Mobile Element Scheduling with Dynamic Deadlines

Arun A. Somasundara, Aditya Ramamoorthy, *Member*, *IEEE*, and Mani B. Srivastava, *Senior Member*, *IEEE*

Abstract—Wireless networks have historically considered support for mobile elements as an extra overhead. However, recent research has provided the means by which a network can take advantage of mobile elements. Particularly in the case of wireless sensor networks, mobile elements can be deliberately built into the system to improve the lifetime of the network and act as mechanical carriers of data. The mobile element, whose mobility is controlled, visits the nodes to collect their data before their buffers are full. In general, the spatio-temporal dynamics of the sensed phenomenon may require sensor nodes to collect samples at different rates, in which case, some nodes need to be visited more frequently than others. This work formulates the problem of scheduling the mobile element in the network so that there is no data loss due to buffer overflow. The problem is shown to be NP-complete and an Integer-Linear-Programming formulation is given. Finally, some computationally practical algorithms for a single mobile and for the case of multiple mobiles are presented and their performances compared.

Index Terms—Controlled mobility, scheduling, wireless sensor networks.

1 Introduction

Recently, there has been an increased focus on the use of sensor networks to sense and measure the environment. This leads to a wide variety of theoretical and practical issues on appropriate protocols for data sensing and transfer. Some practical deployments include the Networked InfoMechanical System (NIMS) [1], [2], James Reserve [3], and Great Duck Island [4]. All of these deployments focus mainly on the problem of habitat and environment monitoring.

One can also envisage scenarios where a sensor network is used to sense pollution levels at strategic locations in a large city. There will be regions in which the variation in pollution levels will be greater, such as industrial areas as compared to residential areas. To capture this behavior, the sensing rates of sensors at different positions will typically need to be different. Consequently, the sensor nodes in regions with higher variation in the phenomenon need to sample more frequently in order to have more reliable statistics. Another example of a practical scenario where the sensing rates of sensors are different is a heterogenous sensor network. For instance, a network deployed for the purposes of ecological study will have sensors of different modalities like light, humidity, pressure, etc. The different types of sensors will typically be sampling at different rates, depending on the physical phenomenon.

- *A.A. Somasundara is with Broadcom Corporation, 3151 Zanker Road, San Jose, CA 95134. E-mail: aruns@broadcom.com.*
- *A. Ramamoorthy is with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011.*
- *M.B. Srivastava is with the Electrical Engineering Department, University of California Los Angeles, MC #951594, 6731-H Boelter Hall, Los Angeles, CA 90095-1594. E-mail: mbs@ee.ucla.edu.*

Manuscript received 9 Sept. 2005; revised 16 Apr. 2006; accepted 27 June 2006; published online 15 Feb. 2007.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-0269-0905.

The data sensed by the sensor nodes needs to be transferred to a base station, where it can be analyzed by the field experts. Usually, the readings taken by the sensor nodes are relayed to a base station for processing using the ad hoc multihop network formed by the sensor nodes. While this is surely a feasible technique for data transfer, it creates a bottleneck in the network. In addition to sensing and transmitting their data, the nodes near the base station have to relay the data from nodes that are farther away. This leads to a nonuniform depletion of network resources and the nodes near the base station will be the first to run out of batteries. If these nodes die, the network is useless for all practical purposes as the sensed data cannot be transmitted to the base station. Periodically replacing the battery of the nodes is infeasible for large-scale deployments.

Researchers have proposed mobility as a solution to this problem of data gathering. Mobile elements, acting as base stations, can traverse the network, collecting data from sensor nodes when they come near them. Existing mobility in the environment can be used [5], [6], [7], [8] or mobile elements can be added to the system [9], [10], [11], which have the luxury of being recharged periodically. This naturally avoids multihop and removes the relaying overhead of nodes near the base station. In addition, the sensor nodes no longer need to form a connected network (in a wireless sense). Thus, a network can be deployed, keeping only the sensing aspects in mind. One need not worry about adding nodes just to ensure that data transfer remains feasible.

Various types of mobility (for the mobile element) that can be used for data collection in a sensor network can be broadly classified as follows:

1. *Random Mobility*. The base station can be mounted on entities moving randomly. These entities are usually those existing in nature. For instance, in the

1536-1233/07/$25.00 © 2007 IEEE Published by the IEEE CS, CASS, ComSoc, IES, & SPS

concept mentioned in [5], humans and animals act as "data mules" and collect data opportunistically from sensor nodes when in range.

2. *Predictable Mobility*. In this case, we do not have any control on the movement of the mobile element, but we can predict its motion. This approach is investigated in [8], where a network access point was mounted on a public transportation bus moving with a periodic schedule. The sensor nodes learn and eventually predict the times at which they have connectivity with the bus and wake up accordingly to transfer their data. Another example of this is the scheme presented in [12], where there is a mobile sink which does not send any query. Instead, the static sensor nodes which are a single hop from the path of the mobile sink (called moles) learn its movement pattern over time and statistically characterize it as a probability distribution function. Whenever a static sensor node needs to send data, it sends the data toward the mole, which should be in the vicinity of the mobile sink.
3. *Controlled Mobility*. Here, the mobile element is part of the network infrastructure and its motion can be controlled for networking needs. Our earlier work [9] presented an implementaion of a system with a robot acting as a mobile base station. The robot moves on a predetermined path, but changes its speed depending on the quality of the wireless channel and density of sensor nodes in a particular area. Thus, the job of the mobile node is exclusively one of a data-gatherer. In addition to this, recently, there has been an emergence of sensor network systems which have a controlled mobile element in them (with mobile element performing a variety of tasks). Two examples include NIMS [2] and Ragobot [13].

Controlled mobility was used in Message Ferrying [10], [11], where the ferry is used to route messages between nodes in sparse networks. A mobile base station was also used in [14] to increase network lifetime. Their technique leads to a more uniform distribution of energy consumption by repeatedly relocating the base station, which changes the bottleneck nodes which are closest to the base station and results in the burden of relaying being shared across the entire network. The two works above were evaluated in simulation.

In this paper, we investigate **scheduling problems** that naturally come up when sensor networks operate under this paradigm of **controlled mobility**. We consider a sensor network that has sensor nodes in different areas operating at different sampling rates. This is in line with our motivational example of pollution sensors. Each sensor node has a finite buffer for storing the sensed values. The network is equipped with a mobile element that does the job of data gathering. When the mobile element visits a sensor node, it transfers the data to its own memory and the sensor node's memory is freed. After visiting a node, the mobile element will visit some other node, empty its buffer, and so on. A static sensor node, after being visited by the mobile element and having its buffer cleared, continues sampling at its assigned rate and starts to fill its memory buffers again. A problem that naturally crops up is the scheduling of the visits of the mobile element so that buffers on none of the sensor nodes overflow. We call this the **Mobile Element Scheduling (MES)** problem.

It is important to clearly outline the differences between this problem and the conventional Traveling Salesman Problem (TSP) [15]. In TSP, the goal is to find a minimum cost tour that visits each node exactly once. However, in our problem, a node may need to be visited multiple times before all other nodes are visited depending on the strictness of its deadline, i.e., frequency of sampling. In addition, as soon as a node is visited, its deadline (time before which it should be revisited to avoid buffer overflow) is updated. Thus, deadlines are "dynamically" updated as the mobile element performs the job of data gathering.

Our earlier work [16] considered the use of a single mobile element and its scheduling. This paper extends it in many directions. We analyze the heuristics presented in it. We extend the algorithms to the case when there are multiple mobile elements to be scheduled. The presence of multiple mobile entities leads to a number of design tradeoffs in the solution space. In addition, we consider an existing piece of work in a related area and enhance it to solve the Mobile Element Scheduling problem.

There are other scenarios where this scheduling problem arises. One such system is a mobile battery charger unit that visits sensor nodes periodically to charge their batteries. An example of such a system is [17], which presented energy harvesting in a mobile sensor network. The goal is to schedule visits so that every sensor node is visited before it is fully discharged. The batteries may be discharging at different rates, depending on the tasks that the sensor nodes are assigned to do or because of different qualities of batteries. Obviously, a node can be visited before it is fully discharged, but, once visited, it should come back before its discharge time.

Calibration of sensor nodes is an important problem [18]. Another potential example is a mobile element used for calibrating the sensor nodes. The sensor nodes drift at different rates. The mobile element visits the nodes to calibrate them and the mobile's visits need to be scheduled.

In addition to the three examples above (mobile element used for data collection, battery charging, and calibration), mobility is being utilized in sensor networks in other ways which require some form of scheduling of the mobile element. For instance, in the NIMS system [2] mentioned in the beginning, the mobile NIMS node is equipped with sensing capability and moves to build a spatio-temporal map of the sensed phenomenon. Scheduling is essential when an event-aware [19] sampling is desired instead of a raster scan. Another example is the Cyclops-based [20] sensor network, where the nodes have a smart vision sensor. Image summary can be transmitted wirelessly, whereas a mobile element can be used to extract the raw images from the cyclops nodes in non-real-time for archival purposes.

We will be using the data collection example in the rest of the paper. Also, there are no real-time constraints, only that there should be no buffer overflow. This paper is organized

as follows: Related work is presented in Section 2, which compares this problem with existing ones. Section 3 provides a formal statement of our problem. We show that the problem is NP-complete in Section 4. An Integer-Linear-Programming formulation is presented in Section 5, followed by Section 6, which provides some heuristic solutions that yield good results. The algorithms for multiple mobiles are presented in Section 7.1, which extends the algorithms for a single mobile. We also consider existing solutions to a related problem in Section 7.2 and show that a suitable modification to it can solve our problem. Simulation methodology and results are discussed in Section 8. Section 9 presents a discussion on the practicality of the scheduling algorithms. Finally, we conclude in Section 10, outlining some directions for future work.

2 Related Work

This section presents the work related to the scheduling problem discussed above. The message ferrying approach [10], [11] deals with using a message ferry to route data from one node to another in a sparse network. In particular, [10] gives schemes to find the route that the ferry has to take. Based on a given traffic matrix (expected traffic from any node to any other node), the goal is to find the optimal route of the ferry so that the average delay from source to destination is minimized while meeting the bandwidth requirements of the traffic. Our problem deals with data gathering and the constraint is on buffer overflow. We present below some relevant literature in routing and scheduling theory.

