**Journal of Intelligent Systems and Internet of Things**
**Vol. 14, No. 01, PP. 114-128, 2025**
DOI: https://doi.org/10.54216/JISIoT.140109

# Enhancing Task Scheduling Process in Fog Computing using GTO-SSSA: A Metaheuristic Approach

**V. Arulkumar[1,*], R. Lathamanju[2], T. Nithya[3], T. Rajendran[4]**

[1]Department of Information Technology, Sri Sivasubramaniya Nadar College of Engineering, Chennai, Tamilnadu, India

[2]SRM Institute of Science and Technology, Ramapuram Campus, Chennai, Tamil Nadu, India

[3]Department of Computer Science and Business Systems, Rajalakshmi Institute of Technology, Chennai, Tamilnadu, India

[4]Department of Computer Science and Engineering (Cyber Security), Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology, Chennai, Tamilnadu, India

Emails: arulkumarv@ssn.edu.in; lathamar@srmist.edu.in; nithya.t@ritchennai.edu.in; tlrajen@gmail.com

**Abstract**

Task scheduling (TS) in fog computing (FC) involves efficiently allocating computing tasks to fog nodes, considering factors such as minimizing execution time, energy consumption, and latency to meet the quality-of-service (QoS) requirements of the Internet of Things (IoT) and edge devices. Efficient TS in FC is crucial for optimizing resource usage, minimizing latency, and ensuring that IoT and edge devices receive timely and high-quality services. The growing complexity of FC environments, along with the dynamic nature of IoT applications, necessitates innovative TS models using metaheuristic algorithms to allocate tasks and meet diverse quality-of-service requirements efficiently. This research introduces the GTO-SSSA (Gorilla Troops Optimization with Skip Salp Swarm Algorithm), a novel model for intelligent TS in FC environments. This model capitalizes on the collaborative nature of the GTO algorithm while incorporating enhanced exploration and exploitation capabilities via the SSSA algorithm's skipping mechanism. The primary objective of GTO-SSSA is to tackle the intricate challenges of TS in FC effectively. This includes the efficient allocation of tasks to fog nodes, considering multiple objectives such as minimizing makespan, execution time, and throughput. The GTO-SSSA model in FC demonstrates improved efficiency, consistently surpassing compared models across various task quantities with significantly reduced makespan values. Performance improvement rates for GTO-SSSA over other models show substantial gains in TS efficiency, ranging from 0.87% to 17.83%. The model exhibits scalability as it maintains its efficiency even with an increased number of tasks, aligning with the dynamic nature of IoT applications.

**Received: February 15, 2024 Revised: April 24, 2024 Accepted: July 11, 2024**

**Keywords:** Task scheduling; Fog computing; IoT; Gorilla Troops Optimization; Skip Salp Swarm Algorithm

## 1. Introduction

Fog computing, also known as fog networking, is a decentralized computing infrastructure that extends computing resources and services closer to the data source or endpoint devices, as opposed to relying solely on centralized cloud servers. This approach is particularly relevant in the context of the IoT and other edge computing applications [1]. Fog computing, a decentralized computing model, positions computing resources at the network's edge, reducing latency and enabling faster response times. This is critical for applications like autonomous vehicles, industrial automation, and smart cities. Serving as an intermediary layer between IoT devices and the cloud, it utilizes devices such as routers and edge servers to process and filter data locally, reducing network data transmission. FC facilitates distributed data processing for real-time analytics and decision-making, addressing bandwidth constraints and immediate action requirements. Its scalability allows for easy expansion by adding more fog nodes, making it ideal for handling increasing IoT data. Distributed processing enhances reliability, ensuring applications continue to function even if one node fails, crucial for mission-critical systems. Additionally, it improves security and privacy by keeping sensitive data at the edge, reducing data breach risks during transmission. FC finds application across various domains, including smart manufacturing, smart cities, healthcare, agriculture, and autonomous vehicles, where it can locally process sensor data for real-time decision-making, reducing reliance on remote cloud servers [2].

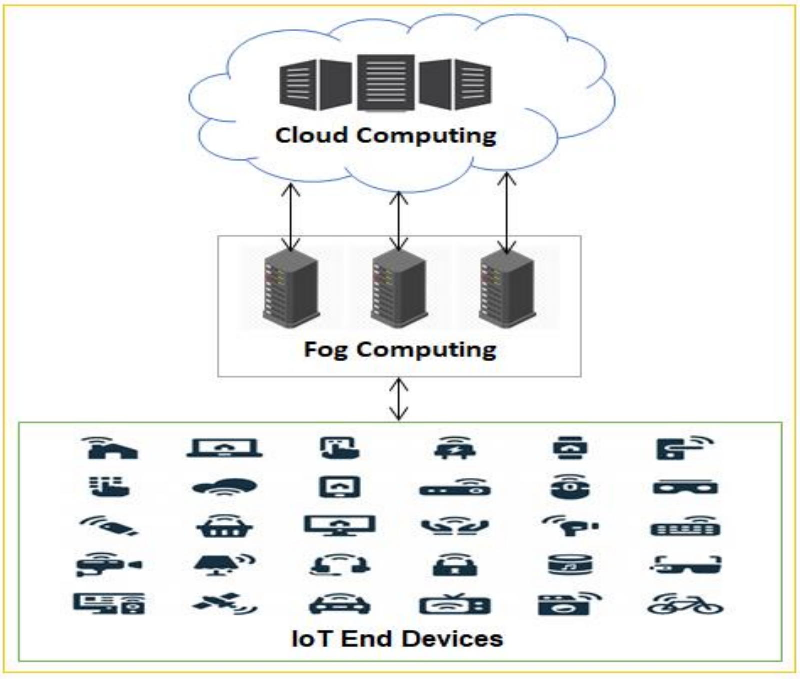

**Figure 1.** Architecture of FC

Figure 1 presents the structure layers of FC: the cloud layer, the fog layer, and the user device layer. The top layer, known as the cloud layer, is equipped with robust computing and storage servers designed to handle extensive data storage and complex computational tasks to support a wide range of application services as required. Positioned at the network's edge, the fog layer is comprised of numerous fog nodes, which may be physical or virtual. The fog nodes are strategically distributed among the cloud layer and end-user devices and possess the capacity to engage in computing tasks, data transmission, and temporary data storage. Real-time applications significantly benefit from data analysis and service delivery within the fog layer, and fog nodes also facilitate connections and collaboration with neighbouring nodes. The user device layer, closest to the end-users and physical environment, encompasses a variety of sensors and IoT devices. In this architectural setup, all the sensors or end devices are linked to a fog node, utilizing either wired or wireless connections such as Wi-Fi, 4G, 5G, wireless local area networks, Bluetooth, and ZigBee. Additionally, fog nodes can establish connections with one another through both wired and wireless communication technologies [1]. Furthermore, FC's flexibility is enhanced by its compatibility with various communication technologies, offering wired connections for reliability and security, as well as wireless technologies for greater mobility and coverage. Fog nodes' ability to communicate through multiple wireless or wired channels enables the creation of resilient, self-organizing networks, ensuring operational continuity, even in the face of network disruptions or congestion [3]. In the field of FC, task scheduling is a diverse process that involves offloading computational tasks from centralized cloud servers to nearby fog nodes, which are better positioned to handle the workload. This decision-making encompasses various factors, such as the complexity of tasks, data size, and the geographical proximity of data sources. A primary objective of FC is to minimize latency, making TS important in achieving this goal by assigning tasks to fog nodes with quick response times, which is particularly critical for real-time applications like autonomous vehicles and industrial automation. Furthermore, TS entails optimizing resource allocation by aligning task requirements with the computational and storage capabilities of fog nodes to ensure efficient execution without overloading any specific node [4].

