





## An Accelerated Crow Search Algorithm for Multifactorial Optimization in Vehicle Routing Problem

Asri Bektı Pratiwi<sup>1,2\*</sup>, Claryta Putri Dedyana Wati<sup>1</sup>, Alfonsus Erhan Cahyana<sup>1</sup>, Auli Damayanti<sup>1,2</sup>,  
Edi Winarko<sup>1,2</sup>, Kamal Z. Zamli<sup>3</sup>

<sup>1</sup> Department of Mathematics, Faculty of Science and Technology, Universitas Airlangga, Surabaya 60115, Indonesia

<sup>2</sup> Research Group of Operations Research and Computation, Faculty of Science and Technology, Universitas Airlangga, Surabaya 60115, Indonesia

<sup>3</sup> Faculty of Computing, College of Computing and Applied Science, Universiti Malaysia Pahang, Pahang 26600, Malaysia

Corresponding Author Email: [asri.bekti@fst.unair.ac.id](mailto:asri.bekti@fst.unair.ac.id)

Copyright: ©2025 The authors. This article is published by IETA and is licensed under the CC BY 4.0 license (<http://creativecommons.org/licenses/by/4.0/>).

<https://doi.org/10.18280/mmep.121135>

### ABSTRACT

**Received:** 26 September 2025

**Revised:** 11 November 2025

**Accepted:** 20 November 2025

**Available online:** 30 November 2025

#### **Keywords:**

*chaotic map, crow search algorithm, multifactorial optimization, routing problem, transfer learning*

This paper presents an efficient multifactorial evolutionary algorithm (MFEA), specifically the enhanced chaotic crow search (CS) algorithm, for addressing multifactorial optimization (MFO), particularly the variants of the Vehicle Routing Problem (VRP). The MFO framework involves multiple tasks, each defined within its own search space but all sharing a common representation. The multifactorial algorithm aims to efficiently obtain the optimal solution for each assigned task simultaneously within a single execution. Our proposed algorithm employs a chaotic map to enhance solution exploration by integrating it into the crow position updating process. Transfer learning is used to enhance the efficiency of learning characteristics across multiple tasks. Computational experiments were carried out to evaluate the algorithm's performance in terms of the best solution obtained and computational time, where multiple tasks are executed simultaneously and independently. Three models were used: the capacitated VRP, VRP with time windows, and VRP with simultaneous pickup and delivery. The findings indicate that increasing the population size generally leads to better objective function values, while more iterations also contribute to further improvement. Moreover, the optimal parameter configuration for multi-task optimization depends on the number of tasks being addressed. To ensure fairness and maintain optimal effectiveness, each additional task can be accompanied by a proportional increase in both the population size and the number of iterations.

## 1. INTRODUCTION

Through the process of optimization, one can discover the best possible solution to a wide variety of problems. Over the years, research has been conducted on optimizing a single optimization problem at a time. An optimization on more than one problem at a time was introduced by Gupta et al. [1], dealing with multiple optimizations in a single execution called multifactorial optimization (MFO). The MFO has multiple tasks, each of which has its own search space, but they proceed in a common representation. Each problem or task contributes a unique factor that influences how individuals evolve within a single population. The objective of MFO is to identify the optimal solution for each task efficiently, rather than determining the best trade-off among multiple objective functions. Consequently, research related to efficient methods or techniques for optimizing each objective function in the corresponding task has become a frequently discussed issue.

Knowledge transfer techniques, or transfer learning between tasks, are highly beneficial in reducing the need for

large training datasets. The multi-task learning framework adopts transfer learning techniques to concurrently learn multiple tasks, despite their differences. A common strategy in multitask learning is to identify and leverage shared features that can support the learning process of each individual task [2]. This approach leverages knowledge from one task to improve performance on a separate task that shares similar characteristics [3]. Transmission processes involving transfer learning can be found in both the multi-population evolutionary framework (MPEF) and the multifactorial evolutionary algorithm (MFEA), which are driven by the behavior of genes during their transmission from parents to offspring [1, 4].

Studies using metaheuristic algorithms for MFO have been conducted, including particle swarm optimization in MFEA, by implementing it on benchmark functions [5]. The functions used are optimization functions with continuous decision variables. The simulation is conducted on pairs of functions with the given dimensions. Meanwhile, Yokoya et al. [6] employed the artificial bee colony in order to work with the car structure design problem formulated as the constrained bi-

objective optimization problem. Both studies consider continuous decision variables, and to the best of our knowledge, there has been no research on applying MFO to problems with binary decision variables.

The classical Vehicle Routing Problem (VRP), also known as the capacitated VRP, is a generalization of the Travelling Salesman Problem with an additional capacity constraint [7, 8]. The capacitated VRP, VRP with time windows, and VRP with simultaneous pickup and delivery have different numbers of constraints, which may influence their computational time. However, the MFO approach can solve all three models because they share a common solution representation in the form of binary variables. When represented in metaheuristic algorithms, these three models also have the same solution representation, namely, route sequences. The similarity in solution representation serves as a characteristic that enables transfer learning to be applied in identifying the strengths of each solution across tasks. Given these similarities in solution representation, the selection of an appropriate metaheuristic becomes crucial for effectively solving VRP variants. One promising approach is the crow search (CS) algorithm, which has shown competitive performance in various optimization problems.

The CS algorithm is a metaheuristic optimization method inspired by the intelligent behavior of crows. The CS algorithm demonstrates strong exploratory behavior but struggles with effective exploitation [9, 10]. A study by Nirmal et al. [11] proposed a hybrid approach combining the CS algorithm with the Bald-Eagle Search algorithm. The position update phase of the CS algorithm is retained, as it plays a crucial role in exploitation. Based on these studies, the weakness of the CS algorithm in terms of exploitation can be addressed by focusing on the position update phase. Therefore, further modifications to this phase can enhance the performance of the CS algorithm. The study conducted by Rizk-Allah et al. [12] modified the CS algorithm by introducing more than one scenario, including the use of a chaotic map for the flight length parameter, position memory, and a comparison with awareness probability. The use of a chaotic map in comparison with awareness probability was also carried out by Sayed et al. [13]. In addition, the study integrated the chaotic map into the generation of random real numbers.

In this paper, we improve the CS algorithm by integrating the chaotic map into the update position process as well as incorporating transfer learning in the fitness evaluation stage. The chaotic map is applied to the random number parameter, while transfer learning is performed before the memory phase. A study by Ma et al. [14] compares the performance of various chaotic maps. Simulation results across twelve problem types indicate that the logistic map consistently yields the highest solution quality compared to other chaotic maps, including the Gauss map. The sine and iterative maps also outperform several others, such as the sinusoidal and Liebovitch maps. While the tent and singer maps are slightly less effective than the sine and iterative maps, they still surpass the Gauss, intermittency, and piecewise maps [15]. Based on these findings, we adopt six chaotic maps, i.e., logistic, tent, sine, singer, iterative, and Chebyshev, to improve the solution quality of the CS algorithm.

Our proposed algorithm is applied to the VRP along with its variants, namely the capacitated VRP, the VRP with time windows, and the VRP with simultaneous pickup and delivery, which are considered as separate tasks. The transfer learning

is employed to simultaneously obtain high-quality solutions for all three VRP variants. Our integration between the chaotic map and transfer learning, computational experiments were conducted to evaluate the performance impact of integrating the chaotic map. Furthermore, the effect of transfer learning is examined by comparing multi-task execution in a single run with independent execution. Research questions are proposed and addressed based on the computational results in terms of effectiveness, speed, and appropriate parameter selection.