2.1 Vehicle Routing Problem

In the Vehicle Routing Problem (VRP) [21], we are given a set of nodes and their service requests (in terms of demand for a certain quantity of goods). There are vehicles available for servicing these requests which are stationed at a special node ($node_0$), which is referred to as the depot. Each vehicle has a certain capacity (in terms of the quantity of goods it can carry). The goal is to find the number of vehicles and the sequence of nodes each vehicle has to visit such that the sum of the distances traveled by the vehicles is minimal, subject to the following constraints:

- Each vehicle starts and ends at the depot.
- Each node is serviced by exactly one vehicle.
- Capacity constraints are met, i.e., the sum of demands from the nodes on a vehicle's list does not exceed the vehicle's capacity.

The Traveling Salesman Problem (TSP) mentioned earlier is a special case of VRP, where there is a single vehicle which visits the nodes and there are no capacity constraints.

There are many variants to the basic VRP mentioned above. The one relevant to our problem is VRP with Time Windows (VRPTW). In VRPTW [22], in addition to the VRP constraints, there is a time window within which each node has to be visited. A special case of this is Deadline-TSP [23], in which case there is only one vehicle. But, both are different from our problem, as, in our case, before visiting all the nodes once, some node (which is sampling at a higher rate) may need to be visited more than once. In addition, the time windows of the visits are not known a priori. Only the current time window is known and the next time window is decided based on the current visit time.

There is a dynamic version of VRP, known as the Dynamic Vehicle Routing Problem (DVRP) [24]. Here, the information (input) is revealed to the decision maker concurrently with the determination of routes. For instance, the VRP mentioned above is solved for the initial requests (nodes to be visited) and routes assigned to vehicles. Then, as the vehicles start their scheduled routes, new requests come which need to be accommodated. The DVRP problem with time window constraints (DVRPTW) has a time window for the new requests. A Tabu search-based heuristic is presented in [25], where the time windows are not strict, i.e., if a node cannot be visited in its requested time window, there is a lateness penalty. The goal is to minimize the weighted sum of travel costs and lateness penalties.

Another related problem is the Dynamic Traveling Repairperson Problem (DTRP) [26], where there is a group of autonomous vehicles and targets are generated stochastically which request service. The goal is to minimize the expected waiting time of the service requests. Decentralized algorithms for motion coordination of the autonomous vehicles are presented. The difference from our problem is that there are no deadlines in DTRP.

2.2 Processor Scheduling

Our problem can also be looked at from the view of processor scheduling. Here, tasks arrive periodically, have an execution time, and need to be finished before their deadline. We have two analogies of this to our problem. The internode travel time can be considered as the context switch time and servicing time at a node can be considered as the execution time. Another way of looking at it is that the travel time is the execution time (Worst-Case Execution Time (WCET) will be the time to reach from its farthest neighbor). A very important distinction in our problem is that the next instance of the task is released (generated) as soon as the previous instance is serviced. This automatically rules out schemes based on static priority assignment, such as Rate Monotonic scheduling [27]. Dynamic schemes such as Earliest Deadline First [27] (called deadline-driven scheduling in the reference) are feasible approaches and are discussed in future sections. Finally, there is no notion of preemption in our problem.

3 Problem Formulation

The Mobile Element Scheduling (MES) problem with a single mobile can be formulated in the following manner. We are given:

- A fully connected graph of n nodes: $node[1 \ldots n]$.
- A matrix $cost[1 \ldots n][1 \ldots n]$ that denotes the time taken to go from one node to another.
- A vector that contains buffer overflow times, $overflow\_time[1 \ldots n]$. The ith element of this vector determines the time to fill the buffer of the ith node. This can be computed using the buffer size and sensing rate.
- A starting node $node_0$.

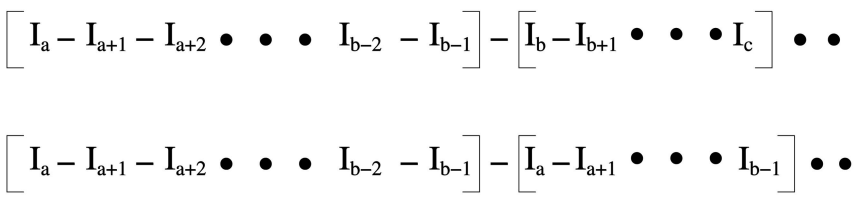


Fig. 1. Construction of a periodic schedule from a given schedule S.

We make the following assumptions:

- The matrix, $cost[1 \ldots n][1 \ldots n]$, and the vector, $overflow\_time[1 \ldots n]$, consist of integer entries.
- At time $t = 0$, all the buffers of the sensor nodes start filling up.
- The actual data transfer time from the sensor node to the mobile element is negligible.

The

$$\textbf{Mobile-Element-Schedule}(cost[1 \ldots n][1 \ldots n], overflow\_time[1 \ldots n], node_0)$$

problem is the problem of finding a sequence of visits to nodes from $node[1 \ldots n]$ starting at $node_0$ so that none of the buffers of the nodes overflow. Once a node is visited, the deadline for its next visit is updated. For example, suppose that node$[k]$ is visited at time t_k and that its overflow time is $overflow\_time[k]$, then the new deadline for $node[k]$ will be $t_k + overflow\_time[k]$.

4 Proof of NP-Completeness

In this section, we shall prove that the problem of deciding whether a valid schedule exists is NP-complete. The proof relies on a reduction from the Hamiltonian Cycle problem. Before embarking on the actual proof, we need a lemma that proves that, if a schedule exists for a given instance of the problem, we can derive a periodic schedule from it in polynomial time.

Lemma 1. *Suppose that we are given an instance of*

$$\text{Mobile-Element-Schedule}(cost[1 \ldots n][1 \ldots n], overflow\_time[1 \ldots n], node_0)$$

that has a solution S, i.e., a schedule such that no node's buffer overflows. Then, a periodic schedule can be derived from S in polynomial time so that, if the periodic schedule is followed, then none of the buffers will ever overflow.

Proof. Let us first compute the maximum of all the overflow times, i.e., let

$$T_O = \max(overflow\_time[k]), k \in [1..n]. \quad (1)$$

S is some sequence of numbers and letters $x_1, x_2, \ldots$, where each $x_i \in \{1, 2, \ldots, n\} \cup M$ denotes the state of the mobile element at time $t = i$. Thus, at any given time, the mobile element is either at one of the sensor nodes or it is in the mobile state M (it is moving toward another sensor node).

The crucial observation is that, if we look at any time window of length T_O in the sequence S, all sensor nodes have to occur at least once. To see this, assume that there exists a node v and a time window $[t, (t + T_O)]$ such that v does not occur in it. This means that, between successive visits to v, at least time T_O elapses, which means that its buffer would surely overflow (note that T_O is the maximum overflow time), which is a contradiction since S is assumed to be a valid schedule.

Now, suppose that we start observing and recording the sequence S in intervals of length T_O, starting at some time t_1, i.e., we record the solution in time intervals $[t_1, t_1 + T_O]$, $[t_1 + T_O + 1, t_1 + 2T_O + 1], \ldots$. The maximum number of distinct types of such intervals is $(n + 1)^{T_O+1}$ since there are $(T_O + 1)$ time instants in an interval and each instant can be labeled with a number from $\{1, 2, \ldots, n\}$ or the letter M. It follows that there exist two intervals, I_a and I_b (without loss of generality, we can assume that I_b comes after I_a), that will be exactly the same if we observe the sequence from t_1 to $t_1 + (n + 1)^{T_O+1}$.

Now, if S is not periodic, we observe that a valid periodic schedule can be constructed, as shown in Fig. 1. Let the original sequence be

$$I_a, I_{a+1}, \ldots, I_{b-2}, I_{b-1}, I_b, I_{b+1}, I_{b+2} \ldots.$$

Since $I_b = I_a$, therefore, the set of deadlines of all the nodes at the end of I_b will be exactly the same as at the end of I_a. Since we know that S is a valid schedule, therefore, we can be sure that none of the buffers overflowed in intervals $I_{a+1}, I_{a+2}, \ldots, I_{b-1}$ and, thus, by repeating them after I_b, we can be sure that none of the buffers overflow. Thus, we have found a new schedule that is periodic. □

We now show that the Mobile-Element-Schedule problem is NP-complete.

Theorem 1. *The*

$$\text{Mobile-Element-Schedule}(cost[1 \ldots n][1 \ldots n], overflow\_time[1 \ldots n], node_0)$$

problem is NP-complete.

Proof. The proof is in two parts:

1. To see that the problem is in NP, we observe that, if we are given a schedule S_1 that is to be verified, by Lemma 1, it is clear that a maximum of $(n + 1)^{T_O+1}$ successive entries of S_1 need to be examined to make sure that the schedule S_1 is indeed valid. Thus, verifying the validity of a schedule can be done in polynomial time.
2. To prove that the problem is NP-hard, we reduce the problem of finding a Hamiltonian Cycle in an

arbitrary graph $G(V, E)$ to the Mobile-Element-Schedule problem. This problem is well-known to be NP-complete [28].

Let $G(V, E)$ be an instance of the Hamiltonian Cycle problem. We construct an instance of Mobile-Element-Schedule as follows:

$$\text{cost}[i][j] \;=\; \begin{cases} 1 & \text{if } (i,j) \in E \\ 2 & \text{otherwise,} \end{cases} \tag{2}$$

$$\text{overflow\_time}[i] \;=\; n, \forall I, \tag{3}$$

$$\text{node}_0 \;=\; 1. \tag{4}$$

If the constructed instance of Mobile-Element-Schedule returns a valid schedule S, then we observe the following:

a. Let x_t denote the state of the mobile element at time t, where $t \neq 0$. In an interval $[\text{t} + 1, \ldots, \text{t} + \text{n}]$, all nodes are visited at least once by an argument similar to the one in Lemma 1. Since all nodes have $overflow\_time = n$, this means that all nodes are visited exactly once since there are exactly n time instants, $(t + 1), \ldots, (t + n)$, and the minimum cost between any two nodes is 1.
b. If we let x_i denote the state of the mobile element when $i \in \{(t + 1), (t + 2), \ldots, (t + n)\}$, then $cost[x_i][x_{i+1}] = 1$. This is because, if there exists an i such that $cost[x_i][x_{i+1}] = 2$, then we cannot fit n nodes in n time slots and will cause at least one node not to satisfy the constraint. This means that the internode travel times are 1, i.e., all these edges are also in G (the instance of the Hamiltonian Cycle problem).
c. $x_{t+n} = x_t$. To see this, suppose $x_{t+n} = k \neq x_t$; then, k must have appeared somewhere in $(t + 1), (t + 2), \ldots, (t + n - 1)$, otherwise its buffer would overflow, but this is a contradiction since we know that each node appears exactly once in $(t + 1), \ldots, (t + n)$ from item a.