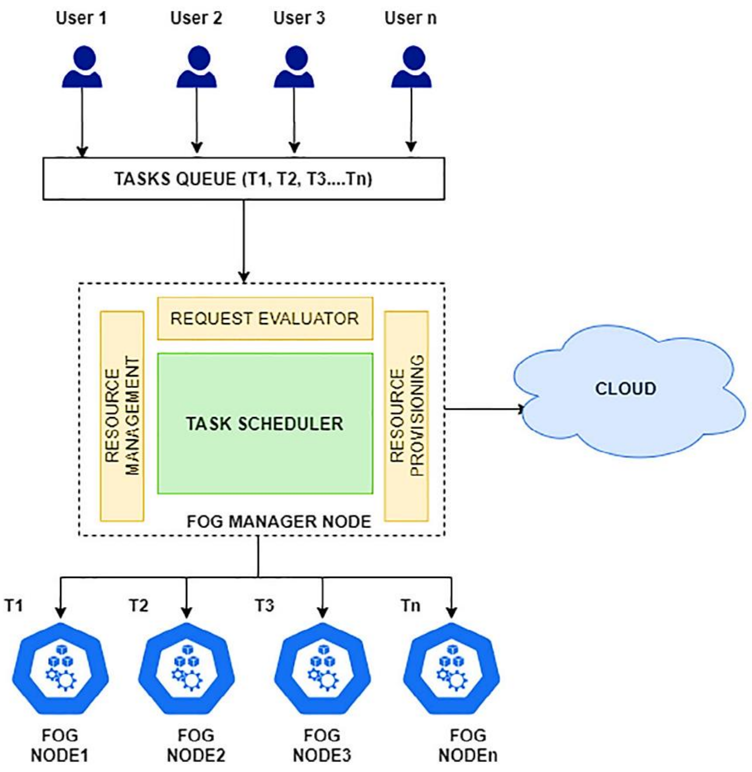

**Figure 2.** Task Scheduling in FC

TS's primary objective is to smartly choose a Fog node that can efficiently complete a task within a set deadline, ensuring uninterrupted user services. This process encompasses several phases depicted in Figure 2. It starts with the reception of requests from IoT sensors or devices, followed by task prioritization through a request evaluator. The subsequent step involves selecting a scheduling node based on factors like computational complexity and resource demands. High-priority real-time tasks are queued and managed by a task scheduler, which assigns them to resource-efficient Fog nodes. This selection is guided by ranking methods, with the task scheduler collaborating with the resource manager to ensure the availability of suitable Fog nodes for scheduling [5]. Finally, advanced techniques like machine learning and optimization algorithms are increasingly harnessed to diffuse intelligence into TS, allowing algorithms to learn from historical data and dynamically adapt to shifting conditions, ultimately enhancing the efficiency and performance of FC systems [7]. Utilizing metaheuristic algorithms for TS in FC is highly significant due to their efficiency in exploring solution spaces and optimizing complex scheduling problems, often achieving near-optimal or optimal solutions where traditional methods may fail. These metaheuristic algorithms offer real-time adaptability, a critical factor in FC, allowing for on-the-fly re-optimization to accommodate dynamic data processing requirements. They play a significant role in reducing latency by swiftly assigning tasks to nearby fog nodes, improving real-time performance. Moreover, their scalability, customization, robustness in handling uncertainty, resource efficiency, capacity to address complex constraints, and capability to enhance system performance make them indispensable techniques for addressing the intricacies and demands of FC environments [8]. The research problem addressed in this work is the need for an efficient and intelligent TS approach within FC environments. As FC gains prominence in managing the vast data generated by IoT devices, ensuring optimal resource allocation and minimizing latency is a complex challenge. The problem revolves around developing a TS framework that can intelligently assign tasks to Fog nodes while considering factors such as computational complexity, resource constraints, and the dynamic nature of FC environments. The objective is to enhance system performance and provide seamless service to users. The primary scope revolves around FC environments, with a focus on TS. The research proposes a novel approach called the GTO-SSSA model to enhance the efficiency of resource allocation and TS in FC.

The contributions of this research are to introduce an innovative model for improving IoT services in cloud-FC, demonstrating the impact of the GTO-SSSA method in terms of performance enhancement and the fulfilment of QoS requirements. This contributes to the broader field of FC and TS optimization.

- To develop a TS method that combines the strengths of GTO and SSSA to address TS challenges in cloud-FC environments efficiently.

- To optimize critical performance metrics, including makespan, improvement rate, and throughput time. The goal is to enhance the QoS requirements for IoT devices by improving these measures.
- To conduct comprehensive experiments involving various tasks to assess the performance and effectiveness of the GTO-SSSA method empirically. These experiments aim to validate the proposed approach and demonstrate its superiority in terms of TS.
- To compare the GTO-SSSA method against other scheduling techniques and methods to establish its advantages and effectiveness in terms of optimizing TS, making it a preferred choice for cloud-FC environments.

The introduction section of this paper introduces the concept of TS in an FC environment and highlights the significance of utilizing metaheuristic algorithms for efficient TS. The subsequent sections cover related works in Section 2, detailed explanations of the GTO and SSSA algorithms and their implementation of TS in Section 3, simulation results and discussion in Section 4, and a conclusion along with recommendations for future suggestions in Section 5.

## 2. Related Works

This section provides a comprehensive overview of existing research and developments in the field of TS for FC environments. This critical analysis highlights various optimization techniques, multi-objective models, and innovative algorithms utilized to enhance resource allocation, reduce energy consumption, and improve QoS. The review offers insights into the evolving landscape of FC task scheduling research, setting the stage for the introduction of the proposed research model, which aims to address and potentially surpass the limitations observed in previous studies. FOG-AMOSM, an adaptive multi-objective optimization TS model for FC for cyber-physical-social services, was developed in [9]. The work in [10] addressed energy-efficient TS in the FC environment by combining Invasive Weed Optimization (IWO) and the Cultural Evolution Algorithm (CEA). They introduced an energy-aware approach utilizing Dynamic Voltage and Frequency Scaling (DVFS) and employed an Invasive Weed Optimizer and Culture (IWO-CA) evolutionary approach to generate valid task sequences. Results confirmed significant energy savings for applications with predefined deadlines, highlighting the approach's effectiveness in reducing energy consumption in cloud data centres. A fog-based architecture for effective TS was developed in [11], formulated the problem as an Integer Linear Programming (ILP) model and proposed an enhanced approach called Opposition-based Chaotic Whale Optimization Algorithm (OppoCWOA) to expedite the problem's solution. This study tackled the TS problem in FC, emphasizing the optimization of time and energy consumption as key QoS parameters. By using partial opposition to increase population diversity, they improved TS performance and achieved faster convergence. A Moth-Flame Optimization (MFO) algorithm was implemented in [12] to address TS in FC for cyber-physical system applications. Their objective was to reduce the overall execution time of tasks while adhering to QoS constraints.

