International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 17 (2025) pp. 387–407

Article

GPU-Accelerated Slime Mould Algorithm for Urgent Transportation in Disaster Response: A COVID-19 Application

Celia Khelfa [1,*], Habiba Drias [1], Ilyes Khennak [1] and Khaled Elleithy [2]

[1] Laboratory for Research in Artificial Intelligence, USTHB, BP 32 El Alia, Bab Ezzouar, Algiers 16111, Algeria; hdrias@usthb.dz (H.D.); ikhennak@usthb.dz (I.K.)
[2] Department of Computer Science and Engineering, University of Bridgeport, Bridgeport, CT 06604, USA; elleithy@bridgeport.edu
* Corresponding author: ckhelfa@usthb.dz

Received date: 13 October 2024; Accepted date: 14 April 2025; Published online: 29 May 2025

Abstract: In the face of increasing natural disasters, ensuring rapid patient transportation by Emergency Medical Services (EMS) is critical to saving lives. The Ambulance Dispatching and Relocation Problem (ADRP) poses a significant challenge, requiring swift allocation of limited ambulance resources. To address this issue, we propose a GPU-Accelerated Slime Mould Algorithm (GPU-SMA) designed for real-time decision-making in disaster scenarios. Our approach leverages the parallel processing power of GPUs using the Compute Unified Device Architecture (CUDA). This significantly reduces computational time, enabling faster and more effective optimization. Additionally, we introduce a new relocation policy that utilizes real-time ambulance data to maintain optimal ambulance positioning. Our method has been using real-world data from the COVID-19 pandemic in Chicago. The results show a remarkable 23x speedup on GeForce RTX 3090 and RTX A4000 GPUs compared to the serial implementation. GPU-SMA outperforms five leading parallel algorithms (GPU-PSO, GPU-APSO, GPU-HHO, GPU-FA, and GPU-BA) in efficiency and effectiveness. A Friedman test confirms the statistical significance of these results.

Keywords: EMS; Ambulance Dispatching and Relocation Problem; Slime Mould Algorithm; GPU; CUDA; COVID-19

1. Introduction

During natural disasters or widespread epidemics, Emergency Medical Services (EMS) personnel are typically the first responders. They play a crucial role in providing urgent medical assistance to those afflicted with sudden illnesses or injuries. Their responsibilities also encompass rescuing survivors and stabilizing patients for transfer to hospitals. This necessity becomes particularly pronounced in large cities. In metropolises with populations exceeding one million, the challenges are intensified by heavy commercial traffic and the presence of industrial centers. These conditions demand a robust prehospital emergency system to ensure rapid response times and effective patient care. A critical component in this healthcare infrastructure is the Ambulance Dispatching and Relocation Problem (ADRP), which is essential for saving lives daily during emergencies. The primary goal of the ADRP is to minimize response times, thereby potentially increasing survival chances for those critically injured by ensuring swift transport to the most suitable hospital. This problem is a specialized variant of the well-known Vehicle Routing Problem (VRP), recognized as an NP-hard problem, which underscores the complexity involved in its resolution.

On the other hand, Swarm Intelligence (SI) algorithms are making notable contributions to tackling

numerous practical challenges [1–5]. One recent and effective SI algorithm is the Slime Mould Algorithm (SMA). It was proposed by Li et al. in 2020 [6], a population-based method that employs swarm intelligence to solve optimization problems. SMA simulates the behavioral patterns of slime moulds, mainly the physarum polycephalum found in nature. These organisms tend to gravitate towards regions with abundant food concentrations while avoiding areas with limited food or obstacles [7].

In this context, the parallelization of metaheuristics emerges not only as a strategy to reduce application execution time but also to enhance method robustness and yield competitive results [8]. Many researchers have significantly advanced metaheuristics by proposing parallel versions of these methods, such as GPU-GA (Genetic Algorithm) [9], GPU-PSO (Particle Swarm Optimization) [10], and GPU-ACO (Ant Colony Optimization) [11]. For instance, parallel metaheuristics have been developed to harness the computational power of GPUs, mainly through NVIDIA CUDA, an extension of the C language. CUDA facilitates the development of GPU kernels by running a grid of threads for parallel processing on GPU cores.

The ADRP has long been a central focus of numerous studies, with many relying on classic optimization methods and heuristics to address its inherent complexity. This work advances the field by shifting toward more sophisticated approaches. We propose a novel parallel algorithm that combines GPU and CUDA technologies with the Slime Mould Algorithm (GPU-SMA) to enhance emergency response capabilities significantly. The primary objective of GPU-SMA is to achieve an optimal balance between exploration and exploitation in a significantly reduced timeframe. Our previous studies [12,13] investigated the SMA and its quantum version for emergency transportation issues. We mainly focused on developing algorithms without utilizing modern parallel computing resources. This limitation restricted their practical use in urgent situations.

The main contributions of this paper are summarized as follows:

1- **Introduction of a mathematical model for the ADRP:** This paper introduces a real-time optimization model for ambulance dispatching, featuring a novel relocation strategy specifically designed for disaster scenarios. Unlike existing models that often rely on random or static relocation strategies, the proposed approach integrates an adaptive relocation mechanism. This mechanism dynamically manages ambulance redeployment based on three critical factors: urgency level, current ambulance availability, and population density.

2- **Development of a GPU-based Slime Mould Algorithm for ADRP:** To our knowledge, this work presents the first GPU-accelerated SMA for crisis management. By leveraging the parallel processing capabilities of GPUs, the approach significantly enhances computational efficiency, enabling faster and more scalable decision-making.

3- **Testing of a real-world scenario:** The GPU-SMA is validated using data from Chicago during the COVID-19 pandemic. We benchmark the performance against traditional sequential SMA and other GPU-based metaheuristics (GPU-PSO, GPU-APSO, GPU-HHO, GPU-FA, and GPU-BA).

The paper is organized into six sections. Section 2 reviews previous works on the ADRP and parallel metaheuristics. Section 3 presents the mathematical formulation of the problem. Section 4 discusses the materials and methods, including a detailed description of the SMA, GPU architecture, and the proposed GPU-SMA for solving the ADRP. The experimental results are discussed in Section 5. Finally, Section 6 concludes the paper and suggests potential areas for further research.

2. Literature Review

This section reviews several studies on the EMS problem, focusing on the ADRP to better position this study within the existing literature. The review first examines traditional approaches, including stochastic and linear programming methods. For instance, Carvalho et al. [14] proposed a mathematical model combined with a heuristic method to optimize emergency vehicle relocation. Their strategy maximizes system coverage using a time-preparedness measure, allowing relocations to any base. Gago-Carro et al. [15] addressed the ambulance relocation–allocation problem through a two-stage stochastic model designed to optimize response times while ensuring equitable emergency coverage. Their work optimizes ambulance allocation by incorporating multi-interval response time thresholds and proposes new strategies for equitable resource distribution. Similarly, Oksuz et al. [16] suggested a dynamic stochastic programming approach combined with a discrete-time markov chain to model uncertainties related to EMS management. Their model integrates multi-period temporary medical center location and staff planning within an uncertain context. While classical methods offer robust solutions, they often encounter computational challenges when applied to large-scale problems.

Due to the NP-hard nature of the ADRP, heuristic and metaheuristic methods have been introduced as practical and efficient solutions for handling the complexity of the problem. Karpova et al. [17] employed a classical model for ambulance location, focusing on developing dynamic relocation

strategies using heuristic tools. Their model minimizes total response time and leverages geographic information systems for real-time data. Shetab-Boushehri et al. [18] used heuristic algorithms to solve the ambulance location and routing problem. They provided a mathematical approach that considered recurrent traffic congestion and the number of requests for EMS in each city zone. Luvaanjala et al. [19] introduced a novel mathematical model for ambulance location in rural areas. Their model aims to maximize coverage of demand zones while minimizing travel times and ensuring more covered zones. The authors employed a Genetic Algorithm (GA) to reduce response times and optimize ambulance placement. Hemici et al. [20] addressed the real-time ADRP using a multi-objective evolutionary algorithm based on decomposition with simulated annealing. Their approach minimizes response times and reduces the number of uncovered priority COVID-19 emergency calls. Bendimerad et al. [21] developed a novel swarm intelligence algorithm called the Deep Self-Learning Artificial Orca Algorithm (DSLAOA) to manage dynamic ambulance dispatching and emergency call systems. Their approach minimizes a global fitness function comprising six sub-functions to optimize the EMS system. The model was validated using a COVID-19 dataset to simulate emergency call arrivals and ambulance positioning over 24 hours.

Despite the advancements in metaheuristics, they often face significant challenges in scalability when applied to high-dimensional problems [22]. To our knowledge, no research has explored the use of parallel metaheuristics for real-time ADRP. However, some studies have investigated GPU-based metaheuristics for other optimization problems. For instance, Zhuo et al. [23] proposed a novel parallel PSO that combines coarse-grained and fine-grained parallelism for global optimization. Using a unified memory access model, their approach minimizes data transfer between the CPU and GPU. In another proposal for GPU-PSO, Alqarni et al. [24] addressed the issue of reduced service quality due to vehicle movements and limited edge coverage. Their approach was tailored to fit the architecture of contemporary GPUs to improve the search for offloading placements. Similarly, Han et al. [25] applied GPU-PSO to the sparse reconstruction problem. In their approach, each particle is managed by CUDA threads, and the swarm is segmented into multiple sub-swarms across CUDA streams to enhance the intensification phase of the algorithm.

Table 1 provides a comprehensive summary of the literature review on the ADRP. The progression highlighted in the table demonstrates a clear evolution from basic models to more sophisticated and complex approaches. However, researchers studying ADRP struggle to tackle large-scale NP-hard problems effectively, especially in complex and dynamic environments. To bridge these gaps, we introduce the first utilization of parallel computing with the SMA for real-time ADRP. A critical challenge in GPU-based implementations is the high cost of CPU-GPU data transfers. Our approach minimizes this by parallelizing key operations, starting with population initialization directly on the GPU. The effectiveness of GPU-SMA is evaluated in a high-dimensional, multi-objective ADRP and compared with other GPU-based algorithms using a COVID-19 case study.

Table 1. Review of ADRP Approaches.

