RESOURCE OPTIMIZATION OF MULTI-PURPOSE IOT WIRELESS SENSOR NETWORKS WITH SHARED MONITORING POINTS

A DISSERTATION SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

COMPUTER ENGINEERING

By

Mustafa Can Çavdar

November 2022

Resource Optimization of Multi-purpose IoT Wireless Sensor Networks with Shared Monitoring Points

By Mustafa Can Çavdar

November 2022

We certify that we have read this dissertation and that in our opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Özgür Ulusoy(Advisor)

İbrahim Körpeoğlu(Co-Advisor)

Uğur Güdükbay

Özcan Öztürk

Ertan Onur

Ahmet Coşar

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

RESOURCE OPTIMIZATION OF MULTI-PURPOSE IOT WIRELESS SENSOR NETWORKS WITH SHARED MONITORING POINTS

Mustafa Can Çavdar
Ph.D. in Computer Engineering
Advisor: Özgür Ulusoy
Co-Advisor: İbrahim Körpeoğlu
November 2022

Wireless sensor networks (WSNs) have many applications and are an essential part of IoT systems. The primary functionality of a WSN is to gather data from certain points that are covered with sensor nodes and transmit the collected data to remote central units for further processing. In IoT use cases, a WSN infrastructure may need to be shared by many applications. Moreover, the data gathered from a certain point or sub-region can satisfy the need of multiple applications. Hence, sensing the data once in such cases is advantageous to increase the acceptance ratio of the applications and reduce waiting times of applications, makespan, energy consumption, and traffic in the network. We call this approach monitoring point-based shared data approach. In this thesis, we focus on both placement and scheduling of the applications, each of which requires some points in the area a WSN covers to be monitored. We propose genetic algorithm-based approaches to deal with these two problems. Additionally, we propose greedy algorithms that will be useful where fast decision-making is required. We realized extensive simulation experiments and compared our algorithms with the methods from the literature. The results show the effectiveness of our algorithms in terms of various metrics.

Keywords: wireless sensor networks, virtualization, Internet of Things, application placement, application scheduling, optimization.

ÖZET

PAYLAŞIMLI İZLENEN NOKTALAR KULLANARAK ÇOK AMAÇLI IOT KABLOSUZ SENSÖR AĞLARININ KAYNAK OPTİMİZASYONU

Mustafa Can Çavdar
Bilgisayar Mühendisliği, Doktora
Tez Danışmanı: Özgür Ulusoy
İkinci Tez Danışmanı: İbrahim Körpeoğlu
Kasım 2022

Kablosuz sensör ağları (WSN) birçok uygulamaya sahiptir ve IoT sistemlerinin önemli bir parçasıdır. Bir WSN'nin birincil işlevi, sensör düğümleri tarafından kapsanan belirli noktalardan veri toplamak ve toplanan verileri daha sonraki işlemler için uzak merkezi birimlere iletmektir. IoT kullanım durumlarında, bir WSN altyapısının birçok uygulama tarafından paylaşılması gerekebilir. Ayrıca belirli bir noktadan veya alt bölgeden toplanan veriler birden fazla uygulama ihtiyacını karşılayabilmektedir. Dolayısıyla, bu gibi durumlarda verilerin bir kez algılanması, uygulamaların kabul oranını artırmak, uygulamaların bekleme sürelerini, toplam çalışma sürelerini, enerji tüketimini ve ağdaki trafiği azaltmak için avantajlıdır. Bu yaklaşımı izleme noktası tabanlı paylaşılan veri yaklaşımı olarak adlandırıyoruz. Bu tezde, her biri bir WSN'nin kapsadığı alanda bazı noktaların izlenmesini gerektiren uygulamaların yerleştirilmesine ve çizelgelenmesine odaklanıyoruz. Bu iki problemi çözmek için genetik algoritma tabanlı yaklaşımlar öneriyoruz. Ek olarak, hızlı karar vermenin gerekli olduğu durumlarda faydalı olacak açgözlü algoritmalar öneriyoruz. Kapsamlı simülasyon deneyleri gerçekleştirdik ve algoritmalarımızı literatürdeki yöntemlerle karşılaştırdık. Elde edilen sonuçlar, çeşitli metrikler açısından algoritmalarımızın etkinliğini göstermektedir.

Anahtar sözcükler: kablosuz sensör ağları, sanallaştırma, Nesnelerin İnterneti, uygulama yerleştirme, uygulama çizelgeleme, optimizasyon.

Acknowledgement

First and foremost, I would like to express my gratitude to my advisors, Prof. Dr. Özgür Ulusoy and Prof. Dr. İbrahim Körpeoğlu, for their guidance and support during my PhD studies.

I would like to thank Prof. Dr. Uğur Güdükbay and Prof. Dr. Ertan Onur for being on my thesis monitoring committee and for their valuable advice during the meetings. I would also thank Prof. Dr. Özcan Öztürk and Prof. Dr. Ahmet Coşar for accepting to be on my dissertation jury.

I would also like to thank my dearest friends, Dr. Hasan Balcı, Dr. Arif Usta, Tolga Yılmaz, Umutcan Turan, and Oğuz Bilgen, with whom I shared a lot during my graduate studies. I would also like to extend my appreciation to thank my colleagues from the CS department.

I would also like to thank my parents, Ayfer and İsmail Çavdar, for their endless love and support throughout my life. None of this would have been possible without them.

Last but not least, I would like to thank my late grandmother, Fatma Nebahat Çıldır, who was the first teacher in my life. This dissertation is dedicated to her.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

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

The range of applications has grown rapidly since the inception of WSNs. 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 [1]. Intelligent parking systems [2], air quality monitoring with NO_2 sensors in cities [3], in-pipe monitoring of the quality of drinkable water [4], and noise monitoring in metropolitan areas [5] are some 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, collecting seismic and infrasonic signals from volcanoes [6], and supply-chain monitoring [7]. Actually, the applications of WSNs are so extensive that there are studies that investigate the use cases by dividing them into groups. For instance, Kandris et al. [8] categorize

the WSN applications into six groups: environmental, flora and fauna, health, industrial, military and urban applications. This broad range of WSN applications makes the optimization and efficient use of WSNs very vital.

Initially, WSNs were task-specific networks. A WSN was designed, developed, and optimized to support a single application. It was impossible to deploy any other application to an already running WSN. This led to redundant WSN deployments and inefficient utilization of WSN resources. However, the trend for WSNs has recently changed from being designed for a single application to designing WSNs that can support various applications with heterogeneous needs within a single network infrastructure [9]. For instance, to a single city-wide WSN infrastructure of sensor nodes with various sensing capabilities, various types of applications such as air quality monitoring, noise monitoring, crime detection, waste management, and traffic monitoring can be deployed. Another example would be a single building-wide WSN, which can be used for both structural health monitoring [10] and fire disaster detection [11] 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.

1.1 Motivation and Contributions

WSNs that can support multiple applications should be designed, utilized, and operated optimally; so that the placed applications can get good quality of services and the owners of the applications are well satisfied. To achieve this, network providers need a centralized controller and related policies. Software-Defined Networking (SDN) provides a mechanism that allows managing a WSN from a centralized controller. Thanks to SDN, decision-making processes for the operations of the networks are moved to logically implemented centralized controller software [12]. SDN enables virtualization that allows sharing of physical resources among multiple services, tasks, or applications and improves flexibility in the network [13]. Hence, Software-Defined Networking has a key role in the development

of next-generation networks and Internet of Things (IoT) [14]. With the help of SDN, applications can be scheduled and placed onto IoT-integrated WSNs with centralized algorithms efficiently and effectively.

In WSNs that support running multiple applications over the same physical network infrastructure, multiple applications may require the same data type (e.g., temperature, image, or video) to be collected from the same monitoring point in the monitored area. For instance, there may be two applications, one of which measures the average speed of vehicles between two points, and the other monitors the traffic density at the same points at certain times during the day by capturing their image. The data collection frequency of these two applications may not be equal to each other, i.e., measuring average speed requires more frequent data collection than monitoring traffic density; however, both applications require the same type of data. For such cases, we propose *monitoring point based shared data* approach, which enables sensing and transmitting data only once for multiple applications registered to sense the same monitoring point. Even though the processing requirement will not change, the network will have more sensing and communication resources available to place and schedule more applications simultaneously. This will help increase the number of applications placed in the network simultaneously and reduce both the total execution time of a set of applications and the waiting time of the newly arriving applications. The differences between shareable and unshareable approaches are summarized in Table 1.1.

To illustrate the benefits of the shareable approach, we consider the following example scenario. Let us consider a sensor network consisting of sensor nodes and base stations. Assume that there are multiple applications requiring a specific monitoring point to be sensed and that the monitoring point is in the sensing range of a single specific sensor node. Assume also that the sensor node can connect to just one particular base station with enough processing capacity. With the shareable approach, we can place many applications whose sensing requirements are at most the sensing capacity of the sensor node. However, with the unshareable approach, each placed application consumes additional sensing resources of the sensing node; therefore, we cannot place many applications. For instance, suppose that the sensor node has a sensing rate capacity of 400 kbps,

Table 1.1: Comparison of shareable and unshareable approaches

	Shareable Approach	**Unshareable Approach**
Sensing	Required resources for each monitoring point are less compared to unshareable since sensing requirement of each monitoring point is the maximum of individual requests of applications.	Required resources for each monitoring point are more since the sensing requirement of each monitoring point is the sum of individual requests of applications.
Transmission	Since less data are sensed by sensor nodes, amount of transmitted data is also less.	Since more data are sensed, amount of transmitted data is more compared to shareable approach.
Benefit	Less resources are used for the same applications. More available resources for applications yet to be deployed.	More resources are used for the same applications. Less available resources for applications yet to be deployed.
	Less energy spent for the deployed applications.	More energy spent for the deployed applications.

and there are four applications with the sensing requirements of 300 kbps, 100 kbps, 150 kbps, and 200 kbps. With the shareable approach, all four applications can be placed on the network since the used sensing rate would be 300 kbps which is less than the capacity of the sensor node. However, with the unshareable approach, at most two applications can be placed on the network, considering the capacity of the sensor node. Therefore, with the shareable approach, a WSN can support many more applications with the same amount of resources. Moreover, the average energy spent per application would be less than the unshareable approach.

In this dissertation, we make the following contributions:

- A monitoring point-based shared data approach: Data sensed from a monitoring point can be shared by multiple applications, and our methods consider this whenever possible. Each application indicates its required sensing rate for a monitoring point, and our scheme multiplexes and satisfies these requests using a single stream of data sensed from the monitoring point. This way, we reduce the sensing and communication resources used per application without violating sensing requirements.

- Network structure: We focus on sensor networks with limited bandwidth and computational resources at the edge. Data are sensed from monitoring points by sensor nodes and sent to base stations (or cluster-heads) to be processed and/or conveyed further towards one or more data centers. We assume that the network among base stations and sinks is a high-speed network. Hence, we are concerned with the efficient use of the limited bandwidth available between sensor nodes and base stations.

- Two algorithms for application placement problem in wireless sensor networks. The first one is a greedy algorithm which is a version of the Worst Fit approach. The other one is GABAP, which is a genetic algorithm-based solution for the placement problem. GABAP aims to increase the number of placed applications at the same time by selecting the applications to be admitted and assigning monitoring points to sensor nodes and base stations in a close-to-optimal way. GABAP can also *migrate* monitoring points from their assigned sensor nodes and base stations to other ones.

- Four algorithms for application scheduling problem in wireless sensor networks. We have three greedy algorithms, LMPF, LMSF, and LTSF, each of which considers different criteria for ordering applications to admit. The last algorithm we propose is GABAS, a novel genetic algorithm that decides which sensor nodes and base stations will be used to sense and process data from the monitoring points requested by the application while admitting and scheduling the applications.

- A linear programming framework for both the application placement and application scheduling problems for small networks. This framework enables us to compare our GABAP and GABAS algorithms against the optimal solution.

- Extensive simulation experiments where we compare our proposed algorithms with the other methods from the literature in terms of various metrics.

1.2 Outline

The rest of the dissertation is organized as follows. In Chapter 2, we describe the resource allocation and task scheduling studies that are present in the literature. In Chapter 3, we give basic information about wireless sensor networks, genetic algorithms, and greedy algorithms. In Chapter 4, we detail the application placement problem and our approaches to solve it. In Chapter 5, the application scheduling problem is presented, and we give information about our proposed algorithms for this problem. In Chapter 6, we describe our simulation experiments and present the results of the experiments. In Chapter 7, we conclude the dissertation.

Chapter 2

Related Work

Optimization of computer networks is a very broad study area in the computer science field. In this chapter, we present the related work in the literature. Previous studies have different optimization objectives, such as increasing efficiency of resource allocation and scheduling, load balancing, increasing the coverage area of the network, and so on.

2.1 Resource Allocation

Resource allocation is the process of the assignment of available resources to various types of applications, tasks, jobs, or services. The assignment of the resources should be optimized to increase the availability and reliability of the network and reduce the energy consumption in the network.

Wei et al. [15] describe NSGA-II which is a genetic algorithm-based approach for solving the resource allocation problem in vehicular cloud computing. They apply a dynamic crossover probability and a dynamic mutation probability to improve the individuals of the population. NSGA-II aims to minimize the cost for the cloud owner and increase the acceptance rate for tasks. Ali et al. [16] propose a supervised deep learning-based approach to develop resource allocation

techniques in multi-tier networks. Chen et al. [17] present a two-level framework based on Q-learning and sequential quadratic programming. Their framework aims to reduce the cost caused by energy consumption and running delay of the tasks in cloud-edge heterogeneous networks. Subhash and Udayakumar [18] propose a sunflower whale optimization algorithm to increase the utilization of resources in a cloud system. Xia et al. [19] detail a method based on both ant colony optimization and genetic algorithm for resource allocation in edge clouds. Their method decreases latency and improves resource utilization.

Resource allocation in wireless sensor networks (WSNs) is also a topic that is studied extensively. In [20], Raee et al. assign tasks to sensor nodes in a way that energy consumption is minimized by using Integer Linear Programming. They compare their solution with a traditional WSN which is a non-virtualized network. Ojha et al. [21] describe a scheme for dynamic IoT applications to preserve energy efficiency in a cloud system. They model the interaction between cloud owners and sensor owners as a Stackelberg game. Lemos et al. [22] propose an Ant Colony Optimization method to handle virtual sensor provisioning in WSNs. Their algorithm selects an optimal set of sensor nodes to respond to user demands while withholding energy consumption in the whole network. Delgado et al. [9] use mathematical programming to solve both application admission and network slicing problems in WSNs. They evaluate their method on realistic WSN infrastructures. The same authors expand their work to handle dynamic application admission in shared sensor networks in [23].

SenShare [24] is a platform that addresses the technical difficulties of transforming a physical sensor network into an infrastructure that supports multiple applications. In [25], Bhattacharya et al. present UMADE which is an application deployment system that allocates applications to sensor nodes by considering their Quality-of-Monitoring (QoM). Therefore, they aim to increase overall QoM in the whole network within resource constraints. The two QoM attributes they use are variance reduction and detection probability. Ajmal et al. [26] propose an admission control algorithm for WSNs to which applications are deployed for a certain time interval. Their method tries to minimize the total execution time of applications. Moreover, they also assess the feasibility of their scheduling based

on co-arrived applications. Cionca et al. [27] propose Judishare, a framework that enables the reuse of sensing and communication resources in shared sensor networks. In [28], Bousnina et al. focus on the resource allocation problem in a virtual sensor network that has sensor nodes with various amounts of resources. They propose a greedy approach to solve the problem. This greedy method is faster than the methods which optimally solve the same problem, however, underperforms compared to them.

Tynan et al. [29] propose a Multi-Agent System architecture that deals with embedding virtual sensor networks on a WSN. They use hibernation of idle resources to reduce power consumption. Xu et al. [30] present a local search algorithm for application allocation in shared sensor networks. They aim to maximize QoM by considering resource constraints such as memory and bandwidth. They compare their method with simulated annealing by using both real-world datasets and simulated networks that are randomly generated. Li et al. [31] propose a framework that considers the load condition of every node in a wireless network and initiates a re-embedding operation if necessary. In [32], Abreu et al. describe a QoS-based application admission for WSNs for the biomedical domain. Instead of applications, their proposed work decides when to admit a new sensor node on the network. In [33], Rahmati et al. present a load balancing algorithm for efficient routing in WSNs. They consider energy consumption and resource allocation and show that a uniform distribution of WSN load leads to the most efficient routing.

Our work described in Chapter 4 of this dissertation is different from the above studies as follows. We propose GABAP, which is a novel genetic algorithm that decides which sensor node will collect data from each monitoring point, which base station will process the collected data of each monitoring point, and change the sensor node and base station assignments of a monitoring point if necessary. Moreover, GABAP also determines which applications should be admitted to the network.

2.2 Task Scheduling

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. Abualigah et al. [34] propose MALO, a multi-objective ant-lion optimization algorithm to solve the task scheduling problem in cloud computing environments. It aims to minimize makespan and maximize the utilization of cloud resources. Shukri et al. [35] present an enhanced version of Multi-verse Optimizer (MVO) [36]. They compare their proposed method with MVO and a Particle Swarm Optimization algorithm and show that Enhanced MVO has better performance in terms of resource utilization and makespan. Velliangiri et al. [37] describe an electro search and genetic algorithm hybrid method that combines the advantages of both algorithms for task scheduling in the cloud. They consider load balancing, makespan, resource utilization, and cost as comparison metrics. Sulaiman et al. [38] present a hybrid heuristic that deals with task scheduling in heterogeneous computing environments. They compare their method with four other methods in terms of average makespan, average running time, and average schedule length ratio.

Alboaneen et al. [39] deal with joint optimization of task scheduling and virtual machine placement in cloud data centers. They propose a metaheuristic optimization algorithm to solve the aforementioned problem. They compare their method with three other evolutionary algorithms and show that it produces better results in terms of execution cost, makespan, and resource utilization. Zhou et al. [40] describe a genetic algorithm-based method called MGGS which is combined with a greedy strategy to attack task scheduling problems in cloud systems. MGGS is compared with several existing algorithms in terms of QoS parameters, average

response time, and total completion time. Yang et al. [41] propose a task scheduling algorithm that considers game theory for cloud environments. The algorithm aims to reduce energy consumption. Chen et al. [42] detail a whale optimization algorithm for task scheduling in the cloud. The proposed algorithm has a better performance compared to metaheuristic algorithms like Ant Colony and Particle Swarm Optimization in terms of resource utilization.

Chhabra et al. [43] present MOHSCFA which is a hybrid of cuckoo search and firefly algorithm to overcome offline parallel job scheduling in a high-performance computing grid. Their proposed algorithm aims to reduce makespan, energy consumption, and average flow time. Padhy and Chou [44] deal with the scheduling problem in virtualized data centers. They propose a genetic algorithm-based scheduling method to minimize virtual machine migrations. Mansouri et al. [45] describe a cost-based job scheduling algorithm that aims to reduce the response time of both data-intensive and computation-intensive jobs in cloud data centers. Li et al. [46] present a job scheduling algorithm to achieve cloud resource utilization.

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. [47] 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. [48] 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. [49] 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. [50] propose an application scheduling algorithm for shared actuator and sensor networks. Their algorithm aims to reduce energy consumption in the network. Edalat and Motani [51] 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. [52] propose Horae, a task scheduler for mobile edge computing. The scheduler aims to improve resource utilization in the MEC environment as well as select the edge server that satisfies placement constraints for each task. Javanmardi et al. [53] present FUPE, a security-aware task scheduler for IoT fog networks. It is a fuzzy-based multi-objective Particle Swarm Optimization algorithm, and the authors show that it has better performance than the other compared algorithms in terms of average response time and network utilization. Li and Han [54] describe an artificial bee colony algorithm (ABC) for task scheduling in the cloud. The proposed ABC algorithm is compared against several works from the literature. The evaluation metrics they consider are makespan, maximum device workload, and total device workload. D'Amico and Gonzalez [55] propose EAMC, which is a multi-cluster scheduling policy. It predicts the energy consumption of jobs and aims to reduce makespan, response time, and total energy consumption. Singhal and Sharma [56] present a Rock Hyrax Optimization algorithm to schedule jobs in heterogeneous cloud systems. They consider evaluation metrics like makespan and energy consumption.

Choudhari et al. [57] propose a priority-based task scheduling algorithm for fog computing systems. Their algorithm first assigns an arriving request to the closest fog server, and within that fog server, it places the task into a priority queue. Xu et al. [58] apply online convex optimization techniques to schedule arriving jobs with multi-dimensional requirements in heterogeneous computing clusters. Psychasand and Ghaderi [59] describe algorithms based on Best Fit and Universal Partitioning to schedule jobs with various resource demands. Fang et al. [60] aim to reduce total job completion time in edge computing systems. They propose an approximation algorithm for both offline and online scheduling. Arri and Singh [61] describe an artificial bee colony algorithm that also makes use of an artificial neural network for job scheduling in fog servers.

Unlike previous works in the literature, we propose a novel genetic algorithm, GABAS, in Chapter 5 of this dissertation which assigns monitoring points to sensor nodes and base stations, and determines the admission order of the waiting

applications to minimize the makespan. Moreover, we also present three greedy algorithms, LMPF, LMSF and LTSF, each of which has different criteria while scheduling the applications.

In this dissertation, the works presented in Chapters 4 and 5 make use of the monitoring point based shared-data approach we propose. With this approach, sensor nodes do not need to collect different data for each application from a single monitoring point. This strategy allows wireless sensor networks to support more applications simultaneously.

Chapter 3

Background

In this chapter, we give brief information on wireless sensor networks, genetic algorithms, and greedy algorithms.

3.1 Wireless Sensor Networks

Sensors are devices that detect changes or events in the area they cover and send the collected data to other electronic devices. The use of sensors is expanded beyond the traditional fields such as temperature, flow measurement, and pressure, thanks to advances in micro-controller platforms and micro-machinery [62]. Actuators are mechanical devices that monitor the power of the sensor node, move the sensor or adjust the settings of the sensor [63].

Wireless sensor networks (WSNs) are a group of spatially dispersed sensors and actuators within a wireless communication infrastructure that aim to monitor environmental and physical conditions at various locations and send the gathered data to the central locations for storage, viewing, and analysis [64]. A sample network model is provided in Figure 3.1. WSNs can consist of several hundreds of nodes and each node can connect to numerous others. These nodes carry out the following tasks in the network: sensing, relaying, processing the data, or

exchanging the data with another network [65]. Sensor nodes are the elements that sense the data, whereas the nodes that relay the data are called routers. Base stations or sink nodes are the nodes that are used for processing and exchanging data.

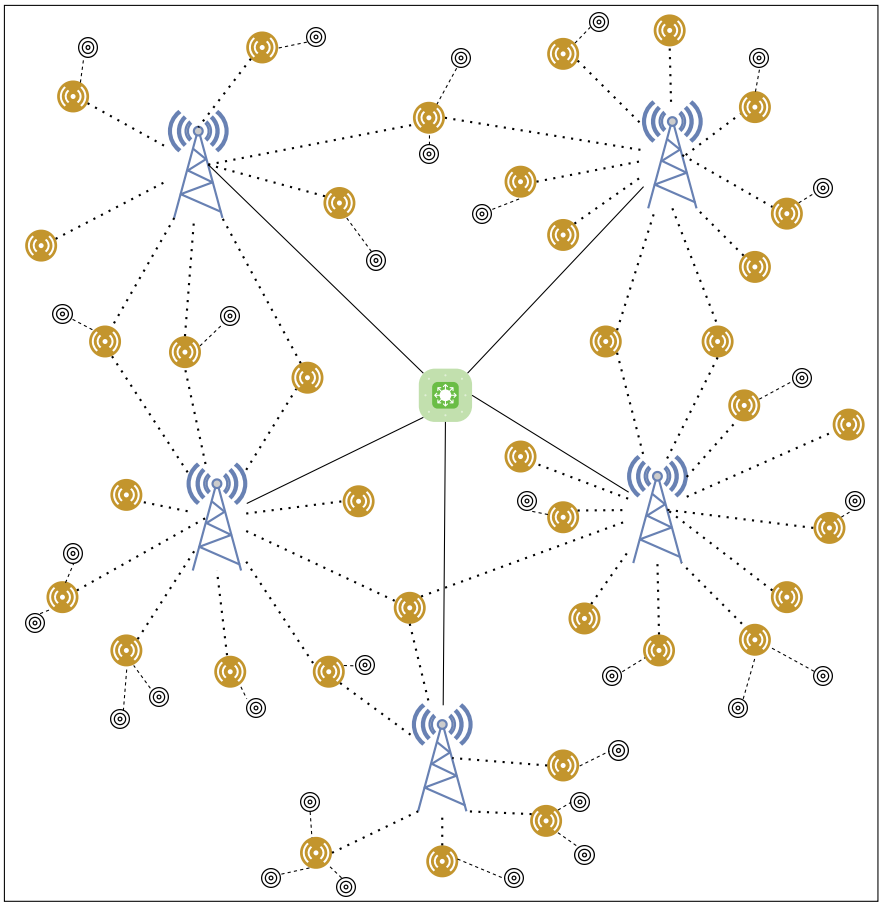


Figure 3.1: Network model

Sensor nodes in a WSN should be lightweight and portable. Each sensor node generally has a power source, a transducer, and a transceiver. The power source can be either an attached battery or an energy-harvesting solar panel, where it is applicable. The transducer is the component that generates electrical signals based on sensed phenomena. The transceiver is used for transmitting the data to a central controller (a base station) and receiving commands from it [66].