This paper is organized as follows. Section 2 overviews the model's formulation. Section 3 proposes a chaotic CS algorithm. Sections 4 and 5 present computational experiments, results, and discussion, respectively. Finally, the conclusion can be found in Section 6.

## 2. VEHICLE ROUTING PROBLEM OVERVIEWS

The VRP was first introduced by Pereira and Tavares [16]. Since then, numerous researchers have proposed various types of the VRP. VRP is a type of problem for planning a distribution route where a number of customers are served by a single depot. Deliveries are carried out using vehicles, each with a certain capacity. All customer demands must be fulfilled, and each customer is served by exactly one vehicle and only once [17]. This argument is supported by the definition provided by Van Breedam [18], which states that each customer's demand is known with certainty.

Research on VRP using chaos optimization has been conducted to solve it as a single problem or single-task problem. Among them is a study on capacitated VRP conducted by Shan and Wang [19]. Both studies employed metaheuristic methods modified with a chaotic map to enhance their performance in solving the capacitated VRP. In line with previous studies, Hu et al. [20] addressed a single-task problem by hybridizing a chaotic map into the algorithm to solve VRP with time windows. The study on the VRP simultaneous pickup and delivery was conducted by Hu and Wu [21], who applied chaos theory to the evolutionary algorithm. While chaos-based algorithms focus on enhancing metaheuristic exploration and exploitation, recent studies have turned to deep learning techniques to model and predict routing decisions in VRP. Despite these advances, existing studies have primarily focused on solving a single variant of the VRP. There is still a lack of research that considers multiple VRP variants simultaneously within one optimization framework.

To better understand the complexity of this problem, an overview of the VRP and its variants is presented below. Mathematically, the VRP is represented as a complete graph  $G$ . Let  $G = (V, E)$ , where  $V = \{0, 1, \dots, N\}$  is the set of edges representing customer locations, and  $E = \{(i, j) | i, j \in V, i \neq j\}$  is the set of edges representing the roads connecting customer locations. Node 0 represents the depot, which is the location where vehicles are stored for distribution and also serves as both the starting and ending points of a vehicle's route. The notation  $d_{ij}$  represents the Euclidean distance from customer location  $i$  to  $j$ . The number of vehicles available at the depot is denoted by  $K$ , and each vehicle has a capacity of  $W$ . Each customer  $i$  has a demand of  $q_i$ .

The objective of the VRP is to minimize the travel cost or total distance traveled, and it can be modeled as follows:

$$\min f = \sum_{i=0}^N \sum_{j=0}^N \sum_{k=1}^K d_{ij} x_{ijk} \quad (1)$$

where, the decision variables are given below:

$$x_{ijk} = \begin{cases} 1, & \text{if the vehicle } k \text{ serves node } j \text{ immediately after node } i, \\ 0, & \text{otherwise} \end{cases}$$

$se_i$ : service starting time for customer  $i$ ,

$r_i$ : vehicle load after serving customer  $j$ .

The constraints used are as follows:

1. Each customer is visited exactly once by one vehicle:

$$\sum_{i=0}^N \sum_{k=1}^K x_{ijk} = 1, \forall j = 1, \dots, N, i \neq j \quad (2)$$

2. For every vehicle that visits a customer, it must leave that customer after service:

$$\sum_{i=0}^N x_{ilk} - \sum_{j=0}^N x_{ijk} = 0, \forall k = 1, \dots, K, l = 0, \dots, N \quad (3)$$

3. There are  $K$  vehicles departing from depot:

$$\sum_{j=0}^N x_{0jk} = 1, \forall k = 1, \dots, K \quad (4)$$

4. The total demand served by each vehicle must not exceed its capacity:

$$\sum_{i=1}^N \sum_{j=0}^N q_i x_{ijk} \leq W, \forall k = 1, \dots, K \quad (5)$$

VRP with time windows (VRPTW) is a type of VRP in which time window constraints are considered [22]. Unlike the study by Taş et al. [23], which uses flexible time windows where vehicles are allowed to deviate from customer time constraints within a given tolerance, this research adopts fixed time windows. The time window, denoted as  $[e_i, l_i]$  is a time interval set by the customer during which service must be provided. Here,  $e_i$  is the earliest time a customer can be served, and  $l_i$  is the latest allowable service time. Vehicles are allowed to arrive before  $e_i$ , but arriving after  $l_i$  is not permitted. If a vehicle arrives before  $e_i$ , it must wait until service can begin. If a vehicle arrives after  $l_i$ , the customer will be served by another vehicle [24, 25]. The completion time of service for customer  $i$  is denoted by  $sl_i$ , while travel duration from customer  $i$  to  $j$  is denoted by  $t_{ij}$ . The decision variable  $se_i$  represents the earliest start time of service for customer  $i$ . The additional constraints used in VRPTW are as follows.

5. Ensure that two neighboring customers are served on the same route:

$$sl_i + t_{ij} \leq se_j, \forall i, j = 1, \dots, N \quad (6)$$

6. Service time window constraints for customer:

$$e_i \leq se_i \leq l_i, \forall i = 1, \dots, N \quad (7)$$

In VRP with simultaneous pickup and delivery (VRPSPD), deliveries and pickups are performed at customer locations using vehicles with identical capacities. Therefore, the vehicle load at each customer must be carefully managed to ensure that it does not exceed the vehicle's maximum capacity. The VRPSPD model can be applied in industrial settings, such as in the distribution of bottled drinking water and liquid petroleum gas (LPG). Customers are visited to receive dual services; for example, in LPG distribution, filled LPG cylinders are delivered to customers, while empty cylinders are picked up to be refilled [26]. The constraints of VRPSPD are an extension of the basic VRP with the following additions [27].

7. The total amount of picked up demand from all customers visited by vehicle  $k$  must not exceed the initial load of the vehicle when departing from the depot:

$$r \geq \sum_{i=0}^N \sum_{j=1}^N q_j x_{ijk}, \forall k = 1, \dots, K \quad (8)$$

8. The combined pickup and delivery demand at customer  $j$  by vehicle  $k$  must not exceed the remaining capacity of the vehicle:

$$r_j \geq r - q_j + p_j - M(1 - x_{0jk}) \\ \forall j = 1, \dots, N, \forall k = 1, \dots, K \quad (9)$$

9. The load at customer  $j$  served by vehicle  $k$  must not exceed the remaining capacity after serving customer  $i$ :

$$r_j \geq r_i - q_j + p_j - M \left( 1 - \sum_{k=1}^K x_{ijk} \right) \\ \forall i, j = 1, \dots, N \quad (10)$$

10. The load carried by vehicle  $k$  must not exceed its maximum capacity:

$$r \leq W \quad (11)$$

$$r_j \leq W, \forall j \in N \quad (12)$$

The notations used in the model are described in Table 1.