But then, the sequence $x_t, x_{t+1}, \ldots, x_{t+n}$ is a valid Hamiltonian cycle in G.

Thus, we have shown that $G(V, E)$ contains a Hamiltonian cycle **if and only if** the above-constructed instance of Mobile-Element-Schedule has a valid solution. Combining the two parts (1 and 2), we have shown that our problem is NP-complete. □

5 ILP Formulation

We now present an Integer-Linear-Programming (ILP) formulation to gain additional insight into the Mobile-Element-Scheduling problem. Linear Programming (LP) problems are optimization problems in which the objective function and the constraints are all linear. If the unknown variables are all required to be integers, then the problem is called an integer programming (IP) or integer linear programming (ILP) problem. We proved in the last section that the schedule is periodic, giving an upper bound to the period. Let the period be T. An ILP formulation consists of variables and constraints on the variables.

Variables:

- x_{ij}: $i \in \{1 \ldots T\}$, $j \in \{1 \ldots n\}$.
  - $x_{ij} = 1$ if, at time i, the mobile element is at node j; 0, otherwise.
- y_i: $i \in \{1 \ldots T\}$.
  - $y_i = 1$ if, at time i, the mobile element is moving; 0, otherwise.

It is obvious that, if we are able to obtain the values of $x_{ij}, \forall i, j$ and $y_i, \forall i$, we can reconstruct the schedule for the problem. We now lay down the constraints that these variables have to follow so that we can obtain a valid schedule.

Constraints:

- At time i, the mobile element is either at some sensor node or it is moving. Therefore,

$$\sum_{j=1}^{n} x_{ij} + y_i = 1, \forall i. \tag{5}$$

- The maximum allowed time between visits to a sensor node j is $overflow\_time[j]$. So,

$$\left.\begin{array}{c} \sum_{i=0}^{overflow\_time[j]} x_{ij} \geq 1 \\ . \\ . \\ . \\ \sum_{i=T-overflow\_time[j]}^{T} x_{ij} \geq 1 \end{array}\right\} \forall j. \tag{6}$$

- Finally, we need a constraint that forces the mobile element to be in the mobile state between visits to two sensor nodes for at least the time determined by the cost matrix. We have

$$\sum_{t=i}^{k} y_t \geq [x_{ij} + x_{kl} - C] \times \frac{cost[j][l]}{2 - C}, \tag{7}$$

$\forall i, k \in \{1 \ldots T\}$, $i < k$, and $j, l \in \{1 \ldots n\}$, $j \neq l$ and C is a constant such that $1 < C < 2$. We can explain this constraint as follows:

  - Suppose $x_{ij} = 1$, $x_{kl} = 1$. This means that, at time i, the mobile element was at sensor node j and, at time k, it was at sensor node l. The RHS of this constraint is then just $cost[j][l]$ and the constraint enforces the fact that it should have taken at least this time to move. (We assume that, between visits to successive nodes, there is at least one time unit when the mobile element is moving. We can handle the pathological case of cost between two nodes $= 1$ by an appropriate discretization of time.)
  - Otherwise, i.e., if either or both of x_{ij}, x_{kl} are 0, then there is no constraint that needs to be enforced. Note that, in this case, the RHS of the

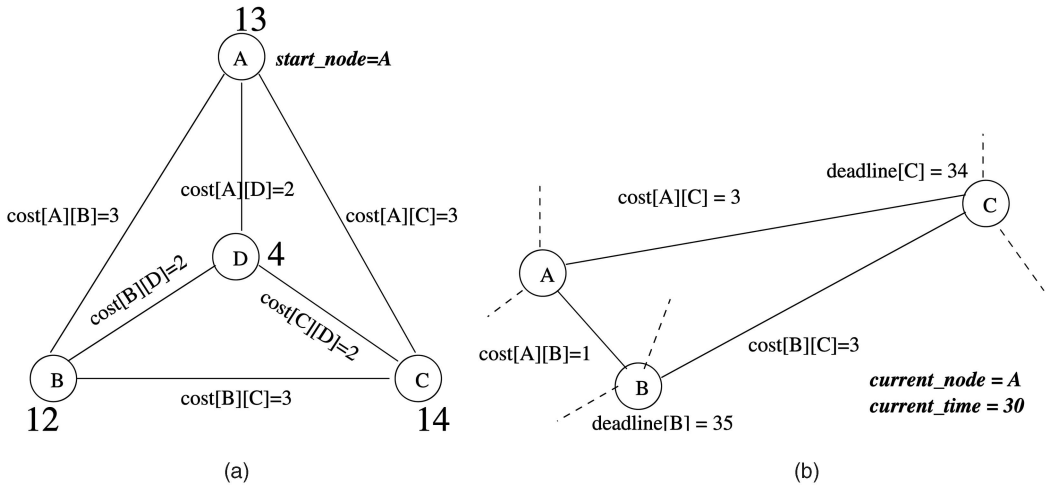


Fig. 2. Earliest Deadline First (EDF) examples. (a) Choosing the next node different from $current\_node$. (b) EDF type of scheduling is not the best.

constraint is negative and, since $y_i \in \{0, 1\}$, it is trivially satisfied.

Thus, the scheduling problem reduces to finding the existence of a feasible set of x_{ij}s and y_is.

In general, we will not know the exact period T beforehand and some amount of experimentation will be required. A systematic approach would be to perform a binary search on T. If the period T is large, the total number of constraints for the ILP would be large and this method may not be computationally practical. Nevertheless, the formulation is presented to provide some insight into the problem.

6 Computationally Practical Solutions

In this section, we present some heuristics for the solution of the Mobile-Element-Scheduling problem. For the kind of dynamic scheduling we need to do, Earliest Deadline First (EDF) would be a natural fit. Here, the node with the closest deadline is visited first.

ALGORITHM: Earliest Deadline First (EDF)

- Input: $cost[1 \ldots n][1 \ldots n]$, $overflow\_time[1 \ldots n]$, $start\_node$
- Initialize:
  $current\_time = 0$, $current\_node = start\_node$,
  $deadline[1..n] = overflow\_time[1..n]$
- Main: Repeat the following
  1. Choose the node $i \neq current\_node$ whose deadline is closest
  2. If $deadline[i] < current\_time + cost[current\_node][i]$
     - Declare failure and stop
  3. Else
     - $current\_time += cost[current\_node][i]$
     - $current\_node = i$
     - $deadline[i] = current\_time + overflow\_time[i]$

END

First, we explain the reason for not choosing the $current\_node$ as the next one. Consider Fig. 2a, with the values on edges indicating the $cost$s (which are symmetric) and those near the nodes indicating their $overflow_time$s. This is an example of a case where there is a radial sensor field such that node D needs to sample at a higher rate than the other nodes. Suppose that the $start\_node$ is A. The first part of Table 1 shows the sequence of visits, where the constraint $i \neq current\_node$ is not enforced. This results in node A missing its deadline. If we did not stay at D, even though it had the earliest deadline, we would get a sequence of visits as shown in the second part of Table 1. Thus, with the constraint of visiting some node other than the current one, we could get the schedule $A, D, B, D, A, D, C, D, \ldots$, and none of the nodes would miss their deadlines. Conversely, it is not possible that not staying at a node caused it to miss its deadline, whereas staying would have prevented it. This is because the mobile has to leave the current node sometime later, if not now, to service other nodes. As a result, the deadline of the current node will be violated at that later time, when the mobile leaves it.

One obvious shortcoming of this algorithm is that it does not take into account the $cost$ values and relies only on deadlines. For instance, consider Fig. 2b, which shows part of a network. Suppose the mobile element has just visited node A and the current time is 30. Various parameters are as shown in the figure. The EDF algorithm will choose to visit node C next as its deadline is closest. Then, it will visit node B at time 36. Clearly, the deadline of B is missed. On the contrary, had it visited node B first and then node C, both of the deadlines would have been met.

This example suggests that one way to account for the $cost$ in addition to deadlines is to have a lookahead.

6.1 EDF with *k*-Lookahead

Instead of going to a node whose deadline is earliest, we can, for instance in the above example, consider two earliest deadline nodes and visit that node so that the deadlines of both the nodes are met. Generalizing this, in k-lookahead, we can consider the $k!$ permutations of the k earliest deadline nodes. Suppose we are at node x_0 and the next k earliest deadline nodes are $x_1, x_2, \ldots, x_k$. We will choose that permutation which leads to none of the k nodes missing their deadlines. There may be many such possible permutations. If so, we will choose the one which leads to x_{k+1} the earliest. The precise algorithm is presented below:

TABLE 1
Analysis of the Example in Fig. 2a

current_time	node	New *deadline* {A,B,C,D}
0	A	13,12,14,4
2	D	13,12,14,6
3	D	13,12,14,7
...	...	...
8	D	13,12,14,12 let us choose to visit B
10	B	13,22,14,12
12	D	13,22,14,16 **A** misses its deadline.

current_time	node	New *deadline* {A,B,C,D}
0	A	13,12,14,4
2	D	13,12,14,6
4	B	13,16,14,6
6	D	13,16,14,10
8	A	21,16,14,10
10	D	21,16,14,14
12	C	21,16,26,14
14	D	21,16,26,18

ALGORITHM: EDF with k-lookahead

- Input: k, $cost[1 \ldots n][1 \ldots n]$, $overflow\_time[1 \ldots n]$, $start\_node$
- Initialize $current\_time$, $current\_node$, $deadline[1 \ldots n]$ as before.
- Main: Repeat the following:
  1. Sort $deadline[1 \ldots n]$ in increasing order.
  2. Using the first k entries:
     - Find an ordering of these k entries so that
       a. none of the k nodes miss their deadlines in the next k steps,
       b. the arrival time of the node at the $(k+1)$th entry is minimal, and
       c. the first node in the resulting permutation is not the $current\_node$.
     - If none exists, declare failure and stop.
  3. Let the first node in the ordering found be i.
     - $current\_time += cost[current\_node][i]$
     - $current\_node = i$
     - $deadline[i] = current\_time + overflow\_time[i]$

END

It is important to note that we are not scheduling k visits at a time, but, instead, for each visit, we are looking at k nodes and choosing only the next node. The reason for doing so is that it may happen that a node i has a very low $overflow\_time[i]$ value. The schedule will look something like x_a, x_i, x_b, x_i, x_c, x_i, where the node x_i is required to be revisited soon after visiting any other node. Now, if we schedule k nodes at a time, we will not be able to achieve this result and the deadline of x_i will be surely missed.