Base stations are the WSN elements that have more processing power and larger memory compared to sensor nodes. Moreover, they are connected to the

power grid, which means they have a healthier power source than sensor nodes. Therefore, the data collected by sensor nodes are conveyed to base stations. Then, base stations can process the data or forward them to other networks or end users [66].

The application domain of wireless sensor networks is vast. Some examples it includes are Internet-of-Things [67], traffic monitoring [68], environmental monitoring [69], natural disaster prevention [70], fire detection [71], smart homes [72], air quality monitoring [73] and landslide detection [74].

There are several architectures in which WSNs can be designed [75]. Some examples are as follows:

- *Point-to-Point Topology*: This topology contains a long-range and high-capacity wireless link between two sensor nodes. A single communication channel between nodes is a secure communication path. However, if it fails, the whole communication will be cut down [76].
- *Star Network*: Every sensor node is connected to a centralized communication hub. Therefore, there is no direct communication between sensor nodes. All communication is realized through the central unit. Any failure in the central hub results in the failure of the entire network [75].
- *Tree Topology*: Nodes in a WSN are connected in a hierarchical way. Collected data is transmitted from the child node to the parent node [77].
- *Mesh Networks*: Each node can communicate directly with any other node. This is the most reliable network structure, as a single-point failure does not affect other nodes. However, networks having this topology are not power-efficient, while a WSN must be designed in an energy-aware way [76].

There are different types of WSNs depending on the environment they are deployed or the data type they are sensing [63]. Some major WSN types are as follows:

- *Terrestrial WSN*: A WSN consists of hundreds or thousands of sensor nodes that are deployed on land. Since the sensor nodes are above ground, solar panels harvesting energy can be used as a secondary power source. Sensor nodes are placed more densely compared to underwater and underground WSNs [75].

- *Underwater WSN*: A WSN consists of several sensor nodes that are placed underwater. In this type of WSN, sensor nodes are more expensive since they must be water-resistant. Sensor nodes are deployed more sparsely than terrestrial WSNs. The transmission of data is realized via acoustic waves [78].

- *Underground WSN*: A WSN consists of sensor nodes that are placed underground. Underground WSNs are generally deployed in mines or caves to monitor the conditions in those environments [79].

- *Mobile WSN*: Unlike traditional WSNs, the sensor nodes of a mobile WSN (MWSN) have the ability to move. Therefore, MWSNs are more adaptable to any topology changes in the area, which makes MWSNs more flexible [80].

- *Multimedia WSN*: A WSN whose sensor nodes are equipped with microphones and cameras to collect data in image, video, or audio form. Therefore, this WSN type requires more resources and advanced compression and transmission techniques [81].

3.2 Genetic Algorithms

Genetic algorithms are a type of search meta-heuristics that mimics the natural selection process of evolution in real life. They are developed to find close-to-optimum solutions for complex optimization problems.

Genetic algorithms were invented in the early 1970s by John Henry Holland and his students at the University of Michigan [82]. Holland's initial aim was to

investigate what an adaptation of natural selection could achieve instead of developing new algorithms for new problems. In his book titled *Adaptation in Natural and Artificial Systems*, Holland describes genetic algorithms as *an abstraction of biological evolution*.

Genetic algorithms are useful for optimization problems with larger search spaces. In the search space of the given problem, a genetic algorithm looks for better solutions rather than scanning all of the search space.

A genetic algorithm is defined by the following:

- a genetic representation of solutions,
- a fitness function,
- operations to create new individuals (selection, crossover, and mutation), and
- a termination condition.

The genetic representation of an individual is the actual solution the individual offers. It is usually an array of bits; however, any data structure can be used to represent the solutions. Any feature of the selected representation is the same for each individual in the population. Therefore, any operation (crossover, mutation) on this representation can be realized easily.

The fitness function is the figure of merit that determines how close a solution is to the optimal one. The output of the fitness function is the fitness score of the given individual. The genetic representation of the individual with the highest fitness score is the optimum solution in the population to the given problem.

In genetic algorithms, selection, crossover, and mutation functions are used to create the individuals of the next generation. The selection operation determines which individuals will be paired up for the crossover operation. There are various types of selection operations, such as roulette wheel selection, tournament

selection, and rank selection. Selection operation should be carefully designed to increase genetic diversity in the population to avoid a local maximum. Crossover operation is the creation of the individuals of the next population. The genetic representation of the new individuals is determined according to their *parents' genes*. In the crossover operation, for each gene, there is a probability for each parent to pass its gene. While designing the crossover operation, one of the parents can be favored, or they can have an equal chance to pass their genes to the offspring. After the crossover operation, the newly created individual is exposed to the mutation operation. Mutation operation somehow changes the individual's genes with a very small probability. This operation is realized to include randomness to the genes of the individual to avoid local maximums.

The output of crossover and mutation operations may violate the constraints of the problem. These violations are prevented by checks during the operations, fixed after the operations are done, or punished heavily in the fitness function.

The termination condition of the algorithm determines when the genetic algorithm finishes. Reaching a certain number of generations or not having a significant improvement in the fitness score of the best individual are two examples of termination conditions.

At the beginning of a genetic algorithm, an initial population (Generation 0) is created. This creation is generally realized by randomly assigning values to the genes of the individuals in the population. After the population is created, the fitness scores of the individuals are calculated. Then, selection and crossover operations are realized to create the individuals of the next generation. Mutation operation is applied to newly created individuals, and fitness scores of the individuals of the new generation are calculated. These operations are repeated until the termination condition is satisfied. The flow of the operations is visualized in Figure 3.2.

Genetic algorithms are widely used in the literature for various optimization problems from different research fields. Computer networks are one of the major areas they are used. Predicting TCP throughput [83], cache placement [84], polar

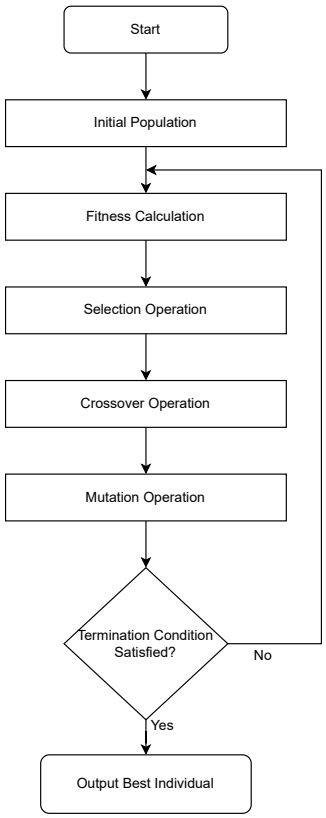


Figure 3.2: Flowchart of genetic algorithms

code design [85], topological design [86, 87], load balancing [88] and broadcast scheduling [89] are some topics in computer networks that can be solved by genetic algorithms.

In image processing, there are studies that make use of genetic algorithms. Image segmentation [90], image enhancement [91] and image restoration [92] are some example topics where genetic algorithms are used. In video processing, they are used in video segmentation [93], gesture recognition [94] and face recognition [95].

In materials science, example use of genetic algorithms are polymer design [96], material structure design [97], hardness prediction of high-entropy alloys [98] and structure interface prediction [99]. In operations management, they can be used to solve facility layout problem [100], job-shop scheduling [101], supply chain network design [102] and forecasting [103].

3.3 Greedy Algorithms

A greedy algorithm is a method to solve a given problem that selects the best possible option based on the current situation [104]. Obviously, this approach does not guarantee and usually does not achieve the overall optimum solution for the problem. However, greedy algorithms are pretty quick compared to other optimization approaches, such as genetic algorithms and dynamic programming, in obtaining a solution which is useful in cases where fast decisions are required. Another advantage of greedy algorithms is that they are easy to design and implement [105].

There is a diverse set of problems to that greedy algorithms can be applied. Best Fit for bin packing problems, Shortest-Job-First for job scheduling, and Prim's Algorithm for finding minimum spanning tree in graphs are some examples of well-known greedy algorithms.

There exists a variety of works in the literature that makes use of greedy approaches. Community detection [106], flow-shop scheduling [107, 108] and anomaly detection in networks [109] are some example problems that can be solved by greedy algorithms.

Chapter 4

Application Placement in Wireless Sensor Networks

In this chapter,[1] we explain the application placement problem in wireless sensor networks. We first describe the problem statement, and then present our algorithms to solve the problem.

4.1 Problem Statement

We consider a wireless sensor network (WSN) that is shared by multiple applications and owned by a single sensor network infrastructure provider. Our WSN model consists of sensor nodes with equal sensing rates, sensing ranges and energy budget, base stations with equal processing capacities, and connections between those sensors and base stations with equal bandwidth capacity. A sensor can gather data from the monitoring points within its sensing range and it is connected to the base stations within its communication range.

[1]This chapter is based on the work [110]; Mustafa Can Çavdar, Ibrahim Korpeoglu, and Özgür Ulusoy, 2022. "Application placement with shared monitoring points in multi-purpose IoT wireless sensor networks." Computer Networks, vol. 217. DOI: https://doi.org/10.1016/j.comnet.2022.109302.

Table 4.1: Parameters used in the placement 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
S_k	set of sensor nodes covering monitoring point k
M_j	set of monitoring points required by application j
M_{il}	set of monitoring points whose data is transferred from sensor node i to base station l
z_j	binary variable indicating whether application j is deployed
x_k	binary variable indicating whether monitoring point k is sensed
x_{ik}	binary variable indicating whether sensor node i is actively sensing monitoring point k
x_{ilm}	constant indicating whether sensor node i is connected to base station l through connection m
x_{jk}	binary variable indicating monitoring point requirement of application j for monitoring point k is satisfied
r_{jk}	sensing rate requirement by application j at monitoring point k
r_k	sensing rate requirement of monitoring point k
u_k	sum of sensing rate requirements of applications for monitoring point k
R_i	sensing rate capacity of sensor node i
E_s	energy budget for each sensor node
P_l	processing capacity of base station l
C_m	bandwidth capacity of connection m
R'_i	used sensing resource of sensor node i
P'_l	used processing resource of base station l
C'_m	used bandwidth of connection m
TC	transmission coefficient
PC	processing coefficient

We aim to maximize the number of applications that are successfully deployed. We can formally express this problem as follows:

$$\sum_{j \in A} z_j \tag{4.1}$$

is maximized subject to

$$z_j = \prod_{k \in M_j} x_{jk} \qquad \forall j \in A \tag{4.2}$$

$$x_{jk} \leq x_k \qquad \forall k \in M \tag{4.3}$$

Eq. 4.2 shows that to place an application, all the monitoring point requirements of the application must be satisfied. Requirements of a monitoring point can be satisfied if the monitoring point is sensed by a sensor node as shown in Eq. 4.3.

$$x_k = \sum_{i \in S_k} x_{ik} \leq 1 \qquad \forall k \in M \tag{4.4}$$

$$r_k = max(r_{jk} \times x_{jk}) \qquad \forall k \in M \tag{4.5}$$

$$u_k = \sum_{j \in A} (r_{jk} \times x_{jk}) \qquad \forall k \in M \tag{4.6}$$

A monitoring point is sensed by at most one sensor node as shown in Eq. 4.4. This assumption is derived from [9]. With Eqs. 4.5 and 4.6, we indicate the calculation of the required sensing rate by a monitoring point for shared and unshared cases, respectively.

$$\sum_{k \in M} x_{ik} r_k \leq R_i \qquad \forall i \in S \tag{4.7}$$

$$\sum_{k \in M} x_{ik} u_k \leq R_i \qquad \forall i \in S \tag{4.8}$$

Eqs. 4.7 and 4.8 are the sensing constraints for shareable and unshareable approaches, respectively. The sensing constraint indicates that the sensing capacity of a sensor node must be at least large enough to meet the sensing requirements of monitoring points it actively senses.

$$\sum_{k \in M_{il}} r_k TC \leq \sum_{m \in C} x_{ilm} C_m \qquad \forall i \in S, \forall l \in B \tag{4.9}$$

$$\sum_{k \in M_{il}} u_k TC \leq \sum_{m \in C} x_{ilm} C_m \qquad \forall i \in S, \forall l \in B \tag{4.10}$$

For the shareable approach, our connection constraint is shown in Eq. 4.9. Eq. 4.10 is the connection constraint of unshareable approach. We have a *Transmission Coefficient* whose value is between zero and one because we assume that the data collected at sensor nodes can be compressed before they are sent to a base station to be processed. The connection constraint specifies that a connection should have enough bandwidth to transfer the data from the sensor node to the base station it connects.

$$\sum_{i \in S} \sum_{k \in M_{il}} u_k PC \leq P_l \qquad \forall l \in B \tag{4.11}$$

Eq. 4.11 is the processing constraint. We use a *Processing Coefficient* since some data sent to base stations may be noise and they do not need to be processed. With the processing constraint, we enforce that the total processing requirement of monitoring point data sent to a base station must not exceed the base station's processing capacity.

A monitoring point's requirement is calculated by considering only placed applications. If our algorithm does not admit an application to the network, the requirements of that application are ignored in calculating the requirement of the monitoring point. We assume that the applications arrive at the network dynamically, in batches or one by one. Therefore, there exist both initially known applications that are already placed and new applications that are yet to be deployed.

4.1.1 Energy Constraint

The following equations are to calculate the energy spent by the sensor nodes and base stations in the network. The total energy spent in the edge network is the sum of energy spent by each sensor node and base station.

$$E_{il}^{t} = x_{ilm} C'_{m} * (\beta_1 + \beta_2 * d_{il}^{4}) \qquad \forall i \in S, \forall l \in B \tag{4.12}$$

$$E_{i}^{s} = \rho * R'_{i} \qquad \forall i \in S \tag{4.13}$$

$$E_i = \sum_{l \in B} (E_{il}^{t}) + E_{i}^{s} + E_{act} + E_{mig} * N_i \qquad \forall i \in S \tag{4.14}$$

Eq. 4.12 is the calculation of energy that is spent by data transmission by a sensor node i to a base station l. Eq. 4.13 describes the calculation of energy cost for sensing data. For each sensor node, the sum of transmission energy cost to all connected base stations, sensing energy cost, activation cost, and the cost of migration to the sensor node is equal to the total energy spent by the sensor node which is shown in Eq. 4.14. $\beta_1 = 50$ nJ/bit, $\beta_2 = 0.0013$ pJ/bit m^4, $\rho = 0.5$ nJ/bit. These values and the formulas are derived from [111]. E_{act} is the activation energy for sensor nodes and base stations. E_{mig} is the migration cost. Both E_{act} and E_{mig} are equal to 10 J [9].

$$E_{il}^{r} = \beta_1 * x_{ilm} C'_m \qquad \forall i \in S, \forall l \in B \tag{4.15}$$

$$E_{l}^{p} = \gamma * P'_l \qquad \forall l \in B \tag{4.16}$$

$$E_l = \sum_{i \in S} (E_{il}^{r}) + E_{l}^{p} + E_{act} \qquad \forall l \in B \tag{4.17}$$

Eq. 4.15 is to calculate the energy dissipation for the reception of the transmitted data to the base stations. Eq. 4.16 is to calculate the energy cost of processing received data at the base stations. Total energy spent by a single base station is the sum of total energy spent for the reception of the data, processing of the data, and the activation cost. It is shown in Eq. 4.17. $\gamma = 5$ nJ/bit. The value of γ and the formulas are derived from [111].

$$E_i \leq E_s, \qquad \forall i \in S. \tag{4.18}$$

Eq. 4.18 shows the only energy constraint in our model. It indicates that the energy spent by each sensor node cannot exceed its energy budget. In our model, we do not have any energy constraints for base stations, since we assume that base stations can be plugged into the grid, as in [9].

4.2 Hardness of Application Placement

The application placement problem which is described above is a resource allocation problem. Here, to prove that the aforementioned problem is NP-hard, we reduce the 3-SAT problem which is a well-known NP-hard problem [112], to the application placement problem.

4.2.1 3-SAT

Boolean satisfiability problem (SAT) is the problem of determining whether there is an interpretation that satisfies a given Boolean formula. In, 3-satisfiability (3-SAT), the formula consists of clauses each having exactly three literals. For instance, $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$ and $c_i = x_a \vee x_b \vee x_c$ where $a, b, c \in \mathbb{N}$ is in 3-SAT form.

4.2.2 Reduction to Application Placement

For each monitoring point request, we need to find a sensor to gather data from the monitoring point and a base station to process the gathered data. The Boolean formula ϕ we use for reduction consists of three literals. Let $\phi = C_1 \wedge C_2 \wedge C_3$. If C_1 and C_2 have common attributes with non-contradictory values, we say that the monitoring point is in the sensing range of the sensor with enough available sensing resources. Similarly, if C_2 and C_3 have common attributes with non-contradictory values, the sensor and the base station with enough processing power have a connection with enough bandwidth. Therefore, if we can find a feasible solution for ϕ, then we can sense that monitoring point. For instance, let $\phi = (x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_5)$. We can reduce it as in Figure 4.1. Common literals of C_1 and C_2 are x_2 and x_4 and the only common literal of C_2 and C_3 is x_2. With $x_1 = false$, $x_2 = false$, $x_3 = true$, $x_4 = true$ and $x_5 = true$, ϕ is feasible and we can find a suitable sensor and a suitable base station for the monitoring point.

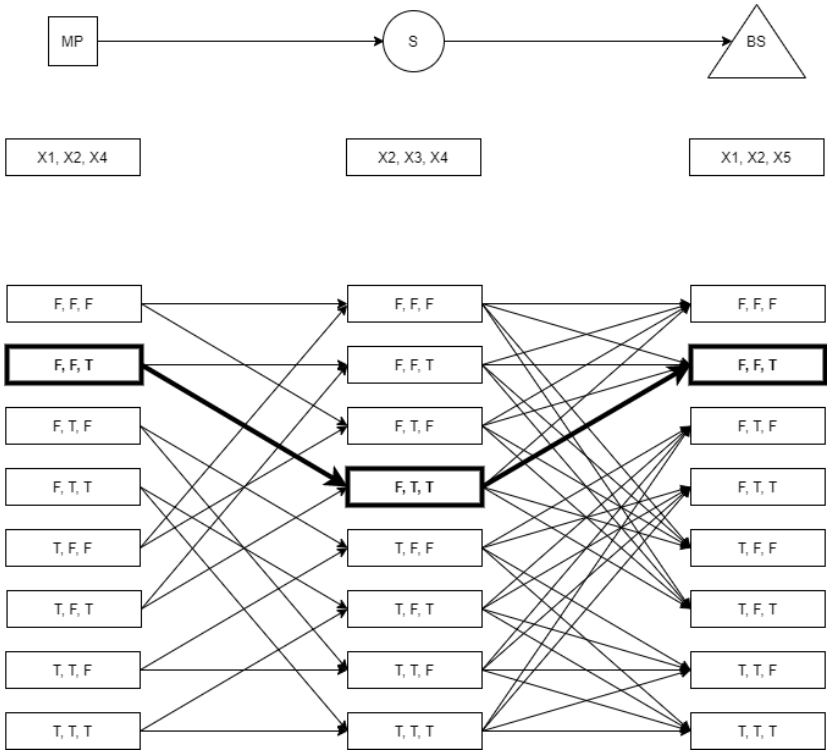


Figure 4.1: 3-SAT reduction to Application Placement.

4.3 Genetic Algorithm Based Application Placement (GABAP)

4.3.1 Chromosome Structure

Each individual has three genes: *Sensor genes*, *Base station genes*, and *Application genes*. All three genes are integer lists. The sizes of *Sensor genes* and *Base station genes* are equal to the number of the monitoring points available in the network. These two genes represent which sensor node and base station will be used for each monitoring point. The size of *Application genes* is equal to the number of arriving applications. The elements of this list can be either 0 or 1 and represent whether applications must be admitted to the network.

4.3.2 Initial Population Creation

In genetic algorithms, each individual of a generation is created by crossover operations whose parameters are individuals from the previous generation. Since there is not any generation before the initial population (Generation 0), we need to create individuals of this generation randomly.

Since *Application Genes* of an individual determines whether the application should be admitted, each application gene of an individual from the initial population is determined with a 50% probability. It is either 0 or 1.

Sensor Genes and *Base Station Genes* of the initial population are determined together. For each monitoring point that is requested by at least one application from the current batch, we first determine a sensor gene. It is randomly determined among the sensor nodes whose sensing range covers the monitoring point. Each possible sensor node has an equal probability. After the sensor gene is determined, we randomly select a base station among the ones the chosen sensor node has a connection to. Again, each possible base station has an equal chance to be selected. For monitoring points that are not requested by any application, we do not determine a sensor gene or base station gene to avoid unnecessary migrations in the current or future batch of applications. Sensor and base station genes for monitoring points that are not requested are set to -1.

4.3.3 Fitness Calculation

Our fitness calculation is designed to measure how close an *Individual* (a candidate solution to the problem) is to the optimum solution. The value obtained from the calculation is called the individual's *fitness score*. Equation 4.19 shows the fitness calculation of GABAP.

$$fitness = PAC - \alpha \times MC - \beta \times (WAPS + WMS) \tag{4.19}$$

Placed Application Count (PAC) is the number of applications that can be placed into the network. For each application j with *ApplicationGene(j)=1*, we check whether all of its requirements can be met without violating network constraints. If we cannot place application j, then we increase *Wrong Application Placement Suggestion (WAPS)* by one.

Our algorithm can also *migrate* a monitoring point from one sensor node and base station to another pair which is explained in Section 4.3.7. If a monitoring point has an already assigned sensor node and an already assigned base station, and the genes of the individual suggest a different sensor node or a different base station, we say that the individual suggests a migration. If a monitoring point whose migration is suggested is required by a successfully placed application, then we increase *Migration Count (MC)* by one. However, if the migration violates the network constraints, then we increase *Wrong Migration Suggestion (WMS)* by one.

Values of α and β have a direct effect on the fitness score. A migration operation may be helpful to place a larger number of applications, however, it is not a free operation. If an application can be placed both with and without migration, we favor placing it without migration because of the cost. Nevertheless, placing an application with migration is preferable to not placing the application at all. Therefore, the value of α is in the interval of (0,1). The sum $WAPS + WMS$ indicates the number of times that the genes of the individual violate the constraints of our network. Since violating the network constraints is undesired, β has a value that is big enough to ensure that not attempting to place any applications at all is a better outcome than having network constraint violations. In our experiments, we observe that $\alpha = 0.1$ has the best value for the application placement and migration trade-off. For β, 1000 is a value that is large enough to prevent violations. A feasible solution for the network means a non-negative fitness score for the individual.

4.3.4 Selection Operation

We use the tournament selection method to find an individual for each individual in the population to pair up for the crossover operation. Tournament selection has its own population (i.e., the tournament population) which includes a subset of individuals that are randomly chosen from the general population. It outputs the individual with the highest fitness score. There is a small possibility of there may be multiple individuals with the highest fitness score in the tournament population. In such cases, we randomly select one of them. We repeat this process for every individual in the general population. We do not use the same tournament population for each individual which means that for each individual in the general population, we create a new tournament population. Therefore, the individuals included in the tournament population are not the same every time. The size of the tournament population is 5% of the general population since this amount is small enough to increase the variety and large enough to have better individuals after the crossover operation.

