A Chaotic Inertia Weight TLBO Applied to Troubleshooting Optimization Problems

Daniel B. P. Coelho
Performance Software Department
Embraer S.A.
Belo Horizonte, Brazil
daniel.coelho@embraer.com.br

Leonardo R. Rodrigues
Electronics Engineering Division
Aeronautics Institute of Technology
São José dos Campos, Brazil
leonardolrr2@fab.mil.br

***Abstract*—When a failure event occurs in a complex system with multiple interconnected components, identifying the faulty component can be a challenging task. A troubleshooting optimization problem consists in finding the sequence of activities that must be followed to find the faulty component and fix the system with minimum expected cost of repair (ECR). Troubleshooting optimization can be modeled as a combinatorial optimization problem, and different algorithms have been proposed to solve it. This paper proposes a chaotic inertia weight Teaching-Learning Based Optimization (TLBO) algorithm for the troubleshooting optimization problem. A chaotic sequence is used to update the inertia weight used in each iteration of TLBO. Numerical experiments using three troubleshooting models and nine chaotic maps are conducted to evaluate the performance of the proposed algorithm. The standard TLBO algorithm is also considered in the experiments to establish a reference baseline. The results showed that the proposed model presented a better performance in terms of average ECR, when compared with the standard TLBO algorithm.**

***Index Terms*—metaheuristic, TLBO, inertia weight, chaotic map, combinatorial optimization, troubleshooting.**

I. Introduction

The problem of identifying the faulty component in a complex system has become a topic of great interest for academy researchers and industry practitioners due to its potential benefits [1]. A maintenance investigation is often required to identify the root cause of a system failure, especially in complex systems with multiple components. Troubleshooting is the process of identifying the root cause and repairing the faulty component.

A troubleshooting optimization problem can be solved in polynomial time under very restrictive assumptions [1]. When these assumptions are not valid, the problem becomes NP-hard [2]. In these situations, an alternative to find good solutions in an acceptable time is to use metaheuristic algorithms. Many solutions using a wide range of methods have been proposed to solve troubleshooting problems [3]–[7].

The Teaching-Learning Based Optimization (TLBO) is a population-based metaheuristic algorithm based on the teaching-learning process observed in a classroom [8]. In recent years, many variants of TLBO have been proposed aiming at improving the performance of the algorithm in different aspects. The use of multiple teachers and the adaptation of the teaching factor are some of the modifications proposed in [9] to increase both the diversification and intensification capabilities of TLBO. In [10], the authors incorporated a local learning and a self-learning method into the original formulation of TLBO. In [11], the authors proposed an elitist TLBO. The elitism approach preserves the best candidates of each iteration and prevents the loss of a good candidate solution.

Many population-based metaheuristic algorithms include in their mathematical formulation a parameter that is randomly changed during the execution of the method. A study on the impact of using chaotic sequences instead of random parameters in evolutionary algorithms is presented in [12]. Many works have been published reporting the successful use of chaotic mechanisms in different algorithms such as Symbiotic Organisms Search (SOS) [13], Firefly algorithm [14], and Whale Optimization [15], among others. Chaotic versions of TLBO have also been proposed [16].

The inertia weight concept was introduced in [17]. Inertia weight plays an important role in the control of the balance between the diversification and intensification capabilities of population-based algorithms. In [17], the authors used a constant inertia weight. Other inertia weight strategies have been proposed such as random [18], linear [19], nonlinear [20], adaptive [21], chaotic [22], among others.

In this paper, we propose a modified version of TLBO to solve troubleshooting optimization problems. Our goal is to minimize the expected cost of repair (ECR). The proposed method incorporates a chaotic inertia weight mechanism into the standard TLBO. A chaotic sequence is used to update the inertia weight used in each iteration of TLBO.

The remaining sections of this paper are organized as follows. Section II describes the troubleshooting problem under consideration. Section III presents the basic principles of TLBO, inertia weights and chaotic maps. Section IV presents the proposed method. Section V illustrates the application of the proposed method in three different troubleshooting models. Concluding remarks are given in section VI.

II. Problem Description

As mentioned earlier, a troubleshooting optimization problem consists in finding the sequence of activities that must be

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

followed to find the faulty component and fix a multiple component system with minimum expected cost of repair (ECR). In this paper, we assume that a probabilistic troubleshooting model of the system under consideration is available. A troubleshooting model describes the system failure modes, as well as the set of available maintenance interventions to fix the system. Also, a troubleshooting model contains the costs associated with each intervention [7].

The interventions that can be made in the failed system are divided into two types: repair actions and diagnostic questions. The number of available repair actions and the number of available diagnostic questions in the model are denoted by n_A and n_Q, respectively. A repair action consists in fixing a component (or group of components). Performing a repair action may fix the system or not. A diagnostic question cannot fix the system, but the result of a diagnostic question reduces the number of possible failure modes that may have caused the system failure.

Groups of interventions (actions and questions) may have a common initialization procedure. For example, it may be necessary to disassemble part of the system to access a region where a group of components is located. The resources need to access this region are required to fix one or all the components in the region. This situation is represented in the troubleshooting model by clusters, which are subsets of actions and questions of the model. When at least one action or question belonging to a cluster is performed, the cost of the cluster is incurred [3].