**Table 1.** Description of notations

Notation	Description
$d_{ij}$	Distance between customer $i$ and customer $j$
$t_{ij}$	Travel duration from customer $i$ to customer $j$
$q_i$	Delivery demand from customer $i$
$p_j$	Pickup demand from customer $j$
$W$	Maximum capacity of the vehicle
$r$	Vehicle load when leaving depot
$K$	Number of vehicles
$N$	Number of customers
$[e_i, l_i]$	Time window for customer $i$
$sl_i$	Service end time at customer $i$

### 3. ACCELERATED CROW SEARCH ALGORITHM

The CS algorithm is an evolutionary optimization algorithm inspired by the social behavior of crows in flocks [9]. Crows

are considered among the most intelligent bird species. They have the ability to observe and remember where other crows hide their food and often return to steal it once the owner has left. Due to their high level of self-awareness, crows tend to relocate their hidden food to avoid being robbed. They also possess excellent memory, enabling them to recall their hiding spots even after several months. The CS algorithm involves two key adjustable parameters: the flight length ( $fl$ ) and the awareness probability ( $AP$ ). The procedure for implementing CS in a  $D$ -dimensional optimization problem using a crow population is described below:

1. Initialization: Initialize the position vector  $x_{i,d}$  and the memory  $m_{i,d}$  of each crow, where  $i = 1, \dots, S$  and  $d = 1, \dots, D$ . Since crows have no prior experience at the start, their initial memory is assumed to be their current position.

2. Fitness evaluation: Calculate the fitness value of each crow's current position.

3. Position update: Randomly select another crow, say the  $j$ -th crow, to follow. Then, update the position of the  $i$ -th crow using the following rule:

$$x_i^{t+1} = \begin{cases} x_i^t + fl \cdot rand_i \cdot (m_j^t - x_i^t), & \text{if } rand_j \geq AP \\ \text{a random position}, & \text{otherwise} \end{cases} \quad (13)$$

where,  $rand$  is a uniformly distributed random number in the interval  $[0,1]$ ,  $fl_i$  is the flight length of crow, and  $AP$  is the awareness probability of crow. This condition reflects that if crow  $j$  is aware of being followed, it will move randomly within the search space to mislead the follower. Consequently, crow  $i$  will move randomly within the search space.

4. Fitness re-evaluation: Compute the fitness of the updated positions for each crow.

5. Memory update: If the new position of crow  $i$  yields a

better fitness value than the memorized one, update the memory  $m_i$  accordingly.

As explained in the first section, the CS algorithm has simple steps, making it easy to implement for various optimization problems. This simplicity results in shorter runtime and less computational effort. However, the algorithm lacks a proper balance between exploration and exploitation processes, which necessitates modifications to improve its performance.

Chaos refers to deterministic yet seemingly random behavior observed in nonlinear dynamical systems that exhibit sensitive dependence on initial conditions. Although the behavior appears random and unpredictable, it is governed by deterministic rules [15, 28]. A chaotic state in a discrete-time dynamical system can be presented as follows.

$$x_{k+1} = f(x_k), 0 < x_k < 1, k = 0, 1, 2 \dots \quad (14)$$

where,  $\{x_k\}_{k=0}^{\infty}$  represents a chaotic sequence or chaotic state, which can be used as a spread-spectrum or pseudo-random number sequence. Chaotic sequences are computationally efficient and require minimal memory, as they only depend on a chaotic map and an initial condition.

This paper aims to incorporate logistic, tent, sine, singer, iterative, and Chebyshev maps to enhance the solution quality of the CS algorithm. The selected chaotic maps are integrated into the CS algorithm. Each chaotic map starts with an initial value randomly chosen between 0 and 1. However, because the early iterations of chaotic systems typically have minimal impact, a fixed initial value of  $x_0 = 0.3$  is used across all chaotic maps to ensure a fair performance comparison. Detailed descriptions of each chaotic map are provided in Table 2.

**Table 2.** Chaotic maps values used in the experiment

Chaotic Maps	Description	Parameter
Logistic	$x_{k+1} = ax_k(1 - x_k)$	$a = 4, x_0 = 0.3$
Iterative	$x_{k+1} = \sin\left(\frac{a\pi}{x_k}\right)$	$a = 0.7, x_0 = 0.3$
Sine	$x_{k+1} = \frac{a}{4} \sin(\pi x_k)$	$a = 4, x_0 = 0.3$
Tent	$x_{k+1} = \begin{cases} \frac{x_k}{0.7}, & \text{if } x_k < 0.7 \\ \frac{10}{3}(1 - x_k), & \text{if } x_k \geq 0.7 \end{cases}$	$x_0 = 0.3$
Singer	$x_{k+1} = \mu(7.86x_k - 23.31x_k^2 + 28.75x_k^3 - 13.3x_k^4)$	$\mu = 1.07, x_0 = 0.3$
Chebyshev	$x_{k+1} = a \cos^{-1}(x_k)$	$a = 0.5, x_0 = 0.3$

The modification to the CS algorithm using a chaotic map is applied to Eq. (15). In the original CS algorithm,  $rand_j$  is uniformly distributed random number between 0 and 1 for crow  $j$ , whereas in this improved CS,  $chaos_i^t$  is a chaotic number between 0 and 1 for crow  $i$  at iteration  $t$ . Therefore, the equation can be rewritten as follows:

$$x_i^{t+1} = \begin{cases} x_i^t + fl \cdot chaos_i^t \cdot (m_j^t - x_i^t), & \text{if } rand_j \geq AP \\ \text{a random position}, & \text{otherwise} \end{cases} \quad (15)$$

In general, MFO aims to obtain an optimal solution for several tasks simultaneously within a single execution. To assess the crows, several attributes related to each individual are defined as follows [29]:

Definition 1 (Factorial Cost): The factorial cost of an

individual  $p_i$ , for  $S$  individuals, on task  $T_j$  refers to the objective function value  $f_j$  of the candidate solution  $p_i$ , and is presented as  $\psi_{ij}$ .

Definition 2 (Factorial Rank): The factorial rank of  $p_i$  on task  $T_j$  is the position of  $p_i$  in the ascendingly sorted list of objective values for that task, and is denoted by  $\delta_i^j$ .

Definition 3 (Skill Factor): The skill factor refers to the index of the task that an individual is assigned. The skill factor of  $p_i$  is defined as:

$$\tau_i = \arg \min_{j \in \{1, 2, \dots, S\}} \delta_i^j \quad (16)$$

Definition 4 (Scalar Fitness): The scalar fitness of  $p_i$  is defined as the inverse of its best factorial rank across all tasks,

given by:

$$\varphi_i = \frac{1}{\min_{j \in \{1,2,\dots,S\}} \delta_i^j} \quad (17)$$

Transfer learning is a paradigm that enables the transfer of knowledge across different optimization problems. Similarly, transfer learning in machine learning leverages knowledge acquired from related tasks to address a new task, rather than learning it from scratch. Based on the definitions above, the algorithm of transfer learning can be summarized as follows:

---

**Algorithm 1:** Transfer learning algorithm

---

Input: solution, fitness

Output: new solutions, new fitness

Update solution using metaheuristic search operator

Rank in ascending order the factorial cost based on the updated solution

Compute skill factor,  $\tau_i = \arg \min_{j \in \{1,2,\dots,S\}} \delta_i^j$

Compute scalar fitness,  $\varphi_i = \frac{1}{\min_{j \in \{1,2,\dots,S\}} \delta_i^j}$

If the updated scalar fitness is better than the previous, then accept the updated solution

---

The accelerated CS algorithm represents a modification of the conventional CS algorithm, in which the position updating mechanism is enhanced through the integration of a chaotic map. This algorithm is employed as a solution approach for the MFO, supported by a transfer learning framework. The procedure begins with the ranking of fitness values, followed by the computation of scalar fitness, which is subsequently utilized to update the crow memory. The scalar fitness is obtained using the transfer learning algorithm described in Algorithm 1 above. The complete algorithmic process is summarized in the pseudocode as Algorithm 2.

