Evolutionary algorithms for modeling non-equilibrium population

Muhammad Fuad, M. M.

Original citation & hyperlink:
https://dx.doi.org/10.1016/j.asoc.2021.108172

DOI 10.1016/j.asoc.2021.108172
ISSN 1568-4946

Publisher: Elsevier

© 2021, Elsevier. Licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International
http://creativecommons.org/licenses/by-nc-nd/4.0/

Copyright © and Moral Rights are retained by the author(s) and/ or other copyright owners. A copy can be downloaded for personal non-commercial research or study, without prior permission or charge. This item cannot be reproduced or quoted extensively from without first obtaining permission in writing from the copyright holder(s). The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the copyright holders.

This document is the author’s post-print version, incorporating any revisions agreed during the peer-review process. Some differences between the published version and this version may remain and you are advised to consult the published version if you wish to cite from it.
Evolutionary Algorithms for Modeling Non-equilibrium Population

Muhammad Marwan Muhammad Fuad

Coventry University
Coventry CV1 5FB, UK
ad02638@coventry.ac.uk

Abstract: During protein synthesis the genetic code links each codon, a triplet of nucleotides, with the corresponding amino acid. Synonymous codons are those that code for the same amino acid. The difference in the frequency of occurrence of certain synonymous codons over other synonymous codons is called the codon usage bias (CUB). The Zeng and Charlesworth model is used to estimate the strength of CUB. In their model the evolutionary process is represented by a Markov model, which allows the population size to vary over time. In this paper we propose a new method that incorporates demographic changes into the model. The method is a hybrid of two optimizers, the first is evolutionary programming and the second is a version of the genetic algorithms that uses chromosomes of variable lengths, which allows for expressing more demographic changes than what the simplified model presented by Zeng and Charlesworth does. We conduct several simulations to show why this hybridization is necessary, and also to show the superior performance of this new hybrid.

Keywords: Evolutionary programming, genetic algorithm, non-equilibrium population, population genetics, variable-chromosome-length.

1 Introduction

A DNA is a chain of molecules called nucleotides. A codon is a triplet of nucleotides. The genetic code links each codon with the corresponding amino acid in the process that synthesizes proteins. There are 64 different codons that code for the 20 amino acids that exist in nature, so different codons can code for the same amino acid. This is known as the degeneracy of the genetic code [1].

Codons can change via an operation called mutation. Codon changes can be divided into two types; non-synonymous codons, which result in a change in the protein sequence, and synonymous codons, which change the DNA sequence, but not the protein sequence. Synonymous nucleotide changes that do not affect the protein are generally assumed to be neutral [2]. Codon usage bias (CUB) is, by definition, the difference in the frequency of occurrence of certain synonymous codons over other
synonymous codons in coding DNA [3], so, when studying CUB, synonymous codons are divided into preferred and unpreferred codons.

The Li-Bulmer model [4], [5], is the standard and most widely accepted model for estimating the strength of CUB. This model assumes that there is a balance between mutation, selection, and random genetic drift.

From a population genetics perspective, the extent of codon usage bias is in fact an effect of selection, mutation, genetic drift, recombination, and demographic history [6]

There have been several attempts to incorporate simple demographic changes into the Li-Bulmer model. One of these attempts was introduced by Zeng and Charlesworth in [6], [7], where the evolutionary process is represented by a Markov model, which allows the population size to vary over time. The authors show that models that do not consider demographic effects such as changes in the population size give biased estimates of CUB.

In a previous paper [8], we proposed a method to incorporate demographic changes into the Zeng and Charlesworth model. The method we proposed is based on a variation of the genetic algorithms that we developed for this purpose. This variation uses chromosomes of variable lengths, which allows for expressing more demographic changes than what the simple model presented by Zeng and Charlesworth does. We validated our method using simulated data.

In this paper we enhance the performance of the method we presented in [8] by hybridizing the genetic algorithm with another optimizer; evolutionary programming. We show why this hybridization is necessary, and why we have chosen this optimizer in particular. As we did in [8], we validate the new hybrid method on simulated data.

The rest of the paper is organized as follows: Section 2 is a background section, where we present the problem and related work. Our new method is introduced in Section 3 and validated experimentally in Section 4. We conclude and discuss our findings in Section 5.

2 Background

A chromosome is a complex three-dimensional structure of DNA nucleotides. A particular location in a chromosome that is polymorphic is called locus. Polymorphic means that the data at this locus can have more than one variant [9]. The different variants at a locus are called alleles [10]. Changes in allele frequencies are caused by mutation, random genetic drift, and selection [11].

Population genetics is a branch of biology that seeks to understand how and why the frequencies of alleles and genotypes change over time within a population and
between populations. It is also concerned with the origin, amount, and distribution in space and time of the genetic variation, as well as the evolutionary factors that explain this variation. Population genetics seeks to provide insights into the different evolutionary processes such as mutation, natural selection, migration of individuals from one population to another, and genetic drift, which refers to major changes in population size and composition [12] [13] [14].

As mentioned above, polymorphic sites can be classified into two classes: synonymous and nonsynonymous. In a wide variety of organisms the frequencies with which such synonymous codons occur are nonrandom [15]. This means that those synonymous codons, are not used uniformly as some of them are used preferentially to others, dividing the synonymous codons into preferred and unpreferred codons (codon usage bias), which involves the disproportionate use of certain codons among the set of codons that code for a given amino acid [16].

The Li-Bulmer model [4], [5] proposes that the observed proportion of preferred codons can be explained by the balance of mutation, selection (in favor of preferred codons), and random genetic drift. The Li-Bulmer model assumes a constant selection coefficient per codon or codon preference group [6].

There has been, however, evidence against the Li-Bulmer model [6], [17], [18], [19]. Experiments also show that the Li-Bulmer model oversimplifies CUB as it ignores some factors such as the population demography, recombinational landscape, also the mutational process may have changed significantly over the evolution of species. Thus they may cause deviation from equilibrium [6]. Population genetics studies have shown the importance of incorporating non-equilibrium factors when studying CUB.

In [6] [7], Zeng and Charlesworth introduced a model that incorporates simple demographic changes. Their method is based on constructing a matrix model that includes the effects of a recent change in population size on estimates of selection of preferred versus unpreferred codons. The evolutionary process is represented by a Markov model. This representation allows the population size to vary over time. The experiments they conducted suggest that patterns of synonymous polymorphisms affecting codon usage can be quite erratic after such a change, so statistical methods that do not take demographic effects into consideration can give incorrect estimates of important parameters. By applying their method, Zeng and Charlesworth show that models that do not consider demographic effects such as changes in the population size give biased estimates of CUB.
Fig. 1. An example of a frequency spectrum. At more than 50% of positions all 20 individuals in the sample have the preferred allele and for 20% of positions all individuals have the unpreferred allele.

More specifically, the extent of CUB from a population genetics perspective is a product of the joint effects of mutation, selection, genetic drift, recombination, and demographic history [6]. It can be estimated by looking at all the codons that encode a specific amino acid and count the frequency distribution of the preferred codons versus unpreferred codons in a sample of individuals. Fig. 1 shows an example of frequency spectrum for a number of preferred alleles equal to 20. As we can see in the figure, the frequency spectrum is skewed in favor of the preferred codon. The data in this figure has been generated by a simulator.

Frequency distribution is dependent of several factors such as selection effect and mutation bias. It is also very dependent of population size [6]. If equilibrium in a population is disturbed by, for instance, a large change in the population size, it often takes a long time for the population to return to a new equilibrium state [20]. In consequence, the current frequency spectrum in a population does not depend only on the current size of the population but also on the historic size of the population, which is what we refer to as the demography of the population [6].