Algorithm 1 GABAP Selection Operation

Require: The tournament population, Pop

Ensure: an individual chromosome

```
1:  procedure SELECTION
2:      bestScore ← 0
3:      selectedInd ← Null
4:      for each Individual x in Pop do
5:          newScore ← fitness(x)
6:          if bestScore ≤ newScore then
7:              bestScore ← newScore
8:              selectedInd ← x
9:          end if
10:     end for
        return selectedInd
11: end procedure
```

4.3.5 Crossover Operation

Crossover operation is performed to create individuals of the next generation. Paired-up candidate solutions in the selection operation are used at this stage. This operation decides which genes are inherited from which parent. The value of *uniformRate* determines the probability of selecting genes from either parent. In our experiments, we choose *uniformRate = 0.5* not to favor either parent to improve diversity. Crossover of *Application Genes* is realized separately from the other two. For each application, a randomly selected parent's gene is inherited. For each monitoring point that is requested by the applications from the current batch, according to the generated random value, a parent is chosen and both *Sensor Gene* and *Base Station Gene* are taken from that parent. This operation guarantees that each sensor gene of the offspring covers the corresponding monitoring point and the base station gene of the corresponding monitoring point is connected to the sensor node since the offspring inherits these from either parent and initially we ensure that these constraints are satisfied as explained in Section 4.3.2. However, the offspring may violate the constraints related to sensing, communication, or processing capabilities and if that is the case, the offspring is punished in terms of fitness score as explained in Section 4.3.3. The operation is presented in Algorithm 2.

Algorithm 2 GABAP Crossover Operation

Require: Six chromosomes from two parents: A_1, S_1, BS_1, A_2, S_2, and BS_2

Ensure: The chromosomes of *Offspring*: A_{new}, S_{new} and BS_{new}

```
1:  procedure CROSSOVER
2:      for x = 1 to |A| do
3:          r ← Random(0, 1)
4:          if r ≤ uniformRate then
5:              A_new[x] ← A_1[x]
6:          else
7:              A_new[x] ← A_2[x]
8:          end if
9:      end for
10:     for y = 1 to |M| do
11:         r ← Random(0, 1)
12:         if r ≤ uniformRate then
13:             S_new[y] ← S_1[y]
14:             BS_new[y] ← BS_1[y]
15:         else
16:             S_new[y] ← S_2[y]
17:             BS_new[y] ← BS_2[y]
18:         end if
19:     end for
        return Offspring
20: end procedure
```

An example GABAP crossover operation is visualized in Figure 4.2. In this scenario, there are eight monitoring points in the area where the network covers and four applications arrive at the network. As it can be seen in the figure, crossover operation of sensor and base station genes are realized together. The crossover operation of application genes is done independently. In the example, sensor node and base station assignment of monitoring points #1, #3, #4, and #5 are inherited from the first parent, whereas the others are inherited from

the second parent. Similarly, the admission decision of application #0 is inherited from the second parent while the decision for the rest of the applications is inherited from the first parent.

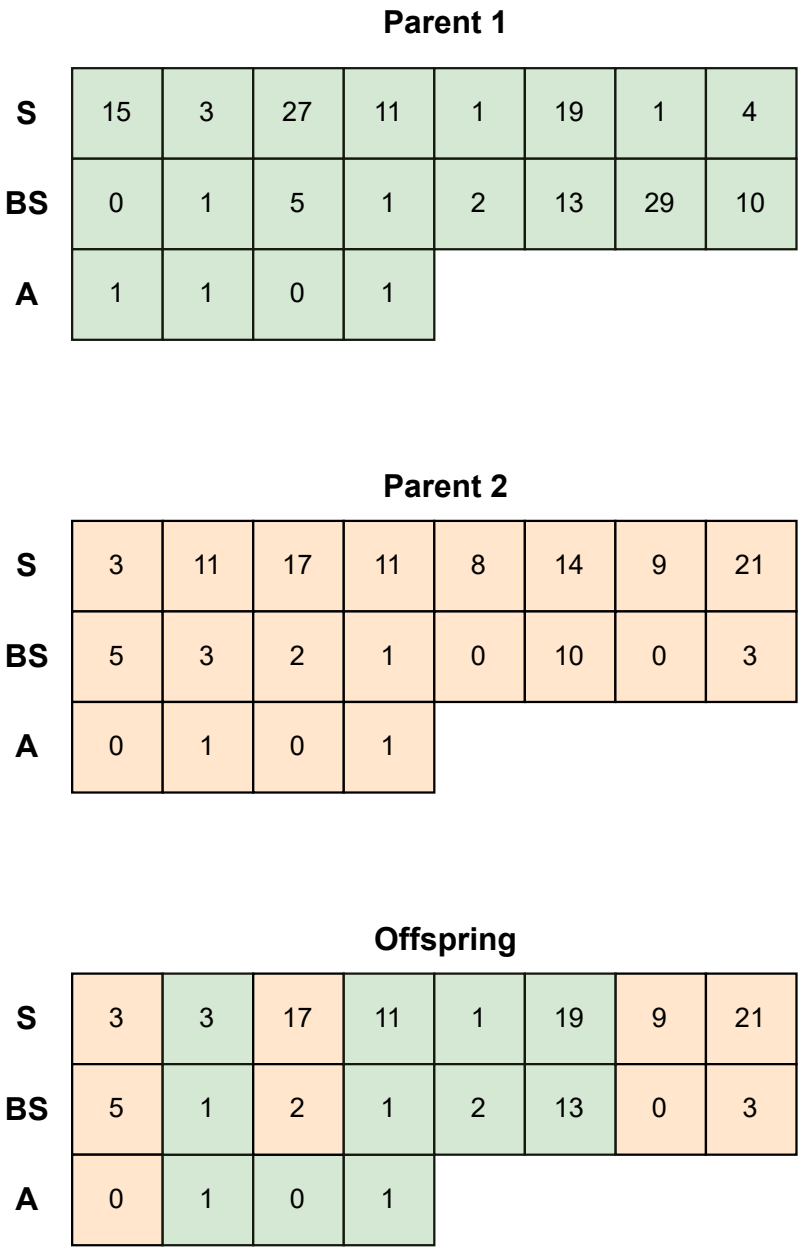


Figure 4.2: An example crossover operation in GABAP

4.3.6 Mutation Operation

Mutation operation is realized after individuals of the new generation are created. For each individual, we apply the mutation operation with the probability determined by the *mutationRate*. It generally has a small value. In our experiments, the value of *mutationRate* is 0.05 which means that there is a 5% chance for the mutation.

For each monitoring point requested by the applications from the current batch and selected for mutation, our mutation operation selects a random sensor node and a random base station, and changes the related genes accordingly. This operation is aware of the network structure. The selected sensor node can cover the monitoring point and there is an active connection between the selected sensor node and the selected base station. For a randomly selected application, our mutation operation flips the value of the corresponding gene of that application. Mutation on application genes and the mutation on sensor and base station genes are independent of each other.

The pseudocode of our mutation operation is presented in Algorithm 3. An example mutation operation is shown in Figure 4.3. Mutation operation of sensor and base station genes are done together again not to violate network constraints as in the figure. In the example, mutation operation changes the assigned sensor node and base station for monitoring points #1 and #7. Moreover, mutation operation changes the decision for admission of application #2.

Before Mutation

S	15	3	27	11	1	19	1	4
BS	0	1	5	1	2	13	29	10
A	1	1	0	1				

After Mutation

S	15	7	27	11	1	19	1	72
BS	0	3	5	1	2	13	29	4
A	1	1	1	1				

Figure 4.3: An example mutation operation in GABAP

Algorithm 3 GABAP Mutation Operation

Require: Three chromosomes of Individual *Ind*: A_{old}, S_{old} and BS_{old}

Ensure: Three mutated chromosomes of Individual *Ind*: A_{new}, S_{new} and BS_{new}

```
1: procedure MUTATION
2:     r1 ← Random(0, 1)
3:     if r1 ≤ mutationRate then
4:         a ← Random(0, |A|)
5:         A_new[a] ← 1 − A_old[a]
6:     end if
7:     for x = 1 to |M| do
8:         r2 ← Random(0, 1)
9:         if r2 ≤ mutationRate then
10:            i ← Random(0, |S|)
11:            l ← Random(0, |BS|)
12:            S_new[x] ← S_old[i]
13:            BS_new[x] ← BS_old[l]
14:        else
15:            S_new[x] ← S_old[i]
16:            BS_new[x] ← BS_old[l]
17:        end if
18:    end for
19: end procedure
```

4.3.7 The Genetic Algorithm

Our genetic algorithm, GABAP, is described in Algorithm 4. The termination condition for the algorithm is that either a candidate solution places all available applications, or no improvement is observed in the best individual's fitness score for 3 generations. The population size is 500. Elitism is enabled in the algorithm meaning that the best individual of a generation passes to the next one.

For each discrete time instant t (which may correspond to a time interval),

Algorithm 4 The Genetic Algorithm

```
1: procedure The GABAP generate a population 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:         end for
6:         for each Individual x in POP do
7:             invoke the SELECTION Operation, that is using tournament selection technique (Alg. 1) to select another individual to pair
8:         end for
9:         for each pair of parents do
10:            use CROSSOVER Operation to produce an offspring which is described in Alg. 2
11:        end for
12:        for each offspring do
13:            apply MUTATION Operation which is described in Alg. 3
14:        end for
15:        newBest ← the best individual among offsprings
16:        if f(newBest) > f(bestIndividual) then
17:            bestIndividual ← newBest
18:        end if
19:    end while
       return best individual
20: end procedure
```

we run our genetic algorithm. We feed the algorithm with applications arriving at time *t*. After our algorithm finds a solution, we apply this close-to-optimal solution to the network, and we continue with time instant *t+1*. For the sensor and the base station of each monitoring point we have three cases:

- At time instant *t*, if a monitoring point is not sensed yet, we set the assigned sensor and base station of that monitoring point according to the solution we have by running our algorithm at time *t*, and mark that monitoring point as *sensed*. We update the remaining resources of the corresponding sensor, base station, and the connection between them.

- At time instant *t*, if a monitoring point is already sensed and the solution we have by running our algorithm at time *t* suggests the same sensor and

base station, we update the required sense rates of the monitoring point and the remaining resources of the corresponding sensor, the base station and the connection between them.

- At time instant t, if a monitoring point is already sensed and the solution we have by running our algorithm at time t suggests a sensor and/or base station different from the monitoring point that is already assigned to, we *migrate* the monitoring point from the old sensor and base station to the new sensor and base station. The resources of two sensors, two base stations, and two connections are updated accordingly.

4.4 Greedy Algorithm

The greedy approach we propose is a modified version of the *Worst Fit* algorithm. Therefore, we use the least utilized sensor node or base station to distribute the load more evenly to have more options available for subsequent requests. At each time instant t, our algorithm processes arriving applications one by one in an unordered way. For each application, we check whether all of its monitoring point requirements are satisfiable. If they are, then we place that application into our network and update resources accordingly. The satisfiability of a requirement is checked as follows:

- If the required monitoring point is already sensed, then, we check whether the resources that the monitoring point uses can meet the extra demands of the new application. Resources mentioned here are the resources of the sensor that is already sensing that monitoring point, the base station at which that point's data is processed, and the connection between them. If the new demand can be met, then this requirement is satisfiable. Here, we do not provide any migration operation as it contradicts the approach of being greedy.

- If the required monitoring point is not sensed, then we apply the *Worst Fit* logic to find a sensor node and a base station to sense the monitoring point

and process the related data. We search through sensor nodes that have that point in their sensing range and select the sensor node with the largest available resource. Then, among the base stations that the sensor node has a connection to, we select the one with the largest available processing resource. If the connection between the selected sensor node and the base station has enough bandwidth, then we assign that monitoring point to the selected sensor node and base station. If the connection cannot meet the demand, we skip that base station and consider the *next* base station with the largest available resource. If none of the base stations connected to the selected sensor node satisfies the demand, then we skip this node and consider the *next* sensor node with the largest available resource. If our algorithm cannot find such a sensor node either, then we consider the requirement unsatisfiable.

4.5 Summary

In this chapter, we first defined the mathematical formulation of our application placement problem. Then, we showed that the problem is NP-hard by reducing the 3-SAT problem to it. We also described our proposed algorithms: GABAP and a greedy one. We first explained the chromosome structure of GABAP, then, we explained how the initial population is created. Moreover, we defined our fitness calculation which determines how an individual performs. Then, we explained how the selection, crossover, and mutation operations are designed. We concluded the section with the overall genetic algorithm. Lastly, we described how our greedy algorithm is designed.

Chapter 5

Application Scheduling in Wireless Sensor Networks

In this chapter[1], we define our application scheduling problem in wireless sensor networks. First, we mathematically describe the problem. Then, we explain our proposed algorithms to solve it.

5.1 Problem Statement

The network structure we consider in this problem is the same as the one described in Section 4.1. In this problem, our major goal is to minimize total run-time (makespan) of applications requiring the services of a WSN. This optimization problem can be formalized as follows:

$$tf_{max} \tag{5.1}$$

[1]This chapter is based on the work [113]; Mustafa Can Çavdar, Ibrahim Korpeoglu, and Özgür Ulusoy, 2022. "Application Scheduling with Multiplexed Sensing of Monitoring Points in Multi-purpose IoT Wireless Sensor Networks" arXiv: 210.06393, DOI: https://doi.org/10.48550/arXiv.2210.06393

Table 5.1: Parameters used in scheduling problem statement

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
r_{kt}	Sensing rate requirement of monitoring point k at time t
u_{kt}	sum of sensing rate requirements of applications for monitoring point k at time t

is minimized subject to

$$tf_{max} = max(tf_j) \qquad \forall j \in A \tag{5.2}$$

Some equations in this section are similar to the ones given in Section 4.1. Parameters in the equations are shown in both Tables 4.1 and 5.1. Eq. 5.2 shows how total run-time (makespan) is calculated. It is the finish time of the last application that used the network. We are assuming that the first application is admitted at time 0.

$$tf_j = t0_j + t_j \qquad \forall j \in A \tag{5.3}$$

$$x_{jt} = \begin{cases} 1, & t0_j < t \leq tf_j \\ 0, & otherwise \end{cases} \qquad \forall j \in A \tag{5.4}$$

Eq.5.3 describes how the finish time of each application is determined. Eq. 5.4 is used to determine in which time interval an application is deployed to the network.

$$r_{kt} = max(r_{jk} \times x_{jt}) \qquad \forall j \in A, \forall t \in T, \forall k \in M \tag{5.5}$$

$$u_{kt} = \sum_{j \in A} (r_{jk} \times x_{jt}) \qquad \forall k \in M, \forall t \in T \tag{5.6}$$

Eqs. 5.5 and 5.6 show the calculation of sensing requirements (i.e., required sensing rates) of monitoring points in shared and unshared cases, respectively.

$$\sum_{k \in M} x_{ik} r_{kt} \leq R_i \qquad \forall i \in S, \forall t \in T \tag{5.7}$$

$$\sum_{k \in M_{il}} r_{kt} TC \leq \sum_{m \in C} x_{ilm} C_m \quad \forall i \in S, \forall l \in B, \forall t \in T \tag{5.8}$$

$$\sum_{i \in S} \sum_{k \in M_{il}} u_{kt} PC \leq P_l, \qquad \forall l \in B, \forall t \in T \tag{5.9}$$

Eqs. 5.7, 5.8, and 5.9 are sensing, transmission and processing constraints for each time instant, respectively. 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; data amount transmitted per second cannot be more than the bandwidth of that connection; and the total processed data per second at one base station should be less than the base station's processing capacity. TC (transmission coefficient) is a constant used to map a sensing rate value to a communication rate requirement. Similarly, PC (processing coefficient) is a constant used to map a sensing rate value to a processing capacity requirement.

5.2 Hardness of Application Scheduling

The application scheduling problem explained above is a variation of the well-known task scheduling problem. Task scheduling problem is proven to be an NP-hard problem [112]. To prove that application scheduling is also an NP-hard problem, we reduce the multiway number partitioning problem to it.

5.2.1 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 [112].

5.2.2 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 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 the 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.

5.3 Genetic Algorithm Based Application Scheduling (GABAS)

5.3.1 Chromosome Structure

The chromosome structure of GABAS is very similar to GABAP with one distinction. *Application Genes* represent the admission order of the applications in GABAS instead of deciding whether the applications must be admitted. Therefore, it is an integer list with length of $|A|$ and the values of the elements in the list are from the interval $[0, |A| - 1]$. Each integer from the interval must appear once in application genes. As in GABAP, *Sensor Genes* and *Base Station Genes* represent which sensor node and base station will sense data from and process the gathered data from the monitoring points.

5.3.2 Initial Population Creation

Since *Application Genes* of an individual determines the admission order of the applications, we basically 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.

5.3.3 Fitness Calculation

The fitness calculation of GABAS is a very simple one. Equation 5.10 shows the fitness calculation of our algorithm.

$$fitness = -1 \times makespan \tag{5.10}$$

Basically, GABAS aims to reduce the makespan, which is the time instant when the last application finishes and leaves the network. Therefore, a smaller makespan means a better individual in the population. Since a higher fitness value means a better individual (solution), we multiply the makespan with -1 to calculate the fitness score, since a lower makespan value is more desirable one. Therefore, fitness scores of all individuals are negative.

Applications are admitted one by one according to their order in Application Genes. If an application cannot be admitted due to shortage of resources, we wait for some already admitted applications to finish, release the resources they use and leave the network. Then, there will be sufficient resources available for the waiting application. We do not try to admit any other application waiting in the queue.

5.3.4 Selection Operation

We use tournament selection for this process. Basically, for each individual i, we create a sub-population which consists of 5% of the whole population and select the individual with the highest fitness score in this sub-population. Then we pair the individual i with the selected individual. Our selection operation is presented in Algorithm 5.

Algorithm 5 GABAS Selection Operation

Require: The tournament population, $tPop$

Ensure: an individual chromosome

```
1: procedure SELECTION
2:     bestScore ← 0
3:     bestInd ← Null
4:     for each Individual x in tPop do
5:         newScore ← f(x)
6:         if bestScore ≤ newScore then
7:             bestScore ← newScore
8:             bestInd ← x
9:         end if
10:    end forreturn bestInd
11: end procedure
```

5.3.5 Crossover Operation

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 6. We set the chance of either parent to pass its genes to the offspring as 50%; therefore, we have *uniformRate* value in the algorithm as 0.5. Similar to GABAP, 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 with 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 which appear twice and the applications which do not appear at all in the offspring's genes. Then, we put applications which are missing into the places of the second appearances of the applications that are present twice in the genes.

An example crossover operation is visualized in Figure 5.1. Crossover of sensor and base station genes are the same as in GABAP. As in the figure, crossover of application genes may result in errors. In the example, applications #2 and #3 have the same rank for admission and this must be repaired. We correct this fault in the gene repair part.

Algorithm 6 GABAS 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: $r \leftarrow Random(0, 1)$
4: **if** $r \leq uniformRate$ **then**
5: $A_{new}[x] \leftarrow A_1[x]$
6: **else**
7: $A_{new}[x] \leftarrow A_2[x]$
8: **end if**
9: **end for**
10: **for** $x = 1$ *to* $|\boldsymbol{M}|$ **do**
11: $r \leftarrow Random(0, 1)$
12: **if** $r \leq uniformRate$ **then**
13: $S_{new}[x] \leftarrow S_1[x]$
14: $BS_{new}[x] \leftarrow BS_1[x]$
15: **else**
16: $S_{new}[x] \leftarrow S_2[x]$
17: $BS_{new}[x] \leftarrow BS_2[x]$
18: **end if**
19: **end for**
20: $Repair(A_{new})$
return A new individual with chromosomes: A_{new}, S_{new} and BS_{new}
21: **end procedure**

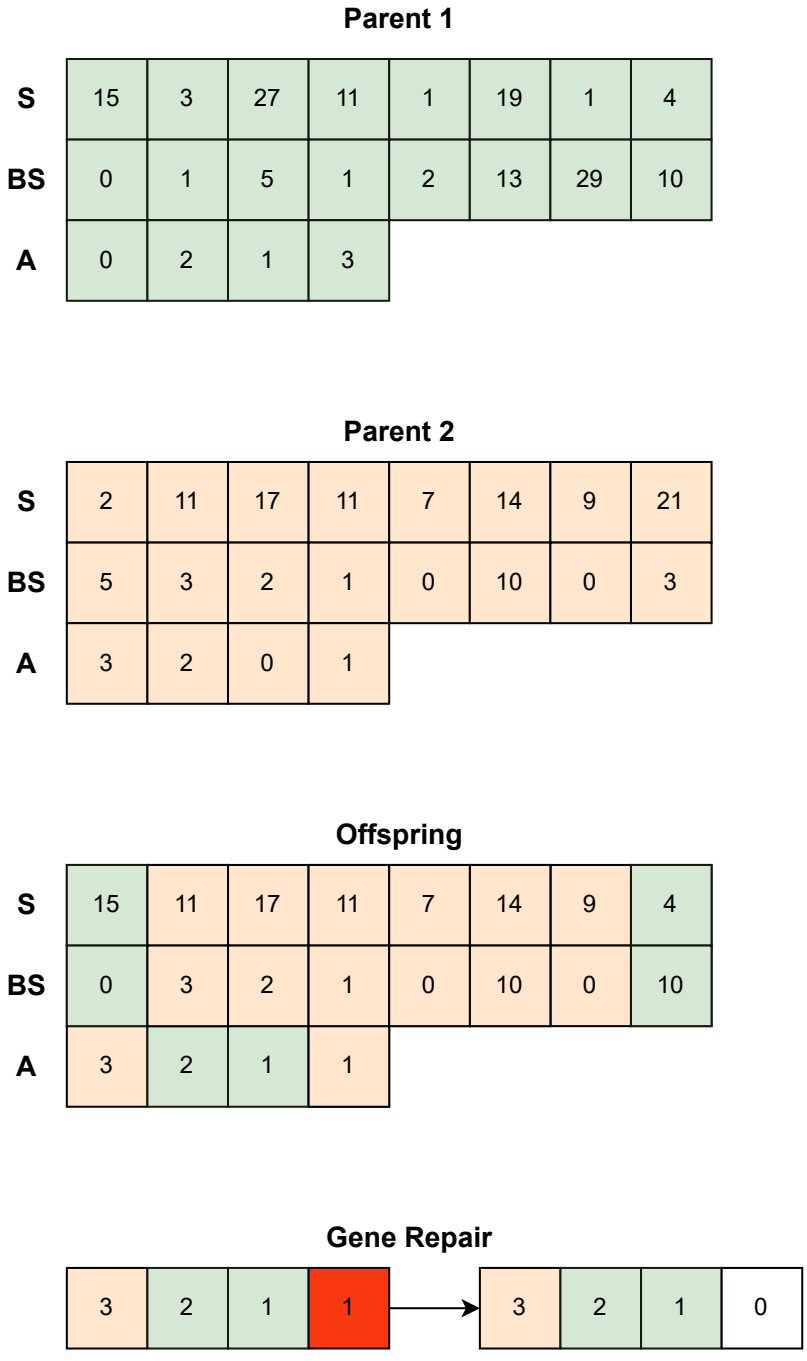


Figure 5.1: An example crossover operation in GABAS

5.3.6 Mutation Operation

The mutation operation is applied to all individuals. The operation is presented in Algorithm 7. As in the crossover operation, mutation operation in *Sensor Genes* and *Base Station Genes* are realized together to guarantee a solution that abides the network structure. This operation is executed for each gene with a probability of 5%.

For mutation in *Application Genes*, we swap the admission order of the two randomly selected applications with 5% mutation rate. An example mutation operation is shown in Figure 5.2. Mutations on sensor genes and base station genes are similar to the mutation in GABAP. In the example, order of applications #2 and #3 are swapped as a result of mutation.

