Coronavirus Optimization Algorithm: A bioinspired metaheuristic based on the COVID-19 propagation model

Preprint · March 2020

CITATIONS
0

READS
3,901

9 authors, including:

Francisco Martínez-Álvarez
Universidad Pablo de Olavide
154 PUBLICATIONS **2,856** CITATIONS
SEE PROFILE

Gualberto Asencio Cortés
University of Pablo de Olavide
55 PUBLICATIONS **599** CITATIONS
SEE PROFILE

José Francisco Torres
Universidad Pablo de Olavide
17 PUBLICATIONS **359** CITATIONS
SEE PROFILE

David Gutiérrez-Avilés
Universidad de Sevilla
26 PUBLICATIONS **192** CITATIONS
SEE PROFILE

Coronavirus Optimization Algorithm: A bioinspired metaheuristic based on the COVID-19 propagation model

F. Martínez-Álvarez[1]*, G. Asencio-Cortés[1], J. F. Torres[1],
D. Gutiérrez-Avilés[1], L. Melgar-García[1], R. Pérez-Chacón[1],
C. Rubio-Escudero[2], J. C. Riquelme[2], A. Troncoso[1]

[1]Data Science & Big Data Lab, Pablo de Olavide University, ES-41013 Seville, Spain

[2]Department of Computer Science, University of Seville, ES-41012 Seville, Spain

Abstract

This work proposes a novel bioinspired metaheuristic, simulating how the coronavirus spreads and infects healthy people. From a primary infected individual (patient zero), the coronavirus rapidly infects new victims, creating large populations of infected people who will either die or spread infection. Relevant terms such as reinfection probability, super-spreading rate, social distancing measures or traveling rate are introduced into the model in order to simulate the coronavirus activity as accurately as possible. The infected population initially grows exponentially over time, but taking into consideration social isolation measures, the mortality rate and number of recoveries, the infected population gradually decreases. The Coronavirus Optimization Algorithm has two major advantages when compared to other similar strategies. Firstly, the input parameters are already set according to the disease statistics, preventing researchers from initializing them with arbitrary values. Secondly, the approach has the ability to end after several iterations, without setting this value either. Furthermore, a parallel multi-virus version is proposed, where several coronavirus strains evolve over time and explore wider search space areas in less iterations. Finally, the metaheuristic has been combined with deep learning models, in order to find optimal hyperparameters during the training phase. As application case, the problem of electricity load time series forecasting has been addressed, showing quite remarkable performance.

Keywords: Metaheuristics, soft computing, deep learning, big data, coronavirus.

*Corresponding author, fmaralv@upo.es

1 Introduction

The severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) is a new respiratory virus, causing coronavirus disease 2019 (COVID-19), firstly discovered in humans in December 2019, that has spread across the globe, having reportedly infected more than 4 million people so far [1]. Much remains unknown about the virus, including how many people who may have very mild, asymptomatic or simply undocumented infections and whether they can transmit the virus or not [2]. The precise dimensions of the outbreak are hard to evaluate [3].

Bioinspired models typically mimic behaviors from the nature and are known for their successful application in hybrid approaches to find parameters in machine learning model optimization [4]. Viruses can infect people and these people can either die, infect other people or simply recover after the disease. Vaccines and the immune defense system typically fight the disease and help to mitigate their effects while an individual is still infected. This behavior is typically modeled by an SIR model, consisting of three types of individual: S for the number of susceptible, I for the number of infectious, and R for the number of recovered [5].

Metaheuristics must deal with huge search spaces, even infinite for the continuous cases, and must find suboptimal solutions in reasonable execution times [6]. The rapid propagation of the coronavirus along with its ability to cause infection in most of the countries in the world impressively fast, has inspired the novel metaheuristic proposed in this work, named Coronavirus Optimization Algorithm (CVOA). A parallel version is also proposed in order to spread different coronavirus strains and achieve better results in less iterations.

The main CVOA advantages regarding other similar approaches can be summarized as follows:

1. Coronavirus statistics are not currently known with precision by the scientific community and some aspects are still controversial, like the reinfection rate [7]. In this sense, the infection rate, the mortality rate, the spreading rate or the reinfection probability cannot be accurately estimated so far, due to several issues like the lack of tests for asymptomatic people. However, the current state of the pandemic suggests certain values, as reported by the World Health Organization [8]. Therefore, CVOA is parametrized with the actual reported values for rates and probabilities, preventing the user from performing an additional study on the most suitable setup configuration.

2. CVOA can stop the solutions exploration after several iterations, with no need to be configured. That is, the number of infected people increases over the first iterations, however, after a certain number of iterations, the number of infected people starts decreasing, until reaching a void infected set of individuals.

3. The coronavirus high spreading rate is useful for exploring promising regions more thoroughly (intensification) while the use of parallel strains ensures that all regions of the search space are

evenly explored (diversification).

4. Another relevant contribution of this work is the proposal of a new discrete and of dynamic length codification, specifically designed for combining Long Short-Term Memory networks (LSTM) with CVOA (or any other metaheuristic).

There is one limitation to the current approach. Since there is no vaccine currently, it has not been included in the procedure to reduce the number of candidates to be infected. This fact involves an exponential increase of the infected population in the first iterations, and therefore an exponential increase of the execution time for such iterations. This, however, is partially solved with the implementation of social isolation measures to simulate individuals who cannot be infected during a particular iteration.

A study case is included in this work which discusses the CVOA performance. CVOA has been used to find the optimal values for the hyperparameters of a LSTM architecture [9], which is a widely used model for artificial recurrent neural network (RNN), in the field of deep learning [10]. Data from the Spanish electricity consumption have been used to validate the accuracy. The results achieved verge on 0.45%, substantially outperforming other well-established methods such as random forest, gradient-boost trees, linear regression or deep learning optimized with other metaheuristics. The code, developed in Phyton with a discrete codification, is available in the supplementary material (along with an academic version in Java for a binary codification).

Finally, the need to further study the performance of well-established fitness functions [11] is acknowledged. However, given the relevance that this pandemic is acquiring throughout the world and the remarkable results achieved when combined with deep learning, this work is shared with the hope that it inspires future research in this direction.

The rest of the paper is organized as follows. Section 2 discusses related and recent works. The methodology proposed is introduced in Section 3. Section 4 proposes a discrete codification to hybridize deep learning models with CVOA and provides some illustrative cases. A sensitivity analysis on how populations are created and evolved over time is discussed in Section 5. The results achieved are reported and discussed in Section 6. Finally, the conclusions drawn and future work suggestions are included in Section 7.

2 Related works

There are many bioinspired metaheuristics to solve optimization problems. Although CVOA has been conceived to optimize any kind of problems, this section focuses on optimization algorithms applied to hybridize deep learning models.

It is hard to find consensus among the researchers on which method should be applied to which problem, and, for this reason, many optimization methods have been proposed during the last decade to

improve deep learning models. Generally, the criterion for selecting a method is its associated performance from a wide variety of perspectives. Low computation cost, accuracy or even implementation difficulty can be accepted as one of these criteria.

The Virus Optimization Algorithm was proposed by Liang and Cuevas-Juárez in 2016 [12] and later improved in [13]. However, as many other metaheuristics, the results of its application are highly dependent on its initial configuration. Additionally, it simulates generic viruses, without adding individualized properties for particular viruses. The results achieved indicate that its usefulness is beyond doubt.

One of the most extended metaheuristics used to improve deep learning parameters is genetic algorithms (GA). Hence, a LSTM network optimized with GA can be found in [14]. To evaluate the proposed hybrid approach, the daily Korea Stock Price Index data were used, outperforming the benchmark model. In 2019, a network traffic prediction model based on LSTM and GA was proposed in [15]. The results were compared to pure LSTM and ARIMA, reporting higher accuracy.