The troubleshooting optimization problem consists in finding a sequence of interventions (actions and questions) that fixes the faulty system and minimizes the expected cost of repair (ECR). Each candidate sequence is called a troubleshooting strategy, which will be denoted by S. Fig. 1 shows an example of a troubleshooting model containing all the basic elements: four failure modes $\{F_1, F_2, F_3, F_4\}$, three repair actions $\{A_1, A_2, A_3\}$, one diagnostic question $\{Q_1\}$, and the associated costs $\{C(A_1), C(Q_1), C(A_2), C(A_3), C(K_1)\}$. Note that cost $C(K_1)$ is the cost associated with the cluster composed by repair action A_2 and diagnostic question Q_1.

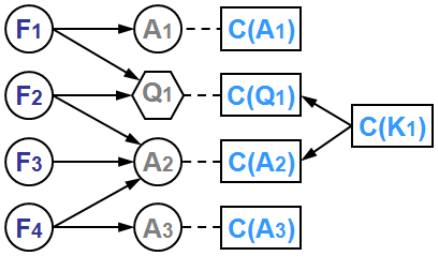

Fig. 1. Example of a troubleshooting model

Failure modes are the ways in which the system can fail. We adopt the assumption that there is one and only one failure mode present in the failed system.

Consider the troubleshooting model presented in Fig. 1. If a system failure is caused by failure mode F_1, then performing action A_1 will be sufficient to fix the system. However, if a system failure is caused by failure mode F_4, then both actions A_2 and A_3 must be performed to fix the system. Also, by performing diagnostic question Q_1 it is possible to know whether the failure mode that caused the system failure belongs to $\{F_1, F_2\}$ or not. Finally, if question Q_1 or action A_2 are performed, the cost of cluster K_1 is incurred.

A troubleshooting strategy S is evaluated by its Expected Cost of Repair (ECR), which is defined according to (1).

$$ECR(S) = \sum_{i=1}^{N_f} p_i \cdot C_i(S) \tag{1}$$

where N_f is the number of failure modes, p_i is the probability associated with failure mode i and $C_i(S)$ is the cost to fix a system failure caused by failure mode F_i using troubleshooting strategy S, which is computed according to (2).

$$\begin{aligned} C_i(S) &= \sum_{j=1}^{N_a} C(A_j) \cdot \alpha_j(S) + \\ &+ \sum_{v=1}^{N_q} C(Q_v) \cdot \beta_v(S) + \\ &+ \sum_{z=1}^{N_k} C(K_z) \cdot \gamma_z(S) \end{aligned} \tag{2}$$

where N_a, N_q and N_k are the number of actions, questions and clusters in the troubleshooting model, respectively. The terms $C(A_j)$, with $j \in \{1, \dots N_a\}$, $C(Q_v)$, with $v \in \{1, \dots N_q\}$ and $C(K_z)$, with $z \in \{1, \dots N_k\}$ are costs associated with each action, question and cluster, respectively. Also, $\alpha_j(S)$, $\beta_v(S)$ and $\gamma_z(S)$ are binary decision variables that assume value 1 if the associated action, question or cluster cost is incurred and zero otherwise.

Consider that the system described by the troubleshooting model presented in Fig. 1 is failed due to failure mode F_4. Also, consider that troubleshooting strategy $S = \{A_2, Q_1, A_1, A_3\}$ is used to fix the system. In this example, the first intervention is to perform action A_2. Costs $C(A_2)$ is incurred. Action A_2 activates cluster K_1, so cost $C(K_1)$ is also incurred. The system is still failed, so the next step is to perform question Q_1. Cost $C(Q_1)$ is incurred. Since cluster K_1 was already activated by action A_2, no additional cost is incurred in this step. The execution of question Q_1 provides the information that the failure mode that caused the system failure belongs to $\{F_3, F_4\}$. Following the sequence of troubleshooting strategy S, the next step would be to perform action A_1. However, action A_1 can fix a failure caused by failure mode F_1 only, and at his step of strategy S it is already known that the failure mode which caused the system failure belongs to $\{F_3, F_4\}$. For this reason, action A_1 is skipped and the next step becomes to perform action A_3, which fixes the system, with cost $C(A_3)$. The total cost associated to this troubleshooting strategy to fix a failure caused by failure mode F_4 is given by $C_4(S) = C(A_2) + C(K_1) + C(Q_1) + C(A_3)$.

III. Theoretical Background

A. Teaching-Learning Based Optimization

The TLBO (Teaching-Learning Based Optimization) algorithm is a population-based metaheuristic algorithm inspired by the teaching-learning process observed in a classroom [8]. This algorithm simulates the influence of a teacher on the output of a group of students in a class. The algorithm has two main phases: the Teacher Phase and the Student Phase [9]. During the Teacher Phase, students learn from the teacher, while in the Student Phase students learn through interactions among themselves.

Consider a group of N students. Each student X has an associated solution that corresponds to a candidate solution for the optimization problem. The quality of each solution is quantified by fitness value $f(X)$, that is computed by evaluating the solution X using the objective function J.

The student with the best solution in each iteration is called the Teacher. Fig. 2 shows the flowchart for implementing the TLBO algorithm [8]. The Teacher Phase and the Student Phase are described in the next sections.