Zeng and Charlesworth show that in an equilibrium population three parameters are enough to calculate the expected frequency spectrum, these are the mutation rate, θ, the selection coefficient, γ, and the mutation bias, κ. Their model assumes that the population is initially in an equilibrium state and then it allows a single change in population size away from equilibrium. This extension adds two new parameters to the equilibrium model; the size of population change and the timing of this population change, as shown in Fig. 2.
Zeng and Charlesworth also show, given the expected frequency spectrum, how to calculate the probability of seeing a specific frequency distribution if a number of individuals are sampled from the population and tested. Consequently, the values of $\gamma$, $\theta$, and $\kappa$ and the demography can be estimated using the maximum likelihood method, which is a popular method in population genetics. The basis of this method is that the probabilities of different outcomes $X$ are conditional on the parameters $\varphi$ controlling the phenomenon. In probability theory language, this is expressed as a conditional probability $\Pr(X|\varphi)$ [21]. However, in many cases this is expressed as an inverse problem, i.e., given observed data $D$, we search for the optimal values for $\varphi$. This is achieved by looking for the values for $\varphi$ that maximize the likelihood. i.e. $L(\varphi|D) = \Pr(X|\varphi)$. Usually, the logarithm of the likelihood is used instead of the likelihood itself.

The model presented in [6] [7] is restricted to a simple demographic scenario where the population is first in equilibrium with population size $N_a$ and then changes instantly to $N_d$ where it remains for $t$ generations at which point a sample is taken. However, the formulas they describe can be extended to calculate the likelihood of an arbitrarily complex demography, which is what we did in [8], where we proposed a method that extends the model presented in [6] [7] by allowing for complex demographies with many changes in the population size. This adds much more complexity to the optimization problem.

In [6] [7], Zeng and Charlesworth use the simplex optimization method as presented in [22] with multiple start points. Clearly, this optimizer is too simple to apply to complex optimization problems. In fact the authors themselves report that this optimizer failed to converge and returned unrealistic results [6]. In addition, this optimizer can be applied to handle parameters with continuous values, like those for $\gamma$, $\theta$, and $\kappa$, but it is not appropriate to handle integer values, like the generation numbers or the population size, so it is not suitable for the generalization model that we are proposing. Besides, the number of parameters to fit in the generalization is not fixed, which is an obstacle the simplex optimization method cannot deal with. For all these reasons, we need an optimizer with a flexible encoding scheme to deal with the generalization we propose.
Consequently, we had to use another, more powerful optimizer for the generalization we are proposing. The optimizer we chose is a variation of the popular genetic algorithms that uses chromosomes of different lengths.

Genetic algorithm (GA) is one of the most prominent global optimization algorithms. Classic GA starts by randomly generating a population of chromosomes that represent possible solutions to the problem at hand. Each chromosome is a vector whose length is equal to the number of parameters. The fitness function of each chromosome is evaluated in order to determine the chromosomes that are fit enough to survive and possibly mate. In the selection step a percentage of chromosomes is selected for mating. Crossover is the next step in which the offspring of two parents are produced to enrich the population with fitter chromosomes. Mutation, which is a random alteration of a certain percentage of chromosomes, enables GA to explore the search space. In the next generation the fitness function of the offspring is calculated and the above steps repeat for a number of generations.

Although the version of GA presented above is widely used, it is not suitable for our optimization problem, because in our problem we do not know a priori the shape of optimal demography we are looking for, so we cannot use a predefined length of the chromosome for the optimization process. What we need is a representation of the solutions that uses chromosomes of variable lengths. This variation of GA is called variable-chromosome-length genetic algorithms, which has the same step as classic GA described above, except that it uses chromosomes of variable lengths.

Using chromosomes of variable lengths to solve optimization problems was first proposed in [23] as a variant of classifier systems. The same concept was applied later to solve optimization problems where the number of parameters is not fixed. In [24] the authors also apply a variation of the genetic algorithms with chromosomes of variable lengths to structural topology optimization. This variation was based on a progressive refinement strategy. It starts with a short chromosome-length and first finds an optimal solution in the simple design space. This optimal solution is then transferred to the next stages with longer chromosomes to get more elaborate solutions. In [25] the authors presented a genetic planner method that uses a genetic algorithm with chromosomes of variable-lengths. This variation applies a particular genetic scheme (complex fitness function, multi-population, population reset, weak mementism, tournament selection and elitist genetic operators). In [26] the authors use a genetic algorithm with variable-length chromosomes for path optimization problems, where they present a new crossover mechanism. In [27] a genetic algorithm with variable length chromosomes is applied to network intrusion detection systems, where each chromosome represents one or more rules. The authors of [28] also use a variable-length chromosome genetic algorithm to solve a road traffic coordination multipath problem, where they design specific selection, crossover and mutation operations.

We developed our own variation of the genetic algorithm that uses chromosomes of variable lengths which is particularly suitable for the problem we discussed earlier.
We call it *Variable-Chromosome-Length Genetic Algorithm* (VCL\_GA). This variation has its own genetic operations that we will present in the next section.

Using chromosomes of variable lengths allows for handling arbitrarily complex demographics. It is suitable for optimizing parameters with integer values, such as the population size and the number of generations. Besides, and this is the main advantage of the variation of genetic algorithms that we developed; since we do not know beforehand how many times the population size will change, and thus cannot tell in advance how many parameters the optimization problem will have, this variation we developed can optimize an arbitrary number of parameters.

### 3 A Hybrid of Variable-Chromosome-Length Genetic Algorithm and Evolutionary Programming

#### 3.1 Variable-Chromosome-Length Genetic Algorithm (VCL\_GA)

In the following we present the different elements of VCL\_GA:

**1-Encoding:** A chromosome $\text{chrom}^i$ is a vector of $\text{nbp}^{(i)}$ tuples defined as:

$$\text{chrom}^i = \left\{ (N_j^i, t_j^i), (N_j^i, t_j^i), \ldots, \left( N_{\text{nbp}^{(i)}}, t_{\text{nbp}^{(i)}} \right) \right\}$$

where both $N_j^i$ and $t_j^i$ are positive integers, and where $1 < j < \text{nbp}^{(i)}$, $t_j^i < t_{j+1}^i < \cdots < t_{\text{nbp}^{(i)}}$, and where $N_{\text{min}} < N_j^i < N_{\text{max}}$ ($N_{\text{min}}$ and $N_{\text{max}}$ are defined by the user). Whereas the value of $\text{nbp}^{(i)}$ is variable in our model, we constraint it to $\text{min}_{\text{nbp}} < \text{nbp}^{(i)} < \text{max}_{\text{nbp}}$ so that the optimization algorithm will converge. In Fig. 3 we show an example of the chromosomes we use in our method.

$$[(47, 7), (35, 12), (21, 15), (40, 16), (14, 20)]$$

![Diagram](image.png)

**Fig. 3.** Example of a chromosome in our VCL\_GA
2-Selection: This operation is performed as follows: of all the chromosomes in the population we create a mating pool whose size is matePoolSize. After each generation we discard matePoolSize/2 chromosomes with the lowest fitness value and replace them with offspring resulting from mating parent chromosomes in the mating pool. We adopt a weighted random pairing selection method, where the chance for a chromosome to be selected for mating is proportional to its fitness.

