30–35, 2013.

[12] Indra M Chakravarty, JD Roy, and Radha Govind Laha. Handbook of methods of applied statistics. 1:392–394, 1967.

[13] M. A. Stephens. Edf statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 69(347):730–737, 1974.

[14] James E Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the second international conference on genetic algorithms*, volume 206, pages 14–21, 1987.

[15] Ontwikkeling covid-19 in grafieken. `https://www.rivm.nl/coronavirus-covid-19/grafieken`. Accessed on: 2020-06-25.

APPENDIX

A. SAMPLE RESULTS FOR DIFFERENT WEIGHTING FACTORS

Test No.	Gene #1	Gene #2	Gene #3	Gene #4	Gene #5	W1	W2	W3	Fit #1	Fit #2	Fit #3	Fit #4	Fit #5
1	Gaussian	4	2	None	None	0.33	0.33	0.33	Gaussian	4.010	1.760	None	None
2	Gaussian	4	2	None	None	1	0	0	Cauchy	4.149	1.544	None	None
3	Gaussian	4	2	None	None	0	1	0	Cauchy	3.893	1.298	None	None
4	Gaussian	4	2	None	None	0	0	1	Cauchy	4.348	1.214	None	None
5	Gaussian	4	2	None	None	0.5	0.5	0	Cauchy	4.260	1.480	None	None
6	Cauchy	3	2	None	None	0.33	0.33	0.33	Cauchy	3.335	2.223	None	None
7	Cauchy	3	2	None	None	1	0	0	Cauchy	5.639	3.823	None	None
8	Cauchy	3	2	None	None	0	1	0	Cauchy	2.239	2.155	None	None
9	Cauchy	3	2	None	None	0	0	1	Cauchy	2.650	1.739	None	None
10	Cauchy	3	2	None	None	0.5	0.5	0	Cauchy	2.856	2.078	None	None
11	Frechet	10	2	4	None	0.33	0.33	0.33	Frechet	11.912	1.746	3.754	None
12	Frechet	10	2	4	None	1	0	0	Log-logistic	12.536	1.587	3.969	None
13	Frechet	10	2	4	None	0	1	0	Frechet	13.222	3.580	2.960	None
14	Frechet	10	2	4	None	0	0	1	Log-logistic	6.035	0.660	4.922	None
15	Frechet	10	2	4	None	0.5	0.5	0	Frechet	8.932	1.634	3.899	None
16	Log-logistic	3	1	2	None	0.33	0.33	0.33	Log-logistic	2.663	0.871	2.221	None
17	Log-logistic	3	1	2	None	1	0	0	Cauchy	2.990	0.358	None	None
18	Log-logistic	3	1	2	None	0	1	0	Log-logistic	1.825	0.643	2.307	None
19	Log-logistic	3	1	2	None	0	0	1	Gaussian	3.126	0.728	None	None
20	Log-logistic	3	1	2	None	0.5	0.5	0	Cauchy	3.076	0.319	None	None
21	Burr12	10	4	1	0	0.33	0.33	0.33	Burr12	8.307	2.307	0.621	0.301
22	Burr12	10	4	1	0	1	0	0	Cauchy	0.849	0.090	None	None
23	Burr12	10	4	1	0	0	1	0	Cauchy	0.828	0.074	None	None
24	Burr12	10	4	1	0	0	0	1	Cauchy	0.844	0.066	None	None
25	Burr12	10	4	1	0	0.5	0.5	0	Cauchy	0.866	0.116	None	None
26	Dagum	7	3	1	2	0.33	0.33	0.33	Dagum	6.407	2.957	0.989	1.978
27	Dagum	7	3	1	2	1	0	0	Frechet	5.902	1.083	2.077	None
28	Dagum	7	3	1	2	0	1	0	Cauchy	3.246	0.131	None	None
29	Dagum	7	3	1	2	0	0	1	Log-logistic	4.180	0.750	2.455	None
30	Dagum	7	3	1	2	0.5	0.5	0	Frechet	5.426	1.022	2.130	None

Table 5. Comparison of results for different weighting factors

Table 5 showcases 30 experiments ran with different values for the weighting factors. Rows number 1, 6, 11, 16, 21 and 26 are the experiments presented in Section 4. It can be seen that most of the other experiments end up in overfitting the input data by evaluating the Cauchy distribution as the most viable solution.