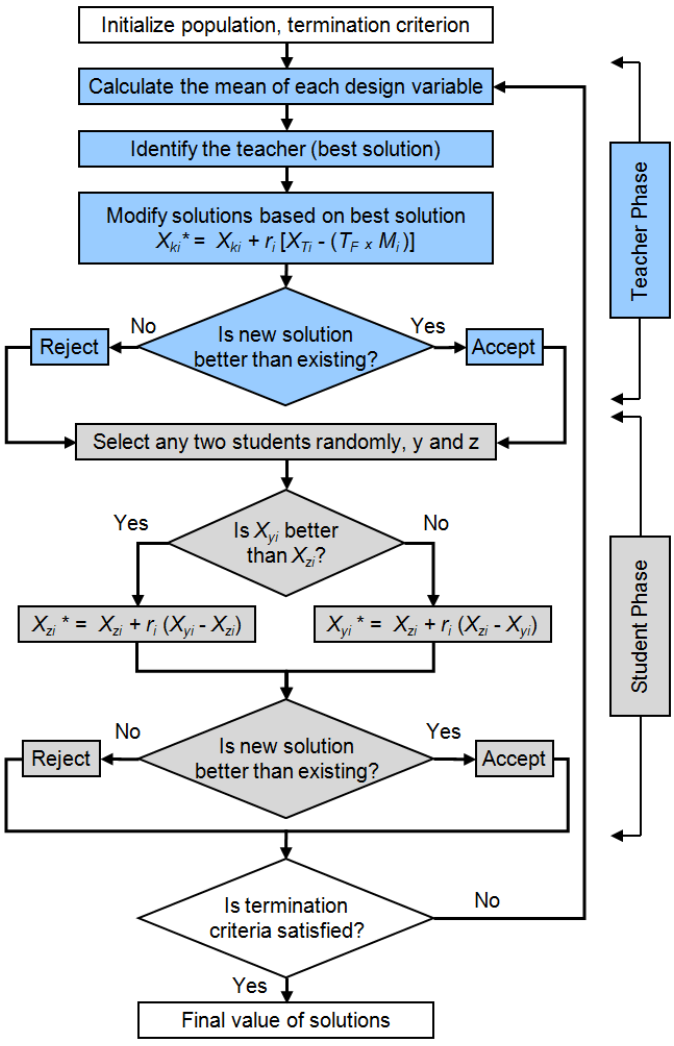

Fig. 2. TLBO flowchart

*1) **Teacher Phase**:* During the Teacher Phase, the algorithm simulates the learning of the students from the teacher (best solution). During this phase, the teacher makes an effort to increase the mean result of the class. Let M_i be the mean solution of all the students and T_i be the teacher in the i-th iteration. The teacher T_i will try to move M_i to its own level. Knowledge is obtained based on the quality of the teacher and the quality of students. The difference D_i between the solution of the teacher, denoted by X_{Ti}, and the mean solution of the students, M_i, is expressed according to (3).

$$D_i = r_i(X_{Ti} - T_F \cdot M_i) \tag{3}$$

where r_i is a random number chosen from a standard uniform distribution, and T_F is the teaching factor for iteration i, which is randomly set to either 1 or 2 according to (4).

$$T_F = round(1 + rand(0, 1)) \tag{4}$$

Based on the difference D_i, the current solution associated with each student k in iteration i, denoted by X_{ki}, with $k \in \{1, 2, \ldots, n\}$, is updated during the teacher phase according to (5).

$$X_{ki}^\star = X_{ki} + D_i \tag{5}$$

where $X_{ki}^\star$ is the updated value of X_{ki}.

If $f(X_{ki}^\star)$ is better than $f(X_{ki})$, then $X_{ki}^\star$ is accepted and replaces X_{ki} for the next iteration. Otherwise, $X_{ki}^\star$ is discarded.

*2) **Student Phase**:* During the Student Phase, TLBO simulates the learning of the students through interactions among themselves. During this phase, students gain knowledge by discussing with other students who have more knowledge [9].

Consider a pair of students y and z. Let X_{yi} and X_{zi} be the solutions of students y and z in iteration i, respectively. If $f(X_{yi})$ is better than $f(X_{zi})$, the solution of student z is updated according to (6). If $f(X_{zi}^\star)$ is better than $f(X_{zi})$, $X_{zi}^\star$ is accepted and replaces X_{zi} for the next iteration. Otherwise, $X_{zi}^\star$ is discarded. Similarly, if $f(X_{zi})$ is better than $f(X_{yi})$, the solution of student y is updated according to (7). If $f(X_{yi}^\star)$ is better than $f(X_{yi})$, $X_{yi}^\star$ is accepted and replaces X_{yi} for the next iteration. Otherwise, $X_{yi}^\star$ is discarded.

$$X_{zi}^\star = X_{zi} + r_i(X_{yi} - X_{zi}) \tag{6}$$

$$X_{yi}^\star = X_{yi} + r_i(X_{zi} - X_{yi}) \tag{7}$$

At the end of each iteration, the termination criteria is checked. Different termination criteria may be adopted. Some commonly used termination criteria are the maximum number of iterations, the maximum number of successive iterations without any improvement, the maximum simulation time, and the maximum number of function evaluations.

B. Inertia Weights

The inertia weight concept, originally introduced in [17], has an important role in optimization processes using population-based metaheuristic algorithms. It provides a good balance between the diversification and the intensification capabilities of the algorithm [23]. High inertia weight values increase the diversification capability, while low inertia weight values increase the intensification capability of the algorithm.