CHMPAD, an approach that combined the chimp optimization with the marine predator algorithm (MPA), was proposed in [13] and a disruption operator to optimize TS for IoT applications in FC. CHMPAD aimed to address ChOA's limitations and enhance its performance by avoiding local optima. Experiments utilizing real and synthetic datasets from the Parallel Workload Archive confirmed the efficiency and consistency of model, particularly in terms of achieving average makespan time improvements. An energy-aware model based on the MPA was implemented in [14] to enhance TS in IoT-related FS applications, aiming to improve user-required QoS. The authors presented two enhanced versions of MPA: the modified MPA and an improved and modified MPA with additional strategies to mitigate local optima. Given the discreteness of TS compared to MPA's continuous nature, a normalization and scaling phase was implemented to adapt MPA for discrete TS. The results affirmed the effectiveness and superiority of the improved and modified MPA model in optimizing TS for enhanced QoS in FC applications. An energy-efficient TS scheme utilizing the artificial rabbit optimization (ARO) inspired by rabbits' survival behaviours, a model called ARO-EETSS, was developed in [15]. ARO-EETSS aimed to optimize TS in Cyber-Physical Systems (CPS), focusing on reducing task completion time and resource utilization. Through simulations, the study demonstrated ARO-EETSS's enhanced performance, emphasizing its efficiency within the CPS environment. The work in [16] addressed TS challenges within the Cloud-FC environment, a highly distributed platform catering to IoT application processing. A mathematical framework incorporating queue theory was devised to optimize workload allocation, ultimately reducing power consumption and delay. A multi-objective TS approach for FC was introduced in [17], combining the marine predator's algorithm with a polynomial mutation mechanism to optimize the makespan and carbon emission ratio. An external archive stored non-dominated solutions, and an improved variant was explored for enhanced convergence. While the improved variant outperformed the standard algorithm, the proposed one remained better. Comparisons with other multi-objective optimization algorithms confirmed the proposed model's consistently better performance across various metrics. A meta-heuristic scheduler called Smart Ant Colony Optimization (SACO) was introduced in [18] for task offloading in an FC environment, specifically for IoT-sensor applications.

The algorithm aimed to optimize task offloading to fog nodes, considering QoS constraints like latency, network characteristics, and node workloads. Results showed that the algorithm effectively reduced latency in IoT-sensor application task offloading.

## 2.1. Research Gap Summary

In the field of FC task scheduling, diverse approaches have been developed to optimize resource allocation, reduce energy consumption, and enhance QoS. These include multi-objective scheduling models, hybrid evolutionary algorithms, and meta-heuristic methods. Notable contributions include FOG-AMOSM, which tackles multi-objective optimization, and approaches combining IWO, CEA, and unique optimization algorithms like OppoCWOA and MFO. Models like CHMPAD, energy-aware MPA and ARO, and MGWO focus on energy efficiency and optimal task distribution. Additionally, various multi-objective algorithms and SACO address TS for IoT applications. These studies collectively illustrate the breadth and evolution of FC task scheduling research, emphasizing resource optimization, energy efficiency, and QoS improvement. The proposed research model offers a compelling solution to address the limitations observed in existing works in TS for FC environments. The research model can overcome the limitations of existing approaches that often focus on specific optimization goals, such as execution time or energy efficiency, without simultaneously addressing other crucial parameters. It optimizes resource allocation efficiently while considering TS, including execution time, energy efficiency, QoS, and workload balancing.

## 3. Proposed Research Model

This research work is structured around a three-layer architecture, as depicted in Figure 2, comprising IoT smart devices at the bottom layer, including sensors, wearables, and medical devices that generate job/task demands. These demands are sent to the middle layer, which consists of fog nodes equipped with computing, storage, and networking capabilities. Fog nodes act as smart servers with limited resources. The top layer comprises the cloud nodes, featuring powerful servers with extensive computing capabilities. The architecture aims to optimize TS, directing time-sensitive tasks to nearby fog nodes to reduce delay and enhance processing efficiency. In contrast, compute-intensive tasks are routed to cloud nodes due to their superior computing capabilities. This layered approach provides a comprehensive framework for efficient task management in FC systems. The central element within the research model was the Fog Broker, which was situated within the layer of fog nodes and comprised three key components: the Resource Monitoring Service, Task Manager (TM), and Task Scheduler. TM is responsible for receiving requests of task from various distributed IoT users and devices, maintaining resource requirements and task characteristics, and hence forwarding these requests to the Tasks Scheduler. The Resource Monitoring Services, on the other hand, collects and monitors the status of available resources, sharing this information with both fog and cloud nodes to aid in scheduling decisions. In Fog Broker, the Task Scheduler runs algorithms for TS based on the features of the incoming tasks and the capabilities of resources available. It resolves the TS problem by mapping the task requests to the appropriate computing node, ultimately processing the requests of tasks, and returning the results to the Fog Broker, which could subsequently be transferred to the respective users. This architecture facilitates efficient task management and allocation in the FC environment.

## 3.1. Formulation of the Problem

The problem formulation for the FC-TS is established as follows: Consider a set of '$n$' independent tasks, denoted as $T = [T_1, T_2, \dots, T_n]$, which are submitted to the Fog Broker for processing within the cloud-FC framework. Every task, $T_k$, possesses specific attributes, including task length (measured in Millions of Instructions, MI), memory requirements, input and output file sizes, and a deadline. The cloud-fog system is comprised of $m$ computing nodes CN, encompassing both cloud and fog nodes $\left[CN = N_{cloud} \cup N_{Fog}\right]$. This can be represented as $CN = [CN_1, CN_2, \dots, CN_m]$, where $CN_j$ signifies the jth CN in the system. Each $CN_j$ is characterized by attributes like CPU processing ratio (measured in Millions of Instructions/ Second, MIPS), network bandwidth, memory capacity, and storage capacity. This mathematical framework forms the foundation for addressing the TS challenge in cloud-FC environments. Accordingly, the matrix ECT, with dimensions $n \times m$, is employed to represent the computing time expected for requests of task on all the CNs. Here, $n$ stands for the total requests of task, while $m$ designates the CNs available. Task Scheduler makes utilize of this ECT matrix to inform scheduling decisions. Specifically, $ECT_{k,j}$ signifies the anticipated execution time for the task $T_k$ on computing node $CN_j$ and can be computed using the following approach.

$$ECT_{k,j} = \frac{Tsk.Len_k}{CN.Pro_j} \tag{1}$$

In this equation, $Tsk.Len_k$ corresponds to the length of the task $T_k$, measured in MI, and $CN.Pow_j$ represents the processing speed of $CN_j$, quantified in MIPS.

The task of assigning requests of task to available CNs was classified as the NP-complete issue. The principal aim involves identifying the optimal schedules those decreases the completion time, commonly known as the "makespan." This minimization is crucial to avoid lengthy task execution delays. The primary objective of this research was to solve the TS problem in a FC system with the key goal of reducing the makespan. For $Z$ any schedule, the makespan (MK) can be computed as follows [18]:

$$MK(Z) = \max_{j\in 1,2,\dots,m} \sum_{k=1}^{n} ECT_{k,j} \quad (2)$$

At this stage, the task scheduling problem can be formally expressed as follows:

$$f(Z) = minMK(Z) = \min \max_{j\in 1,2,\dots,m} \sum_{k=1}^{n} ECT_{k,j} \quad (3)$$

## 3.2. Gorilla Troops Optimizer

The GTO algorithm is a nature-inspired optimization technique rooted in the social behaviour of gorilla troops. This algorithm draws inspiration from the way gorillas coordinate and cooperate to solve problems, find food, and navigate their environment. Gorilla Troops Optimization is a heuristic search algorithm designed for solving complex optimization problems, particularly those that involve constraints and multiple objectives. The core concept of GTO lies in the collaboration and communication among the individuals in a gorilla troop. Each individual (representing a potential solution) communicates with others to collectively find the best solutions. This concept is adapted to optimization problems where a population of potential solutions collaboratively works to explore the search space and find optimal or near-optimal solutions [19]. In the context of TS in FC, the GTO algorithm can be applied to solve the challenging problem of allocating tasks to fog nodes in a manner that optimizes various objectives, such as minimizing execution time, energy consumption, and latency while satisfying QoS constraints. The GTO algorithm draws inspiration from the collaborative and communicative behaviours of gorilla troops. Much like gorillas working together to solve problems and share knowledge, GTO employs a population of potential solutions that collaborate and share information to discover the optimal TS arrangement in the FC environment collectively. Communication, a cornerstone of gorilla troop dynamics, is mirrored in GTO as solutions exchange information, such as the quality of their schedules, through a designated mechanism. GTO's essence lies in the exploration of the solution space, similar to gorillas exploring their environment to find resources. Solutions adjust their positions within this space, striving to find optimal or near-optimal TS arrangements. Adaptability is key for both gorillas and GTO solutions; the latter adjust their positions based on information from peers and the TS problem's objectives and constraints. Finally, cooperation, fundamental to gorilla troops' success, is mirrored in GTO as solutions cooperate by sharing knowledge and collectively enhancing the TS solution, effectively optimizing FC task allocation. In the exploration phase of GTO, each member is considered a potential solution, with the best solution termed the "silverback gorilla." The exploration strategies include moving to unfamiliar regions, balancing exploration, and exploitation by minimizing search space among gorillas, and moving to known areas. The choice of strategy depends on a random value, rand, and a parameter, p. If rand < p, migration to an uncertain location is preferred. If rand is greater than or equal to 0.5, a mobility technique involving other gorillas was chosen, while rand lesser than 0.5 results in moving towards the designated place. These strategies are represented numerically in Equation (4).