Multi-agents systems have also been applied to optimize deep learning models. The use of Particle Swarm Optimization (PSO) can be found in [16]. The authors proposed a model based on kernel principal component analysis and back propagation neural network with PSO for midterm power load forecasting. The hybridization of deep learning models with PSO was also explored in [17] but, this time, the authors applied the methodology with image classification purposes.

Ants colony optimization (ACO) models have also been used to hybridize deep learning. Thus, Desell et al. [18] proposed an evolving deep recurrent neural networks using ACO applied to the challenging task of predicting general aviation flight data. The work in [19] introduced a method based on ACO to optimize a LSTM recurrent neural networks. Again, the field of application was flight data records obtained from an airline containing flights that suffered from excessive vibration.

Some papers exploring the Cuckoo Search (CS) properties have been published recently as well. In [20], CS was used to find suitable heuristics for adjusting the hyper-parameters of another LSTM network. The authors claimed an accuracy superior to 96% for all the datasets examined. Nawi et al. [21] proposed the use of CS to improve the training of RNN in order to achieve fast convergence and high accuracy. Results obtained outperformed those than other metaheuristics.

The use of the artificial bee colony (ABC) optimization algorithm applied to LSTM can also be found in the literature. Hence, and optimized LSTM with ABC to forecast the bitcoin price was introduced in [22]. The combination of ABC and RNN was also proposed in [23] for traffic volume forecasting. This time the results were compared to standard backpropagation models.

From the analysis of these works, it can be concluded that there is an increasing interest in using metaheuristics in LSTM models. However, not as many works as for artificial neural networks can be found in the literature and, none of them, based on a virus propagation model. These two facts, among others, justify the application of CVOA to optimize LSTM models.

3 Methodology

This section introduces the CVOA methodology. Thus, Section 3.1 describes the steps for a single strain. Section 3.2 introduces the modifications added to use CVOA as a parallel version. Section 3.3 suggests how the input parameters must be set. Section 3.4 includes the CVOA pseudo codes.

3.1 Steps

Step 1. Generation of the initial population. The initial population consists of one individual, the so-called patient-zero (PZ). As in the coronavirus pandemic, it identifies the first human being infected. If no previous local minima have been found, a random initialization for the PZ is suggested.

Step 2. Disease propagation. Depending on the individual, several cases are evaluated:

1. Each infected individual has a probability of dying (P_DIE), according to the COVID-19 death rate. Such individuals cannot spread the disease to new ones.

2. The individuals who do not die, will cause infection to new individuals (intensification). Two types of spreading are considered, according to a given probability ($P\_SUPERSPREADER$):

   (a) Ordinary spreaders. Infected individuals will infect new ones according to a regular spreading rate ($SPREADING\_RATE$).

   (b) Super-spreaders. Infected individuals will infect new ones according to a super-spreading rate ($SUPERSPREADING\_RATE$).

3. There is another consideration, since it is needed to ensure diversification. Both ordinary and super-spreaders individuals can travel and explore solutions quite dissimilar. Therefore, individuals have a probability of traveling (P_TRAVEL) to propagate the disease to solutions that may be quite different ($TRAVELER\_RATE$). In case of not being traveler, new solutions will change according to an $ORDINARY\_RATE$. Note that one individual can be both super-spreader and traveler.

Step 3. Updating populations. Three populations are maintained and updated for each generation.

1. Deaths. If any individual dies, it is added to this population and can never be used again.

2. Recovered population. After each iteration, infected individuals (after spreading the coronavirus according to the previous step) are sent to the recovered population. It is known that there is a reinfection probability ($P\_REINFECTION$). Hence, an individual belonging to this population could be reinfected at any iteration provided that it meets the reinfection criterion. Another situation must be considered, since individuals can be isolated simulating they are implementing the social distancing measures. For the sake of simplicity, it is considered that an isolated individual is sent to the recovered population when the isolation probability is met ($P\_ISOLATION$).

3. New infected population. This population gathers all individuals infected at each iteration, according to the procedure described in the previous steps. It is possible that repeated new infected individuals are created at each iteration and, consequently, it is recommended to remove such repeated individuals from this population before the next iteration starts running.

Step 4. Stop criterion. One of the most interesting features of the proposed approach lies on its ability to end without the need of controlling any parameter. This situation occurs because the recovered and dead populations are constantly growing as time goes by, and the new infected population cannot infect new individuals. It is expected that the number of infected individuals increases for a certain number of iterations. However, from a particular iteration on, the size of the new infected population will be smaller than that of the current one because recovered and dead populations are too big, and the size of the infected population decays over time. Additionally, a preset number of iterations ($PANDEMIC\_DURATION$) can be added to the stop criterion. The social distancing measures also contributes to reach the stop criterion.

3.2 Remarks for a parallel CVOA version

It must be noted that it is very simple to use CVOA in a multi-virus version since it can be implemented as a population-based algorithm, when considering the pandemic as a set of intelligent agents each of them evolving in parallel. In contrast to trajectory-based metaheuristics, population-based focuses on the diversification in the search space.

For this case, a new variable must be defined, *strains*, which determines the number of strains that will be launched in parallel. Each strain can explore different regions and can be differently configured so that each of them intensifies with their own rates.

Several considerations must be done for this case:

1. Every strain is run independently, following the steps in the previous section.

2. A wise strategy must be followed to generate PZs for each strain. For instance, it is suggested the generation of PZs evenly spaced or, at least, with high Hamming distances. That way, the exploration of distinct regions of the search space is facilitated (diversification).

3. The interaction between the different strains is done by means of dead and recovered populations, that must be shared by all the strains. Operations over these populations must be handled as concurrent updates [24].

4. New infected populations, on the contrary, are different for each strain and no concurrent operations are required.

5. This version may help to simulate different rates for different strains. That way, if there is any initial information about the search space, some strains could be more focused on diversification and some others on intensification.

Depending on the hardware resources and how busy they are, every strain may evolve at different speeds. This situation should not pose any problems since it is known that the pandemic evolves at different rates and starts at different time stamps depending on region of the world.

Last, another application can be found for this parallel version. CVOA simulates an SIR model and consequently, any other global pandemic can be modeled by using the specific rates. Different pandemics could be run in parallel.

3.3 Suggested parameters setup

Since CVOA simulates the COVID-19 propagation, most of the rates (propagation, isolation or mortality) are already known. This fact prevents the researcher from wasting time in selecting values for such rates and turns the CVOA into a metaheuristic quite easy to execute.

However, it must be noted that the current rates are still changing and it is expected they will vary over time, as the pandemic evolves. Maybe these values will not be stable until 2021 or even 2022. The suggested values have been retrieved from the World Health Organization [25] and are discussed below:

1. P_DIE. An infected individual can die with a given probability. The case-fatality ratio (CFR) [26] varies by location, age of person infected and the presence of underlying health conditions but, currently, this rate is set to almost 5% by the scientific community [27]. Therefore, $P\_DIE = 0.05$.

2. $P\_SUPERSPREADER$. It is the probability that an individual spreads the disease to a greater number of healthy individuals. It is believed that this situation affects to a 10% of the infected population [28], therefore, $P\_SUPERSPREADER = 0.1$. After this condition is validated, two situations can be found:

   (a) $ORDINARY\_RATE$. If the infected individual is not a super-spreader, then the infection rate (also known as reproductive number, R_0) is 2.5. It is suggested that this rate is controlled by a random number in the range $[0, 5]$.

   (b) $SUPERSPREADER\_RATE$. If the infected individual turns out to be a super-spreader, then up to 15 healthy individuals can be infected. It is suggested that this rate is controlled by a random number in the range $[6, 15]$.