A constant inertia weight w was initially used in Particle Swarm Optimization (PSO) to update the velocity of each particle [17]. The inertia weight was used as a memory element, representing the contribution of the previous velocity to the updated velocity value. In the last years, several works proposed different strategies to implement dynamic inertia weights in different population-based metaheuristic algorithms. Some of the inertia weight strategies are listed below.

- **Constant**: As mentioned earlier, a constant inertia weight was initially used. In this strategy, the inertia weight w is fixed during the execution on the algorithm [17]. Commonly used values for constant weight inertia are in the interval $[0.8, 1.2]$.
- **Linear**: In this strategy, an interval $[w_{min}, w_{max}]$ is defined. The inertia weight value in each iteration decreases linearly from w_{max} to w_{min} [19]. Using this strategy, the diversification capability of the algorithm is enhanced during the first iterations of the method, when the whole search space must be explored. During the final iterations, when the algorithm refines the final solution, the intensification capability of the method is enhanced.
- **Nonlinear**: This strategy is similar to the linear strategy. However, the weight inertia value at each iteration is obtained through a nonlinear function of the iteration number [20].
- **Random**: In a random strategy, the inertia weight is randomly chosen from an interval $[w_{min}, w_{max}]$ [18]. Uniform distributions are commonly considered, but other distributions can be used.
- **Fuzzy**: In this strategy, a fuzzy inference system is used to define the inertia weight for each iteration. In [24], the authors used the iteration number and the relative velocity to define the inertia weight in a PSO model.
- **Adaptive**: An adaptive strategy updates the inertia weight value based on a set of parameters of the execution such as current best solution and current average solution [21].
- **Chaotic**: In this strategy, a chaotic map is used to generate the inertia weight for the current iteration based on the inertia used in the previous iteration [22].

C. Chaotic Maps

The mathematical definition of chaos is a randomness generated by simple deterministic systems [25]. Randomness is related to the sensitivity of chaotic systems to their initial conditions. In other words, small changes in parameters or initial state of chaotic systems may lead to significant changes in future behaviors.

Chaotic maps are mathematical models used to generate chaotic sequences. Chaotic maps are iterative functions that return, in each iteration, an output value that is a function of the output obtained in the last iteration. The sequence of values generated by a chaotic map is called an orbit. Chaotic maps have the following characteristics [12]:

- The rule of generating the sequence of numbers is deterministic;
- The orbits are non-periodic;
- The orbits are bounded (the chaotic variables assume a value between an upper and a lower limit); and
- The sequence has a sensitive dependence on the initial condition.

Different chaotic maps have been reported in the literature. Since different maps may lead to different results, a set of chaotic maps must be investigated to find the best one for the problem under consideration. In this paper, nine different chaotic maps are considered. These maps are presented in Table I [26], [27].

IV. Proposed Optimization Method

In this section, we present the proposed chaotic inertia weight TLBO algorithm, denoted by TLBOCWI. As mentioned earlier, high inertia weight values increases the diversification capability of the algorithm, while low inertia weight values increase its intensification capability. So, in the proposed algorithm, we use a chaotic sequence that generates a value $z_i \in [0, 1]$ for the i-th iteration. Then, the lower bound and the upper bound inertia weights for the i-th iteration, denoted by LB_i and UB_i, respectively, are computed. Finally, the inertia weight w_i for the i-th iteration is computed according to (8).

$$w_i = LB_i + z_i \cdot (UB_i - LB_i) \tag{8}$$

In each iteration of TLBOCWI, the update solution $X^{\star}_{ki}$ during the Teacher Phase is computed according to (9).

$$X^{\star}_{ki} = w_i \cdot X_{ki} + D_i \tag{9}$$

Based on experimental observations, in this paper we use a lower bound that decreases linearly from 0.7 to 0.4, and an upper bound that decreases linearly from 1.5 to 0.6. Fig. 3 shows a sequence of inertia weights obtained with the upper and lower bounds aforementioned. In this example, a Logistic chaotic map with an initial value $z_1 = 0.8$ was used. We used an initial value $z_1 = 0.8$ in all experiments conducted in this paper. This value has been adopted to initiate chaotic sequences in other works [28].

A. Solution Representation

The candidate solution X associated with each student in TLBO is represented by a vector with $n_A + n_Q$ elements. Each element x_i of X is a real number. Elements $x(1)$ to $x(n_A)$ corresponds to the priority of repair actions A_1 to A_{n_A}, respectively. Elements $x(n_A + 1)$ to $x(n_A + n_Q)$ corresponds to the priority of diagnostic questions Q_1 to Q_{n_Q}, respectively. The conversion of vector X into a troubleshooting strategy is

