Contemporary Mathematics
http://ojs.wiserpub.com/index.php/CM/

Research Article

Intelligent Trajectory for Mobile Element in WSNs with Obstacle Avoidance

Dasari Gowthami[1], Ebenezer Jangam[2], Suman Prakash P.[3*], Pallavi Joshi[4]

[1]Department of Electronics and Communication Engineering, S V College of Engineering, Karakambadi, Tirupati 517507, India
[2]Division of Artificial Intelligence and Machine Learning, Karunya Institute of Technology and Sciences, Coimbatore, India
[3]Department of Computer Science and Engineering-Artificial Intelligence (CAI), G.Pullaiah College of Engineering and Technology, Kurnool, Andhra Pradesh 518002, India
[4]Department of Computer Science, Amrita Vishwa Vidyapeetham, Bogadi, Mysuru, Karnataka 570026, India
Email: sumancse@gpcet.ac.in

Received: 11 May 2023; **Revised:** 24 May 2023; **Accepted:** 26 May 2023

Abstract: In wireless sensor networks (WSNs), mobile sink-driven data acquisition can mitigate hotspot issues, which further increases WSN efficiency, such as throughput, lifetime, and energy efficiency, while reducing delay and packet loss. Recently, most mobile sink algorithms have focused on efficient paths, and few consider obstacles in the network environment. Nevertheless, constructing an obstacle-aware trajectory in a WSN is challenging. In this context, this paper proposes a bug algorithm based on an obstacle-aware intelligent trajectory (CSOBUG) for a mobile sink to acquire data from sensor nodes in WSNs efficiently with the help of cat swarm optimization (CSO). The proposed CSOBUG algorithm has two phases: selecting visiting points and constructing a trajectory. A CSO-based clustering approach is used to select visiting points, and a bug algorithm is used to select a trajectory. Comparing CSOBUG with existing techniques, it is found that CSOBUG is less computationally intensive than the existing techniques. As well as outperforming traditional methods based on multiple performance metrics, the CSOBUG achieves superior results in a variety of scenarios.

Keywords: Bug 2 algorithm, data collection, mobile sink, obstacle-aware path, WSNs

MSC: 68M10

1. Introduction

A wireless sensor network (WSN) collects data packets from the field through sensor nodes (SNs) and transmits them to a base station (BS). This process continues in different applications such as climate analysis, forest fires, smart homes, healthcare, smart cities, air quality, etc. [1]. This process uses multi-hop or single-hop communication. This process uses some SNs between the transmitting node and the BS, which are called relay nodes. But these SNs operate using a battery, whereas these batteries are running on limited battery power, and replacing or recharging batteries is a hectic task [2, 3]. Relay nodes die soon because heavy data transmission through multiple SNs consumes more energy. This causes an interruption in data routing, and this problem is called the sink-hole or energy-hole problem [4, 5].

The mobile sink (MS) has been used in several works in recent years to address sink-hole issues. It acquires data

DOI: https://doi.org/10.37256/cm.5120243026

packets from a set of nodes, traverses these packets to the sink, and then visits another node with the data [6]. Instead of acquiring from every node, identify a set of nodes called visiting points (VPs), where an MS takes the data from them and moves to a BS. However, identifying such nodes is a challenging task. Further, determining a visiting order among them for the MS is another challenging task. especially, the path must not be long, and data packets should not overflow from the nodes and MS. In the case of the shortest path, it might be a challenging task. So finding an optimal VP and path is a challenging task. In the literature, some of the works address this issue, but they have their own benefits and limitations [7, 8].

Currently, most existing works are identified as VPs using clustering approaches [9, 10], and the path is found using the Traveling Salesman Problem (TSP) for visiting orders for MS. These clustering approaches identify the cluster head or centroid as a VP. But there are some cases, such as when some environments are associated with obstacles. The path construction team must be aware of these obstacles during the construction of the path. It is another challenging issue. In such cases, use the non-human intervention approach, which can easily categorize the difference between regular nodes and obstacles while constructing the path. This path must be optimal, which means not so long or shorter [11, 12]. In this context, self-learning approaches are used to determine the VPs and the path between them. Most of the terminology used in this paper is summarized in Table 1.

Table 1. Terminology

Terminology	Definition
WSN	It is a spatially and randomly deployed interconnected sensor node network to monitor a field of interest.
SNs	An SN is also called a mote, and it can perform the sensing, transmitting, and collecting of data from the field of interest with limited processing capabilities.
Sink/BS	A sink node is also called a BS, and it is a central hub for a network to analyze the data that is transmitted by the SNs. The sink node can act as a gateway to the Internet of Things [5].
VPs	The VP is a sensor node, and it can collect data from a set of SNs until an MS collects it. Some algorithms also consider it a cluster head [13].

This paper proposes an algorithm to determine the VPs using a clustering algorithm called Cat Swarm Optimization (CSO). In this process, we identify efficient VPs for data collection. Additionally, we construct the route using the bug algorithm in order to avoid obstacles in the WSNs.

The proposed algorithm is named the CSO-based bug (CSOBUG) approach for efficient data collection for WSNs. The contributions are summarized as follows:

- The proposed work identifies the VPs using cat swarm optimization, where the load among the VPs is equally distributed.
- By utilizing the bug algorithm, the obstacle-aware path can be constructed efficiently, determining the best route.
- Python is used to implement the proposed CSOBUG, and its quality metrics are compared with recent and related works.

The remaining paper is organized as follows: the literature of the paper is summarized along with its pitfalls in Section 2. The WSN network, energy, and system models, along with the problem formulation, are presented in Section 3. The proposed CSOBUG algorithm along with its complexity derivations are presented in Section 4. The simulation results of the proposed work are presented in Section 5, and Section 6 concludes the proposed work with a future research plan.

2. Related works

Recently, many algorithms have been published with respect to data collection using MS. Some of the MS-based

algorithms were studied and survived in [14], which extended the discussion, including limitations and benefits. In this context, all path-based algorithms are presented using obstacles and non-obstacles.

In WSNs, the static paths are determined by Lin et al. [15] using MS, and data fusion is performed. Path planning algorithms are based on ant colony optimization (ACO) for dynamic WSNs [7]. An ACO can be used in conjunction with pathfinding to find RPs in this approach. Additionally, the authors extend this work by adding virtual RPs with a modified probability function of the ACO [16]. Aside from providing an optimal path, these approaches also improve efficiency in terms of time, energy, lifetime, throughput, delay, and so on. A particle swarm optimization (PSO) approach was used by Mehto et al. [17] to construct efficient paths between RPs. To partition WSNs into clusters and cluster heads, Mehto et al. [17] used the squirrel search algorithm. However, none of these approaches are implemented to construct a trajectory in obstacle environments for MS.

In [18], MSs are used for data collection in grid-based sensor deployment and uniform data rates. It has been considered a density-based network during the simulation, and a route that provides the best performance along with an efficient path has been provided for this work. Multiple MSs are employed [19] to acquire data from a WSN, which generates uniform data. There are multiple-hop RPs partitioning the network, which further degrades WSN performance. An MS-based data collection using delay as a primary objective is presented in [20]. A dynamic routing procedure is used for sensitive data, while an MS collects delay-insensitive data. Further, the hotspot issue was mitigated by an extension of this work [21]. As a result, this approach requires a longer path and is computationally complex. An approach using a spanning tree-based approach to minimize delay and energy consumption is presented in [22]. The tree and trajectory for MS would take a long time to be constructed using this traditional approach.

According to Gupta and Saha [23], the artificial bee colony (ABC) approach was adopted for path construction and data collection for WSNs where data rates are equally distributed. An optimal set of RPs is identified in [24] using PSO for a longer lifetime with efficient energy utilization in the network. This is where the data is uniformly distributed. Data rate adaptive control is controlled through data collection using MS [25]. In addition to being low in computational complexity, this approach balances the network and energy as efficiently as possible. But this approach is not considered an obstacle-aware network. For efficient data collection based on MS, a method using fuzzy logic for clustering the network and a method for path construction are developed. An efficient path for MS using geometric approaches has been presented in [26]. Overall, none of the above algorithms are considered obstacles in the WSNs while constructing an MS trajectory.

