# Application Scheduling with Multiplexed Sensing of Monitoring Points in Multi-purpose IoT Wireless Sensor Networks

Mustafa Can Çavdar, Ibrahim Korpeoglu, and Özgür Ulusoy

***Abstract*—Wireless sensor networks (WSNs) play a crucial role in Internet-of-Things (IoT) systems serving a variety of applications. They gather data from specific sensor nodes and transmit it to remote units for processing. When multiple applications share a WSN infrastructure, efficient scheduling becomes vital. In our research, we address the problem of application scheduling in WSNs. Specifically, we focus on scenarios where applications request data from monitoring points within the coverage area of a WSN. We propose a shared-data approach that reduces the network's sensing and communication load by allowing multiple applications to use the same sensing data. To tackle the scheduling challenge, we introduce a genetic algorithm named GABAS and three greedy algorithms: LMPF, LMSF, and LTSF. These algorithms determine the order in which applications are admitted to the WSN infrastructure, considering various criteria. To assess the performance of our algorithms, we conducted extensive simulation experiments and compared them with standard scheduling methods. We also evaluated the performance of GABAS as compared to another genetic scheduling algorithm that has recently appeared in the literature. The overall experimental results show that the methods we propose outperform the compared approaches across various metrics, namely makespan, turnaround time, waiting time, and successful execution rate. In particular, our genetic algorithm proves to be highly effective in scheduling applications and optimizing the mentioned metrics.***

***Index Terms*—wireless sensor networks, Internet of Things, application scheduling, algorithms.**

## I. Introduction

Wireless sensor networks (WSNs) have become the key components of Internet-of-Things and smart environments due to improvements in sensing technologies, wireless communications, and mobile computing. Wireless sensor networks are heterogeneous systems consisting of sensor nodes that can collect different types of data from the points within their sensing range. The collected data can be processed at sensor nodes or higher-level distributed or centralized units, like base stations or cloud data centers.

The application types of WSNs are very broad. Some domains include smart cities, smart houses, and some other intelligent systems that are used in daily life. Smart city management is one of the major areas for which WSN applications are very useful. Intelligent parking systems [1] and noise monitoring in metropolitan areas [2] are two examples of the applications that a smart city can make use of. Other examples of WSN applications include disaster prevention systems, agriculture management, habitat monitoring, intelligent lighting control, and supply-chain monitoring [3].

Previously, WSNs were task-specific. A WSN was designed, developed, and optimized to support a single application. Another application deployment was impossible; therefore, most WSN resources were underutilized. However, recently, WSNs are started to be designed in a way that they can support multiple applications, similar to other systems. For instance, to a single city-wide WSN infrastructure, various types of applications such as air quality monitoring, noise monitoring, and crime detection can be deployed. Another example would be a single building-wide WSN, which can be used for both structural health monitoring [4] and fire disaster detection [5] at the same time. Various other applications can be run over such a WSN infrastructure, such as occupancy estimation and automatic air-conditioning control, without disturbing other applications.

In this paper, we focus on the management of a WSN operated by a single infrastructure provider. The network is available to application providers who want their applications to be admitted to the network for a certain amount of time. Admitted applications need some points monitored with specific data types, and the collected data from those points need to be processed in base stations and centralized units. While we focused on the application placement problem in our previous work [6] in a similar network structure, in this work, we deal with the application scheduling problem. Since the applications will use the network's resources for a particular time, it is crucial to schedule the applications (i.e., arrange the order of the admission of the applications into the network) efficiently and effectively. One of the critical parameters to minimize is the total execution time (makespan) of the applications, but there are other metrics that can be important, like average waiting time, turnaround time, and rate of completing the applications before deadlines, if any.

We propose several algorithms for application scheduling in wireless sensor networks. We first propose a genetic algorithm called GABAS that effectively schedules applications onto a sensor network. GABAS reduces the total execution time of the applications by both assigning monitoring points to sensor nodes and base stations and determining the admission order in the best possible way. We also propose three greedy algorithms, considering different criteria, which can be used when fast decisions are needed.

M. C. Çavdar, I. Korpeoglu and Ö. Ulusoy are with the Department of Computer Engineering, Bilkent University, Cankaya, Ankara, 06800, Turkey. E-mail: mustafa.cavdar@bilkent.edu.tr, korpe@cs.bilkent.edu.tr, oulusoy@cs.bilkent.edu.tr

We present a comprehensive evaluation of our innovative algorithms designed for application scheduling in Wireless Sensor Networks (WSNs). Through a rigorous series of simulation experiments, we have meticulously compared our algorithms with well-established standard scheduling methods. Our results unequivocally showcase the remarkable effectiveness of our algorithm, GABAS, in substantially reducing the overall execution time while consistently achieving exceptional performance across critical metrics such as average turnaround time, average waiting time, and the successful completion ratio. Furthermore, our suite of proposed greedy algorithms, including LMPF, LMSF, and LTSF, demonstrate striking advantages in terms of both speed and efficiency when juxtaposed with conventional scheduling algorithms like FCFS and SJF. Among these, LMPF emerges as the premier performer in terms of makespan optimization, while LMSF and LTSF outshine their counterparts, both in the greedy and standard algorithm categories, concerning waiting time, turnaround time, and successful execution rates. Additionally, our comparison with another genetic scheduling algorithm, HGECS [7], consistently positions GABAS as the superior choice across all metrics, underscoring its unmatched performance. In summary, our evaluation results unequivocally endorse GABAS as the most effective algorithm in reducing total execution time, average turnaround time, average waiting time, and enhancing the successful completion ratio.

The rest of the paper is organized as follows: Section II provides a contextual discussion for the motivation that drives the research presented in this paper. Section III gives and discusses the related work in literature. Section IV presents our network model and problem formulation. Section V describes our approach and algorithms in detail, and Section VI provides the results of our simulation experiments. Finally, Section VII concludes the paper.

## II. Contextualization and Discussion

In the context of a WSN capable of handling multiple applications, it becomes crucial to design and operate the network in a manner that ensures good quality of service for applications and satisfaction for application owners. This requirement often necessitates the implementation of a centralized mechanism and related policies. Software-Defined Networking (SDN) offers a powerful solution by enabling the management of a WSN through a centralized controller [8]. SDN also facilitates virtualization, allowing sharing of physical resources among multiple services, tasks, or applications. Consequently, SDN has emerged as an essential component of next-generation networks and the Internet of Things [9]. An integral component of our proposed solution, GABAS, seamlessly aligns with the principles of SDN. GABAS operates as a centralized algorithm that meticulously manages the network's intricate scheduling dynamics, akin to the fundamental ethos of SDN.

In WSNs that support multiple applications running on the same physical network, it is possible for different applications to require the same type of data from specific monitoring points in the monitored area. For instance, consider the scenario where one application monitors traffic density at a particular point while another application measures the average speed of vehicles between two points. Although these applications may have different data collection frequency requirements, they both rely on the same type of data. To address this challenge, we propose a shared-data approach with multiplexed sensing for running multiple applications on a WSN. Our approach aims to enable applications to efficiently share data from common monitoring points, thereby reducing the overall sensing and communication load on the WSN. By leveraging the shared-data approach, the network virtually offers more sensing and communication resources, resulting in improved efficiency and performance.

While the processing requirements of applications remain constant, the adoption of a shared-data approach introduces a significant enhancement in the efficient utilization of sensing and communication resources within the Wireless Sensor Network (WSN). This transformative shift allows the WSN to accommodate and schedule a greater number of applications concurrently, resulting in a notable reduction in the total execution time for a suite of applications. Moreover, the waiting time for newly arriving applications can be dramatically minimized, courtesy of the availability of surplus resources. A compelling real-world example of this is the application of WSNs in traffic management systems. These systems play a pivotal role in monitoring and analyzing traffic patterns, congestion levels, and incident detection, especially in densely populated urban environments grappling with heavy traffic. In such scenarios, the imperative for swift data collection and analysis cannot be overstated, as it enables timely interventions and improvements in traffic management strategies.

## III. Related Work

Scheduling problem exhibits itself in all types of computer systems and networks, where resources are limited and there are tasks, applications, or services that need to time-share those resources. The resources can be the processors of a computer, the sensing and communication units of a wireless sensor network, the physical servers and switches of a cloud data center, or the edge computing nodes of a fog network.

There are many scheduling algorithms proposed in the literature for processor and cloud scheduling [10]–[22]. They use different meta-heuristics and evolutionary algorithms, such as particle swarm optimization, genetic algorithms, and ant colony optimization. Some of them also use greedy approaches or classical approaches for scheduling, such as first come first served, and shortest job first algorithms. The metrics they usually use include makespan, average response time, energy efficiency, utilization, execution cost, and average running time. They schedule tasks, jobs, or virtual machines on local or cloud computing components, like physical servers.

There are also studies on scheduling in WSNs, IoT, fog, and edge computing. Fog and edge computing are usually integrated with WSNs in IoT systems. Porta et al. [23] propose EN-MASSE, a framework that deals with dynamic mission assignment for WSNs whose sensor nodes have energy harvesting capabilities. It is an integer programming method that assigns missions to sensor nodes and aims to minimize the total run-time of the missions. Uchiteleva et al. [24] describe a resource scheduling algorithm for WSNs. The proposed scheduling algorithm is a resource management solution for isolated profiles in WSNs, and the authors compare their algorithm with Round Robin and Proportionally Fair scheduling algorithms. Wei et al. [25] present a Q-learning algorithm called ISVM-Q for task scheduling in WSNs. It optimizes application performance and total energy consumption. De Frias et al. [26] propose an application scheduling algorithm for shared actuator and sensor networks. Their algorithm aims to reduce energy consumption in the network. Edalat and Motani [27] propose a method for task scheduling and task mapping in a WSN consisting of sensor nodes with energy harvesting capabilities. They consider task priority and energy harvesting to increase fairness.

Liu et al. [28] introduce Horae, a task scheduler tailored for mobile edge computing (MEC). Horae is designed with the dual objectives of optimizing resource utilization within MEC environments and meticulously selecting edge servers that align with the specific placement constraints of each task. Javanmardi et al. [29] present FUPE, a security-aware task scheduler for IoT fog networks. FUPE harnesses the power of a fuzzy-based multi-objective Particle Swarm Optimization algorithm. Their comprehensive study demonstrates that FUPE outperforms its peers in terms of both average response time and network utilization. Li and Han [30] offer ABC, an Artificial Bee Colony algorithm meticulously designed for task scheduling in cloud computing systems. Their work places ABC in a comparative spotlight against various existing approaches, with a focus on evaluation metrics such as makespan, maximum device workload, and total device workload. D'Amico and Gonzalez [31] introduce EAMC, a multi-cluster scheduling policy engineered to predict job energy consumption. EAMC aspires to reduce critical metrics including makespan, response time, and total energy consumption, ushering in efficiency and sustainability. Singhal and Sharma [32] describe the Rock Hyrax Optimization algorithm, specifically tailored to the intricate task of job scheduling in heterogeneous cloud systems. Their research scrutinizes performance using evaluation metrics like makespan and energy consumption for the optimization of cloud-based workloads.