Before Mutation

S	2	9	51	24	1	8	1	4
BS	5	0	12	1	9	12	29	10
A	3	2	0	1				

After Mutation

S	2	7	51	24	1	8	1	72
BS	5	3	12	1	9	12	29	4
A	3	2	1	0				

Figure 5.2: An example mutation operation in GABAS

Algorithm 7 GABAS Mutation Operation

Require: Three chromosomes: A_{old}, S_{old} and BS_{old}

Ensure: Three mutated chromosomes: A_{new}, S_{new} and BS_{new}

procedure MUTATION
  $A_{new} \leftarrow A_{old}$
  $r1 \leftarrow Random(0, 1)$
  if $r1 \leq mutationRate$ **then**
    $x \leftarrow Random(0, |M|)$
    $y \leftarrow Random(0, |M|)$
    $Swap(A_{new}[x], A_{new}[y])$
  end if
  for $x = 1$ *to* $|\boldsymbol{M}|$ **do**
    $r2 \leftarrow Random(0, 1)$
    if $r2 \leq mutationRate$ **then**
      $i \leftarrow Random(0, |S|)$
      $l \leftarrow Random(0, |BS|)$
      $S_{new}[x] \leftarrow S_{old}[i]$
      $BS_{new}[x] \leftarrow BS_{old}[l]$
    else
      $S_{new}[x] \leftarrow S_{old}[i]$
      $BS_{new}[x] \leftarrow BS_{old}[l]$
    end if
  end for
  return Mutated individual with chromosomes: A_{new}, S_{new} and BS_{new}
end procedure

5.3.7 The Genetic Algorithm

Our overall genetic algorithm is presented in Algorithm 8. 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 smaller finish time. Elitism is enabled in the algorithm, which means that the fittest individual of one generation is carried over to the next generation.

Algorithm 8 The Genetic Algorithm

```
1: procedure THE GABAS
       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:         end for
6:         for each Individual x in POP do
7:             Create a tournament population, tPop
8:             y = SelectionOperation(tPop)
9:         end for
10:        for each pair of parents, x and y do
11:            z = CrossoverOperation(x, y)
12:        end for
13:        for each offspring do
14:            z = MutationOperation(z)
15:        end for
16:        find the best individual among offsprings, newBest
17:        if newBest is better than current best individual then
18:            replace current best individual with newBest
19:        end if
20:    end while
       return best individual
21: end procedure
```

5.4 Greedy Algorithms

Other than GABAS, we propose three greedy algorithms to solve the application scheduling problem. They all use the Worst Fit approach described in Section 4.4 in assigning monitoring points to sensor nodes and base stations to use the less utilized resources among the available ones to provide load balancing. The only difference between the three algorithms is the ordering of the application admission queue.

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 complete. We do not preempt a running application.

5.4.1 Least Monitoring Point First (LMPF)

In the LMPF algorithm, we order the applications according to the number of monitoring points they require to be sensed. Among waiting applications, the ones with smaller number of monitoring points to be sensed are admitted earlier. Figure 5.3 shows an example ordering of applications by using LMPF.

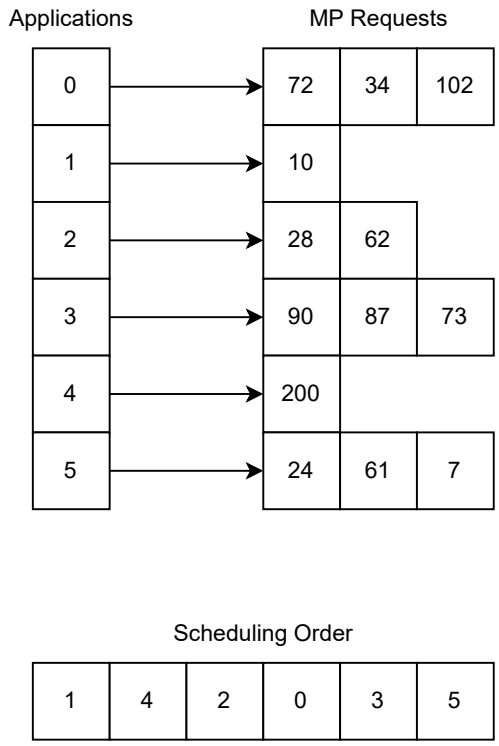


Figure 5.3: Example application scheduling order by LMPF

5.4.2 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. Figure 5.4 visualizes how applications are ordered with LMSF.

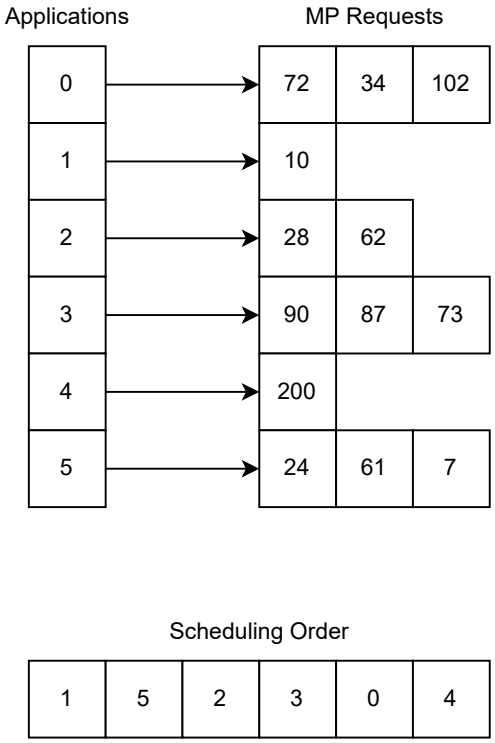


Figure 5.4: Example application scheduling order by LMSF

5.4.3 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. Figure 5.5 shows the determining of the scheduling order of the applications with LTSF.

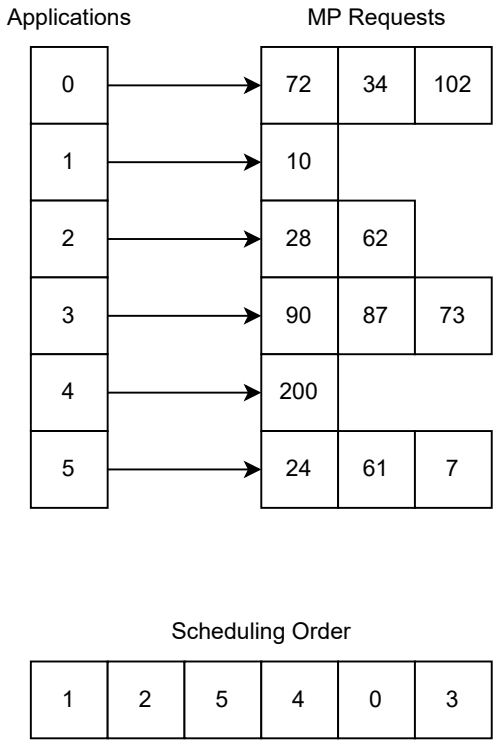


Figure 5.5: Example application scheduling order by LTSF

5.5 Summary

In this chapter, we first mathematically defined our application scheduling problem. Then, we showed that the problem is NP-hard by reducing the Multiway Number Partitioning problem to it. We also described our proposed algorithms GABAS, LMPF, LMSF and LTSF. We first explained how chromosome structure of GABAS is designed. Then, we continued with the way of creation of the initial population, and described the fitness calculation. Then, we explained the selection, crossover and mutation operations in a detailed way. We concluded the section with the overall genetic algorithm.

We also explained the logic of our three greedy algorithms and how they differ from each other in terms of ordering the applications that were yet to be deployed.

Chapter 6

Experimental Results

6.1 Experimental Setup

We did extensive simulation experiments to evaluate the proposed algorithms. In our simulations, we consider a model of a WSN spanning a 2D plane that has a 1000 m width and 1000 m height. Monitoring points, sensor nodes and base stations have their own coordinates (x and y). The coordination of these network elements is randomly determined in each simulation and our network creation method ensures that they do not overlap but the coordinates of any two elements can be very close to each other. Moreover, our network creation method guarantees that each monitoring point is within the sensing range of at least one sensor node and each sensor node can be connected to at least one base station.

In our network, we assume that applications arriving at the network may only require sensing rates in 3 major scales (rate types). Sense rate requirements of applications for each scale are randomly determined between the values shown in Table 6.1. We assume that from a single monitoring point only one data type can be requested.

Table 6.1: Sensing rate requirements.

Data Type	**sensing rate**
Data Type 0	5 kb/s - 20 kb/s
Data Type 1	15 kb/s - 40 kb/s
Data Type 2	25 kb/s - 60 kb/s

The quality of the connections in a WSN has a vital role in the reliable transmission of sensed data from sensor nodes to base stations. To simulate the effect of the link quality, we assign a Packet Delivery Ratio (PDR) to each sensor node base station connection in the network. PDR is one of the main metrics for modelling link quality [114]. For a particular link, PDR is calculated as the number of packets delivered to the receiving node divided by the number of packets sent by the transmitting node. In this way, we model the delivery probability of a packet to be equal to the PDR of the connection over which the packet is sent. An undelivered packet is retransmitted. That means, of course, more energy cost per packet. We limit the number of retransmissions for a packet to 10 [115]. We additionally assume that a packet is delivered after the 10th retransmission, if not delivered earlier. In our simulations, PDR of each connection is randomly determined between 0.7 and 1. We assume that the packet size is fixed and 512 bits.

Our network constraints are shown in Table 6.2. 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 algorithms that are compared. The number of applications is 1000 and the number of monitoring points is 300. The energy budget for each sensor node is 20000 J.

Table 6.2: Network parameters.

Parameter	Value
Monitoring Points per Application	1 - 3
Communication Range of Sensors	200 m - 250 m
Sensing Range of Sensor Nodes	30 m - 50 m
Sensing Rate Capacity of Sensor Nodes	400 kb/s
Bandwidth Capacity of Connections	100 kb/s
Processing Capacity of Base Stations	1000 kb/s
Energy Budget of Sensor Nodes	20000 J
Transmission Coefficient	0.7
Processing Coefficient	0.9

6.2 Application Placement

In application placement simulations, applications arrive in ten different batches. We assign a batch number to each application randomly. At each time instant t, one batch of applications arrives. When a batch of applications is requested to be placed, a selected subset of those applications is admitted, and the resources used by these admitted applications become unavailable for the next batches. Applications run in our network for a limited amount of time. When an application completes, it releases all resources it has used. The released resources become available for the applications in later batches. The computation time for each application is 12 h.

We compared our two proposed algorithms, GABAP, and the Greedy algorithm, with the algorithms proposed in [9, 31] which we call DH and RSVN, respectively. The four algorithms are run with both shared and unshared approaches. In the plots, for each algorithm, results of shared and unshared approaches are presented with the suffix "S" and "U", respectively. We experiment with various numbers of applications, monitoring points, sensors, and base stations. Evaluation of the methods is reported in terms of the number of placed applications and the average energy spent per placed application. Our experiments are realized in 7 scenarios:

- *Scenario P1 (Application Count)*: The number of monitoring points is fixed

at 300. The application count is started at 500 and is incremented by 200 until 1500.

- *Scenario P2 (Monitoring Point Count)*: The number of applications is fixed at 1000. The monitoring point count is started at 50 and is incremented by 50 until 250.

- *Scenario P3 (Monitoring Point per Application)*: The number of applications is fixed at 1000, and the number of monitoring points is fixed at 300. For the number of monitoring point requests per application, instead of randomly determining between the values shown in Table 6.2, we start with 1 monitoring point per application and increment it by 1 until 6.

- *Scenario P4 (Communication Range)*: The number of applications is fixed at 1000, and the number of monitoring points is fixed at 300. For the communication range of sensors, instead of randomly determining between the values shown in Table 6.2, we start with 50 m as the communication range for each sensor node and increment it by 50 m until 250 m.

- *Scenario P5 (Sensing Range)*: The number of applications is fixed at 1000, and the number of monitoring points is fixed at 300. For the sensing range of sensor nodes, instead of randomly determining between the values shown in Table 6.2, we start with a 30 m sensing range for each sensor node and increment it by 5 until 50 m.

- *Scenario P6 (Batch Size)*: The number of monitoring points is fixed at 300, and the batch count is fixed at 10. We experiment with various batch sizes: 50, 100, 200, 250, and 500. Application count is calculated as the number of batches times the batch size.

- *Scenario P7 (Batch Count)*: Same constraints as in Scenario P6 except that application count is fixed at 1000. The batch count is calculated as the number of applications divided by the batch size.

In Scenario P1, we investigate how the change in the number of arriving applications affects the performance of the algorithms. Figures 6.1 and 6.2 show

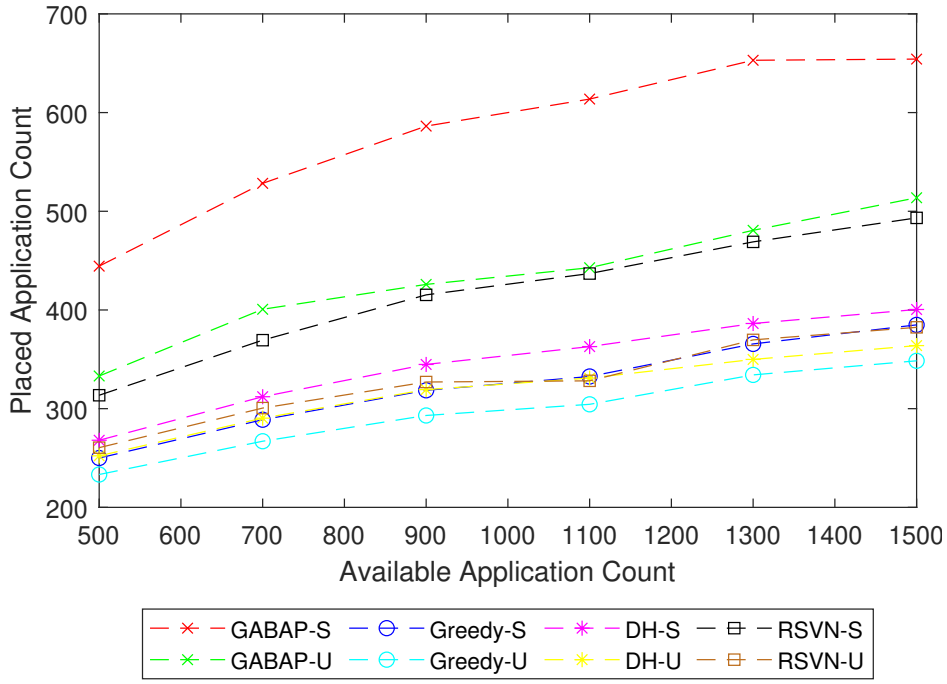


Figure 6.1: Comparison of algorithms in terms of placed application count in Scenario P1.

the average count of placed applications and average energy cost per placed application, respectively averaged over 1000 runs. Each algorithm performs better with the shareable approach. Obviously, as the number of applications arriving to the network gets larger, all algorithms achieve to place a larger number of applications. GABAP-S is clearly superior to all other algorithms. RSVN-S and GABAP-U have the next-best performance. DH and Greedy come last, where DH has a slightly better performance than Greedy.

In terms of energy spent, all algorithms perform better with the shareable approach. In fact, the average energy spent with the shareable approach is around half of the energy spent with the unshareable approach. As expected, we see that with a larger number of available applications, the shareable approach costs less energy per placed application since the sensing and transmission overheads are far less (even zero in some cases) than they are in the unshareable approach. However, we also observe a small reduction in the results of the unshareable approach with all algorithms. The reason for this result is that the possibility of the arriving applications with less data sense requirements increases with the

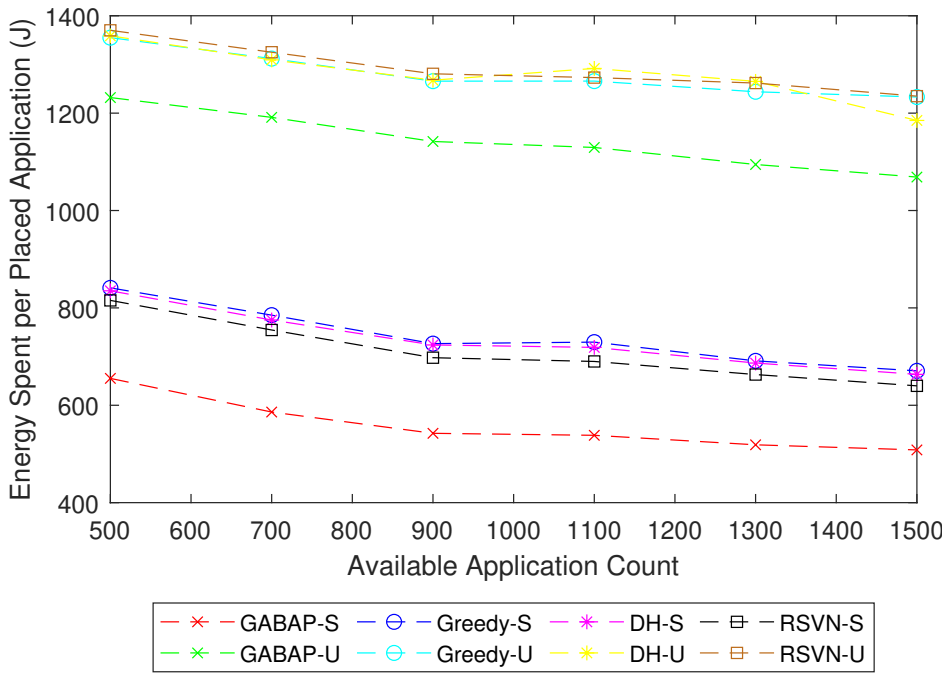


Figure 6.2: Comparison of algorithms in terms of energy cost in Scenario P1.

larger number of available applications. Therefore, the average energy spent decreases slightly. With the unshareable approach, all four algorithms perform similarly when GABAP-U has slightly better performance. With the shareable approach, GABAP-S has the best performance followed by RSVN-S.

In Scenario P2, we evaluate the impact of the number of monitoring points on the performance of the algorithms. Figures 6.3 and 6.4 show the results for this scenario. We expect that with less number of monitoring points, placing applications becomes harder because there will be more application requests for each monitoring point as the number of applications is fixed. Moreover, fewer monitoring points affect the performance of the algorithms with the unshareable approach more since the sensing and transmission overheads increase drastically. Figure 6.3 shows that the performance of the algorithms improves clearly with the increasing number of monitoring points. With the shareable approach, the improvement is not that visible since the sensing and transmission overheads are much smaller. GABAP-S still has the best performance; the gap between its performance and the others increases as the number of monitoring points decreases.

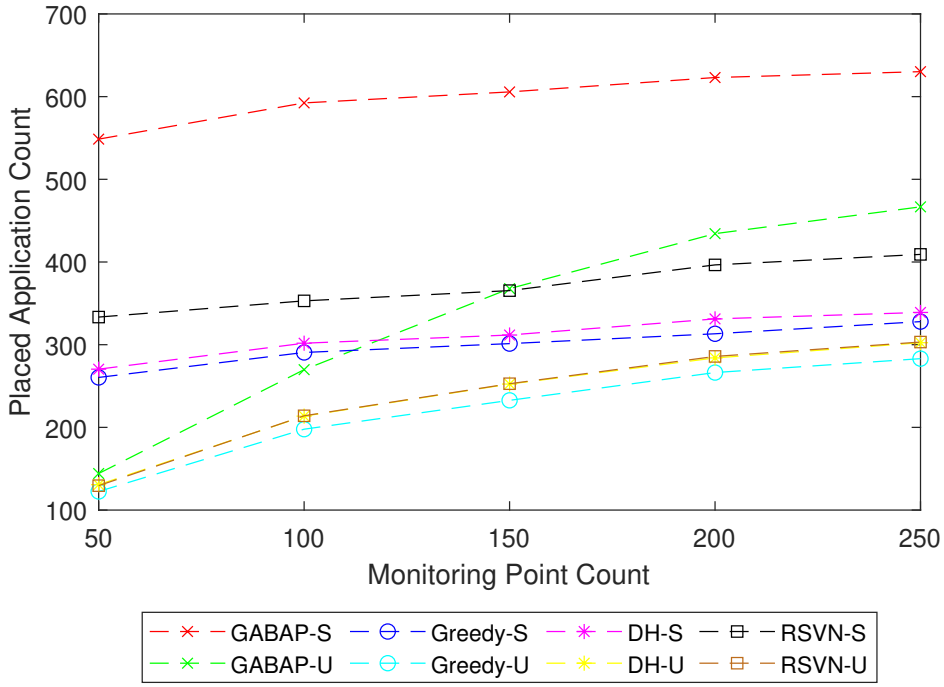


Figure 6.3: Comparison of algorithms in terms of placed application count in Scenario P2.

We again observe that with the shareable approach, the energy cost is much less than the unshareable one. With the increasing number of monitoring points, the energy cost is also increasing since there are more data to sense, transmit, and process. With the unshareable approach, all four algorithms show similar performance in terms of energy cost. With the shareable one, GABAP-S produces the lowest energy cost.

In Scenario P3, we have a fixed number of applications and a fixed number of monitoring points. We experimented with different numbers of monitoring points each application requires to be sensed, instead of randomly determined values according to Table 6.2. Figures 6.5 and 6.6 show the results of Scenario P3. With the increasing number of monitoring point requests per application, it is getting harder to place applications since the demand for a single application increases while network resources remain the same. We observe that the shareable approach makes algorithms place more applications compared to the unshareable one. Again, GABAP has the best performance, RSVN comes second, while Greedy and DH have the least number of applications placed. DH performs

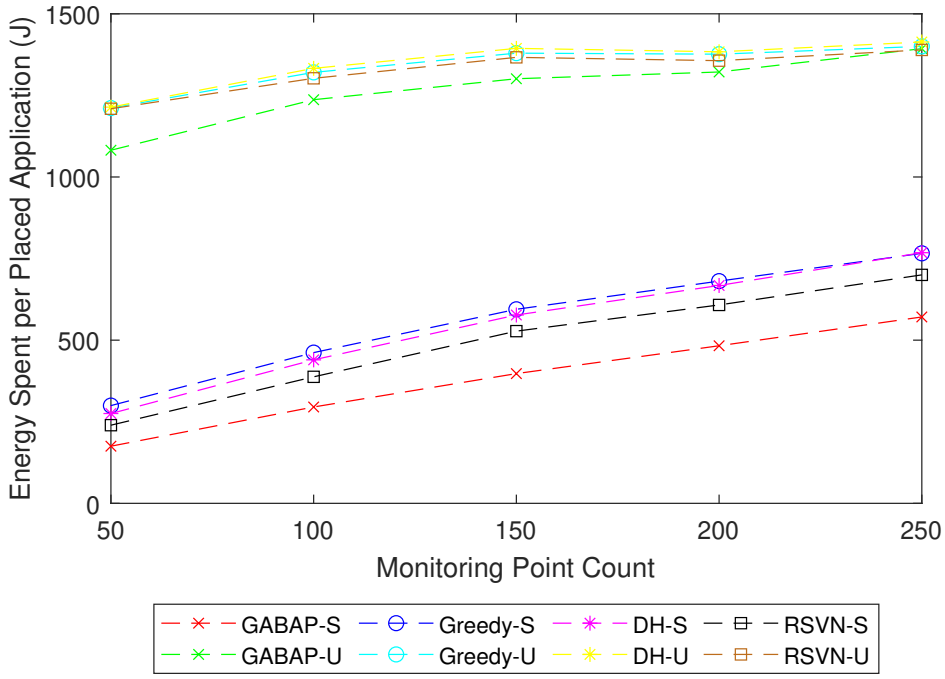


Figure 6.4: Comparison of algorithms in terms of energy cost in Scenario P2.

slightly better than Greedy.

In terms of energy cost, it is expected that the cost per placed application increases when the applications require more monitoring points to be sensed. Since the sensing and transmission overheads are more with the unshareable approach, the energy cost is more compared to the shareable approach. The performance gap between the shareable and unshareable approaches is proportional to the number of requests per application. GABAP-S has the best performance, and again, RSVN-S is the second-best. Energy cost results of the algorithms do not differ much with the unshareable approach; however with the shareable one, especially with the higher number of requests, the performance difference between GABAP-S and others becomes more visible.