3. $P\_REINFECTION$. This is a very controversial issue, since the scientific community does not agree on whether a recovered individual can be retested positive or not. As claimed by the WHO, no study has evaluated whether the presence of antibodies to COVID-19 confers immunity to

subsequent infection by this virus in humans [29]. Some tests performed in South Korea suggest a rate of 2% according to the Korea Centers for Disease Control and Prevention [30]. Therefore, $P\_REINFECTION = 0.02$, but this value will be reevaluated, for sure, in the near future.

4. $P\_ISOLATION$. This value is uncertain because countries are taking different measures for social isolation. This parameter helps to reduce the exponential growth of the infected population after each iteration. In other words, this parameter helps to reduce R_0 and it is crucial to ensure the pandemic ends. Therefore, a high value must be assigned to this probability. It is suggested that $P\_ISOLATION \geq 0.7$, since this value ensures $R_0 < 1$ (please refer to Figure 5 to see discussion).

5. P_TRAVEL. This probability simulates how an infected individual can travel to any place in the world and can infect healthy individuals. It is known that almost a 10% of the population travel during a week (simulated time for every iteration) [31], so $P\_TRAVEL = 0.1$.

6. $SOCIAL\_DISTANCING$. It is the number of iterations without social distancing measures. Since the populations grow exponentially at the beginning of the pandemic, this value must be carefully selected and must be set according to the size of the problem. Empirical values that suit for any codification vary from 7 to 12, so it is suggested that $7 \leq SOCIAL\_DISTANCING \leq 12$.

7. $PANDEMIC\_DURATION$. This parameter simulates the duration of the pandemic, that is, the number of iterations. Currently, this data is unknown so this number can be adjusted to the size of the problem. It is suggested that $PANDEMIC\_DURATION = 30$.

8. *strains*. This parameter should be adjusted according to the size of the problem and the hardware availability, and it is difficult to suggest a value suitable for all situations. But a tentative initial value could be five, in an attempt to simulate one different strain per continent. Therefore, $strains = 5$. Another important decision that must be made is how to initialize every PZ associated with the strains. When just one strain is considered, PZ is suggested to be randomly initialized. However, with $strains > 1$ the user should search for orthogonal PZs and to uniformly distribute them in the search space. This strategy should help to cover bigger search spaces in less iterations and to explore individuals with maximal distances.

3.4 Pseudo codes

This section provides the pseudo code of the most relevant functions for the CVOA, along with some comments to better understand them.

3.4.1 Function $CVOA$

This is the main function and its pseudo code can be found in Algorithm 1. Four lists must be maintained: dead, recovered, infected (the current set of infected individuals) and new infected individuals (the set

of new infected individuals, generated by the spreading of the coronavirus from the current infected individuals).

The initial population is generated by means of the patient zero (PZ), which is a random solution.

The number of iterations is controlled by the main loop, evaluating the duration of the pandemic (preset value) and if there is still any infected individual. In this loop, every individual can either die (it is sent to the dead list) or infect, thus enlarging the size of the new infected population. This infection mechanism is coded in function $infect$ (see Section 3.4.2).

Once the new population is formed, all individuals are evaluated and if any of them outperforms the best current one, the latter is updated.

Algorithm 1 Function **cvoa**

```
1: define infectedPopulation, newInfectedPopulation as set of Individual
2: define dead, recovered as list of Individual
3: define PZ, bestIndividual, currentBestIndividual, aux as Individual
4: define time as integer
5: define bestSolutionFitness, currentbestFitness as real
6: time ← 0
7: PZ ← InfectPatientZero()
8: infectedPopulation ← PZ
9: bestIndividual ← PZ
10: while time < PANDEMIC_DURATION AND sizeof(infectedPopulation) > 0 do
11:    dead ← die(infectedPopulation)
12:    for all i ∈ infectedPopulation do
13:       aux ← infect(i,recovered,dead)
14:       if notnull(aux) then
15:          newInfectedPopulation ← aux
16:       end if
17:    end for
18:    currentBestIndividual ← selectBestIndividual(newInfectedPopulation)
19:    if fitness(currentBestIndividual) > bestIndividual then
20:       bestIndividual ← currentBestIndividual
21:    end if
22:    recovered ← infectedPopulation
23:    clear(infectedPopulation)
24:    infectedPopulation ← newInfectedPopulation
25:    time ← time + 1
26: end while
27: return bestIndividual
```

3.4.2 Function $infect$

This function receives an infected individual and returns the set of new infected individuals. Two additional lists, recovered and dead, are also received as input parameters since they must be updated after the evaluation of every infected individuals. The pseudo code is shown in Algorithm 2.

Two conditions are evaluated to determine the number of new infected individuals (use of $SPREADER\_RATE$ or $SUPERSPREADER\_RATE$) or how different the new individuals will be ($ORDINARY\_RATE$ or $TRAVELER\_RATE$. The implementation on how these new infected individuals are encoded according

to such rates is carried out in the function $newInfection$.

Algorithm 2 Function **infect**

Require: infected **as of** *Individual*; recovered, dead **as** *list* **of** *Individual*

define R1, R2 **as** *real*
define newInfected **as** *list* **of** *Individual*
R1 ← RandomNumber()
R2 ← RandomNumber()
if R1 < P_TRAVEL **then**
  if R2 < $P\_SUPERSPREADER$ **then**
    newInfected ← newInfection (infected, recovered, dead, $SPREADER\_RATE$, $ORDINARY\_RATE$)
  else
    newInfected ← newInfection (infected, recovered, dead, $SUPERSPREADER\_RATE$, $ORDINARY\_RATE$)
  end if
else
  if R2 < $P\_SUPERSPREADER$ **then**
    newInfected ← newInfection (infected, recovered, dead, $SPREADER\_RATE$, $TRAVELER\_RATE$)
  else
    newInfected ← newInfection (infected, recovered, dead, $SUPERSPREADER\_RATE$, $TRAVELER\_RATE$)
  end if
end if
return newInfected

3.4.3 Function $newInfection$

Given an infected individual, this function generates new infected individuals according to the spreading and traveling rates. This function also controls that the new infected individuals are not already in the dead list (in such case this new infection is ignored) or in the recovered list (in such case the $P\_REINFECTION$ is applied to determine whether the individual is reinfected or if it remains in the recovered list). Additionally, it considers that the new potential infected individual might be isolated, which is controlled by $P\_ISOLATION$. Although the use of an extra list could be implemented, it has been decided to treat these individuals as recovered. Therefore, if an isolated individual is attempted to be infected, it is added to the recovered list.

The effective generation of the new infected individuals must be carried in the function *replicate*, whose pseudo code is not provided because it depends on the codification and the nature of the problem to be optimized. This function must return a set of new infected individuals, according to the aforementioned rates. Specific information on how this codification and replication is done for LSTM models is provided in Section 4.

The pseudo code for the described procedure can be found in Algorithm 3.

Algorithm 3 Function **newInfection**

Require: infected **as** *Individual*; recovered, dead **as** list **of** *Individual*
1: **define** R3, R4 **as** *real*
2: **define** newInfected **as** *list* **of** *Individual*
3: R3 ← RandomNumber()
4: R4 ← RandomNumber()
5: aux ← replicate(infected, $SPREAD\_RATE$, $TRAVELER\_RATE$)
6: **for all** $i \in aux$ **do**
7: **if** i $\notin$ dead **then**
8: **if** i $\notin$ recovered **then**
9: **if** $R4 > P\_ISOLATION$ **then**
10: newInfected ← i
11: **else**
12: $recovered \leftarrow i$
13: **end if**
14: **else if** $R3 < P\_REINFECTION$ **then**
15: newInfected ← i
16: **remove** i **from** recovered
17: **end if**
18: **end if**
19: **end for**
20: **return** newInfected

3.4.4 Function *die*

This function is called from the *main* function. It evaluates all individuals in the infected population and determines whether they die or not, according to the given P_DIE. Those meeting this condition, are sent to the dead list. Algorithm 4 describes this procedure.