Choudhari et al. [33] contribute a priority-based task scheduling algorithm for the fog computing systems. Their algorithm stands out by initially assigning incoming requests to the nearest fog server, followed by the placement of tasks into a priority queue within that server, optimizing task execution. Xu et al. [34] employ the power of online convex optimization techniques to tackle the task of scheduling jobs with multi-dimensional requirements in heterogeneous computing clusters. Their approach adds a layer of efficiency and adaptability to the scheduling process, ensuring optimal resource allocation. Psychasand and Ghaderi [35] present strategies grounded in Best Fit and Universal Partitioning techniques for job scheduling tasks. These algorithms offer effective solutions to the challenge of scheduling jobs with diverse resource demands, catering to the dynamic needs of modern computing environments. Fang et al. [36] target the vital objective of reducing total job completion time within the context of edge computing systems. They introduce an approximation algorithm capable of handling both offline and online scheduling scenarios, thereby contributing to the optimization of task execution in edge environments. Arri and Singh [37] present a distinctive approach with their artificial bee colony algorithm, augmented by an artificial neural network, designed specifically for job scheduling within fog servers. This combination of nature-inspired algorithms and neural network technology aims to enhance energy efficiency and overall scheduling performance.

In our research, as presented in this paper, we introduce a distinctive and innovative approach that sets it apart from prior studies in several compelling ways:

- Our pioneering contribution revolves around the introduction of GABAS, a groundbreaking genetic algorithm meticulously designed to schedule applications within Wireless Sensor Networks (WSNs) proficiently. GABAS not only handles application admission and scheduling but also excels in making intelligent decisions regarding the optimal utilization of sensor nodes and base stations for data acquisition from specified monitoring points, thus enhancing resource efficiency. Across a spectrum of critical metrics, including makespan, average turnaround time, average waiting time, and successful completion ratio, our genetic algorithm delivers remarkable performance.
- Complementing GABAS, we also put forth a trio of innovative greedy algorithms, each tailored to distinct criteria for prioritizing the admission of applications. These algorithms prove particularly valuable in scenarios where swift decision-making is imperative, offering versatility and efficiency.
- Our research delves into a sophisticated network architecture that showcases an innovative approach to resource optimization. Within this network structure, specific monitoring points are served by sensor nodes in a multiplexed fashion, enabling data to be concurrently shared and harnessed by multiple applications. This ingenious strategy facilitates the seamless coexistence of numerous applications, enhancing overall efficiency. This optimization paves the way for the concurrent scheduling of a higher number of applications. Our research primarily focuses on urban-area networks, benefitting from seamless interconnections of base stations via high-speed wired or wireless networks.

## IV. Problem Statement

The sensor network type we consider in this paper is a wireless sensor network (WSN) that is owned by a single provider and covers an urban region (like a city or town). Application owners require some points in the region to be sensed and sensed data to be collected and processed in cluster-heads. We assume the sensor network has a clustered two-tier architecture, consisting of sensor nodes and cluster-head nodes. We will call the cluster-head nodes also as base stations throughout the paper since they will act like base stations in a wireless network. They will be mains-powered and connected to the wired or wireless backbone network of a city. They will also be able to do local processing for the incoming sensor data. In an urban environment, cluster-head nodes can easily be mounted on top of the elements of the city-wide infrastructure (like lamp poles); therefore, they can easily be connected to the backbone network. In this way, the use of cluster-heads in our architecture is different from the classical WSN architectures, where cluster-heads are just intermediate nodes on multi-hop paths from sensor nodes to sink nodes. In an urban environment, we assume that each sensor node can directly connect to one of the base stations in its range. Each base station and the connected sensor nodes form a star topology.

We assume sensor nodes have equal sensing rates but various sensing ranges. Base stations have equal processing capacity. Additionally, we assume that any link between a sensor node and a base station has the same bandwidth capacity. A sensor node can collect data from the monitoring points which fall into its sensing range. The collected data is processed, partially or totally, in the base stations. It is possible that a base station can send the processed data further to a centralized location for additional processing or analysis. However, this part of the problem is not within the scope of this paper. We assume the bandwidth of the links connecting base stations to the rest of the network is abundant; hence is not a constraint in our formulation. We only consider the bandwidth capacity of connections between sensor nodes and base stations as a constraint. Similarly, we assume that the centralized servers have the abundant capacity to process the incoming data if needed. Therefore, we only consider the processing capacity of the base stations as a constraint in our model.

Figure 1 shows an example of our network model. In the figure, radio towers represent base stations (cluster-heads), large circles represent sensor nodes and small circles represent monitoring points. A dashed line between a sensor node and a monitoring point indicates that the sensor node actively senses the monitoring point. If a sensor node is connected to a base station, there is a dotted line between the sensor node and the base station. Base stations are connected to each other through a high-speed wired or wireless network.

Applications that arrive at the network require some points to be sensed and related data to be processed for a certain amount of time. One important goal is to finish all applications using the network as early as possible. If the requirements of an application are satisfiable and the application is scheduled to use the network, the application is deployed to the network and uses the required resources for the desired amount of time. After this period of time has expired, the application releases the resources and leaves the network.

The parameters used for a formal description of our problem are shown in Table I.

Our primary goal is to minimize the total run-time of applications requiring WSN service. This optimization problem can be formalized as follows:

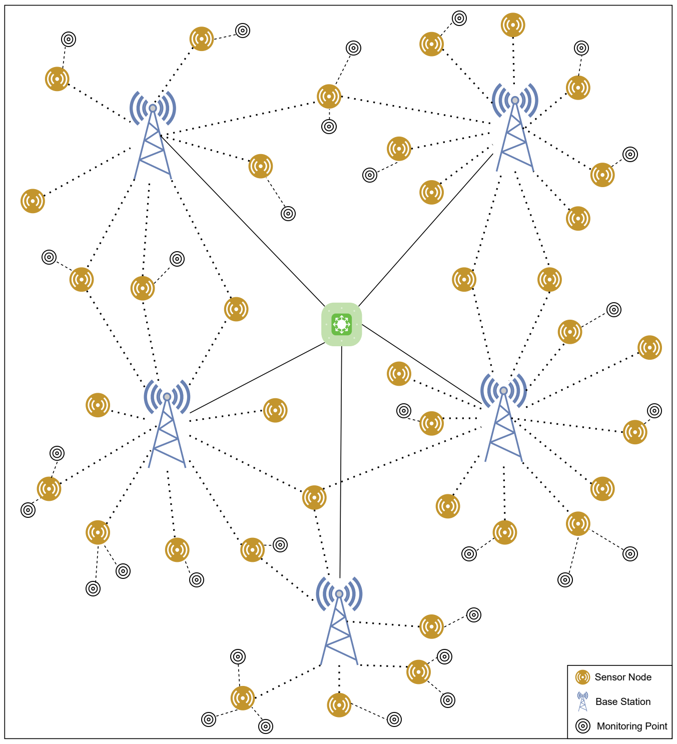


Fig. 1. Network model.

TABLE I
PARAMETERS USED IN THE PROBLEM STATEMENT.

| | |
|---|---|
| $S$ | Set of sensor nodes |
| $B$ | Set of base stations |
| $C$ | Set of connections |
| $A$ | Set of applications |
| $M$ | Set of monitoring points |
| $T$ | Set of time instants |
| $t_j$ | Time period that application $j$ wants to use the network |
| $t0_j$ | Time at which application $j$ is admitted |
| $tf_j$ | Time at which application $j$ finishes |
| $tf_{max}$ | Time the last application finishes |
| $x_{jt}$ | Binary variable indicating whether application $j$ is deployed at time $t$ |
| $M_j$ | Set of monitoring points to be sensed for application $j$ |
| $r_{jk}$ | Sensing rate requirement of application $j$ for monitoring point $k$ |
| $r_{kt}$ | Sensing rate requirement of monitoring point $k$ at time $t$ |
| $S_k$ | Set of sensor nodes covering monitoring point $k$ |
| $M_{il}$ | Set of monitoring points whose data is transferred from sensor node $i$ to base station $l$ |
| $x_{ik}$ | Binary variable indicating whether sensor node $i$ is actively sensing monitoring point $k$ |
| $x_{ilm}$ | Binary variable indicating whether sensor node $i$ is connected to base station $l$ through connection $m$ |
| $r_k$ | Sensing rate requirement of monitoring point $k$ |
| $R_i$ | Sensing rate capacity of sensor node $i$ |
| $P_l$ | Processing capacity of base station $l$ |
| $C_m$ | Bandwidth capacity of connection $m$ |
| $\alpha$ | Transmission coefficient |
| $\beta$ | Processing coefficient |

$$tf_{max} \tag{1}$$

is minimized subject to

$$tf_{max} = max(tf_j) \qquad \forall j \in A \tag{2}$$

$$tf_j = t0_j + t_j \qquad \forall j \in A \tag{3}$$

$$x_{jt} = \begin{cases} 1, & t0_j < t \le tf_j \\ 0, & otherwise \end{cases} \qquad \forall j \in A \tag{4}$$

$$r_{kt} = max(r_{jk} \times x_{jt}) \qquad \forall j \in A, \forall t \in T, \forall k \in M \tag{5}$$

$$u_{kt} = \sum_{j \in A}(r_{jk} \times x_{jt}) \qquad \forall k \in M, \forall t \in T \tag{6}$$

$$\sum_{k \in M} x_{ik} r_{kt} \le R_i \qquad \forall i \in S, \forall t \in T \tag{7}$$

$$\sum_{k \in M_{il}} \alpha r_{kt} \le \sum_{m \in C} x_{ilm} C_m \quad \forall i \in S, \forall l \in B, \forall t \in T \tag{8}$$

$$\sum_{i \in S} \sum_{k \in M_{il}} \beta u_{kt} \le P_l \qquad \forall l \in B, \forall t \in T \tag{9}$$

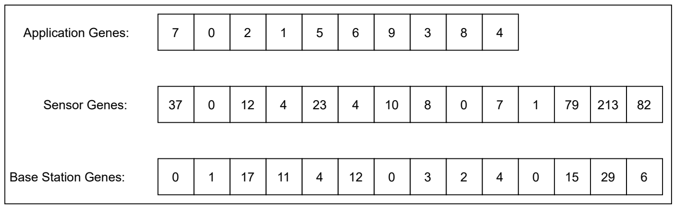
Eq. 2 shows how total run-time (makespan) is calculated. It is the finish time of the last application that used the network. We are assuming the first application is admitted at time 0. Eq.3 describes how the finish time of each application is determined. Eq. 4 is used to determine the time interval an application is deployed to the network. Eqs. 5 and 6 show the calculation of sensing requirements (i.e., required sensing rates) of monitoring points in shared and unshared cases, respectively. Eqs. 7, 8, and 9 are sensing, transmission and processing constraints for each time instant. At each time instant, the total sensing rate requirement of monitoring points sensed by one sensor node must not exceed that sensor node's sensing capacity. The data amount transmitted per second cannot be more than the bandwidth of that connection. Also, the total processed data per second at one base station should be less than the base station's processing capacity. $\alpha$ (transmission coefficient) is a constant that maps a sensing rate value to a communication rate requirement. Similarly, $\beta$ (processing coefficient) is a constant used to map a sensing rate value to a processing capacity requirement. For example, if data is sensed at a rate of 3 Kbps, the required communication rate may be 4 Kbps, including communication protocol overheads.

The application scheduling problem explained above is a variation of the well-known task scheduling problem. The task scheduling problem is proven to be an NP-hard problem [38]. To prove that application scheduling is also an NP-hard problem, we reduce the multiway number partitioning problem to it. Reduction is shown in Appendix A.