---

**Algorithm 2:** Accelerated crow search algorithm

---

Input: input data

Output: best solution found

Initialize parameters

Generate initial crow positions

Set initial memory  $\leftarrow$  initial positions

Evaluate initial crow fitness

Rank fitness values

Compute scalar fitness

While  $iter \leq Max_{iter}$  do

    For  $i \leftarrow 1$  to  $popsize$  do

        Update crow position

        Evaluate fitness of modified position

        Rank fitness values

        Compute scalar fitness

        If  $new\ scalar\ fitness > old\ scalar\ fitness$

then

            Update crow memory

        End

    End

End

Print the best solution

---

#### 4. COMPUTATIONAL EVALUATION

This section presents a computational evaluation aimed at

achieving several objectives: (1) to assess the impact of chaotic modifications on the performance of the MFO algorithm; (2) to identify suitable parameter settings for achieving optimal performance; and (3) to evaluate the algorithm's capability in solving multi-task problems compared to single-task problems.

In line with the objectives of this study, computational simulations are conducted with a focus on addressing the following research questions (RQ):

RQ1: How do the parameters affect the outcome of the objective function?

RQ2: What is the impact of the chaotic function on the performance of the MFO?

RQ3: How does the performance of MFO compare to single-task optimization?

RQ4: What are the optimal parameter settings to achieve comparable performance between single-task and multi-task optimization?

RQ5: What is the general computational complexity of the MFO compared to single-task approaches?

RQ6: How does the performance of the proposed algorithm compare to other MFO algorithms?

To assess the effectiveness of the proposed algorithm, the accelerated CS algorithm, in addressing the MFO, the algorithm was implemented in C++ and executed on a personal computer processor, 11th Gen Intel® Core(TM) i3-1115G4 @3.00GHz RAM 8,00 (7,75 GB usable) 64-bit operating system, x64-based processor.

The test instances were derived from a publicly available Solomon benchmark dataset for the VRP, which can be accessed at <http://vrp.galgos.inf.puc-rio.br/index.php/en/> [30]. The dataset includes information on minimum vehicle capacity, customer demand, the locations of depots and customers, specified time windows, and the duration of customer service. A random pickup and delivery quantity is used. The instances used in this study are categorized into three groups: small-scale (25 customers), medium-scale (50 customers), and large-scale datasets (100 customers). The computational experiments in this study were conducted over 10 independent runs, and the best results were reported.

**Table 3.** Comparison results of the objective function values

	Logistics	Iterative	Sine
VRP	623.92	782.2	735.12
VRPTW	760.24	1020.31	968.52
VRPSPD	760.56	790.56	792.41
Time (s)	0.187	0.192	0.21
	Tent	Chebyshev	Singer
VRP	712.1	689.03	701.42
VRPTW	823.54	982.77	906.53
VRPSPD	800	771.52	797.91
Time (s)	0.211	0.212	0.22

Before addressing the proposed research questions, a comparative simulation is conducted to determine the most effective type of chaotic map. The selected chaotic map will be used throughout the subsequent experiments to answer all the research questions. To determine the most suitable type of chaotic map to use, six types were tested on a dataset consisting of 25 customers. The experiments were conducted with 100 iterations, a crow population of 10,  $fl = 2$ , and  $AP = 0.2$ . The performance results of each chaotic map, based on the objective function values, are presented in Table 3. Based on the table, the logistic map demonstrates the best performance compared to the other five maps in solving the

multi-task problem, both in terms of the objective function values for each task and the execution time.

To ensure a fair comparison, different settings for population size ( $popsiz$ ), maximum iterations ( $Max_{iter}$ ), and  $AP$  were used according to its associated research question. Subsequently, to address RQ1, the experiment was conducted by considering three dataset sizes (25; 50; 100) based on the objective function values and execution time for each dataset type. The value of  $fl$  was kept constant at 2. The  $popsiz$  used were 10, 50, and 100; the  $Max_{iter}$  considered were 10, 100, and 1000; and the  $AP$  values tested were 0.2, 0.5, and 0.8.

Meanwhile, for RQ2 and RQ3, the computational results were obtained when implemented on 25 customers using  $popsiz = 10$ ,  $fl = 2$ ,  $AP = 0.2$ , and  $Max_{iter} = 1000$  applied on four different datasets. The computational experiment to answer RQ4 was conducted using  $fl = 2$  and  $AP = 0.2$ , applied to dataset consisting of 100 customers. Variations in the proportion of population size and iteration count were tested, starting with  $popsiz = 50$  and  $Max_{iter} = 20$ , followed by  $popsiz = 100$  and  $Max_{iter} = 40$ , and finally  $popsiz = 150$  and  $Max_{iter} = 60$ . The final set of experiments is conducted to answer RQ6. The experiment was conducted using  $fl = 2$  and  $AP = 0.2$ , applied to dataset consisting of small-scale (25 customers). Proportion of population size and iteration is  $popsiz = 150$  and  $Max_{iter} = 60$ . The accelerated CS algorithm is benchmarked against two MFO algorithms: the MFEA incorporating a particle swarm operator [5] and an artificial bee colony operator [6]. For particle swarm operator, the parameter of adaptive was set to 2,  $\varphi_1 + \varphi_2 = 4.1$ ,  $k = 0.729$ ,  $rmp = 0.3$ . For the artificial bee colony,  $\alpha = 5.0$ ,  $\beta = 0.05$  and  $\gamma = 0.001$ .