$$GX(t+1) = \Big((UL - LL)\times r_1 + LL,\ rand < p(r_2 - C)\times X_r(t) + L\times H,\ rand \geq 0.5\ X(i) - L\times \big(L\times (X(t) - GX_r(t))\big) + r_3 \times (X(t) - GX_r(t))\Big),\ rand < 0.5 \quad (4)$$

The equation involves two gorilla positions, $X(t)$ and $GX(t+1)$, both determined in the following iteration $t$. Rand, $r_1$ $to$ $r_3$ are values within the [0, 1] scale. The parameter $p$ signifies the likelihood of selecting the movement scheme for the unknown region. $X_r$ represents one randomly chosen gorilla while $GX_r$ represents a group of gorillas in a designated zone, also selected randomly. UL and LL denote the upper and lower variable boundaries, respectively. The variables C, H, and L are defined numerically by equations (5), (7), and (8), respectively.

$$C = F\times(-1) \quad (5)$$

$$F = \cos(2\times r_4) + 1 \quad (6)$$

$$L = C\times l \quad (7)$$

$$H = Z\times X(t) \quad (8)$$

$$Z = [-C, C] \quad (9)$$

In this context, $t$ signifies the iteration count, while $r_4$ is a randomly generated value within the [0, 1] range. Additionally, the term '$l$' represents a random value within the range [-1, 1]. Following the exploration step, the objective values for $GX$ are computed. If $GX(t)$ has a lower cost than the current solution $X(t)$, $GX(t)$ becomes the preferred alternative over $X(t)$ (the silverback). In exploitation process, two strategies are employed: monitoring the silverback and competing for the attention of mature females. The choice between these strategies is determined by comparing the outputs of the equation (6) with $W$. The silverback is considered the leader of the gorilla group, responsible for making decisions and guiding the swarm towards resources. This technique is employed when $C \geq W$. To describe this behaviour mathematically, Equation (10) is used.

$$GX(t+1) = L \times M \times (X(t) - X_{sb}) + X(t) \tag{10}$$

The position vector of a gorilla is represented by $X(t)$, and the position vector of the silverback is represented as $X_{sb}$. Additionally, in the Equation $r_4$, the variable M is defined in the following equation:

$$M = \left( \left| \left(\frac{1}{N}\right) \sum_{i=1}^{N} GX_i(t) \right|^{g} \right)^{(1/g)}, \; g = 2^{l} \tag{11}$$

More precisely, every candidate gorilla's vector location at $t$ was denoted as $GX_i(t)$, with $N$ representing the total number of gorillas.

In the case where C < W, the subsequent approach in the exploitation phase involves competing for the attention of mature females. Like the behaviour of young gorillas entering adolescence and competing for the favour of mature females, this behaviour can be mathematically represented using Equation (12) as a foundation.

$$GX(i) = X_{sb} - (X_{sb} \times Q - X(t) \times Q) \times A \tag{12}$$

$$Q = 2 \times r_5 - 1 \tag{13}$$

$$A = \beta \times E \tag{14}$$

$$E = \{ N_1 \; rand \geq 0.5 \;\; N_2 \; rand < 0.5 \tag{15}$$

In Equation (13), Q represents impact force, and $r_5$ stands for random values in the [0, 1] range. Equation (14) calculates parameter $A$, reflecting the intensity of the violence phase, with β preset. 'E' simulates the disruptive effect in Equation (15). At the end of the exploitation phase, $GX$'s objective rate is evaluated, and if $GX(t)$ has a lower rate than $X(t)$, $GX(t)$ is selected as the optimal solution over $X(t)$. To calculate individual fitness, a threshold is essential, as outlined in Equation (16),

$$BX_{ij} = \{ 1 \; if \; X_{ij} > 0.50, \; otherwise \; 0 \tag{16}$$

Each search agent, denoted as i, has a dimension value at dimension j represented as $X_{ij}$. Subsequently, the algorithm employs Equation (17) to compute the fitness of each gorilla as follows.

$$fit_i = \alpha \times \frac{|BX_i|}{D} \tag{17}$$

In this context, α falls within the [0, 1] range, and D represents the dimension of the input. The best choice is determined based on the smallest fitness values. The updating phase is iterated until the specified condition is met. Eventually, the GTO algorithm yields the optimal solution. The following outlines the iterative process of the GTO algorithm to find the best solution. The GTO algorithm follows these steps [20]:

```
Initialization: Initialize the gorillas and adjust parameters (Max_iter, Popsize, β, p)
Compute initial fitness functions as in Equation (17)
If the iteration count (t) is less than or equal to Max_iter:
Update comparison parameters C and L as per equations (8) and (7)
For each gorilla (i) in the population (up to Popsize):
Update the gorilla's position using Equation (4)
Compute the fitness as in Equation (17), and if it is better than prior fitness, update it.
Set the silverback gorilla as the best solution (i=1)
If C is greater than or equal to W:
Update the position following Equation (10)
Else
Update the position as per Equation (13)
Compute the fitness as in Equation (17), and if it is better than prior fitness, update it.
Return the silverback gorilla as the best solution.
```