Scenario P4 is investigating how the communication range of sensor nodes affects the performance of the algorithms. Instead of the 200 m communication range as in Table 6.2, we experimented with the communication ranges between 50 to 250 m. Figures 6.7 and 6.8 present the results. We observe that with a longer communication range for sensor nodes, the number of placed applications

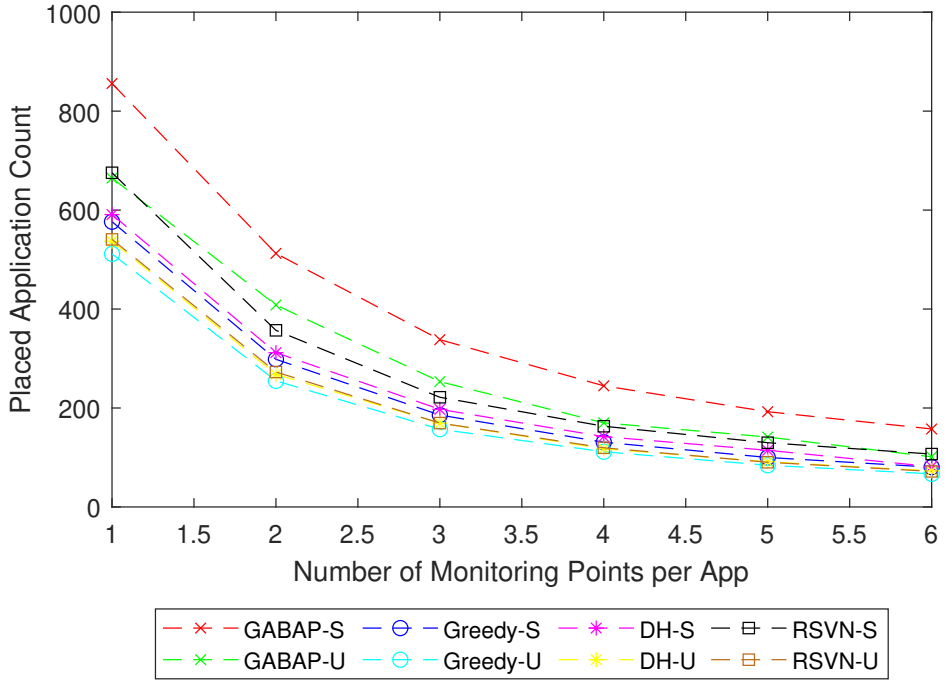


Figure 6.5: Comparison of algorithms in terms of placed application count in Scenario P3.

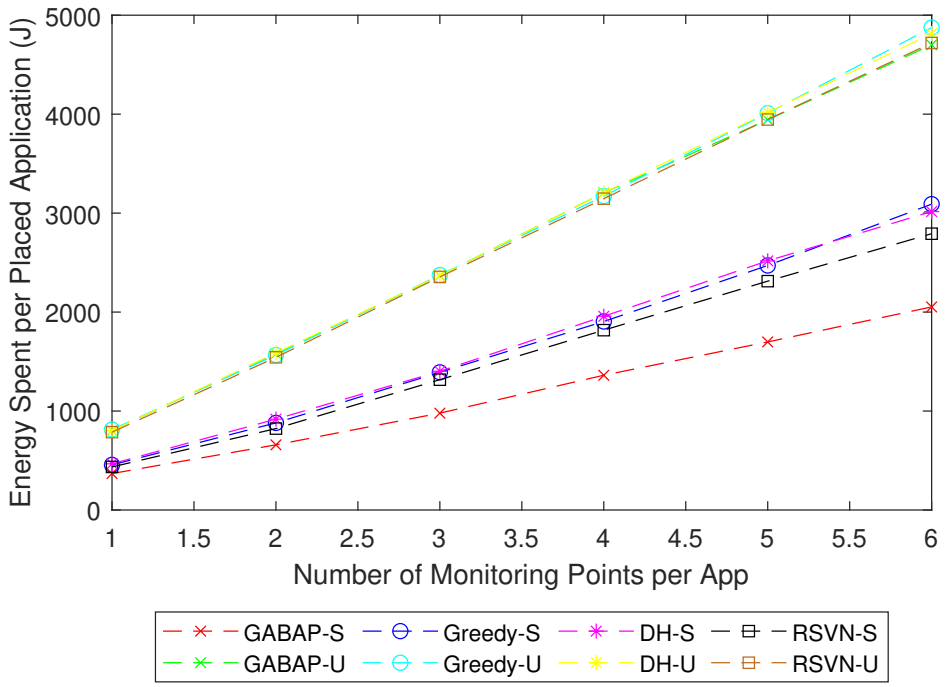


Figure 6.6: Comparison of algorithms in terms of energy cost in Scenario P3.

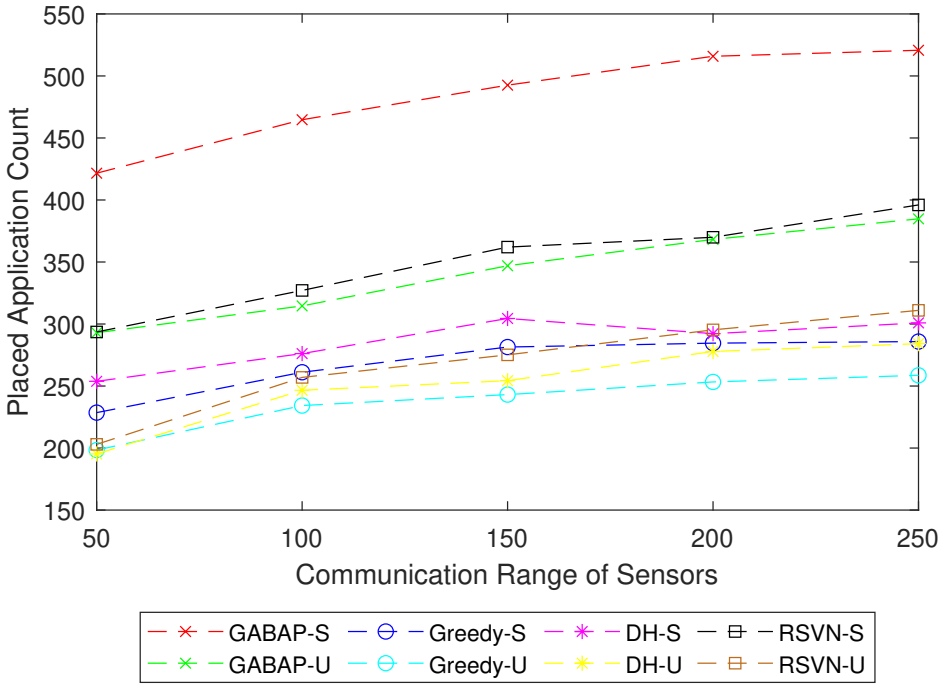


Figure 6.7: Comparison of algorithms in terms of placed application count in Scenario P4.

increases slightly. Similar to the previous scenarios, GABAP-S is superior among all compared. RSVN-S and GABAP-U have the next-best performances.

The energy spent increases with the broader communication range of sensor nodes. The reason for this result is that with a longer range, sensor nodes can connect to base stations that are more distant, and sending data to more distant base stations costs more compared to sending to closer base stations as shown in Eq. 4.12. The gap between the shareable and unshareable approaches in terms of energy cost increases with the longer communication ranges. This also applies to the gap among the results of the algorithms. GABAP-S has the least energy cost per application which is followed by RSVN-S. DH-S is slightly better than Greedy-S. With the unshareable approach, the energy cost is similar among all compared algorithms.

The experiments corresponding to Scenario P5 are conducted to investigate the performance with various sensing ranges of sensor nodes. Figures 6.9 and 6.10 show the placed application count and the average energy cost, respectively,

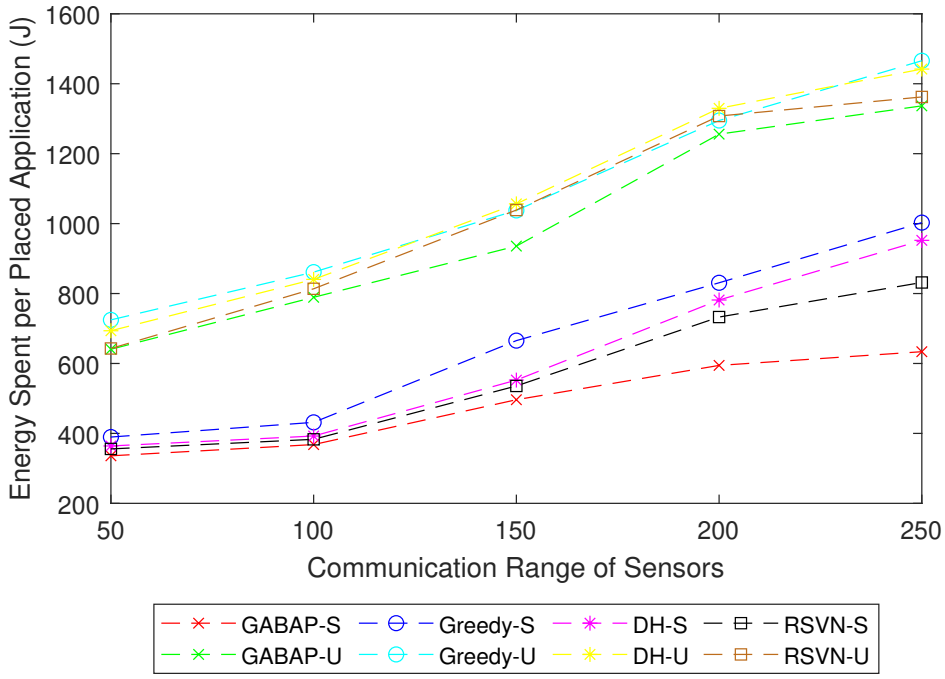


Figure 6.8: Comparison of algorithms in terms of energy cost in Scenario P4.

for Scenario P5. The sensing range of the sensor nodes does not affect the results too much. Placed application count increases slightly when the sensing range increases. Energy cost does not change much since the sensing range is not involved in energy calculations in our model directly as the communication range. The performance in terms of both the placed application counts and the energy cost is similar to the previous four scenarios. GABAP-S is the best for both metrics, GABAP-U and RSVN-S have the next-best performance. In terms of energy cost, the shareable approach consumes far less energy than the unshareable one.

In Scenario P6, we investigate the impact of batch size on the performance of the algorithms. The total application count is not limited, and since we have 10 batches, in this scenario, it is equal to $10 \times BatchSize$. Figures 6.11 and 6.12 present the results of Scenario P6. Since we do not limit the total application count, the results are similar to Scenario P1, where the effect of the number of applications on performance is measured.

Since we do not limit the number of applications in Scenario P6, the application

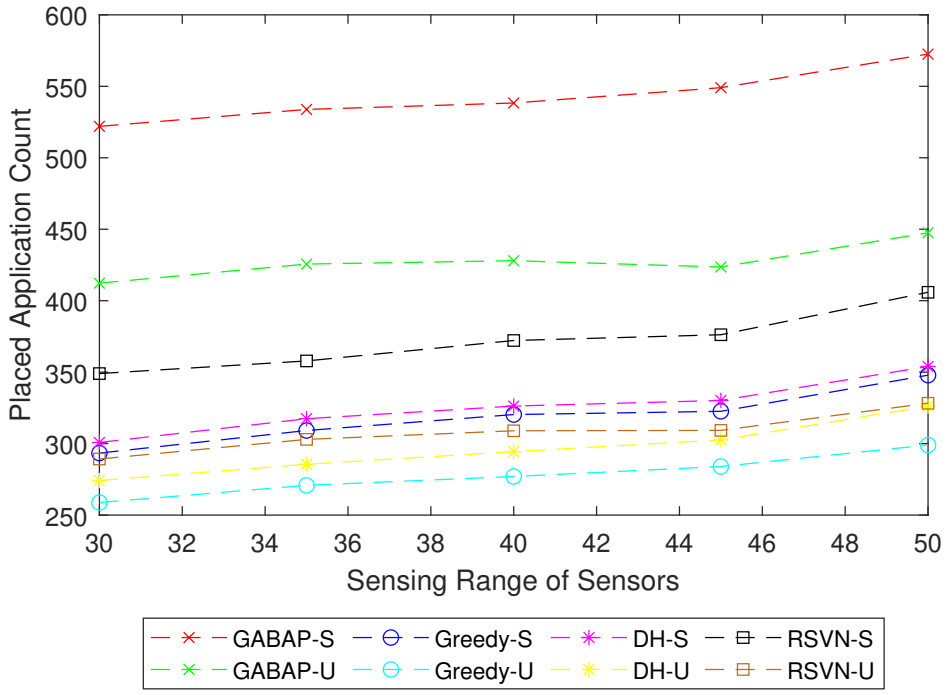


Figure 6.9: Comparison of algorithms in terms of placed application count in Scenario P5.

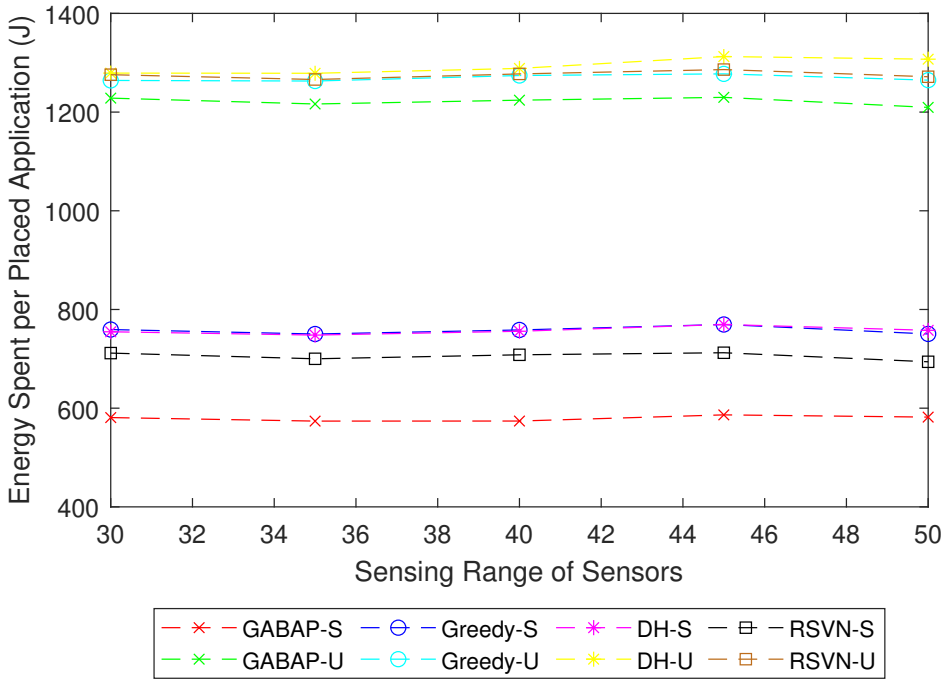


Figure 6.10: Comparison of algorithms in terms of energy cost in Scenario P5.

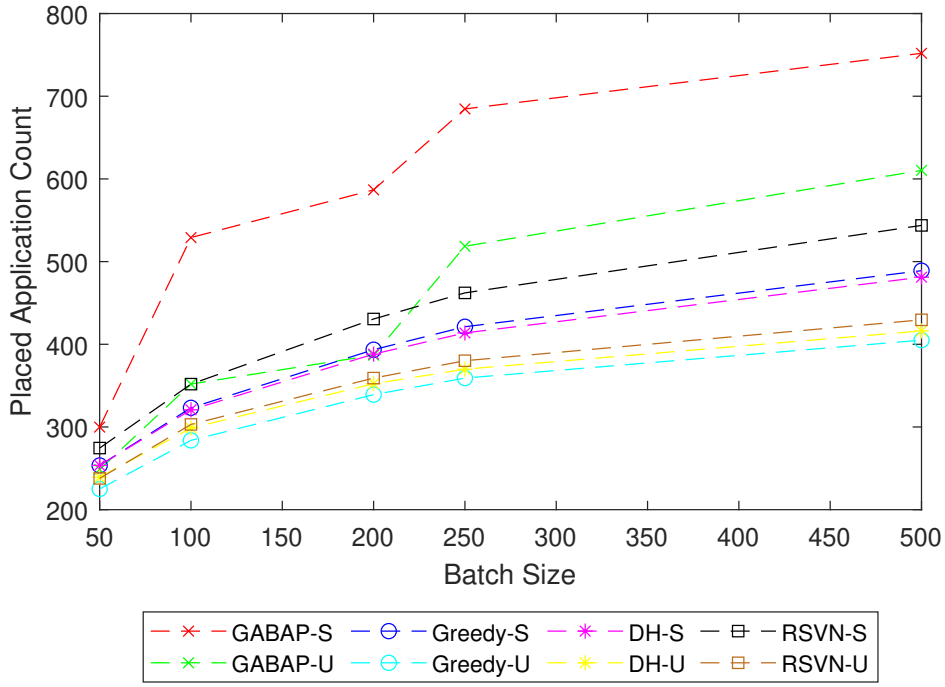


Figure 6.11: Comparison of algorithms in terms of placed application count in Scenario P6.

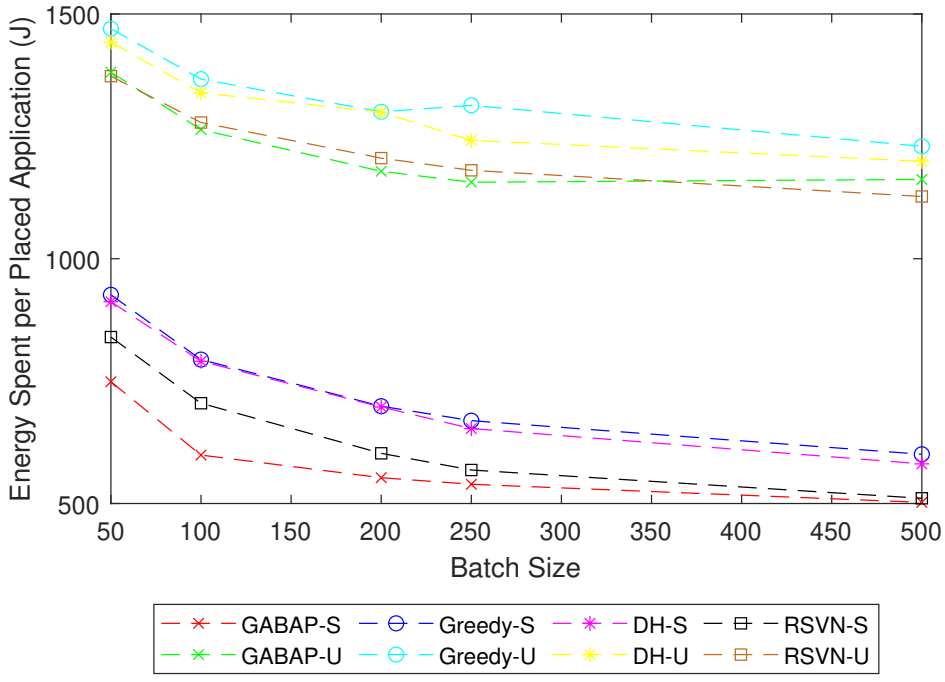


Figure 6.12: Comparison of algorithms in terms of energy cost in Scenario P6.

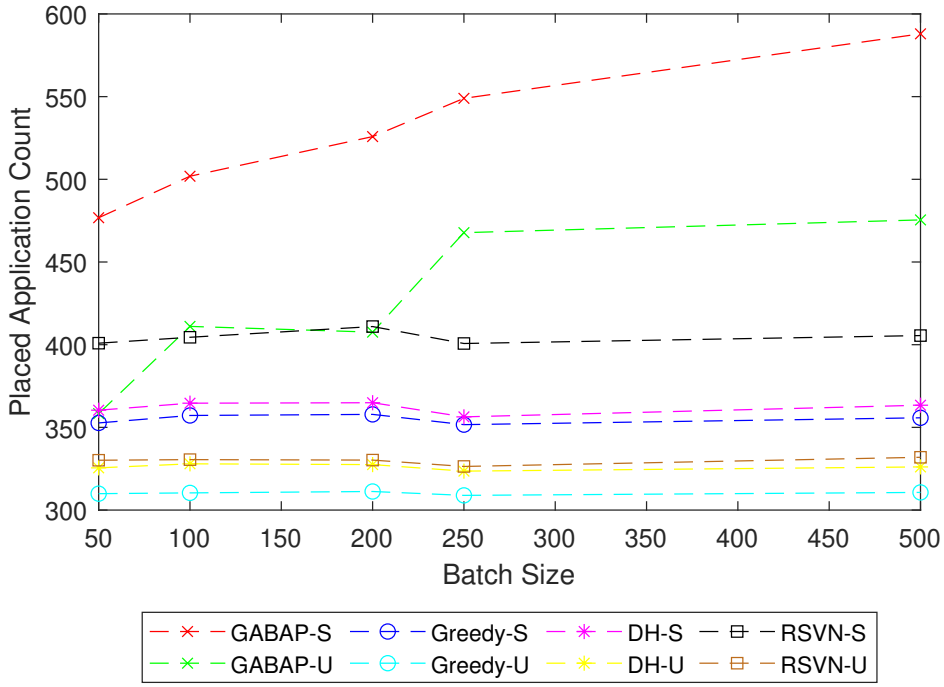


Figure 6.13: Comparison of algorithms in terms of placed application count in Scenario P7.

count may affect the results we gathered from the experiments. Therefore, to observe the effect of the batch size on the algorithms more clearly, in Scenario P7, the number of applications is set to 1000. Thus, the total batch count changes for each batch size. Figures 6.13 and 6.14 present the results for Scenario P7. Because we investigated the impact of the batch size in Scenario P7, only the performance of GABAP is affected since the others admit applications one by one when GABAP selects a subset of arriving applications at each batch. With larger batch sizes, GABAP performs better.

Energy cost results are similar to the previous scenarios. The shareable approach is better compared to the unshareable one. GABAP performs the best and RSVN follows it with both shareable and unshareable approaches.

We can infer the following conclusions from our results of application placement experiments:

- In all scenarios, the shareable approach has much better performance than

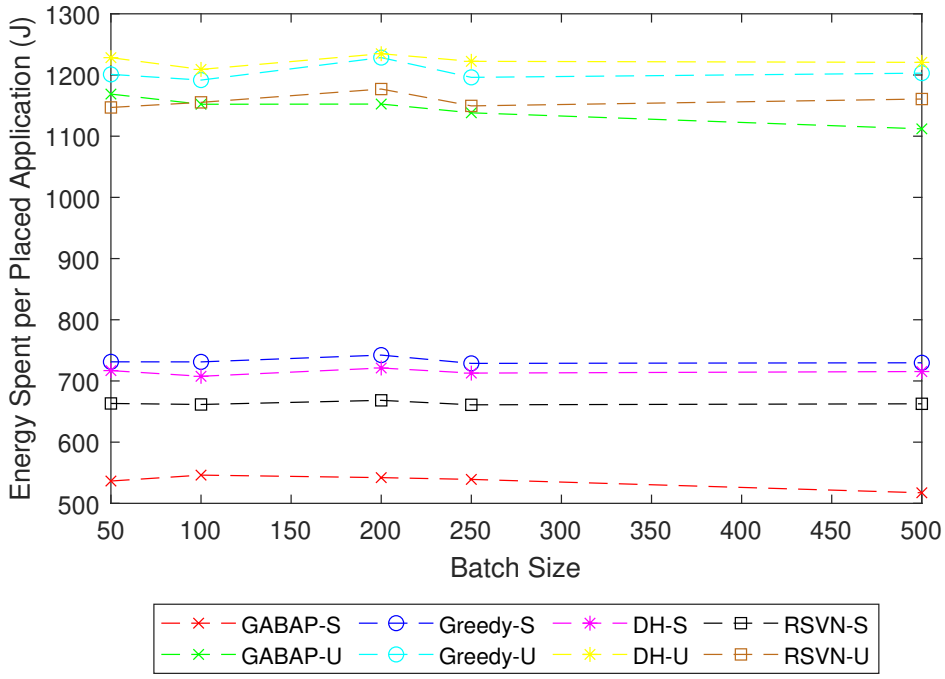


Figure 6.14: Comparison of algorithms in terms of energy cost in Scenario P7.

the unshareable one both in terms of placed application and the energy cost per application.