3-Crossover: The purpose of this operation in evolutionary algorithms that adopt it is to produce offspring that combine the superior traits of the parent chromosomes of that offspring. For our algorithm we propose the following crossover operation: an integer \( cp \) that satisfies \( \min_{nbp} < cp < \max_{nbp} \) is sampled from a uniform distribution. We then project \( cp \) on the two parent chromosomes. The first offspring is composed of the left part of the first parent chromosome and the right part of the second parent chromosome. The second offspring is composed of the left part of the second parent chromosome and the right part of the first parent chromosome. This is expressed formally as follows; let:

\[
\text{chrom}^l = [(N_1^l, t_1^l), (N_2^l, t_2^l), ..., (N_p^l, t_p^l), (N_{p+1}^l, t_{p+1}^l), ..., (N_{nbp}^l, t_{nbp}^l)]
\]

be the first parent chromosome and

\[
\text{chrom}^l = [(N_1^l, t_1^l), (N_2^l, t_2^l), ..., (N_q^l, t_q^l), (N_{q+1}^l, t_{q+1}^l), ..., (N_{nbp}^l, t_{nbp}^l)]
\]

be the second parent chromosome. Suppose that the crossover point is \( cp \) where \( p, q \leq cp \leq p + 1, q + 1 \). This crossover point splits the two parent chromosomes into:

\[
\text{chrom}^l_{\text{left}} = [(N_1^l, t_1^l), (N_2^l, t_2^l), ..., (N_p^l, cp)]
\]

and

\[
\text{chrom}^l_{\text{right}} = [(N_{p+1}^l, t_{p+1}^l), ..., (N_{nbp}^l, t_{nbp}^l)]
\]

Notice that \( cp \) may have introduced a new \( t \) point in \( \text{chrom}^l \) (i.e. \( p < cp < p + 1 \)). In this case \( t_{cp}^{p+1} \) takes the value of \( t \) after \( \text{chrom}^l \) has been split, given that \( t \) represents the endpoint of a segment. However, the value of \( t_{cp}^{p+1} \) remains the same after the split.

In the same manner, this crossover point splits \( \text{chrom}^l \) into (if \( q < cp < q + 1 \)):

\[
\text{chrom}^l_{\text{left}} = [(N_1^l, t_1^l), (N_2^l, t_2^l), ..., (N_q^l, cp)]
\]

and

\[
\text{chrom}^l_{\text{right}} = [(N_{q+1}^l, t_{q+1}^l), ..., (N_{nbp}^l, t_{nbp}^l)]
\]

The resulting offspring are:
\[ \text{offspring}_1 = [(N_1^i, t_1^i), (N_2^i, t_2^i), \ldots, (N_p^i, t_p^i), (N_{p+1}^i, t_{p+1}^i), \ldots, (N_{n_{nbp}}^i, t_{n_{nbp}}^i)] \]

and

\[ \text{offspring}_2 = [(N_1^i, t_1^i), (N_2^i, t_2^i), \ldots, (N_p^i, t_p^i), (N_{p+1}^i, t_{p+1}^i), \ldots, (N_{n_{nbp}}^i, t_{n_{nbp}}^i)] \]  

The crossover operation as described above may generate offspring that violate feasibility. For this reason, a feasibility check is performed after each crossover to verify that the lengths of the two produced offspring are within the boundaries of \( \min_{n_{nbp}} \) and \( \max_{n_{nbp}} \). If this condition is not satisfied, a new random value \( cp \) is generated and the above steps are repeated.

4-Mutation: This is an important operation because our method strongly depends on it, as our method requires new mutation operators designed specifically for it. In fact, a major part of developing our method was devoted to designing the appropriate mutation operations for the problem.

We apply two categories of mutation operations; weak mutation, in which the original length of the mutated chromosome remains the same after mutation, and strong mutation, in which the length of the mutated chromosome decreases or increases as a result of the mutation operation.

a-Weak mutation: We refer to this operation by \text{mut}. It is applied to \text{mutRate} chromosomes in \text{pop}. \text{mut} is performed on a mutated chromosome \text{chrom}\text{'}\text{,} which we choose by sampling an integer from the population, and which has \text{nbp}\text{'}\text{ tuples by randomly generating, from a uniform distribution, an integer randInt}^{(i)} \text{ that satisfies: } 1 \leq \text{randInt}^{(i)} \leq \text{nbp}\text{'}\text{. The chosen tuple} (N_{\text{randInt}}^{(i)}, t_{\text{randInt}}^{(i)}) \text{ is altered as follows: the value of} N_{\text{randInt}}^{(i)} \text{ is altered by generating a random integer randInt}_N^{(i)} \text{ that satisfies} \min_{n_{nbp}} \leq \text{randInt}_N^{(i)} \leq \text{max}_{n_{nbp}}. \text{The value of} t_{\text{randInt}}^{(i)} \text{ is altered by generating a random integer randInt}_t^{(i)} \text{ that satisfies} t_{\text{randInt}}^{(i)} < \text{randInt}_t^{(i)} < t_{\text{randInt}}^{(i+1)}. \text{In the case where randInt}^{(i)} = 1 \text{ or randInt}^{(i)} = \text{nbp}\text{' we sample the range of randInt}_t^{(i)} \text{ from the range:} \min_{n_{nbp}} \leq \text{randInt}_t^{(i)} < \text{randInt}_t^{(i+1)} \text{ and} \text{randInt}_t^{(i)} < \text{randInt}_t^{(i) + 1} \leq \text{max}_{n_{nbp}} \text{, respectively.}

To give an illustrating example of \text{mut}; say we have the following chromosome: [(12,3), (6,5), (23,10), (42,16), (10,20)] (taking \( \min_{n_{nbp}} = 4 \), \( \max_{n_{nbp}} = 50 \), \( \min_{n_{nbp}} = 2 \), \( \max_{n_{nbp}} = 20 \) and suppose that randInt\text{'}\text{.} = 3. The chosen tuple for \text{mut} is thus (23,10). 23 in this case can be mutated to any integer value that satisfies \( \min_{n_{nbp}} \leq \text{randInt}_t^{(i)} \leq \text{max}_{n_{nbp}} \) whereas 10 can be mutated to any integer value that satisfies: \( 5 < \text{randInt}_t^{(i)} < 16. \)
Notice that this latter operation is not always feasible. For example, if we have \((12,3), (6,9), (23,10), (42,11), (10,20)\), and for the same value for \(\text{randInt}^{(i)}\) as above. In this case \(\text{randInt}^{(i)} = 10\) cannot be altered and we can only alter \(\text{randInt}^{(i)} = 23\) in this case.

b- Strong mutation: Our method applies two types of strong mutation, each either decreases or increases the number of tuples in a chromosome by one:

i. Split mutation: As is clear by its name, this operation, which we refer to by \(\text{splitMut}\), splits one of the tuples of the chromosome to which it is applied into two tuples, increasing the number of tuples of that chromosome by one. \(\text{splitMut}\) is applied to \(\text{splitRate}\) chromosomes in \(\text{pop}\) in each generation. \(\text{splitMut}\) is applied as follows: from a uniform distribution we sample a chromosome from \(\text{pop}\). For that chromosome \(\text{chrom}^{t}\) we generate a random integer \(\text{randInt}^{(i)}\), drawn from a uniform distribution to determine the tuple which will be split. Let that tuple be \(\left(N_{\text{randInt}^{(i)}}^{t}, t_{\text{randInt}^{(i)}}^{t}\right)\). The first step is to check the feasibility of splitting; i.e. the following condition should be valid:

\[
N_{\text{randInt}^{(i)}}^{t} - t_{\text{randInt}^{(i)}}^{t} > 1
\]  

(1)

If the condition is valid we randomly generate a split point \(sp\) that satisfies

\[
t_{\text{randInt}^{(i)}}^{t} < sp < N_{\text{randInt}^{(i)}}^{t}
\]

thus producing two tuples: \((a, sp), (b, t_{\text{randInt}^{(i)}}^{t})\).