In [10], artificial intelligence (AI) and artificial computational approaches are employed to construct a path within an obstacle WSN environment and collect data through MSs. ACO is initially used to build the clustering and AI-enabled approaches for path construction in WSNs for MSs. By utilizing mobile sensors, connectivity can be maintained while avoiding obstacles [27]. The collected data were analyzed, and the AI-based approach was compared to traditional clustering algorithms. The results showed that AI-based approaches could improve WSN performance with regard to obstacle avoidance and coverage. A sink-hole problem for WSNs is addressed using an information-based clustering approach in [28]. The [29] paper uses clustering for an obstacle-aware approach to WSN data acquisition using MS, whereas the [30] paper uses trajectory optimization for WSN data collection. As described in [31], Selvaraj and Vasanthamani propose a dynamic routing approach that eliminates obstacles in the WSNs by avoiding obstacles.

In the literature reviewed above, few works suggest an obstacle-aware pathway for MS to acquire sensed data from SNs, or RPs. Currently, only a few approaches have been published that use obstacle-aware paths for MS to accumulate sensing from SNs for WSNs. The existing approaches are computationally expensive and not optimal. Furthermore, they are not suitable for dynamic environments. Therefore, there is a need to develop an efficient and robust obstacle-aware pathfinding approach for MS in WSNs.

3. Problem formulation

The WSNs are assumed and represented as a graph G, where the SNs are considered the nodes (S) and the connections are treated as relations or edges D. So, in simple terms, the WSN is treated as $G(S, D)$. The $S = \{s_i\} \forall\, i \in (1, n)$. All the S are deployed in random places, and it is decided using Sah et al. [32]. The BS is represented as S_0. The total SNs in the network are n. The distance between two SNs is treated as d_{ij}, where s_i and s_j are the two nodes. The communication range is considered r_c. The VPs in the network are calculated as $V = \{V_1, V_2, ..., V_k\}$, and the

total number of VPs is treated as k. The order of the path is stored in $M = \{s_0 \cup order(V)\}$. The energy is measured using E (initially it is full), and the buffer is shown as B (initially it is empty). Buffer availability of a node s_i is measured as B_i. MS speed is v m/s. The total distance traveled is considered L. The obstacles are mentioned in the paper using $O = \{o_1, o_2,..., o_p\}$, where p is the number of obstacles. The frequently used notations used in this paper are listed in Table 2.

Table 2. Frequently used notations and their meanings

Notation	Meaning
G	WSNs
S	SNs
D	Relational edges
V	Set of VPs
O	Set of obstacles
E_{ij}^t	Energy required to transfer from node i and j
p	Number of obstacles
v	Velocity of MS
Λ	Amount of data packet transfer from the SN s_i to s_j
a	The energy consumption (EC) of amplification
b	The EC of processing a bit data
$\mathcal{N}$	Network lifetime
Υ_i	Length of trajectory of MS
B_i	The buffer occupancy of the s_i
$\mathcal{A}$	Adjacency matrix
$\mathcal{W}$	Similarity matrix
$\mathbb{D}$	Degree matrix
$\mathcal{L}$	Laplacian matrix
λ	Eigenvalues
E_a	Average energy consumption
δ	Number of rounds completed by the MS
ζ	Latency

This paper uses the energy model according to Donta et al. [16]. For reader convenience, we present it here again. Λ data packet sent from node s_i to s_j consumes energy as per equation (1).

$$E_{ij}^t = a\Lambda + b\Lambda d_{ij} \tag{1}$$

where a is amplification energy, b is processing energy for a bit data. The number of bits transferred is considered as Λ from node i to j. The energy required for receiving Λ-bits from nodes $i \forall i \in S$ is computed using equation (2).

$$E_i^{rx} = r\Lambda \tag{2}$$

where r is the energy for receiving a bit data from other nodes. The energy required for $i \forall i \in S$ for receiving and transferring at a time t is extracted from equation (3).

$$E_i = E_i^{rx} + E_{ij}^{tx} \tag{3}$$

We can calculate the remaining energy of s_i, as shown in equation (4).

$$\xi_i = E - E_i \tag{4}$$

The lifespan of network ($\mathcal{N}$) is the time which is calculated using the time until the first node dies, the number of rounds $\lfloor \varphi \rfloor$ completed. We consider it in terms of minutes. The proposed CSOBUG, $\mathcal{N}$ is calculated using equation (5)

$$\mathcal{N} = \sum_{i=0}^{\lfloor \varphi \rfloor} \Upsilon_i \times \left(\frac{60}{v} \right) \tag{5}$$

The distance traveled by MS in ith round at a velocity v is represented as Υ_i.

The ultimate objective of the proposed CSOBUG is to maximize the $\mathcal{N}$ through improving the utilization efficiency of E_i and B_i, respectively.

$$\max \mathcal{N} \tag{6}$$

subjected to min E_i and max B_i.

4. Proposed CSOBUG

Two steps are involved in the proposed CSOBUG: VP selection and trajectory finding. The VP selection uses the CSO algorithm to balance the load between the nodes, reducing the number of multi-hop communications as much as possible. Further, the bug algorithm helps to find an obstacle-aware path between the VPs in the WSN. So, the detailed contributions are detailed in the subsequent sections.

4.1 *VP selection*

CSO mainly works by recognizing cat behaviors, such as tracking and seeking. It starts by doing a clustering mechanism, then optimizes the clusters. This algorithm takes as input the number of clusters denoted as k, and it converts these into a network of nodes n. In this algorithm, the optimal k is determined by the estimated cost of the proposed work. Once the estimated cost is determined, the algorithm can begin. The optimal number of clusters is decided based on the approaches used in Donta et al. [33].

The initial virtual cluster heads are identified by using a distance function once the k (amount of clusters) has been fixed, as shown in equation (7)

$$d_{ij} = \sqrt{\sum_{i=1}^{n} (s_i - s_j)^2} \ \forall \ j = ((i+1)\%n) \tag{7}$$

To optimize the clusters using equation (8), we must update the fitness function during the seeking and tracing operations.

$$Fit = \sum_{i=1}^{k} \sum_{s \in C_i} \left(\| s - H_i \| \right)^2 \tag{8}$$

It is necessary to achieve minimal clustering results during the running of the clustering CSO algorithm. The fitness values are initially assigned to infinity. Algorithm 1 shows the pseudocode of the CSO clustering and VP selection. Initially, we define k random cluster heads for the partition H_i. It is clustered through a distance function defined in

equation (7). Once the initial cluster heads for H_i are determined, the fitness function is updated with multiple iterations of running *Seeking()* and *Tracing()* process.

Algorithm 1. Clustering through CSO

1: $Fit_s = Fit_t = \infty$

2: k random points are initialized in the network as temporary cluster heads $H_i \forall 1 \leq i \leq k$

3: Group the nearest points from the cluster head to make a cluster using an Euclidean distance using equation (7)

4: Calculate the fitness value for the algorithm using equation (8)

5: **while** $i \leq I_{limit}$ **do**

6: *while* $Fit_s \leq Fit$ **do**

7: Operate Seeking(); (Call Algorithm 2)

8: **end while**

9: **while** $Fit_t \leq Fit$ **do**

10: Operate Tracing(); (Call Algorithm 3)

11: **end while**

12: **end while**

4.1.1 *Seeking()*

There is no movement during *Seeking()*, but the cats keep on looking around to see if they can find anything to eat or strive for. It requires four main functionalities to execute *Seeking()*, but we only use three in this approach: seeking memory pool (*SMP*), self-position consideration (*SPC*), and range of dimensions (*SRD*). This metric is of little importance because it is taken into account by default at 100%. In *SMP*, the number of cluster center copies is expressed as some Boolean value [*TRUE, FALSE*], while *SPC* represents the ratio of mutations between 0 and 1. When the *Seeking()* parameter is called, the j value is updated based on the *SPC* value and starts at *SMP*. Following that, we calculate the mutative ratio by multiplying *SRD* by H_i. Determine the fitness value for each H_i using equation (8). At each H_i, begin by modifying or moving the H_i based on the computed fitness. A constant equation (9) will be used to find the H_i that had the highest probability among the multiple H_i.

$$Prob_i = \frac{|\, Fit_i - Fit_{\max} \,|}{Fit_{\max} - Fit_{\min}} \forall\ 0 < i < j \tag{9}$$

You now need to keep updating the fitness rate with each iteration and change the existing H_i to the updated H_i. You also need to update the cluster members based on equation (7). As long as Fit_s is below the minimum of Fit_s previous, *Seeking()* is continued. Algorithm 2 shows the *Seeking()* operation.