TABLE I
CHAOTIC MAPS

Number	Name	Equation
1	Logistic map	$z(t+1) = 4z(t) \cdot (1 - z(t))$
2	PWLCM	$z(t+1) = \begin{cases} z(t)/0.7 & ,0 < z(t) \leq 0.7 \\ (1 - z(t)) \cdot (1 - 0.7) & ,0.7 < z(t) \leq 1 \end{cases}$
3	Sine map	$z(t+1) = \sin(\pi z(t))$
4	Tent map	$z(t+1) = \begin{cases} z(t)/0.4 & ,0 < z(t) \leq 0.4 \\ (1 - z(t))/0.6 & ,0.4 < z(t) \leq 1 \end{cases}$
5	Bernoulli map	$z(t+1) = \begin{cases} z(t)/0.6 & ,0 < z(t) \leq 0.6 \\ (z(t) - 0.6)/0.4 & ,0.6 < z(t) < 1 \end{cases}$
6	Chebyshev map	$z(t+1) = \cos(0.5\cos^{-1} z(t))$
7	ICMIC	$z(t+1) = \sin(70/z(t))$
8	Cubic map	$z(t+1) = 2.59z(t) \cdot (1 - z^2(t))$
9	Singer map	$z(t+1) = 1.073[7.86z(t) - 23.31z^2(t) + 28.75z^3(t) - 13302875z^4(t)]$

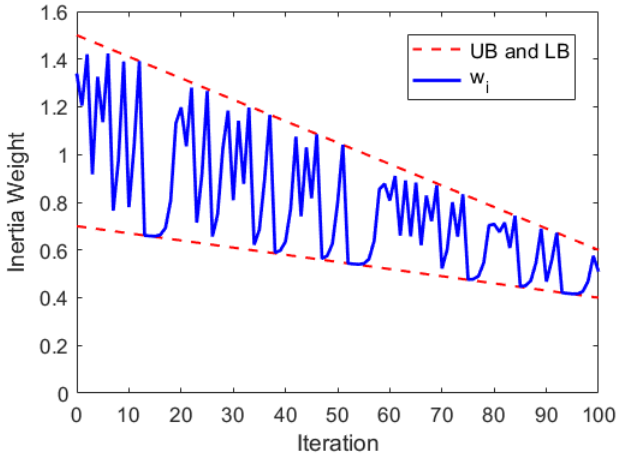

Fig. 3. Example of a chaotic inertia weight sequence using a Logistic map

made by sorting the elements of X. Actions or questions with higher numbers are executed first. The strategy obtained after the conversion is evaluated according to (1).

For illustration purposes, consider a troubleshooting optimization problem with $n_A = 4$ and $n_Q = 2$. The vector X associated with one of the students is presented in Table II. It can be seen that the last element of X is the one with the highest value. Since the last element represents the priority of question Q_2, the resulting troubleshooting strategy will start with question Q_2. Following this procedure, the strategy obtained is $S = \{Q_2, A_4, A_1, Q_1, A_3, A_2\}$.

TABLE II
CONVERSION OF VECTOR X INTO A TROUBLESHOOTING STRATEGY

j	1	2	3	4	5	6
$x(j)$	6.22	1.08	2.59	7.67	5.45	8.65
A/Q	A_1	A_2	A_3	A_4	Q_1	Q_2

V. NUMERICAL EXPERIMENTS

This section presents the results obtained in numerical experiments conducted to evaluate the application of the proposed method in three different troubleshooting models. Experiments using the TLBO algorithm in its original formulation are also conducted for comparison purposes. Table III summarizes the main characteristics of each model. The topology of Models 1, 2 and 3 are shown in Figs. 4, 5 and 6, respectively.

Table IV presents the probability associated with each failure mode in each troubleshooting model. Table V shows the costs associated with each repair action, diagnostic question and cluster in each model.

TABLE III
TROUBLESHOOTING MODELS

	Model 1	Model 2	Model 3
Failures	6	10	15
Actions	5	10	16
Questions	2	4	6
Clusters	2	4	6

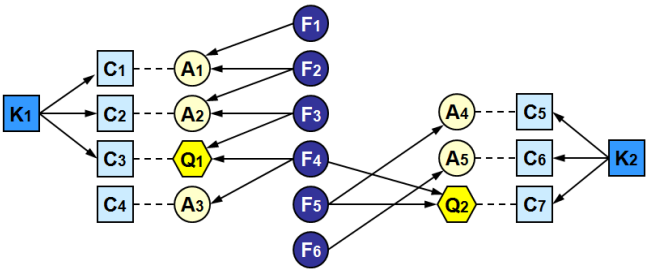

Fig. 4. Troubleshooting model 1

A. Parameter Settings

The performance of metaheuristic algorithms dependents on the choice of parameter values. In TLBO, the population size (denoted by PS) and maximum number of generations (denoted by GN) are the parameters to be defined. A design of experiments (DOE) approach was used in this paper to define good values for these parameters. DOE is an investigative

TABLE IV
FAILURE PROBABILITIES FOR EACH MODEL

	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
Model 1	10%	5%	20%	30%	15%	20%	-	-	-	-	-	-	-	-	-
Model 2	1%	5%	10%	15%	8%	4%	13%	17%	16%	11%	-	-	-	-	-
Model 3	1%	5%	10%	15%	2%	4%	13%	14%	1%	11%	2%	7%	10%	1%	4%

TABLE V
COSTS FOR REPAIR ACTIONS, DIAGNOSTIC QUESTIONS AND CLUSTERS FOR EACH MODEL

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$C(A_i)$	10	20	30	40	50	40	35	5	20	15	20	30	70	10	20	15
$C(Q_i)$	5	10	30	10	30	20	-	-	-	-	-	-	-	-	-	-
$C(K_i)$	10	40	20	50	10	30	-	-	-	-	-	-	-	-	-	-