- GABAP is clearly superior to all compared algorithms. RSVN has the second-best performance. DH is slightly better than the Greedy algorithm.
- The performance difference between the shareable and unshareable approaches is greater when network resources are more limited with a smaller number of applications. With an application count that is large enough, the performance gap also increases with plenty of network resources.
- The performance difference between the shareable and unshareable approaches is larger in GABAP's results compared to other algorithms.
- Since GABAP selects a subset of applications among applications that arrive at the network at each batch while the other three try to admit applications one by one, GABAP can place applications much more optimally.
- DH and Greedy have very similar performance in all scenarios since their logic to place applications is similar. In some scenarios, DH performs

slightly better than Greedy, since it allows migrations while Greedy does not.

6.2.1 Comparison with Linear Programming

To investigate how close our algorithm performs to the optimal solution, we compare the results of our algorithm with the optimal results of a linear programming model. We use the Java API of IBM ILOG Cplex optimizer to code our linear program.

We use the constraints described in Section 4.1 to build our linear programming model. Some constraints there, however, are not linear. Therefore, we have to linearize those constraints.

Table 6.3: Comparison of GABAP with linear programming

Application Count	Mon. Point Count	Sensor Count	Base Station Count	Online GABAP		Offline GABAP		Linear Prog.	
				Result	Time (ms)	Result	Time (ms)	Result	Time (ms)
30	20	30	3	1.23	27	1.54	21	1.54	3,320
30	20	30	4	1.57	36	1.72	19	1.72	3,397
30	20	30	5	2.01	42	2.17	27	2.17	4,245
50	20	30	4	2.60	51	2.92	38	2.92	3,973
50	40	50	7	3.14	121	3.97	93	3.97	23,806
50	100	50	7	3.71	198	4.60	99	4.60	53,749
100	100	25	25	2.30	240	3.17	199	3.21	65,651
20	150	100	30	4.02	1,007	4.02	291	4.14	914,380
30	150	100	30	4.74	1,092	4.98	426	5.20	955,542
50	150	100	30	8.12	944	8.12	487	8.26	964,397

Linearization of maximum function is realized as follows. Let,

$$C = max(a_1, ...a_n) \tag{6.1}$$

Then, we transform this maximum function into a linear form with the following:

$$C \geq a_i, \qquad \forall i \in N \tag{6.2}$$

$$C \leq a_i + (1 - b_i) \times M \quad \forall i \in N \tag{6.3}$$

$$\sum_{i \in N} b_i = 1, \tag{6.4}$$

where b_i is a binary variable that indicates the maximum value, x_i. Therefore, if $b_i = 1$, then x_i is the maximum of all. M is a very large number.

Linearization of the product of a binary variable and a continuous variable is done as follows. Let,

$$z = A \times x, \tag{6.5}$$

where A is a continuous variable and x is a binary variable. Moreover, let $\bar{\text{A}}$ be the upper bound of A. Then, transformation into linear form is done as follows:

$$z \leq \bar{A} \times x \tag{6.6}$$

$$z \leq A \tag{6.7}$$

$$z \geq A - (1 - x)\bar{A} \tag{6.8}$$

$$z \geq 0 \tag{6.9}$$

In the equations above, if x is equal to zero, then Eqs. 6.6 and 6.9 ensure that z is also equal to zero. If x is equal to one, Eqs. 6.7 and 6.8 state that z is equal to A. This transformation is a variation of the general product linearization where the lower bound of the continuous variable is zero. In our problem statement, the

continuous variable is application demands which cannot be negative. Therefore, this transformation is applicable to our work.

Because the computation time required by a linear programming solver is too long, we use small values for the parameters of our network. We experiment with the shareable approach only. Sensing rate requirements of various data-rate types and network constraints are shown in Tables 6.1 and 6.2, respectively. The results are presented in Table 6.3. Both the placed application count and running-time results given in the table are the averages of 100 runs. At each run, we input the same network to all algorithms. The results show that our GABAP algorithm has a very good performance. For each parameter set, GABAP has a very close performance to the optimal results obtained from linear programming. Besides, the running time of our algorithm is much less than the running time of linear programming. Time values are also the average of 100 runs. The gap between the running times of GABAP and linear programming solution becomes larger as the number of network elements increases.

6.3 Application Scheduling

In our scheduling simulations, applications arrive at the network in 25 batches. 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 S1 (Application Count)*: The number of applications starts at 500 and is incremented by 100 until 1500.
- *Scenario S2 (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 S3 (Monitoring Point Count per Application)*: The number of

monitoring points requested per application is initially 1. It is incremented by 1 until 7.

- *Scenario S4 (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 S5 (Sensing Range)*: Similar to Scenario S4, each sensor node starts with a 30 m sensing range, and the sensing range is incremented by 5 m until 50 m.

- *Scenario S6 (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.

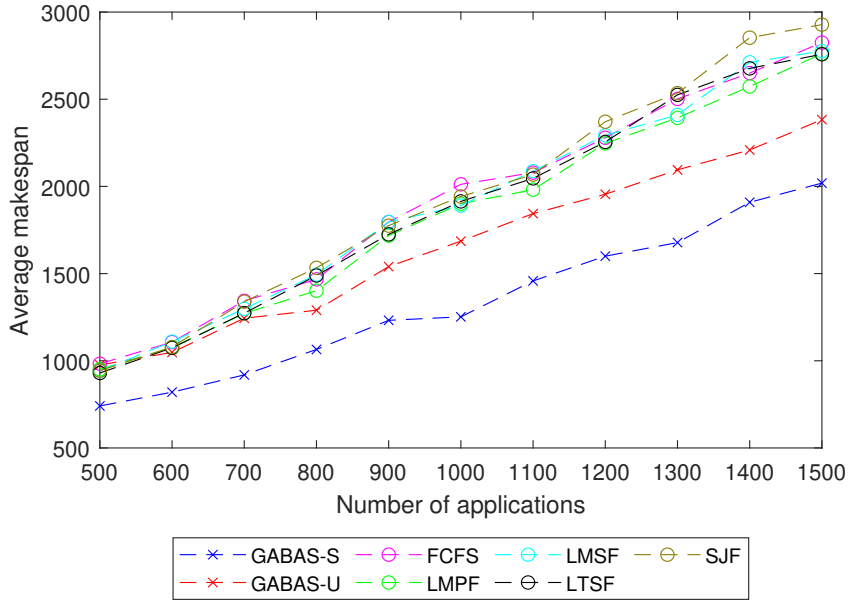
- *Average Turnaround Time*: The turnaround time of an application is the time between its arrival time to the network and its finish time. We report the turnaround time averaged over all applications admitted.

- *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.

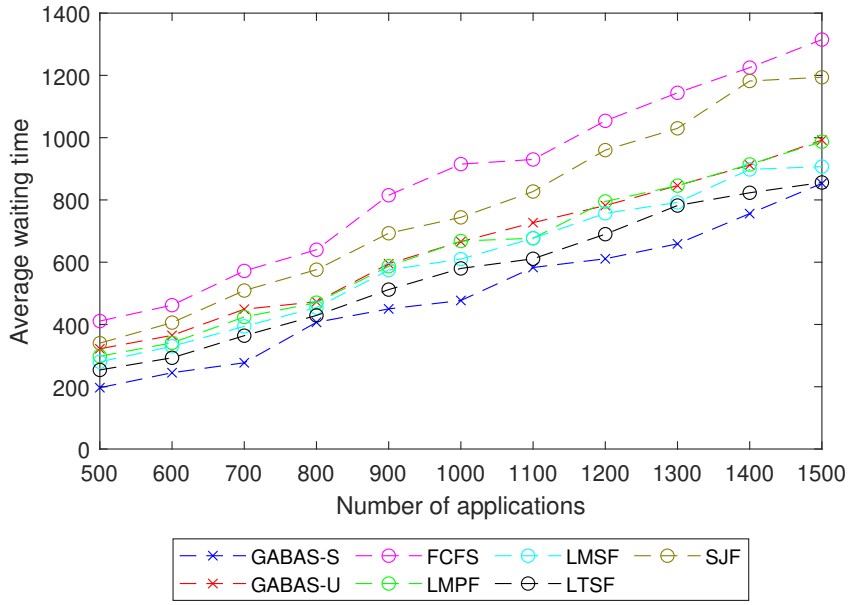
We compare our proposed algorithms with well-known standard task scheduling algorithms First Come First Served (FCFS) and Shortest Job First (SJF). For FCFS and SJF algorithms, we also used the *Worst Fit* approach to determine which sensor node and the base station are assigned to each monitoring point. In the figures provided below, both shared (GABAS-S) and unshared (GABAS-U) approaches for GABAS are reported. For the 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 S1, we investigate how the number of applications affects the performance of the algorithms. Figure 6.15a 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 6.15b, 6.16a and 6.16b 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; however, in terms of average successful execution rate GABAS-U beats LMPF.

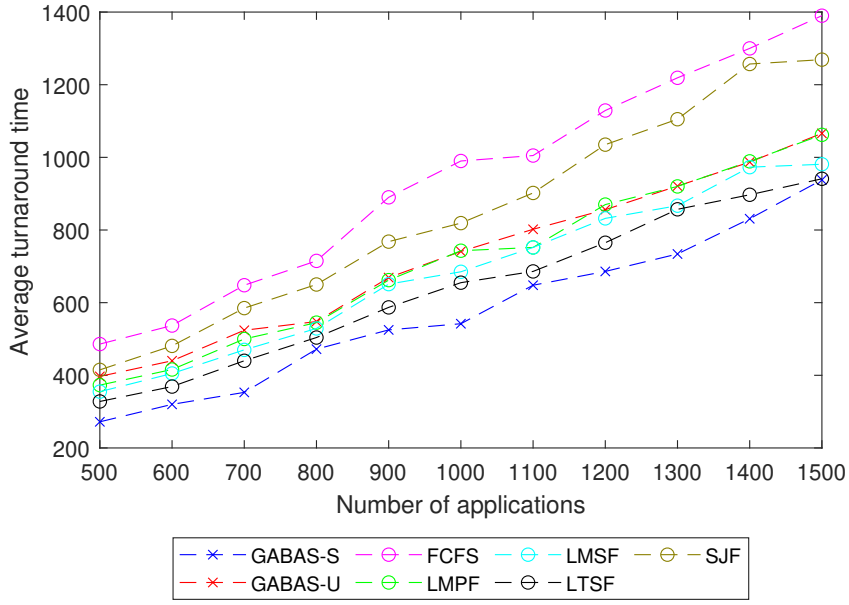


(a)

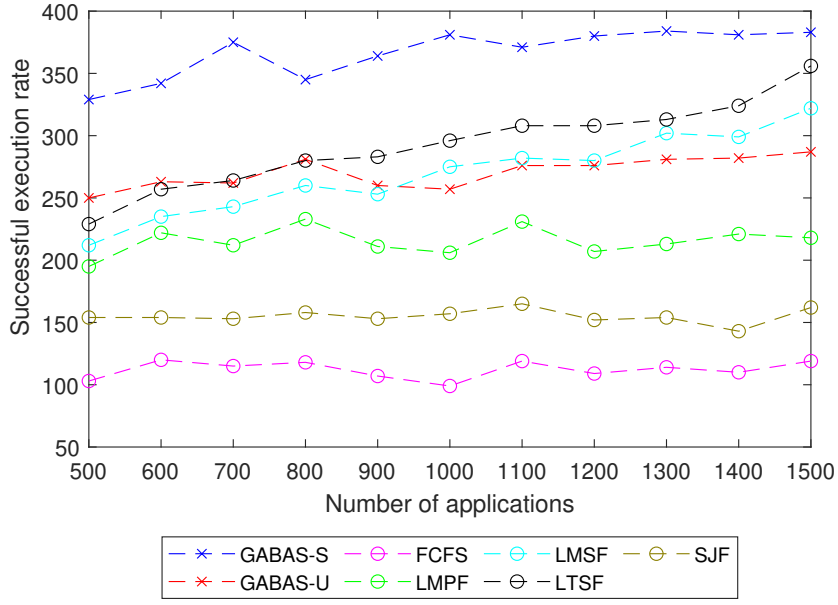


(b)

Figure 6.15: Comparison of algorithms in terms of (a) average makespan and (b) average waiting time in Scenario S1.



(a)

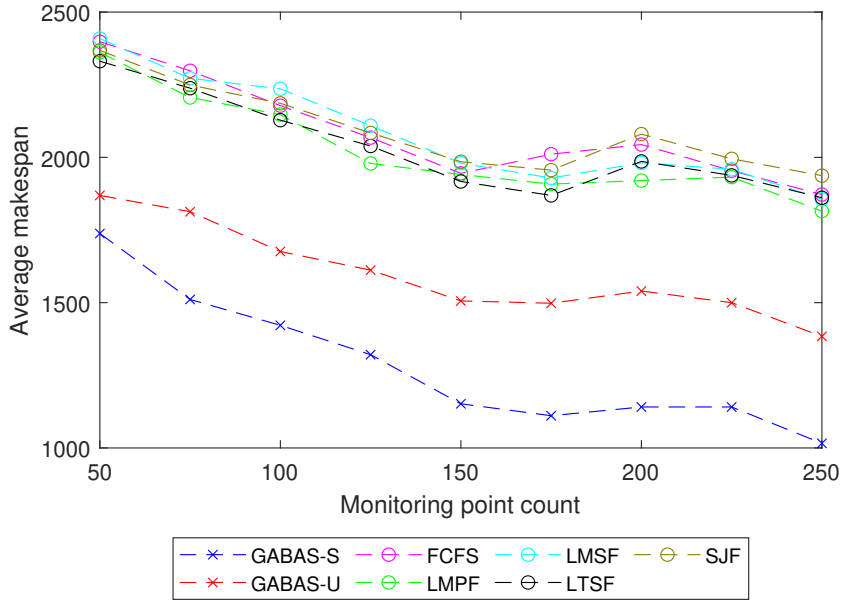


(b)

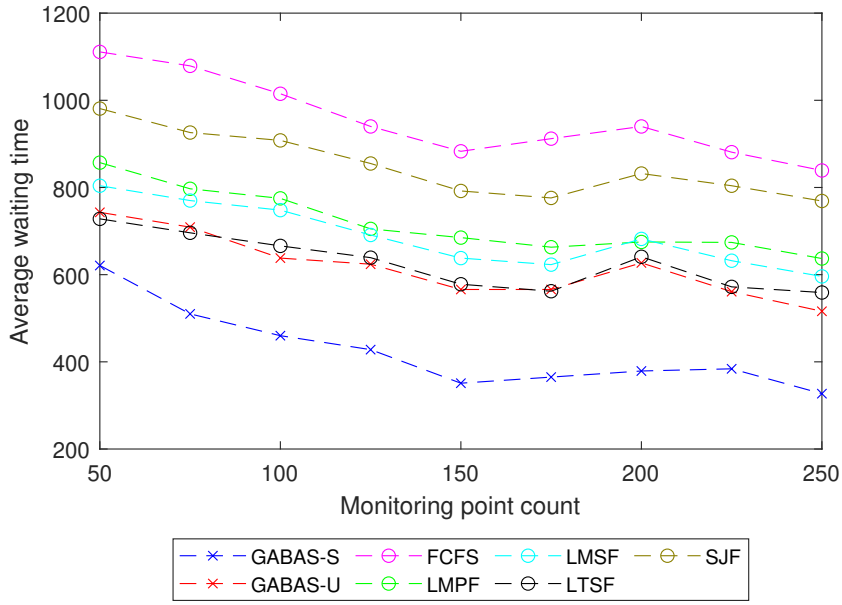
Figure 6.16: Comparison of algorithms in terms of (a) average turnaround time and (b) successful execution rate in Scenario S1.

In Scenario S2, the simulations are done to observe the effect of the number of monitoring points on the performance of the algorithms. Figure 6.17a presents the average makespan for Scenario S2. 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 the others diminishes as the number of monitoring points in the region gets larger, because it becomes easier to admit applications.

Average waiting time, average turnaround time, and average successful execution rate results of Scenario S2 are shown in Figures 6.17b, 6.18a, and 6.18b, respectively. As in the previous scenario, FCFS and SJF have the worst performance among all algorithms, while GABAS-S has the best one. The results for GABAS-U, LMSF, and LTSF are very close to each other; however, LMPF performs slightly worse among these algorithms.

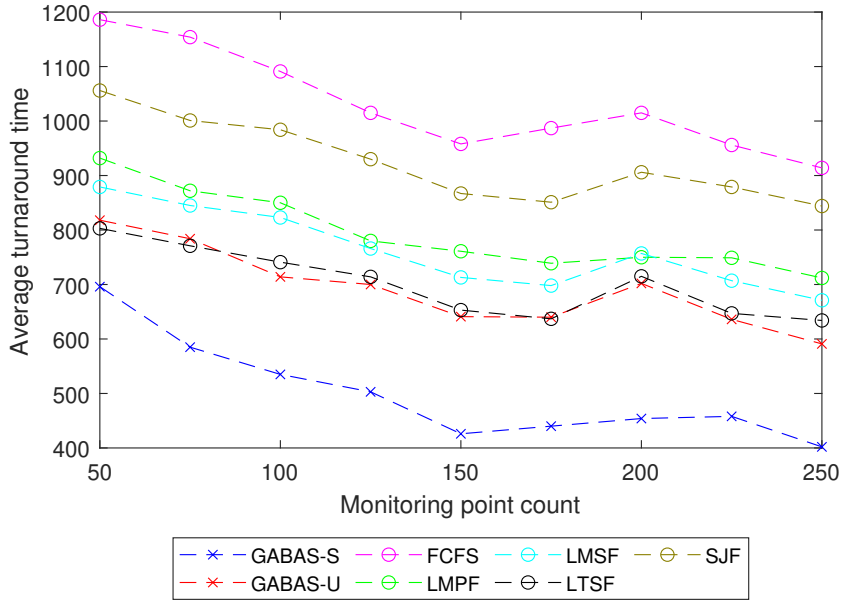


(a)

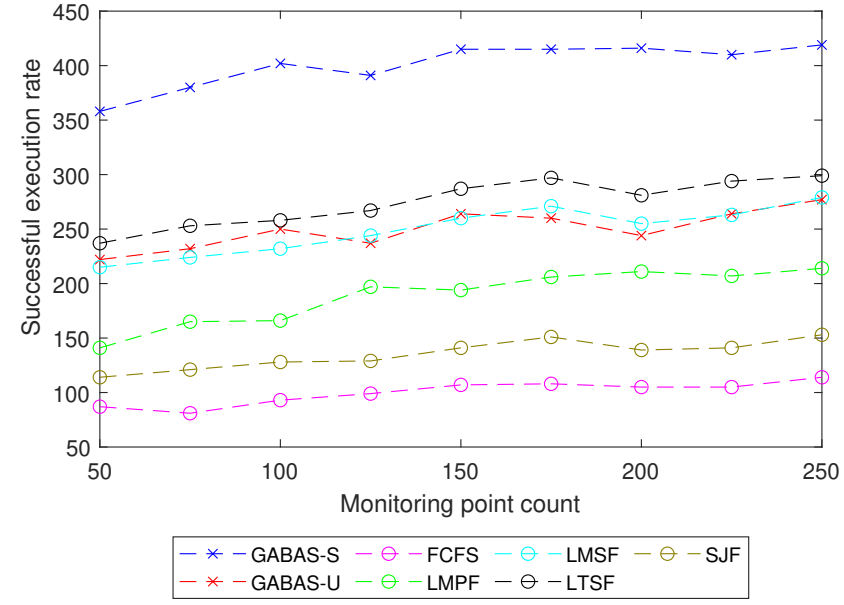


(b)

Figure 6.17: Comparison of algorithms in terms of (a) average makespan and (b) average waiting time in Scenario S2.



(a)

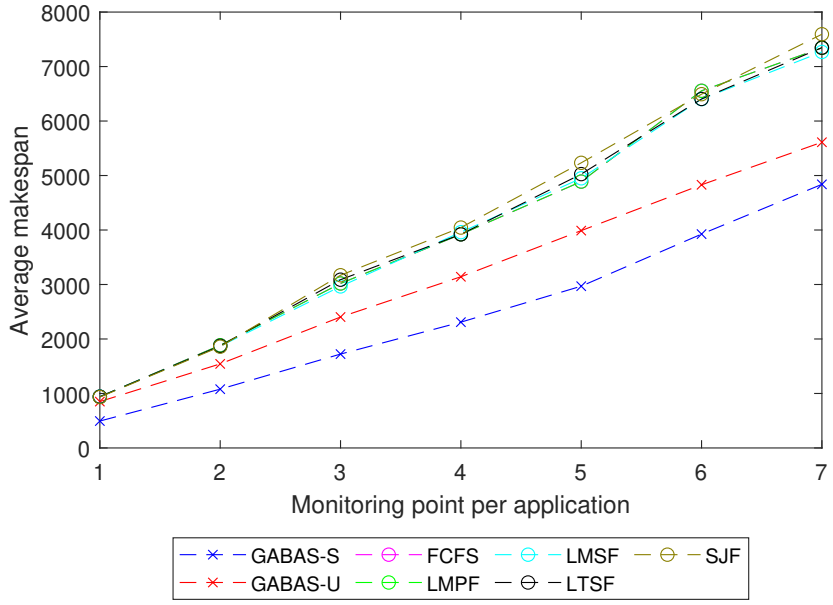


(b)

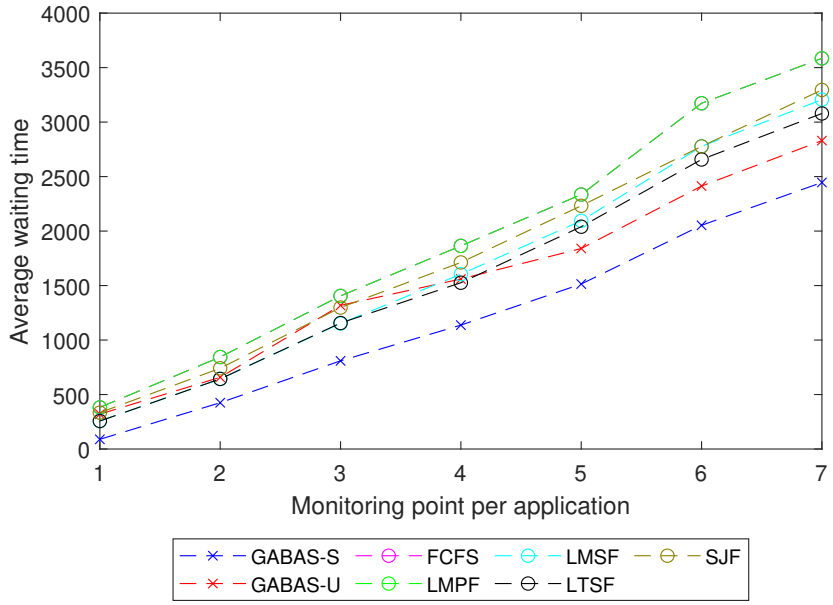
Figure 6.18: Comparison of algorithms in terms of (a) average turnaround time and (b) successful execution rate in Scenario S2.

In Scenario S3, we aim to see the impact of the number of monitoring point requests per application on the algorithms' performance. Figure 6.19a presents the results of this scenario in terms of makespan. GABAS-S still has the best results. After 5 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 greedy algorithms.

Figures 6.19b, 6.20a, and 6.20b display the results of average waiting time, turnaround time, and successful execution rate, respectively. GABAS-S is the superior one among all. For these criteria, LMPF and FCFS have the worst performance, 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.