Study	**Problem**	**Approach**	**Objective**	**Originality**	**Limits**
Carvalho et al. [14] (2020)	Real-time ADRP	Linear programming + heuristic	Response time + uncovered calls	Preparedness measure for emergencies	Rare relocations; small case study
Gago-Carro et al. [15] (2024)	ADRP	Two-stage stochastic programming	Response time	Multi-layer model for faster response times	High time complexity
Oksuz et al. [16] (2024)	Location + allocation	Linear programming	Response time + maximize coverage	Integrates dynamic modeling with realistic constraints	Limited real-world validation
Karpova et al. [17] (2023)	Dynamic relocation	Heuristic algorithms	Response time	Use of geographic information systems	Small case study; no dispatching decisions
Shetab-Boushehri et al. [18] (2022)	Routing + relocation	Heuristic algorithms	Response time	Addresses traffic congestion	Lack of validation and comparison

					with other methods
Luvaanjala et al. [19] (2024)	ADP	GA	Response time + maximize coverage	Addresses rural area challenges with constraints	Tested on random scenarios and small problem size
Hemici et al. [20] (2023)	Real-time ADRP	MOEA/D + Simulated Annealing	Response time + uncovered calls	Introduces a mechanism to timestamp and track non-dominated solutions over time	High time complexity; random relocation strategy
Bendimerad et al. [21] (2024)	Real-time ADRP	DSLAOA	Response time + uncovered calls	Learning approach with AOA and real case study	High time complexity; random relocation strategy
Proposed approach (2025)	Real-time ADRP	New parallel SMA	Response time + critical call prioritization + relocation cost reduction	Leverages GPU parallelism with SMA to efficiently manage high-demand emergency situations	First utilization in ADRP, needs more validation

3. Problem Definition and Mathematical Model

EMS is vital in the healthcare system, focusing on pre-hospital care and immediate emergency medical assistance. Its main objective is to provide quick medical help, stabilize patients, and transport them to healthcare facilities using ambulances. The ADRP is a critical challenge faced by EMS, highlighting the complex decision-making process involved in emergency medical response. This problem consists of determining the most efficient way to assign a limited number of ambulances to emergency calls, ensuring that each patient receives timely medical assistance. This challenge becomes particularly pronounced in natural disasters, where the volume of calls can overwhelm available resources. To make informed decisions, EMS has access to a wealth of information to aid in planning ambulance deployments, including the locations of patients, the availability of ambulances, and the capacity of hospitals [26]. Figure 1 illustrates the process taken by the dispatch center to respond to emergency demands in disaster situations.

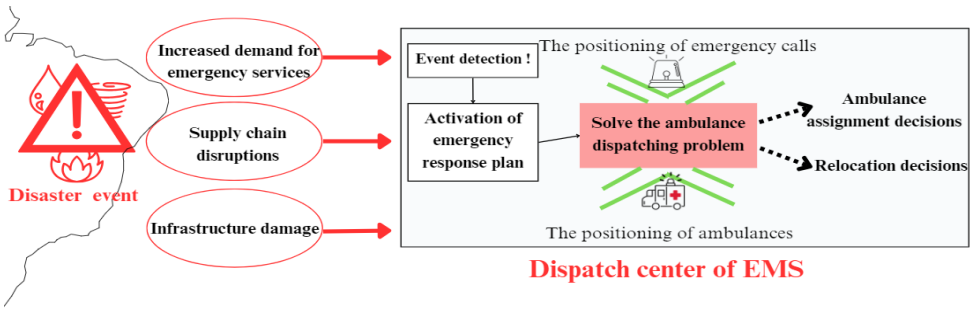

Figure 1. Ambulance deployment process during natural disasters.

The challenge of determining the optimal placement of ambulance resources, known as the ambulance relocation problem, is critical for providing effective emergency medical services. The ambulance relocation problem is crucial as it allows EMS to adjust ambulance positions in real-time based on demand and emergency severity. This approach is especially vital during times of crisis, when specific areas may witness a sudden increase in calls of high severity that require immediate medical attention. By swiftly redistributing ambulances from other locations, the system ensures adequate coverage and timely responses to the escalated demand.

Mathematical Model

The study presents a detailed mathematical model designed to optimize the ADRP. This model operates in real-time, employing discrete periods to accurately reflect the distribution of incoming emergency calls throughout the day in disaster situations. It focuses on managing emergency calls collected in a set C, coming from specific sub-zones q, within a larger area Q. These calls are reported in real-time at each discrete time interval t, and each is distinctively characterized by its location and urgency level. This necessitates the prompt dispatch of ambulances from a predefined set of hospitals, denoted as set A, to the emergency sites. The model presumes a uniform ambulance type, asserting that each ambulance is equipped to handle any emergency. A solution within our model comprises a set of ambulance-emergency call assignments, systematically organizing how ambulances are dispatched to emergencies.

To address the dynamic challenges inherent in ambulance deployment, we propose a novel relocation strategy that dynamically adjusts ambulance allocation in real-time. This approach adeptly responds to the changing demands of emergency situations across various zones. Our strategy prioritizes zones for ambulance deployment based on three pivotal factors: the volume of high-urgency calls U_q a zone receives, the current availability of ambulances V_q within that zone, and the population size Pop_q of the zone, as outlined in Eq. 1. Through careful evaluation of these elements, we determine the relative importance of each zone, directing the strategic relocation of ambulances to where they are most urgently required. This method ensures a responsive and effective deployment of ambulance services, tailored to meet the most critical needs first.

$$P_q = U_q - a \cdot V_q + b \cdot \mathrm{Pop}_q \tag{1}$$

The objective function expressed by Eq. 2 is defined to minimize the total response time. The objective function of Eq. 3 seeks to move the available ambulances in the shortest possible time so that the move has the least cost for the emergency system (relocation cost). The objective function of Eq. 4 prioritizes critical emergency calls, ensuring ambulances reach these cases quickly to potentially save more lives. Table 2 provides the notation for the parameters and decision variables used in the mathematical model.

To combine these multiple objective functions into a singular objective function, a simple method known as weighted sum is employed [27]. This method creates a unified objective function, denoted in Eq. 5. Where w_1, w_2 and w_3 are the weights indicating the importance of minimizing the response time, the relocation cost, and the priority of emergency calls, respectively.

$$F_1 = \sum_{i \in A} \sum_{j \in C} t_{ij} x_{ij} \tag{2}$$

$$F_2 = \sum_{i \in A} \sum_{k \in A} t_{ik} c_{ik} \tag{3}$$

$$F_3 = \sum_{j \in C} Priority_j q_j^t \tag{4}$$

Minimize $$\mathrm{F} = w_1 F_1 + w_2 F_2 + w_3 F_3 \tag{5}$$

Subject to

$$\sum_{i \in A} x_{ij} = 1 \; (j \in C) \tag{6}$$

$$\beta_k^t + \sum_{j \in C} t_{kj} C_{kj} < U \; (j \in C)(k \in A) \tag{7}$$

$$\sum_{i \in A} \sum_{j \in C} x_{ij} + C_{ik} = 1 \; (k \in A) \tag{8}$$

$$V_k^t + \sum_{i \in A} C_{ik} \leq h_k (k \in A) \tag{9}$$

$$\sum_{i \in A} \sum_{k \in A} C_{ik} Pr_i \leq \sum_{i \in A} \sum_{k \in A} C_{ik} Pr_j \tag{10}$$

$$x_{ij} \in \{0,1\} (\forall i \in A, \forall j \in C) \tag{11}$$

$$c_{ik} \in \{0,1\} (\forall i, \forall j \in A) \tag{12}$$

Eq. (6) guarantees that at most one ambulance is dispatched for each emergency. Eq. 7 pertains to the

redistribution of ambulance workload during shifts. The purpose of this equation is to ensure that the accumulated relocation time of each ambulance does not surpass the maximum allowed limit. Eq. 8 ensures that each available ambulance will be either dispatched to an emergency or assigned to another hospital, but never both at the same time. Eq. (9) takes into account the number of ambulances currently present at a given hospital as well as the number of ambulances entering and leaving the hospital. The objective of this equation is to prevent any violation of the maximum allowed number of ambulances at the hospital. Eq. 10 ensures that ambulances are allocated in a way that prioritizes zones with higher urgency levels in the relocation decision. Finally, Eqs. (11) and (12) state that the types of the decision variables must all be binary.

Table 2. Parameters and decision variables notation in the mathematical model.

Parameter	**Description**
t_{ij}	Travel time between the location of ambulance i and location j at time t.
Priority_j	The level of urgency or severity assigned to a particular emergency call j.
Pr_i	The urgency level of the zone q to which ambulance i is assigned.
β_k^t	Accumulated relocation time of ambulance k at time t.
U	Maximum allowed accumulated relocation time for each ambulance.
V_k^t	Number of ambulances currently stationed at hospital k at time t.
h_k	Maximum number of ambulances that can be stationed at hospital k.
Decision variable	**Description**
x_{ij}	Binary variable, 1 if an ambulance from hospital $i \in A$ is assigned to call $j \in C$.
C_{ik}	Binary variable, 1 if an ambulance moves from hospital $i \in A$ to hospital $k \in A$.
q_j^t	Binary variable, 1 if an emergency call $j \in C$ is not covered at time t.

4. Materials and Methods

This section introduces the materials and methods used to solve the ADRP. First, we present the slime mould algorithm in detail, along with the basic concepts of GPU technology. Then, we outline the steps of our new GPU-SMA approach for solving the ADRP.

4.1. Slime Mould Algorithm (SMA)

The slime mould algorithm [6] is inspired by the behavior of slime moulds in identifying the optimal path to reach food in nature. This metaheuristic has been widely applied to both discrete and continuous optimization problems and is recognized for its strong optimization capabilities [28–30]. The SMA starts by initializing a random slime mould population, evaluating the fitness of each solution, and sorting them to assess the weight of each solution. The solutions are then updated depending on whether the algorithm is in the exploration or exploitation phase.

Initially, the slime mould approaches food by sensing the concentration of odor in the air. At iteration t+1, the slime mould can be guided by the best solution and two other random individuals with probability p_i, or it can follow its own way with probability $1 - p_i$, as shown in Eq. 13.

$$X_i(t+1) = \begin{cases} X_{best}(t) + v_b.\left(w_i(t).X_{r1}(t) - X_{r2}(t)\right) & \text{if } r1 < p_i \\ v_c.X_i(t) & \text{otherwise} \end{cases} \tag{13}$$

where, $X_i(t)$ denotes the position of the slime mould i at iteration t and $X_i(t+1)$ refers to its position at iteration $t+1$. $X_{best}(t)$ represents the current best position, indicating the location of the individual with the highest odor concentration. $X_{r1}(t)$ and $X_{r2}(t)$ are two randomly selected individuals in the slime mould population, and $r1$ is a random value between 0 and 1. The velocity v_b controls the intensity of movement of solutions toward the best areas. It ranges from $-a$ to a, where a decreases over iterations. On the other hand, v_c determines how much a slime mould moves randomly within the search space. It ranges from $-b$ to b, where b also decreases over iterations. both a and b are computed using Eq. 14 and Eq. 15, respectively. Max_iteration represents the maximum number of iterations.

$$a = \text{arctanh}\left(1 - \frac{t}{\text{Max\_iteration}}\right) \tag{14}$$

$$b = 1 - \frac{t}{\text{Max\_iteration}} \tag{15}$$

The parameter p_i indicates the probability of focusing on exploitation towards finding the best solution. It is calculated using Eq. 16, where $\text{f}(X_i)$ denotes the fitness value of the slime mould X_i and DF represents the highest fitness achieved across all iterations.