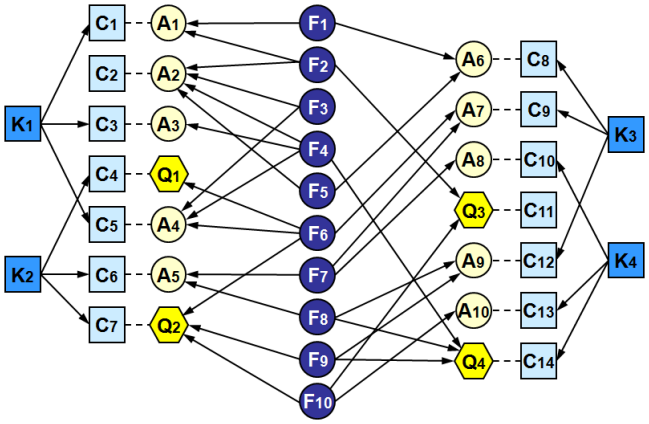

Fig. 5. Troubleshooting model 2

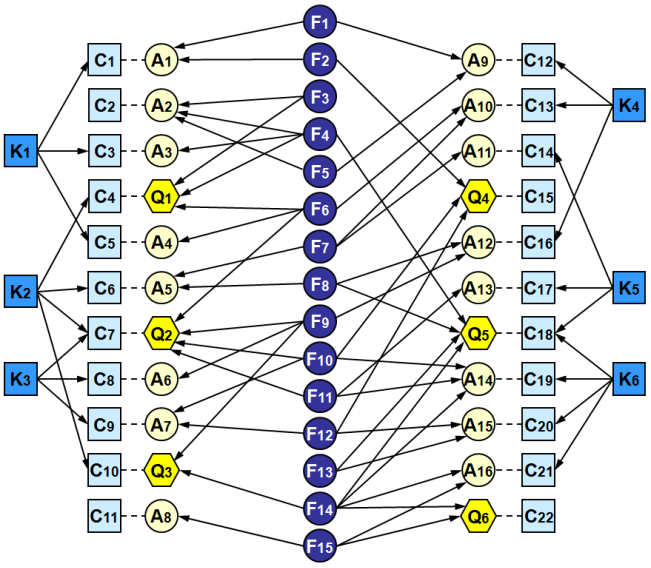

Fig. 6. Troubleshooting model 3

approach used to evaluate the effect of multiple factors on a process [29].

The original TLBO and troubleshooting model 2 were used in the DOE approach. Four different candidate values were considered for GN: 100, 200, 300, and 400. We defined the population size as a function of the solution size, which for the troubleshooting problem is the sum of the number of repair actions (N_a) and the number of diagnostic questions (N_q). The population size is obtained by multiplying the sum $N_a + N_q$ by a factor h. Four candidate values were considered for factor h: 1, 2, 3, and 4. A full factorial DOE layout was used. For each combination, a Monte Carlo approach with 30 repetitions was adopted.

A statistical analysis of variance (ANOVA) was conducted to determine which parameter effects are significant. Table VI shows the results observed. A significance level of 5% is adopted for each main effect and interactions. In Table VI, df are the degrees of freedom, SS is the sum of squares, MS is the mean square, F is the F-test statistics and p is the probability value used to test the null hypothesis that a parameter effect is not significant. The values computed for p are lower than 5%, so we can conclude that both factors are significant.

Tables VII and VIII show the average ECR computed for each candidate value of GN and h, respectively. Based on the results, the final values adopted for GN and for factor h were 300 and 3, respectively.

TABLE VI
ANOVA RESULTS

Source	SS	df	MS	F	p
GN	5.45	3	1.82	5.28	0.0225
h	29.28	3	9.76	28.36	0.0001
Error	3.10	9	0.34		
Total	37.83	15			

TABLE VII
AVERAGE ECR FOR EACH GN VALUE

GN	100	200	300	400
Avg. ECR	202.20	201.04	200.74	200.85

TABLE VIII
AVERAGE ECR FOR EACH h VALUE

h	1	2	3	4
Avg. ECR	203.24	201.62	199.84	200.13

B. Simulation Results

Ten models were considered during the experiments: the original TLBO and nine chaotic inertia weight variants. Each proposed variant used a different chaotic map, as presented in Table I. For each algorithm and each model, a Monte Carlo approach with 150 repetitions was used. All the experiments reported in this paper were carried out on a personal computer with Intel® Core™ i3, 1.9 GHz processor and 4GB RAM. The algorithms were coded in Matlab®.

Tables IX, X, and XI show the results obtained with each algorithm for troubleshooting models 1, 2, and 3, respectively. In terms of average result, the proposed chaotic inertia weight algorithm presented a better performance with eight chaotic maps for model 1, and with seven chaotic maps for models 2 and 3. All chaotic maps outperformed the original TLBO in at least one model. Also, six chaotic maps outperformed the original TLBO in all models.

For models 1 and 2, all algorithms found the optimal solution in at least one Monte Carlo repetition. For model 3, the best known ECR is 206.1. The proposed model with the sine was the only algorithm that could not find this solution during the simulations.

The worst result found by the original TLBO was outperformed by the proposed algorithm with five chaotic maps for model 1, two chaotic maps for model 2, and six chaotic maps for model 3.