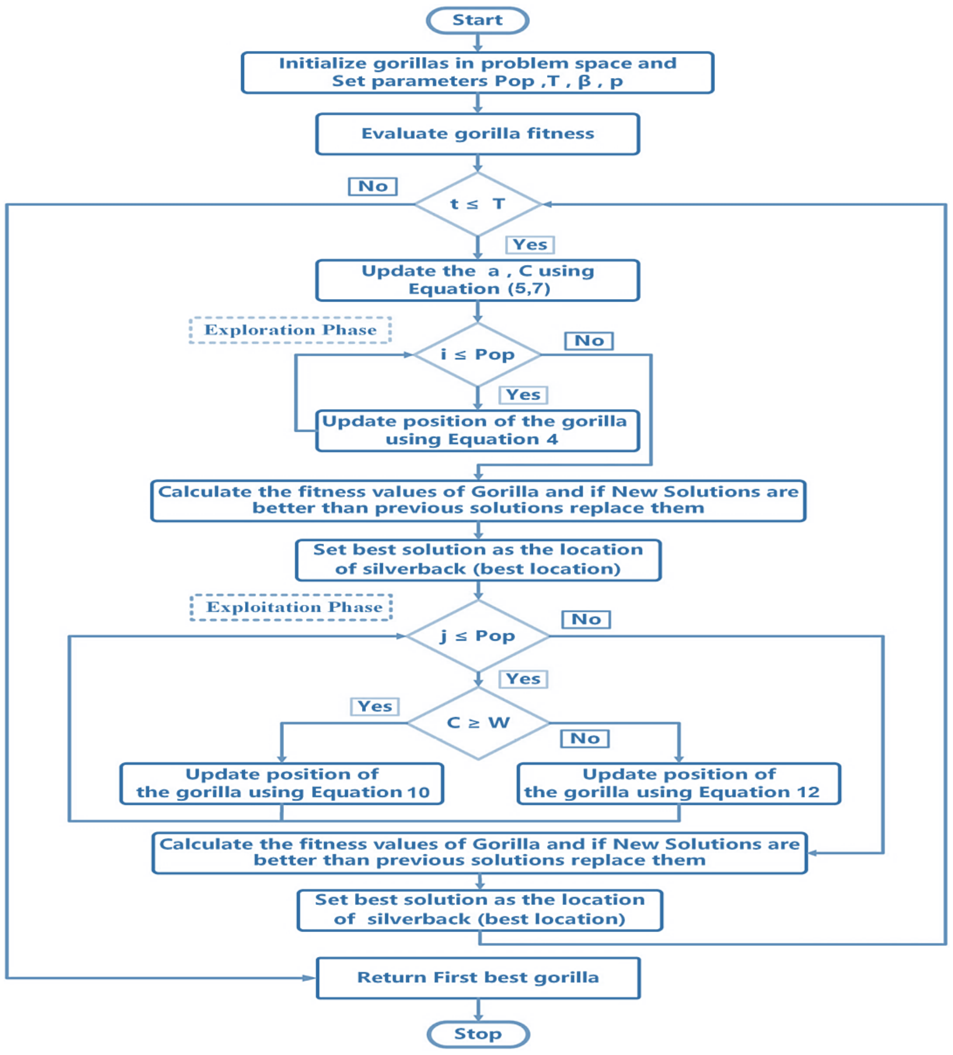
Start
Initialize gorillas in problem space and Set parameters Pop ,T , β , p
Evaluate gorilla fitness
No
t ≤ T
Yes
Update the a , C using Equation (5,7)
Exploration Phase
i ≤ Pop
No
Yes
Update position of the gorilla using Equation 4
Calculate the fitness values of Gorilla and if New Solutions are better than previous solutions replace them
Set best solution as the location of silverback (best location)
Exploitation Phase
j ≤ Pop
No
Yes
Yes
C ≥ W
No
Update position of the gorilla using Equation 10
Update position of the gorilla using Equation 12
Calculate the fitness values of Gorilla and if New Solutions are better than previous solutions replace them
Set best solution as the location of silverback (best location)
Return First best gorilla
Stop

**Figure 3.** Workflow of GTO

### 3.3. Skip Salp Swarm Algorithm

The SSSA is a swarm intelligent optimization approach that combines the Salp Swarm Algorithm (SSA) with a skipping mechanism. SSA itself is inspired by the behaviour of salps, a type of marine organism known for their coordinated and efficient movement in a swarm. In SSA, individuals emulate the swarming behaviour of salps to collectively explore a solution space and find NP or optimal solutions. The SSSA introduces a skipping mechanism, which enhances the algorithm's exploration and exploitation abilities, making it particularly useful for solving complex optimization problems. The core concept of the SSSA algorithm, particularly in the context of TS in FC, involves a two-fold approach [21]:

- Swarming Behavior: The SSA component of SSSA mimics the cooperative and swarming behaviour observed in salps. In this phase, individuals (representing potential solutions) work together in a coordinated manner to explore the solution space efficiently. They communicate, exchange information, and collectively improve the solutions.
- Skipping Mechanism: The novel aspect of SSSA is the introduction of a skipping mechanism. This mechanism allows individuals to occasionally skip their current positions in the solution space and jump to new locations. This skipping behaviour enhances the algorithm's exploration ability by introducing randomness and diversity into the search process.

In the proposed research, the SSSA algorithm is seamlessly integrated with GTO to augment its exploitation capabilities in the context of TS for FC environments. This integration brings forth several critical contributions: firstly, the SSSA's skipping mechanism injects randomness into the search processes, enriching exploration of an extensive solution space, which is particularly valuable in the intricate field of FC task scheduling. Secondly, it enhances GTO's exploitation ability by introducing a more robust strategy to fine-tune solutions. The integration of SSSA with GTO yields a powerful hybrid algorithm that combines the collaborative strengths of GTO with the advanced exploration and exploitation capabilities of SSSA. The primary goal is to optimize FC task scheduling, efficiently allocating tasks to fog nodes while simultaneously addressing multiple objectives like minimizing makespan, execution time and throughput. This approach aims to effectively navigate the intricacies of FC task scheduling, ultimately enhancing its efficiency and performance. The salps are randomly initialized in the first iteration. The chain leader repositions based on food positions in each dimension using Eq. 18, where $F_j$ represents the food location in jth dimension, and $z$ is random number in [0, 1], while $lb_j$ $and$ $ub_j$ denote the lower and upper bounds. The parameters control the balance between exploration and exploitation.

$$s_j^i = \begin{cases} F_j + z * \left(\left(ub_j - lb_j\right) * r_1 + lb_j\right) r_2 \geq 0.5 \\ F_j - z * \left(\left(ub_j - lb_j\right) * r_1 + lb_j\right) r_2 < 0.5 \end{cases} \tag{18}$$

In the original SSA, the entire chain follows a single leader, limiting information sharing and hindering exploitation. There is also a lack of exploration, as salps often follow their predecessors, leading to premature clustering and reduced exploration. To address the limited exploration seen in salps, the GWO introduced the encircling parameter in both head and following location updated equation. The encircling parameter modified, denoted as A and determined by Equation (19) involving $z$ among 0 and 1, helps enhance exploration. The updated head location equation for all salps ($i^{\text{th}}$) with encircling parameters is given in Equation (20).

$$A = \frac{2*z*r_3 - z}{2} + 1 \tag{19}$$

$$s_j^i = \begin{cases} A * F_j + z * \left(\left(ub_j - lb_j\right) * r_1 + lb_j\right) r_2 \geq 0.5 \\ A * F_j - z * \left(\left(ub_j - lb_j\right) * r_1 + lb_j\right) r_2 < 0.5 \end{cases} \tag{20}$$

To enhance information flow and exploration, Skip connections, inspired by neural networks, are introduced in the SSSA. These connections resemble those in neural networks and transmit information from the food location to internal salps, ensuring better information dissemination. In this approach, odd-numbered salps were upgraded utilizing a new head location equation (20), while even-numbered salps were upgraded with a follower location modified equation (21), incorporating an encircling parameter and dual neighbour guidance for exploration [22].

$$\left.\begin{aligned} cent_j^i &= \tfrac{1}{2}\left(A * s_j^{i-1} + A * s_j^{i+1}\right) \\ vel_j^i &= abs\left(cent_j^i + s_j^i\right) \\ s_j^i &= cent_j^i - vel_j^i \end{aligned}\right\} i\%2 == 0 \tag{21}$$

$$Fit = \varphi \times \delta(E) + \alpha \times \frac{|k|}{|K|} \tag{22}$$

The fitness function of the SSSA algorithm is represented in equation (22), where the binary value of the threshold value is 0.5. The k values are given as 5 and 10 population agents. The distance between the salp and centroid was computed as the velocity according to which the newer position was determined.

```
Initialize each salp within the specified bounds randomly.
While the current iteration is less than or equal to the maximum iterations:
Evaluate the fitness of each salp.
Identify the salp with the best fitness as the "Food."
For each salp ($s_j^i$) in the swarm:
If $s_j^i$'s index (i) is odd (i % 2 == 1), update its head position using Equation 4.
For each salp ($s_j^i$) in the swarm:
If $s_j^i$ is the last salp, update its position using Equation 20.
Otherwise, if i is even (i % 2 == 0) and not the last salp, compute the velocity, centroid, and update follower position using Equation 21.
Reposition salps that are outside the specified bounds.
Return the best fitness value (F).
```