Algorithm 2. CSO *Seeking()*

1: Define *SMP*, *SRD*, and *SPC*
2: for $i = 1$ to k **do**
3: $H_i \rightarrow SMP$
4: **if** $SPC == TRUE$ **then**
5: $j = SMP - 1$
6: **end if**
7: $SV \leftarrow SRD * H_i$ //SV indicates shifting value
8: **end for**
9: **for** $h = 1$ to SMP **do**
10: $SV \pm RandomInt()$
11: **end for**
12: Use equation (7) to group the data
13: Calculate Fit_s using equation (8)
14: Identify the new cluster head H_i

4.1.2 *Tracing()*

The *Tracing()* operation starts once the *Seeking()* holds. In this process, the cat starts reacting to the targets. Tracking aims to shift cluster head to an optimal location by shifting it from one to another. The *Tracing* updates cat *i*'s velocity in (H_i) according to the equation (10).

$$\lambda_i = \lambda_i + (H_{best} - H_i) \times \zeta \times \xi \tag{10}$$

where a constant is represented using ξ, and a random value between the [0,1] can be assigned to ζ. H_{best} is the best position get while optimizing the fitness value. With equation (11), the position of H_i will be updated from its current position to its optimal position.

$$H_i = H_i + \lambda_i \tag{11}$$

Until reaching the minimum fitness value, the tracing function iterates for each H_i, where $1 \leq i \leq k$.

During iteration of the CSO algorithm, the optimal partitions are determined, and they are used to construct paths. Algorithm 3 shows the pseudo code for *Tracing()* operation.

Algorithm 3. CSO *Tracing()*

1: **for** $i = 1$ to k **do**
2: Update λ_i using equation (10)
3: Update pos(i); using equation (11)
4: H_i = new(H_i)
5: **end for**
6: Use equation (7) to group the data
7: Calculate Fit_t using equation (8)

4.2 *Obstacle-aware path*

An obstacle-aware path uses the bug algorithm [34], one of the traditional, low-computation algorithms. There are many advantages to using this approach in local maxima, including its efficiency and ease of use. Once obstacles are identified, they are traversed along their edges, and the path is constructed in a straight and short line. In addition, it checks the minimum distance between the MS and the RP simultaneously. This flow chart (Figure 1) illustrates the detailed process of constructing a trajectory using the bug algorithm. This makes it easier to find a feasible path and reduces the risk of getting stuck in local minima. Furthermore, the bug algorithm is able to quickly identify the shortest path between two points while avoiding obstacles.

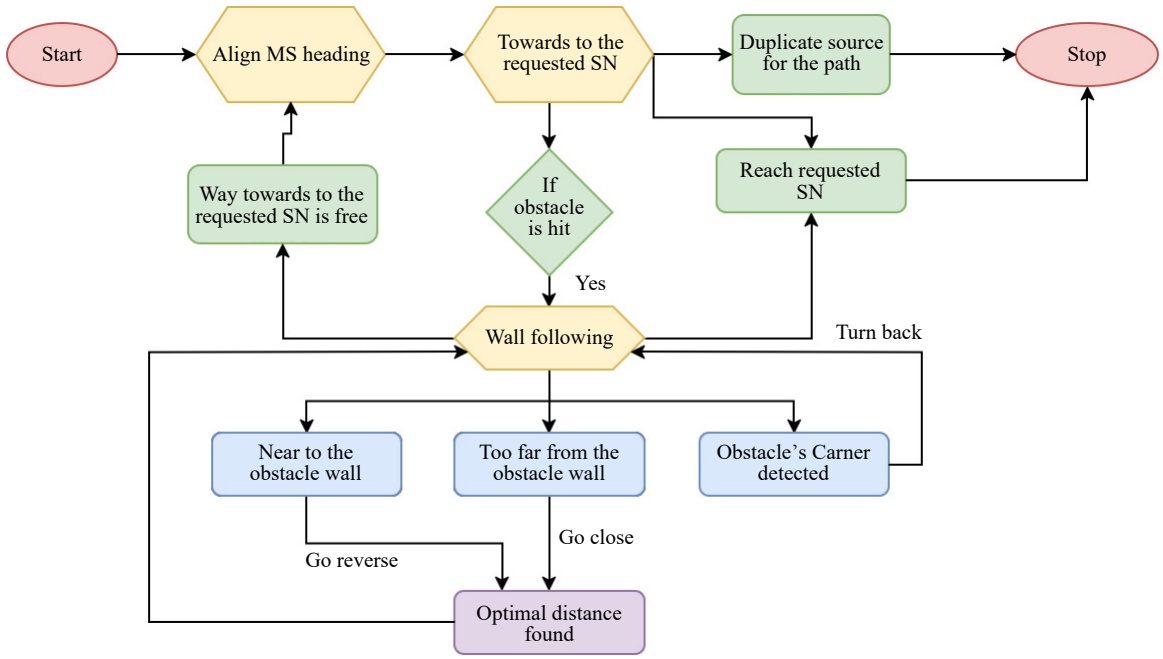


Figure 1. The general working procedure of the bug algorithm during path construction

As discussed earlier, once an obstacle is detected, the bug algorithm begins traveling adjacent to it. The bug algorithm will then trace the obstacle until a clear path is found. It will repeat this process until it reaches its destination. The bug algorithm is a useful tool for navigating a complex environment. Using MS' current location (p, q), it calculates a new path, and the next VP (x, y) is taken as a straight line. The slope and y-intercept are calculated using equations (12) and (13).

$$slope = \tanh \frac{y-q}{x-p} \tag{12}$$

$$intcpt = q - (slope \times p) \tag{13}$$

Once the obstacle is found, the bug algorithm turns to identify the edges of the obstacle and traverses according to them. A number of options are available for circumnavigating the obstacle, but it determines which one is the shortest. By using the shortest route, the obstacle can be circumnavigated quickly and efficiently. It can also save time and resources. As shown in Algorithm 4, Initially, the MS was located at BS. From BS, it moves towards the first VP, which is the nearest. During this journey, it will check for any obstacles presented between them. By eliminating the straight line in the event of an obstacle, it constructs a curved path between the MS and the next VP.

For a better understanding of the path approach, we illustrate it through an example as shown in Figure 2. This

example shows the path construction between an MS and a VP. Initially, in Figure 2(a), an MS, VP, and two obstacles are considered. Two obstacles are of different sizes and shapes. From Figure 2(b), a virtual path is built between MS and VP. This path is a straight line where no obstacle-aware path is considered. From Figure 2(c), the bug algorithm is able to determine the edge for each obstacle. These edges help avoid the intersection with the obstacles to the MS, finally constructing the path. Once the MS approaches the edge, it continues traversing around the edge until it touches the virtual line, as shown in Figure 2(d). Further, it follows the virtual line until it touches any edge of another obstacle or VP. If it identifies any obstacles again, it repeats the same process until the VP is reached. It repeats the same process until it returns to the BS.