Algorithm 4 Function **die**

Require: infectedPopulation **as** *list* **of** *Individual*
1: **define** dead **as** *list* **of** *Individual*
2: **define** R5 **as** *real*
3: **for all** i ∈ infectedPopulation **do**
4: R5 ← RandomNumber()
5: **if** R5 < P_DIE **then**
6: dead ← i
7: **end if**
8: **end for**
9: **return** dead

3.4.5 Function *selectBestIndividual*

This is an auxiliary function used to find the best fitness in a list of infected individuals. Its peudo code is shown in Algorithm 5.

4 Hybridizing deep learning with CVOA

This section describes the codification proposed for an individual, in order to hybridize deep learning with CVOA. The term hybridize is used in this context as the combination of two computational techniques

Algorithm 5 Function **selectBestIndividual**

Require: infectedPopulation **as** *list* **of** *Individual*

```
define bestIndividual as Individual
define bestFitness as real
bestFitness ← MINVALUE
for all i ∈ infectedPopulation do
    if fitness(i) > bestFitness then
        bestFitness ← fitness(i)
        bestIndividual ← i
    end if
end for
return bestIndividual
```

(deep learning and CVOA) so that the best hyperparameter values are discovered. This strategy is very common in machine learning for optimizing models during the training process [32, 33, 34].

Hence, the individual codification shown in Figure 1 has been implemented in order to apply CVOA to optimize deep neural network architectures.

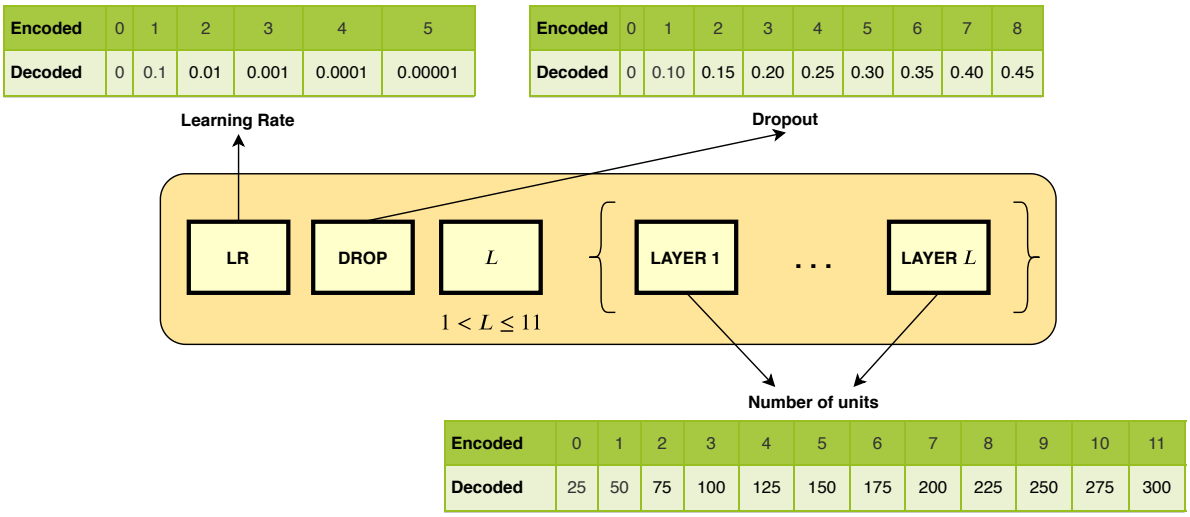

Figure 1: Individual codification for hybridizing deep learning architectures using the proposed CVOA algorithm.

As it can be seen in Figure 1, each individual is composed of the following elements. The element LR encodes the learning rate used in the neural network algorithm. It can take a value from 0 to 5 and its corresponding decoded values are 0, 0.1, 0.01, 0.001, 0.0001 and 0.00001.

The element DROP encodes the dropout rate applied to the neural network. It can take values from 0 to 8 that correspond to 0, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40 and 0.45, respectively. The dropout rate is distributed uniformly for all the layers of the network. That is, if the dropout is 0.4 and the network has 4 layers, then the 10% (0.1) of the neurons of each layer will be removed.

The element L of the individual stores the number of layers of the network. It is restricted to $1 < L \leq 11$. The first layer is referred to the input layer of the neural network. The rest of layers are hidden layers. The output layer is excluded from the codification. Therefore, the optimized network can

contain from 1 to 10 hidden layers.

The proposed individual codification has a variable size. Thus, its size depends on the number of layers indicated in the element L. Consequently, a list of elements (LAYER 1, ..., LAYER L) are also included in the individual, which encode the number of units (neurons) for each network layer. Each of these elements can take values from 0 to 11, and their corresponding decoded values range from 25 to 300, with a step of 25.

4.1 PZ generation

The PZ, as it has been described previously, is the individual of the first iteration in the CVOA algorithm. Following the hybridization proposed, a random individual is created considering the codification defined above.

In first place, a random value for the learning rate of the PZ is generated. Specifically, a number between 0 and 5 is generated randomly in a uniform distribution. Such limits are indicated in Figure 1, according to the possible encoded values of the learning rate element. The same process is carried out to produce a random value for the dropout element. In such case, a random number between 0 and 8 is generated.

In second place, a random number of layers is generated for the element L of PZ. Such number of layers is a random number between 2 and 11. Note that the first layer is reserved for the input layer of the neural network, as it has been discussed before.

In last place, for each one of the L layers, a random number of units is generated between 0 and 11, covering the possible encoded values for the number of units previously defined (see Figure 1).

4.2 Infection procedure

The infection procedure described here corresponds to the functionality of *replicate*(), introduced in the line 4 of the Algorithm 3. This procedure takes an individual as input and returns an infected individual according to the following procedure.

The first step is to determine the element L of the infected individual that will be mutated. The probability of such mutation occurs has been set to $\frac{1}{3}$ so that every element has the same probability to mutate. If the mutation occurs, then the element L of the individual is modified according to the process described in Section 4.4.

If the element L (the number of layers of the network) changes, then the elements encoding the different layers within the individual (LAYER 1, ..., LAYER L) must be resized accordingly. Such resizing process is explained in Section 4.3.

The second step is to determine how many elements of the individual will be infected. If the $TRAVELER\_RATE < 0$, then the number of infected elements is generated randomly from 0 to the

length of the individual (excluding the element L). Else, the $TRAVELER\_RATE$ indicates itself the number of infected elements.

As third step, once it is determined the number of infected elements of the individual, a list of random positions is generated. For example, if three positions of the individual must be changed, then the random positions affected could be, for instance, whose referred to the elements {DROP, LAYER 2, LAYER 4}.

Finally, the selected positions of the individual are mutated. Such mutation is described in Section 4.4.

4.3 Individual resizing process

When an individual is infected at the position of the element L, the list of elements that encodes the number of units per layer (LAYER 1, ..., LAYER L) must be resized accordingly.

In the case that the new number of layers after the infection is lower than its previous value, then the last leftover elements are removed. For instance, if the initial individual is $\{2, 0, 4\}\{3, 2, 1, 6\}$ (four layers), the element $L = 4$ is infected and the new value is $L = 2$, then the resulting individual will be $\{2, 0, 2\}\{3, 2\}$.