### 3.4. Proposed TS Model

This section introduces the task scheduler developed for an FC environment, which utilizes the GTO algorithm with SSSA. In this model, solutions are updated competitively during the exploitation phase using both SSSA and GTO operators. However, during the exploration phase, only GTO operators are employed to enhance the TS process. The GTO-SSSA model for addressing the TS problem commences with an initialization phase. Here, the solutions are initially defined, and their representation is established. These solutions are assigned initial values and are transformed into integer solutions. Subsequently, each solution's quality is evaluated based on its fitness, as denoted in (Equation 3). The next step involves the improvement of these solutions, which occurs during both the exploration and exploitation phases. During the exploration phase, solution updates are carried out using GTO operators. In the subsequent phase, updates are determined through competition between GTO and SSSA operators. This enhancing process the solution continues till the termination criteria were met. The initial step in the GTO-SSSA model involves transforming the randomly generated population Xi (i = 1, 2, ..., n) to adapt it for the problem of TS within the FC. This transformation was necessary for modifying the GTO's behaviour and enable it to address discrete problems, as TS in an FC environment is inherently an integer problem.

$$X_{ij} = fl\left(Lb_{ij} + \alpha \times \left(Ub_{ij} - Lb_{ij}\right)\right), \alpha \in [0,1], j = 1,2,\dots,n \tag{23}$$

Equation (23) defines the limits of the search domain, where $Lb$ = 1 and $Ub = m$. The function $fl$ is used for converting real numbers into integers in this context. In the updating stage, the GTO-SSSA initiates by identifying the solutions with the small makespan values and designating it as best solutions. Subsequently, the remaining solutions are updated based on their respective behaviours. However, the other solutions aimed at uncovering the best positions within reliable regions were upgraded utilizing either GTO or SSSA operators. This selection is made based on the probability (P) assigned to each of these solutions, as defined below.

$$P = \frac{F_i}{\sum_{i=1}^{N} F_i} \tag{24}$$

The solution Xi undergoes updates according to the equation provided in (16). In this equation, the variable rs plays a pivotal role in managing the update process, determining whether GTO or SSSA operators are applied. While $r_s$ could be set as the fixed points, the approach presents challenges in determining the most suitable value. Therefore, the formula presented below is used to define its value.

$$X_i = \begin{cases} operators\ of\ GTO & P > r_s \\ operators\ of\ SSSA & otherwise \end{cases} \tag{25}$$

$$r_s = \min(P) + rand \times \left(max(P) - min(P)\right),\ rand \in [0,1] \tag{26}$$

The preceding steps concerning the update of achieved solutions were iteratively reiterated till the specified stop criteria are met.

```
Initialize the population of solutions $X_i$ (i = 1 to N) randomly
Convert the solutions into integer representations
Compute the fitness of every solution based on the objective functions
Repeat until termination criteria are met:
 for each solution $X_i$:
   if exploration_phase:
     Update $X_i$ using GTO operators
   else:
     Determine the competition probability P for $X_i$
   if P < threshold:  # Choose GTO operator
     Update $X_i$ using GTO operators
   else:  # Choose the SSSA operator
     Update $X_i$ using SSSA operators
  Identify the best solution with the smallest makespan
    Update the exploration_phase
Return the best solution
```

## 4. Experimental Analysis & Discussion

This section covers the details of the experimental setup, including the tests conducted, result interpretation, and data analysis. The evaluations and comparison with existing TS models are carried out utilizing MATLAB R2017a. The simulations were performed on a personal computer with an Intel Core i7-11700T processor clocked at 4.60GHz and 8 GB of memory. The operating system used was Windows 10, 64-bit. The experimental environment employed for all tests included a cloud-fog setup comprising four host machines, two data centres, and 25 virtual machines (VMs). The experiments in this section are aimed at assessing and comparing the performances of the GTO-SSSA model with existing solutions, providing insights into its effectiveness and efficiency within the specified cloud-fog environment.

**Table 1:** Synthetic Dataset Description

| **Parameter** | **Range/Size** |
|---|---|
| File size | (300, 600) MB |
| Length of the task | (300, 600) MI |
| Number of tasks | (200, 1000) |
| Output size | (300, 600) MB |

The experiments exclusively employed a synthetic dataset to assess the effectiveness of the GTO-SSSA model we proposed. This synthetic dataset was designed for the evaluation, and it consisted of 1000 tasks. These tasks were produced utilizing the uniform distributions, where the task lengths varied between 1000 and 20,000 MI. It is important to note that these tasks were independent and non-pre-emptive, meaning they could not be interrupted once started. This dataset allowed us to evaluate the performance and robustness of the scheduling model.

## 4.1. Performance Metrics

In computing environments, makespan serves as a fundamental performance metric employed to evaluate the schedule quality. Makespan can be defined as completion time of the final tasks within the schedule. Its calculation is based on Equation (2). This metric plays a crucial role in evaluating how efficiently tasks are processed within a given computing environment. The Performance Improvement Ratio (PIR) was a parameter utilized to quantify the performance enhancement percentage achieved by the proposed model when compared to the existing models. PIR is calculated using Equation (27). This metric enables a clear assessment of the extent to which the proposed method outperforms existing solutions, providing valuable insights into the efficiency of the approach.

$$PIR = \frac{(Z_c - Z_p)}{Z_p} \times 100\% \qquad (27)$$

In this equation, $Z_p$ represents the fitness value achieved by the proposed GTO-SSSA model, while $Z_c$ denotes the fitness value achieved by one of the compared models. This formula is instrumental in quantifying the performance improvement rate (PIR) by comparing the fitness values obtained through the proposed approach with those from existing algorithms. It provides a clear measure of how much better the proposed technique performs in relation to the comparative methods.

Throughput is a metric that quantifies the total number of tasks completed within a specified timeframe and is essential for evaluating the efficiency of scheduling algorithms. An effective scheduling algorithm is expected to enhance the system's throughput, which can be determined using Equation (28).

$$Throughput = \sum_{t^i \in T} ECT(t^i) \qquad (28)$$

In this equation, the notation $ECT(t^i)$ represents the execution time of the ith task. This metric serves as a valuable measure of algorithm performance, reflecting how efficiently tasks are processed and completed within the system. Table 2 presents a comprehensive comparison of makespan results for various TS models, including ARO, MFO, GTO, SACO, and the proposed GTO-SSSA model across different task quantities. The makespan values, representing the time taken to complete all tasks, are evaluated for varying task loads of 200, 400, 600, 800, and 1000.

**Table 2:** Comparison of Makespan Results

| **Models** | **Tasks** | | | | |
|---|---|---|---|---|---|
| | **200** | **400** | **600** | **800** | **1000** |
| ARO | 41.39 | 83.62 | 117.49 | 152.99 | 203.41 |
| MFO | 39.46 | 82.78 | 115.77 | 149.08 | 200.15 |
| GTO | 38.13 | 79.25 | 113.36 | 145.29 | 192.90 |
| SACO | 36.26 | 73.49 | 109.94 | 143.10 | 188.32 |
| GTO-SSSA | 34.01 | 71.24 | 108.06 | 141.85 | 177.63 |