## 5. RESULTS AND DISCUSSION

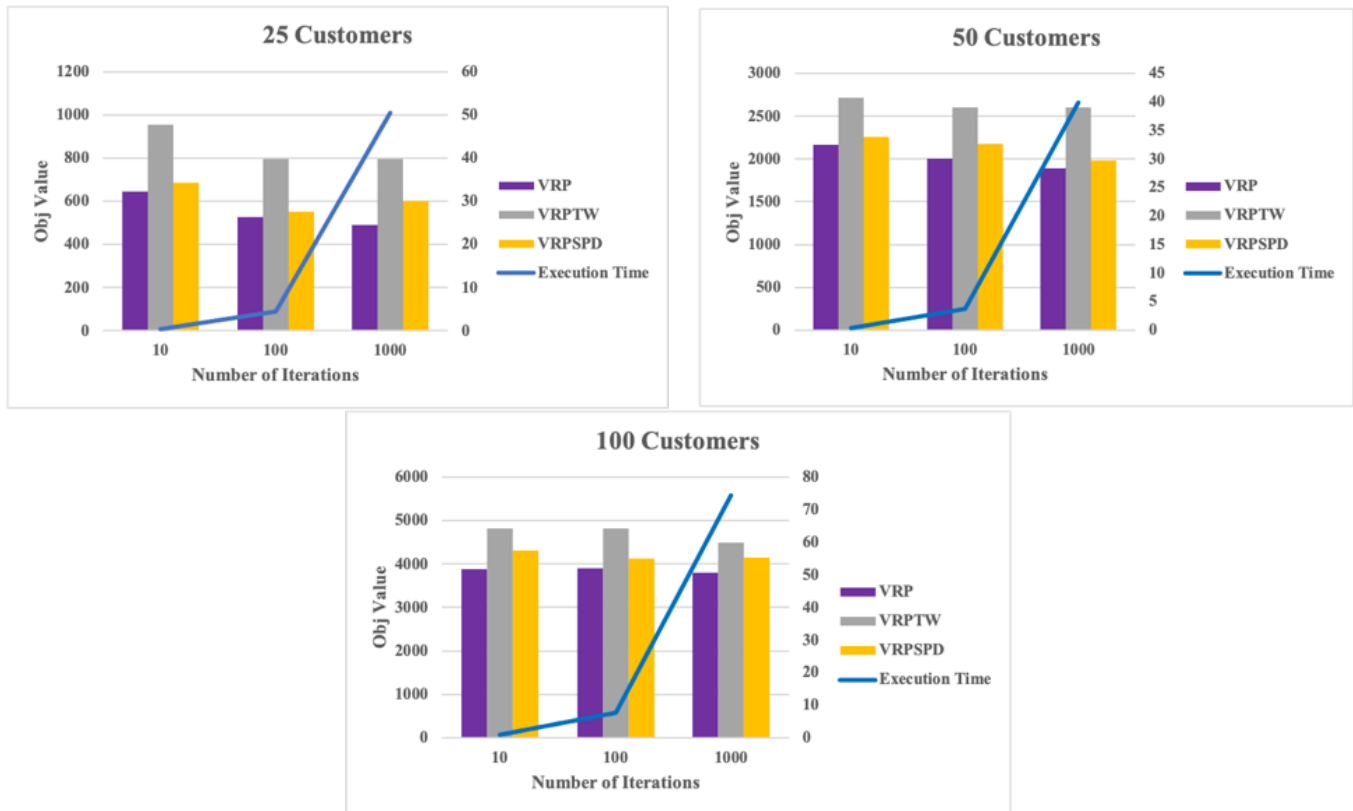
In this section, the influence of parameters in the accelerated CS algorithm on improving performance in terms of both accuracy and speed is discussed. Additionally, the effect of incorporating a chaotic map on the algorithm's exploitation capability is analyzed. Furthermore, this section aims to determine the optimal parameter formulation for solving MFO problems involving a larger number of tasks and more complex constraints. The resulting parameter configuration is then validated by comparing its performance with that of existing MFO algorithms.

Results are organized to correspond with the identified research questions as follows:

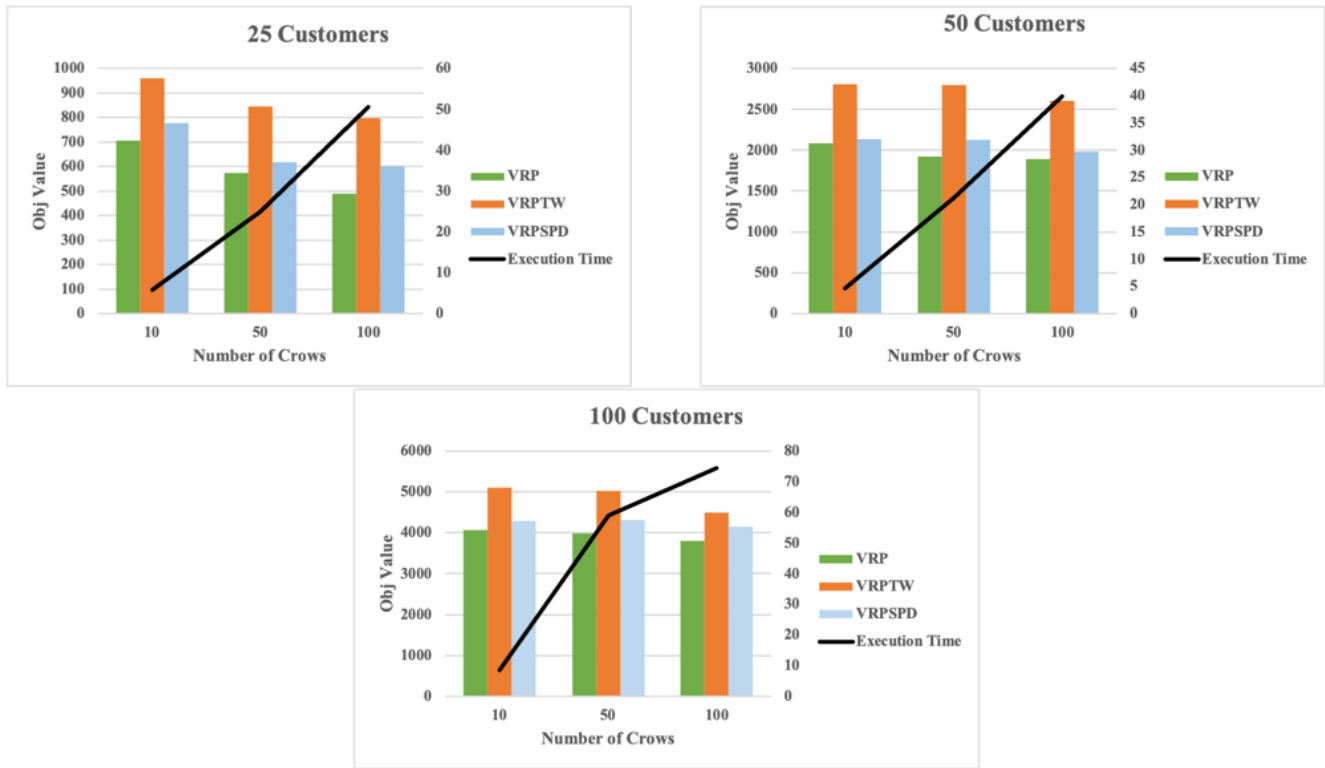
RQ1: How do the parameters affect the outcome of the objective function?

Figure 1 presents the objective function values and execution times as the number of iterations increases on three sizes of dataset. The experiments were conducted using a population size of 100. The results indicate that the number of iterations significantly affects the performance of the objective function value. Additionally, the execution time increases as the number of required iterations becomes larger.

To validate the effect of the number of individuals/crows on the objective function values and execution time, Figure 2 presents the results of experiments conducted on three dataset sizes using 1000 iterations. Based on these figures, an increase in the number of crows leads to improved solution quality across all three tasks. Each task, i.e., VRP, VRPTW, and VRPSD, collectively achieves better objective function values as the number of crows increases. Naturally, this also results in longer execution times when solving the three tasks with a large number of crows.



**Figure 1.** Performance between objective function value vs. execution time when  $Max_{iter}$  increase



**Figure 2.** Performance between objective function value vs. execution time when *popsize* increase

Overall, based on the objective function values obtained from small, medium, and large datasets, there is a tendency indicating that a larger population size leads to better objective function values, a higher number of iterations tends to improve the objective function value. Increasing the number of iterations and individuals enhances the algorithm's ability to explore a wider solution space. In discrete optimization, a larger population size leads to a greater number of possible solution combinations. Moreover, increasing the number of iterations not only expands the exploration scope but also allows the algorithm to continuously refine its best solution through the position updating process.

RQ2: What is the impact of the chaotic function on the performance of MFO?