The special case of $k = 1$ reduces to the EDF algorithm presented before. The set whose permutations are considered has only one element and, hence, only one choice for the next step. Note that the lookahead algorithm takes care of nodes with same $deadline$ values, whereas EDF would have chosen randomly depending on in what order they appeared in the sorted array.

6.2 Minimum Weighted Sum First (MWSF) Heuristic

The lookahead algorithm, in addition to deadline, takes cost into account when making a decision by seeing into the future. Instead, an algorithm can be designed which gives weights to deadlines and cost and goes to the node which has the minimum weighted sum. The value of deadline to be considered in the weighted sum is not the absolute value, but relative to current time, i.e., $deadline[i] - current\_time$, for the node i.

ALGORITHM: Minimum Weighted Sum First

- Input: Weight $\alpha_{mwsf} \in (0, 1]$, $cost[1 \ldots n][1 \ldots n]$, $overflow\_time[1 \ldots n]$, $start\_node$.
- Initialize $current\_time$, $current\_node$, $deadline[1 \ldots n]$ as before.
- Main: Repeat the following:
  1. $\forall i$, calculate,
     $weighted\_sum[i] = \alpha_{mwsf} * (deadline[i] - current\_time) + (1 - \alpha_{mwsf}) * cost[current\_node][i]$
  2. Choose the node $i \neq current\_node$ whose $weighted\_sum[i]$ is minimal.
     - If $deadline[i] < current\_time + cost[current\_node][i]$.
       * Declare failure and stop.
     - Else
       * $current\_time += cost[current\_node][i]$
       * $current\_node = i$
       * $deadline[i] = current\_time + overflow\_time[i]$

END

Table 2 shows the effect of different α_{mwsf} values. To illustrate the contents of the table, suppose we are at node x_i and two nodes x_a and x_b have costs 25 and 50, respectively, and relative deadlines 200 and 175, respectively. Clearly, x_a is the closer node and x_b has an earlier deadline. When $\alpha_{mwsf} < 0.5$, x_a will be chosen, and when $\alpha_{mwsf} > 0.5$, x_b will be chosen. When $\alpha_{mwsf} = 0$, when at a node, the mobile will go to the one which is closest to the current node. In the limit, the mobile will end up in a locally minimum set of smallest-cost nodes and will be revolving among the nodes in that set. All other nodes miss their deadline.

Combining the previously mentioned two approaches of lookahead and using the minimum weighted sum, we can perform lookahead on the weighted sum metric.

6.3 Discussion

For the EDF with lookahead, at each step, we first sort the nodes based on deadline. This takes $O(n \log n)$ time. Doing a lookahead in k-lookahead is $O(k)$, where we check if the deadlines of all k nodes are met. This is done for all the

TABLE 2
Effect of α_{mwsf} Values on the Minimum Weighted Sum First Heuristic

$\alpha_{mwsf} \in [0,1]$	Result
1	Weight to deadlines only. Same as EDF.
0	Weight to cost only. Results in back and forth motion between a locally minimum set of smallest-cost nodes. All other nodes miss their deadlines.
small	Higher priority to closer nodes
big	Higher priority to closer deadlines

$k!$ permutations. Thus, the total complexity of the algorithm for finding a next node to visit from the current node is $O(n \log n + k.k!)$. Each step of MWSF takes $O(n)$ for calculating the weighted sum for all nodes and $O(n)$ for choosing the minimum, resulting in $O(n)$. Optimally, the time to find a schedule if one exists requires n-lookahead, which is too high to be computationally practical.

It would be easier if there were some necessary and sufficient conditions to check the existence of a schedule.

- *Necessary Condition*. From Lemma 1 in Section 4, a necessary condition for a schedule to exist is the existence of a solution for $\mathrm{TSP}(T_O)$, where T_O was defined to be

  $$\max(overflow\_time[i]), i \in [1 \ldots num\_nodes].$$

  TSP(C) is the decision version of TSP and has a solution if there is a Hamiltonian cycle of length at most C. This is because, if TSP(C) does not have a solution, we cannot expect to have a solution to our problem as there are $overflow\_time$ values which are less than this. Since this is an even tighter constraint, it rules out the possibility of a solution for MES.
- *Sufficient Condition*. A sufficient condition for a schedule to exist is the existence of a solution for $\mathrm{TSP}(\min(overflow\_time[i]))$, $i \in [1 \ldots num\_nodes]$. All nodes having the same minimum overflow time is the strictest case. If a solution exists for this, we surely have a solution for MES as all overflow times are the same or larger than $\min(overflow\_time[i])$.

It may be argued that, for a checking necessary and sufficient condition for the existence of a solution to our problem which is NP-complete, we are using another NP-complete problem. However, we note that there is a large body of literature that deals with designing efficient heuristics for TSP [15], which can be leveraged for this purpose.

7 Mobile Element Schedule with Multiple Mobiles

We saw in the previous section with a single mobile that a feasible schedule (leading to no buffer overflow at any of the nodes) may not always exist. We now discuss the algorithms for Mobile-Element-Scheduling when there are multiple mobiles. First, we describe how the algorithms in the previous section can be adapted to handle multiple mobiles in Section 7.1. We then describe a well-known heuristic for solving the VRPTW (Vehicle Routing Problem with Time Windows) problem and demonstrate that a suitable modification of the heuristic can handle the case of multiple mobiles in Section 7.2. Finally, we compare the approaches in Section 7.3.

7.1 Adapting the Algorithms for Single Mobile

The EDF with lookahead and Minimum Weighted Sum First algorithms can be used when more than one mobile is present in the system. For this, the problem formulation of Section 3 needs to be slightly modified. All mobiles are initially stationed at the same location, say, the center of the area (and not at a particular node as earlier).

7.1.1 Divide the Area into Equal Parts

If the nodes are distributed uniformly at random, we can divide the area into parts of equal area, with the number of parts being the same as the number of mobiles. Then, each mobile is responsible for the area it is assigned to and can follow the single Mobile-Element-Scheduling algorithms presented in the previous section. Henceforth, this will be called Scheme-1.

7.1.2 Combined Scheduling

A second alternative would be to remove the restriction of having each mobile responsible for only a part of the area. Suppose there are n nodes and m mobiles $(m << n)$. Initially, we find a node to be visited by the first mobile (using either EDF with lookahead or Minimum Weighted Sum First). Then, we find a node to be visited by the second mobile. For this, we consider the $n-1$ remaining nodes. We continue this for all m mobiles, ending with each mobile headed toward one node. This leaves $n-m$ nodes yet to be scheduled.

When a mobile visits the node it is scheduled to visit, we run the algorithm on $n-m$ nodes to find the next node for this mobile. We do this whenever a mobile visits a node. We will label this as Scheme-2.

There seem to be trade-offs in the two approaches mentioned above. For instance, in the first case, on average, each mobile is responsible for only $\frac{n}{m}$ nodes, whereas each mobile has a global view in the second case. We will evaluate both possibilities in detail in Section 8.

7.2 Vehicle Routing Problem with Time Windows

There is a large amount of literature for the Vehicle Routing Problem. We have given a brief introduction to this earlier in Section 2.1. In the Vehicle Routing Problem with Time Windows (VRPTW), there is a time window $[a, b]$ constraint for each node. The node cannot be serviced before a and has to be serviced before b. Since the VRP is NP-hard, by restriction, VRPTW is NP-hard. Furthermore, [29] has shown that finding a feasible schedule when the number of vehicles is fixed is itself an NP-complete problem. As a result, the development of heuristic algorithms for this problem class has been of interest. In this section, we will first briefly explain the Insertion heuristic of Type-1 proposed in [22]. We use this algorithm as it is one of the most cited algorithms for VRPTW and used as a benchmark

in the literature for comparison whenever any new algorithm for VRPTW is proposed. We then present a modification that can be used to handle our problem. The terms "vehicle" and "mobile" are used interchangeably, as are "node" and "customer."

7.2.1 Insertion Heuristic for VRPTW

This heuristic builds the tours sequentially, vehicle by vehicle [22]. It starts with a single vehicle and initializes a node to be on its tour (leading to the schedule list depot-node-depot for the current vehicle). It then finds the best node to be inserted into the tour of the current vehicle and its corresponding location. This continues till no nodes (which are yet to be scheduled) can be inserted at any location. At this point, a new vehicle is called to service and the procedure is repeated. Thus, the number of vehicles is also the result of the algorithm. If a vehicle reaches a node before its beginning time window (a), it will wait.

Two items are to be specified in the above described algorithm:

1. An algorithm for finding the first node to be inserted when a new vehicle is called for service. The following criteria for choosing the first node on a vehicle's route is specified in [22]:
   - either the farthest unrouted node, or
   - the unrouted node with the earliest deadline.

   After initializing the current route (route for the current vehicle) with a node according to one of the above criteria, in every iteration, the method inserts a new node u into the current partial route.
2. The definition of cost.

We now define the cost functions used in [22] for making the choice of node to be inserted and its location. It uses four parameters, μ, α_1, α_2, and λ. Let $(i_0, i_1, \ldots, i_z)$ be the current route, with $i_0 = i_z = 0$ (the depot). For each unrouted customer u, first compute its best feasible insertion place in the emerging route. Let i, j be adjacent nodes in the current partial route. For u being inserted between i and j, define two cost functions,

$$c_{11}(i,u,j) = cost_{iu} + cost_{uj} - \mu * cost_{ij}, \mu \geq 0, \tag{8}$$

$$c_{12}(i,u,j) = time_{j_u} - time_j, \tag{9}$$

where $cost_{ab}$ is the cost of going from node a to b. $time_{j_u}$ is the new time for service to begin at node j, given that u is on the route. $time_j$ is the time at which the service begins at node j prior to insertion. $c_{11}(i,u,j)$ is a measure of the extra cost incurred if u is inserted between i and j. $c_{12}(i,u,j)$ is a measure of extra time (delay in visiting node j due to visit of u before it). The best feasible insertion place for node u is defined to be the one that minimizes the weighted combination of its distance and time insertion. For this, define

$$\begin{aligned} &c_1(i,u,j) = \alpha_1 c_{11}(i,u,j) + \alpha_2 c_{12}(i,u,j), \\ &where\ \alpha_1 + \alpha_2 = 1, \alpha_1 \geq 0, \alpha_2 \geq 0. \end{aligned} \tag{10}$$

It should be noted that inserting u between i and j could potentially alter all the times to begin service at customers k (which follow j in the existing route) and may also make the route infeasible. In such a case, this insertion spot is discarded.