Algorithm 4. Path construction using bug algorithm

```
1: while TRUE do
2:     repeat
3:         Move MS from current location to next VP.
4:     until O_i = TRUE
5:     if status(S) == VISITED then
6:         STOP
7:     end if
8:     repeat
9:         Travel near O_i's edges
10:    until status(S) == VISITED
11:    Calculate shortest distance
12:    Move to the initial position
13:    if Move the MS towards SNs then
14:        GOAL not reached
15:        EXIT
16:    end if
17: end while
```

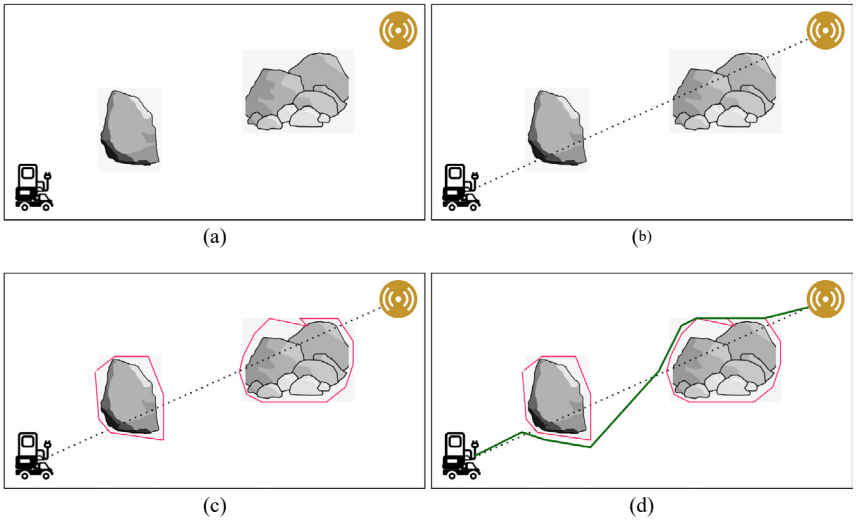


Figure 2. (a) Illustrative example of path construction approach; (b) initial obstacles, sensor node, and MS positions; (c) virtual path between the MS and SNs; (d) finding the borders for the obstacles

4.3 *Time complexity analysis*

The CSOBUG algorithm's time complexity consists of two main components: selecting VPs and constructing a trajectory. The VP selection uses CSO, and the time complexity of CSO is computed according to Ahmed et al. [35]. It is approximately $O(n^2)$, where n is the number of SNs, as discussed previously. As we derived the path construction for the MS using the bug algorithm, it takes approximately $O(n^2 \log n)$ asymptotic time complexity. So, the final asymptotic time complexity for the CSOBUG is $O(n^2) + O(n^2 \log n) \approx O(n^2 \log n)$ which is better than existing approaches.

5. Simulation results

The Python simulator (Python 3.11) is used to implement and test the proposed CSOBUG and existing approaches. In an area of 600 square meters, 500-1,000 SNs are considered, and deployment is based on the dataset generation of Sah et al. [32]. The SNs battery capacity is 192.8 KJ (E_i^0), and all the nodes' batteries are initially fully charged. The simulation time for the proposed and existing models was evaluated at 10 hours. MS velocity is 1 m/s. As part of the simulation, we used parameters for determining CSO, the lifespan of the WSN, the average energy drain (AED), the fairness index of the AED, the utilization of the buffer, the throughput, the delay, and the travel length of the MS. Other metrics used in this paper are summarized using Table 3. These metrics are estimated under two main scenarios, such as event- and time-driven WSNs. The properties of event- and time-driven are similar to those of Kafi et al. [36] and are summarized as follows:

Event-driven WSNs: WSNs with event-driven acquisition acquire packets only when there is an event on the network.

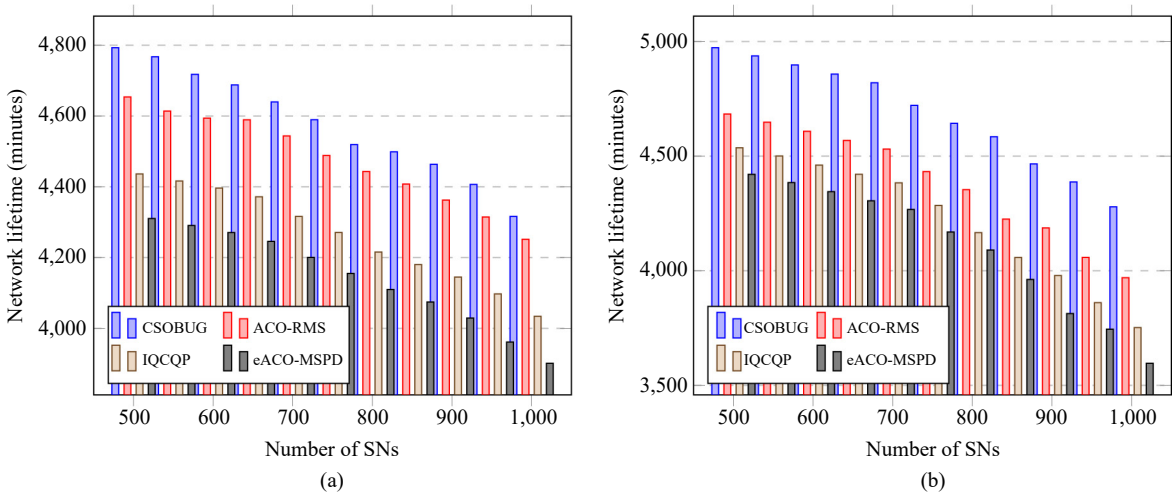
Time-driven WSNs: WSNs that are time-driven acquire the data continuously, regardless of the events happening in the environment.

Table 3. Frequently used notations and their meanings

Parameter	Value
Simulator	Python 3.10
n	500-1,000
A	600 sq.m
Node type	Zolertia Z1 mote
$\mathcal{B}_{max}$	96 kB
S_0	(0,0)
a	0.05 J
b	0.02 J
c	0.02 J
r_t	15 m
r_c	20 m
MAC protocol	TDMA
Data transfer rate	40 Kbps
Packet generation	3 pkt/sec
Packet size	128 bits
E_{max}	192.4 KJ
Channel bandwidth	10^3 Hz
Noise spectral power density	10^{-9} W/Hz
Length of a frame	6s
Topology	Mesh topology

5.1 *Network lifetime*

An important metric to determine the performance of a WSN is its lifetime ($\mathcal{N}$). Our study considers how long, in minutes, it takes a node to drain its energy until its battery no longer holds any charge. Equation (5) shows how to calculate it. Figure 3(a) and (b) show two scenarios under which the $\mathcal{N}$ is estimated: event-driven and time-driven scenarios, respectively.



Note: Ant colony optimization-based RPs selection and MS scheduling (ACO-RMS); integer quadratically constrained quadratic programming (IQCQP); extended ACO-based MS path determination (eACO-MSPD)

Figure 3. Network lifespan of (a) event-driven WSNs and (b) time-driven WSNs

Based on two different scenarios, we analyze the $\mathcal{N}$ analysis, including a time-driven and event-driven WSN. Due to the fact that most existing networks follow this strategy, the time-driven network is considered to be a regular network. It can be seen from Figure 3(a) that $\mathcal{N}$ improves by around 1.89-23.02% compared with ACO-RMS, 7.65-73.84% over IQCQP, and 9.87-11.21% compared with eACO-MSPD for event-driven WSNs. Accordingly, in the time-driven network (Figure 3(b)), we can notice that the proposed CSOBUG approach outperforms the existing approaches ACO-RMS, IQCQP, and eACO-MSPD in terms of $\mathcal{N}$. ACO-RMS, IQCQP, and eACO-MSPD algorithms all exhibit lower $\mathcal{N}$ than CSOBUG, around 5.62-7.21%, respectively. MS scheduling over the network and efficient VP selection are among the reasons for the improvements in the proposed CSOBUG.