## V. Proposed Methods

### A. Genetic Algorithm Based Application Scheduling (GABAS)

To effectively schedule applications to a wireless sensor network, we propose a novel genetic algorithm called GABAS. A genetic algorithm is a meta-heuristic solution that mimics natural selection. It runs for *generations* until a termination condition is met. Each generation consists of *individuals*, which are candidate solutions for the problem. The actual solution that an individual provides is called its *chromosome*. The chromosome structure of an individual consists of one or more genes, each of which is a list or array, usually. In GABAS, the chromosome structure of individuals contains three lists of integers identifying applications, sensor nodes, and base stations. The first list is *Application Genes* which represents the scheduling order of applications. Its size is equal to the number of applications. The other two lists are *Sensor Genes* and *Base Station Genes*, which represent the proposed sensor nodes and base stations for the monitoring points, respectively. The size of the lists is equal to the number of monitoring points in the whole network area. Examples of these lists are shown in Figure 2. In the figure, according to the given sequence of *Application Genes*, the first application to be admitted is Application 7, which is followed by Application 0, and so on. Moreover, according to the *Sensor Genes* and *Base Station Genes*, Monitoring Point 0 is assigned to Sensor 37 and Base Station 0, Monitoring Point 1 is assigned to Sensor 0 and Base Station 1, and so on.



| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Application Genes: | 7 | 0 | 2 | 1 | 5 | 6 | 9 | 3 | 8 | 4 | | | | |
| Sensor Genes: | 37 | 0 | 12 | 4 | 23 | 4 | 10 | 8 | 0 | 7 | 1 | 79 | 213 | 82 |
| Base Station Genes: | 0 | 1 | 17 | 11 | 4 | 12 | 0 | 3 | 2 | 4 | 0 | 15 | 29 | 6 |

Fig. 2. Example genes.

*1) Initial Population Creation:* In genetic algorithms, individuals are created by crossover operation. Therefore, their *genes* are determined by two individuals from the previous generation. However, since there is no previous generation for the first generation, we need to create the individuals of the first generation randomly.

Since *Application Genes* of an individual determines the admission order of the applications, we assign value $i$ to the $i^{th}$ application gene and shuffle the list. Therefore, initially, the admission order is randomly determined.

For the individuals of the initial population, *Sensor Genes* and *Base Station Genes* are determined together. For each monitoring point, first, we randomly determine a sensor gene among the sensor nodes that cover the monitoring point. After that, we randomly select a base station gene among the base stations to which the selected sensor node has a connection.

*2) Fitness Calculation:* Fitness calculation in a genetic algorithm is designed and used to measure how close an individual is to the optimum solution. The result of fitness calculation is called *fitness value*. Equation 10 shows the fitness calculation of our algorithm.

$$fitness = -1 \times makespan \tag{10}$$

Basically, GABAS aims to reduce the makespan, which is the time instant when the last application finishes and leaves the network. Since a higher fitness value means a better individual (solution), we multiply the makespan with $-1$, since a lower makespan value is a more desirable one.

Applications are admitted one by one according to their order in Application Genes. If an application cannot be admitted due to a shortage of resources, we wait for some already admitted applications to finish and release the resources they use. Then there will be sufficient resources available for the waiting application.

*3) Selection Operation:* The selection operation is performed to choose an individual to pair up for each individual in the population of the current generation to create the next generation. We use tournament selection for this process. Basically, for each individual $i$, we create a sub-population consisting of 5% of the whole population and select the individual with the highest fitness score in this subpopulation. We also experimented with the population sizes, 3%, 7%, 10%, and 15% of the whole population. We observed that the population size of 5% yielded the best results among all the tested options. Based on this finding, we decided to select the 5% population size. We pair the individual $i$ with the selected individual. Our selection operation is presented in Algorithm 1.

**Algorithm 1** Selection Operation.

```
Require: The tournament population, $tPop$
Ensure: an individual chromosome
1: procedure SELECTION
2:     $bestScore \leftarrow 0$
3:     $bestInd \leftarrow Null$
4:     for each Individual $x$ in $tPop$ do
5:         $newScore \leftarrow fitness(x)$
6:         if $bestScore \leq newScore$ then
7:             $bestScore \leftarrow newScore$
8:             $bestInd \leftarrow x$
       return $bestInd$
```

*4) Crossover Operation:* The crossover operation is executed to create the population of the next generation. Individuals that are paired up with the selection operation are used in the crossover operation. The operation determines which genes of the offspring come from which parent (a pair of individuals). This operation is presented in Algorithm 2. We assume the chance of either parent to pass its genes to the offspring is 50%; therefore, we have *uniformRate* value in the algorithm as 0.5. We also investigated the impact of different selection rates, specifically focusing on rates of 0.6 and 0.7, where the fitter parent was favored. The choice of a uniform rate of 0.5 was determined based on the objective of increasing diversity within the population, thereby avoiding convergence to local optima. The Crossover operation of *Sensor Genes* and *Base Station Genes* is realized together to avoid producing a candidate solution that conflicts with the network structure.

Crossover of the *Application Genes* may result in some applications appearing twice in the Application Genes of the offspring. Therefore, we need a *gene repairing* algorithm to fix this problem. For that, we first determine the applications that appear twice and those that do not appear at all in the offspring's genes. Then, we put applications that are missing into the places of second appearances of the applications that are present twice in the genes.

**Algorithm 2** Crossover Operation.

```
Require: Six chromosomes from two parents: $A_1$, $S_1$, $BS_1$, $A_2$, $S_2$, and $BS_2$
Ensure: Three offspring chromosomes: $A_{new}$, $S_{new}$ and $BS_{new}$
1: procedure CROSSOVER
2:     for $x = 1$ $to$ $|\boldsymbol{A}|$ do
3:         randomly create a value between 0 and 1, $r$;
4:         if $r \leq uniformRate$ then
5:             set gene $x$ of $A_{new}$ as gene $x$ of $A_1$
6:         else
7:             set gene $x$ of $A_{new}$ as gene $x$ of $A_2$
8:     for $x = 1$ $to$ $|\boldsymbol{M}|$ do
9:         randomly create a value between 0 and 1, $r$;
10:        if $r \leq uniformRate$ then
11:            set gene $x$ of $S_{new}$ as gene $x$ of $S_1$
12:            set gene $x$ of $BS_{new}$ as gene $x$ of $BS_1$
13:        else
14:            set gene $x$ of $S_{new}$ as gene $x$ of $S_2$
15:            set gene $x$ of $BS_{new}$ as gene $x$ of $BS_2$
       return A new individual with chromosomes: $A_{new}$, $S_{new}$ and $BS_{new}$
```

*5) Mutation Operation:* The mutation operation is applied to all individuals. The operation is presented in Algorithm 3. As in the crossover operation, mutation operation in *Sensor Genes* and *Base Station Genes* is realized together to guarantee a solution that abides the network structure. This operation is executed for each gene with a chance of 5%.

For mutation in *Application Genes*, we swap the admission order of the two randomly selected applications with a 5% mutation rate. The selection of this specific mutation rate is based on recommendations and findings from numerous works in the literature, such as [39].

*6) The Genetic Algorithm:* Our overall genetic algorithm is presented in Algorithm 4. The termination condition of the algorithm is that no improvement is observed in the fitness score of the best individual for seven generations. The population size is 200. The fitness calculation for each individual is $-1$ times the completion time of the last finished application. All individuals have negative fitness scores since a better solution means a shorter finish time. Elitism is enabled in the algorithm, which means that the fittest individual of one generation is carried over to the next generation.

Next, we describe our greedy algorithms for WSN application scheduling.

**Algorithm 3** Mutation Operation.

```
Require: Three chromosomes: $A_{old}$, $S_{old}$ and $BS_{old}$
Ensure: Three mutated chromosomes: $A_{new}$, $S_{new}$ and $BS_{new}$
 1: procedure MUTATION
 2:     $A_{new} \leftarrow A_{old}$;
 3:     randomly create a value between 0 and 1, $r1$;
 4:     if $r1 \leq mutationRate$ then
 5:         randomly create a value between 0 and $|M|$, $x$;
 6:         randomly create a value between 0 and $|M|$, $y$;
 7:         swap gene $x$ and gene $y$ of $A_{new}$
 8:     for $x = 1$ to $|M|$ do
 9:         randomly create a value between 0 and 1, $r2$;
10:         if $r2 \leq mutationRate$ then
11:             randomly select a sensor node $i$ from possible sensor nodes, where $1 \leq i \leq |S|$
12:             randomly select a base station $l$ from possible base stations, where $1 \leq l \leq |B|$
13:             set gene $x$ of $S_{new}$ as gene $i$
14:             set gene $x$ of $BS_{new}$ as gene $l$
15:         else
16:             set gene $x$ of $S_{new}$ as gene $x$ of $S_{old}$
17:             set gene $x$ of $BS_{new}$ as gene $x$ of $BS_{old}$
    return Mutated individual with chromosomes: $A_{new}$, $S_{new}$ and $BS_{new}$
```

**Algorithm 4** The Genetic Algorithm.

```
 1: procedure THE GA
        generate a population of $POPSIZE$ number of random individuals, $POP$;
 2:     while THE TERMINATION CONDITION is not $true$ do
 3:         for each Individual $x$ in $POP$ do
 4:             calculate its fitness value $f(x)$
 5:         for each Individual $x$ in $POP$ do
 6:             Create a tournament population, $tPop$
 7:             $y$ = SelectionOperation($tPop$)
 8:         for each pair of parents, $x$ and $y$ do
 9:             $z$ = CrossoverOperation($x, y$)
10:         for each offspring do
11:             $z$ = MutationOperation($z$)
12:         find the best individual among offsprings, $newBest$
13:         if $newBest$ is better than the current best individual then
14:             replace current best individual with $newBest$
        return best individual
```

## B. Greedy Algorithms

For scenarios where fast decisions are needed, we also propose three simple greedy algorithms for scheduling applications onto a sensor network. All these greedy algorithms use the *Worst Fit* approach in assigning monitoring points to sensor nodes and base stations. The *Worst Fit* approach uses the less utilized resources among the available ones to provide load balancing. The difference between the three algorithms is the criteria they use for ordering the applications to admit to the network.

Our greedy algorithms consider only the waiting applications to determine their order. The applications that are already admitted to the network are allowed to run until they are complete. We do not preempt a running application.

Next, we give more information about each of our greedy algorithms.

*1) Least Monitoring Point First (LMPF):* In LMPF algorithm, we order the applications according to the number of monitoring points they require to be sensed. Among waiting applications, the ones with a smaller number of monitoring points to be sensed are admitted earlier.

*2) Least Maximum Sense Requirement First (LMSF):* In LMSF, the order of applications is determined according to their sensing rate requirements. Applications are admitted in an order in which they are sorted according to their maximum sensing rate requirement from a monitoring point (among all monitoring points requested to be sensed by an application). The order is a non-decreasing one; therefore, a waiting application with the least maximum requirement is placed first onto the network.

*3) Least Total Sense Requirement First (LTSF):* In LTSF, the application admission order is again determined according to the sensing rate requirements. However, for an application, instead of considering the maximum sensing rate requirement from a monitoring point, we consider the sum of sensing rate requirements for all its monitoring points.