Next, to answer RQ2, Table 4 presents the impact of modifying the CS algorithm by incorporating a chaotic map, specifically the logistic map. Based on this table, the modification of the CS algorithm with the incorporation of the logistic maps has improved performance in terms of both the objective function value and execution time (in seconds). The logistic map replaces the use of the *rand* operator in each crow modification. Although both produce real numbers in the same range, 0 to 1, the chaotic sequence generated by the logistic map is not repetitive and exhibits self-similar patterns at various scales. The logistic map proves to be the most effective in enhancing the performance of the algorithm compared to the iterative, sine, tent, Chebyshev, and singer maps. Despite its simplicity, the logistic map illustrates how gradual variations in the parameter  $a$  can result in highly complex and chaotic behavior. The logistic formulation evolves into a system that is both intricate and difficult to predict. This enables the algorithm to efficiently avoid being trapped in local optima. These characteristics enhance the exploration capability within the search space, making it well-suited for solving multi-task problems.

**Table 4.** Effect of chaotic map on algorithm performance

	D1			
	VRP	VRPTW	VRPSD	Time (s)
Chaotic CS	541.97	846.308	596.703	78.755
CS	618.223	876.486	693.105	79.266
	D2			
	VRP	VRPTW	VRPSD	Time (s)
Chaotic CS	600.488	879.624	649.315	81.342
CS	611.83	934.759	722.094	87.136
	D3			
	VRP	VRPTW	VRPSD	Time (s)
Chaotic CS	523.253	816.758	600.809	66.501
CS	611.142	878.534	680.03	50.264
	D4			
	VRP	VRPTW	VRPSD	Time (s)
Chaotic CS	577.719	836.854	672.484	59.927
CS	601.577	865.067	591.38	68.057

In addition, selecting an awareness probability ( $AP$ ) value lower than 0.5 results in better algorithmic performance compared to values greater than 0.5. Choosing an  $AP$  value below 0.5, particularly when it approaches zero, causes the position update process to rely more frequently on the chaotic approach rather than on random search.

RQ3: How does the performance of MFO compare to the single-task optimization?

Next, to evaluate the performance of the accelerated CS algorithm in solving MFO, Tables 5 and 6 present the objective function values and execution time (in seconds) of the algorithm when solving three VRP variants (VRP, VRPTW, and VRPSD) simultaneously in a one-time execution and separately in a single-task manner, respectively.

Based on both tables, the execution time (in seconds) required to solve the three problems in a single run is shorter than solving them separately. Moreover, the objective function values obtained are also competitive compared to those



achieved through separate executions. This indicates that the algorithm successfully solves the multi-task problem.

**Table 5.** Objective function values and execution time of multi-task

Dataset	$f$			Time (s)
	VRP	VRPTW	VRSPD	
D1	4196.28	5021.47	4359.34	1.1
D2	4188.37	4773.38	4443.38	0.848
D3	4088.33	4833.97	4464.46	0.892
D4	4243.58	4696.25	4377.01	0.89

**Table 6.** Objective function values and execution time of single-task

Dataset	$f$			Overall Time (s)
	VRP	VRPTW	VRSPD	
D1	4118.29	4914.32	4360.07	
D2	4193.85	4642.58	4308.64	
D3	4727.18	4698.94	4317.28	
D4	4149.16	4513.01	4262.01	
Dataset	Time (s)			Overall Time (s)
	VRP	VRPTW	VRSPD	
D1	0.453	0.711	0.508	1.672
D2	0.351	0.517	0.338	1.206
D3	0.55	0.265	0.359	1.174
D4	0.22	0.371	0.456	1.047

The proposed algorithm, which employs the same solution representation, accelerates the execution process due to the use of transfer learning to retain its best solution. Although the execution in the multi-task setting is fairly competitive in searching for the optimal solution, the algorithm is not yet sufficiently superior in achieving the best result. This limitation arises from the relatively narrow exploration space when solving the three tasks simultaneously. Therefore, to ensure a fairer comparison, increasing the number of individuals or iterations is necessary to expand the exploration space. Therefore, further experiments are required to address RQ4 and RQ5.

RQ4: What are the optimal parameter settings to achieve comparable performance between single-task and multi-task optimization?

Table 7 presents the objective function values and execution times for each task solved independently using the accelerated CS algorithm. The table indicates that solving the VRP yielded the fastest execution time, with VRSPD and VRPTW following closely behind. The execution time for VRSPD was shorter than for VRPTW.

**Table 7.** Objective function values and execution time of one-task

Parameters	Dataset	$f$		
		VRP	VRPTW	VRSPD
$popsiz = 50$ , $Max_{iter} = 20$	D1	4362.95	4778.59	4465.8
	D2	4305.43	4839.56	4537.74
	D3	4218.86	4867.45	4510.68
	D4	4328.71	4848.54	4488.56
Parameters	Dataset	Time (s)		
		VRP	VRPTW	VRSPD
$popsiz = 50$ , $Max_{iter} = 20$	D1	0.463	0.615	0.504
	D2	0.472	0.617	0.501
	D3	0.458	0.598	0.494
	D4	0.451	0.555	0.484

**Table 8.** Objective function values and execution time of two-tasks

(a) VRP; VRPTW				
Parameters	Dataset	$f$		Time (s)
		VRP	VRPTW	
$popsiz = 50$ ; $Max_{iter} = 20$	D1	4213.61	4979.87	0.868
	D2	4060.97	4982.69	0.746
	D3	4142.73	4672.87	0.731
	D4	4037.11	4792.42	0.705
$popsiz = 100$ ; $Max_{iter} = 40$	D1	4011.14	4606.13	2.834
	D2	4139.31	4711.1	2.941
	D3	4047.3	4756	2.647
	D4	4110.57	4664.01	2.715
(b) VRP; VRSPD				
Parameters	Dataset	$f$		Time (s)
		VRP	VRSPD	
$popsiz = 50$ ; $Max_{iter} = 20$	D1	4204.92	4338.47	0.595
	D2	4165.22	4367.68	0.602
	D3	3981.98	4225.53	0.61
	D4	4147.94	4434.02	0.704
$popsiz = 100$ ; $Max_{iter} = 40$	D1	4131.73	4330.19	2.137
	D2	4219.99	4373.77	2.254
	D3	4124.11	4354.46	2.394
	D4	4022.46	4305.17	2.302
(c) VRPTW; VRSPD				
Parameters	Dataset	$f$		Time (s)
		VRPTW	VRSPD	
$popsiz = 50$ ; $Max_{iter} = 20$	D1	4946.65	4304.22	0.719
	D2	5014.36	4244.36	0.776
	D3	4916.1	4341.17	0.728
	D4	4777.44	4226.77	0.72
$popsiz = 100$ ; $Max_{iter} = 40$	D1	4920.8	4261.2	2.668
	D2	4756.52	4117.19	2.919
	D3	4615.85	4267.96	2.624
	D4	4659.21	4340.27	2.612

**Table 9.** Objective function values and execution time of three-tasks

<i>popsize = 50; Max<sub>iter</sub> = 20</i>				
Dataset	<i>f</i>			Time (s)
	VRP	VRPTW	VRSPD	
D1	4129.49	5025.03	4298.32	0.866
D2	4188.41	5000.38	4348.72	0.927
D3	4265.97	4938.73	4477.07	0.818
D4	4163.76	4998.99	4392.98	0.851
<i>popsize = 100; Max<sub>iter</sub> = 40</i>				
Dataset	<i>f</i>			Time (s)
	VRP	VRPTW	VRSPD	
D1	3975.14	4944.02	4191.76	3.346
D2	4103.68	5024.12	4343.71	3.33
D3	4093.74	4849.41	4131.77	3.12
D4	4048.03	4903.57	4277.21	3.355
<i>popsize = 150; Max<sub>iter</sub> = 60</i>				
Dataset	<i>f</i>			Time (s)
	VRP	VRPTW	VRSPD	
D1	3975.14	4891.45	4154.14	7.838
D2	4116.17	4770.25	4165.96	7.825
D3	4042.3	4797.38	4018.9	7.563
D4	4024.91	4797.85	4218.36	7.558

In the next experiment, one additional task was introduced, resulting in two task pairs to be tested. As shown in Table 8, using the same parameter settings, the results obtained for solving two task pairs were fairly competitive, although not superior. A further experiment was then conducted by doubling the parameter values, specifically the population size



and the number of iterations. The simulation results with the doubled parameters showed that the objective function values for all tasks were better than those obtained using the single-task approach.