The first components \(a, b\) of the newly produced tuples are determined as follows: we randomly choose one of the newly generated tuples, the original value of the tuple before splitting took place is \(N_{\text{randInt}^{(i)}}^{t}\). This value is altered by randomly sampling an integer value between \(N_{\text{randInt}^{(i)}}^{t} - \text{minRange} \) and \(N_{\text{randInt}^{(i)}}^{t} + \text{minRange}\), where \(\text{minRange} = \min\left\{N_{\text{randInt}^{(i)}}^{t} - \text{min}, N_{\text{max}} - N_{\text{randInt}^{(i)}}^{t}\right\}\).

(This condition is necessary for feasibility as we will show in the example). The new value \(N_{\text{randInt}^{(i)}}^{t}\) is attributed to one of the two newly generated tuples, which, as we indicated above, is chosen randomly. \(N_{\text{randInt}^{(i)}}^{t}\), the \(N\) value of the second split tuple, is calculated as: \(N_{\text{randInt}^{(i)}}^{t} = 2N_{\text{randInt}^{(i)}}^{t} - N_{\text{randInt}^{(i)}}^{t}\). So the final values for the newly produced tuples are

\[
\left(N_{\text{randInt}^{(i)}}^{t}, sp\right), \left(N_{\text{randInt}^{(i)}}^{t} + t_{\text{randInt}^{(i)}}^{t}\right)\]

(Assuming \(N_{\text{randInt}^{(i)}}^{t}\) was attributed to the first newly produced tuple).

The rationale behind performing \(\text{splitMut}\) in this manner is that while one of the \(N\) values of the two tuples resulting from \(\text{splitMut}\) is allowed to arbitrarily take any value (within the boundaries), the \(N\) value of the other tuple should take a specific value so that the average of the two \(N\) values after splitting is the original \(N\) of the split tuple.
As in the case with mut, certain restrictions are imposed if \( 1 = \text{randInt}(^j) \) or \( \text{randInt}(^j) = \text{nbp}(^j) \).

If condition (1) is not valid, i.e. \( t_{\text{randInt}}(^0) - t_{\text{randInt}}(^0-1) = 1 \), thus the tuple cannot be split. For example, the second tuple in \([(23,4), (16,5), (31,6), (17,20)]\), then splitRate is abandoned for this chromosome for this generation.

Notice also that applying splitMut to a chromosome chrom\(^i\) chosen for splitting is only feasible if \( \text{nbp}(^j) < \max_{\text{nbp}} \)

To illustrate the application of splitMut, let: \([(10,4), (16,8), (42,10), (31,12), (20,16), (10,20)]\) be the chromosome chosen for splitMut. Let the tuple chosen for splitting be \((20,16)\) (i.e. \( \text{randInt} = 5 \)). Condition (1) is valid \((16 - 12 > 1)\), so splitMut is feasible. We start by randomly generating \( sp \) in the range \( 12 < sp < 16 \). Say the generated value is \( sp = 13 \). So the new tuples resulting from splitMut are \((\overline{N}, 13)\) and \((\overline{N}, 16)\).

We randomly choose one of these tuples, say the first one. The value \( N = 20 \) is the one to be mutated, \( \minRange = \min[20 - 4, 50 - 20] = 16 \). The range from which we sample a random integer value for \( \overline{N} \) is thus \([20 - 16, 20 + 16]\). This condition will guarantee that the value of \( \overline{N} \) will be within boundaries. Let the value we sampled be 32, this will be the value of \( \overline{N} \). The value of \( \overline{N} \) is \( 2 \times 20 - 32 = 8 \). The new resulting chromosome after splitMut is applied is thus: \([(10,4), (16,8), (42,10), (31,12), (32,13), (8,16), (10,20)]\).

ii. Merge mutation: This operation, which we refer to by mergeMut, merges two neighboring tuples into one tuple. The operation decreases the number of tuples of the chromosome to which it is applied by one.

It is important to mention that mergeMut is not the “opposite” of splitMut, given the stochastic nature of these operations.

mergeMut is applied to mergeRate chromosomes in pop in each generation as follows: for a chromosome \( \text{chrom}^k \), sampled from a uniform distribution, we randomly draw an integer \( \text{randInt}^{(k)} \) which satisfies \( 1 \leq \text{randInt}^{(k)} \leq \text{nbp}^{(k)} - 1 \). The tuple \((\overline{N}_{\text{randInt}^{(k)}}, t_{\text{randInt}^{(k)}})\) is merged with tuple \((N_{\text{randInt}^{(k)+1}}, t_{\text{randInt}^{(k)+1}})\) to produce the new tuple: \((\overline{N}_{\text{randInt}^{(k)}}, t_{\text{randInt}^{(k)}})\) whose two components are defined as follows:

\[
\overline{N}_{\text{randInt}^{(k)}} = \left[ \frac{n_{\text{randInt}^{(k)+1}} + n_{\text{randInt}^{(k)+1}}}{2} \right]
\]

\[
t_{\text{randInt}^{(k)+1}} = t_{\text{randInt}^{(k)+1}}
\]
Less formally, the $t$ value of the merged tuples is the $t$ value of the second tuple. The rationale of this scheme is that $t$ represents the endpoint of a segment, so when the two neighboring tuples are merged, the two tuples form one long tuple, whose endpoint is the endpoint of the second original tuple.

The $N$ value of the merged tuples is the average of their two $N$ values rounded down to the nearest integer.

Notice also that \textit{mergeMut} can only be applied if $\min_{n_{bp}} < n_{bp}^{(k)}$. □

It is important to mention that, as indicated earlier, all the mutation operations above included sampling an integer from a uniform distribution.

3.2 Remarks on VCL\_GA

The preliminary experiments we conducted in [8] on the problem at hand, using VCL\_GA, gave rather satisfactory results. However, in extended, more complex experiments that we conducted later, the results were less satisfactory. On the other hand, these extensive experiments helped us gain more insight into the optimization problem in question, and consequently, we were able to spot the limitations of VCL\_GA applied to this problem. These can be summarized in the following remarks:

i. Although the genetic algorithm is a powerful optimizer, as it can quickly locate high performance regions of vast and complex search spaces, it has a weakness in that it is not well suited for fine-tuning solutions [29], [30].

ii. While conducting our extensive experiments, and examining the detailed results of every generation, we noticed how, in many cases, the objective function would stagnate for several generations then it would drop suddenly (the problem was handled as an minimization optimization problem, as is conventionally the case) before it stagnated again for several generations. It seemed like the optimization problem was “waiting for the right mutation”. This phenomenon was mainly clear in later generations. It is important to mention here that increasing the mutation rate, a solution we tried, did not eliminate this phenomenon. It seemed the algorithm needed to take the traditional path of GA, until the solutions matured enough, then it arrived at a stage where it depended mainly on mutation to improve the value of the objective function. This phenomenon (the sudden drop in the objective function) can be attributed to premature convergence and/or that the fitness landscape has many local minima. It could also be the result of the fitness function traversing a flat region in the fitness landscape.