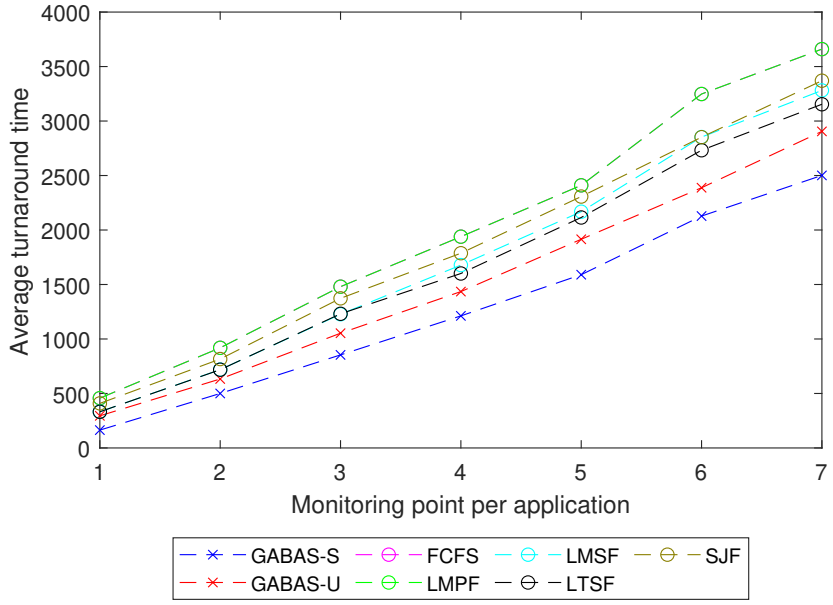


(a)

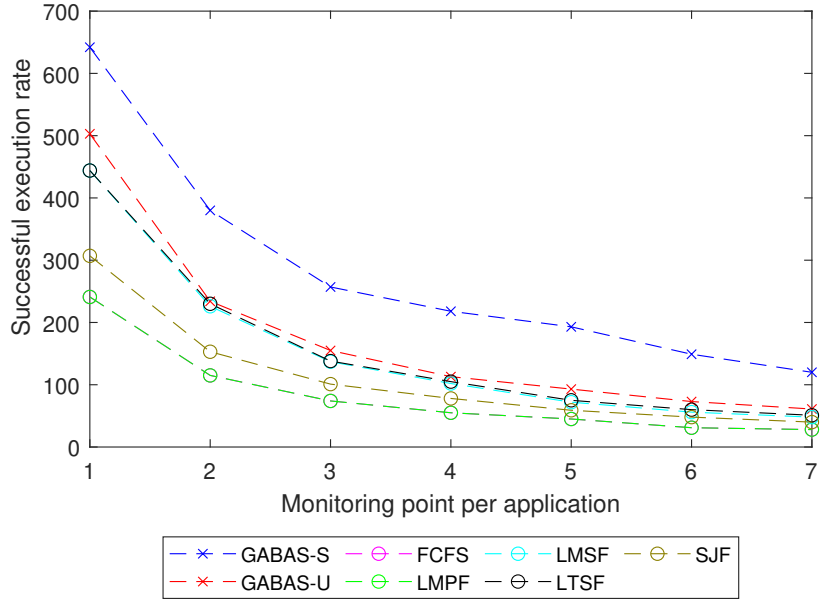


(b)

Figure 6.19: Comparison of algorithms in terms of (a) average makespan and (b) average waiting time in Scenario S3.



(a)

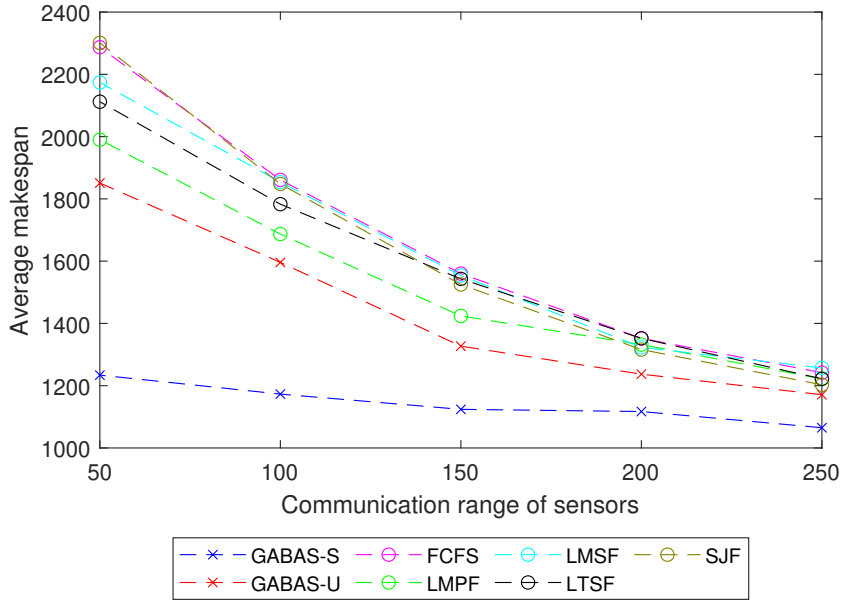


(b)

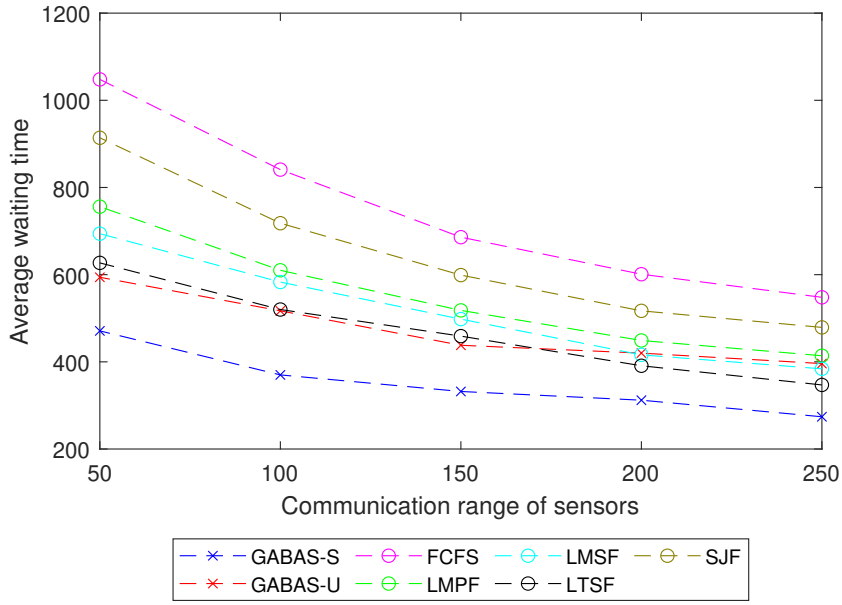
Figure 6.20: Comparison of algorithms in terms of (a) average turnaround time and (b) successful execution rate in Scenario S3.

In Scenario S4, we evaluate the effect of change in the communication range of the sensor nodes on the results. Figure 6.21a 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 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 6.21b, 6.22a and 6.22b, respectively. GABAS-S again has 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.

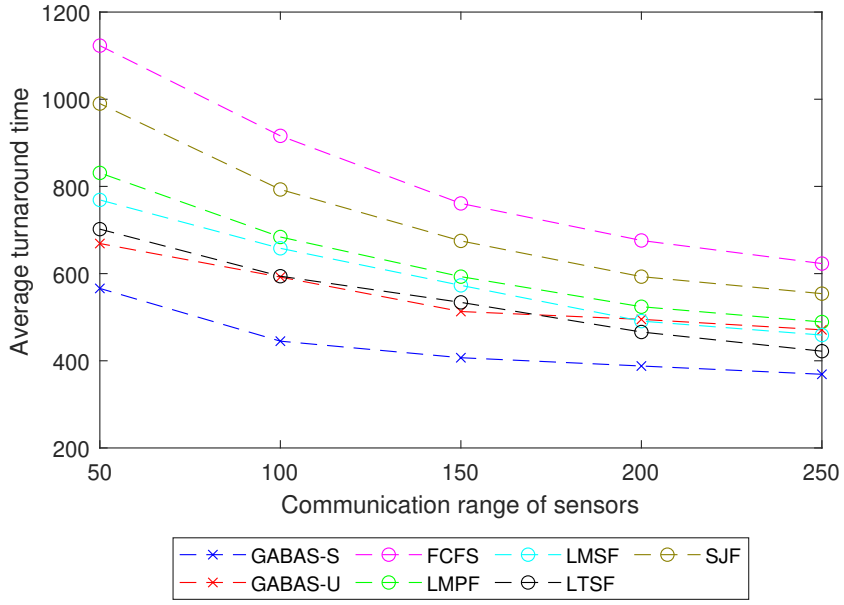


(a)

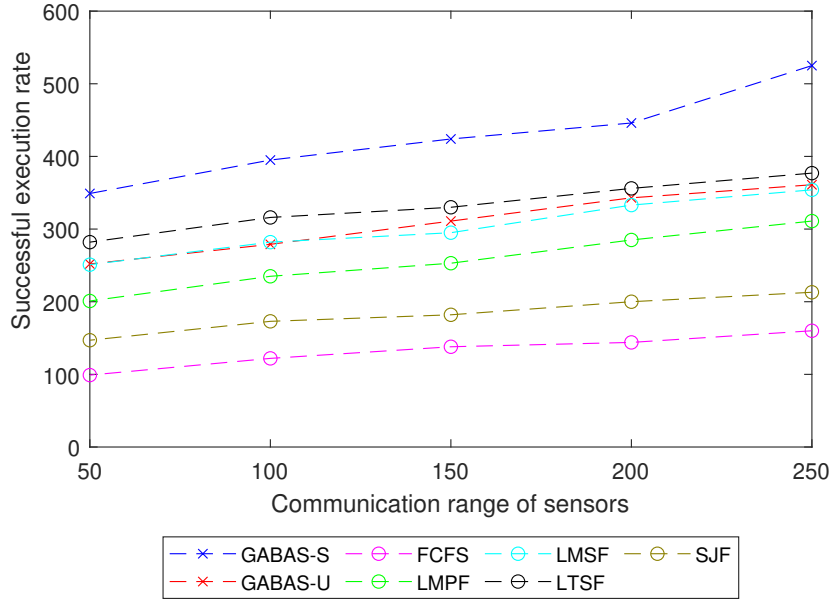


(b)

Figure 6.21: Comparison of algorithms in terms of (a) average makespan and (b) average waiting time in Scenario S4.



(a)

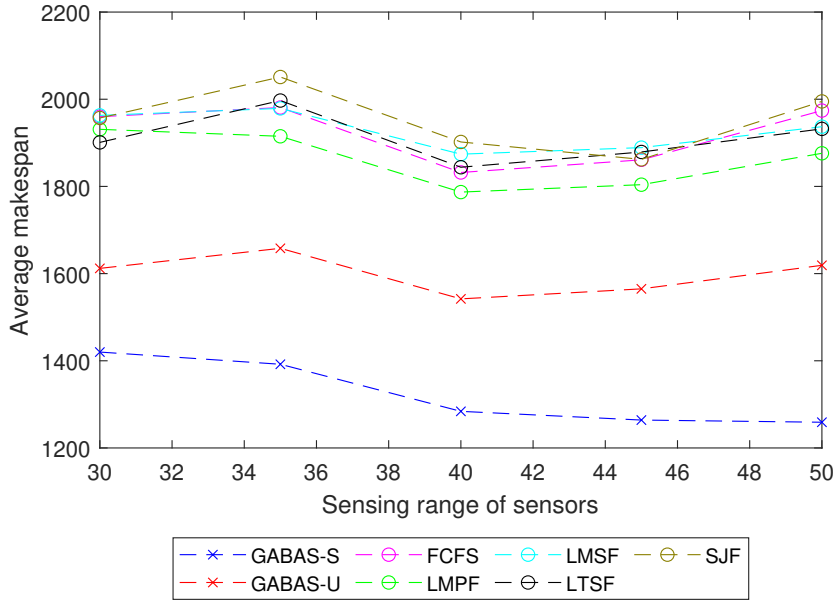


(b)

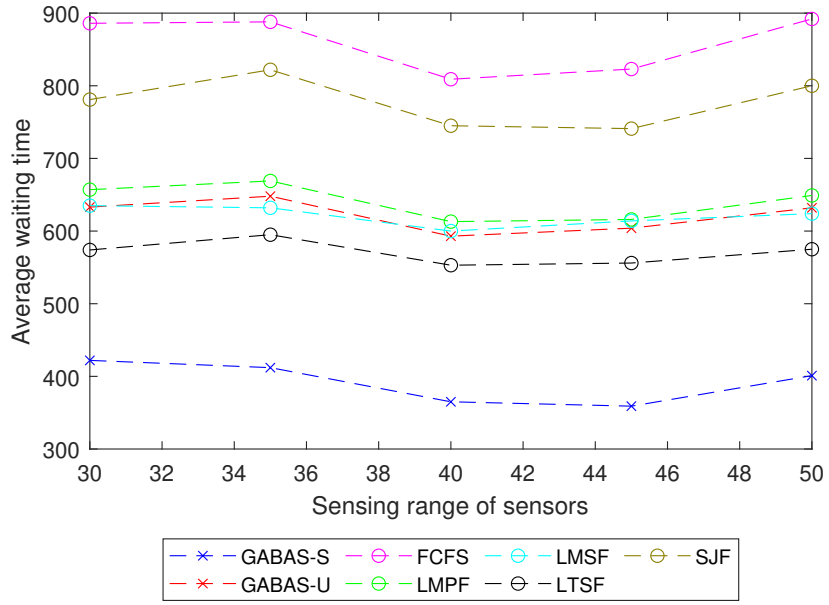
Figure 6.22: Comparison of algorithms in terms of (a) average turnaround time and (b) successful execution rate in Scenario S4.

In Scenario S5, we investigate the impact of the sensing range of sensor nodes on the results. Makespan results for this scenario are shown in Figure 6.23a. 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 6.23b, 6.24a, and 6.24b, 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.

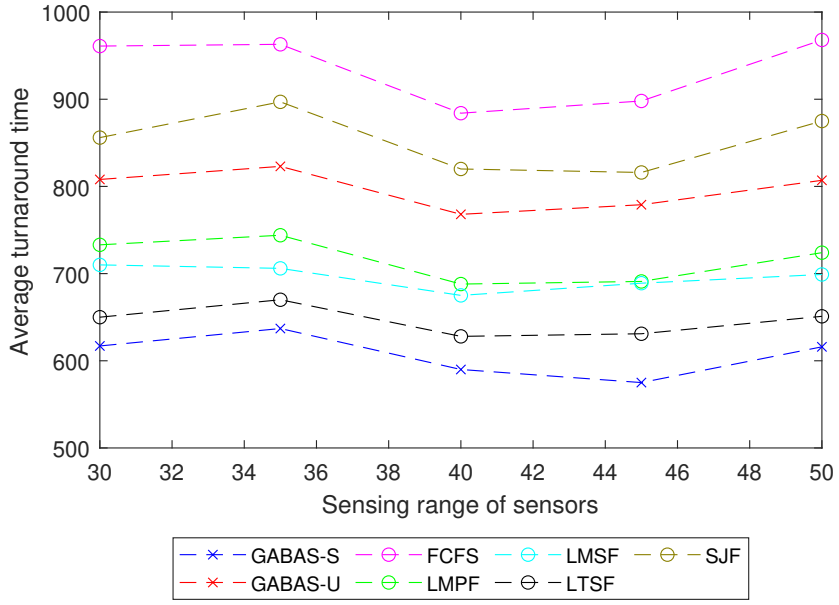


(a)

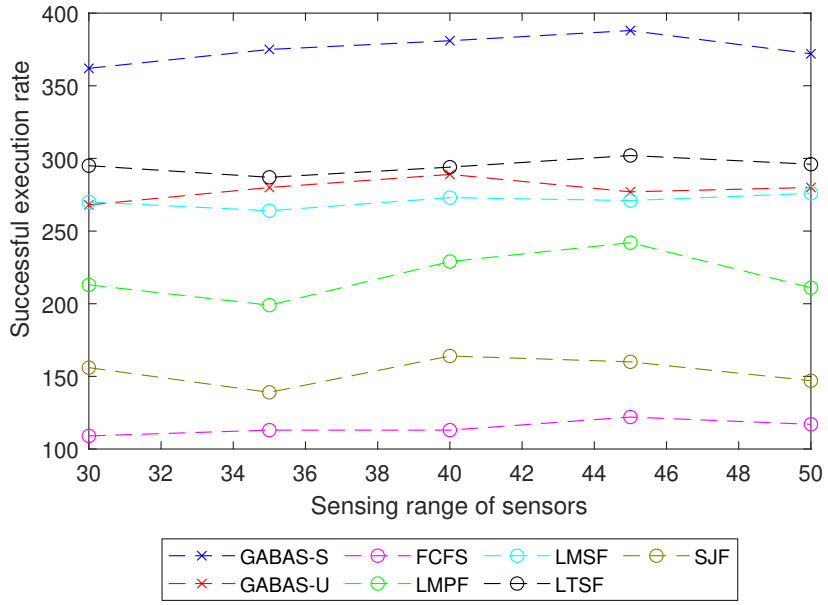


(b)

Figure 6.23: Comparison of algorithms in terms of (a) average makespan and (b) average waiting time in Scenario S5.



(a)

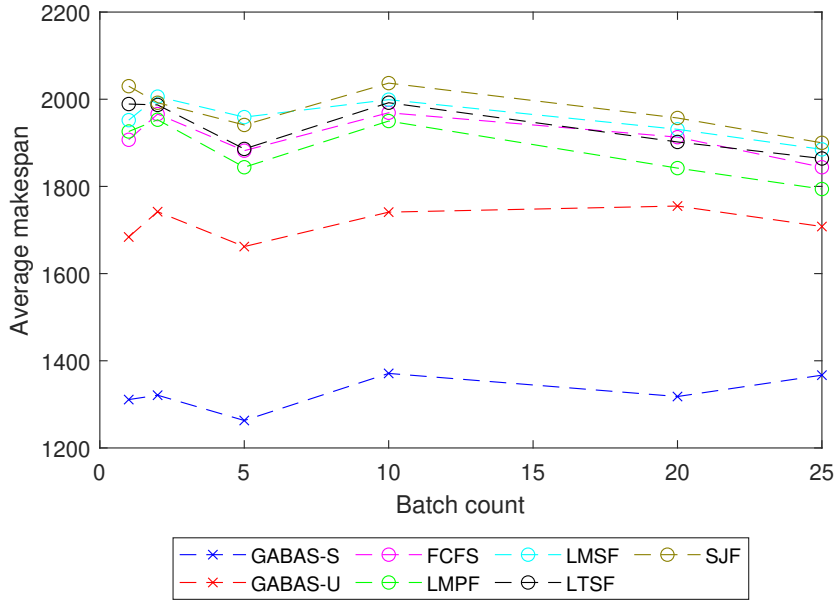


(b)

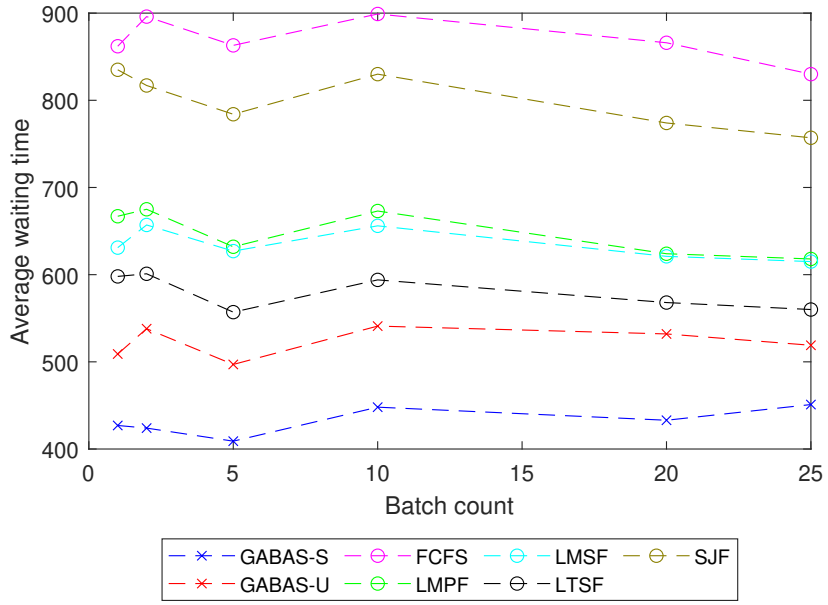
Figure 6.24: Comparison of algorithms in terms of (a) average turnaround time and (b) successful execution rate in Scenario S5.

In Scenario S6, we observe how the number of batches affects the performance of the algorithms. Figure 6.25a shows respectively the average makespan 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 a very similar performance whereas LMPF is slightly better compared to the others.

In terms of average waiting time, turnaround time, and successful execution rate, batch count again has no effect on 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 6.25b, 6.26a, and 6.26b, respectively.

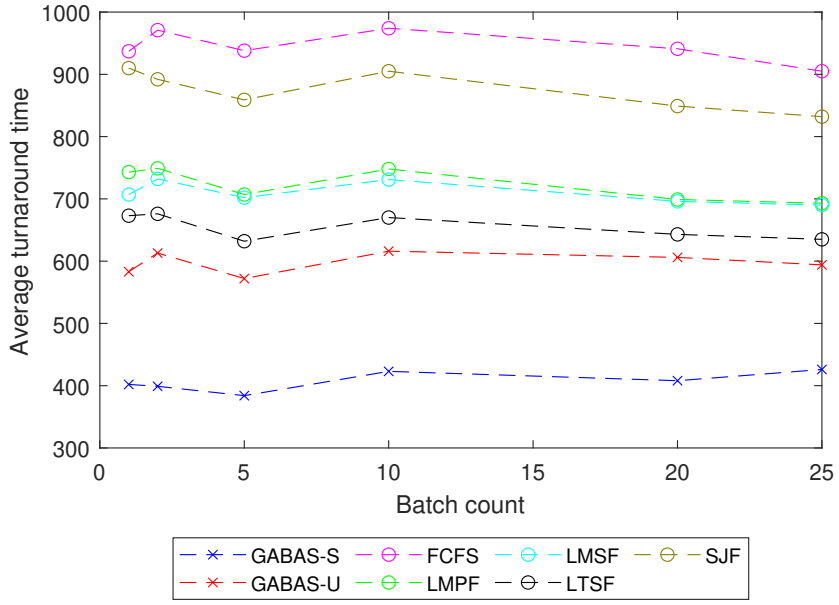


(a)

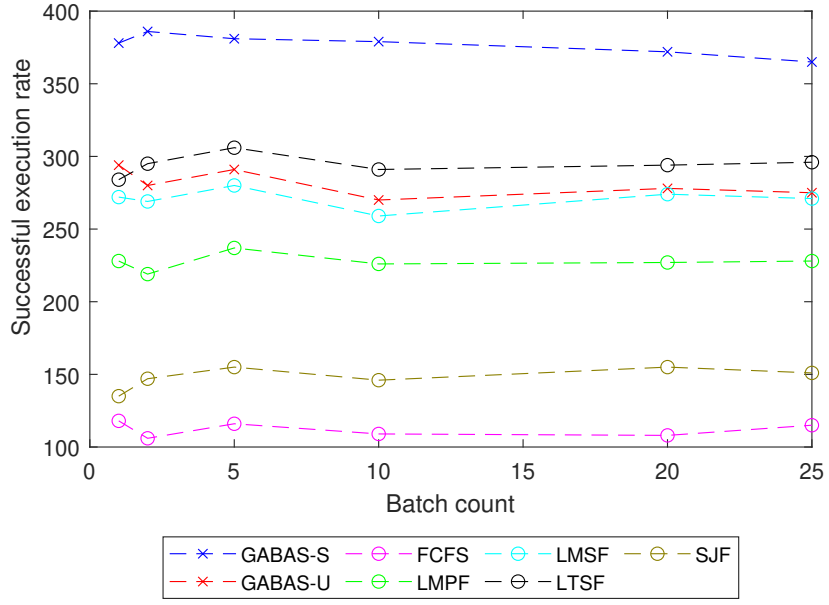


(b)

Figure 6.25: Comparison of algorithms in terms of (a) average makespan and (b) average waiting time in Scenario S6.



(a)



(b)

Figure 6.26: Comparison of algorithms in terms of (a) average turnaround time and (b) successful execution rate in Scenario S6.

As a summary, from all our results from experiments for application scheduling, we can draw the following conclusions:

- GABAS-S is superior to all other algorithms. 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 shows that the use of meta-heuristic algorithms is very effective in the application scheduling problem for WSNs.
- 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.