$$p_i = \tanh(\mathrm{f}(X_i) - DF) \tag{16}$$

Slime moulds adapt their network by adjusting vein thickness based on food availability, strengthening connections in food-rich areas and exploring new regions when food is scarce. In the SMA, the weight w reflects this behavior by assigning higher weights to high-quality solutions and lower weights to low-quality solutions. For each slime mould i, $w(i)$ is calculated according to Eq. 17. The $Condition$ shows that X_i ranks in the top half of the population, and $SmellIndex$ corresponds to the sequence of fitness values of solutions sorted in ascending order. In this context, B_f and W_f represent the best and worst fitness values generated up to the current evaluation.

$$w(SmellIndex(i)) = \begin{cases} 1 + r.\log\left(\dfrac{B_f - f(X_i)}{B_f - W_f} + 1\right) & \text{if } Condition \\ 1 - r.\log\left(\dfrac{B_f - f(X_i)}{B_f - W_f} + 1\right) & \text{otherwise} \end{cases} \tag{17}$$

$$SmellIndex = Sort(f(X)) \tag{18}$$

In nature, the slime mould can randomly reinitialize its position to explore new areas. This process is controlled by a parameter z, which represents the probability of the slime mould being randomly reinitialized within the search space. LB and UB are the lower and upper bounds of the search space. The update equation of SMA is formulated in Eq. 19, which is based on Eq. 13 by adding a random component.

$$X_i(t+1) = \begin{cases} rand.(UB - LB) + LB & if\ r2 < z, \text{(a)} \\ X_{best}(t) + v_b.\left(w_i(t).X_{r1}(t) - X_{r2}(t)\right) & if\ r1 < p_i, \text{(b)} \\ v_c.X_i(t) & if\ r1 \geq p_i (c) \end{cases} \tag{19}$$

The pseudo-code of the SMA is outlined in Algorithm 1.

Algorithm 1 Pseudo code of the SMA

Input: Population size N, Parameter z, Lower Bound LB, Uper Bound UB, Maximum number of iterations Max_Iteration
Output: The best or a near-optimal solution
Initialize the population of N slime mould solutions;
t=1;
while *t* < Max_Iteration and not-optimal solution **do**
    Evaluate the fitness of each solution in the population;
    Sort the solutions based on fitness values;
    Update the best solution S_{best};
    Calculate w using Eq. 5;
    for each solution in the population **do**
        Generate a random number $r2$;
        If $r2 < z$ **then**
            Update the solution position using Eq. (19a);
        Else
            Calculate p_i, vb, and vc;
            Generate a random number $r1$;
            If $r1 < p_i$ **then**
                Update the solution position using Eq. (19b);
            Else
                Update the solution position using Eq. (19c);
    End for
    T=t+1;
End while
return S_{best};

4.2. Basic Concepts in GPU Architecture

GPUs have recently become increasingly important in scientific and academic research. The most distinct feature of a GPU compared to a conventional CPU is its unique architecture and design. A GPU is built with a massively parallel architecture, which enables it to process a large amount of data simultaneously. This feature makes GPUs helpful in handling complex applications.

CUDA is a parallel computing platform and programming model developed by NVIDIA and launched in 2006. CUDA encompasses a comprehensive library suite for complex calculations, including

linear algebra, signal processing, and optimization tasks [31]. A CUDA kernel is a function designed to run on the GPU and launched from the CPU. This kernel operates within a grid, an assembly of blocks executing the same function. Each block comprises multiple threads, which represent the most granular execution unit on the GPU. Threads within this architecture are uniquely identified by an ID. The calculation of ThreadID depends on whether the thread configuration is one-dimensional (1D) or two-dimensional (2D), as defined by Eq. 20.

$$\text{ThreadID} = \begin{cases} \text{threadIdx.x} & \text{if 1D,} \\ \text{threadIdx.x} + \text{threadIdx.y} \cdot \text{blockDim.x} & \text{if 2D.} \end{cases} \tag{20}$$

In CUDA parallel architecture, thread allocation schemes determine how computational tasks are mapped on the GPU. The most well-known parallel computing approaches are: (1) Coarse-grained parallelism [32], where each task is assigned to an individual thread. This approach is suitable for situations that require processing a large number of small tasks. (2) Fine-grained parallelism [33], where each particle is assigned to a block of threads, and its elementary operations are processed by individual threads within that block. This scheme is ideal for complex tasks where particles have multiple dimensions.

4.3. GPU-SMA

In the GPU-SMA implementation, we use a master-slave model, where the CPU oversees the overall process by initializing the algorithm variables, launching kernel functions on the GPU, and returning the best solution found. Meanwhile, the GPU performs all computationally intensive tasks in parallel. The parallel SMA operates iteratively, maintaining the sequential order of steps from the original algorithm, but each step is executed in parallel using CUDA kernels. The detailed specifics of the algorithmic steps and kernels are elucidated below.

Step 1: Data transfer from CPU (Host) to GPU (Device)

In contrast to the rapid computational capabilities of GPUs, the communication speed between the GPU and CPU is relatively slow. To address this, the population is generated directly on the GPU using a kernel, avoiding the need to transfer it from the CPU. Only the algorithm variables and problem-specific data are transferred to the GPU using the *cudaMemcpy* function. Memory is allocated on the GPU for the algorithm parameters and for the population matrix, the fitness array, and the weights array using *cudaMalloc*.

Step 2: Population generation

After transferring the necessary data to the GPU, a population of size N and dimension D is generated in parallel using CUDA threads. The *Population Generation* kernel (defined in Algorithm 2) utilizes block and thread indices (blockIdx, blockDim, and threadIdx) to map each thread to a specific row and column in the 2D array population of size $N \times D$. The population matrix is generated in parallel on the GPU, with each thread initializing one variable of a solution. In this context, each row represents a solution, while each column corresponds to a variable within that solution.

Algorithm 2 Pseudo code of *Population Generation* kernal (Device)

Input: Population size d_*N, D*imention d_D, Lower Bound d_LB, Uper Bound d_UB
Output: The generated population d_population

```
Int row = blockIdx.x × blockDim.x + threadIdx.x;     // Solution index
Int col = blockIdx.y × blockDim.y + threadIdx.y;     // Variable index
if (row < d_N and col < d_D) Then
    d_population[row,col] = d_LB[col] + rand(). (d_UB[col] - d_LB[col]); //Generate random solutions in parallel.
End if;
Return d_population;
```

Step 3: Compute fitness values

The evaluation of the fitness function involves complex arithmetic calculations with high data density. Since the computation of the fitness value for a specific element is independent of the others, it is well-suited for parallelization using the *Fitness Evaluation* kernel described in Algorithm 3. In this kernel, a fitness array of size N is allocated, where each element stores the fitness value of a corresponding solution. The computation is performed in parallel, with each thread evaluating the fitness of one solution independently.

Algorithm 3 Pseudo code of *Fitness Evaluation* kernal (Device)

Input: Population size d_*N,* Population d_population
Output: Fitness solutions d_fitness
Int *row* = blockIdx.x × blockDim.x + threadIdx.x; // Solution index
if (row < d_*N*) **Then**
    d_fitness [row]=Calculate fitness value (d_population[row]); //Calculate the fitness values in parallel for each solution
End if;
Return d_fitness;

Step 4: Update the weight of the Slime Mould

Multiple parameters influence the performance of the SMA, including the weight associated with the slime mould solution. The *Weights Calculation* kernel, defined in Algorithm 4, assigns one thread to calculate the weight value of a single solution. This kernel operates on a weights array of size *N*, where each element stores the weight value of a corresponding solution.

Algorithm 4 Pseudo code of *Weights Calculation* kernal (Device)

Input: Population size d_*N,* Population d_population, Fitness solutions d_fitness
Output: Weights value d_w
Int *row* = blockIdx.x × blockDim.x + threadIdx.x; // Solution index
if (row < d_*N*) **Then**
    d_w [row]=Calculate w (d_population[row], d_fitness[row]); //Calculate the weight value in parallel for each solution (Eq. 17)
End if;
Return d_w;

Step 5: Update Slime Mould position

This phase adjusts the positions of all solutions by applying the position update formula outlined in Eq. 19. The kernel *Update Solutions* shown in Algorithm 5, is responsible for updating the population within the GPU-SMA. Each thread is assigned to update a specific variable within a solution, allowing multiple threads to work concurrently. This parallel processing enables the simultaneous updating of all variables across each solution in the population.

Algorithm 5 Pseudo code of *Update Solutions* kernal (Device)

Input: Population size d_*N,* Population d_population, *D*imention d_D, Weights value d_w, Fitness solutions d_fitness, Parameter d_z
Output: Updated population d_population
Int *row* = blockIdx.x × blockDim.x + threadIdx.x; // Solution index
Int *col* = blockIdx.y × blockDim.y + threadIdx.y; // Variable index
if (row < d_N and col < d_D) **Then**
    d_population[row,col] = Update solution (d_z, d_w, d_fitness, d_population, d_D); // Update solutions in parallel (Eq. 19)
End if;
Return d_population;

Figure 2 illustrates the process of GPU-SMA. The process begins by transferring essential parameters to the GPU, followed by the execution of four kernels. The algorithm iteratively generates a population of potential solutions, evaluates their fitness, and dynamically updates their weights and positions within the search space. These operations are executed in parallel through distinct GPU kernels. The iterations continue until specific stopping criteria are met. Once complete, the best solution found is transferred back to the host.