iii. The following remark is in fact the most important one in regard to the performance of VCL\_GA. The problem we are optimization has, as indicated above, the following parameters: $\gamma$, $\theta$ and $\kappa$ - the original parameters in [6]
[7], and also the new parameters related to generalization (the population demography) that we propose. The optimization problem in our generalization is implemented by dividing it into two optimization problems; the first is the one that optimizes $\gamma$, $\theta$ and $\kappa$; i.e. fitting the $\gamma$, $\theta$ and $\kappa$ parameters given a demography, which uses the simplex optimization algorithm, and the second optimization problem is the one that optimizes the population demography, which uses VCL_GA as an optimizer, and whose objective function is the maximum likelihood. This latter optimization problem is the "generalization" part of the problem at hand. The whole process starts by calling the second optimization problem with initial values of the parameters $\gamma$, $\theta$ and $\kappa$, which, during the progress of the second optimization problem, VCL_GA uses to update the values of $\gamma$, $\theta$ and $\kappa$ by calling the first optimization problem, and this continues back and forth between the two optimization problems for a number of generations controlled by VCL_GA. Whereas this approach is suitable because of the different nature of the parameters $\gamma$, $\theta$ and $\kappa$ on the one hand, and the demography on the other hand, this approach should take into account that the objective function of the second optimization problem, which is the outcome of the first optimization problem, is not a deterministic one, i.e. VCL_GA could give different values of the objective function for the same chromosome (although the difference in the values is small). As such, the problem we are handling, if we are considering the first and the second optimization problems together, is in fact a coevolutionary one, as the fitness of individuals in one of these populations depends on the individuals of another population [31], or it is optimization in a dynamic environment, if we are considering the second optimization problem (the demography) only, which is actually the way we approached it in this work.

iv. It has been suggested by several researchers that metaheuristics give better results when a good share of domain knowledge is incorporated [32] [33] [34], so whereas our VCL_GA is a first step in this direction, hybridizing it with evolutionary programming, with its mutational capacities, is a step further in this direction, since, as we learned from our experiments, the optimization problem we are dealing with is sensitive to mutation.

3.3 A Hybrid of Genetic Algorithms and Evolutionary Programming

Hybridization of different optimization algorithms has been widely used to solve different optimization problems [35] [36] [37]. The main merit of hybridizing two optimization algorithms is that the resulting hybrid algorithm takes advantage of the strengths of the two algorithms, or it avoids their weaknesses. A key to hybridization is to analyze the two optimizers to get insight into their components [38].

One way to improve VCL_GA is to hybridize it with another optimizer. The optimizer we choose is Evolutionary Programming, but before we explain why we chose this optimizer we start by introducing evolutionary programming. It is
important to mention however that like most other evolutionary algorithms, evolutionary programming does not have a canonical form.

**Evolutionary Programming (EP):** is one of the members of the evolutionary algorithms. In EP, new offspring are obtained by perturbing the original individuals. This means all offspring for the next generation are generated in the neighborhood of current solutions \[39\]. In its most generic form, for a population \( \text{pop}_\text{total} \) whose size is \( \text{popSize} \), in each generation we mutate each individual to form a new extended population \( \text{pop}_\text{extended} \) whose size is \( 2 \times \text{popSize} \), and of this extended population, \( \text{popSize} \) individuals, based on their fitness, are selected and kept to form \( \text{pop}_\text{new} \), whereas the other \( \text{popSize} \) individuals are discarded \[40\] \[27\].

The reasons we chose to hybridize VCL_GA with EP are the following:

1. VCL_GA uses chromosomes of different lengths, which is its main characteristic. EP can comply with this characteristic easily.

2. EP emphasizes on mutation and does not use recombination \[41\]. This makes it particularly suitable for the hybrid we are searching for as we showed in (ii) in Section 3.2.

There are mainly two different ways to hybridize two optimization algorithms; the first is “sequential”, where the second optimizer starts where the first one ends. The second is “nested”, where the second optimizer executes within one run of the first optimizer. We apply our hybrid sequentially by first applying VCL_GA for a number of generations, then we apply EP for the rest of the optimization process.

Given remark (ii) in Section 3.2, our strategy is to start by applying VCL_GA to obtain the basin of attraction of the optimal solution and then to refine the solution using EP.

The first phase of our hybrid starts with an initial population for VCL_GA that evolves for a predetermined number of generations (which is one of the control parameters of our method). The final population produced by VCL_GA constitutes the initial population for EP. The second phase of our hybrid, the EP phase, is initialized, as we said above, with the final population of the VCL_GA phase. It then evolves for another predetermined number of generations, which in the general case is different from that of the VCL_GA phase. The final outcome of this second phase is the optimal solution to our problem.

As indicated earlier, EP mutates the parent chromosomes of the last population of VCL_GA. Because EP is based on mutating this last generation resulting from VCL_GA, it also generates chromosomes of different lengths (which is not the case with classic EP). For this reason, we call this version of EP that we use VCL_EP. The hybrid optimizer we introduce in this work is thus called VCL_GA_EP. As we will see later in this paper, we present and test two variations of VCL_GA_EP to solve the problem at hand.
In the following we present the different components of VCL_GA_EP. Some of them have already been presented when we introduced VCL_GA, so to avoid repetition, we will only refer to which components concern the two phases, and which concern only either of them, and if there is any difference in application for these components that concern the two phases.

1-**Encoding**: It is related to both the VCL_GA phase, and the VCL_EP phase and it is performed the same way as we explained before for the two phases.

2-**Selection**: This concerns the two phases. Selection is performed in two ways according to the variation of VCL_GA_EP we use, as we will show later. But in general, for the VCL_EP phase, in each generation new chromosomes are generated and then added to the chromosomes resulting from \( \text{gen}_{i-1} \), which temporarily produces a new mating pool. Of this pool, the chromosomes with the lowest value of the fitness function are discarded, the remaining chromosomes constitute the population of generation \( \text{gen}_i \). We choose \( \text{matePoolSize} \) to be the same for the two phases.

3-**Crossover**: This operation concerns the VCL_GA phase only.

4-**Mutation**: It is applied in the same way for both the VCL_GA and VCL_EP phases. The mutation rate could be different for the two phases or it could be the same. It is important to mention however that in the VCL_GA phase we sample an integer from a uniform distribution, whereas in the VCL_EP phase we sample from a normal distribution, which is how evolutionary programming is applied. For our

**Table 1. Summary of notations**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{chrom} )</td>
<td>chromosome ( i )</td>
</tr>
<tr>
<td>( \text{nbit} )</td>
<td>length of chromosome ( i )</td>
</tr>
<tr>
<td>( \text{pop} )</td>
<td>population</td>
</tr>
<tr>
<td>( \text{popSize} )</td>
<td>population size</td>
</tr>
<tr>
<td>( \text{matePoolSize} )</td>
<td>size of mating pool</td>
</tr>
<tr>
<td>( \text{mut} )</td>
<td>weak mutation</td>
</tr>
<tr>
<td>( \text{mutRate} )</td>
<td>mutation rate</td>
</tr>
<tr>
<td>( \text{splitMut} )</td>
<td>split mutation</td>
</tr>
<tr>
<td>( \text{splitRate} )</td>
<td>split mutation rate</td>
</tr>
<tr>
<td>( \text{mergeMut} )</td>
<td>merge mutation</td>
</tr>
<tr>
<td>( \text{mergeRate} )</td>
<td>merge mutation rate</td>
</tr>
<tr>
<td>( \text{VCL} )</td>
<td>variable-chromosome-length</td>
</tr>
<tr>
<td>( \text{GA} )</td>
<td>genetic algorithms</td>
</tr>
<tr>
<td>( \text{EP} )</td>
<td>evolutionary programming</td>
</tr>
<tr>
<td>( \text{nrGenGA} )</td>
<td>number of generations of the GA phase</td>
</tr>
<tr>
<td>( \text{nrGenEP} )</td>
<td>number of generations of the EP phase</td>
</tr>
</tbody>
</table>
problem, since we are dealing with integers, we sample from a discrete normal distribution.

Table 1 shows a summary of the notations we used in our algorithm.

3.4 The Different Hybrid Versions

Hybridizing VCL_GA and EP can be implemented in several ways. In this paper, we present two such hybrids; we refer to the first one as VCL_GA_EP1, and to the second one as VCL_GA_EP2.