5.2 *AED*

In WSNs, the AED is calculated as a ratio of the total number of SNs to the total energy drain of each SN. A number of factors determine SNs' energy drain, which is discussed in Section 3. To analyze the performance of SNs, we estimate their AED. In this context, we estimate the AED by using different kernel density estimation (KDE) functions in conjunction with the two scenarios of WSNs as depicted in Figure 4.

$$E_a = \frac{1}{n}\sum_{i=1}^{n} E_i \tag{14}$$

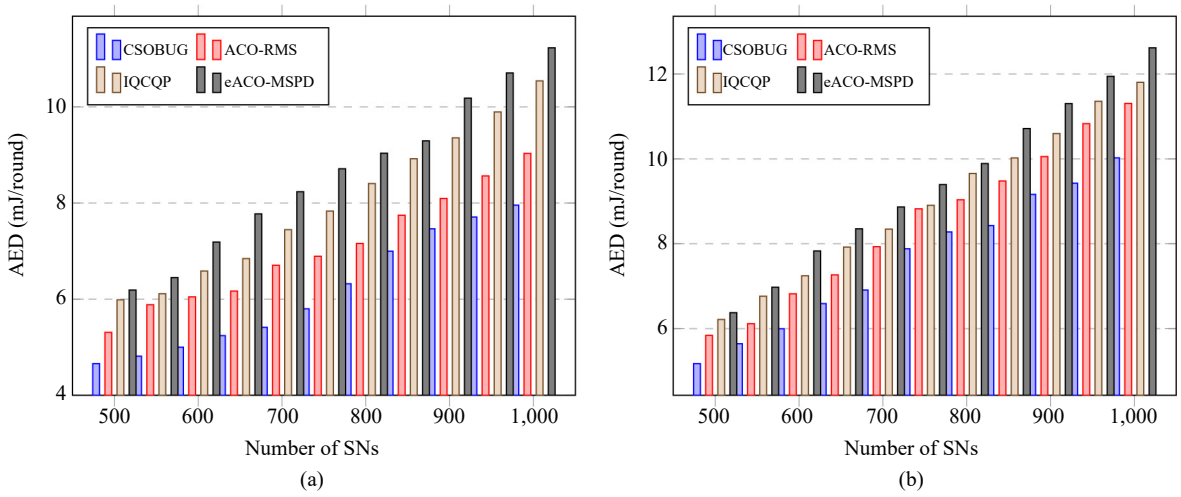


Figure 4. AED of (a) event-driven WSNs and (b) time-driven WSNs

Based on two different scenarios, including an event-driven and time-driven WSN, we analyze the ADE of the CSOBUG and existing work. With a low impact on the existing works, we have noticed a considerable improvement in the results from Figure 4(a). ACO-RMS approaches have variations of 9.65-19.23%, IQCQP approaches have variations of 20.09-27.27%, and eACO-MSPD approaches have variations of 24.69-34.34%. Similarly, Figure 4(b) is a comparison of the performance variations in time-driven WSN applications between the CSOBUG algorithm and available algorithms. EC is reduced by around 2.22-6.49×, 4.1-7.35×, and 6.3-11.87× by using ACO-RMS, IQCQP, and eACO-MSPD approaches over the proposed one. The proposed CSO algorithm for determining the optimal clusters is responsible for the improvement in the results of the proposed CSOBUG algorithm. SNs are also able to minimize their energy consumption by using weight functions and order selection.

5.3 *Fairness index of energy drain*

For a specific reasoning behind the WSNs and an inability to identify them through the AED metric, the fairness index ($\mathcal{F}$) is used to identify the energy drain bottleneck. This range of $\mathcal{F}$ can be found in the range [0, 100], where the higher the value, the better the result, and vice versa. This calculation was based on Hoßfeld et al. [37], as shown in equation (15).

$$\mathcal{F} = \left(1 - \frac{\zeta \times 2}{\max\{E_a^t\} - \min\{E_a^t\} + \epsilon}\right) \times 100 \tag{15}$$

where ζ is the standard deviation of energy drain, $\max\{E_a^t\}$ indicates the maximum AED at time t, and $\min\{E_a^t\}$ is the minimum AED at time t. In the event of similar max and min energies, ϵ prevents the [divided by zero] error. In order to determine the ζ value, we apply equation (16).

$$\zeta = \frac{1}{\sqrt{n}} \sqrt{\sum_{i=1}^{n} (E_i^c - E_a^t)^2} \tag{16}$$

Based on two scenarios, event- and time-driven, the $\mathcal{F}$ of the proposed CSOBUG is compared with the $\mathcal{F}$ of existing works. SNs are vary between 500 and 1,000 for these two scenarios.

The performance of the CSOBUG algorithm is significantly better than that of the ACO-RMS algorithm, ranging from 0.116 to 0.489%. A similar relationship exists between IQCQP and eACO-MSPD, with range from 0.276-0.448× and 0.379-6.487×, respectively, being lower than CSOBUG. WSNs were also generated according to time-driven parameters using Figure 5(b). The CSOBUG algorithm has shown significant improvements in terms of $\mathcal{F}$ over the

existing approaches that range from 0.134-0.489×, 0.236-0.501×, and 0.474-0.699× as compared to ACO-RMS, IQCQP, and eACOMSPD. By implementing CSO, $\mathcal{F}$ has been clustered efficiently, and VP selection has been improved.

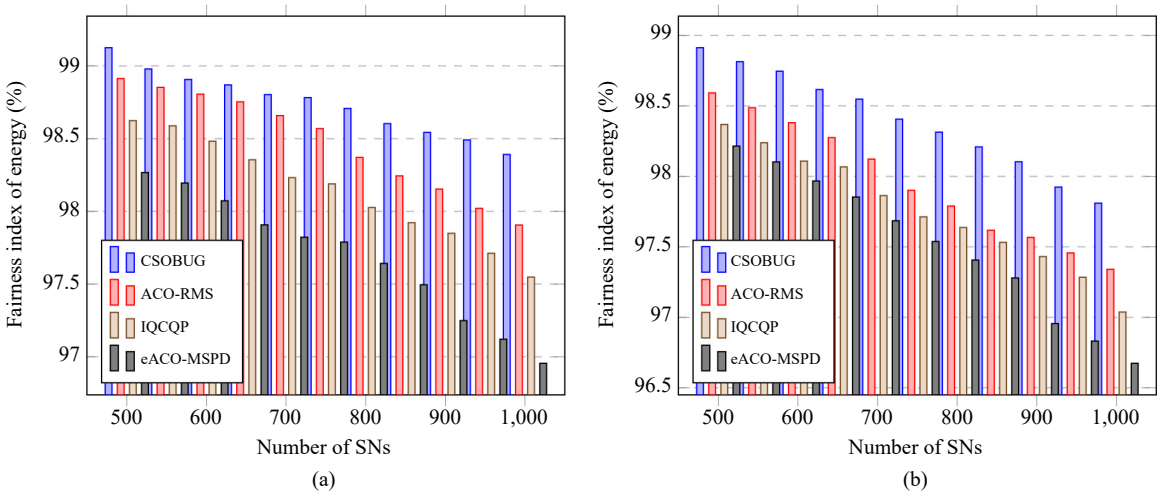


Figure 5. Fairness index of energy drain of WSNs in (a) event-driven and (b) time-driven

5.4 *Buffer utilization (BU)*

Sensing data accumulated during a single MS cycle is known as data packets, and they occupy a certain amount of buffer space in order to accumulate them from the network environment and be used during the next MS cycle. By using this metric, it can be defined as a metric that defines how efficient the process of collecting data is and how efficient the use of resources can be. The BU ultimately determines how much data is collected, the amount of data lost in the network, and the efficiency of the data collection method. There is a formula that can be used for this, which is given in equation (17).

$$BU = \frac{\sum_{i=1}^{n} \mathcal{B}_i}{\mathcal{B}_{\max} \times n} \tag{17}$$

For event-based and time-dependent WSNs, the BU of the CSOBUG algorithm is estimated by varying the number of SNs between 500 and 1,000. Even though CSOBUG outperforms existing applications, even time-driven applications perform better. ACO-RMS's performance compares poorly with that of the proposed CSOBUG algorithm in event-oriented WSN applications shown in Figure 6(a). This is approximately 1.02-2.05% better than the performance of CSOBUG compared to ACO-RMS. Furthermore, CSOBUG is better by 1.53-3.86% than IQCQP and by 2.96-5.24% than eACO-MSPD. We see improvement variations in BU performances in comparison to published and recommended approaches by CSOBUG by at least 1.53% in Figure 6(b). By approximately 1.31-1.67%, 2.24-3.62%, and 4.49-6.21%, CSOBUG outperforms ACO-RMS, IQCQP, and eACO-MSPD approaches. It can be concluded from these analyses that the proposed CSOBUG algorithm organizes the buffer efficiently in order to prevent network data loss and congestion. MS are scheduled efficiently, and VP data is accumulated timely, maximizing memory use.