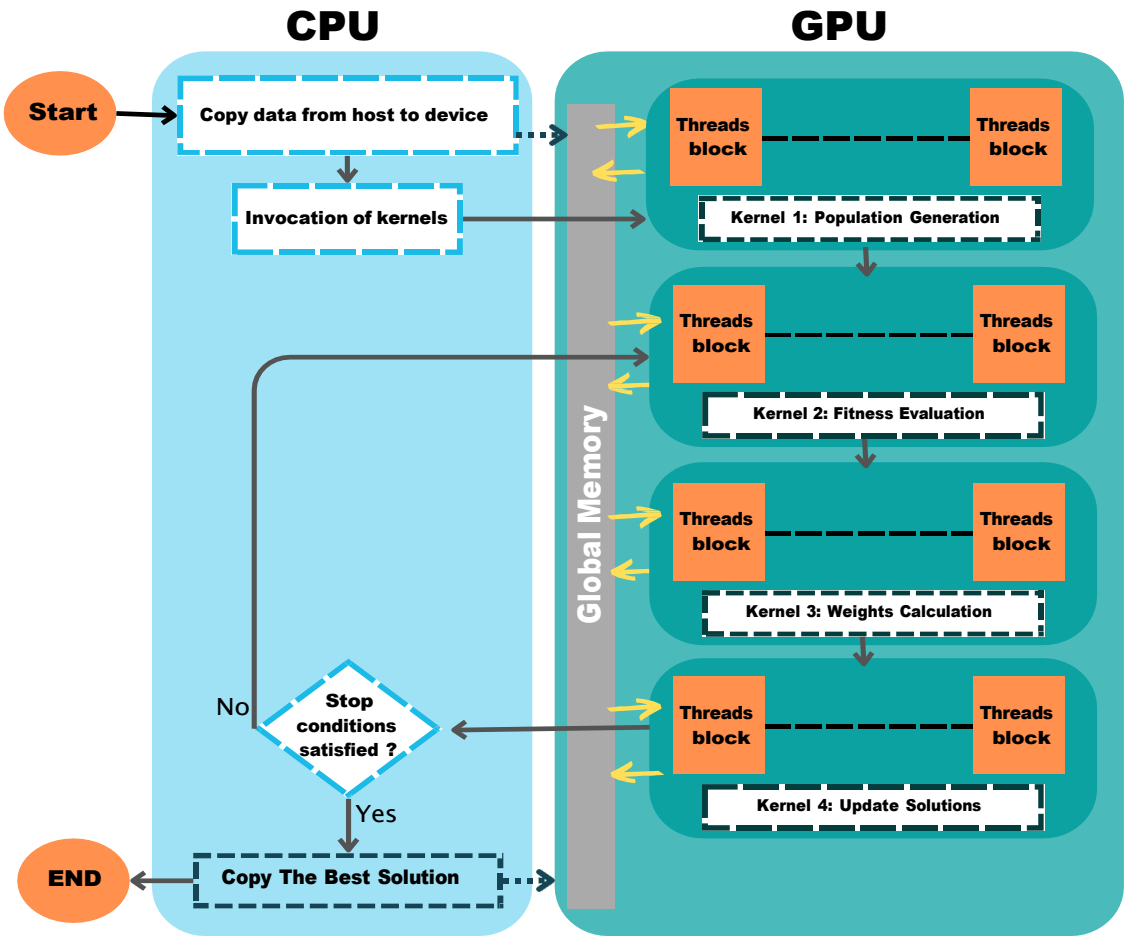

Figure 2. Illustration of the main steps of the GPU-SMA.

4.4. Adapting GPU-SMA to ADRP

Using GPU-SMA to solve the ADRP involves several critical steps to ensure the effectiveness of the algorithm in real-world settings. Below is an overview of the process.

Solution representation: The solution representation for the ADRP involves modeling the assignment of ambulances to patients. We use a vector S consisting of m elements, where m represents the number of emergency calls. Each element corresponds to an ambulance index, thereby delineating the ambulance-call assignments. A binary representation is then generated, using bits to encode ambulance indices.

Fitness function: The fitness function guides the GPU-SMA during the optimization process. The optimization objective is to minimize a weighted function, as defined in Eq. 13. The problem includes seven constraints. The constraints in Eqs. (6, 11, and 12) are inherently satisfied by the solution encoding. Constraint in Eq. 8 is explicitly verified during solution generation to ensure that an ambulance is not simultaneously assigned to both an emergency and a relocation. Constraints in Eqs. (7, 9, and 10) are integrated into the fitness function through penalty terms, which penalize violations and reduce the likelihood of the algorithm selecting infeasible solutions.

The penalty terms $P1$, $P2$, and $P3$ are designed to penalize violations of constraints in Eqs. (7, 9, and 10), respectively. Each penalty term returns 0 if the constraint is satisfied and a positive value proportional to the square of the violation if the constraint is violated.

$$P1=\text{Max}\,(0, \beta_k^t + \sum_{j\in C}\, t_{kj} C_{kj} - U)^2 \tag{21}$$

$$P2=\text{Max}\,(0, V_k^t + \sum_{i\in A}\, C_{ik} - h_k)^2 \tag{22}$$

$$P3=\text{Max}\,(0, \sum_{i\in A}\sum_{i\in A}\, C_{ik} Pr_i - \sum_{i\in A}\sum_{i\in A}\, C_{ik} Pr_j)^2 \tag{23}$$

$$\text{P} = P1 + P2 + P3 \tag{24}$$

The fitness function F' define in Eq. 25 is computed by integrating both the objective function and the penalty term, as follows:

$$F' = F + \lambda . P \tag{25}$$

where λ is a penalty coefficient used to strongly penalize constraint violations.

Transfer function method: Transfer functions are essential for converting the continuous outputs from SMA into discrete decisions that align with the binary representation. This adaptation is crucial because the SMA inherently operates as a continuous metaheuristic algorithm, producing continuous values. The primary advantage of utilizing transfer functions lies in their ability to retain the continuous structure of the algorithm while enabling its application to discrete scenarios. Our research involves systematic experimentation with three well-known transfer functions to pinpoint the most effective ones for steering the search process, as delineated in Table 3.

Table 3. Transfer functions for GPU-SMA adaptation.

Name	Definition of the transfer function
S-shaped [34]	$T(x) = \frac{1}{1+e^{-x}}$
V-shaped [35]	$T(x) = \left\lvert \frac{x}{\sqrt{1+x^2}} \right\rvert$
Z-shaped [36]	$T(x) = \sqrt{1-2x}$

4.5. Computational Complexity Analysis

Table 4 compares the computational complexities of the original SMA and GPU-SMA. This comparison highlights how leveraging GPU technology can significantly reduce the computational complexity of various components within the SMA. The complexity analyzed is influenced by the dimension of the problem D, the population size N, the maximum number of iterations T, the number of threads K ,and C the cost of data transfer from CPU to GPU. The complexity of the GPU-SMA is directly affected by the number of threads K. If K is sufficient ($N \times D < K$), the complexity of all steps becomes $O(1)$ due to full parallelization. Otherwise, Table 4 provides the complexity when K is insufficient ($N \times D \geq K$).

Table 4. Comparative analysis of computational complexities: CPU-SMA and GPU-SMA.

Component	SMA complexity	GPU-SMA complexity
Initialization	$O(N \times D)$	$O(N \times D/K) + C$
Fitness evaluation	$O(N \times D)$	$O(N/K)$
Weight update	$O(N \times D)$	$O(N/K)$
Solution update	$O(N \times D)$	$O(N \times D/K)$
Overall complexity	$O(N \times D + T \times N \times (D + \log N))$	$O(N/K \times (D + T \times (D + \log N)) + C$

5. Experimental Study

This section evaluates the GPU-SMA approach for solving the ambulance dispatching and relocation problem during disasters. We specifically focused on the COVID-19 pandemic scenario due to its exceptional strain on emergency services. This scenario provided a critical and realistic test environment for assessing the effectiveness of the proposed method in managing emergency logistics under extreme conditions. We start by giving a summary description of the datasets from Chicago and its hospital network. Subsequently, we introduce the performance evaluation metrics and parameter settings. Finally, we compare the performance of GPU-SMA with the original SMA [6] and other popular swarm intelligence algorithms such as PSO [37], APSO [38], HHO [39], FA [40], and BA [41]. For each algorithm, we replicate the processes used in the proposed approach, developing GPU-accelerated versions (GPU-PSO, GPU-APSO, GPU-HHO, GPU-FA, and GPU-BA) to ensure fair and consistent comparisons across all methods.

5.1. COVID-19 DCDH Dataset

The COVID-19 Daily Cases, Deaths, and Hospitalizations dataset (COVID-19 DCDH) is an important resource in the health and human services category. It contains 9781 items, each representing a distinct week, and 21 columns providing information on various aspects of the data. This dataset offers a comprehensive daily overview of the impact of the pandemic. It is particularly useful for addressing

challenges such as ambulance dispatching, as it includes the total number of COVID-19 cases and population in each zone.

5.1.1. Emergencies Modeling

We have utilized the COVID-19 DCDH dataset to create a new dataset of emergency calls that are randomly distributed across 144 time periods within a single day. We chose a day during the peak week of COVID-19 cases from 12/26/2021 to 1/1/2022, during which ambulance dispatching decisions were critical for patients and the healthcare system. Each entry in the dataset represents an emergency call with a specific geographic location and priority level. We have analyzed the distribution of COVID-19 cases in each zone, resulting in a total of 6837 calls within the day. These calls have been modeled using a Non-Homogeneous Poisson Distribution (NHPD) and a Normal Distribution (ND) to mimic the varying call rates over time. Figure 3 illustrates the daily variation in the volume of emergency calls throughout the day.

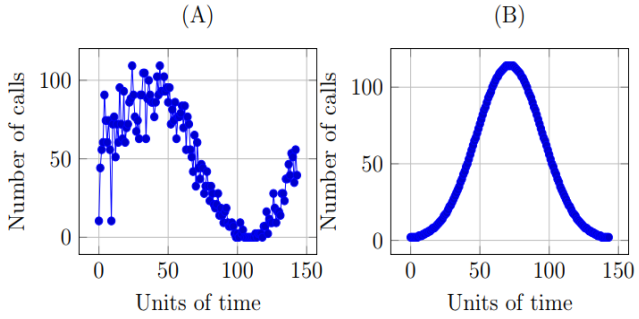

Figure 3. Distribution of emergency calls over 144 time units. (**A**) Represents the graph of the Non-Homogeneous Poisson Distribution. (**B**) Represents the graph of the Normal Distribution.

It is crucial to take into account the age of the patient when determining the appropriate level of emergency response required. We have conducted an analysis of the number of cases by age from the COVID-19 DCDH dataset. To help illustrate how the system works in real-world situations, we have provided two scenarios in Table 5. In the first scenario, emergency calls involving patients over the age of 45 are given the highest priority, while in the second scenario, patients over the age of 60 are prioritized with a high sense of urgency.

Table 5. Age-Based priority classification for emergency call response.

	Low Priority	**Mid Priority**	**High Priority**
Scenario 1	$age < 35$	$30 < age < 45$	$age > 45$
Scenario 2	$age < 45$	$45 < age < 60$	$age > 60$

5.1.2. Ambulance Shift Modeling

Models for ambulances were developed using actual data from 43 different hospital locations in Chicago. Each ambulance is distinguished by its geographic location, a unique identifier, and the identifier of the hospital it belongs to. This system enables efficient tracking and allocation of ambulance resources, improving emergency response logistics. Figure 4 illustrates the distribution of hospitals and emergency calls across Chicago on the selected peak day.

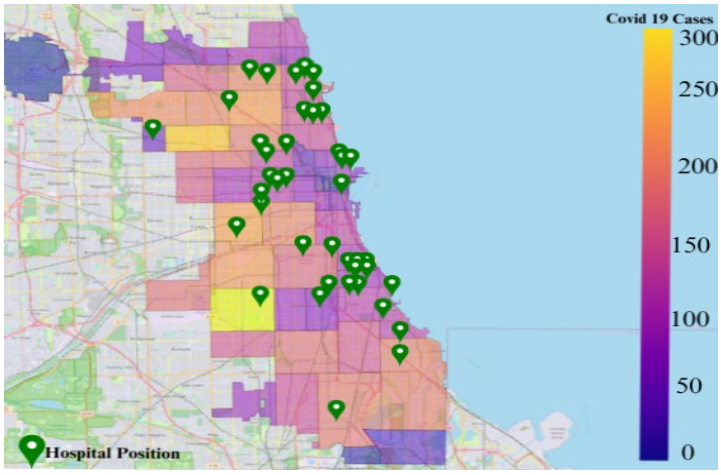

Figure 4. Chicago hospitals and COVID-19 cases distribution.