In the case that the new number of layers after the infection is higher than its previous value, the new random elements are added at the end of the individual. For instance, if the initial individual is $\{2, 0, 4\}\{3, 2, 1, 6\}$ (four layers), the element $L = 4$ is infected and the new value is $L = 6$, then the resulting individual could be $\{2, 0, 6\}\{3, 2, 1, 6, 0, 4\}$.

4.4 Single position mutation

The process carried out to change the value of a specific element of an individual is described in this section.

First, a signed change amount $C \in \{-2, -1, +1, +2\}$ is randomly determined using the following criteria. A random real number P between 0 and 1 is generated using a uniform distribution. If $P < 0.25$, then the change amount will be $C = -2$. Else if $P < 0.5$, then the change amount will be $C = -1$. Else if $P < 0.75$, then the change amount will be $C = +1$. Else, the change amount will be $C = +2$.

Once the amount of change is determined, the new value for the infected element is computed. If its previous value is V, then the new value after the single position mutation will be $V' = V + C$. If the new value V' exceeds the limits defined for the individual codification, such value is set to the maximum or minimum allowed value accordingly.

5 CVOA sensitivity analysis

This section discusses several aspects about the sensitiveness of CVOA to different configurations. Hence, Section 5.1 evaluates the evolution of the populations for a different number of strains. Section 5.2 assesses the performance when other well-known viruses are modeled. Finally, Section 5.3 provides information about R_0 and how it varies when social distancing measures change.

5.1 Sensitivity to the number of strains

This section provides an overview on how populations evolve over time and how the search space is explored, when a different number of strains is used.

A binary codification has been used, with 20 bits, to conduct this experimentation. A simple fitness function has been evaluated, $f(x) = (x-15)^2$, because the goal of this section is to evaluate the growth of the populations, and not to find challenging optimum values. This function reaches the minimum value at $x = 15$, that is, $f(15) = 0$.

According to Section 3.3, the following configuration has been used: $P\_DIE = 0.05$, $P\_ISOLATION = 0.8$, $P\_SUPERSPREADER = 0.1$, $P\_REINFECTION = 0.02$, $SOCIAL\_DISTANCING = 8$, $P\_TRAVEL = 0.1$ and $PANDEMIC\_DURATION = 30$.

Every experiment has been launched 50 times and, on average, the optimum value was found during the iteration number 13, 6 and 3, for 1, 4 and 8 strains, respectively.

Figure 2 illustrates the evolution of the new infected population over time, for 1, 4 and 8 strains. The number of new infected people increases exponentially during the first $SOCIAL\_DISTANCING = 8$ iterations because $R_0 > 0$ but, from iteration 9 on, an acute decrease is reported because R_0 becomes less than 0. This fact is controlled by $P\_ISOLATION = 0.8$ (a deeper study on R_0 and $P\_ISOLATION$ can be found in Section 5.3). It must be noted that iteration 0 (PZ infection) counts as a regular iteration.

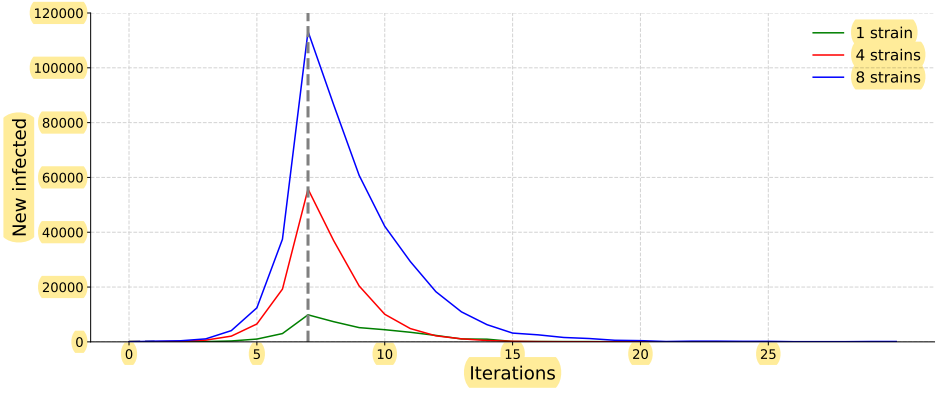

Figure 2: Number of new infected individuals for a 20-bit binary codification execution, with 1, 4 and 8 strains.

Figures 3 and 4 show the accumulated number of recovered people and accumulated deaths, respec-

tively. Note that deaths and recovered individuals cannot be infected again (except for the individuals in the recovered list that can be reinfected with a given probability, $P\_REINFECTION$). These two curves are a direct consequence of the number of new infected people so, once the number of new infections decreases or even disappears, these values remain almost constant. Also, it can be observed that $P\_ISOLATION = 0.8$ after $SOCIAL\_DISTANCING = 8$ iterations help to flatten the curves. A directly proportional relationship is reported between the number of strains and the number of explored individuals at the end of the pandemic.

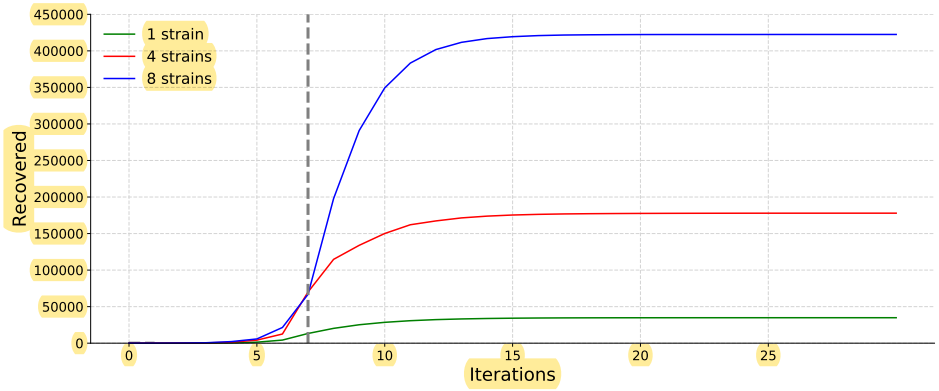

Figure 3: Total number of recovered people for a 20-bit binary codification execution, with 1, 4 and 8 strains.

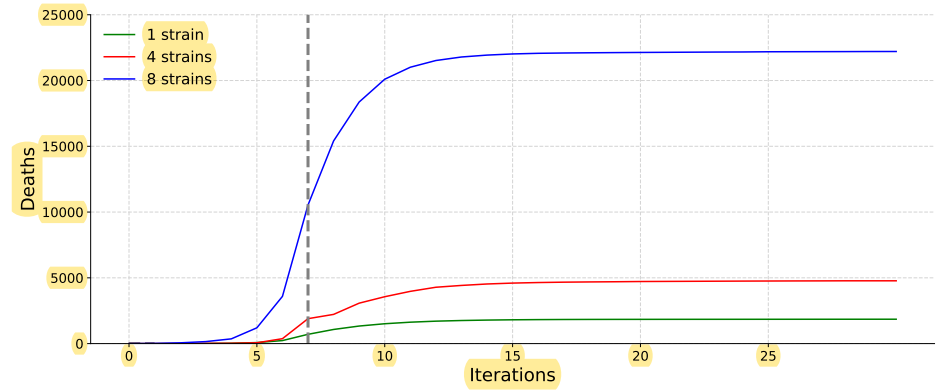

Figure 4: Total number of deaths for a 20-bit binary codification execution, with 1, 4 and 8 strains.

Four main conclusions can be drawn from the analysis of these figures:

1. The number of new infected individuals, accumulated recovered and deaths is directly proportional to the number of strains.

2. The higher the number of strains, the lower number of iterations that are required to reach the optimal value.

3. The number of individuals evaluated increases at each iteration on an almost linear basis, as the

number of strains increases. In case no random numbers were generated, the relationship would be directly proportional, that is, four strains would evaluate four times the number of individuals than one strain would do.