## VI. Experimental Results

The network model in our simulations spans a 2D plane of size 1000 m $\times$ 1000 m. Monitoring points, sensor nodes, and base stations have their coordinates ($x$ and $y$), which are randomly determined and do not overlap. Therefore, at each coordinate, there is at most one network element.

We assume that applications arriving at the network may only require sensing rates on three major scales. Sense rate requirements of applications for each scale are randomly determined between the values shown in Table II. We ensure that only one data type can be requested from a single monitoring point. Our network constraints are shown in Table III. Each application can request 1, 2, or 3 monitoring points to be sensed. The communication range of a sensor node is randomly selected between 200 m and 250 m, and the range determines to which base stations the sensor node can get connected. Similarly, the sensing range for each sensor node is between 30 m and 50 m, and it is used to determine which monitoring points can be sensed by the sensor node. Additionally, the number of sensor nodes and base stations is 250 and 30, respectively. We observe that these values are large enough to cover the whole network area and small enough to let us understand the performance differences of the compared algorithms. The number of applications is 1000, and the number of monitoring points is 300. Applications arrive at the network in 25 batches, and the batch number for each application is randomly determined.

TABLE II
SENSING RATE REQUIREMENTS.

| Data Type | sensing rate |
|---|---|
| Data Type 0 | 5 - 20 |
| Data Type 1 | 15 - 40 |
| Data Type 2 | 25 - 60 |

TABLE III
NETWORK CONSTRAINTS.

| Constraint | Value |
|---|---|
| Monitoring Points per Application | 1 - 3 |
| Communication Range of Sensors | 200 - 250 |
| Sensing Range of Sensors | 30 - 50 |

Our simulation experiments are realized to investigate how certain parameters of the network affect the performance of algorithms that are compared. Unless otherwise stated below, the parameter values are set as explained above. For each scenario, we provide makespan, waiting time, turnaround time, and successful execution rate values averaged over 100 runs. We divide our experiments into six scenarios as follows:

- *Scenario 1 (Application Count)*: The number of applications starts at 500 and is incremented by 100 until 1500.
- *Scenario 2 (Monitoring Point Count)*: We start with 50 monitoring points in the area and increment the total number of monitoring points by 25 until 250.
- *Scenario 3 (Monitoring Point Count per Application)*: The number of monitoring points requested per application is initially 1. It is incremented by 1 until 7.
- *Scenario 4 (Communication Range)*: In the beginning, each sensor node has a 50 m communication range. We increase the communication range by 50 m until 250 m.
- *Scenario 5 (Sensing Range)*: Similar to Scenario 4, each sensor node starts with a 30 m sensing range, and the sensing range is incremented by 5 m until 50 m.
- *Scenario 6 (Batch Count)*: We experiment with the following values for the number of batches (batch count): 1, 2, 5, 10, 20 and 25. Applications are equally distributed into batches. The number of applications in each batch (batch size) is the number of applications divided by the batch count.

We use four different metrics to provide a comparative evaluation of our proposed algorithms.

- Average Makespan: Average of the total execution time of applications in 100 runs.
- Average Waiting Time: The waiting time of an application is the time between its arrival to the network and its admission. We report the average of the waiting times of all applications admitted.

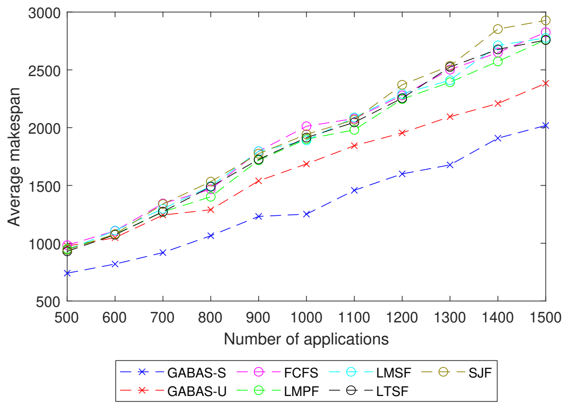


Fig. 3. Comparison of algorithms in terms of average makespan in Scenario 1.

- Average Turnaround Time: The turnaround time of an application is the time between its arrival time to the network and its finish time. We report the turnaround time averaged over all applications admitted.
- Average Successful Execution Rate: This metric is the number of applications that could finish before their deadlines. For each application, a deadline value is determined. The deadline value for an application is set to be the sum of its arrival time, its required running time (application duration), and a random value between 100 and 200.

We compare our proposed algorithms with well-known standard task scheduling algorithms First Come First Served (FCFS) and Shortest Job First (SJF). For FCFS and SJF algorithms, we also used the *Worst Fit* approach to determine which sensor node and the base station are assigned to each monitoring point. In the figures provided below, both shared (GABAS-S) and unshared (GABAS-U) approaches for GABAS are reported. For other algorithms, only shared approach results are provided to improve the readability since their performance with the shared approach is always better than the unshared approach.

In Scenario 1, we investigate how the number of applications affects the performance of the algorithms. Figure 3 shows the average makespan. GABAS-S has the best performance compared to others. GABAS-U is the worst for the small number of applications. However, with the increasing number of applications, its performance becomes better than the greedy ones. Our LMPF algorithm comes third, while LMSF and LTSF have similar performance as FCFS and SJF.

Figures 4, 5, and 6 present the average waiting time, average turnaround time, and average successful execution rate in the first scenario, respectively. FCFS and SJF have the worst performance for all three metrics. GABAS-S is superior to all other algorithms. LTSF comes second, and LMSF comes third with a close performance to LTSF. In terms of average waiting and turnaround times, GABAS-U and LMPF have similar performance, but in terms of average successful execution rate, GABAS-U beats LMPF.

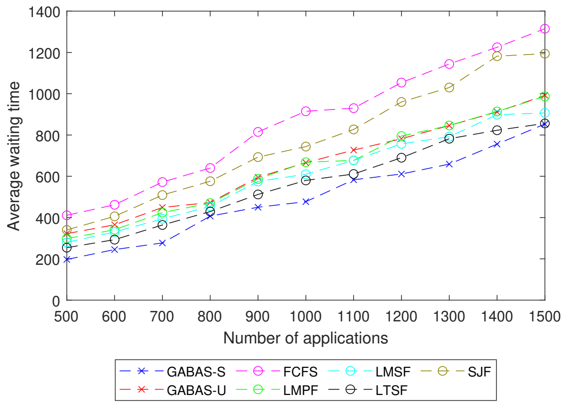


Fig. 4. Comparison of algorithms in terms of average waiting time in Scenario 1.

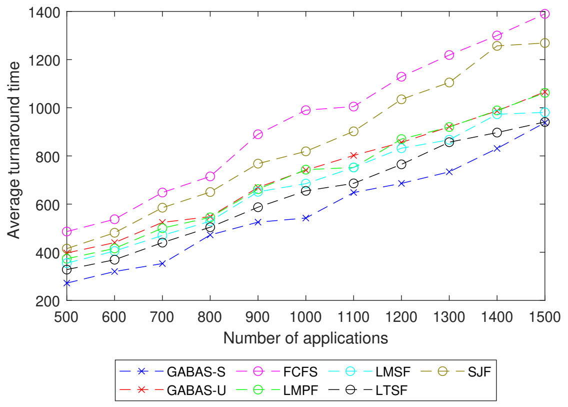


Fig. 5. Comparison of algorithms in terms of average turnaround time in Scenario 1.

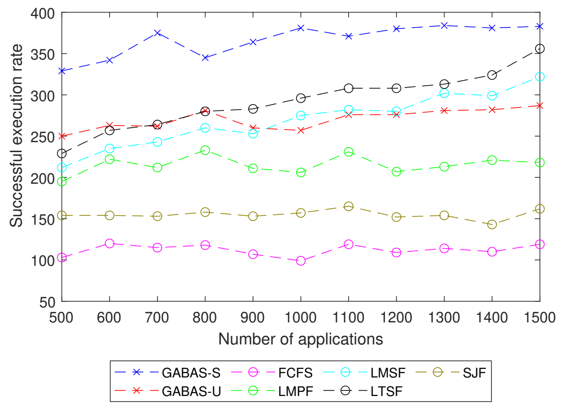


Fig. 6. Comparison of algorithms in terms of successful execution rate in Scenario 1.

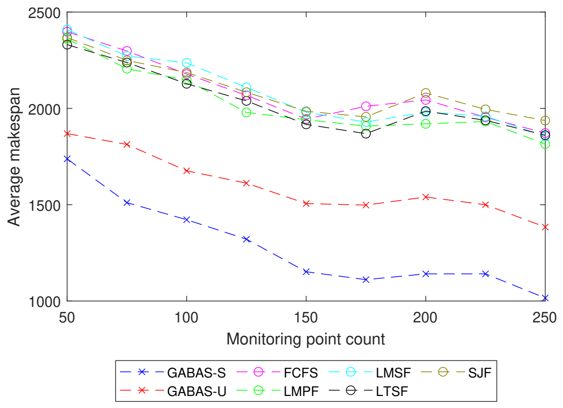


Fig. 7. Comparison of algorithms in terms of average makespan in Scenario 2.

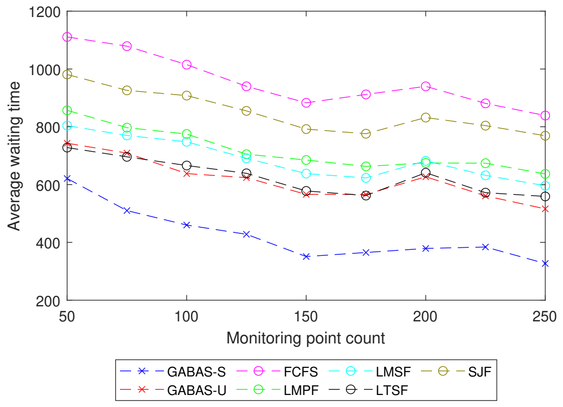


Fig. 8. Comparison of algorithms in terms of average waiting time in Scenario 2.

In Scenario 2, the simulations are done to observe the effect of the number of monitoring points on the performance of the algorithms. Figure 7 presents the average makespan for Scenario 2. GABAS-S is clearly superior to other algorithms, while GABAS-U comes second. LMPF again has the best performance among greedy algorithms. SJF generally has the worst performance of all. The performance gap between GABAS and others diminishes as the number of monitoring points in the region increases because it becomes easier to admit applications.

Average waiting time, average turnaround time, and average successful execution rate results of Scenario 2 are shown in Figures 8, 9, and 10, respectively. As in the previous scenario, FCFS and SJF have the worst performance among all algorithms, while GABAS-S has the best. The results for GABAS-U, LMSF, and LTSF are very close, but LMPF performs slightly worse among these algorithms.

In Scenario 3, we aim to see the impact of the number of monitoring point requests per application on the algorithms' performance. Figure 11 presents the results of this scenario in terms of makespan. GABAS-S still has the best results.

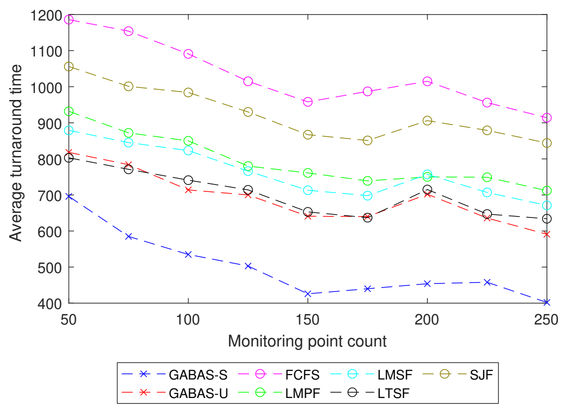


Fig. 9. Comparison of algorithms in terms of average turnaround time in Scenario 2.