The best troubleshooting strategies are presented below:

- **Model 1**: $S_1^* = \{Q_1, A_1, A_3, Q_2, A_5, A_2, A_4\}$
- **Model 2**: $S_2^* = \{Q_4, A_9, A_{10}, A_2, A_1, A_4, A_6, A_7, A_8, A_3, Q_1, Q_3, A_5, Q_2\}$
- **Model 3**: $S_3^* = \{Q_4, A_2, Q_1, Q_6, A_3, A_{12}, A_5, A_9, A_8, A_{11}, A_{10}, A_1, A_7, A_{15}, A_{16}, Q_5, Q_3, A_{14}, A_4, A_6, Q_2, A_{13}\}$

TABLE IX
SIMULATION RESULTS FOR MODEL 1

Algorithm	Average	Best	Worst	Sim. Time
TLBO	78.85	78.50	84.00	2.02
TLBOCWI1	**78.68**	78.50	84.00	2.11
TLBOCWI2	**78.66**	78.50	**80.50**	2.15
TLBOCWI3	**78.72**	78.50	89.50	2.09
TLBOCWI4	**78.75**	78.50	**82.00**	2.10
TLBOCWI5	**78.80**	78.50	**82.50**	2.08
TLBOCWI6	**78.67**	78.50	**81.50**	2.06
TLBOCWI7	78.96	78.50	90.50	2.09
TLBOCWI8	**78.80**	78.50	89.00	2.13
TLBOCWI9	**78.67**	78.50	**80.50**	2.13

TABLE X
SIMULATION RESULTS FOR MODEL 2

Algorithm	Average	Best	Worst	Sim. Time
TLBO	201.81	199.00	215.60	9.08
TLBOCWI1	**200.86**	199.00	218.50	9.54
TLBOCWI2	202.71	199.00	217.40	9.29
TLBOCWI3	201.91	199.00	215.60	9.17
TLBOCWI4	**201.56**	199.00	215.60	9.22
TLBOCWI5	**201.27**	199.00	**214.95**	9.99
TLBOCWI6	**200.82**	199.00	**215.00**	9.71
TLBOCWI7	**201.28**	199.00	215.60	9.50
TLBOCWI8	**200.75**	199.00	215.60	9.37
TLBOCWI9	**200.80**	199.00	215.60	9.66

TABLE XI
SIMULATION RESULTS FOR MODEL 3

Algorithm	Average	Best	Worst	Sim. Time
TLBO	220.85	206.10	257.40	32.08
TLBOCWI1	**219.51**	206.10	262.60	35.06
TLBOCWI2	222.95	206.70	263.30	33.71
TLBOCWI3	221.41	206.10	**251.40**	34.04
TLBOCWI4	**219.86**	206.10	**252.80**	33.12
TLBOCWI5	**218.99**	206.10	**250.45**	32.62
TLBOCWI6	**217.81**	206.10	**247.80**	34.33
TLBOCWI7	**218.66**	206.10	262.00	32.78
TLBOCWI8	**219.54**	206.10	**245.00**	32.28
TLBOCWI9	**217.56**	206.10	**246.35**	32.75

VI. CONCLUSIONS

This paper proposed a chaotic inertia weight algorithm to solve the troubleshooting optimization problem. Three different troubleshooting models were used in the numerical experiments. Nine different chaotic maps were considered. The performance of the proposed algorithm was compared with the original TLBO.

When compared with the original TLBO, the proposed model presented a better performance in terms of average result with eight chaotic maps for model 1, and with seven chaotic maps for models 2 and 3. All the chaotic maps considered in this paper outperformed the original TLBO for at least one model. The increase in simulation time required by the proposed approach in comparison with the original TLBO was, on average, 4.18% for model 1, 4.56% for model 2, and 4.15% for model 3. The additional simulation time remained stable for the different models.

The optimal troubleshooting strategy depends on the probability of each failure mode to cause a system failure event. The use of information generated by fault detection and isolation algorithms can improve the efficiency of the troubleshooting process.

Future research may extend the scope of this paper by comparing the performance of the proposed algorithm in other instances of the troubleshooting optimization problem. Another opportunity is to investigate different approaches for defining the lower and upper bonds for the inertia weight for each execution.

ACKNOWLEDGMENT

The authors acknowledge the support of the Brazilian National Council for Scientific and Technological Development - CNPq (grant 423023/2018-7).

REFERENCES

[1] V. Lín, "Decision-theoretic troubleshooting: Hardness of approximation," *International Journal of Approximate Reasoning*, vol. 55, pp. 977–988, 2014.

[2] M. Vomlelová, "Complexity of decision-theoretic troubleshooting," *International Journal of Intelligent Systems*, vol. 18, no. 2, pp. 267–277, 2003.

[3] H. Langseth and F. V. Jensen, "Heuristics for two extensions of basic troubleshooting," *Frontiers in Artificial Intelligence and Applications*, vol. 66, pp. 80–89, 2001.

[4] F. V. Jensen, U. Kjærulff, B. Kristiansen, H. Langseth, C. Skaanning, J. Vomlel, and M. Vomlelová, "The SACSO methodology for troubleshooting complex systems," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 15, no. 4, pp. 321–333, 2001.

[5] M. Vomlelová and J. Vomlel, "Troubleshooting: NP-hardness and solution methods," *Soft Computing*, vol. 7, no. 5, pp. 357–368, 2003.

[6] T. J. Ottosen, "Solutions and heuristics for troubleshooting with dependent actions and conditional costs," Ph.D. dissertation, Aalborg University, 2011.