VCL_GA_EP1: In this version, only the best chromosome $\text{chrom}^{\text{opt}}$ resulting from the VCL_GA phase is passed to the VCL_EP phase to generate $\text{popSize} - 1$ chromosomes, which, together with $\text{chrom}^{\text{opt}}$, will constitute the initial population of the VCL_EP phase. These $(\text{popSize} - 1)$ chromosomes are generated by mutating $\text{chrom}^{\text{opt}}$ $(\text{popSize} - 1)$ times. For each of these $(\text{popSize} - 1)$ chromosomes, a random integer, sampled uniformly from the set $\{0,1,2\}$, is chosen to determine if the chromosome in question will be generated by a weak mutation, split mutation, or merge mutation, respectively, of $\text{chrom}^{\text{opt}}$. The objective function corresponding to each chromosome is calculated and the chromosomes are ranked. The chromosome with the best value of the objective function survives to the next generation, whereas all the other chromosomes are discarded. This is repeated for a number of generations $\text{nrGenEP}$, where $\text{nrGenEP}$ refers to the number of generations of the second phase (The evolutionary programming phase) to differentiate it from the number of generations of the genetic algorithms phase $\text{nrGenGA}$.

VCL_GA_EP2: In this version each chromosome in the population is mutated in the same manner as above, i.e. an integer is sampled uniformly from $\{0,1,2\}$ to determine the kind of mutation performed. This temporarily creates a population, whose size is $2 \times \text{popSize}$. These $(2 \times \text{popSize})$ chromosomes are ranked according to the corresponding value of their objective function. Half of these chromosomes, with the best value of the objective function, are kept, while the others are discarded, rendering the population size back to its original value. This process is repeated for a number of generations $\text{nrGenEP}$.

4 Experimental Evaluation

We ran two main experiments; in the first one we ran three simulations to compare the performance of VCL_GA, VCL_GA_EP1, and VCL_GA_EP2. In the second one we tested VCL_GA_EP1, and VCL_GA_EP2 on three simulations of two spectra, which is a new experiment that we did not conduct in [8].
Table 2. The three simulations of the first experiment.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Demography</th>
<th>$\theta$</th>
<th>$\kappa$</th>
<th>$\gamma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation1</td>
<td>$[50,51,52,53,54,55,56,57,58,59]$</td>
<td>0.009</td>
<td>1.4</td>
<td>0.003</td>
</tr>
<tr>
<td>Simulation2</td>
<td>$[110,120,130,140,150,160,170,180,190,200]$</td>
<td>0.01</td>
<td>1.0</td>
<td>0.0075</td>
</tr>
<tr>
<td>Simulation3</td>
<td>$[210,220,230,240,250,260,270,280,290,300]$</td>
<td>0.0065</td>
<td>1.5</td>
<td>0.002</td>
</tr>
</tbody>
</table>

4.1 The First Experiment

We ran three simulations shown in Table 2 and Fig 4. For this first experiment, we meant to choose simulations that demonstrate more complex demographic changes than those in the experiments we conducted in [8] because the main motivation of our present work is to show how VCL_GA_EP1 and VCL_GA_EP2 are better at handling populations with complex demographic changes than VCL_GA.

![Fig. 4. The demographics of the three simulations of the first experiment.](image)

The objective of this experiment is to see which of VCL_GA, VCL_GA_EP1, and VCL_GA_EP2 can better model these simulations in terms of the changes in demography as they appear in the original simulations, and also in terms of estimating the values of $\gamma$, $\theta$, and $\kappa$. The fitness function for the optimization problem, as we indicated in Section 2, is the logarithm of the likelihood, which we aim to maximize. This however is transformed into a minimization problem as we mentioned earlier.

The aim of comparing each of VCL_GA_EP1 and VCL_GA_EP2 with VCL_GA is to show that the two former methods can obtain the same or better results than VCL_GA by using a smaller number of objective function evaluations $nrObjFunEval$, where this number is given by:

$$nrObjFunEval = nrGen \ast popSize$$

In this experiment, we run VCL_GA for a number of generations $nrGenGA = 500$. For each of VCL_GA_EP1 and VCL_GA_EP2, we tested two settings; in the first setting we ran the first phase for $nrGenGA = 100$ and the second phase also for $nrGenEP = 100$, so $nrObjFunEval$ requires 40% of $nrObjFunEval$ compared to that of VCL_GA. In the second setting we ran the first phase for $nrGenGA = 200$ and the second phase for $nrGenEP = 100$, so $nrObjFunEval$ requires 60% of
**nrObjFunEval** compared to that of **VCL_GA**. The reason for these two settings is that we believe that running the first phase of **VCL_GA_EP1** and **VCL_GA_EP2** for a small number of generations (for example **nrGenGA = 100**) is not enough and will likely result in premature convergence. We believe the second phase of **VCL_GA_EP1** and **VCL_GA_EP2** should not start until the solutions of the first phase have matured enough. We wanted to test this experimentally and we also wanted to set a reasonable minimum number of generations for running the first phase of **VCL_GA_EP1** and **VCL_GA_EP2**. This is the reason why we included the first setting above (i.e. **nrGenGA = 100** for the first phase and **nrGenEP = 100** for the second phase) in our experiments even though we had thought before conducting the experiment that terminating the GA phase early in **VCL_GA_EP1** and **VCL_GA_EP2**, would result in premature convergence.

We used the same values for the control parameters that we used in [8] which are **mutRate = 0.1, mergeRate = 0.1, splitRate = 0.2, popSize = 24, and matePoolSize = 12** (for GA). The other constraints were set to: **Nmin = 4, Nmax = 50, minnabp = 2, maxnabp = 8**. For each simulation, the three optimizers we are testing were run 10 times.

As we said above, the objective function is the logarithm of the likelihood, which the optimization algorithm aims to maximize. This is transformed into a minimization optimization problem, as we mentioned above. In Table 3 we present the values of the objective function that we obtained for the three compared methods: **VCL_GA for nrGenGA = 500, VCL_GA_EP1 for nrGenGA = 100 and nrGenEP = 100, VCL_GA_EP1 for nrGenGA = 200 and nrGenEP = 100, VCL_GA_EP2 for nrGenGA = 100 and nrGenEP = 100, and finally VCL_GA_EP2 for nrGenGA = 200, and nrGenEP = 100**. These methods were applied to Simulation1, Simulation2, and Simulation3. For brevity, we refer to that in the tables and figures hereafter as **GA_500, EP1_100_100, EP1_200_100, EP2_100_100, EP2_200_100**, respectively. As we said earlier, each method was run 10 times and we show in Table 3 the best results of the 10 runs. They are shown in bold, yellow-shaded, underlined printing. The best result for Simulation1 is that of **EP1_200_100**, followed by that of **EP2_200_100**, followed by that of **GA_500**, followed by **EP2_100_100**, followed by **EP1_100_100**. As for Simulation2, the best result is again that of **EP1_200_100**, followed by **EP1_100_100**, followed by **EP2_200_100**, followed by **EP2_100_100**, followed by **GA_500**. As for Simulation3, the best result is also that of **EP1_200_100**, followed by **GA_500**, followed by **EP2_200_100**, followed by **EP1_100_100**, followed by **EP2_200_100**. In order to compare the general performance of the five

![Table 3](image)

**Table 3.** The value of the objective functions (the logarithm of the likelihood) of the three compared methods: **GA_500, EP1_100_100, EP1_200_100, EP2_100_100, and EP2_200_100**, on Simulation1, Simulation2, and Simulation3.
methods on the three simulations, we give the best method on each simulation 5 points, the second best 4 points, etc. Then we sum the points each method obtained on the three simulations. By doing so, EP1_200_100 comes first as it obtained 15 points, EP2_200_100 comes second with 10 points, GA_500 comes third with 8 points, EPI_100_100 comes fourth with 7 points, and EP2_100_100 comes last with 5 points.