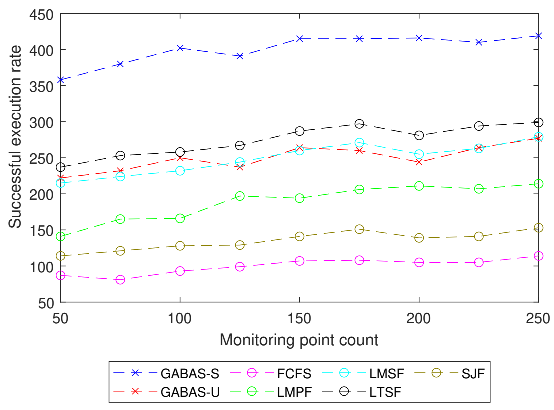


Fig. 10. Comparison of algorithms in terms of successful execution rate in Scenario 2.

After five requests per application GABAS-U has the next best performance. Greedy approaches have very similar results. SJF is the slightly worst of all. In this scenario, LMPF behaves like FCFS since all applications have an equal number of requests. GABAS-S performs better with a higher number of requests per application compared to the greedy algorithms.

Figures 12, 13 and 14 display the results of average waiting time, turnaround time and successful execution rate, respectively. GABAS-S is the superior one among all. LMPF and FCFS have the worst performance for these criteria, while SJF is slightly better than them. GABAS-U has the second-best performance, while LMSF and LTSF produce the best results of all greedy algorithms.

In Scenario 4, we evaluate the effect of change in the communication range of the sensor nodes on the results. Figure 15 presents the algorithms' makespan results. GABAS-S has the lowest makespan value, which makes it the best method among all compared algorithms. In terms of makespan, GABAS-U has the second-best performance. Greedy algorithms have similar results between 150-m and 250-m communication ranges. Between the 50-m and 150-m ranges, the performance of LMPF

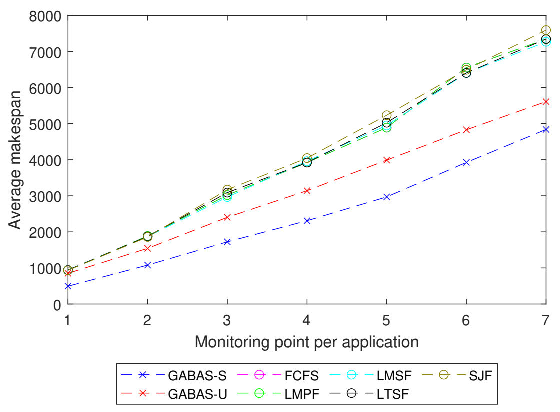


Fig. 11. Comparison of algorithms in terms of average makespan in Scenario 3.

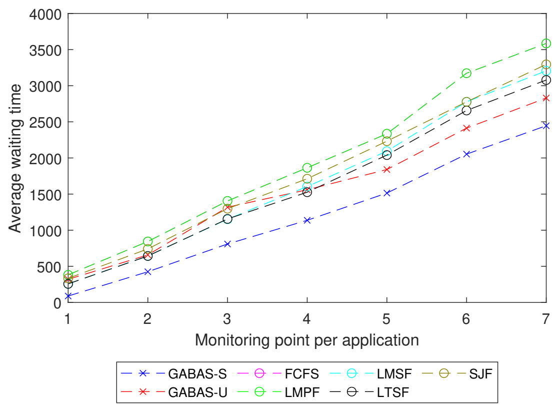


Fig. 12. Comparison of algorithms in terms of average waiting time in Scenario 3.

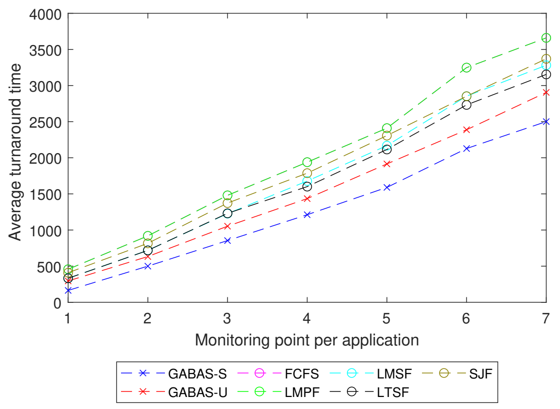


Fig. 13. Comparison of algorithms in terms of average turnaround time in Scenario 3.

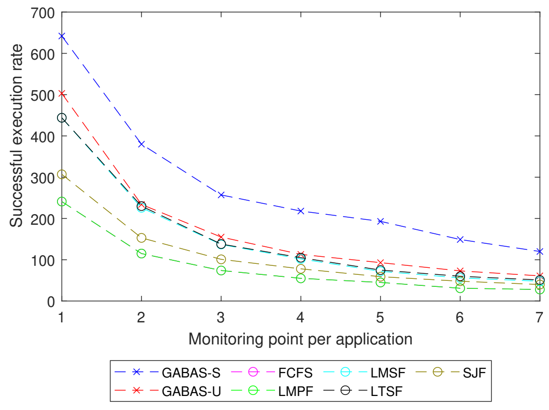


Fig. 14. Comparison of algorithms in terms of successful execution rate in Scenario 3.

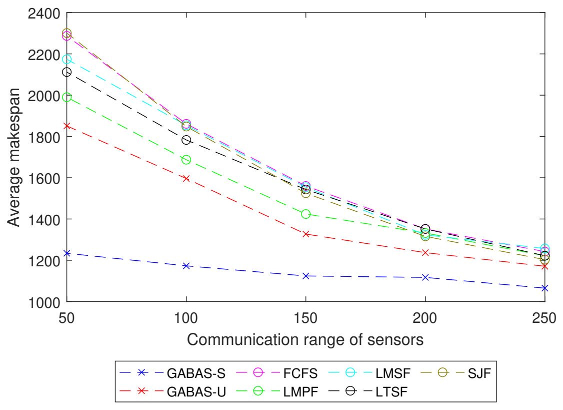


Fig. 15. Comparison of algorithms in terms of average makespan in Scenario 4.

is distinguishable from the other four algorithms. SJF has the worst performance, especially with larger communication ranges.

Average waiting and turnaround time for applications, and average successful execution rate are displayed in Figures 16, 17 and 18, respectively. GABAS-S again has the superior performance. FCFS has the worst turnaround and waiting times as well as the least successful execution rate. GABAS-U, LTSF, and LMSF perform close to each other.

In Scenario 5, we investigate the impact of the sensing range of sensor nodes on the results. Makespan results for this scenario are shown in Figure 19. We can see that the sensing range does not affect the results drastically. With GABAS-S, the total execution time of applications is much smaller compared to other algorithms. GABAS-U produces the next best results, while the performance results of the greedy algorithms are close to each other. Again, LMPF has the best performance among all greedy algorithms in terms of makespan.

Average waiting time, turnaround time and successful execution rate results for this scenario are presented in Figures

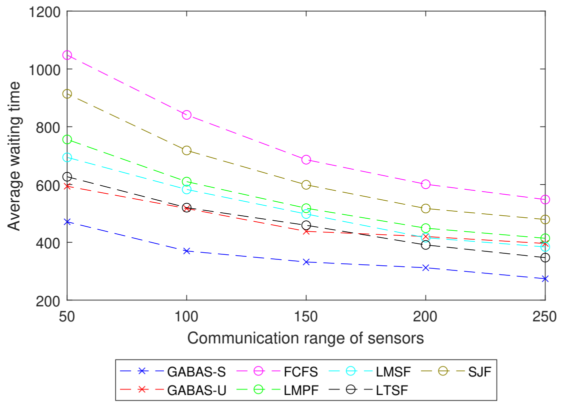


Fig. 16. Comparison of algorithms in terms of average waiting time in Scenario 4.

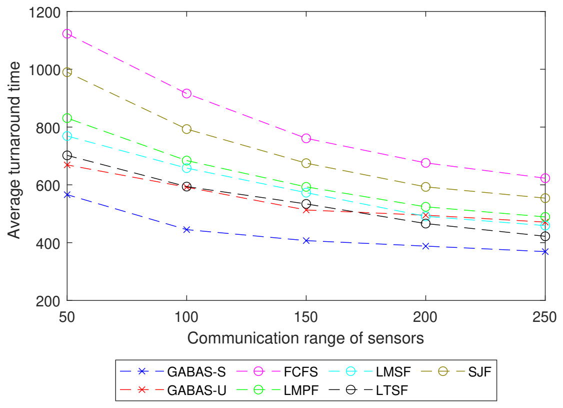


Fig. 17. Comparison of algorithms in terms of average turnaround time in Scenario 4.

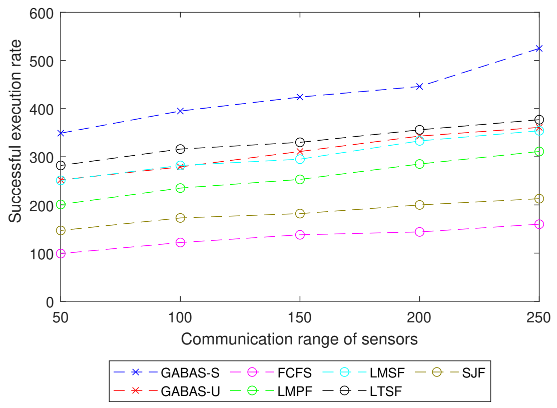


Fig. 18. Comparison of algorithms in terms of successful execution rate in Scenario 4.

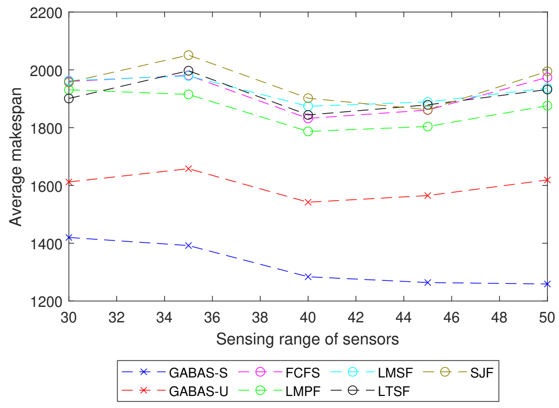


Fig. 19. Comparison of algorithms in terms of average makespan in Scenario 5.

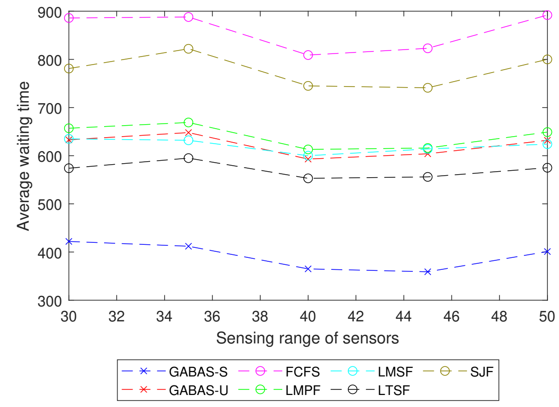


Fig. 20. Comparison of algorithms in terms of average waiting time in Scenario 5.