At the end of this, we can calculate the best insertion spot for u as being between that pair of adjacent nodes i, j, which minimizes (10). For the node u, these adjacent nodes are denoted by $i(u)$ and $j(u)$. Thus, we have

$$c_1(i(u),u,j(u)) = \min[c_1(i_{p-1},u,i_p)], p = 1,2,\ldots,z. \tag{11}$$

The above procedure gives us the best insertion spot for each unrouted customer u. Next, we need to find the best node u to be inserted. For this, define

$$c_2(i,u,j) = \lambda * cost_{0u} - c_1(i,u,j), \lambda \geq 0. \tag{12}$$

We need to choose the u that maximizes $c_2(i,u,j)$. For each u, we already know the best (i,j) from c_1 (given by $i(u), j(u)$). So, it suffices to evaluate c_2 at only these pairs.

$$c_2(i(u),u,j(u)) = \lambda * cost_{0u} - c_1(i(u),u,j(u)), \lambda \geq 0. \tag{13}$$

Intuitively, this is a measure of the benefit derived from servicing a customer on a partial route being constructed rather than on a direct route (depot-u-depot). This benefit has to be maximized. Thus, we can obtain the best node to be inserted, u^*, as the node for which c_2 is highest.

$$c_2(i(u^*),u^*,j(u^*)) = \max[c_2(i(u),u,j(u))]. \tag{14}$$

One iteration of the algorithm concludes when such a u^* is found and inserted between $i(u^*)$ and $j(u^*)$. When no node with feasible insertion can be found, the method starts a new route (with a new vehicle) unless all nodes have been scheduled.

In the upcoming sections, we fix the four inherent parameters, μ, α_1, α_2, and λ of VRPTW to 1, 1, 0, and 0, respectively.

7.2.2 Modifying VRPTW to Solve the Mobile-Element-Scheduling (MES) Problem

The above-mentioned Insertion heuristic (of Type-1) can be enhanced to solve MES. In MES, we are given a set of nodes with $overflow\_time$ values. If a node i is serviced at time t, it should be visited again before it fills its buffer. This implies that the node can be visited any time in the window $[t, t + overflow\_time[i]]$. Ideally, it would be better for the node to be revisited as close to the upper time window $(t + overflow\_time[i])$ as possible. This is because, if it is visited earlier, the node would not have generated sufficient data and would require more frequent visits.

Define a parameter, $\alpha_{vrptw} \in [0,1]$. The algorithm consists of the following steps (given an α_{vrptw} value):

1. Solve the VRPTW algorithm of the previous section with initial time windows

   $$\left[(1 - \alpha_{vrptw}) * overflow\_time[i]),\ overflow\_time[i]\right]. \tag{15}$$

   This results in a set of mobiles and a list of nodes to be visited by each.

TABLE 3
Effect of α_{vrptw} on Schedule

$\alpha_{vrptw} \rightarrow$	Closer to 0	Closer to 1
Waiting time at the nodes	High	Low
Number of visits to the nodes	Less	More
Distance traveled by the mobiles	Less	More

2. Whenever a node is visited by a mobile (mobile reaches the first node in its list), say at time t,
   a. Remove the node from the list.
   b. Create a new service request for this node with time window given by

$$[t + (1 - \alpha_{vrptw}) * overflow\_time[i], \quad t + overflow\_time[i]]. \tag{16}$$

   c. Find a mobile and the insertion spot to insert this request. For each mobile, first find the best insertion spot given by (11) (where u is the node just visited and whose new request is being inserted). Insert it in that mobile's list which has the least cost. It may be possible that there is no feasible insertion spot in any of the mobiles' existing schedules. For this, we can do one of two things:
      - Call a new mobile for service.
      - Use the mobile (and the corresponding insertion spot) which gives the least amount of deadline overflow. Overflow happens for the node being inserted and/or the nodes following it in the list.

Let us look at the effect of α_{vrptw} on the schedule presented in Table 3. In the table, waiting time refers to the time the mobile has to wait if it reaches a node before its lower limit of time window. When α_{vrptw} is closer to 0, the time window is small, or rather, the lower limit of the time window is high (the upper limit of the time window is the same, irrespective of the parameter value). As the node is to be visited closer to the upper limit of the time window, time is spent waiting at nodes when the mobiles reach their scheduled nodes early. Over a certain period of time, the number of visits to a node is less. Also, this implies that the mobiles travel a shorter distance in a given total time (as time is spent in waiting).

7.3 Discussion

There is a basic difference in the approaches followed by the algorithms of the previous sections. In EDF with lookahead and Minimum Weighted Sum First (using either Scheme-1 or Scheme-2), each mobile is assigned one node. On visiting a node, the mobile is assigned the next node to be visited. On the contrary, in VRPTW and its modified version for MES, each mobile maintains a list of nodes it needs to visit. On visiting a node, the node is inserted into the existing schedule of one of the mobiles (or a new mobile is called for service).

Let us look at the complexity of the various algorithms. Suppose there are n nodes and m mobiles. The complexity of EDF with k-lookahead is $O(\frac{n}{m}\log\frac{n}{m} + k.k!)$ for finding a next node for a mobile to visit when it visits a node with Scheme-1. It is $O((n-m)\log(n-m) + k.k!)$ with Scheme-2. Similarly, the complexity of Minimum Weighted Sum First is $O(\frac{n}{m})$ when using Scheme-1 and $O(n-m)$ with Scheme-2.

The complexity of the VRPTW algorithm (Section 7.2.1) is $O(n^4)$. There are n nodes to be scheduled. In each iteration, (13) is evaluated for each unrouted node. For this, (11) is evaluated for each possible insertion spot. Finally, for any given insertion spot, all nodes following it need to be tested if the current insetion is feasible. Thus, there are four nested loops, each being $O(n)$. After this initial schedule, the modified algorithm (Section 7.2.2) takes $O(m.n^2)$ time at each step whenever a node is visited and it needs to be inserted into the existing schedule of one of the mobiles. This is because there are m mobiles, insertion in whose schedules needs to be evaluated. For each mobile, there can be $O(n)$ insertion spots and, for each insertion spot, all nodes following this spot need to be checked if they miss their deadline due to the insertion of this node.

We note that the ability to add mobiles when they are required in the VRPTW-based algorithm lets us handle the case of temporal changes in overflow times, i.e., $overflow\_time[i]$ changing as the system evolves. This is clearly useful when the dynamics of the phenomenon change. A node that was earlier sampling at a lower rate may now be required to sample faster. First, the node informs the scheduling entity of the new value of $overflow\_time$. This is similar to the case in the Dynamic Vehicle Routing Problem (DVRP) [24] or the Dynamic Traveling Repairperson Problem (DTRP) [26], where the entity generating the request calls the central office, for instance, to schedule a pickup or request a repair. When the node's $overflow\_time$ is updated and the scheduling entity gets this information, the node might have already sampled for some amount of time. Using the older sampling rate and $overflow\_time$ in addition to the new values, we can know when in the future its buffer will be filled (say, $future\_fill\_time$). Now (unless this node is being headed toward by one of the mobiles), we can delete this node from the schedule of the mobile it was previously inserted into and insert a new request for this node with time window $[current\_time, current\_time + future\_fill\_time]$. It may have to be inserted at a different location from where it was deleted and the visit times of other nodes may change or there may be a need to add a new mobile too. Once this node is visited for the first time after $overflow\_time$ changed, we can use the main procedure.

8 Simulation Methodology and Results

This section presents the evaluation of the algorithms presented in the previous two sections. Though we evaluate them in simulation, the various parameters used are for real systems. Consider a packbot [30] used as a mobile base station, which moves comfortably at $1\ m/s$ (from practical experience, as it was used in our earlier work [9]). The commonly used wireless sensor nodes are mica2 motes [31].

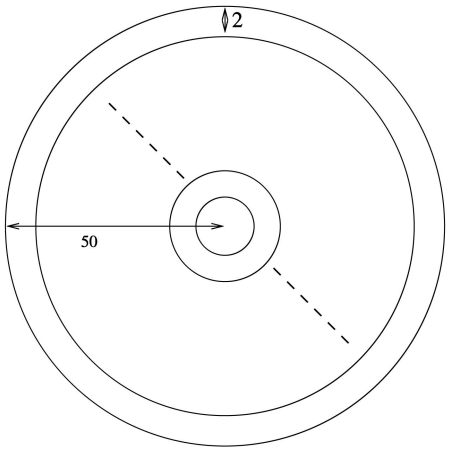


Fig. 3. Topology for evaluating MES.

Consider a sampling phenomenon which requires the motes to sense and store one sample every second and suppose that 4 bytes are required to store each sample. These motes have 4 kB of RAM, of which let us use 2 kB for storing the samples. Thus, the buffer will overflow in about 500 seconds, and the node needs to be visited by the mobile before this time. We assume a realistic deployment area of $500\,m$ radius. An important point to be noted here is that it is the ratio of *cost* and $overflow\_time$ which impacts the schedule. In simulations mentioned next, we use the values which closely relate (ratio wise) to the above-mentioned numbers.

8.1 Simulation Methodology

As the parameter space is huge, we fix some of the values:

- *Topology.* We consider a circular area of radius 50 units. The speed is taken to be 1, so the distance between two nodes is the same as *cost* values.
- *Simulation Time*. We simulated for 100,000 time units.
- Location and number of nodes n. The nodes are placed uniformly at random in the area. We use 100 nodes.
- $overflow\_time$ values. One option was to assign these values randomly to the nodes. But, to simulate a real-world situation, we assumed that the point of interest is located in the center of the topology and the nodes closer to the center have smaller $overflow\_time$ values as they are sampling more frequently. We placed concentric circles, the smallest one being of radius 2. Also, the radius increased by 2 from one circle to the next. This is shown in Fig. 3. The innermost region had the smallest $overflow\_time$, called $basic\_overflow\_time$, and the regions radially outward were a constant factor of it, i.e., $\{1, 1.2, 1.3, \ldots\} * basic\_overflow\_time$. We use three values for $basic\_overflow\_time$: 50, 75, and 100.
- Unless mentioned otherwise, all results are averaged over 25 topologies.