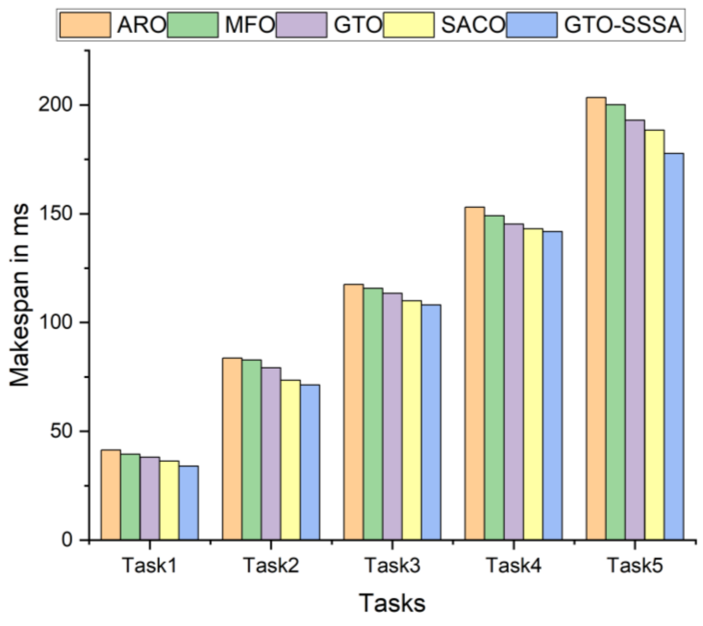

**Figure 4.** Graphical Plot of Makespan Comparison

ARO exhibits makespan values that range from 41.39 for 200 tasks to 203.41 for 1000 tasks. As the task quantity increases, there is a substantial increase in makespan, indicating that ARO's performance degrades with higher workloads. The difference in makespan between ARO and GTO-SSSA is significant, with the latter consistently showing much lower values. MFO offers improved makespan values compared to ARO. However, the makespan ranges from 39.46 for 200 tasks to 200.15 for 1000 tasks. Like ARO, MFO also experiences an increase in makespan as the number of tasks rises. The difference between MFO and GTO-SSSA remains considerable in terms of TS efficiency. GTO performs better than both ARO and MFO, with makespan values ranging from 38.13 for 200 tasks to 192.90 for 1000 tasks. While GTO exhibits improved efficiency, there is still a notable gap between its makespan values and those of GTO-SSSA. SACO further improves TS, with makespan values ranging from 36.26 for 200 tasks to 188.32 for 1000 tasks. Although SACO offers enhanced efficiency compared to ARO and MFO, it still lags behind the GTO-SSSA model. The proposed GTO-SSSA model consistently outperforms the other models, offering the lowest makespan values across all task quantities. Its makespan ranges from 34.01 for 200 tasks to 177.63 for 1000 tasks. The values demonstrate that GTO-SSSA excels in TS efficiency, providing significantly shorter task completion times compared to the other models.

**Table 3:** Comparison of Throughput Results

| **Models** | **Tasks** | | | | |
|---|---|---|---|---|---|
| | **200** | **400** | **600** | **800** | **1000** |
| ARO | 979 | 1925 | 2907 | 3702 | 4815 |
| MFO | 951 | 1897 | 2640 | 3549 | 4501 |
| GTO | 934 | 1805 | 2488 | 3503 | 4319 |
| SACO | 876 | 1660 | 2328 | 3198 | 4176 |
| GTO-SSSA | 801 | 1586 | 2291 | 2950 | 3824 |

Table 3 presents a comprehensive comparison of the throughput results achieved by different models for varying numbers of tasks. The throughput results for ARO indicate that it delivers throughput values ranging from 979 for 200 tasks to 4815 for 1000 tasks. As the number of tasks increases, there is a substantial improvement in throughput, showcasing ARO's ability to handle larger workloads efficiently. MFO's throughput results are competitive, with values spanning from 951 for 200 tasks to 4501 for 1000 tasks. Similar to ARO, MFO demonstrates improved throughput as the task quantity increases, underlining its effectiveness for larger workloads. GTO consistently delivers solid throughput values, ranging from 934 for 200 tasks to 4319 for 1000 tasks. These results highlight the model's reliability and efficiency in achieving high throughput across various task quantities. SACO showcases enhanced throughput performance, with values extending from 876 for 200 tasks to 4176 for 1000 tasks. The model's throughput consistently improves with larger workloads, emphasizing its competence in handling increased tasks.

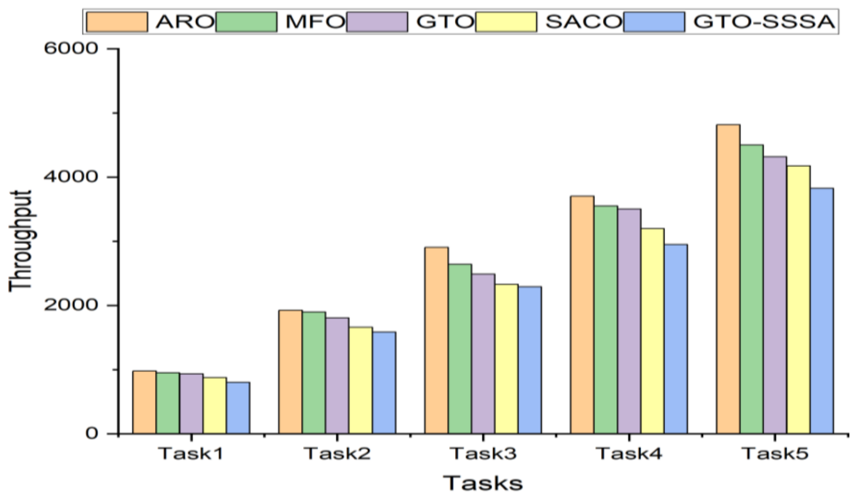

**Figure 5.** Graphical Plot of Throughput Comparison

The proposed GTO-SSSA model excels in providing substantial throughput improvement, surpassing the other models. Its throughput ranges from 801 for 200 tasks to 3824 for 1000 tasks. These results underscore GTO-SSSA's ability to efficiently manage tasks efficiently, yielding significantly improved throughput, particularly for larger task quantities.

**Table 4:** Comparison of Performance Improvement Rate Results

| **Models** | **Tasks** | | | | |
|---|---|---|---|---|---|
| | **200** | **400** | **600** | **800** | **1000** |
| ARO | 17.83 | 14.80 | 8.02 | 7.28 | 12.67 |
| MFO | 13.81 | 13.94 | 6.65 | 4.84 | 11.25 |
| GTO | 10.80 | 10.10 | 4.67 | 2.36 | 7.91 |
| SACO | 6.20 | 3.06 | 1.71 | 0.87 | 5.67 |

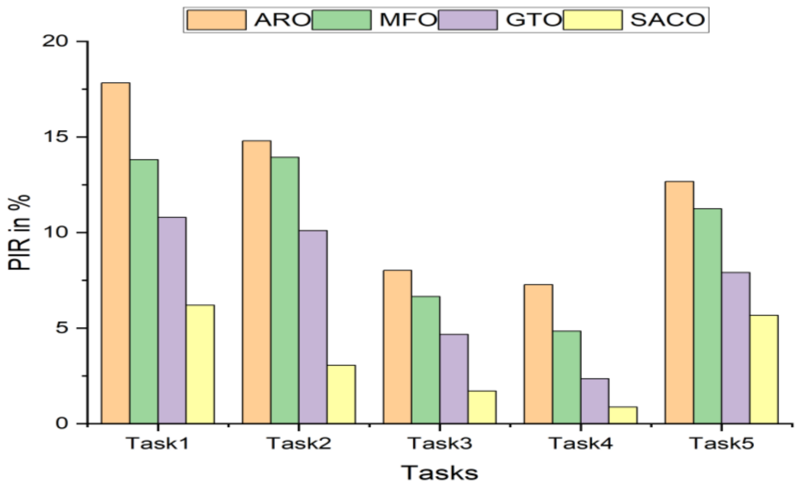

**Figure 6.** Graphical Plot of PIR Comparison

Table 4 provides a comprehensive comparison of the performance improvement rates of the proposed GTO-SSSA model over the compared models (ARO, MFO, GTO, and SACO) across different task quantities (200, 400, 600, 800, and 1000) in the context of fog computing TS. The performance improvement rate represents the extent to which GTO-SSSA outperforms the compared models. For ARO, GTO-SSSA exhibits substantial improvements, with performance improvement rates ranging from 12.67% for 1000 tasks to a significant 17.83% for 200 tasks. This demonstrates that GTO-SSSA significantly enhances the scheduling efficiency of ARO across various task quantities. In comparison to MFO, GTO-SSSA maintains consistent improvements. The performance improvement rates range from 11.25% for 1000 tasks to 13.94% for 400 tasks. GTO-SSSA consistently shows its efficiency in reducing makespan compared to MFO. Even when compared to GTO, which is itself an efficient model, GTO-SSSA achieves notable improvements. The performance improvement rate varies from 7.91% for 1000 tasks to 10.80% for 200 tasks, indicating that GTO-SSSA excels in optimizing TS. SACO, while an improved model, experiences enhancements from GTO-SSSA as well. The performance improvement rates range from 5.67% for 1000 tasks to 6.20% for 200 tasks. GTO-SSSA consistently shows its efficiency over SACO in reducing makespan across different task quantities. Overall, this emphasizes the superiority and efficiency of the GTO-SSSA model in enhancing TS performance over the compared models across varying task quantities in FC environments. It underlines the significant benefits of the GTO-SSSA approach in achieving efficient and optimized TS solutions.