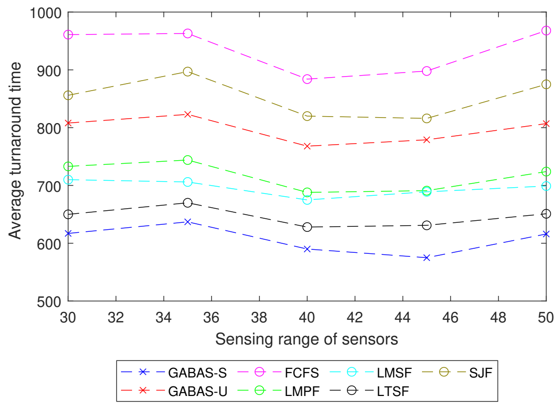


Fig. 21. Comparison of algorithms in terms of average turnaround time in Scenario 5.

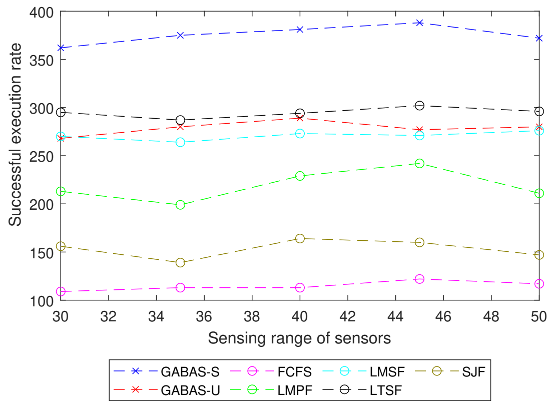


Fig. 22. Comparison of algorithms in terms of successful execution rate in Scenario 5.

20, 21 and 22, respectively. Similarly, GABAS-S has the best performance, and FCFS has the worst. SJF is slightly better than FCFS but still behind the proposed greedy methods. LTSF comes second. GABAS-U, LMPF, and LMSF have close results in terms of waiting time; however, GABAS-U and LMPF perform slightly worse in terms of turnaround time and successful execution rate, respectively.

In Scenario 6, we observe how the number of batches affects the performance of the algorithms. Figure 23 shows the average makespan respectively in this scenario. In general, different batch counts do not affect much the performance of the algorithms. GABAS-S, again, has the best performance, which is followed by GABAS-U. Greedy methods have very similar performance, whereas LMPF is slightly better than the others.

Regarding average waiting time, turnaround time, and successful execution rate, batch count again does not affect the results. Similar to the other scenarios, GABAS-S has the best performance. GABAS-U is slightly better than greedy methods. Among the greedy methods, LTSF performs the best which is followed by LMSF and LTSF. SJF and FCFS have the worst results. The waiting time, turnaround time, and successful execution rate results of this scenario are shown in Figures 24, 25, and 26, respectively.

In order to assess the reliability and precision of the obtained experimental results, we calculated confidence intervals for each metric in each scenario. These confidence intervals provide a range of values within which we can be reasonably confident that the true population parameter lies. We found that the largest confidence interval, with 95% confidence level, across all metrics and scenarios, was less than 5% of the reported result for each respective metric. This implies that the uncertainty associated with the estimated performance results is relatively small, suggesting a high level of confidence in the reported outcomes. To provide further clarity and transparency, we have included the confidence intervals of the results for the average makespan specifically for Scenario 1 in Table IV.

We also compared our algorithm GABAS with another genetic scheduling algorithm called HGECS that has recently appeared in the literature [7]. We used Scenario 3 for this comparison, as the number of monitoring point requests per application is the parameter that has the most impact on

TABLE IV
CONFIDENCE INTERVALS OF MAKESPAN RESULTS IN SCENARIO 1.

| | GABAS-S | GABAS-U | LMPF | LMSF | LTSF | SJF | FCFS |
|---|---|---|---|---|---|---|---|
| App: 500 | 726.12 ± 18.38 | 954.71 ± 18.96 | 916.34 ± 13.56 | 892.81 ± 13.48 | 876.70 ± 13.54 | 894.85 ± 15.76 | 984.30 ± 14.51 |
| App: 600 | 776.72 ± 19.23 | 1093.46 ± 18.15 | 1137.91 ± 10.14 | 1204.38 ± 14.75 | 1135.52 ± 12.02 | 1149.13 ± 16.36 | 1198.35 ± 13.61 |
| App: 700 | 831.79 ± 24.71 | 1269.98 ± 27.75 | 1383.53 ± 11.80 | 1321.85 ± 9.28 | 1286.72 ± 10.11 | 1379.63 ± 12.57 | 1391.36 ± 14.45 |
| App: 800 | 1057.56 ± 26.25 | 1284.40 ± 27.58 | 1450.29 ± 15.06 | 1484.72 ± 14.56 | 1481.26 ± 15.99 | 1525.17 ± 16.31 | 1467.62 ± 15.06 |
| App: 900 | 1249.70 ± 37.89 | 1530.52 ± 35.67 | 1707.92 ± 21.00 | 1787.09 ± 12.34 | 1715.33 ± 15.09 | 1765.30 ± 17.60 | 1787.24 ± 20.30 |
| App: 1000 | 1254.53 ± 31.14 | 1677.42 ± 38.07 | 1887.55 ± 20.56 | 1880.91 ± 12.82 | 1901.28 ± 25.54 | 1930.83 ± 28.49 | 2003.51 ± 17.04 |
| App: 1100 | 1430.48 ± 26.83 | 1834.62 ± 42.02 | 1972.91 ± 24.53 | 2070.88 ± 24.53 | 2027.03 ± 13.87 | 2052.95 ± 22.60 | 2063.37 ± 18.12 |
| App: 1200 | 1624.97 ± 15.94 | 1946.49 ± 12.30 | 2230.94 ± 15.78 | 2279.41 ± 21.45 | 2239.92 ± 13.17 | 2358.06 ± 16.23 | 2267.20 ± 16.65 |
| App: 1300 | 1667.90 ± 13.08 | 2085.96 ± 37.78 | 2379.61 ± 17.99 | 2393.49 ± 14.41 | 2507.24 ± 17.19 | 2523.72 ± 29.83 | 2489.13 ± 12.69 |
| App: 1400 | 1908.12 ± 19.16 | 2193.51 ± 29.29 | 2553.48 ± 15.09 | 2683.60 ± 14.82 | 2647.14 ± 16.43 | 2637.51 ± 20.72 | 2610.17 ± 17.03 |
| App: 1500 | 2011.04 ± 26.01 | 2364.72 ± 39.31 | 2739.50 ± 31.10 | 2761.27 ± 34.97 | 2728.42 ± 33.11 | 2810.03 ± 39.34 | 2906.92 ± 33.72 |

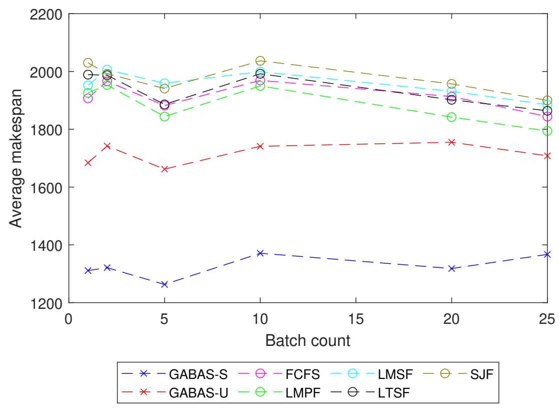


Fig. 23. Comparison of algorithms in terms of average makespan in Scenario 6.

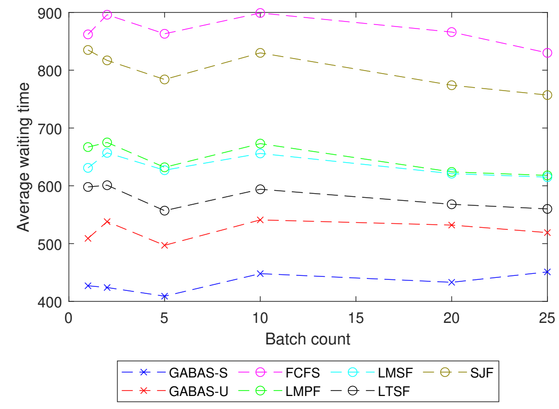


Fig. 24. Comparison of algorithms in terms of average waiting time in Scenario 6.

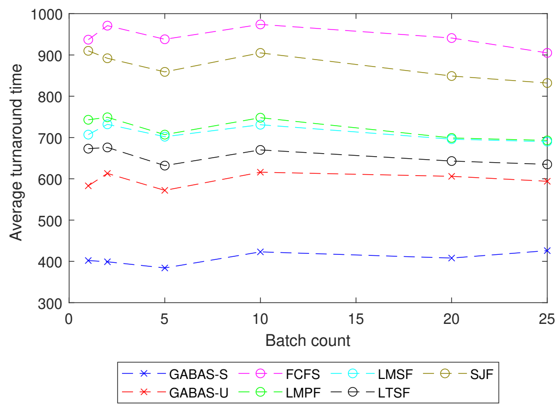


Fig. 25. Comparison of algorithms in terms of average turnaround time in Scenario 6.

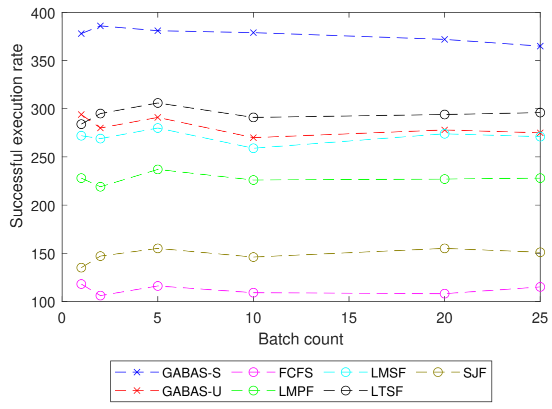


Fig. 26. Comparison of algorithms in terms of successful execution rate in Scenario 6.

the performance of the algorithms among all our scenarios. The comparison metrics used are makespan, average waiting time (AWT), average turnaround time (ATT), and successful execution rate (SER). Only the shared-data approach was employed in these experiments, as this approach was shown to perform better in general than the unshared-data approach. The performance results obtained are shown in Figures 27, 28, 29, and 30. The results presented are the average of 100 independent runs of the experiments. A clear trend emerges after examining the obtained results, indicating that the GABAS algorithm outperforms the HGECS algorithm across all four comparison metrics. This finding suggests that GABAS exhibits superior performance regarding these specific evaluation criteria.

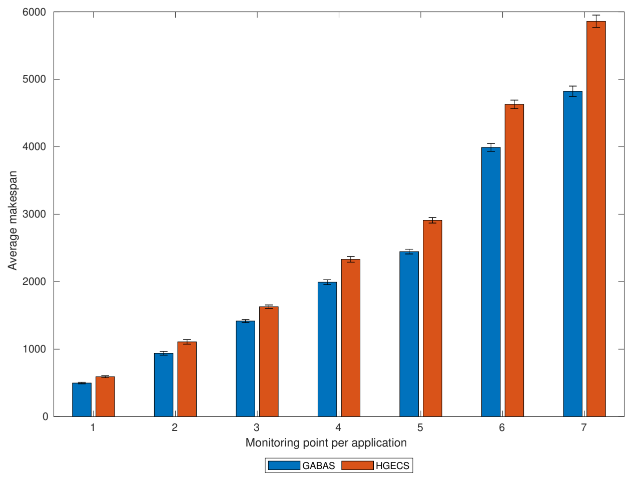


Fig. 27. Comparison of GABAS and HGECS in terms of average makespan in Scenario 3.