4. To reach the optimum values, the search space explored is smaller as the number of strains increases. This is due to the generation of PZ evenly spaced, which makes easier to explore wider areas.

5.2 Sensitivity to the parameters

Several well-known viruses with deep impact in human beings' health are modeled in this section, in order to assess the CVOA robustness to different input parameters values.

Middle East Respiratory Syndrome (MERS), Severe Acute Respiratory Syndrome (SARS), influenza (seasonal strains) and ebola have been selected, with the parametrization shown in Table 1. It is worth mentioning that the modeling of each virus requires much research and a approximate parametrization has been used, according to the references in the rightmost column.

Table 1: Parametrization for other viruses.

Disease	R_0	Fatality rate	Vaccine	Super-spreaders	References
SARS	1.4-2.5	11%	No	Yes	[35, 36]
MERS	0.3-0.8	34.4%	No	Yes	[28, 35, 37]
Influenza	0.9-2.1	0.1%	Yes	No	[38]
Ebola	1.5-1.9	50%	Yes	No	[39, 40]

All experiments have been conducted with four strains and 30 iterations. The viruses with vaccines have been simulated by using $P\_ISOLATION = 0.95$ after 5 iterations, since this feature is not implemented in CVOA.

Table 2 summarizes the percentage of search space explored and the best fitness found, on average. Codifications of 10, 20, 30, 40 and 50 bits have been used, with associated search spaces of length 1024, 1.05E+6, 1.07E+09, 1.10E+12 and 1.13E+15, respectively. Several findings are revealed:

1. CVOA finds the optimal values even for the longest codification (50 bits) and it is done by exploring a similar search space size as the other configurations do.

2. SARS is the second best parametrization, reaching remarkable fitness even for 50 bits. But it required the evaluation of a greater number of individuals and, therefore, the execution time was greater as well.

3. MERS obtained the poorest results in terms of fitness but it explored a smaller space search. This situation may be explained due to the low associated reproductive number ($R_0 < 1$).

4. Influenza has obtained slightly worse results in terms of fitness than CVOA but with less solutions explored. This configuration may be useful to obtain satisfactory results in a reduced execution time.

5. The high death fatality rate of ebola prevents from exploring most of the search space. This fact makes difficult to visit optimal values. However, results for 40 bits are satisfactory in terms of fitness. For 50 bits, its use is discouraged considering the poor fitness value reached.

Table 2: CVOA performance when different configurations.

	10 bits		20 bits		30 bits		40 bits		50 bits	
Disease	Explored	Fitness	Explored	Fitness	Explored	Fitness	Explored	Fitness	Explored	Fitness
SARS	57.32%	0	0.54%	0	6E-03%	1	1E-05%	4	3E-08%	252
MERS	20.34%	0	0.04%	16	1E-02%	36	1E-05%	112	2E-09%	3210
Influenza	13.23%	0	0.02%	0	8E-04%	2	1E-06%	14	1E-08%	310
Ebola	62.93%	0	0.44%	0	7E-02%	4	2E-05%	15	1E-09%	810
COVID-19	15.63%	0	0.21%	0	1.4E-03%	0	1.6E-05	0	2.0E-08	0

It can be concluded that variations in the input parameters values lead to results varying both in fitness and execution time. This feature is very useful for the CVOA parallel version, since strains with different rates and probabilities can be simultaneously launched. That is, strains aiming at diversifying can be combined with strains aiming at intensifying.

5.3 Sensitivity to the social distancing measures

In this section an analysis on how $P\_ISOLATION$ modifies R_O is conducted. The purpose is to discover when $R_0 < 1$, situation in which the pandemic prevalence declines. A study with a 10-bit to 50-bit codification has been done as well as using different number of strains (1, 4 and 8).

Figure 5 illustrates how R_0 varies for a 40-bit codification, with probabilities of isolation ranging from 0 to 1, and with 1, 4 and 8 strains. Quite similar behaviors have been achieved for all codifications.

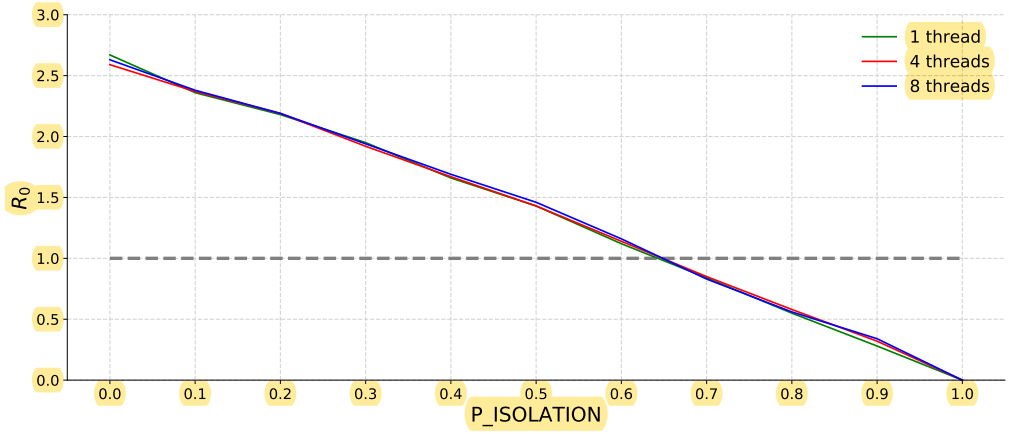

Figure 5: R_0 sensitivity to $P\_ISOLATION$, in a 40-bit codification.

From the analysis of this figure, several conclusions are drawn:

1. R_0 is linear and inversely proportional to $P\_ISOLATION$.

2. The same negative slope is shown, with variations no higher than 10E-2 on average for all codifications and number of strains.

3. R_0 is less than 1 with $P\_ISOLATION$ values close to 0.65 (and higher). This fact involves a decline of the infectious disease.

6 Results

This section reports the results achieved by hybridizing a deep learning model with CVOA. Section 6.1 describes the study case selected to prove the effectiveness of the proposed algorithm. Section 6.2 describes the dataset used. Section 6.3 discusses the results achieved and includes some comparative methods.

6.1 Study case: electricity demand time series forecasting

The forecasting of future values fascinates the human being. To be able to understand how certain variables evolve over time has many benefits in many fields.

Electricity demand forecasting is not an exception, since there is a real need for planning the amount to be generated or, in some countries, to be bought.

The use of machine learning to forecast such time series has been intensive during the last years [41]. But, with the development of deep learning models, and, in particular of LSTM, much research is being conducted in this application field [42].

6.2 Dataset description

The time series considered in this study is related to the electricity consumption in Spain from January 2007 to June 2016, the same as used in [43]. It is a time series composed of 9 years and 6 months with a 10-minute sampling frequency, resulting in 497832 measures.

As in the original paper, the prediction horizon is 24, that is, this is a multi-step strategy with $h = 24$. The size of samples used for the prediction of these 24 values is 168. Furthermore, the dataset was split into 70% for the training set and 30% for the test set, and in addition, a 30% of the training set has also been selected for the validation set, in order to find the optimal parameters. The training set covers the period from January 1, 2007 at 00:00 to August 20, 2013 at 02:40. Therefore, the test set comprises the period from August 20, 2013 at 02:50 to June 21, 2016 at 23:40.

6.3 Performance analysis

This section reports the results obtained by hybridizing LSTM with CVOA, by means of the codification proposed in Section 4, to forecast the Spanish electricity dataset described in Section 6.2.