For the purposes of evaluation, instead of stopping the algorithm when a node missed its deadline, we continue, noting this fact and updating its deadline. Thus, in a simulation time of 100,000, each node i would have been visited $num\_visits[i]$ times, of which $num\_deadline\_misses[i]$ times the deadline would have missed. We calculate the ratio $100 \times \frac{num\_deadline\_misses[i]}{num\_visits[i]}$. This ratio is averaged across all the nodes. We call this **percentage failure** and use it as a metric for evaluation.

Another related metric is the **amount of overflow**, which is the amount of time by which the deadline was missed. For this, we calculate the total amount of time by which the mobile was late whenever a node's deadline was missed. This is averaged across all the nodes.

Latency is an important metric in wireless data collection. We define latency as the time taken between the generation of a sample and the time of collection by the mobile element. This is averaged over all the samples, across all the nodes. However, we would like to mention that the model of data collection considered in this paper is non-real-time in nature. As a result, there are no latency constraints as long as there is no buffer overflow.

8.2 Mobile Element Scheduling with Single Mobile

Our earlier work [16] presented in detail the results for the algorithms with a single mobile element. The key findings were:

1. Minimum Weighted Sum First performed better than EDF with lookahead (evaluated with lookahead up to 7).
2. The α_{mwsf} parameter for which Minimum Weighted Sum First performed best varied with topology. As the algorithm is $O(n)$, the best α_{mwsf} can be found by experimentation, by running the algorithm for all possible values of the parameter. For the topologies and distribution of $overflow_time$s considered, α_{mwsf} around 0.1 gave good results.
3. There were no additional gains on doing lookahead over a set of least weighted sum nodes.

For the detailed results of scheduling with single mobile, we refer the reader to [16]. Here, we evaluated the effect of the α_{mwsf} parameter on the results of the Minimum Weighted Sum First heuristic. We noticed that it is hard to conclude about the dependence of α_{mwsf} value on the result (percentage failure). (Plots omitted due to lack of space.)

8.3 Mobile Element Scheduling with Multiple Mobiles

Let us look at the results of scheduling when there are multiple mobiles present. Due to space constraints, we do not show the percentage failure metric out of the metrics presented earlier. We noticed that the relative behavior of percentage failure among the algorithms and parameters was almost similar to the amount of the overflow metric.

Consider the VRPTW algorithm presented in Section 7.2. Fig. 4a presents the number of mobiles required in the static solution (presented in Section 7.2.1) using (15). Note that VRPTW inherently is not dynamic. We explicitly mention it as static to differentiate it from our problem. The X-axis is the α_{vrptw} parameter value. The Y-axis is the number of mobiles. We see that, as α_{vrptw} nears 1, a smaller number of mobiles is required. This is because the time window is large and the mobiles have greater flexibility as to when to visit the nodes. The three curves correspond to the different $basic\_overflow\_time$ value. This is the $overflow\_time$ of the innermost region and it increases as one moves radially outward.

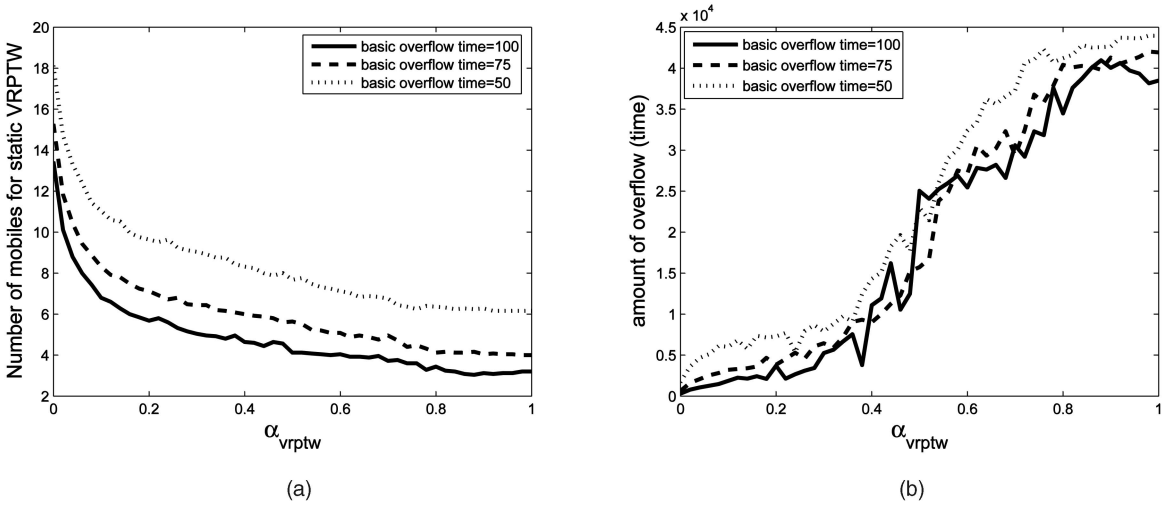


Fig. 4. VRPTW modified for MES. (a) Number of mobiles in solution of static VRPTW. (b) MES as Dynamic VRPTW: Amount of overflow.

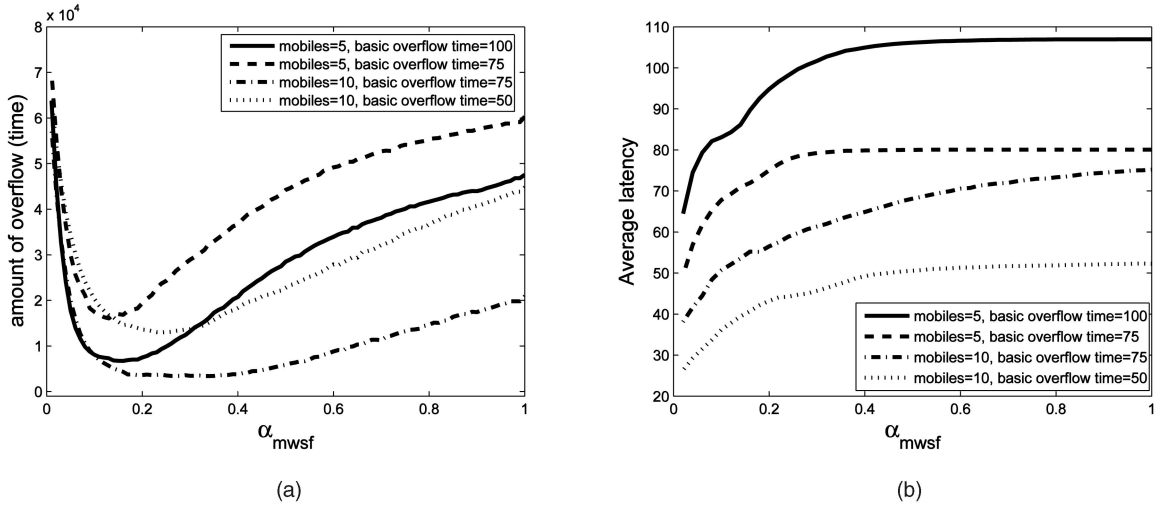


Fig. 5. Minimum Weighted Sum First with the algorithm of Section 7.1.1 (Scheme-1). (a) Amount of overflow. (b) Latency.

Next, we use the number of mobiles obtained above and run the system for 100,000 time units (simulation time). The system is now dynamic, with new requests coming in as soon as the previous request is serviced (with the time window dependent on the α_{vrptw} value, given by (16)). We found that the number of mobiles required in the static VRPTW solution is not sufficient for MES (which can be thought of as Dynamic VRPTW). As a result, some nodes miss their deadline. Fig. 4b presents the result. Again, the X-axis is the α_{vrptw} parameter and the Y-axis is the metric of amount of overflow mentioned earlier. We see that, as α_{vrptw} increases, so does the overflow. This is because a smaller number of mobiles are used (Fig. 4a).

For fair comparison with the other solutions to MES with multiple mobiles (Section 7.1), we fixed the number of mobiles $(num\_mobiles)$. We use these four combinations of ($num\_mobiles$, $basic\_overflow\_time$) in simulations: (5, 100), (5, 75), (10, 75), and (10, 50).

Fig. 5 presents the results for the Minimum Weighted Sum First for varying α_{mwsf} values, with the Scheme-1 of Section 7.1.1 of dividing the area into equal parts and having one mobile in each independently. Fig. 6 presents the results for the Scheme-2 of Section 7.1.2.

Fig. 7 presents the results of using the VRPTW modified for our problem. The difference between this and Fig. 4b is that the number of mobiles is fixed now, whereas, in Fig. 4b, each α_{vrptw} had a different number of mobiles (as obtained by the solution to static problem).

We see that, in all cases, Scheme-2 with Minimum Weighted Sum First (Fig. 6a) performs better than Scheme-1 (Fig. 5a). This is because, with Scheme-2, the mobiles have a global knowledge of the whole system and are not constrained to their assigned areas. Comparing Minimum Weighted Sum First with Scheme-2 (Fig. 6a) and VRPTW modified for MES (Fig. 7a), we see that, for the case with $num\_mobiles = 5$, $basic\_overflow\_time = 75$, Minimum Weighted Sum First performs better, whereas, for the other three cases, VRPTW modified for MES performs better. By better performance, we mean a smaller amount of overflow.

Looking at the latency plots, we see that, for α_{mwsf} values closer to 0, the latency is less (Figs. 5b and 6b). This is

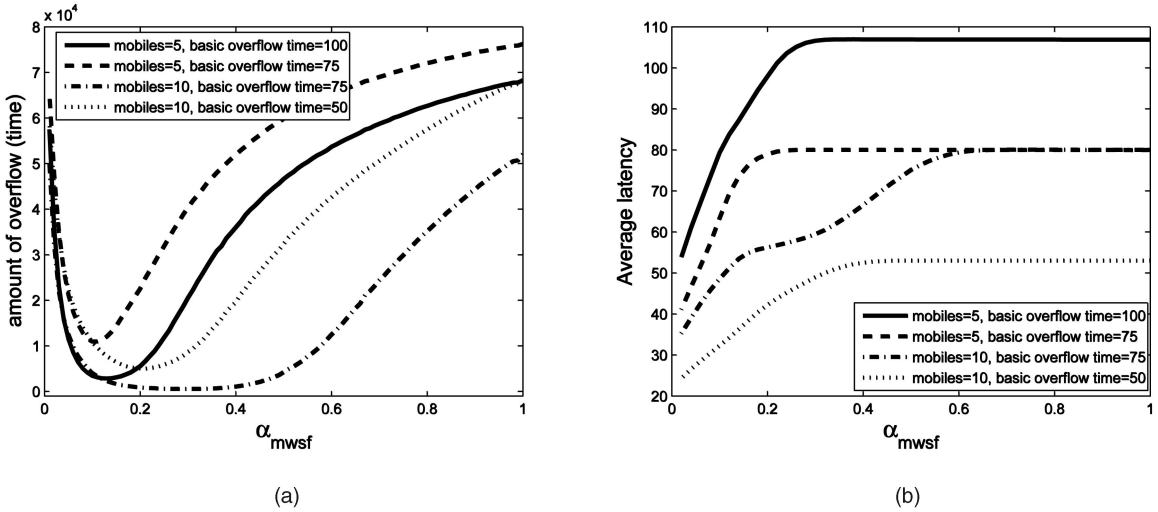


Fig. 6. Minimum Weighted Sum First with the algorithm of Section 7.1.2 (Scheme-2). (a) Amount of overflow. (b) Latency.