## 5. Conclusion

The research introduced the GTO-SSSA, a novel TS model designed for FC environments. This model capitalizes on the collaborative nature of the GTO algorithm while incorporating enhanced exploration and exploitation capabilities via the SSSA algorithm's skipping mechanism. The primary objective of GTO-SSSA is to tackle the intricate challenges of TS in FC effectively. This includes the efficient allocation of tasks to fog nodes, considering multiple objectives such as minimizing makespan, execution time, energy consumption, and latency. A comprehensive evaluation and comparative analysis involving existing models (ARO, MFO, GTO, SACO) yielded several notable findings. The introduction of the GTO-SSSA model in FC demonstrates improved efficiency, consistently surpassing compared models across various task quantities with significantly reduced makespan values. Performance improvement rates for GTO-SSSA over other models show substantial gains in TS efficiency, ranging from 0.87% to 17.83%. The model's scalability is noteworthy, maintaining efficiency even with an increased number of tasks essential for dynamic IoT applications. By balancing multiple objectives such as makespan, execution time, and throughput, GTO-SSSA enhances the quality of service for FC applications. Future research can focus on implementing and evaluating the GTO-SSSA model in practical FC scenarios and further refining it through parameter optimization and additional quality-of-service criteria.

## References

**[1]** Navjeet Kaur, Ashok Kumar and Rajesh Kumar, “A systematic review on task scheduling in Fog computing: Taxonomy, tools, challenges, and future directions,” Concurrency and Computation: Practice and Experience, vol. 33, no. 21, e6432, 2021.

**[2]** Ambeth Kumar, V.D. Vaishali,S. Shweta, B. (2015). Basic Study of the Human Foot. Biomedical and Pharmacology, 8(1), 435-444.

**[3]** Zhanyang Xu, Yanqi Zhang, Haoyuan Li, Weijing Yang and Quan Qi, “Dynamic resource provisioning for cyber-physical systems in cloud-fog-edge computing,” Journal of Cloud Computing: Advances, Systems and Applications, vol. 9, no. 32, pp. 1-16, 2020.

**[4]** Mostafa Haghi Kashani, Amir Masoud Rahmani and Nima Jafari Navimipour, “Quality of service-aware approaches in fog computing,” International Journal of Communication Systems, vol. 33, no. 8, e4340, 2020.

**[5]** Mohammad Reza Alizadeh, Vahid Khajehvand, Amir Masoud Rahmani and Ebrahim Akbari, Task scheduling approaches in fog computing: A systematic review, International Journal of Communication Systems, vol. 33, no. 16, e4583, 2020.

**[6]** Ambeth Kumar, V.D. (2016). Human Life Protection In Trenches Using Gas Detection System. Journal of Biomedical Research. .27 (2), 475-484

**[7]** Xin Yang and Nazanin Rahmani, “Task scheduling mechanisms in fog computing: review, trends, and perspectives,” Kybernetes, vol. 50, no. 1, pp. 22-38, 2020.

**[8]** Dadmehr Rahbari, “Analyzing Meta-Heuristic Algorithms for Task Scheduling in a Fog-Based IoT Application,” Algorithms, vol. 15, 397, 2022.

**[9]** Ming Yang, Hao Ma, Shuang Wei, You Zeng, Yefeng Chen and Yuemei Hu, “A Multi-Objective Task Scheduling Method for Fog Computing in Cyber-Physical-Social Services,” IEEE Access, vol. 8, pp. 65085-65095, 2020.

**[10]** Kumar, I., Kumar, A., Kumar, V.D.A. et al. (2022) Dense Tissue Pattern Characterization Using Deep Neural Network. Cogn Comput 14, 1728–1751.

**[11]** Zahra Movahedi, Bruno Defude and Amir Mohammad Hosseininia, “An efficient population-based multi-objective task scheduling approach in fog computing systems,” Journal of Cloud Computing: Advances, Systems and Applications, vol. 10, no. 53, pp. 1-31, 2021.

**[12]** Hemamalini, Selvamani, and Visvam Devadoss Ambeth Kumar. (2022). Outlier Based Skimpy Regularization Fuzzy Clustering Algorithm for Diabetic Retinopathy Image Segmentation. Symmetry, 14(12), 2512

**[13]** Ibrahim Attiya, Laith Abualigah, Doaa Elsadek, Samia Allaoua Chelloug and Mohamed Abd Elaziz, “An Intelligent Chimp Optimizer for Scheduling of IoT Application Tasks in Fog Computing,” Mathematics, vol. 10, 1100, 2022.

**[14]** Mohamed Abdel-Basset, Reda Mohamed, Mohamed Elhoseny, Ali Kashif Bashir, Alireza Jolfaei and Neeraj Kumar, “Energy-Aware Marine Predators Algorithm for Task Scheduling in IoT-based Fog Computing Applications,” IEEE Transactions on Industrial Informatics, vol. 17, no. 7, pp. 5068-5076, 2021.

**[15]** Kumar, I., Kumar, A., Kumar, V.D.A. et al. Dense Tissue Pattern Characterization Using Deep Neural Network. Cogn Comput (2022). https://doi.org/10.1007/s12559-021-09970-2.

**[16]** Faten A. Saif, Rohaya Latip, Zurina Mohd Hanapi and Kamarudin Shafinah, “Multi-Objective Grey Wolf Optimizer Algorithm for Task Scheduling in Cloud-Fog Computing,” IEEE Access, vol. 11, pp. 20635-20646, 2023.

**[17]** Mohamed Abdel-Basset, Nour Moustafa, Reda Mohamed, Osama M. Elkomy and Mohamed Abouhawwash, “Multi-Objective Task Scheduling Approach for Fog Computing,” IEEE Access, vol. 9, pp. 126988-127009, 2021.

**[18]** S. Hemamalini ,V. D. Ambeth Kumar ,R. Venkatesan,S. Malathi. (2023). Relevance Mapping based CNN model with OSR-FCA Technique for Multi-label DR Classification. Journal of Fusion: Practice and Applications, 11 ( 2 ), 90-110.

**[19]** Piyush K. Pareek, Pixel Level Image Fusion in Moving objection Detection and Tracking with Machine Learning “,Fusion: Practice and Applications, Volume 2 , Issue 1 , PP: 42-60, 2020

**[20]** Shivam Grover, Kshitij Sidana, Vanita Jain, “Egocentric Performance Capture: A Review”, Fusion: Practice and Applications, Volume 2, Issue 2 , PP: 64-73, 2020.

**[21]** Abdel Nasser H. Zaied, Mahmoud Ismail and Salwa El- Sayed, A Survey on Meta-heuristic Algorithms for Global Optimization Problems, Journal of Intelligent Systems and Internet of Things,Volume 1 , Issue 1 , PP: 48-60, 2020

**[22]** Mahmoud H.Alnamoly, Ahmed M. Alzohairy, Ibrahim M. El-Henawy, “A survey on gel images analysis software tools, Journal of Intelligent Systems and Internet of Things,Volume 1 , Issue 1 , PP: 40-47, 2021.