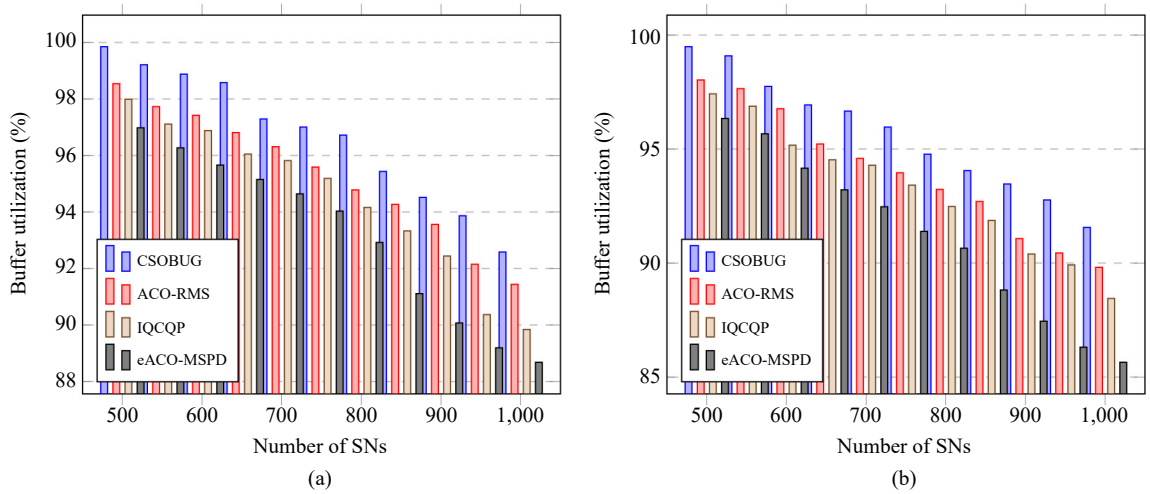


Figure 6. BU of (a) event-driven WSNs and (b) time-driven WSNs

5.5 *Network throughput*

Network throughput (Γ) is calculated by dividing the number of packets acquired during simulation time (T) by the number of packets acquired during T. A mathematical representation of it is shown in equation (18).

$$\Gamma = \frac{1}{T}\sum_{i=1}^{n}|P_i| \tag{18}$$

where $|P_i|$ is the number of packets available at $\S_i$ at time T by the BS.

Now, we estimate the Γ by varying the n value between 500-1,000 in Figure 7(a). With a maximum of 0.8%, 3.14%, and 4.50%, respectively, CSOBUG has improved Γ over ACO-RMS, IQCQP, and eACO-MSPD algorithms. ACO-RMS, IQCQP, and eACO-MSPD algorithms, as shown in Figure 7(b), have a minimum Γ of 0.04%, 0.01%, and 0.03% higher than CSOBUG, respectively. Thus, CSOBUG and existing strategies perform better in terms of throughput based on these numerical analyses. It has been observed that the improved Γ of CSOBUG is a result of the bug approach used to determine the near-optimal travel route for MS in the WSNs. According to the proposed algorithm, the maximum throughput in the WSNs indicates efficient data accumulation and low congestion.

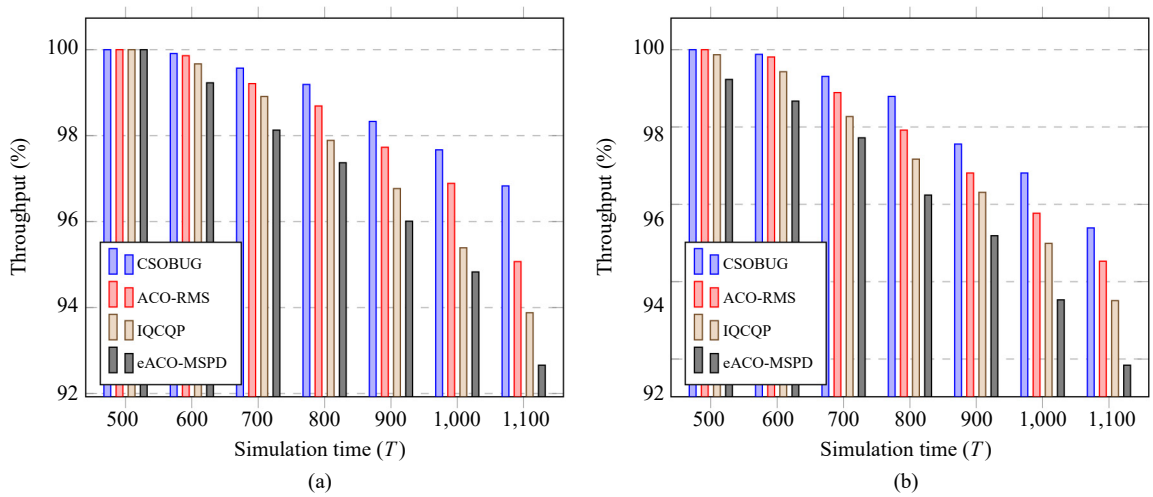


Figure 7. Throughput of (a) event-driven WSNs and (b) time-driven WSNs

5.6 *End-to-end (E2E) delay*

Data packets returned to the BS (S_0) are delayed (in milliseconds) by the E2E delay (Δ) in this research. Algorithms with lower E2Es perform more efficiently. From the time packets are generated to the time they reach the BS, the delay Δ is calculated using the formula equation (19).

$$\Delta = \frac{\sum_{i=1}^{n}\sum_{j=1}^{P_i}(\varphi_{ij}^{w} + \varphi_{ij}^{m})}{n \times P_i} \tag{19}$$

where φ_{ij}^{m} is the time await of jth packet of node i at MS, and φ_{ij}^{w} is the waiting time of jth packet of node i at SN, before it is accumulated by the MS and after it is acquired from the environment. P_i indicates the total packets generated by node i. A number of existing approaches are evaluated, including the Δ of CSOBUG and CSO parameter evaluation, as well as the integration of time-driven WSNs into the paper.

Continuing our testing, we switch from event- to time-driven scenarios and vary the number of SNs. Based on the results shown in Figure 8(a), we understand how the E2E delay varies among the different algorithms when they are applied to event-driven WSN applications, and we also see that the CSOBUG has a lower delay than the other algorithms. The proposed CSOBUG performs significantly better than the existing ACO-RMS, IQCQP, and eACO-MSPD, with approximate performance improvements of 1.54-3.41×, 1.82-4.48×, and 4.18-6.19×, respectively. In time-driven applications, we estimate the E2E delay of WSNs with 100 SNs in Figure 8(b). Although the CSOBUG algorithm performs better than the existing ACO-RMS, IQCQP, and eACO-MSPD strategies, which are 1.87-4.19×, 2.11-5.09×, and 4.91-7.43×, respectively. As a result of efficient tour construction between the VPs, the performance of CSOBUG is superior to the performance of the ACO-RMS, IQCQP, and eACO-MSPD approaches. Using an efficient route will lead to a shorter distance and faster accumulation of data, as well as delivering the data packets to the BS on time, which will result in a lower overall delay of the data packets in the WSN.