5.1.3. Instances Description

Experiments were conducted to assess the effectiveness of GPU-SMA in various situations. Instances were created to address the ADRP, varying in terms of the number of ambulances available, the frequency distribution of calls throughout the day, and the priority levels assigned to the calls. This resulted in twelve unique datasets for testing purposes. Table 6 presents the instances generated for the experiment.

Table 6. Description of scenarios utilized in the experimental section.

		Number of ambulances per hospital		
Priority classification	**Calls distribution**	**2**	**4**	**6**
Scenario 1	ND	Instance I1	Instance I2	Instance I3
	NHPD	Instance I4	Instance I5	Instance I6
Scenario 2	ND	Instance I7	Instance I8	Instance I9
	NHPD	Instance I10	Instance I11	Instance I12

5.2. Evaluation Metrics

To evaluate the performance of the proposed GPU-SMA for the ADRP, we consider different evaluation metrics and utilize a statistical test. This approach allows us to systematically assess the efficacy of our method in comparison to other methods.

5.2.1. Fitness Metrics

The evaluation focuses on fitness values, as discussed in Section 4.4. It includes the overall fitness function and sub-objective functions: predicted travel time, priority response, and relocation time. Detailed results for each aspect enable a thorough comparison of our algorithm with other approaches.

5.2.2. Execution Time

The performance evaluation of an algorithm and the design of a faster one primarily depend on its execution time. To examine the impact of GPU technology on SMA, we need to compare the execution time of GPU-SMA with that of traditional sequential CPU (CPU-SMA) implementations. To determine the speedup achieved through GPU optimization, we calculate the ratio of the processing times of a serial CPU implementation to the execution time of the GPU.

5.2.3. Statistical Test

We employ the nonparametric Friedman test to identify significant performance differences across twenty-one algorithms, encompassing seven main algorithms with varying transfer functions. By ranking the performance outcomes of each algorithm under varied conditions, the test enables us to ascertain the presence of statistically significant variances [42]. The foundational assumption of the Friedman test, known as the null hypothesis, posits that no substantive differences exist across the algorithms, attributing any observed variations to chance. The test statistic is given according to Eq. 26 as follows:

$$F_f = \frac{12}{bk(k+1)} \sum_{i=1}^{k} (r^2 - 3b(k+1)) \quad (26)$$

where b represents the number of blocks or attempts, k is the number of treatments, and r denotes the rank sum for the i^{th} treatment across all subjects. In our study, the treatments are the different metaheuristic algorithms, and the blocks are their performance on the twelve datasets. The null hypothesis asserts that the performance measures of the algorithms are equivalent across the datasets. In contrast, the alternative hypothesis suggests that at least one algorithm performs differently.

5.3. Experimental Environment and Parameter Settings

All algorithms are implemented using C++ and the Visual Studio 2019 integrated development environment on Windows 10. The GPU versions of the algorithms are compiled using NVIDIA CUDA version 12.1. After conducting experiments, we have standardized the number of threads per block to 256 for the CUDA implementation. Table 7 shows the computing platforms used in this paper. Both the GPU-SMA and CPU-SMA algorithms are tested under identical conditions. We conducted extensive experiments with various parameter values for the algorithms and selected the optimal ones that significantly impact their performance. These values are presented in Table 8.

Table 7. Computing platforms utilized in the experiments.

Name	Model
CPU-SMA	Intel i7-8750H @ 2.20 GHz
GPU-SMA	GPU1-SMA: NVIDIA GPU RTX A4000
	GPU2-SMA: NVIDIA GPU GeForce RTX 3090

Table 8. Parameter settings for different algorithms.

Algorithm	Parameter	Description	Best Value
GPU-SMA	z	Control parameter	0.13
CPU-SMA	z	Control parameter	0.13
GPU-PSO	w	Inertia weight	0.21
	α	Learning parameter	2.3
	β	Learning parameter	4.1
GPU-APSO	α_0	Initial randomization parameter	0.9
	β_0	Initial randomization parameter	0.6
GPU-HHO	β	Levy flight exponent	1.39
GPU-FA	γ	Light absorption coefficient	0.15
	α	Randomization parameter	0.41
	β_0	Attractiveness parameter	1.42
GPU-BA	$f_{\min}$	Lower bound frequency	0
	$f_{\max}$	Upper bound frequency	2
	α	Initial Loudness parameter	0.98
	γ	Initial Pulse Rate	0.44

5.4. Experimental Results

We conducted two sets of experiments. First, we compared the running times of GPU-SMA and CPU-SMA under various conditions. Second, we evaluated the performance of our proposed algorithm in comparison to other GPU-accelerated algorithms, utilizing twelve datasets and three transfer functions. The results were quantified using the performance metrics described earlier.

Table 9 shows the execution times for GPU1-SMA, GPU2-SMA and CPU-SMA across varying population sizes (32, 64, 128) and different maximum iteration settings (50, 100, 150). Some important information can be seen from Table 9, the GPU version of SMA is significantly better than the serial CPU version of SMA in terms of computing efficiency when solving the ADRP. As a case in point, for instance I1 at 150 max iterations, the CPU-SMA takes 6255 seconds at a population size of 128, whereas GPU1-SMA and GPU2-SMA show a drastic reduction in time with 255 seconds and 245 seconds respectively. This significant decrease highlights the efficiency of GPU architectures in handling large-scale computations through parallel processing. The table also includes a best speed up metric, which compares the performance improvement of GPU-SMA over CPU-SMA. The GPU models consistently exhibit a significant speed advantage, with speedups typically ranging around 20-fold improvement over CPU. This underscores the ability of GPU utilization to handle complex calculations more efficiently.

Table 9. Comparative analysis of SMA running times across CPU and GPU implementations based on varying population sizes.

		Execution time (s)										
		CPU-SMA			GPU1-SMA			GPU2-SMA				
		Population Size			Population Size			Population Size			Best speed up	
Instance	Iterations	**32**	**64**	**128**	**32**	**64**	**128**	**32**	**64**	**128**	**GPU1**	**GPU2**
I1	50	222	383	861	40	51	62	43	59	68	12	13
	100	1469	2738	3986	48	99	185	39	81	179	20	21
	150	3888	4167	6255	108	145	255	98	140	245	20	21
I2	50	315	793	1691	48	59	90	39	51	85	19	20
	100	1843	2329	3898	67	85	190	60	83	182	20	21
	150	3732	4452	6598	95	105	298	84	91	290	20	23
I3	50	450	852	1736	52	69	106	40	54	89	16	20
	100	1521	2405	3932	69	94	208	61	98	198	18	19
	150	3022	4923	6969	98	184	315	92	181	304	20	21
I4	50	321	458	896	41	58	95	36	48	90	9	10
	100	1395	2542	3014	54	75	184	48	62	175	17	18
	150	3765	4954	6312	61	185	274	52	175	268	20	21
I5	50	485	647	1648	45	61	104	39	51	92	16	17
	100	1524	2254	3784	59	85	204	41	78	197	18	19
	150	3658	4525	6524	78	190	304	62	185	294	21	22
I6	50	584	1025	1985	51	78	110	45	70	102	18	19
	100	1524	2852	3845	63	89	208	56	81	198	18	19
	150	3745	4014	6758	94	201	325	90	194	316	20	21
I7	50	221	376	896	25	45	50	23	45	49	19	20
	100	1465	2735	3159	40	57	191	40	84	174	18	19
	150	3854	4958	6254	104	146	259	95	139	248	20	21
I8	50	312	795	1684	45	60	89	40	49	84	19	20
	100	1845	2365	3893	66	84	184	59	84	180	18	19
	150	3715	4462	6295	95	102	296	89	95	105	20	21
I9	50	435	856	1750	55	71	105	39	53	87	17	18
	100	1522	2400	3940	67	84	185	60	84	195	18	19
	150	3023	4965	6569	98	185	315	98	154	302	20	21
I10	50	322	465	895	42	51	102	34	49	90	9	10
	100	1385	2552	3047	54	74	285	42	79	174	11	12
	150	3754	4965	6314	64	187	274	54	178	266	21	20
I11	50	487	649	1698	47	65	103	36	55	95	16	18
	100	1597	2258	3786	55	85	205	42	79	198	14	14
	150	3659	4526	6529	79	185	302	69	180	295	21	20
I12	50	584	1028	1866	52	69	105	46	65	95	18	20
	100	1526	2263	3896	64	87	209	56	81	199	14	15
	150	3735	4018	6788	96	206	326	91	193	315	21	20

As shown in Table 9, the execution time increases with the number of ambulances per hospital. Instances I3, I6, I9, and I12, which have the highest number of ambulances, also show the highest execution times. Notably, the performances of the two GPU models differ, with GPU-2 consistently producing superior results. For example, the most considerable speedup observed was generated by GPU-2, suggesting a performance 23 times greater than that of the CPU.

In the second part of testing, we compare the results obtained by GPU-SMA with those from CPU-SMA and other GPU-based swarm intelligence algorithms, including GPU-PSO, GPU-APSO, GPU-HHO, GPU-FA, and GPU-BA. Each method was run independently 40 times. Following the initial experiments, we set the population size to 64 and the maximum number of iterations to 200 for all methods. Tables 10, 11, and 12 present detailed results of the global fitness and its sub-objective functions: Response time, Priority response, and Relocation time across the twelve instances, represented by I1 to I12. Note that all the sub-objective functions have to be minimized. For each instance, the three tables display the average values calculated over 144 unit times to facilitate a comprehensive comparison of the different methods. The Time column denotes the total running time for each approach. Additionally, each algorithm was tested using three different transfer functions: S-shape, V-shape, and Z-shape, which are essential when employing continuous metaheuristics, as in this study.

Table 10. Performance results of GPU-SMA and other algorithms across twelve instances using the S-shape transfer function.