Linear regression (LR), decision tree (DT), gradient-boosted trees (GBT) and random forest (RF) models have been used with a parametrization setups according to those studied in [44, 45]. A deep neural network optimized with a grid search (DNN-GS) according to [43] has also been applied. Another deep neural network, but optimized with random search (DNN-RS) and smoothed with a low-pass filter (DNN-RS-LP) [46], has also been applied. Furthermore, CVOA has been combined with DNN (DNN-CVOA), using the same codification as in LSTM.

These results along with those of LSTM, and combinations with GS, RS, RS-LP and CVOA are summarized in Table 3, expressed in terms of the mean absolute percentage error (MAPE). It can be observed that LSTM-CVOA outperforms all evaluated methods which have showed particularly remarkable performance for this real-world dataset. Additionally, DNN-CVOA outperforms all other DNN configurations which confirms the superiority of CVOA with reference to GS, RS, and RS-LP.

Another relevant consideration that must be taken into account is that the compared methods generated 24 independent models, each of them for every value forming h. So, it would expected that LSTM-CVOA performance increases if independent models are generated for each of the values in h.

Table 3: Results in terms of MAPE for CVOA-LSTM compared to other well established methods.

Method	MAPE (%)
LR	7.34
DT	2.88
GBT	2.72
RF	2.20
DNN-GS	1.68
DNN-RS	1.57
DNN-RS-LP	1.36
DNN-CVOA	1.18
LSTM-GS	1.22
LSTM-RS	0.84
LSTM-RS-LP	0.82
LSTM-CVOA	**0.47**

These results have been achieved with the individual $\{4, 0, 8\}\{9, 7, 2, 7, 2, 7, 10, 7\}$, which decoded involves the following architecture parameters:

1. Learning rate: 10E-04.
2. Dropout: 0.
3. Number of layers: 8.
4. Units per layer: $[250, 200, 75, 200, 75, 200, 275, 200]$

Finally, Figure 6 depicts the first five predicted days versus their actual values, expressed in watts.

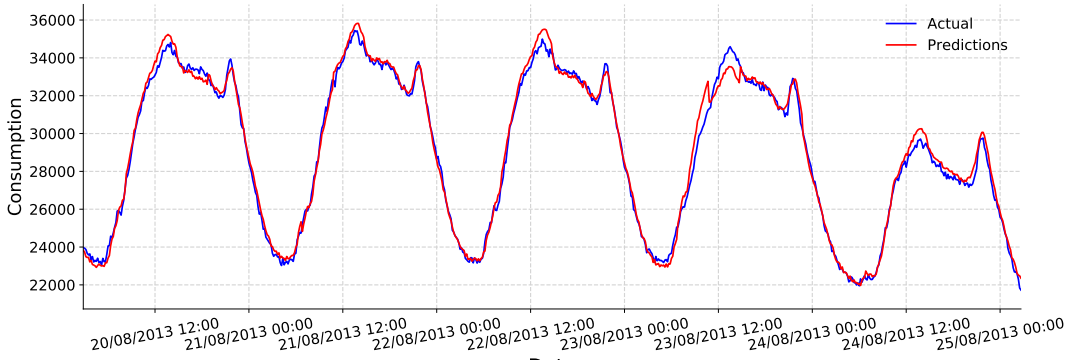

Figure 6: Actual versus predicted values for the first five days in the test set (in watts).

7 Conclusions and future works

This work has introduced a novel bioinspired metaheuristic, based on the COVID-19 pandemic behavior. On the one hand, CVOA has three major advantages. First, its highly relation to the coronavirus spreading model, prevents the users from making any decision about the inputs' values. Second, it ends after a certain number of iterations due to the exchange of individuals between healthy and dead/recovered lists.

Additionally, a novel discrete and dynamic codification has been proposed to hybridize deep learning models. On the other hand, it exhibits some limitations. Such is the case for the exponential growth of the infected population as time (iterations) goes by.

Furthermore, a parallel version is proposed so that CVOA is easily transformed into a multi-virus metaheuristic, in which different coronavirus strains search for the best solution in a collaborative way. This fact allows to model every strain with different initial setups (higher $DEATH\_RATE$, for instance), sharing recovered or dead lists.

Additional experimentation must be conducted in order to assess its performance on standard F functions and find out the search space shapes in which it can be more effective.

As for future work, some actions might be taken to reduce the size of the infected population after several iterations, that grows exponentially. In this sense, a vaccine could be implemented. This case would involve adding to the recovered list, at a given $VACCINE\_RATE$ healthy individuals. This rate will remain unknown until a vaccine is developed.

Another suggested research line is using dynamic rates. For instance, the observation of the preliminary effects of the social isolation measures in countries like China, Italy or Spain, suggests that the $INFECT\_RATE$ could be simulated as a Poisson process, but more time and country recoveries is required to confirm this trend.

For the multi-step forecasting problem analyzed, it would be desirable to generate independent models for each of the values that form the prediction horizon h.

Finally, further research has to be conducted in order to assess the CVOA performance when applied to other fields and combined with other networks.

Supplementary material

Along with this paper, an academic version in Java for a binary codification is provided, with a simple fitness function in a GitHub repository (`https://github.com/DataLabUPO/CVOA_academic`). The master branch includes a simple implementation whereas the sets branch provides an optimized version with a command line interface. Additionally, the code in Phyton for the deep learning approach is also provided, with a more complex codification and the suggested implementation, according to the pseudocode provided (`https://github.com/DataLabUPO/CVOA_LSTM`).

Acknowledgments

The authors would like to thank the Spanish Ministry of Economy and Competitiveness for the support under project TIN2017-88209-C2.

References

[1] T. P. Velavan and C. G. Meyer. The COVID-19 epidemic. *Tropical Medicine and International Health*, 25:278–280, 2020.

[2] R. Li, S. Pei, B. Chen, Y. Song, T. Zhang, W. Yang, and J. Shaman. Substantial undocumented infection facilitates the rapid dissemination of novel coronavirus (SARS-CoV-2). *Nature*, 368:489–493, 2020.

[3] G. Giordano, F. Blanchini, R. Bruno, P. Colaneri, A. Di Filippo, A. Di Matteo, and M. Colaneri. Modelling the COVID-19 epidemic and implementation of population-wide interventions in Italy. *Nature Medicine*, 2020. `https://doi.org/10.1038/s41591-020-0883-7`.

[4] J. Del Ser, E. Osaba, D. Molina, X.-S. Yang, S. Salcedo-Sanz, D. Camacho, S. Das, P. N. Suganthan, C. A. Coello Coello, and F. Herrera. Bio-inspired computation: Where we stand and what's next. *Swarm and Evolutionary Computation*, 48:220–250, 2019.

[5] D. Tolić, K. Kleineberg, and N. Antulov-Fantulin. Simulating SIR processes on networks using weighted shortest paths. *Scientific Reports*, 8:6562, 2018.

[6] I. Boussaïd, J. Lepagnot, and P. Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013.

[7] M. Z. Tay, C. M. Poh, L. Rénia, P. A. MacAry, and L. F. P. Ng. The trinity of COVID-19: immunity, inflammation and intervention. *Nature Reviews Immunology*, 2020. https://doi.org/10.1038/s41577-020-0311-8.

[8] World Health Organization. https://www.who.int/es/emergencies/diseases/novel-coronavirus-2019. Accessed: 2020-03-20.

[9] A. Kelotra and P. Pandey. Stock Market Prediction Using Optimized Deep-ConvLSTM Model. *Big Data*, 8(1):5–24, 2020.

[10] S. De-Cnudde, Y. Ramon, D. Martens, and F. Provost. Deep learning on big, sparse, behavioral data. *Big Data*, 7(4):286–307, 2019.

[11] F. Glover and G. A. Kochenberger. *Handbook of metaheuristics*. Springer, 2003.