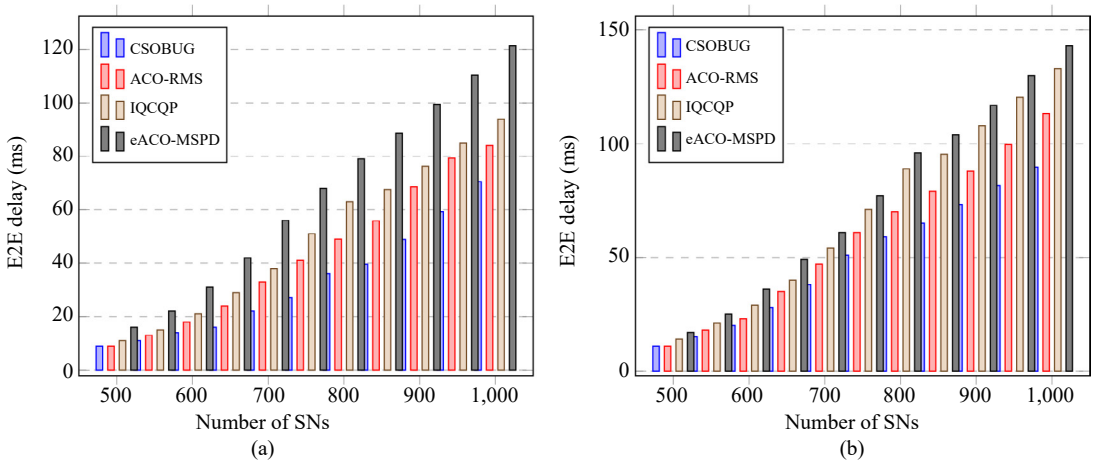


Figure 8. E2E delay of (a) event-driven WSNs and (b) time-driven WSNs

5.7 *Average MS travel length*

In MS-based data accumulation, the average tour length (ATL) of an MS is an important metric since a longer MS results in slower delay, throughput, and buffering. The tour length of an MS is extremely difficult to trade-off; however, minimizing it is beneficial for achieving low E2E delays, high BUs, and fast speeds. To determine whether other performance metrics influence ATL of MS, we evaluate ATL of MS similar to Donta et al. [33].

According to Figure 9, the ATL is presented under both event- and time-driven WSN deployment scenarios, with SNs deployed randomly in both scenarios. Accordingly, the number of SNs in the event-driven scenario ranges from 500

to 1,000, as shown in Figure 9(a). In the proposed CSOBUG approach, the minimum ATL is 1.76 km and the maximum is 3.331 km. In the same way, the minimum ATLs for ACO-RMS, IQCQP, and eACO-MSPD approaches are 2046.8, 2119.9, and 2165.8 m, respectively. ACO-RMS can achieve a maximum ATL of 3620.44 m, IQCQP can reach 3915.7 m, and eACO-MSPD can reach 4013.4 m. Time-driven WSN application scenarios require a minimum ATL of 2.08 km for the proposed CSOBUG and 2.13 km for ACO-RMS, IQCQP, and eACO-MSPD, respectively, as shown in Figure 9(b). The maximum ATLs for CSOBUG, ACO-RMS, IQCQP, and eACO-MSPD algorithms are 3.56 km, 3.75 km, 3.92 km, and 4.26 km, respectively. Based on the optimal clustering of the SNs and the optimal selection of VPs, the proposed CSOBUG yields the minimum ATL over the existing works.

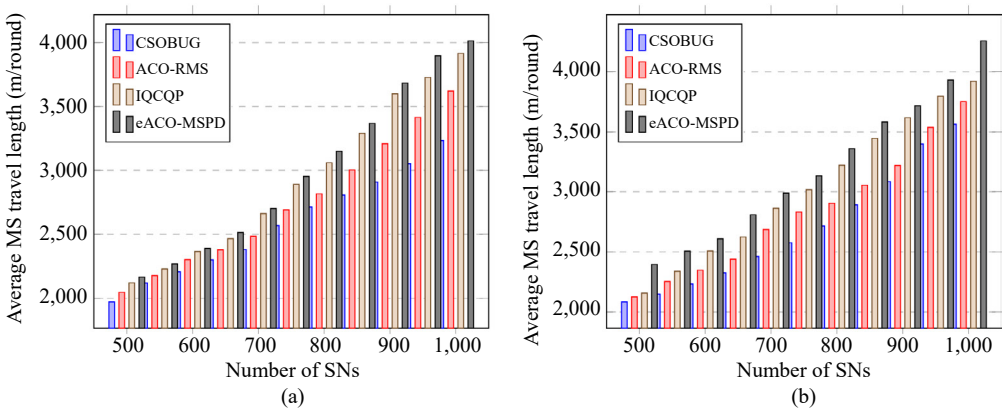


Figure 9. Average MS travel length of (a) event-driven WSNs and (b) time-driven WSNs

6. Conclusion

This paper proposes a CSO-based algorithm for VP selection and a bug algorithm for obstacle-aware path construction for MS in WSNs. Unlike the existing approach, this approach is very efficient in finding the most appropriate VPs and an optimal path that is not too long or short and is treated as optimal. In comparison to existing algorithms such as ACO-RMS, IQCQP, and eACO-MSPD, these optimal results improve the proposed CSOBUG. Moreover, this method is also less computational than these approaches. Besides, this approach cannot identify obstacles, whereas the authors need to indicate their locations. Furthermore, the work can be extended to consider different scenarios and identify obstacles more quickly. There was also an extension of this work to other applications where different obstacles such as buildings, roads, and trees were employed. The authors have also proposed the use of this approach in different environments, such as urban and rural areas.

Conflict of interest

There is no conflict of interest in this study.

References

[1] Kumar DP, Amgoth T, Annavarapu CSR. Machine learning algorithms for wireless sensor networks: A survey. *Information Fusion*. 2019; 49: 1-25. Available from: https://doi.org/10.1016/j.inffus.2018.09.013.

[2] Yu S, Zhang B, Li C, Mouftah HT. Routing protocols for wireless sensor networks with mobile sinks: A survey. *IEEE Communications Magazine*. 2014; 52(7): 150-157. Available from: https://doi.org/10.1109/MCOM.2014.6852097.

[3] Donta PK, Srirama SN, Amgoth T, Annavarapu CSR. Survey on recent advances in IoT application layer protocols

and machine learning scope for research directions. *Digital Communications and Networks*. 2022; 8(5): 727-744. Available from: https://doi.org/10.1016/j.dcan.2021.10.004.

[4] Wang Y-C, Chen K-C. Efficient path planning for a mobile sink to reliably gather data from sensors with diverse sensing rates and limited buffers. *IEEE Transactions on Mobile Computing*. 2019; 18(7): 1527-1540. Available from: https://doi.org/10.1109/TMC.2018.2863293.

[5] Donta PK, Amgoth T, Annavarapu CSR. Delay-aware data fusion in duty-cycled wireless sensor networks: A Q-learning approach. *Sustainable Computing: Informatics and Systems*. 2022; 33: 100642. Available from: https://doi.org/10.1016/j.suscom.2021.100642.

[6] Srinivas M, Donta PK, Amgoth T. Finding the minimum number of mobile sinks for data collection in wireless sensor networks. In: *2020 IEEE International Conference on Communication, Networks and Satellite* (*Comnetsat*). IEEE; 2020. p.256-260. Available from: https://doi.org/10.1109/Comnetsat50391.2020.9328947.

[7] Kumar DP, Tarachand A, Rao ACS. ACO-based mobile sink path determination for wireless sensor networks under non-uniform data constraints. *Applied Soft Computing*. 2018; 69: 528-540. Available from: https://doi.org/10.1016/j.asoc.2018.05.008.

[8] Redhu S, Hegde RM. Cooperative network model for joint mobile sink scheduling and dynamic buffer management using Q-learning. *IEEE Transactions on Network and Service Management*. 2020; 17(3): 1853-1864. Available from: https://doi.org/10.1109/TNSM.2020.3002828.

[9] Nandan AS, Singh S, Awasthi LK. An efficient cluster head election based on optimized genetic algorithm for movable sinks in IoT enabled HWSNs. *Applied Soft Computing*. 2021; 107: 107318. Available from: https://doi.org/10.1016/j.asoc.2021.107318.