		Instance											
Algorithm	Criteria	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12
GPU-SMA	Response time	189	308	336	**198**	**319**	**340**	**185**	301	**334**	**200**	317	320
	Priority response	**130**	**231**	241	**134**	**221**	**219**	**127**	**211**	**230**	**135**	**218**	**221**
	Relocation time	**36**	**48**	102	**36**	**47**	**98**	**45**	98	152	96	142	**156**
	Global fitness	**154**	**272**	293	**155**	**274**	**297**	**151**	270	294	155	272	276
	Time	**220**	262	**281**	244	**271**	**254**	232	279	303	237	274	**284**
CPU-SMA	Response time	**188**	**305**	**335**	201	339	342	187	**300**	338	204	**316**	**319**
	Priority response	131	232	**234**	136	224	225	130	211	235	217	219	221
	Relocation time	47	111	**121**	49	96	103	46	**97**	154	**94**	143	157
	Global fitness	155	271	**291**	157	282	285	151	**268**	**289**	**158**	**269**	**273**
	Time	6733	7757	8659	7320	8130	8250	6960	8370	9090	7110	8220	8520
GPU-PSO	Response time	216	340	363	236	349	416	339	359	371	334	342	344
	Priority response	154	263	269	154	246	243	149	231	254	159	263	266
	Relocation time	56	76	116	58	69	119	64	119	133	167	176	183
	Global fitness	195	294	316	177	283	343	243	271	286	293	285	281
	Time	264	293	314	183	273	314	230	247	296	213	243	299
GPU-APSO	Response time	201	325	348	221	334	401	324	344	356	319	327	329
	Priority response	139	248	254	139	231	228	134	216	239	144	248	251
	Relocation time	41	61	101	43	54	104	49	104	118	102	**121**	168
	Global fitness	166	286	308	158	276	330	225	268	299	225	278	279
	Time	262	291	312	**181**	271	312	**228**	**245**	**294**	**211**	**241**	297
GPU-HHO	Response time	194	312	335	201	324	391	309	339	349	305	318	325
	Priority response	134	236	249	135	225	220	129	212	234	139	221	229
	Relocation time	39	51	110	42	51	101	46	103	**116**	99	149	159
	Global fitness	162	280	302	159	272	320	210	274	289	206	269	289
	Time	221	**261**	283	243	275	256	231	281	301	238	275	285
GPU-FA	Response time	195	315	338	211	329	398	319	340	351	309	317	326
	Priority response	136	246	250	137	229	225	130	214	238	140	241	249
	Relocation time	39	51	110	42	51	101	46	103	116	99	149	159
	Global fitness	160	277	297	159	274	319	210	**262**	290	261	268	278
	Time	332	361	382	251	341	382	298	315	364	281	311	367
GPU-BA	Response time	236	362	380	260	370	431	357	375	396	357	362	363
	Priority response	175	285	284	177	264	260	173	247	279	178	283	284
	Relocation time	79	96	138	76	84	136	83	135	158	141	157	206
	Global fitness	206	320	338	192	302	353	260	294	316	259	290	301
	Time	338	369	390	257	349	391	304	323	370	289	320	375

Table 11. Performance results of GPU-SMA and other algorithms across twelve instances using the V-shape transfer function.

		Instance											
Algorithm	Criteria	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12
GPU-SMA	Response time	**203**	**319**	**345**	**211**	**333**	**350**	**201**	**317**	**346**	**213**	**329**	**330**
	Priority response	**141**	**243**	254	148	**231**	**234**	**138**	**223**	**243**	**149**	**228**	**236**
	Relocation time	**48**	**61**	**116**	**46**	**62**	**109**	**57**	111	166	**106**	**157**	**167**
	Global fitness	**160**	**252**	**281**	**165**	**258**	**279**	**160**	**257**	**289**	**179**	**274**	**279**
	Time	**220**	**262**	**281**	244	**270**	255	231	276	305	235	272	285
CPU-SMA	Response time	206	328	354	217	358	359	206	325	354	220	333	335
	Priority response	141	251	**254**	**146**	245	240	148	229	253	234	236	239
	Relocation time	60	125	135	67	107	116	60	**108**	169	110	158	172
	Global fitness	164	272	290	173	284	287	165	262	297	201	279	283
	Time	6735	7754	8661	7322	8132	8251	6961	8375	9092	7110	8221	8522
GPU-PSO	Response time	257	389	408	283	398	464	389	406	418	382	391	392
	Priority response	195	301	318	200	296	290	198	277	300	209	310	312
	Relocation time	97	119	163	104	115	165	107	166	178	161	186	227

	Global fitness	183	293	317	178	297	299	179	286	317	181	285	288
	Time	263	291	314	182	274	315	231	**242**	**291**	**212**	**241**	298
GPU-APSO	Response time	243	373	397	266	380	445	369	393	400	362	374	378
	Priority response	185	295	298	186	276	273	179	261	285	194	298	300
	Relocation time	85	109	149	87	99	152	95	149	161	144	166	215
	Global fitness	223	325	351	230	328	392	276	339	357	276	321	336
	Time	262	291	314	**181**	272	312	**229**	245	294	212	242	297
GPU-HHO	Response time	236	362	380	260	370	431	357	375	396	357	362	363
	Priority response	175	285	284	177	264	260	173	247	279	178	283	284
	Relocation time	79	96	138	76	84	136	83	135	158	141	157	206
	Global fitness	206	320	338	192	302	353	260	294	316	255	290	301
	Time	221	262	285	243	270	**246**	241	282	305	239	270	**281**
GPU-FA	Response time	230	353	378	241	368	430	356	376	385	340	350	365
	Priority response	176	279	281	171	261	263	169	250	268	177	276	288
	Relocation time	71	90	146	79	84	135	77	141	**146**	139	184	198
	Global fitness	197	316	335	193	310	350	242	295	320	239	287	288
	Time	331	362	382	251	342	387	295	315	364	282	311	368
GPU-BA	Response time	276	399	412	290	408	466	390	406	432	391	401	400
	Priority response	207	321	322	207	301	300	206	278	313	213	322	316
	Relocation time	116	136	169	114	117	166	122	171	192	176	189	243
	Global fitness	244	350	372	223	339	392	292	327	352	295	325	334
	Time	335	369	391	257	340	391	311	323	370	288	321	375

Based on the results presented in Tables 10, 11, and 12, it is clear that the proposed algorithm outperforms CPU-SMA in terms of Response time. The proposed algorithm produces the best response time values compared to CPU-SMA in seven out of twelve instances when using the S-shape transfer function, ten out of twelve instances when using the V-shape transfer function, and six out of twelve datasets when using the Z-shape transfer function. The Z-shape function consistently generates the best response time values across all instances, with a response time of 181 minutes for GPU-SMA compared to 187 minutes for CPU-SMA in instance I7. Furthermore, the proposed method shows superior performance compared to other GPU-based algorithms in all instances.

Table 12. Performance results of GPU-SMA and other algorithms across twelve instances using the Z-shape transfer function.

		Instance											
Algorithm	Criteria	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12
GPU-SMA	Response time	**186**	**308**	329	**193**	**319**	**337**	**181**	302	331	**200**	310	316
	Priority response	125	228	241	**130**	**218**	**214**	123	208	**225**	135	214	**218**
	Relocation time	**32**	45	102	**31**	**44**	94	40	95	152	**92**	137	**153**
	Global fitness	**143**	239	**266**	**148**	**243**	**263**	141	241	274	**165**	256	**263**
	Time	221	**262**	**282**	243	271	**251**	**232**	279	301	238	**241**	**281**
CPU-SMA	Response time	187	308	**328**	196	335	339	187	**301**	**330**	201	**309**	**315**
	Priority response	128	228	**229**	136	221	221	**122**	**208**	231	212	216	221
	Relocation time	43	106	118	49	92	100	**39**	**93**	**151**	94	138	153
	Global fitness	146	251	266	154	262	265	**140**	**239**	**273**	180	**255**	264
	Time	6734	7755	8659	7321	8135	8255	6960	8396	9091	7112	8224	8520
GPU-PSO	Response time	212	337	363	231	345	413	339	354	367	331	337	344
	Priority response	149	260	265	154	241	240	145	226	251	159	258	262
	Relocation time	53	72	116	53	66	115	59	119	130	113	131	183
	Global fitness	192	296	325	171	286	354	239	259	274	227	258	273
	Time	265	292	314	184	273	311	232	247	296	214	244	298
GPU-APSO	Response time	198	320	348	217	329	398	320	344	353	314	323	329
	Priority response	**124**	245	249	139	227	225	129	212	239	141	243	247
	Relocation time	38	56	**101**	39	49	101	45	104	115	97	117	168
	Global fitness	162	274	310	173	286	345	224	264	302	212	263	293
	Time	261	290	311	**181**	**270**	312	227	**245**	**294**	**210**	242	295

GPU-HHO	Response time	189	309	335	197	319	388	305	339	344	302	314	325
	Priority response	131	**221**	245	135	222	215	125	212	231	**134**	**214**	229
	Relocation time	35	**42**	110	37	48	**95**	41	103	113	94	145	159
	Global fitness	146	**238**	272	152	245	294	216	266	275	226	260	272
	Time	**220**	262	285	243	276	256	232	284	301	239	270	282
GPU-FA	Response time	191	310	338	208	325	393	319	337	347	304	314	326
	Priority response	133	242	245	137	226	221	125	214	235	136	236	249
	Relocation time	34	48	106	42	46	98	42	103	111	96	145	159
	Global fitness	159	261	307	171	276	328	212	261	295	205	261	273
	Time	330	362	382	250	340	382	299	315	364	282	312	368
GPU-BA	Response time	232	359	380	255	366	428	357	370	392	354	357	363
	Priority response	172	281	279	177	261	256	168	247	276	174	278	284
	Relocation time	74	93	138	72	79	133	79	135	153	138	153	206
	Global fitness	194	303	325	189	295	343	238	273	318	238	273	299
	Time	335	364	391	258	348	391	305	323	370	288	321	377

As shown in Tables 10, 11, and 12, we can see that GPU-SMA consistently prioritizes critical tasks with the Priority response metric versus CPU-SMA in nine out of twelve instances when using the S-shape transfer function, seven out of twelve instances when using the V-shape transfer function, and eight out of twelve datasets when using the Z-shape transfer function. The best priority response was generated by CPU-SMA in instance I7 with 122 responses, outperforming GPU-SMA. GPU-HHO also produced competitive results, with the Z-shape transfer function generating the best results in three out of twelve instances.

It can be observed from Tables 9, 10, and 11 that GPU-SMA shows the best Relocation time when compared to CPU-SMA in nine out of twelve instances while using the S-shape transfer function, eleven out of twelve instances with the V-shape transfer function, and nine out of twelve datasets with the Z-shape transfer function. GPU-SMA generated the best Relocation time in Instance I4, which took 31 minutes using the Z-shape transfer function. Additionally, GPU-HHO also produced favorable outcomes using the Z-shape transfer function in two out of twelve instances.

The results presented in Tables 10, 11, and 12 demonstrate that GPU-SMA performs better than CPU-SMA regarding Global fitness values. Specifically, GPU-SMA surpasses CPU-SMA in six out of twelve scenarios using the S-shape transfer function, in all instances with the V-shape, and in nine out of twelve for the Z-shape. Notably, CPU-SMA records the best global fitness score in instance I7 with the Z-shape transfer function. The GPU-SMA outperforms the CPU-SMA in instances I3, I6, and I12 when using the Z-shape transfer function, indicating its robustness in handling the ADRP as the problem scale increases. It can also be observed from Tables 10, 11, and 12 that the total running times highlight the efficiency of GPU-SMA, showcasing significant speedups compared to CPU-SMA and modest improvements over other GPU algorithms. GPU-SMA outperforms CPU-SMA in all instances, with a substantial speedup across all transfer functions, averaging around 30.