Moreover, to further clarify this pattern, an additional task was introduced, resulting in a three-task optimization using the MFO algorithm, as presented in Table 9. Based on the table, solving three tasks using parameter values twice as large as those in Table 7 yielded competitive results but still did not outperform the single-task approach. However, when the parameter values were tripled, the objective function values obtained for each task were better than those achieved using the single-task method.

Overall, the optimal parameter settings for multi-task optimization depend on the number of tasks being executed. For fairness and effective optimization, each additional task requires a proportional increase in both the number of iterations and population size, specifically,  $n$  times the parameter values for  $n$  number of tasks.

Another important aspect is that the number of constraints in each executed task requires particular attention. The mathematical model of the capacitated VRP involves only one binary decision variable and fewer constraints than the other two models, resulting in faster total computational time for its paired task. In contrast, the mathematical models of the VRP with time windows and the VRP with simultaneous pickup and delivery each contain two decision variables, both binary and continuous, as well as a larger number of constraints compared to the capacitated VRP. Consequently, the computational time for each of these tasks is slower. It can be highlighted that computational time is influenced by the mathematical complexity of the model used for each task. The following research question further discusses time complexity from the algorithmic perspective.

RQ5: What is the general computation complexity of the MFO algorithm compared to the single-task approaches?

Regarding time complexity, Big O notation serves as a practical tool to compare the computational performance between the MFO algorithm and single-task approaches. The complexity of the accelerated CS algorithm with transfer learning is primarily influenced by the population size ( $popsiz$ ), problem dimension or number of customers ( $d$ ), maximum number of iterations ( $Max_{iter}$ ), and the number of tasks to be solved ( $n$ ). In general, the algorithm's time complexity can be expressed as follows:

$$O(ProposedMFOalgorithm) = O(popsiz \times d) \times (n + Max_{iter} \times (n + 1)) \quad (18)$$

The equation above consists of the complexity of fitness evaluations during initialization, followed by the complexity of fitness evaluations and crow memory updates. Whereas the single-task approach yields a time complexity as follows:

$$O(Single - taskapproach) = O(popsiz \times d) \times (1 + 2 \times Max_{iter}) \quad (19)$$

If simplified and dominant term is considered, the Big O complexity for the single-task is  $O(Max_{iter} \times popsiz \times d)$ , whereas for the accelerated CS algorithm, it becomes  $O(Max_{iter} \times popsiz \times d \times n)$ . This indicates that the total complexity increases linearly with the number of tasks  $n$ . If  $n$  is large (i.e., many tasks), the algorithm requires greater computational effort, as its complexity grows proportionally

with  $n$ .

RQ6: How does the performance of the proposed algorithm compare to other MFO algorithms?

To validate the effectiveness of the proposed algorithm, a performance comparison was conducted in terms of objective function value and computational time between the accelerated CS algorithm and two existing MFEA algorithms, namely MFEA with particle swarm operator and MFEA with artificial bee colony selection. The comparison results are presented in Table 10.

**Table 10.** Comparison performances

Accelerated CS Algorithm				
Dataset	$f$			Time (s)
	VRP	VRPTW	VRPSPD	
D1	3939.4	4891.45	4154.14	7.838
D2	4116.17	4770.25	4165.96	7.825
D3	4042.3	4797.38	4018.9	7.563
D4	4024.91	4797.85	4218.36	7.558
MFEA with Particle Swarm Operator				
Dataset	$f$			Time (s)
	VRP	VRPTW	VRPSPD	
D1	4167.06	5034.69	4297.16	7.901
D2	4199.18	4856.06	4257.18	7.986
D3	4058.48	4823.57	4086.22	7.785
D4	4089.06	4936.5	4299.14	7.695
MFEA with Artificial Bee Colony				
Dataset	$f$			Time (s)
	VRP	VRPTW	VRPSPD	
D1	4124.67	5019.98	4319.74	7.998
D2	4134.83	4990.16	4308.77	8.005
D3	4194.54	4876.71	4056.94	7.693
D4	4047.67	4838.73	4288.33	7.644

Based on the computational results, the accelerated CS algorithm outperforms both MFEA algorithms in terms of both criteria. For each dataset and task considered, the proposed algorithm consistently achieves better results. The accelerated CS algorithm hybridized with a chaotic map produces better results than the MFEA with a particle swarm operator, both of which employ transfer learning to enhance their MFO effectiveness. This finding further highlights the superior exploitation capability of the accelerated CS algorithm.

The proposed algorithm consistently outperformed the other methods across all instances, demonstrating strong robustness in handling various levels of problem complexity. Moreover, the trade-off between accuracy and computation time indicates that the proposed approach achieves higher computational efficiency, particularly for large-scale MFO problems. Overall, this robustness and efficiency make the accelerated CS algorithm well-suited for real-world routing applications that involve diverse and dynamic constraints.

## 6. CONCLUSION

This paper presents a chaotic map integrated into the CS algorithm with transfer learning, aiming to solve multiple tasks of the VRP. The proposed algorithm is referred to as the accelerated CS algorithm. This algorithm employs a chaotic map for the position update process and a transfer learning mechanism to search for the best solution across the given tasks. The considered tasks correspond to three variants of the VRP: the capacitated VRP, the VRP with time windows, and the VRP with simultaneous pickup and delivery, each

characterized by distinct numbers of decision variables and constraints. The computational results on MFO suggest that a higher number of crows enhances the quality of the solution. Additionally, the awareness probability parameter yields improved performance when set to a value below 0.5, particularly as it approaches 0. This suggests that the position update process driven by the chaotic maps is more effective than the purely random search. Furthermore, multi-task execution demonstrates superior time efficiency compared to single-task execution. Among the chaotic functions tested, the logistic map consistently enhances the algorithm's performance.

The accelerated CS algorithm demonstrates superior performance for mathematical models with binary decision variables or combinatorial structure. However, it has not yet been tested on mathematical models with higher levels of complexity. Moreover, the algorithm requires parameter adjustment when applied to MFO, particularly VRP instances with larger data sizes. For example, to maintain the algorithmic effectiveness, we need to increase the population size and the number of iterations. The algorithm also requires adaptation as the number of tasks increases; similar to data size, a larger number of tasks demands a proportional increase in the number of individuals and iterations. Overall, the algorithm remains competitive for solving MFO.

## ACKNOWLEDGMENT

The authors were financially supported by the Basic Research Scheme Airlangga Research Fund 2023 by Universitas Airlangga [Grant No.: 1894/UN3.1.8/PT/2023].