In Table 4 and Fig. 5 we show the values of the three parameters $\theta$, $\kappa$, $\gamma$ corresponding to the best results shown in Table 3, for GA_500, EPI_100_100, EP1_200_100, EP2_100_100, and EP2_200_100, on Simulation1, Simulation2, and Simulation3. Fig. 6. shows a boxplot of the parameters $\theta$, $\kappa$, $\gamma$ in the 10 runs for GA_500, EPI_100_100, EP1_200_100, EP2_100_100, and EP2_200_100. Interestingly, the best global performance is that of EPI_100_100, which has the best performance in 5 out of 9 estimations, followed by EP2_200_100, which has the best performance in 2 out of 9 estimations, followed by EP2_200_100, followed by both EP2_100_100 and EPI_100_100, each has the best performance in one estimation only. GA_500 comes last and was beaten by all other methods.

When interpreting these results, it is important to remember that the objective of our optimization problem is not to optimize the three parameters $\theta$, $\kappa$, $\gamma$, so these values are a “by-product” of our optimization problem.

**Table 4.** The estimation of the parameters $\theta$, $\kappa$, $\gamma$ of the corresponding best solutions for GA_500, EPI_100_100, EP1_200_100, EP2_100_100, and EP2_200_100, for Simulation1, Simulation2, and Simulation3

<table>
<thead>
<tr>
<th>Simulation</th>
<th>$\gamma$</th>
<th>$\kappa$</th>
<th>$\theta$</th>
<th>$\gamma$</th>
<th>$\kappa$</th>
<th>$\theta$</th>
<th>$\gamma$</th>
<th>$\kappa$</th>
<th>$\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation1</td>
<td>0.01032</td>
<td>0.00986</td>
<td>0.00984</td>
<td>0.00992</td>
<td>0.01003</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simulation2</td>
<td>1.40331</td>
<td>1.40263</td>
<td>1.40265</td>
<td>1.40273</td>
<td>1.40281</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simulation3</td>
<td>0.00427</td>
<td>0.00427</td>
<td>0.00427</td>
<td>0.00366</td>
<td>0.00520</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simulation2</td>
<td>0.01238</td>
<td>0.01212</td>
<td>0.01219</td>
<td>0.01213</td>
<td>0.01206</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simulation1</td>
<td>1.00482</td>
<td>1.00447</td>
<td>1.00462</td>
<td>1.00455</td>
<td>1.00449</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simulation3</td>
<td>0.00069</td>
<td>0.00020</td>
<td>0.00074</td>
<td>0.00064</td>
<td>0.01651</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simulation2</td>
<td>0.00054</td>
<td>0.00169</td>
<td>0.00114</td>
<td>0.00070</td>
<td>0.00095</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simulation3</td>
<td>1.49248</td>
<td>1.49242</td>
<td>1.49341</td>
<td>1.49276</td>
<td>1.49310</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simulation1</td>
<td>0.00314</td>
<td>0.00342</td>
<td>0.00418</td>
<td>0.00569</td>
<td>0.00424</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fig. 5. The values of the three parameters $\theta$, $\kappa$, $\gamma$ of Simulation1, Simulation2, and Simulation3, corresponding to the best results of the objective function of GA_500, EP1_100_100, EP1_200_100, EP2_100_100, and EP2_200_100, compared to the true values of these parameters.
Fig. 6. A boxplot of the parameters $\alpha$, $\kappa$, $\gamma$ in the 10 runs for GA_500, EP1_100_100, EP1_200_100, EP2_100_100, and EP2_200_100, for Simulation1, Simulation2, and Simulation3. The thick line in magenta shows the true value of the corresponding parameter.
When analyzing the results on the parameter level, we see that evaluating $\kappa$ was quite well, as most methods gave a very good approximation of this parameter. Evaluating $\gamma$ was good, although not as good as that of $\kappa$. Evaluating $\theta$ seems to be the farthest from the true value, compared with the other two parameters.

### 4.2 The Second Experiment

Whereas the purpose of the first experiment was to compare the performance of the hybrid method with that of VCL_GA, which uses the genetic algorithms only, the purpose of this second experiment is to compare the performance of VCL_GA_EP1 to that of VCL_GA_EP2, namely comparing EP1_400_100 with EP2_400_100, in a more complex optimization problem, which is that of demographics with two spectra.

![Simulations](image)

Fig. 7. The demographics of the three simulations of the second experiment.

As with the first experiment, we ran three simulations, which we show in Fig. 7. Table 5 shows the values of these demographics together with the corresponding values for the six parameters $\gamma_1$, $\gamma_2$, $\kappa_1$, $\kappa_2$, $\theta_1$, $\theta_2$. It is important to mention that although some of these simulations, as we can see in Table 5, have the same values for some parameters to reflect a more simplified simulation, EP1_400_100 or EP2_400_100 may not necessarily give the same estimation for these parameters, as each of these parameters is estimated independently, given that these simulations are completely invisible to both EP1_400_100 and EP2_400_100, which have no prior knowledge of these simulations, so even though each simulation has four distinguished values only for the parameters, the optimization algorithm will give estimations for six values for the parameters, because this information - that some parameters have the same value - is hidden from the optimization algorithm.

As with the first experiment, the objective of the second experiment is to compare EP1_400_100 and EP2_400_100 to find out which method can better model the demography and the parameters of these simulations.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Demography</th>
<th>$\gamma_1$</th>
<th>$\gamma_2$</th>
<th>$\kappa_1$</th>
<th>$\kappa_2$</th>
<th>$\theta_1$</th>
<th>$\theta_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation4</td>
<td>$[50,51],[65,20],[40,25],[0,45]$</td>
<td>0.01</td>
<td>0.005</td>
<td>1.5</td>
<td>1.5</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>Simulation5</td>
<td>$[10,15],[90,18],[0,25],[80,40]$</td>
<td>0.075</td>
<td>0.01</td>
<td>1.25</td>
<td>1.25</td>
<td>0.0075</td>
<td>0.0075</td>
</tr>
<tr>
<td>Simulation6</td>
<td>$[20,15],[70,25],[50,37],[150,45]$</td>
<td>0.005</td>
<td>0.075</td>
<td>1.0</td>
<td>1.0</td>
<td>0.005</td>
<td>0.005</td>
</tr>
</tbody>
</table>
The experimental settings of the second experiment are a bit different. In this second experiment we have \( nrGenGA = 400, \) \( nrGenEP = 100 \), whereas in the first one we had \( nrGenGA = 200, \) \( nrGenEP = 100, \) and \( nrGenGA = 100, \) \( nrGenEP = 100 \). There are two reasons for this choice; the first is that our focus in this second experiment is on performance, so we wanted to test our method using a higher number of generations compared with that of the first experiment, yet we made sure that the total number of generations is the same as that we used in \([8]\), which is \( nrGenGA = 500 \). The second reason is that when we developed this new hybrid method we believed that VCL\_GA\_EP1 and VCL\_GA\_EP2, will have a different performance if the solutions have a chance to mature enough in the first phase (the GA phase) of the optimization process, so we wanted to test this hypothesis.

We also conducted the second experiment using a wider range for some control parameters, so we tested \( \text{mutRate} = 0.1, 0.2, 0.3 \), \( \text{mergeRate} = 0.1, 0.2, 0.3 \), \( \text{splitRate} = 0.1, 0.2, 0.3 \). The other control parameters are the same as in the first experiment.