Figure 5 demonstrates the progression of global fitness and its components over iterations for 110 emergency calls. The results show that using the Z-shaped transfer function significantly improves both the exploration and exploitation phases, leading to better performance. Notably, GPU-SMA achieves faster convergence to the optimal solution compared to its competitors. When using the S-shaped transfer function, there is a noticeable slow start in convergence, but it still manages to achieve moderate results by the end. For the V-shaped transfer function, most approaches experience periods of stagnation, which indicates that achieving improvements is a challenge.

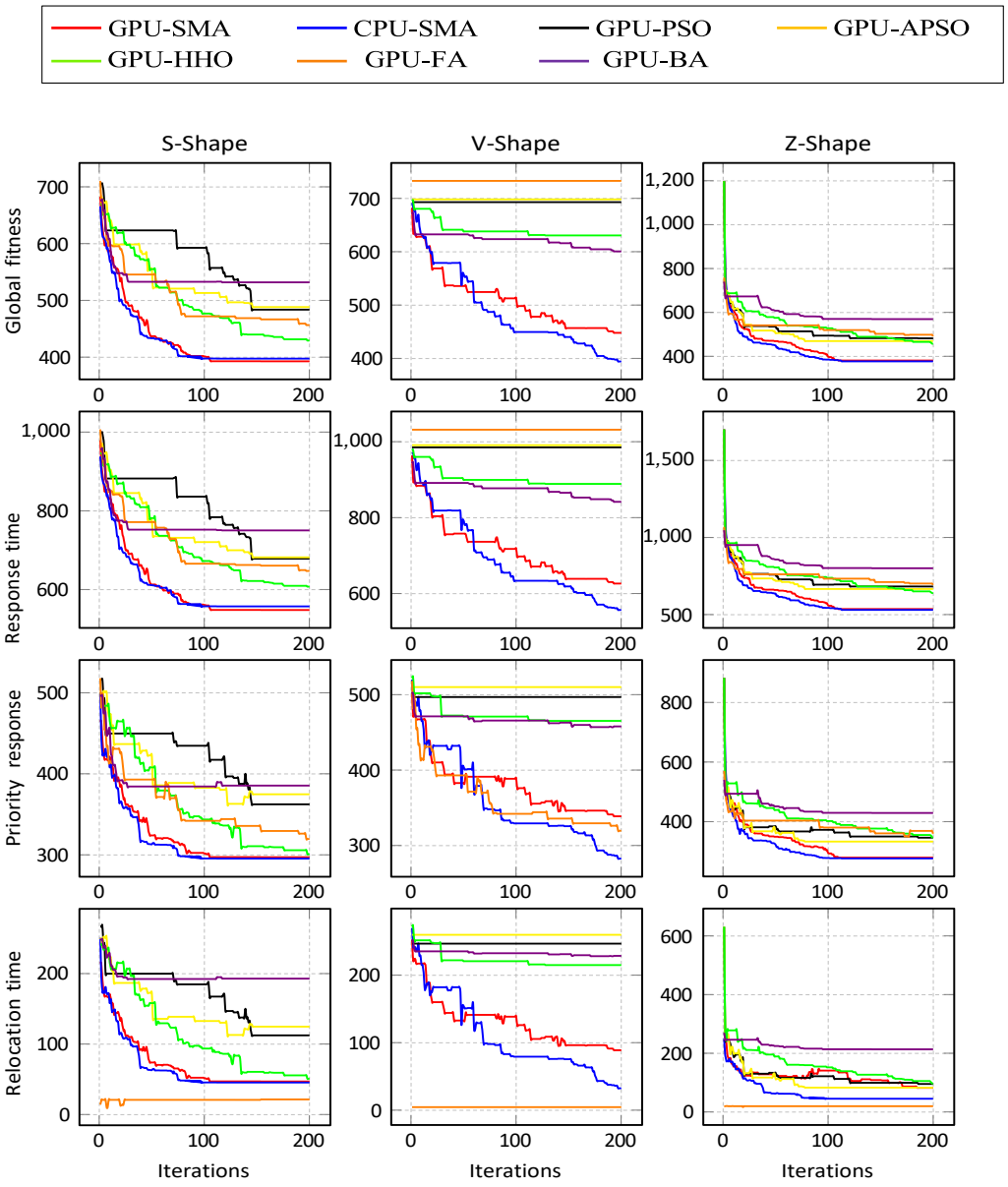

Figure 5. Comparison of fitness score convergence for GPU-SMA and other algorithms in scenario with 110 emergency calls.

To evaluate the statistical significance of the obtained results, we applied the Friedman test to compare the performance of algorithms as presented in Tables 10, 11, and 12. Table 13 outlines the Chi-square values and p-values for the five performance metrics. The high Chi-square values and very low p-values lead to the rejection of the null hypothesis for each metric, confirming the statistical significance of the differences observed.

Table 13. Chi-square test results for various performance metrics.

Metric	Chi-square	p-value	Hypothesis Result
Response time	222.44	3.89×10^{-36}	Rejected
Priority response	215.53	9.30×10^{-35}	Rejected
Relocation time	194.66	1.28×10^{-30}	Rejected
Global fitness	187.91	2.73×10^{-29}	Rejected
Time	207.51	3.66×10^{-33}	Rejected

5.5. Discussion

The experimental results demonstrate that the parallelization strategy employed in GPU-SMA not only significantly reduces overall computational time but also maintains, and in some cases enhances,

the performance of the CPU-based SMA. This improvement in convergence is primarily attributed to the superior ability of GPU-SMA to explore the search space more effectively, facilitated by the parallel generation of random numbers across multiple CUDA random states. This parallelization enhances population diversity, providing a distinct advantage when tackling large and complex problem spaces.

However, despite these advantages, the performance of the algorithm is influenced by hardware constraints, which can restrict scalability as the problem size grows significantly. For instance, as the population size or problem dimension increases, GPU memory limitations and communication bottlenecks can lead to diminishing returns in performance. This is particularly evident when the number of threads exceeds GPU capacity, forcing the use of slower global memory.

6. Conclusion

This paper presents a notable advancement in addressing the Ambulance Dispatching and Relocation Problem (ADRP) through the introduction of a parallel implementation of the Slime Mould Algorithm (GPU-SMA) on a GPU architecture. The GPU-SMA framework effectively addressed the multi-objective purposes of minimizing patient response times, prioritizing emergency calls, and decreasing ambulance relocation costs. Unlike traditional methods that only parallelize specific components of an algorithm, a holistic approach executes all aspects of the SMA on the GPU, maximizing overall performance. Additionally, a novel strategy for ambulance relocation prioritizes the zones most affected by the event.

By leveraging the parallel GPU implementations of the SMA, real-time decision-making in disaster scenarios is significantly enhanced. The primary advantages of the parallel algorithm include the reduction of computational complexity through the simultaneous processing of multiple tasks, leading to faster execution times and the capability to handle more significant instances of the problem. Furthermore, the observed improvements in execution times due to GPU utilization demonstrate the effectiveness of this approach in maintaining high performance in time-sensitive situations. The GPU-SMA addresses the challenge of preserving solution quality compared to the sequential method. Utilizing CUDA to generate random states increases the diversity of solutions, resulting in better qualitative outcomes.

In the experiments, we utilize data from Chicago and COVID-19 case statistics across different zip codes to simulate emergency demand in natural disasters. We generate twelve datasets that vary the call distribution and propose a system to classify the priority of emergency calls and adjust the number of ambulances at each hospital. Compared with the sequential version of SMA and other GPU-based swarm intelligence algorithms, the proposed algorithm consistently demonstrates its effectiveness and superiority regarding both efficacy and computational complexity. These results provide reassurance in the potential of our approach to improve emergency response services.

The presented model could be an advanced tool for planning resources within healthcare infrastructure and could be further adapted for other emergency response services. We plan to implement more detailed operational constraints, such as hospital capacities and specific ambulance models. In future work, we aim to explore comparisons with multi-core CPU-based methods and extend testing to platforms equipped with multi-GPU configurations to tackle more complex problems. Further enhancements will optimize memory management and reduce thread divergence to maximize performance.

Author contributions
Supervision H.D. and I.K.; Concept and Design C.K., H.D., K.E and I.K.; Data Collection and/or Processing C.K., K.E and I.K.; Analysis and Interpretation C.K.; Literature Search C.K.; Manuscript Writing C.K.; Critical Review H.D. and I.K. All authors have read and agreed to the published version of the manuscript.

Funding
The authors declare that no funding was obtained for this study.

Conflicts of interest
The authors declare that they have no conflicts of interest.

Data availability statement
Currently, the data used in this study is not shared publicly. It will be made available online following the publication of the article.