## REFERENCES

- [1] Gupta, A., Ong, Y.S., Feng, L. (2016). Multifactorial evolution: Toward evolutionary multitasking. *IEEE Transactions on Evolutionary Computation*, 20(3): 343-357. <https://doi.org/10.1109/tevc.2015.2458037>
- [2] Pan, S. J., Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10): 1345-1359. <https://doi.org/10.1109/tkde.2009.191>
- [3] Saleh, A.W., Gupta, G., Khan, S.B., Alkhaldi, N.A., Verma, A. (2023). An Alzheimer's disease classification model using transfer learning Densenet with embedded healthcare decision support system. *Decision Analytics Journal*, 9: 100348. <https://doi.org/10.1016/j.dajour.2023.100348>
- [4] Li, G., Lin, Q., Gao, W. (2020). Multifactorial optimization via explicit multipopulation evolutionary framework. *Information Sciences*, 512: 1555-1570. <https://doi.org/10.1016/j.ins.2019.10.066>
- [5] Xie, T., Gong, M., Tang, Z., Lei, Y., Liu, J., Wang, Z. (2016). Enhancing evolutionary multifactorial optimization based on particle swarm optimization. In 2016 IEEE Congress on Evolutionary Computation (CEC), Vancouver, BC, Canada, pp. 1658-1665. <https://doi.org/10.1109/cec.2016.7743987>
- [6] Yokoya, G., Xiao, H., Hatanaka, T. (2019). Multifactorial optimization using Artificial Bee Colony and its application to car structure design optimization. In 2019 IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, pp. 3404-3409. <https://doi.org/10.1109/cec.2019.8789940>
- [7] Toth, P., Vigo, D. (2002). *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics.
- [8] Braekers, K., Ramaekers, K., Nieuwenhuyse, I.V. (2016). The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, 99: 300-313. <https://doi.org/10.1016/j.cie.2015.12.007>
- [9] Askarzadeh, A. (2016). A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm. *Computers & Structures*, 169: 1-12. <https://doi.org/10.1016/j.compstruc.2016.03.001>
- [10] Pratiwi, A.B. (2017). A hybrid cat swarm optimization - crow search algorithm for vehicle routing problem with time windows. In 2017 2nd International conferences on Information Technology, Information Systems and Electrical Engineering (ICITISEE), Yogyakarta, Indonesia, pp. 364-368. <https://doi.org/10.1109/icitisee.2017.8285529>
- [11] Nirmal, A., Jayaswal, D., Kachare, P.H. (2024). A hybrid Bald Eagle-crow search algorithm for Gaussian mixture model optimisation in the speaker verification framework. *Decision Analytics Journal*, 10: 100385. <https://doi.org/10.1016/j.dajour.2023.100385>
- [12] Rizk-Allah, R.M., Hassanien, A.E., Bhattacharyya, S. (2018). Chaotic crow search algorithm for fractional optimization problems. *Applied Soft Computing*, 71: 1161-1175. <https://doi.org/10.1016/j.asoc.2018.03.019>
- [13] Sayed, G.I., Hassanien, A.E., Azar, A.T. (2017). Feature selection via a novel chaotic crow search algorithm. *Neural Computing and Applications*, 31(1): 171-188. <https://doi.org/10.1007/s00521-017-2988-6>
- [14] Ma, Z., Yuan, X., Han, S., Sun, D., Ma, Y. (2019). Improved chaotic particle swarm optimization algorithm with more symmetric distribution for numerical function optimization. *Symmetry*, 11(7): 876. <https://doi.org/10.3390/sym11070876>
- [15] Sasmito, A., Pratiwi, A.B. (2021). Chaotic student psychology based optimization algorithm for bi-objective permutation flowshop scheduling problem. *International Journal of Intelligent Engineering & Systems*, 14(3): 109-118. <https://doi.org/10.22266/ijies2021.0630.10>
- [16] Pereira, F.B., Tavares, J. (2009). *Bio-Inspired Algorithms for the Vehicle Routing Problem*. Springer Berlin Heidelberg.
- [17] Baker, B.M., Ayechew, M.A. (2003). A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5): 787-800. [https://doi.org/10.1016/s0305-0548\(02\)00051-5](https://doi.org/10.1016/s0305-0548(02)00051-5)
- [18] Van Breedam, A. (1995). Improvement heuristics for the vehicle routing problem based on simulated annealing. *European Journal of Operational Research*, 86(3): 480-490. [https://doi.org/10.1016/0377-2217\(94\)00064-j](https://doi.org/10.1016/0377-2217(94)00064-j)
- [19] Shan, Q., Wang, J.C. (2013). Solve capacitated vehicle routing problem using hybrid chaotic particle swarm optimization. In 2013 Sixth International Symposium on Computational Intelligence and Design, Hangzhou, China, pp. 422-427. <https://doi.org/10.1109/iscid.2013.218>
- [20] Hu, W.B., Liang, H.L., Peng, C., Du, B., Hu, Q. (2013). A hybrid chaos-particle swarm optimization algorithm

- for the vehicle routing problem with time window. *Entropy*, 15(4): 1247-1270. <https://doi.org/10.3390/e15041247>
- [21] Hu, F.J., Wu, B. (2009). Quantum evolutionary algorithm for vehicle routing problem with simultaneous delivery and pickup. In *Proceedings of the 48th IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, Shanghai, China, pp. 5097-5101. <https://doi.org/10.1109/cdc.2009.5399632>
- [22] Ombuki, B., Ross, B. J., Hanshar, F. (2006). Multi-objective genetic algorithms for vehicle routing problem with time windows. *Applied Intelligence*, 24(1): 17-30. <https://doi.org/10.1007/s10489-006-6926-z>
- [23] Taş, D., Jabali, O., Van Woensel, T. (2014). A vehicle routing problem with flexible time windows. *Computers & Operations Research*, 52: 39-54. <https://doi.org/10.1016/j.cor.2014.07.005>
- [24] Kallehauge, B., Larsen, J., Madsen, O.B.G. (2006). Lagrangian duality applied to the vehicle routing problem with time windows. *Computers & Operations Research*, 33(5): 1464-1487. <https://doi.org/10.1016/j.cor.2004.11.002>
- [25] Li, H., Lim, A. (2003). Local search with annealing-like restarts to solve the VRPTW. *European Journal of Operational Research*, 150(1): 115-127. [https://doi.org/10.1016/S0377-2217\(02\)00486-1](https://doi.org/10.1016/S0377-2217(02)00486-1)
- [26] Çatay, B. (2010). A new saving-based ant algorithm for the vehicle routing problem with simultaneous pickup and delivery. *Expert Systems with Applications*, 37(10): 6809-6817. <https://doi.org/10.1016/j.eswa.2010.03.045>
- [27] Dethloff, J. (2001). Vehicle routing and reverse logistics: The vehicle routing problem with simultaneous delivery and pick-up. *OR Spektrum*, 23(1): 79-96. <https://doi.org/10.1007/pl00013346>
- [28] Pluhacek, M., Senkerik, R., Davendra, D., Oplatkova, Z.K., Zelinka, I. (2013). On the behavior and performance of chaos driven PSO algorithm with inertia weight. *Computers & Mathematics with Applications*, 66(2): 122-134. <https://doi.org/10.1016/j.camwa.2013.01.016>
- [29] Xu, Q.Z., Wang, N., Wang, L., Li, W., Sun, Q. (2021). Multi-task optimization and multi-task evolutionary computation in the past five years: A brief review. *Mathematics*, 9(8): 864. <https://doi.org/10.3390/math9080864>
- [30] Solomon, M.M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2): 254-265. <https://doi.org/10.1287/opre.35.2.254>