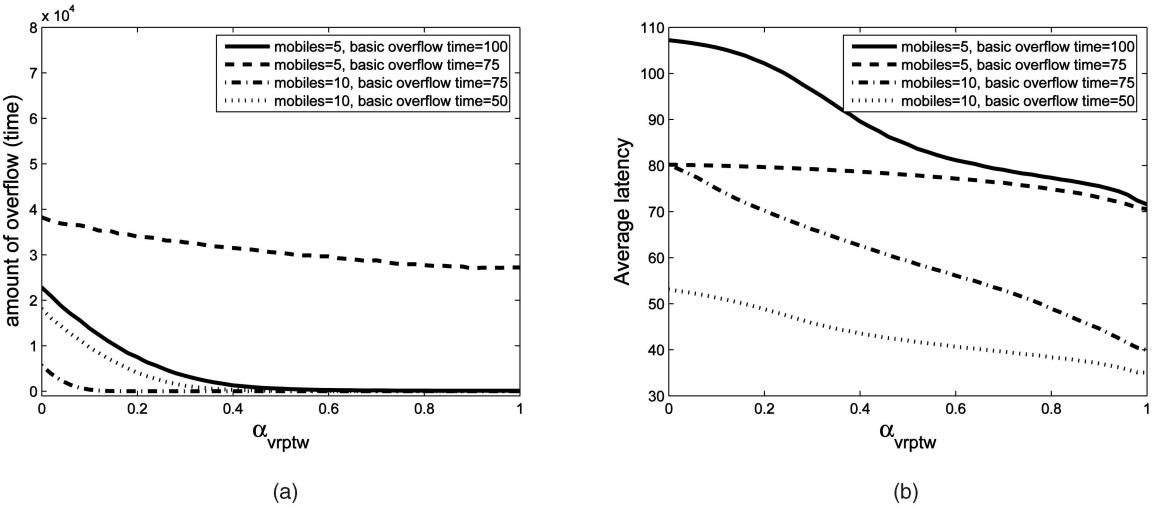


Fig. 7. VRPTW modified for MES, fixed number of mobiles. (a) Amount of overflow. (b) Latency.

because a higher weight is given to *cost* and the mobiles tend to revolve around a cluster of nodes for some time (leading to low latency values for these nodes). The mobiles go to a farther node when it is close to deadline or has missed its deadline. Finally, averaging over all the nodes leads to a smaller average latency. Also, the average latency is monotonically decreasing as the α_{vrptw} parameter increases (Fig. 7b). This is because, as the width of the time window increases, mobiles spend less time waiting and data is collected from nodes much before they reach their full capacity. This observation is in accordance with Table 3.

Note. We have not presented the results of EDF with lookahead. It performed poorly (in terms of percentage failure) for up to lookahead of 7. We noticed an interesting behavior during simulation of EDF with lookahead. For the scheme of Section 7.1.1, where each mobile is responsible for its area, the nodes may form clusters depending on physical proximity. When this happens, the mobile may stay in only one cluster and never leave it to visit the other nodes (which are not in this cluster). This behavior is somewhat similar to the case of Minimum Weighted Sum First with $\alpha_{mwsf} = 0$. This was mentioned in Table 2, where the mobile would be revolving across a locally minimum set of smallest-cost nodes. However, this behavior of EDF with lookahead was not noticed with Scheme-2 (Section 7.1.2) of scheduling together. This happens only with Scheme-1 because the mobile is responsible for few number of nodes, which may lead to this clustering behavior. For instance, if there are 10 mobiles and 100 nodes, each mobile is approximately responsible for 10 nodes., whereas, in the other scheme, each mobile has to choose from 90 nodes as to which is to be visited next.

8.4 Uncertainty

In a dynamic mobile system such as the one considered here, there will be uncertainties in the motion of the mobile(s). One of the main reasons for this is obstacles on the path of the mobile(s). As a result, the actual travel time from a node to another may be more than what is given in the *cost* matrix. To model the effect of this on the scheduling algorithms, we define a parameter $uncertainty \in [0, 1]$. The real cost (in the face of uncertainty) between nodes i and j is calculated as:

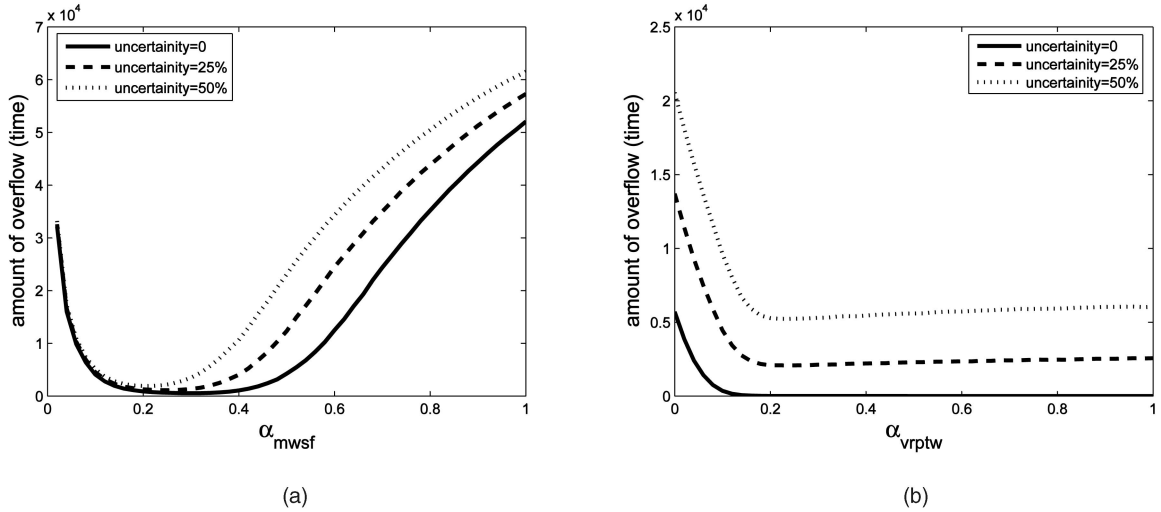


Fig. 8. Effect of uncertainty on schedule. (a) Minimum Weighted Sum First, Scheme-2. (b) MES as dynamic VRPTW.

$$if(rand[0..1] < uncertainty)$$
$$cost[i][j] = (1 + rand[0..1]) \times cost[i][j].$$

Fig. 8 plots the amount of overflow in light of the above-mentioned uncertainty. For this, $mobiles$ and $basic\_overflow\_time$ are fixed to 10 and 75, respectively.We use $uncertainty = 0, 0.25, 0.5$. We see that, as $uncertainty$ increases, so does the amount of overflow for a given value of the parameter (α_{mwsf} or α_{vrptw}).

8.5 Minimum Number of Mobiles for No Overflow

Ideally, it is desirable to have no node missing its deadline. When using the modified VRPTW algorithm, a new mobile can be added to visit a node if there is no feasible insertion spot for it. Thereafter, this mobile becomes part of the system and can be used for servicing other nodes in the future too. This process will not work with Minimum Weighted Sum First or EDF with lookahead due to the inherent difference in the algorithms, as mentioned in Section 7.3. In these two algorithms, a mobile is assigned a node only when the mobile visits its currently scheduled node. However, in MES using VRPTW, a node is scheduled to be revisited by one of the mobiles as soon as it is visited. As a result, due to the lack of the luxury of adding mobiles as and when required, for these two algorithms (Minimum Weighted Sum First and EDF with lookahead), we need to fix the number of mobiles a priori. An efficient way to find the minimum number of mobiles (which leads to no deadline misses) in these cases is to experiment, doing a binary search on the number of mobiles.

We measure the minimum number of mobiles required for the modified VRPTW algorithm to achieve the case of no overflow. This is shown in Fig. 9a. For a given α_{vrptw} value, for each topology, we find the number of mobiles required which leads to no node missing its deadline over the whole simulation time (by adding a new mobile as and when there is no feasible insertion spot for the node just visited). We average this over the 25 random topologies.

To see if these mobiles are sufficient for Minimum Weighted Sum First, we calculate the minimum number of mobiles (across all α_{vrptw} values) for each topology. We run the Minimum Weighted Sum First algorithm with Scheme-2, using this number of mobiles for the corresponding topologies. The results are shown in Fig. 9b. We see that the number of mobiles obtained by the modified VRPTW algorithm is insufficient to meet all deadlines.

The various simulations overall suggest that modified VRPTW may be a better strategy for the MES problem with multiple mobiles. As the problem is not dynamic (not considering the temporal changes in $overflow_time$s mentioned in Section 7.3) in the true sense, i.e., all inputs (node locations and $overflow\_time$ values) are known in advance, the following strategy can be followed: The two better performing heuristics (modified VRPTW and Minimum Weighted Sum First with Scheme-2), being computationally nonintensive, can be run for the various parameter values (α_{vrptw} and α_{mwsf}, respectively) for a sufficiently long time. The heuristic and the corresponding parameter value which give the best result (minimal deadline misses) can be used. The mobile base stations can then follow the sequence of visits as obtained for this value of the parameter.

9 Discussion

The scheduling problem presented here to address the mobility issues in sensor networks (for purposes like data collection, battery charging, calibration) can be applied to many other practical problems in other domains. One such example is mobile gas tankers used to replenish the gas stations. The gas stations run out of gas at a rate dependent on the population of the area. The problem is to schedule the gas tankers so that none of the gas stations go empty.