References

1. L. Pratiwi, Y.-H. Choo, A. K. Muda, and S. F. Pratama, "Covariance Search Model for Identifying ncRNA using Particle Swarm Optimized Agglomerative Clustering," *Int. J. Comput. Inf. Syst. Ind. Manag. Appl.*, vol. 15, pp. 9–9, 2023.
2. J. Ferreira, R. Puga, J. Boaventura, A. Abtahi, and A. S. Santos, "Application of Bio-Inspired Optimization Techniques for Wind Power Forecasting," *Int. J. Comput. Inf. Syst. Ind. Manag. Appl.*, vol. 15, pp. 13–13, 2023.
3. M. S. Shaikh, C. Hua, M. A. Jatoi, M. M. Ansari, and A. A. Qader, "Parameter Estimation of AC Transmission Line Considering Different Bundle Conductors Using Flux Linkage Technique," *IEEE Can. J. Electr. Comput. Eng.*, vol. 44, no. 3, pp. 313–320, 2021, doi: 10.1109/ICJECE.2021.3069143.
4. M. Suhail Shaikh *et al.*, "Optimal parameter estimation of 1-phase and 3-phase transmission line for various bundle conductor's using modified whale optimization algorithm," *Int. J. Electr. Power Energy Syst.*, vol. 138, p. 107893, Jun. 2022, doi: 10.1016/j.ijepes.2021.107893.
5. M. S. Shaikh, C. Hua, M. A. Jatoi, M. M. Ansari, and A. A. Qader, "Application of grey wolf optimisation algorithm in parameter calculation of overhead transmission line system," *IET Sci. Meas. Technol.*, vol. 15, no. 2, pp. 218–231, Mar. 2021, doi: 10.1049/smt2.12023.
6. S. Li, H. Chen, M. Wang, A. A. Heidari, and S. Mirjalili, "Slime mould algorithm: A new method for stochastic optimization," *Future Gener. Comput. Syst.*, vol. 111, pp. 300–323, 2020.
7. H. Chen, C. Li, M. Mafarja, A. A. Heidari, Y. Chen, and Z. Cai, "Slime mould algorithm: a comprehensive review of recent variants and applications," *Int. J. Syst. Sci.*, vol. 54, no. 1, pp. 204–235, Jan. 2023, doi: 10.1080/00207721.2022.2153635.
8. S. Memeti, S. Pllana, A. Binotto, J. Kołodziej, and I. Brandic, "A Review of Machine Learning and Meta-heuristic Methods for Scheduling Parallel Computing Systems," in *Proceedings of the International Conference on Learning and Optimization Algorithms: Theory and Applications*, in LOPAL '18. New York, NY, USA: Association for Computing Machinery, mai 2018, pp. 1–6. doi: 10.1145/3230905.3230906.
9. S. Tkatek, O. Abdoun, J. Abouchabaka, and N. Rafalia, "A Parallel Genetic Algorithm to Optimize the Massive Recruitment Process," *Int. J. Comput. Inf. Syst. Ind. Manag. Appl.*, vol. 13, pp. 8–8, 2021.
10. H. Ling, X. Zhu, T. Zhu, M. Nie, Z. Liu, and Z. Liu, "A Parallel Multiobjective PSO Weighted Average Clustering Algorithm Based on Apache Spark," *Entropy*, vol. 25, no. 2, p. 259, 2023.
11. A. Kechid and H. Drias, "Cultural coalitions detection approach using GPU based on hybrid bat and cultural algorithms," *Appl. Soft Comput.*, vol. 93, p. 106368, 2020.
12. C. Khelfa, I. Khennak, H. Drias, Y. Drias, Y. Belharda, and M. Smail, "Slime Mould Algorithm for Solving Ambulance Dispatching Problem," in *Proceedings of the 14th International Conference on Soft Computing and Pattern Recognition (SoCPaR 2022)*, vol. 648, A. Abraham, T. Hanne, N. Gandhi, P. Manghirmalani Mishra, A. Bajaj, and P. Siarry, Eds., in Lecture Notes in Networks and Systems, vol. 648. , Cham: Springer Nature Switzerland, 2023, pp. 822–831. doi: 10.1007/978-3-031-27524-1_80.
13. C. Khelfa, H. Drias, and I. Khennak, "Quantum Slime Mould Algorithm and Application to Urgent Transportation," in *Quantum Computing: Applications and Challenges*, vol. 2, H. Drias and F. Yalaoui, Eds., in Information Systems Engineering and Management, vol. 2. , Cham: Springer Nature Switzerland, 2024, pp. 77–90. doi: 10.1007/978-3-031-59318-5_7.
14. A. S. Carvalho, M. E. Captivo, and I. Marques, "Integrating the ambulance dispatching and relocation problems to maximize system's preparedness," *Eur. J. Oper. Res.*, vol. 283, no. 3, pp. 1064–1080, 2020.
15. I. Gago-Carro, U. Aldasoro, J. Ceberio, and M. Merino, "A stochastic programming model for ambulance (re)location–allocation under equitable coverage and multi-interval response time," *Expert Syst. Appl.*, vol. 249, p. 123665, Sep. 2024, doi: 10.1016/j.eswa.2024.123665.
16. M. K. Oksuz and S. I. Satoglu, "Integrated optimization of facility location, casualty allocation and medical staff planning for post-disaster emergency response," *J. Humanit. Logist. Supply Chain Manag.*, vol. 14, no. 3, pp. 285–303, Jun. 2024, doi: 10.1108/JHLSCM-08-2023-0072.
17. Y. Karpova, F. Villa, E. Vallada, and M. Á. Vecina, "Heuristic algorithms based on the isochron analysis for dynamic relocation of medical emergency vehicles," *Expert Syst. Appl.*, vol. 212, p. 118773, Feb. 2023, doi: 10.1016/j.eswa.2022.118773.
18. S.-N. Shetab-Boushehri, P. Rajabi, and R. Mahmoudi, "Modeling location–allocation of emergency medical service stations and ambulance routing problems considering the variability of events and recurrent traffic congestion: A real case study," *Healthc. Anal.*, vol. 2, p. 100048, Nov. 2022, doi: 10.1016/j.health.2022.100048.
19. B. Luvaanjalba and E. Y.-L. Wu, "Using Genetic Algorithm and Mathematical Programming Model for Ambulance Location Problem in Emergency Medical Service," *IEICE Trans. Inf. Syst.*, vol. E107-D, no. 9, pp. 1123–1132, Sep. 2024.
20. M. Hemici, D. Zouache, B. Brahmi, A. Got, and H. Drias, "A decomposition-based multiobjective evolutionary algorithm using Simulated Annealing for the ambulance dispatching and relocation problem during COVID-19," *Appl. Soft Comput.*, vol. 141, p. 110282, Jul. 2023, doi: 10.1016/j.asoc.2023.110282.
21. L. S. Bendimerad and H. Drias, "Intelligent contributions of the artificial orca algorithm for continuous problems and real-time emergency medical services," *Evol. Intell.*, vol. 17, no. 3, pp. 1491–1526, Jun. 2024, doi: 10.1007/s12065-023-00846-y.
22. M. Essaid, L. Idoumghar, J. Lepagnot, and M. Brévilliers, "GPU parallelization strategies for metaheuristics: a survey," *Int. J. Parallel Emergent Distrib. Syst.*, vol. 34, no. 5, pp. 497–522, Sep. 2019, doi: 10.1080/17445760.2018.1428969.

23. Y. Zhuo, T. Zhang, F. Du, and R. Liu, “A parallel particle swarm optimization algorithm based on GPU/CUDA,” *Appl. Soft Comput.*, vol. 144, p. 110499, 2023.
24. M. A. Alqarni, M. H. Mousa, and M. K. Hussein, “Task offloading using GPU-based particle swarm optimization for high-performance vehicular edge computing,” *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 10, pp. 10356–10364, 2022.
25. W. Han, H. Li, M. Gong, J. Li, Y. Liu, and Z. Wang, “Multi-swarm particle swarm optimization based on CUDA for sparse reconstruction,” *Swarm Evol. Comput.*, vol. 75, p. 101153, 2022.
26. C. Khelfa and I. Khennak, “A Survey on Recent Optimization Strategies in Ambulance Dispatching and Relocation Problems,” *Artif. Intell. Dr. Symp.*, vol. 1852, pp. 192–203, 2023, doi: 10.1007/978-981-99-4484-2_15.
27. J.-H. Cho, Y. Wang, R. Chen, K. S. Chan, and A. Swami, “A survey on modeling and optimizing multi-objective systems,” *IEEE Commun. Surv. Tutor.*, vol. 19, no. 3, pp. 1867–1901, 2017.
28. R. Nithiavathy, S. Janakiraman, and M. Deva Priya, “Adaptive Guided Differential Evolution-based Slime Mould Algorithm-based efficient Multi-objective Task Scheduling for Cloud Computing Environments,” *Trans. Emerg. Telecommun. Technol.*, vol. 35, no. 1, p. e4902, Jan. 2024, doi: 10.1002/ett.4902.
29. X. Zhou *et al.*, “Boosted local dimensional mutation and all-dimensional neighborhood slime mould algorithm for feature selection,” *Neurocomputing*, vol. 551, p. 126467, 2023.
30. M. Li *et al.*, “Exponential slime mould algorithm based spatial arrays optimization of hybrid wind-wave-PV systems for power enhancement,” *Appl. Energy*, vol. 373, p. 123905, Nov. 2024, doi: 10.1016/j.apenergy.2024.123905.
31. B. Hagedorn, A. S. Elliott, H. Barthels, R. Bodik, and V. Grover, “Fireiron: A Scheduling Language for High-Performance Linear Algebra on GPUs,” Mar. 13, 2020, *arXiv*: arXiv:2003.06324. Accessed: Oct. 12, 2024. [Online]. Available: http://arxiv.org/abs/2003.06324
32. A. Seewald, U. P. Schultz, E. Ebeid, and H. S. Midtiby, “Coarse-Grained Computation-Oriented Energy Modeling for Heterogeneous Parallel Embedded Systems,” *Int. J. Parallel Program.*, vol. 49, no. 2, pp. 136–157, Apr. 2021, doi: 10.1007/s10766-019-00645-y.
33. B. A. de Melo Menezes, L. F. de Araujo Pessoa, H. Kuchen, and F. Buarque De Lima Neto, “Parallelization Strategies for GPU-Based Ant Colony Optimization Applied to TSP,” in *Parallel Computing: Technology Trends*, IOS Press, 2020, pp. 321–330. doi: 10.3233/APC200057.
34. J. E. S. C. Custodio and M. A. Lejeune, “Spatiotemporal Data Set for Out-of-Hospital Cardiac Arrests,” *Inf. J. Comput.*, vol. 34, no. 1, pp. 4–10, Jan. 2022, doi: 10.1287/ijoc.2020.1022.
35. S. Mirjalili and A. Lewis, “S-shaped versus V-shaped transfer functions for binary particle swarm optimization,” *Swarm Evol. Comput.*, vol. 9, pp. 1–14, 2013.
36. N. H. Abd Rahman and A. F. Zobaa, “Integrated mutation strategy with modified binary PSO algorithm for optimal PMUs placement,” *IEEE Trans. Ind. Inform.*, vol. 13, no. 6, pp. 3124–3133, 2017.
37. D. Freitas, L. G. Lopes, and F. Morgado-Dias, “Particle Swarm Optimisation: A Historical Review Up to the Current Developments,” *Entropy*, vol. 22, no. 3, Art. no. 3, Mar. 2020, doi: 10.3390/e22030362.
38. M. A. Memon, S. Mekhilef, and M. Mubin, “Selective harmonic elimination in multilevel inverter using hybrid APSO algorithm,” *IET Power Electron.*, vol. 11, no. 10, pp. 1673–1680, Aug. 2018, doi: 10.1049/iet-pel.2017.0486.
39. A. A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, and H. Chen, “Harris hawks optimization: Algorithm and applications,” *Future Gener. Comput. Syst.*, vol. 97, pp. 849–872, 2019.
40. X.-S. Yang, “Firefly Algorithms for Multimodal Optimization,” in *Stochastic Algorithms: Foundations and Applications*, vol. 5792, O. Watanabe and T. Zeugmann, Eds., in Lecture Notes in Computer Science, vol. 5792. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 169–178. doi: 10.1007/978-3-642-04944-6_14.
41. X.-S. Yang, “A New Metaheuristic Bat-Inspired Algorithm,” in *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, vol. 284, J. R. González, D. A. Pelta, C. Cruz, G. Terrazas, and N. Krasnogor, Eds., in Studies in Computational Intelligence, vol. 284. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 65–74. doi: 10.1007/978-3-642-12538-6_6.
42. Vigya, S. Raj, C. K. Shiva, B. Vedik, S. Mahapatra, and V. Mukherjee, “A novel chaotic chimp sine cosine algorithm Part-I: For solving optimization problem,” *Chaos Solitons Fractals*, vol. 173, p. 113672, Aug. 2023, doi: 10.1016/j.chaos.2023.113672.