[12] Y. C. Liang and J. R. Cuevas-Juárez. A novel metaheuristic for continuous optimization problems: Virus optimization algorithm. *Engineering Optimization*, 48(1):73–93, 2016.

[13] Y. C. Liang and J. R. Cuevas-Juárez. A self-adaptive virus optimization algorithm for continuous optimization problems. *Soft Computing*, https://doi.org/10.1007/s00500-020-04730-0, 2020.

[14] H. Chung and K.-S. Shin. Genetic Algorithm-Optimized Long Short-Term Memory Network for Stock Market Prediction. *Sustainability*, 10(10):3765, 2018.

[15] J. Chen, H. Xing, H. Yang, and L. Xu. Network Traffic Prediction Based on LSTM Networks with Genetic Algorithm. *Lecture Notes in Electrical Engineering*, 550:411–419, 2019.

[16] Z. Liu, X. Sun, S. Wang, M. Pan, Y. Zhang, and Z. Ji. Midterm power load forecasting model based on kernel principal component analysis and back propagation neural network with particle swarm optimization. *Big Data*, 7(2):130–138, 2019.

[17] F. E. Fernandes-Junior and G. G. Yen. Particle swarm optimization of deep neural networks architectures for image classification. *Swarm and Evolutionary Computation*, 49:62–74, 2019.

[18] T. Desell, S. Clachar, J. Higgins, and B. Wild. Evolving deep recurrent neural networks using ant colony optimization. *Lecture Notes in Computer Science*, 9026:86–98, 2015.

[19] A. ElSaid, F. ElJamiy, J. Higgings, B. Wild, B. Wild, and T. Desell. Using ant colony optimization to optimize long short-term memory recurrent neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 13–20, 2018.

[20] D. Srivastava, Y. Singh, and A. Sahoo. Auto Tuning of RNN Hyper-parameters using Cuckoo Search Algorithm. In *Proceedings of the International Conference on Contemporary Computing*, pages 1–5, 2019.

[21] N. M. Nawi, A. Khan, and M. Z. Rehman. A New Optimized Cuckoo Search Recurrent Neural Network (CSRNN). In *Proceedings of the International Conference on Robotic, Vision, Signal Processing & Power Applications*, pages 335–341, 2014.

[22] A. D. Yuliyono and A. S. Girsang. Artificial Bee Colony-Optimized LSTM for Bitcoin Price Prediction. *Advances in Science, Technology and Engineering Systems Journal*, 4(5):375–383, 2019.

[23] A. Bosire. Recurrent Neural Network Training using ABC Algorithm for Traffic Volume Prediction. *Informatica*, 43:551–559, 2019.

[24] V. Dhar, C. Sun, and P. Batra. Transforming Finance Into Vision: Concurrent Financial Time Series as Convolutional Net. *Big Data*, 7(4):276–285, 2019.

[25] World Health Organization. Coronavirus Disease 2019 (COVID-19): Situation Report 74. Technical report, WHO, 2020. Accessed: 2020-05-09, `https://www.who.int/docs/default-source/coronaviruse/situation-reports/20200403-sitrep-74-covid-19-mp.pdf`.

[26] A. C. Ghani, C. A. Donnelly, D. R. Cox, J. T. Griffin, C. Fraser, T. H. Lam, and G. M Leung. Methods for estimating the case fatality ratio for a novel, emerging infectious disease. *American Journal of Epidemiology*, 162(5):479–486, 2005.

[27] K. Mizumoto and G Chowell. Estimating Risk for Death from 2019 Novel Coronavirus Disease, China, January–February 2020. *Emerging Infectious Diseases*, 26(6), `https://doi.org/10.3201/eid2606.200233`, 2020.

[28] J. Y. Wu, K. Leung, and G. M. Leung. Nowcasting and forecasting the potential domestic and international spread of the 2019-nCoV outbreak originating in Wuhan, China: a modelling study. *Lancet*, `http://dx.doi.org/10.1016/S0140-6736(20)30260-9`, 2020.

[29] World Health Organization. Immunity passports in the context of COVID-19. Technical report, WHO, 2020. Published: 2020-04-29, `https://www.who.int/news-room/commentaries/detail/immunity-passports-in-the-context-of-covid-19`.

[30] Korea Centers for Disease Control and Prevention. Coronavirus Disease-19. `https://www.cdc.go.kr/cdc_eng/`. Accesed: 2020-05-09.

[31] M. C. González, C. A. Hidalgo, and A. L. Barabási. Understanding individual human mobility patterns. *Nature*, 453:779–782, 2008.

[32] L. Calvet, J.D. Armas, D. Masip, and A. A. Juan. Learnheuristics: Hybridizing metaheuristics with machine learning for optimization with dynamic inputs. *Mathematics Open*, 15:261–280, 2017.

[33] A. Darwish, A. E. Hassanien, and S. Das. A survey of swarm and evolutionary computing approaches for deep learning. *Artificial Intelligence Review*, 53(3):1767–1812, 2020.

[34] D. Devikanniga, K. Vetrivel, and N. Badrinath. Review of meta-heuristic optimization based artificial neural networks and its applications. *Journal of Physics: Conference Series*, 1362(1):012074, 2019.

[35] A. Trilla. One world, one health: The novel coronavirus COVID-19 epidemic. *Medicina Clinica*, 154:175–177, 2020.

[36] World Health Organization. Consensus document on the epidemiology of severe acute respiratory syndrome (SARS). Technical report, WHO, 2003. Accessed: 2020-05-10, `https://www.who.int/csr/sars/en/WHOconsensus.pdf`.

[37] World Health Organization. Middle East respiratory syndrome coronavirus (MERS-CoV). Technical report, WHO, 2019. Accessed: 2020-05-10, `https://www.who.int/emergencies/mers-cov/en/`.

[38] B. J. Coburn, B. G. Wagner, and S. Blower. Modeling influenza epidemics and pandemics: insights into the future of swine flu (H1N1). *BMC Medicine*, 7:30, 2009.

[39] A. Khan, M. Naveed, and M. Dur e Ahmad. Estimating the basic reproductive ratio for the Ebola outbreak in Liberia and Sierra Leone. *Infectious Diseases of Poverty*, 4:13, 2015.

[40] World Health Organization. Ebola virus disease. Technical report, WHO, 2020. Accessed: 2020-05-10, `https://www.who.int/news-room/fact-sheets/detail/ebola-virus-disease`.

[41] F. Martínez-Álvarez, A. Troncoso, G. Asencio-Cortés, and J. C. Riquelme. A survey on data mining techniques applied to electricity-related time series forecasting. *Energies*, 8(11):13162–13193, 2015.

[42] J. Bedi and D. Toshniwal. Deep learning framework to forecast electricity demand. *Applied Energy*, 238:1312–1326, 2019.

[43] J. F. Torres, A. Galicia, A. Troncoso, and F. Martínez-Álvarez. A scalable approach based on deep learning for big data time series forecasting. *Integrated Computer-Aided Engineering*, 25(4):335–348, 2018.

[44] A. Galicia, J. F. Torres, F. Martínez-Álvarez, and A. Troncoso. Scalable forecasting techniques applied to big electricity time series. *Lecture Notes in Computer Science*, 10306:165–175, 2019.

[45] A. Galicia, R. L. Talavera-Llames, A. Troncoso, I. Koprinska, and F. Martínez-Álvarez. Multi-step forecasting for big data time series based on ensemble learning. *Knowledge-Based Systems*, 163:830–841, 2019.

[46] J. F. Torres, D. Gutiérrez-Avilés, A. Troncoso, and F. Martínez-Álvarez. Random hyper-parameter search-based deep neural network for power consumption forecasting. *Lecture Notes in Computer Science*, 11506:259–269, 2019.