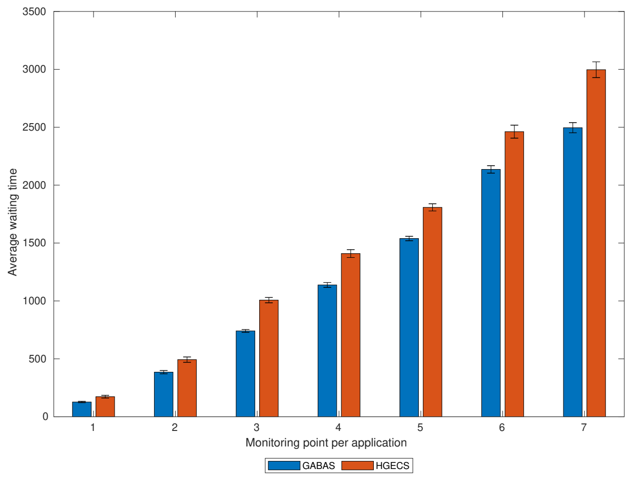


Fig. 28. Comparison of GABAS and HGECS in terms of average waiting time in Scenario 3.

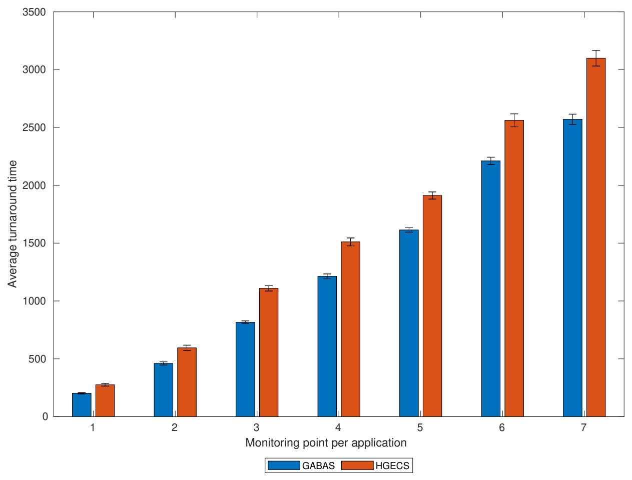


Fig. 29. Comparison of GABAS and HGECS in terms of average turnaround time in Scenario 3.

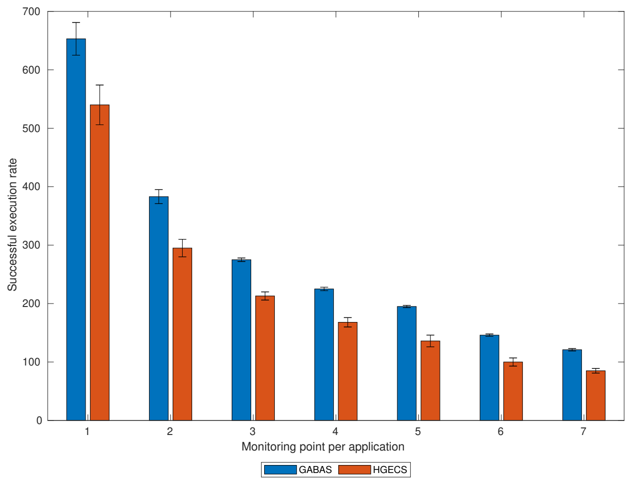


Fig. 30. Comparison of GABAS and HGECS in terms of successful execution rate in Scenario 3.

Confidence intervals at a 95% confidence level were obtained for the performance results of the GABAS and HGECS algorithms. To visually represent these confidence intervals, we incorporated error bars in the corresponding figures. By incorporating confidence intervals and error bars, we aim to provide a more comprehensive understanding of the performance results obtained from the GABAS and HGECS algorithms. This allows for a more accurate assessment of their effectiveness and provides insights into the level of confidence we can have in these results.

We also measured the running times of the algorithms. A selected set of results are provided in Table V. Our GABAS-S and GABAS-U algorithms are the slowest due to the nature of the genetic algorithms. GABAS-S is faster than GABAS-U since, with the shared-data approach, it is easier to place more applications at the same time; therefore, the total computation time is less compared to the unshared-data approach. Our greedy algorithms are much faster, as expected, compared to GABAS. Hence they are useful when fast decisions are required. FCFS is the fastest among all algorithms because it does not reorder the applications in the arrival queue.

In summary, from all our experimental results, we can draw the following conclusions:

- GABAS-S is clearly superior to all the algorithms used for comparison, including the standard scheduling methods (FCGS, SJF), greedy algorithms (LMPF, LMSF, LTSF) and a genetic scheduling algorithm (HGECS). It outperforms GABAS-U as well, and therefore we can conclude that the shared-data approach with multiplexed sensing is very effective in increasing the performance of the scheduling algorithms for various metrics.
- GABAS-U generally performs better than the greedy methods, even if it uses the unshared-data approach, while all the greedy methods use the shared-data approach. Even in the experiments measuring waiting time, turnaround time, and successful execution rate, GABAS-U has a better performance compared to greedy algorithms, especially when network resources are scarcer, even though it does not target these metrics directly. This result shows that using meta-heuristic algorithms is very effective in the application scheduling problem for WSNs.

TABLE V
RUNNING TIMES OF THE ALGORITHMS IN MILLISECONDS.

| Scenario | GABAS-S | GABAS-U | LMPF | LMSF | LTSF | FCFS | SJF |
|---|---|---|---|---|---|---|---|
| Scenario1 #A: 500 | 204 | 286 | 38 | 24 | 25 | 3 | 46 |
| Scenario1 #A: 1000 | 352 | 524 | 105 | 63 | 70 | 2 | 90 |
| Scenario1 #A: 1500 | 2588 | 3094 | 378 | 253 | 282 | 6 | 378 |
| Scenario2 #MP: 50 | 377 | 899 | 93 | 67 | 71 | 3 | 103 |
| Scenario2 #MP: 100 | 326 | 574 | 102 | 69 | 75 | 2 | 98 |
| Scenario2 #MP: 250 | 229 | 289 | 73 | 51 | 64 | 1 | 92 |
| Scenario3 #MP/A: 1 | 335 | 562 | 56 | 32 | 31 | 3 | 67 |
| Scenario3 #MP/A: 3 | 1051 | 1508 | 362 | 256 | 245 | 3 | 413 |
| Scenario3 #MP/A: 5 | 2254 | 5023 | 886 | 740 | 752 | 7 | 974 |

TABLE VI
SUMMARY OF RESULTS.

| | |
|---|---|
| GABAS-S | Performs better than other algorithms in terms of all four metrics |
| GABAS-U | Performs better than other algorithms (except GABAS-S) in terms of all four metrics |
| LMPF | Performs better than LMSF, LTSF in terms of makespan |
| LMSF, LTSF | Performs better than LMPF in terms of waiting time, turnaround time, successful execution rate |
| FCFS, SJF | Performs worse than other algorithms in terms of all four metrics |

- In terms of makespan, LMPF is the best greedy algorithm among the compared algorithms.
- In terms of waiting time, turnaround time, and successful execution rate, LMSF and LTSF have the best performance among greedy and standard algorithms.
- All proposed algorithms perform better than the standard FCFS and SJF algorithms.

A summary of experimental results is provided in Table VI.

## VII. CONCLUSION

In this paper, we studied the application scheduling problem in wireless sensor networks. First, we proposed a shared-data approach that provides an opportunity to perform multiplexed sensing of monitoring points for multiple applications that can share data, and in this way to reduce sensing and communication resource usage. Then, we proposed a genetic algorithm called GABAS, for scheduling applications effectively. We also proposed three greedy algorithms, LMPF, LMSF, and LTSF, that can be used for scenarios where fast decisions are needed. All our proposed algorithms decide both on the assignments of sensor nodes and base stations to monitoring points and the admission order of the waiting applications. We compared our proposed algorithms with each other and the well-known task scheduling algorithms, First Come First Served and Shortest Job First, in terms of makespan, waiting time, turnaround time, and successful execution rate by performing extensive simulation experiments. We observed that GABAS outperforms all other algorithms in all comparison metrics. Among other algorithms, LMPF provides the best results in terms of makespan, and LMSF and LTSF provide the best performance in terms of waiting time, turnaround time, and successful execution rate. In addition to the evaluation of the proposed GABAS algorithm, we conducted a comparative analysis of it against another genetic algorithm-based scheduling method called HGECS [7]. Upon comparing the performance of GABAS and HGECS using the designated metrics, the results consistently demonstrated that GABAS outperforms HGECS in all four metrics.

## APPENDIX A
## REDUCTION OF APPLICATION SCHEDULING TO MULTIWAY NUMBER PARTITIONING

### A. Multiway Number Partitioning

Multiway number partitioning (MNP) is the problem of partitioning a multi-set of numbers into *k* different subsets in a way that sums of numbers in each subset are as similar as possible. It is a generalized version of the partitioning problem where $k = 2$. Partitioning is proven to be NP-hard [38].

### B. Reduction to Application Scheduling

We assume that there are *n* applications waiting to be deployed. Each application requires a single monitoring point to be sensed with a unit sensing rate. Applications need to be deployed to the network for a certain amount of time which is denoted by $t_j$ for application *j*. There are *k* sensor nodes and each sensor node is connected to a single base station. Each sensor node and the base station have unit sensing and processing capacity, respectively. Any sensor node to base station connection has unit bandwidth. Both transmission and processing coefficients are equal to 1. Each monitoring point is required to be sensed by a single application.

In MNP, we have a set $S$ of $n$ numbers $a_1, a_2, .., a_n$. The set is to be partitioned into *k* subsets such that the maximum subset-sum is minimized. Transformation is done as follows:

Each number in the MNP set is the running time requirement of an application. In other words, $a_j = t_j$, where $t_j$ is the running time requirement of application $j$. Each partition corresponds to a sensor node and base station pair that will take part in sensing and processing the data of the applications assigned to them. If $a_j$ is in partition *i*, then the sensing and processing requirement of the application *j* for a monitoring point is handled by the sensor node and base station pair *i*. The sum of numbers assigned to a partition represents the amount of time during which the corresponding sensor node and base station pair will be active, i.e., sensing and processing for the applications assigned to them. The maximum sum among the sums for all partitions is equal to the maximum active time of a sensor node and base station pair, which is equal to the finish time of the last application. Therefore, minimizing the maximum sum is equal to minimizing $tf_{max}$ in application scheduling.

## Appendix B
## Complexity Analysis of GABAS

In this subsection, we provide a complexity analysis for the GABAS algorithm. In this analysis, we denote the number of applications as $a$ and the number of monitoring points as $m$. In creating the initial population, for each individual, the creation of application genes involves creating and shuffling the list, both of which take $O(a)$ time. Creating the individual's sensor and base station genes takes $0(m)$ time each. Therefore, the complexity of the total time for initial population creation is $O(n*(2m+a))$, where $n$ is the size of the population.