We also measured the running times of the algorithms. A selected set of results are provided in Table 6.4. Our GABAS-S and GABAS-U algorithms are the slowest algorithms due to the nature of the genetic algorithms. GABAS-S is faster than GABAS-U; because, with the shared-data approach it is easier to place more applications at the same time. Therefore, 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. Among all algorithms, FCFS is the fastest because it does not reorder the applications in the arrival queue.

Table 6.4: Running times of the algorithms for application scheduling in milliseconds.

Scenario	GABAS-S	GABAS-U	LMPF	LMSF	LTSF	FCFS	SJF
S1 #A: 500	204	286	38	24	25	3	46
S1 #A: 1000	352	524	105	63	70	2	90
S1 #A: 1500	2588	3094	378	253	282	6	378
S2 #MP: 50	377	899	93	67	71	3	103
S2 #MP: 100	326	574	102	69	75	2	98
S2 #MP: 250	229	289	73	51	64	1	92
S3 #MP/A: 1	335	562	56	32	31	3	67
S3 #MP/A: 3	1051	1508	362	256	245	3	413
S3 #MP/A: 5	2254	5023	886	740	752	7	974

6.3.1 Comparison with Linear Programming

As in the previous section, we compare our GABAS algorithm with linear programming to observe how close it performs to the optimal solution. We use the constraints from Section 5.1 to model a linear programming solution. We use the linearization methods from the previous section when necessary.

The results for the comparison are shown in 6.5. We only compare GABAS with LP in terms of average makespan since the main objective in the problem statement is minimizing the makespan. The results are the average of 100 runs. At each run, both algorithms are given the same network instance. Since linear programming takes too much time, we experimented with smaller network parameters. Other parameters and constraints are the same as in Tables 6.1 and 6.2. The results show that GABAS performs pretty well and has a very close performance to the optimal solution. Moreover, the running time of GABAS is very low compared to linear programming. The gap between running times increases while the network becomes larger.

Table 6.5: Comparison of GABAS with linear programming

Application Count	Mon. Point Count	Sensor Count	Base Station Count	GABAS		Lin. Prog.	
				Result	Time (ms)	Result	Time (ms)
30	20	30	3	116	32	114	5,314
30	20	30	4	105	37	99	5,532
50	20	30	4	132	104	124	16,815
50	40	50	7	114	167	101	52,438
50	100	50	7	126	210	103	124,871
100	100	25	25	195	361	172	411,627

6.4 Summary

In this chapter, we explained how our simulations are realized. We first detailed our experimental setup where we describe our network constraints. Then, we explained the scenarios in our experiments of both placement and scheduling problems. Then, we showed the results where we compare the performance of our proposed algorithms with the other methods from the literature.

Another result we showed is the running time of the algorithms for application scheduling. In these results, obviously, greedy algorithms perform faster compared to GABAS. FCFS is the fastest among greedy algorithms since it does not reorder the applications.

Using a linear programming framework, it was also shown that our algorithms GABAP and GABAS can perform very close to the optimum.

Chapter 7

Conclusion and Future Work

In this dissertation, we deal with application placement and application scheduling problems in wireless sensor networks. We first define the problems mathematically and show that the problems are NP-hard by reducing appropriate problems to them. Then, we detail our proposed approaches to solve the aforementioned problems.

We propose a monitoring point-based 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 reduce sensing and communication resource usage. Therefore shared-data approach lets the network support more applications at the same time.

For the application placement problem, we propose two algorithms, a greedy algorithm and a genetic algorithm called GABAP. The algorithms decide which applications should be admitted to the network, which monitoring point is sensed by which sensor, and which base station will process the related data. We provide extensive simulation results, which show the effectiveness of the sharing data approach and our algorithms. We compare the performance of our GABAP and greedy algorithms with DH [9] and RSVN [31] which are the state-of-the-art algorithms in the literature. The algorithms are run with both shareable and

unshareable approaches. We demonstrate that GABAP is superior to the other compared algorithms.

For the application scheduling problem, we propose a genetic algorithm, called GABAS, for scheduling applications effectively. We also provide 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 compare our proposed algorithms with each other and also with 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 show that GABAS outperforms all other algorithms in all comparison metrics. Among the other algorithms, LMPF provides better results than the others in terms of makespan, and LMSF and LTSF provide better performance in terms of waiting time, turnaround time, and successful execution rate.

We also compare the performance of GABAP and GABAS algorithms with optimal results obtained from our linear programming formulation. The results show that GABAP and GABAS perform very close to the optimum while taking much less time compared to the linear programming solution.

As a future work, the algorithms we propose can be extended for the use in WSNs having multi-hop architecture. In this way, sensor nodes that cannot directly send the collected data to the base station due to communication constraints can send the data to another node that can transmit the data to a base station. As a result, we can use underutilized connections.

Another future work can be multiple sensor node and base station support for a monitoring point. According to our assumption, a monitoring point can be assigned to a single sensor node and base station. Due to this restriction, if the sensor node, the base station, or the connection between them is being utilized, we cannot admit another application that requires the monitoring point to be sensed.

It is also possible to observe the effects of interference on the connections and the performance of the proposed algorithms. Moreover, some other communication QoS metrics, such as reliability and delay, can be investigated.

Another future work we propose is the study of the area coverage problem for WSNs. Given a set of applications and their monitoring point requirements, we can design algorithms that optimally place sensor nodes and base stations in the area to minimize the total cost while maximizing the number of placed applications.

Bibliography

[1] R. Mondal and T. Zulfi, "Internet of things and wireless sensor network for smart cities," *International Journal of Computer Science Issues*, vol. 14, no. 5, pp. 50–55, 2017.

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

[3] Q. Jiang, F. Kresin, A. K. Bregt, L. Kooistra, E. Pareschi, E. Van Putten, H. Volten, and J. Wesseling, "Citizen sensing for improved urban environmental monitoring," *Journal of Sensors*, vol. 2016. Article no. 5656245, 9 pages, 2016.

[4] A. Nasir, B.-H. Soong, and S. Ramachandran, "Framework of wsn based human centric cyber physical in-pipe water monitoring system," in *Proceedings of the 11th International Conference on Control Automation Robotics & Vision*, pp. 1257–1261, IEEE, 2010.

[5] 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*, pp. 96–103, Association for Computing Machinery, 2009.

[6] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh, "Deploying a wireless sensor network on an active volcano," *IEEE Internet Computing*, vol. 10, no. 2, pp. 18–25, 2006.

[7] 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*, vol. 8, no. 4, p. 611, 2011.

[8] D. Kandris, C. Nakas, D. Vomvas, and G. Koulouras, "Applications of wireless sensor networks: an up-to-date survey," *Applied System Innovation*, vol. 3, no. 1, p. 14, 2020.

[9] C. Delgado, M. Canales, J. Ortín, J. R. Gállego, A. Redondi, S. Bousnina, and M. Cesana, "Joint application admission control and network slicing in virtual sensor networks," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 28–43, 2017.

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

[11] 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 *Proceedings of the Third International Conference on Vocational Education and Electrical Engineering (ICVEE)*, pp. 1–6, IEEE, 2020.

[12] 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.

[13] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, and S. Shenker, "Rethinking enterprise network control," *IEEE/ACM Transactions on Networking*, vol. 17, no. 4, pp. 1270–1283, 2009.

[14] 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.

[15] W. Wei, R. Yang, H. Gu, W. Zhao, C. Chen, and S. Wan, "Multi-objective optimization for resource allocation in vehicular cloud computing networks," *IEEE Transactions on Intelligent Transportation Systems*, In press.

[16] S. Ali, A. Haider, M. Rahman, M. Sohail, and Y. B. Zikria, "Deep learning based joint resource allocation and rrh association in 5g-multi-tier networks," *IEEE Access*, vol. 9, pp. 118357–118366, 2021.

[17] Q. Chen, Z. Kuang, and L. Zhao, "Multiuser computation offloading and resource allocation for cloud–edge heterogeneous network," *IEEE Internet of Things Journal*, vol. 9, no. 5, pp. 3799–3811, 2021.

[18] L. S. Subhash and R. Udayakumar, "Sunflower whale optimization algorithm for resource allocation strategy in cloud computing platform," *Wireless Personal Communications*, vol. 116, no. 4, pp. 3061–3080, 2021.

[19] W. Xia and L. Shen, "Joint resource allocation at edge cloud based on ant colony optimization and genetic algorithm," *Wireless Personal Communications*, vol. 117, no. 2, pp. 355–386, 2021.

[20] V. M. Raee, D. Naboulsi, and R. Glitho, "Energy efficient task assignment in virtualized wireless sensor networks," in *Proceedings of the IEEE Symposium on Computers and Communications (ISCC)*, pp. 00976–00979, IEEE, 2018.

[21] T. Ojha, S. Misra, N. S. Raghuwanshi, and H. Poddar, "DVSP: Dynamic virtual sensor provisioning in sensor-cloud based internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5265–5272, 2019.

[22] M. Lemos, R. Rabêlo, C. de Carvalho, D. Mendes, V. Costa, *et al.*, "An energy-efficient approach to enhance virtual sensors provisioning in sensor clouds environments," *Sensors*, vol. 18, no. 3, p. 689, 2018.

[23] C. Delgado, S. Batista, M. Canales, J. R. Gállego, J. Ortín, and M. Cesana, "An implementation for dynamic application allocation in shared sensor networks," in *Proceedings of the 11th IFIP Wireless and Mobile Networking Conference (WMNC)*, pp. 1–8, IEEE, 2018.

[24] I. Leontiadis, C. Efstratiou, C. Mascolo, and J. Crowcroft, "Senshare: transforming sensor networks into multi-application sensing infrastructures," in *Proceedings of the European Conference on Wireless Sensor Networks*, pp. 65–81, Springer, 2012.

[25] S. Bhattacharya, A. Saifullah, C. Lu, and G.-C. Roman, "Multi-application deployment in shared sensor networks based on quality of monitoring," in *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 259–268, IEEE, 2010.

[26] S. M. Ajmal, S. Paris, Z. Zhang, and F. N. Abdesselam, "An efficient admission control algorithm for virtual sensor networks," in *Proceedings of the IEEE Intl Conf on High Performance Computing and Communications, IEEE 6th Intl Symp on Cyberspace Safety and Security, IEEE 11th Intl Conf on Embedded Software and Systems (HPCC, CSS, ICESS)*, pp. 735–742, IEEE, 2014.

[27] V. Cionca, R. Marfievici, R. Katona, and D. Pesch, "Judishare: Judicious resource allocation for QoS-based services in shared wireless sensor networks," in *Proceedings of the IEEE wireless communications and networking conference (WCNC)*, pp. 1–6, IEEE, 2018.

[28] S. Bousnina, M. Cesana, J. Ortín, C. Delgado, J. R. Gállego, and M. Canales, "A greedy approach for resource allocation in virtual sensor networks," in *Proceedings of the Wireless Days*, pp. 15–20, IEEE, 2017.

[29] R. Tynan, G. M. O'Hare, M. J. O'Grady, and C. Muldoon, "Virtual sensor networks: An embedded agent approach," in *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing with Applications*, pp. 926–932, IEEE, 2008.

[30] Y. Xu, A. Saifullah, Y. Chen, C. Lu, and S. Bhattacharya, "Near optimal multi-application allocation in shared sensor networks," in *Proceedings of the Eleventh ACM International Symposium on Mobile Ad hoc Networking and Computing*, pp. 181–190, ACM, 2010.

[31] Y. Li, Z. Zhang, S. Xia, and H.-H. Chen, "A load-balanced re-embedding scheme for wireless network virtualization," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 4, pp. 3761–3772, 2021.

[32] C. Abreu, F. Miranda, and P. Mendes, "Smart context-aware qos-based admission control for biomedical wireless sensor networks," *Journal of Network and Computer Applications*, vol. 88, pp. 134–145, 2017.

[33] V. Rahmati, "Near optimum random routing of uniformly load balanced nodes in wireless sensor networks using connectivity matrix," *Wireless Personal Communications*, vol. 116, no. 4, pp. 2963–2979, 2021.

[34] 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.

[35] 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, no. 114230, 2021.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] Z. Zhou, F. Li, H. Zhu, H. Xie, J. H. Abawajy, and M. U. Chowdhury, "An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments," *Neural Computing and Applications*, vol. 32, no. 6, pp. 1531–1541, 2020.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] N. Mansouri and M. M. Javidi, "Cost-based job scheduling strategy in cloud computing environments," *Distributed and Parallel Databases*, vol. 38, no. 2, pp. 365–400, 2020.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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 *Proceedings of the International Conference on Information Networking*, pp. 648–653, IEEE, 2013.

[51] 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.

[52] 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, no. 101970, 2021.

[53] 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.

[54] 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.

[55] 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*, In press.

[56] S. Singhal and A. Sharma, "A job scheduling algorithm based on rock hyrax optimization in cloud computing," *Computing*, vol. 103, pp. 2115–2142, 2021.

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

[58] H. Xu, Y. Liu, and W. C. Lau, "Optimal job scheduling with resource packing for heterogeneous servers," *IEEE/ACM Transactions on Networking*, vol. 29, no. 4, pp. 1553–1566, 2021.

[59] 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.

[60] 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.

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

[62] S. Bennett, *A History of Control Engineering, 1930-1955*. No. 47, IET, 1993.

[63] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer networks*, vol. 52, no. 12, pp. 2292–2330, 2008.

[64] S.-H. Yang and Y. Cao, "Guest editorial: Networked control systems and wireless sensor networks: theories and applications," *International Journal of Systems Science*, vol. 39, no. 11, pp. 1041–1044, November 2008.

[65] K. Yang, *Wireless Sensor Networks*. Springer, 2014.

[66] W. Dargie and C. Poellabauer, *Fundamentals of Wireless Sensor Networks: Theory and Practice*. John Wiley & Sons, 2010.

[67] N. Khalil, M. R. Abid, D. Benhaddou, and M. Gerndt, "Wireless sensors networks for Internet of Things," in *Proceedings of the IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pp. 1–6, IEEE, 2014.

[68] M. A. Kafi, Y. Challal, D. Djenouri, M. Doudou, A. Bouabdallah, and N. Badache, "A study of wireless sensor networks for urban traffic monitoring: applications and architectures," *Procedia Computer Science*, vol. 19, pp. 617–626, 2013.

[69] H. Liu, Z. Meng, and S. Cui, "A wireless sensor network prototype for environmental monitoring in greenhouses," in *Proceedings of the International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 2344–2347, IEEE, 2007.

[70] I. Benkhelifa, N. Nouali-Taboudjemat, and S. Moussaoui, "Disaster management projects using wireless sensor networks: An overview," in *Proceedings of the 28th International Conference on Advanced Information Networking and Applications Workshops*, pp. 605–610, IEEE, 2014.

[71] L. Yu, N. Wang, and X. Meng, "Real-time forest fire detection with wireless sensor networks," in *Proceedings of the International Conference on Wireless Communications, Networking and Mobile Computing, 2005.*, vol. 2, pp. 1214–1217, IEEE, 2005.

[72] M. Li and H.-J. Lin, "Design and implementation of smart home control systems based on wireless sensor networks and power line communications," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 7, pp. 4430–4442, 2014.

[73] S. Mansour, N. Nasser, L. Karim, and A. Ali, "Wireless sensor network-based air quality monitoring system," in *Proceedings of the International Conference on Computing, Networking and Communications*, pp. 545–550, IEEE, 2014.

[74] M. V. Ramesh, "Real-time wireless sensor network for landslide detection," in *Proceedings of the Third International Conference on Sensor Technologies and Applications*, pp. 405–409, IEEE, 2009.

[75] N. Singh, M. Mittal, *et al.*, "Review of wireless sensor networks-architecture and applications.," *International Journal of Advanced Research in Computer Science*, vol. 7, no. 6, 2016.

[76] S. Sharma, D. Kumar, and K. Kishore, "Wireless sensor networks-a review on topologies and node architecture," *International Journal of Computer Sciences and Engineering*, vol. 1, no. 2, pp. 19–25, 2013.

[77] M. A. Hamid, M. M. Alam, M. S. Islam, and C. S. Hong, "Enforcing fairness for data collection in wireless sensor networks," in *Proceedings of the 8th Annual Communication Networks and Services Research Conference*, pp. 192–198, IEEE, 2010.

[78] J. Heidemann, Y. Li, A. Syed, J. Wills, and W. Ye, "Underwater sensor networking: Research challenges and potential applications," *Technical Report ISI-TR-2005-603, USC/Information Sciences Institute*, 2005.

[79] M. Li and Y. Liu, "Underground structure monitoring with wireless sensor networks," in *Proceedings of the 6th International Symposium on Information Processing in Sensor Networks*, pp. 69–78, IEEE, 2007.

[80] T. Hayes and F. Ali, "Mobile wireless sensor networks: Applications and routing protocols," in *Handbook of Research on Next Generation Mobile Communication Systems*, pp. 256–292, IGI Global, 2016.

[81] I. T. Almalkawi, M. G. Zapata, J. N. Al-Karaki, and J. Morillo-Pozo, "Wireless multimedia sensor networks: current trends and future directions," *Sensors*, vol. 10, no. 7, pp. 6662–6717, 2010.

[82] M. Mitchell, *An Introduction to Genetic Algorithms*. MIT press, 1998.

[83] C. H. Benet, A. Kassler, and E. Zola, "Predicting expected tcp throughput using genetic algorithm," *Computer Networks*, vol. 108, pp. 307–322, 2016.

[84] M. K. Somesula, R. R. Rout, and D. V. Somayajulu, "Contact duration-aware cooperative cache placement using genetic algorithm for mobile edge networks," *Computer Networks*, vol. 193, p. 108062, 2021.

[85] A. Elkelesh, M. Ebada, S. Cammerer, and S. Ten Brink, "Decoder-tailored polar code design using the genetic algorithm," *IEEE Transactions on Communications*, vol. 67, no. 7, pp. 4521–4534, 2019.

[86] R. Elbaum and M. Sidi, "Topological design of local-area networks using genetic algorithms," *IEEE/ACM transactions on networking*, vol. 4, no. 5, pp. 766–778, 1996.

[87] C.-S. Wang and C.-T. Chang, "Integrated genetic algorithm and goal programming for network topology design problem with multiple objectives and multiple criteria," *IEEE/ACM Transactions on Networking*, vol. 16, no. 3, pp. 680–690, 2008.

[88] M. Kaliappan, S. Augustine, and B. Paramasivan, "Enhancing energy efficiency and load balancing in mobile ad hoc network using dynamic genetic algorithms," *Journal of Network and Computer Applications*, vol. 73, pp. 35–43, 2016.

[89] C. Y. Ngo and V. O. Li, "Centralized broadcast scheduling in packet radio networks via genetic-fix algorithms," *IEEE Transactions on Communications*, vol. 51, no. 9, pp. 1439–1441, 2003.

[90] A. Khan, M. A. Jaffar, J. Ullah, A. Din, A. Ali, N. Ullah, *et al.*, "Color image segmentation using genetic algorithm with aggregation-based clustering validity index CVI," *Signal, Image and Video Processing*, vol. 13, no. 5, pp. 833–841, 2019.

[91] C. Zhang, X. Wang, and C. Duanmu, "Adaptive typhoon cloud image enhancement using genetic algorithm and non-linear gain operation in undecimated wavelet domain," *Engineering Applications of Artificial Intelligence*, vol. 23, no. 1, pp. 61–73, 2010.

[92] D. Verma, V. P. Vishwakarma, and S. Dalal, "A hybrid self-constrained genetic algorithm (HSGA) for digital image denoising based on PSNR improvement," in *Advances in Bioinformatics, Multimedia, and Electronics Circuits and Signals*, pp. 135–153, Springer, 2020.

[93] E. Y. Kim and S. H. Park, "Automatic video segmentation using genetic algorithms," *Pattern Recognition Letters*, vol. 27, no. 11, pp. 1252–1265, 2006.

[94] R. Kaluri and P. Reddy, "Sign gesture recognition using modified region growing algorithm and adaptive genetic fuzzy classifier," *International Journal of Intelligent Engineering and Systems*, vol. 9, no. 4, pp. 225–233, 2016.

[95] M. Moussa, M. Hmila, and A. Douik, "A novel face recognition approach based on genetic algorithm optimization," *Studies in Informatics and Control*, vol. 27, no. 1, pp. 127–134, 2018.

[96] C. Kim, R. Batra, L. Chen, H. Tran, and R. Ramprasad, "Polymer design using genetic algorithm and machine learning," *Computational Materials Science*, vol. 186, p. 110067, 2021.

[97] T. Matsuoka, S. Yamamoto, and M. Takahara, "Prediction of structures and mechanical properties of composites using a genetic algorithm and finite element method," *Journal of Materials Science*, vol. 36, no. 1, pp. 27–33, 2001.

[98] S. Li, S. Li, D. Liu, R. Zou, and Z. Yang, "Hardness prediction of high entropy alloys with machine learning and material descriptors selection by improved genetic algorithm," *Computational Materials Science*, vol. 205, no. 111185, 2022.

[99] A. L.-S. Chua, N. A. Benedek, L. Chen, M. W. Finnis, and A. P. Sutton, "A genetic algorithm for predicting the structures of interfaces in multicomponent systems," *Nature Materials*, vol. 9, no. 5, pp. 418–422, 2010.

[100] S. Vitayasak, P. Pongcharoen, and C. Hicks, "A tool for solving stochastic dynamic facility layout problems with stochastic demand using either a genetic algorithm or modified backtracking search algorithm," *International Journal of Production Economics*, vol. 190, pp. 146–157, 2017.

[101] R. Zhang, S. Ong, and A. Y. Nee, "A simulation-based genetic algorithm approach for remanufacturing process planning and scheduling," *Applied Soft Computing*, vol. 37, pp. 521–532, 2015.

[102] H. Soleimani, K. Govindan, H. Saghafi, and H. Jafari, "Fuzzy multi-objective sustainable and green closed-loop supply chain network design," *Computers & Industrial Engineering*, vol. 109, pp. 191–203, 2017.

[103] F. Yu and X. Xu, "A short-term load forecasting model of natural gas based on optimized genetic algorithm and improved BP neural network," *Applied Energy*, vol. 134, pp. 102–113, 2014.

[104] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT Press, 2022.

[105] S. S. Skiena, *The Algorithm Design Manual*, vol. 2. Springer, 1998.

[106] J. Sanchez-Oro and A. Duarte, "Iterated greedy algorithm for performing community detection in social networks," *Future Generation Computer Systems*, vol. 88, pp. 785–791, 2018.

[107] Y.-Z. Li, Q.-K. Pan, R. Ruiz, and H.-Y. Sang, "A referenced iterated greedy algorithm for the distributed assembly mixed no-idle permutation flowshop scheduling problem with the total tardiness criterion," *Knowledge-Based Systems*, vol. 239, no. 108036, 2022.

[108] C. Lu, Q. Liu, B. Zhang, and L. Yin, "A pareto-based hybrid iterated greedy algorithm for energy-efficient scheduling of distributed hybrid flowshop," *Expert Systems with Applications*. Article no. 117555, 17 pages, 2022.

[109] Ç. Ateş, S. Özdel, M. Yıldırım, and E. Anarım, "Network anomaly detection using header information with greedy algorithm," in *Proceedings of the 27th Signal Processing and Communications Applications Conference*, pp. 1–4, IEEE, 2019.

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

[111] L. Malathi, R. Gnanamurthy, and K. Chandrasekaran, "Energy efficient data collection through hybrid unequal clustering for wireless sensor networks," *Computers & Electrical Engineering*, vol. 48, pp. 358–370, 2015.

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

[113] M. C. Çavdar, I. Korpeoglu, and Ö. Ulusoy, "Application scheduling with multiplexed sensing of monitoring points in multi-purpose IoT wireless sensor networks," 2022. arXiv: 2210.06393, Available at https://arxiv.org/abs/2210.06393, Accessed: 24 October 2022.

[114] A. Vlavianos, L. K. Law, I. Broustis, S. V. Krishnamurthy, and M. Faloutsos, "Assessing link quality in ieee 802.11 wireless networks: Which is the right metric?," in *Proceedings of the IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications*, pp. 1–6, IEEE, 2008.

[115] B. Demirel, A. Aytekin, D. E. Quevedo, and M. Johansson, "To wait or to drop: On the optimal number of retransmissions in wireless control," in *Proceedings of the European Control Conference*, pp. 962–968, IEEE, 2015.