In Fig. 8 we show a box plot of the values of the objective functions of the 10 results we obtained for each of EP1\_400\_100 and EP2\_400\_100 on Simulation4, Simulation5, and Simulation6. As we can see, for Simulation4, the performance of EP2\_400\_100 clearly outperforms that of EP1\_400\_100. EP2\_400\_100 also outperforms EP1\_400\_100 for Simulation6, whereas EP1\_400\_100 outperforms EP2\_400\_100 for Simulation5. Interestingly, there are more outliers in EP1\_400\_100 than in EP2\_400\_100, which is actually related to the different ways these two optimizers behave, as EP1 focuses on tracking the best chromosome, whereas EP2 focuses on increasing the fitness of the whole population.

The results also show that while EP1\_200\_100 clearly outperformed EP2\_200\_100 in optimizing the logarithm of the likelihood in the first experiment, this is no longer the case in the second experiment where EP2\_400\_100 outperformed EP1\_400\_100 in 2 out of 3 simulations.

In Fig. 9 we present the values of the parameters \( \theta_1, \theta_2, \kappa_1, \kappa_2, \gamma_1, \gamma_2 \) that correspond to the 10 results of the objective function that we showed in Fig. 8. It is important to remember again that the objective function of our optimization problem is the logarithm of the likelihood, so the estimation of the parameters \( \theta_1, \theta_2, \kappa_1, \kappa_2, \gamma_1, \gamma_2 \) is a "by-product" of the optimization process.
Fig. 8. The values of the objective function (the logarithm of the likelihood) of the 10 results for each of EP1\_400\_100 and EP2\_400\_100 on Simulation4, Simulation5, and Simulation6
Fig. 9. The values of the parameters $\theta_1$, $\theta_2$, $\kappa_1$, $\kappa_2$, $\gamma_1$, $\gamma_2$ that correspond to the 10 results of the logarithm of the likelihood for each of EP1_400_100 and EP2_400_100 on Simulation4, Simulation5, and Simulation6.
In general the estimation of the parameters for the three simulations is good to very good for parameters ρ1, κ1, and κ2, and less satisfactory for parameters γ2, θ1, θ2. There does not seem to be a difference in performance between EP1_400_100 and EP2_400_100. The only exception is for the two parameters θ1, θ2 in Simulation6 where the estimations given by EP2_400_100 are clearly better than those given by EP1_400_100.

Finally, in Fig. 10 we show how well each of EP1_400_100 and EP2_400_100 can model the demographics of the three simulations. For each simulation we plot the demography associated with the best five solutions having the best value of the logarithm of the likelihood. Again, this demography is a by-product of the optimization problem. For each group, the original demography is shown in red at the top left corner, the top five solutions are shown in order from top to bottom, left to right in Fig. 10.

Fig. 10. The demographics of the best five solutions with the best value of the logarithm of the likelihood for Simulation4, Simulation5, and Simulation6 using EP1_400_100 and EP2_400_100. The original demography of each simulation is shown in red at the top left corner, the top five solutions are shown in order from top to bottom, left to right.
The results in Fig. 10 show that the demographies corresponding to the five best solutions of the optimization process can capture the main demographic changes of the original demography for each simulation. We do not see a difference in the performance of EP1_400_100 and EP2_400_100. Needless to say, our assessment for the results obtained for this part of the experiment is subjective, as it is based on a visual similarity between the demographies resulting from the optimization problem and the original demography.

We have to mention that the time axes of the figures resulting from the optimizers are scaled differently from those of the simulations themselves, because, as we mentioned earlier, the optimizers do not see the simulations, so the scale is different. The comparison between the demographies resulting from the optimizers, and their counterparts of the simulation, should be based on the changes in the demography (i.e. the general shape of the demography).

5 Discussion and Conclusion

The Zeng and Charlesworth model is the most popular model in population genetics to estimate the strength of codon usage bias. This model represents the evolutionary process by a Markov model, which allows the population size to vary over time. Their model, however, is restricted to a simple demographic scenario. In this paper we extended their model by allowing for complex demographies with several changes in the population size. This extension calls for a much more sophisticated optimizer than the one used in the original model. In this work we used a hybrid of genetic algorithms and evolutionary programming to incorporate more complex demographic changes into the model presented by Zeng and Charlesworth. We presented two versions of this hybrid; VCL_GA_EP1 and VCL_GA_EP2, which differ in their evolutionary programming phase. The genetic algorithm phase of the two versions is the same, and it uses chromosomes of variable lengths. We conducted two experiments to test the two versions of our hybrid on several simulations.

In the first experiment we compared the two hybrid versions VCL_GA_EP1 and VCL_GA_EP2, with the method that uses the genetic algorithms only, VCL_GA, on three simulations to see which of the three methods can better model these simulations. The results show that VCL_GA_EP1 and VCL_GA_EP2 can obtain the same, or even better, results than VCL_GA using a smaller number of objective function evaluations, with a clear advantage to VCL_GA_EP1 not only over VCL_GA, but also over VCL_GA_EP2. This first experiment also shows that the three methods give a good to a very good estimation of the parameters $\theta, \kappa, \gamma$, although this estimation is not the direct objective of the optimization problem.

In the second experiment we focused exclusively on the two hybrid versions VCL_GA_EP1 and VCL_GA_EP2, and we compared their performance on three simulations in a more complex setting than the first experiment as the demographies have two spectra, so we have six parameters to estimate. In this second experiment the
two optimizers, VCL_GA_EP1 and VCL_GA_EP2, have almost the same performance in optimizing the logarithm of the likelihood, which is the objective function of the optimization problem. The estimations of the six parameters varies from satisfactory to very good, although this estimation is a by-product of the optimization problem. We also showed in this second experiment how the solutions obtained by the optimizers, VCL_GA_EP1 and VCL_GA_EP2, can capture the main demographic changes of the original demography for each of the three tested simulations. There is no difference between the two hybrids. However, the evaluation of performance for the demographic changes is subjective and is based on visual assessment.

The two experiments give us insight into how the two hybrid methods work in terms of the recommended version to use, as the main difference between the two experiments, apart from using more complex demographies in the second experiment, is the change in nrGenGA compared with the first experiment. From what we see, it seems if the user’s main objective is to obtain close to optimal results, regardless of time and computational resources, then we recommend the use of VCL_GA_EP2, with a high number of nrGenGA (1000, for example) before moving to the EP phase, which should be run for a reasonable number of nrGenGA (200-300, for example). In the case where the objective is to give a reasonably “optimal” solution fast or under restricted resources, we recommend applying VCL_GA_EP1. Apart from the experimental results that support this, the rationale behind this is that VCL_GA_EP1 focusses on one chromosome, which is faster, but it could also result in local minima, whereas VCL_GA_EP2 attempts to improve the fitness of the whole population, which is time consuming, but will likely avoid getting trapped in local minima. It is also important to remember here that the two hybrids depend on mutation to a reasonable degree, so they are relatively immune from premature convergence.

Another point to mention concerning the difference between VCL_GA_EP1 and VCL_GA_EP2 is related to our remark (iii) in Section 3.2 that this optimization problem can be viewed as optimization in a dynamic environment, so VCL_GA_EP2 has an advantage over VCL_GA_EP1 as it keeps more diversity in the population, which is important when handling optimization in a dynamic environment [42].

There are different directions for future research, the first and most important is to extend the optimization problem to include, not only the logarithm of the likelihood, but also the parameters, and may be also the demographic changes, which will be more difficult.

In order to objectively evaluate the quality of solutions in terms of demographic changes, it is important to define appropriate similarity measures. We think that the similarity measures used in time series mining, such as those for discrete wavelet transforms [43] will be quite handy.

Although in this work we developed several new evolutionary operations particularly designed to handle this optimization problem. Extending the optimization problem to
include parameters and demographic changes may also require developing other specific evolutionary operations.

References