In fitness calculation, we simulate the placement of the applications to the network. Theoretically, each application can have up to $m$ monitoring point requests. Therefore, the time complexity for fitness calculation is $O(ma)$.

For the selection operation of each individual, if we assume that the tournament population has the size of $t$, creating the tournament population and selecting the best one in the population takes $O(t)$ time.

A single crossover operation of GABAS is basically the creation of three lists: One with the size $a$ and two with the size $m$. The time complexity of creating these genes is $O(2m+a)$. In the crossover of application genes, at most half of the genes may repeat and require gene repairing, which takes $O(a)$ time. As a result, a crossover operation takes $O(2m+a)+O(a) = O(m+a)$ time.

A single mutation operation includes at most one swap operation in application genes and up to $m$ reassignment of monitoring points to sensor nodes and base stations. Swapping takes $O(1)$ time. Selecting a random sensor node and base station pair for each monitoring point also takes $O(1)$ time. In total, a mutation operation has $O(1)+O(m) = O(m)$ time complexity.

As a result of all these analyses, for each individual, we have the time complexity of $O(ma)+O(t)+O(m+a)+O(m) = O(ma)$ since the dominant factor is $O(ma)$. For the whole algorithm, if the number of generations is $g$, then the time complexity is $O(gnma)$.

## References

[1] S. Lee, D. Yoon, and A. Ghosh, "Intelligent parking lot application using wireless sensor networks," in *2008 International Symposium on Collaborative Technologies and Systems*. IEEE, 2008, pp. 48–57.

[2] N. Maisonneuve, M. Stevens, M. E. Niessen, P. Hanappe, and L. Steels, "Citizen noise pollution monitoring," in *Proceedings of the 10th International Digital Government Research Conference*. Association for Computing Machinery, 2009, pp. 96–103.

[3] S. Taruna, K. Jain, and G. Purohit, "Application domain of wireless sensor network:-a paradigm in developed and developing countries," *International Journal of Computer Science Issues (IJCSI)*, vol. 8, no. 4, p. 611, 2011.

[4] H. Nigam, A. Karmakar, and A. K. Saini, "Wireless sensor network based structural health monitoring for multistory building," in *2020 4th International Conference on Computer, Communication and Signal Processing (ICCCSP)*. IEEE, 2020, pp. 1–5.

[5] I. D. Wahyono, K. Asfani, M. M. Mohamad, H. Rosyid, A. Afandi *et al.*, "The new intelligent wireless sensor network using artificial intelligence for building fire disasters," in *2020 Third International Conference on Vocational Education and Electrical Engineering (ICVEE)*. IEEE, 2020, pp. 1–6.

[6] M. C. Çavdar, I. Korpeoglu, and Özgür Ulusoy, "Application placement with shared monitoring points in multi-purpose iot wireless sensor networks," *Computer Networks*, vol. 217, p. 109302, 2022.

[7] G. Agarwal, S. Gupta, R. Ahuja, and A. K. Rai, "Multiprocessor task scheduling using multi-objective hybrid genetic algorithm in fog–cloud computing," *Knowledge-Based Systems*, vol. 272, p. 110563, 2023.

[8] F.-Y. Wang, L. Yang, X. Cheng, S. Han, and J. Yang, "Network softwarization and parallel networks: beyond software-defined networks," *IEEE Network*, vol. 30, no. 4, pp. 60–65, 2016.

[9] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for sdn? implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[10] L. Abualigah and A. Diabat, "A novel hybrid antlion optimization algorithm for multi-objective task scheduling problems in cloud computing environments," *Cluster Computing*, vol. 24, no. 1, pp. 205–223, 2021.

[11] S. E. Shukri, R. Al-Sayyed, A. Hudaib, and S. Mirjalili, "Enhanced multi-verse optimizer for task scheduling in cloud computing environments," *Expert Systems with Applications*, vol. 168, p. 114230, 2021.

[12] S. Mirjalili, S. M. Mirjalili, and A. Hatamlou, "Multi-verse optimizer: a nature-inspired algorithm for global optimization," *Neural Computing and Applications*, vol. 27, no. 2, pp. 495–513, 2016.

[13] S. Velliangiri, P. Karthikeyan, V. A. Xavier, and D. Baswaraj, "Hybrid electro search with genetic algorithm for task scheduling in cloud computing," *Ain Shams Engineering Journal*, vol. 12, no. 1, pp. 631–639, 2021.

[14] K. X. Kang, D. Ding, H. M. Xie, Q. Yin, and J. Zeng, "Adaptive drl-based task scheduling for energy-efficient cloud computing," *IEEE Transactions on Network and Service Management*, 2021.

[15] M. Sulaiman, Z. Halim, M. Lebbah, M. Waqas, and S. Tu, "An evolutionary computing-based efficient hybrid task scheduling approach for heterogeneous computing environment," *Journal of Grid Computing*, vol. 19, no. 1, pp. 1–31, 2021.

[16] D. Alboaneen, H. Tianfield, Y. Zhang, and B. Pranggono, "A metaheuristic method for joint task scheduling and virtual machine placement in cloud data centers," *Future Generation Computer Systems*, vol. 115, pp. 201–212, 2021.

[17] J. Yang, B. Jiang, Z. Lv, and K.-K. R. Choo, "A task scheduling algorithm considering game theory designed for energy management in cloud computing," *Future Generation Computer Systems*, vol. 105, pp. 985–992, 2020.

[18] X. Chen, L. Cheng, C. Liu, Q. Liu, J. Liu, Y. Mao, and J. Murphy, "A woa-based optimization approach for task scheduling in cloud computing systems," *IEEE Systems Journal*, vol. 14, no. 3, pp. 3117–3128, 2020.

[19] A. Chhabra, G. Singh, and K. S. Kahlon, "Performance-aware energy-efficient parallel job scheduling in hpc grid using nature-inspired hybrid meta-heuristics," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 2, pp. 1801–1835, 2021.

[20] S. Padhy and J. Chou, "Mirage: A consolidation aware migration avoidance genetic job scheduling algorithm for virtualized data centers," *Journal of Parallel and Distributed Computing*, vol. 154, pp. 106–118, 2021.

[21] H. Sun, H. Yu, and G. Fan, "Contract-based resource sharing for time effective task scheduling in fog-cloud environment," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 1040–1053, 2020.

[22] C. Li, J. Tang, T. Ma, X. Yang, and Y. Luo, "Load balance based workflow job scheduling algorithm in distributed cloud," *Journal of Network and Computer Applications*, vol. 152, p. 102518, 2020.

[23] T. L. Porta, C. Petrioli, C. Phillips, and D. Spenza, "Sensor mission assignment in rechargeable wireless sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 10, no. 4, p. 60, 2014.

[24] E. Uchiteleva, A. Shami, and A. Refaey, "Virtualization of wireless sensor networks through mac layer resource scheduling," *IEEE Sensors Journal*, vol. 17, no. 5, pp. 1562–1576, 2016.

[25] Z. Wei, F. Liu, Y. Zhang, J. Xu, J. Ji, and Z. Lyu, "A q-learning algorithm for task scheduling based on improved svm in wireless sensor networks," *Computer Networks*, vol. 161, pp. 138–149, 2019.

[26] C. M. de Farias, L. Pirmez, F. C. Delicato, W. Li, A. Y. Zomaya, and J. N. de Souza, "A scheduling algorithm for shared sensor and actuator networks," in *The International Conference on Information Networking 2013 (ICOIN)*. IEEE, 2013, pp. 648–653.

[27] N. Edalat and M. Motani, "Energy-aware task allocation for energy harvesting sensor networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2016, no. 1, p. 28, 2016.

[28] B. Liu, X. Xu, L. Qi, Q. Ni, and W. Dou, "Task scheduling with precedence and placement constraints for resource utilization improvement in multi-user mec environment," *Journal of Systems Architecture*, vol. 114, p. 101970, 2021.

[29] S. Javanmardi, M. Shojafar, R. Mohammadi, A. Nazari, V. Persico, and A. Pescapè, "Fupe: A security driven task scheduling approach for sdn-based iot–fog networks," *Journal of Information Security and Applications*, vol. 60, p. 102853, 2021.

[30] J.-q. Li and Y.-q. Han, "A hybrid multi-objective artificial bee colony algorithm for flexible task scheduling problems in cloud computing system," *Cluster Computing*, vol. 23, no. 4, pp. 2483–2499, 2020.

[31] M. D'Amico and J. C. Gonzalez, "Energy hardware and workload aware job scheduling towards interconnected hpc environments," *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[32] S. Singhal and A. Sharma, "A job scheduling algorithm based on rock hyrax optimization in cloud computing," *Computing*, pp. 1–28, 2021.

[33] T. Choudhari, M. Moh, and T.-S. Moh, "Prioritized task scheduling in fog computing," in *Proc. of the ACMSE 2018 Conf.*, 2018, pp. 1–8.

[34] H. Xu, Y. Liu, and W. C. Lau, "Optimal job scheduling with resource packing for heterogeneous servers," *IEEE/ACM Transactions on Networking*, 2021.

[35] K. Psychasand and J. Ghaderi, "High-throughput bin packing: Scheduling jobs with random resource demands in clusters," *IEEE/ACM Transactions on Networking*, vol. 29, no. 1, pp. 220–233, 2020.

[36] X. Fang, Z. Cai, W. Tang, G. Luo, J. Luo, R. Bi, and H. Gao, "Job scheduling to minimize total completion time on multiple edge servers," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 2245–2255, 2020.

[37] H. S. Arri and R. Singh, "Energy optimization-based optimal trade-off scheme for job scheduling in fog computing," in *2021 8th International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE, 2021, pp. 551–558.

[38] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*. Springer, 1972, pp. 85–103.

[39] R. L. Haupt, "Optimum population size and mutation rate for a simple real genetic algorithm that optimizes array factors," in *IEEE Antennas and Propagation Society International Symposium. Transmitting Waves of Progress to the Next Millennium. 2000 Digest. Held in conjunction with: USNC/URSI National Radio Science Meeting (C*, vol. 2. IEEE, 2000, pp. 1034–1037.

**Mustafa Can Çavdar** received his Ph,D, and MS degrees in computer engineering from Bilkent University, Ankara, Turkey, in 2022 and 2016, respectively. His research interests include cloud computing, wireless networks, and computer networks.

**Ibrahim Korpeoglu** received his Ph.D. and M.S. in Computer Science from University of Maryland at College Park in 2000 and 1996, respectively. He received his B.S. degree (summa cum laude) in Computer Engineering, from Bilkent University in 1994. Currently he is a full professor in Department of Computer Engineering at Bilkent University. Before joining Bilkent, he worked in Ericsson, IBM T.J. Watson Research Center, Bell Laboratories, and Bellcore, in USA. He received Bilkent University Distinguished Teaching Award in 2006. He speaks Turkish, English and German. His research interests include computer networks, wireless networks, cloud computing, and distributed systems. He is a member of ACM and a senior member of IEEE.

**Özgür Ulusoy** is a professor at the Department of Computer Engineering, Bilkent University, Ankara, Turkey. He has a Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign, USA. His current research interests include web databases and web information retrieval, multimedia database systems, social networks, and cloud computing. He has published over 150 articles in archived journals and conference proceedings. He is a member of IEEE and ACM.