[10] Najjar-Ghabel S, Farzinvash L, Razavi SN. Mobile sink-based data gathering in wireless sensor networks with obstacles using artificial intelligence algorithms. *Ad Hoc Networks*. 2020; 106: 102243. Available from: https://doi.org/10.1016/j.adhoc.2020.102243.

[11] Donta PK, Dustdar S. The promising role of representation learning for distributed computing continuum systems. In: *2022 IEEE International Congress on Intelligent and Service-Oriented Systems Engineering (SOSE)*. IEEE; 2022. Available from: https://doi.org/10.1109/SOSE55356.2022.00021.

[12] Al-Janabi TA, Al-Raweshidy HS. A centralized routing protocol with a scheduled mobile sink-based AI for large scale I-IoT. *IEEE Sensors Journal*. 2018; 18(24): 10248-10261. Available from: https://doi.org/10.1109/JSEN.2018.2873681.

[13] Yadav RK, Mahapatra RP. Hybrid metaheuristic algorithm for optimal cluster head selection in wireless sensor network. *Pervasive and Mobile Computing*. 2022; 79: 101504. Available from: https://doi.org/10.1016/j.pmcj.2021.101504.

[14] Kamble AA, Patil BM. Systematic analysis and review of path optimization techniques in WSN with mobile sink. *Computer Science Review*. 2021; 41: 100412. Available from: https://doi.org/10.1016/j.cosrev.2021.100412.

[15] Lin Z, Keh H-C, Wu R, Roy DS. Joint data collection and fusion using mobile sink in heterogeneous wireless sensor networks. *IEEE Sensors Journal*. 2021; 21(2): 2364-2376. Available from: https://doi.org/10.1109/JSEN.2020.3019372.

[16] Donta PK, Amgoth T, Annavarapu CSR. An extended ACO-based mobile sink path determination in wireless sensor networks. *Journal of Ambient Intelligence and Humanized Computing*. 2021; 12: 8991-9006. Available from: https://doi.org/10.1007/s12652-020-02595-7.

[17] Mehto A, Tapaswi S, Pattanaik KK. PSO-based rendezvous point selection for delay efficient trajectory formation for mobile sink in wireless sensor networks. In: *2020 International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE; 2020. p.252-258. Available from: https://doi.org/10.1109/COMSNETS48256.2020.9027330.

[18] Wen W, Shang C, Chang C-Y, Roy DS. DEDC: Joint density-aware and energy-limited path construction for data collection using mobile sink in WSNs. *IEEE Access*. 2020; 8: 78942-78955. Available from: https://doi.org/10.1109/ACCESS.2020.2989763.

[19] Naghibi M, Barati H. EGRPM: Energy efficient geographic routing protocol based on mobile sink in wireless sensor networks. *Sustainable Computing: Informatics and Systems*. 2020; 25: 100377. Available from: https://doi.org/10.1016/j.suscom.2020.100377.

[20] Jain S, Pattanaik KK, Verma RK, Bharti S, Shukla A. Delay-aware green routing for mobile-sink-based wireless sensor networks. *IEEE Internet of Things Journal*. 2021; 8(6): 4882-4892. Available from: https://doi.org/10.1109/JIOT.2020.3030120.

[21] Jain S, Pattanaik KK, Verma RK, Shukla A. EDVWDD: Event-driven virtual wheelbased data dissemination for mobile sink-enabled wireless sensor networks. *The Journal of Supercomputing*. 2021; 77: 11432-11457. Available from: https://doi.org/10.1007/s11227-021-03714-7.

[22] Farzinvash L, Najjar-Ghabel S, Javadzadeh T. A distributed and energy-efficient approach for collecting emergency data in wireless sensor networks with mobile sinks. *AEU - International Journal of Electronics and Communications*. 2019; 108: 79-86. Available from: https://doi.org/10.1016/j.aeue.2019.06.007.

[23] Gupta GP, Saha B. Load balanced clustering scheme using hybrid metaheuristic technique for mobile sink based wireless sensor networks. *Journal of Ambient Intelligence and Humanized Computing*. 2022; 13: 5283-5294. Available from: https://doi.org/10.1007/s12652-020-01909-z.

[24] Fu X, He X. Energy-balanced data collection with path-constrained mobile sink in wireless sensor networks. *AEU - International Journal of Electronics and Communications*. 2020; 127: 153504. Available from: https://doi.org/10.1016/j.aeue.2020.153504.

[25] Chang C-Y, Chen S-Y, Chang I-H, Yu G-J, Roy DS. Multirate data collection using mobile sink in wireless sensor networks. *IEEE Sensors Journal*. 2020; 20(14): 8173-8185. Available from: https://doi.org/10.1109/JSEN.2020.2981692.

[26] Gutam BG, Donta PK, Annavarapu CSR, Hu Y-C. Optimal rendezvous points selection and mobile sink trajectory construction for data collection in WSNs. *Journal of Ambient Intelligence and Humanized Computing*. 2023; 14: 7147-7158. Available from: https://doi.org/10.1007/s12652-021-03566-2.

[27] Kumar R, Amgoth T, Das D. Obstacle-aware connectivity establishment in wireless sensor networks. *IEEE Sensors Journal*. 2021; 21(4): 5543-5552. Available from: https://doi.org/10.1109/JSEN.2020.3032144.

[28] Yang Y, Yang W, Wu H, Miao Y. A mobile sink-integrated framework for the collection of farmland wireless sensor network information based on a virtual potential field. *International Journal of Distributed Sensor Networks*. 2021; 17(7). Available from: https://doi.org/10.1177/15501477211030122.

[29] Xie G, Pan F. Cluster-based routing for the mobile sink in wireless sensor networks with obstacles. *IEEE Access*. 2016; 4: 2019-2028. Available from: https://doi.org/10.1109/ACCESS.2016.2558196.

[30] Park J, Kim S, Youn J, Ahn S, Cho S. Iterative sensor clustering and mobile sink trajectory optimization for wireless sensor network with nonuniform density. *Wireless Communications and Mobile Computing*. 2020; 2020: 8853662. Available from: https://doi.org/10.1155/2020/8853662.

[31] Selvaraj S, Vasanthamani S. Energy efficient dynamic routing mechanism (EEDRM) with obstacles in WSN. *Wireless Personal Communications*. 2020; 112(4): 2761-2776. Available from: https://doi.org/10.1007/s11277-020-07174-9.

[32] Sah DK, Cengiz K, Donta PK, Inukollu VN, Amgoth T. EDGF: Empirical dataset generation framework for wireless sensor networks. *Computer Communications*. 2021; 180: 48-56. Available from: https://doi.org/10.1016/j.comcom.2021.08.017.

[33] Donta PK, Rao BSP, Amgoth T, Annavarapu CSR, Swain S. Data collection and path determination strategies for mobile sink in 3D WSNs. *IEEE Sensors Journal*. 2020; 20(4): 2224-2233. Available from: https://doi.org/10.1109/JSEN.2019.2949146.

[34] McGuire KN, de Croon GCHE, Tuyls K. A comparative study of bug algorithms for robot navigation. *Robotics and Autonomous Systems*. 2019; 121: 103261. Available from: https://doi.org/10.1016/j.robot.2019.103261.

[35] Ahmed AM, Rashid TA, Saeed SAM. Cat swarm optimization algorithm: A survey and performance evaluation. *Computational Intelligence and Neuroscience*. 2020; 2020: 4854895. Available from: https://doi.org/10.1155/2020/4854895.

[36] Kafi MA, Djenouri D, Ben-Othman J, Badache N. Congestion control protocols in wireless sensor networks: A survey. *IEEE Communications Surveys & Tutorials*. 2014; 16(3): 1369-1390. Available from: https://doi.org/10.1109/SURV.2014.021714.00123.

[37] Hoßfeld T, Skorin-Kapov L, Heegaard PE, Varela M. Definition of QoE fairness in shared systems. *IEEE Communications Letters*. 2017; 21(1): 184-187. Available from: https://doi.org/10.1109/LCOMM.2016.2616342.