[7] W. O. L. Vianna, L. R. Rodrigues, T. Yoneyama, and D. I. Mattos, "Troubleshooting optimization using multi-start simulated annealing," in *10th Annual IEEE Systems Conference, Orlando, FL, USA, 18-21 April 2016*, 2016, pp. 1–6.

[8] R. V. Rao, D. P. Vakharia, and V. J. Savsani, "Teaching-learning-based optimization:a novel method for constrained mechanical design optimization problems," *Computer-Aided Design*, vol. 43, pp. 303–315, 2011.

[9] R. V. Rao and V. Patel, "An improved teaching-learning-based optimization algorithm for solving unconstrained optimization problems," *Scientia Iranica*, vol. 20, pp. 710–720, 2013.

[10] D. Chen, F. Zou, Z. Li, J. Wang, and S. Li, "An improved teaching-learning-based optimization algorithm for solving global optimization problem," *Information Sciences*, vol. 297, pp. 171–190, 2015.

[11] R. V. Rao and V. Patel, "An elitist teaching-learning based optimization algorithm for solving complex constrained optimization problems," *International Journal of Industrial Engineering Computations*, vol. 3, pp. 535–560, 2012.

[12] E. Emary and H. M. Zawbaa, "Impact of chaos functions on modern swarm optimizers," *Plos One*, vol. 11, no. 7, pp. 1–26, 2016.

[13] M. Z. M. Khairuzzaman, I. Musirin, S. S. Izwan, and T. Bouktir, "Chaos embedded symbiotic organisms search technique for optimal FACTS device allocation for voltage profile and security improvement," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 8, no. 1, pp. 146–153, 10 2017.

[14] I. Brajević and P. Stanimirović, "An improved chaotic firefly algorithm for global numerical optimization," *International Journal of Computational Intelligence Systems*, vol. 12, pp. 131–148, 2018.

[15] G. Kaur and S. Arora, "Chaotic whale optimization algorithm," *Journal of Computational Design and Engineering*, vol. 5, no. 3, pp. 275–284, 2018.

[16] A. Farah, T. Guesmi, H. H. Abdallah, and A. Ouali, "A novel chaotic teaching-learning-based optimization algorithm for multi-machine power system stabilizers design problem," *International Journal of Electrical Power & Energy Systems*, vol. 77, pp. 197–209, 2016.

[17] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, 1998, pp. 69–73.

[18] R. C. Eberhart and Y. Shi, "Tracking and optimizing dynamic systems with particle swarms," in *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*, vol. 1, 2001, pp. 94–100.

[19] J. Xin, G. Chen, and Y. Hai, "A particle swarm optimizer with multi-stage linearly-decreasing inertia weight," in *2009 International Joint Conference on Computational Sciences and Optimization*, vol. 1, 2009, pp. 505–508.

[20] R. F. Malik, T. A. Rahman, S. Z. M. Hashim, and R. Ngah, "New particle swarm optimizer with sigmoid increasing inertia weight," *International Journal of Computer Science and Security*, vol. 1, no. 2, pp. 35–44, 2007.

[21] A. K. Shukla, P. Singh, and M. Vardhan, "An adaptive inertia weight teaching-learning-based optimization algorithm and its applications," *Applied Mathematical Modelling*, vol. 77, no. 1, pp. 309–326, 2020.

[22] C.-H. Yang, Y.-H. Cheng, L.-Y. Chuang, and C.-H. Yang, "Chaotic inertia weight particle swarm optimization for PCR primer design," *Journal of Systemics, Cybernetics and Informatics*, vol. 11, no. 3, pp. 44–49, 2013.

[23] J. C. Bansal, P. K. Singh, M. Saraswat, A. Verma, S. S. Jadon, and A. Abraham, "Inertia weight strategies in particle swarm optimization," in *2011 Third World Congress on Nature and Biologically Inspired Computing*, 2011, pp. 633–640.

[24] P. Yadmellat, S. M. A. Salehizadeh, and M. B. Menhaj, "A new fuzzy inertia weight particle swarm optimization," in *2009 International Conference on Computational Intelligence and Natural Computing*, vol. 1, 2009, pp. 507–510.

[25] R. Sheikholeslami and A. Kaveh, "A survey of chaos embedded metaheuristic algorithms," *International Journal of Optimization in Civil Engineering*, vol. 3, no. 4, pp. 617–633, 2013.

[26] I. Fister, M. Perc, S. M. Kamal, and I. Fister, "A review of chaos-based firefly algorithms: Perspectives and research challenges," *Applied Mathematics and Computation*, vol. 252, pp. 155–165, 2015.

[27] M. Mitić, N. Vuković, M. Petrović, and Z. Miljković, "Chaotic metaheuristic algorithms for learning and reproduction of robot motion trajectories," *Neural Computing and Applications*, vol. 30, pp. 1065–1083, 2018.

[28] C.-H. Yang, Y.-H. Cheng, L.-Y. Chuang, and C.-H. Yang, "Chaotic inertia weight particle swarm optimization for PCR primer design," *Journal of Systemics, Cybernetics and Informatics*, vol. 11, no. 3, pp. 44–49, 2013.

[29] D. C. Montgomery, *Design and Analysis of Experiments*, 6th ed. John Wiley & Sons, Inc., 2005.