The approach presented in the previous sections requires the mobile to be in close proximity of (or rather, at) the sensor nodes. This is essential for calibration and battery charging. Wireless communication capability can be leveraged for the purposes of data collection. Nodes can form clusters and the mobiles can be made to visit the centroid of this cluster. The nodes in the cluster can send their data to an elected cluster-head. This approach is in between the fully multihop system (all nodes sending data to a static base station) and the purely one-hop system (mobiles visiting all the nodes).

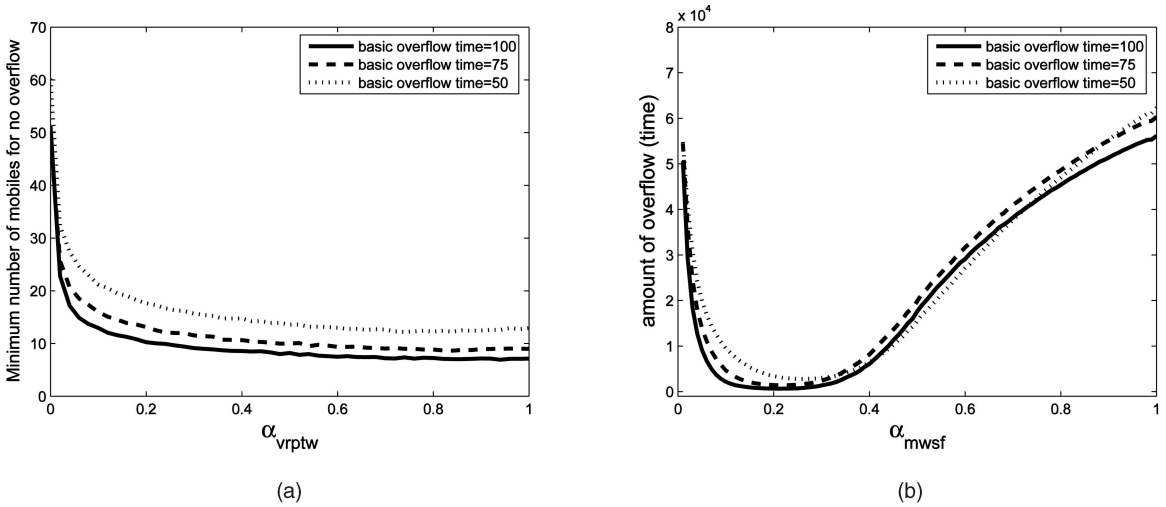


Fig. 9. Minimum number of mobiles for no overflow (VRPTW-MES), overflow in MWSF. (a) MES as dynamic VRPTW: minimum number of mobiles. (b) MWSF, Scheme-2: Amount of overflow.

There is the problem of autonomous navigation when dealing with any system involving mobility. One solution is to make the mobile element be driven by humans. This is the obvious choice for cases such as the gas tanker example mentioned above. Another choice is to make the mobile move on a constrained path. This approach is followed in the NIMS [2] system and in [9], where a mobile element traverses a fixed path, collecting data from nodes on the path. All nodes need not necessarily be on the path of the mobile. Such nodes send their data to nodes on the path which then relays to the mobile when it is in range.

The paper assumes that the *overflow_time*s of the nodes are fixed. The algorithms can handle the case with the *overflow_time*s changing over time. One reason for this change is the dynamics in the phenomenon which the nodes are sensing. The static sensor nodes need to discover this. The mobiles (or the scheduling entity) need to know of the updated values. One possibility is to have a long range radio link between the static sensor nodes and the mobiles (e.g., [11]). Once a static sensor node starts sampling at a different rate (leading to a change in *overflow_time*), it can inform the scheduling entity of this change, which can incorporate it in the future scheduling. Another possibility is for the static sensor nodes to inform the mobile when it is servicing the node. It is intersting to note that, on one end of the spectrum, we have nodes having the same *overflow_time* values. On the other end, we have a truly call and response system [1], which is aperiodic (rather than changing period) wherein the mobile is called for service only when requested by the static sensor node.

10 Conclusions and Future Work

Deployments of sensor networks for sensing the environment are increasingly taking place. Using a controlled mobile element is a promising approach for data gathering in these networks. We considered a network where different nodes may need to sample at different rates. In this context, we introduced a scheduling problem where the mobile element needs to visit the nodes so that none of their buffers overflow. It was shown that the problem is NP-complete and an ILP formulation was given. Heuristics were presented for scheduling in the presence of a single mobile element. We showed ways to use these for the case when multiple mobiles are available and also looked at the modification of the Vehicle Routing Problem with Time Windows for solving our problem. The performances of the heuristics were compared and we saw that the modified Vehicle Routing Problem with Time Windows gave better results in most of the cases.

The current formulation does not include the energy expenditure for movement. However, energy can be easily modeled as a function of distance moved. The heuristics other than the VRPTW-based solution (Minimum Weighted Sum First and EDF with lookahead) can be modified to allow waiting (currently, the mobiles are always in motion). If the deadlines are less strict, this always-moving results in more frequent visits to nodes than required and leads to unnecessary resource consumption and it would be better to spend time waiting. In addition, the speed of the mobile may be made variable, which may result in added benefit.

Acknowledgments

The authors would like to acknowledge the support of the US National Science Foundation and the Center for Embedded Networked Sensing (CENS) in funding this research.

References

[1] M. Batalin, M. Rahimi, Y. Yu, D. Liu, A. Kansal, G. Sukhatme, W. Kaiser, M. Hansen, G. Pottie, M. Srivastava, and D. Estrin, "Call and Response: Experiments in Sampling the Environment," *Proc. Second ACM Conf. Embedded Networked Sensor Systems (SenSys '04)*, Nov. 2004.

[2] R. Pon, A. Batalin, J. Gordon, A. Kansal, D. Liu, M. Rahimi, L. Shirachi, Y. Yu, M. Hansen, W.J. Kaiser, M. Srivastava, G. Sukhatme, and D. Estrin, "Networked Infomechanical Systems: A Mobile Embedded Networked Sensor Platform," *Proc. Fourth Int'l Conf. Information Processing in Sensor Networks (IPSN-SPOTS '05)*, Apr. 2005.

[3] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao, "Habitat Monitoring: Application Driver for Wireless Communications Technology," *Proc. ACM SIGCOMM Workshop Data Comm. in Latin America and the Caribbean*, Apr. 2001.

[4] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless Sensor Networks for Habitat Monitoring," *Proc. ACM Int'l Workshop Wireless Sensor Networks and Applications (WSNA '02),* Sept. 2002.

[5] R.C. Shah, S. Roy, S. Jain, and W. Brunette, "Data MULEs: Modeling a Three-Tier Architecture for Sparse Sensor Networks," *Proc. IEEE Workshop Sensor Network Protocols and Applications (SNPA '03),* May 2003.

[6] T. Small and Z. Haas, "The Shared Wireless Infostation Model—A New Ad Hoc Networking Paradigm (or where There Is a Whale, There Is a Way)," *Proc. ACM MobiHoc,* June 2003.

[7] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein, "Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with Zebranet," *Proc. 10th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS '02),* Oct. 2002.

[8] A. Chakrabarti, A. Sabharwal, and B. Aazhang, "Using Predictable Observer Mobility for Power Efficient Design of Sensor Networks," *Proc. Second Int'l Workshop Information Processing in Sensor Networks (IPSN '03),* 2003.

[9] A. Somasundara, A. Kansal, D. Jea, M. Srivastava, and D. Estrin, "Controllably Mobile Infrastructure for Low Energy Embedded Networks," *IEEE Trans. Mobile Computing,* vol. 5, no. 8, Aug. 2006.

[10] W. Zhao and M. Ammar, "Message Ferrying: Proactive Routing in Highly-Partitioned Wireless Ad Hoc Networks," *Proc. Ninth IEEE Workshop Future Trends of Distributed Computing Systems (FTDCS '03),* May 2003.

[11] W. Zhao, M. Ammar, and E. Zegura, "A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks," *Proc. ACM MobiHoc,* May 2004.

[12] P. Baruah, R. Urgaonkar, and B. Krishnamachari, "Learning-Enforced Time Domain Routing to Mobile Sinks in Wireless Sensor Fields," *Proc. First IEEE Workshop Embedded Networked Sensors (EmNetS-I),* Nov. 2004.

[13] J. Friedman, D.C. Lee, I. Tsigkogiannis, S. Wong, D. Chao, D. Levin, W.J. Kaiser, and M.B. Srivastava, "RAGOBOT: A New Platform for Wireless Mobile Sensor Networks," *Proc. First IEEE Int'l Conf. Distributed Computing in Sensor Systems (DCOSS '05),* 2005.

[14] J. Luo and J.-P. Hubaux, "Joint Mobility and Routing for Lifetime Elongation in Wireless Sensor Networks," *Proc. INFOCOM,* Mar. 2005.

[15] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, *Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* John Wiley & Sons, 1990.

[16] A. Somasundara, A. Ramamoorthy, and M. Srivastava, "Mobile Element Scheduling for Efficient Data Collection in Wireless Sensor Networks with Dynamic Deadlines," *Proc. 25th IEEE Int'l Real-Time Systems Symp. (RTSS '04),* 2004.

[17] M. Rahimi, H. Shah, G.S. Sukhatme, J. Heidemann, and D. Estrin, "Studying the Feasibility of Energy Harvesting in a Mobile Sensor Network," *Proc. IEEE Int'l Conf. Robotics and Automation (ICRA '03),* Sept. 2003.

[18] V. Bychkovskiy, S. Megerian, D. Estrin, and M. Potkonjak, "A Collaborative Approach to In-Place Sensor Calibration," *Proc. Second Int'l Workshop Information Processing in Sensor Networks (IPSN '03),* 2003.

[19] M. Batalin, W. Kaiser, R. Pon, G.S. Sukhatme, G. Pottie, Y. Yu, J. Gordon, M.H. Rahimi, and D. Estrin, "Task Allocation for Event-Aware Spatiotemporal Sampling of Environmental Variables," *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems,* Aug. 2005.

[20] M. Rahimi, R. Baer, O. Iroezi, J. Garcia, J. Warrior, D. Estrin, and M. Srivastava, "Cyclops: In Situ Image Sensing and Interpretation in Wireless Sensor Networks," *Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys '05),* 2005.

[21] *The Vehicle Routing Problem,* P. Toth and D. Vigo, eds. SIAM, 2001

[22] M. Solomon, "Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints," *Operations Research,* vol. 35, no. 2, Mar.-Apr. 1987.

[23] N. Bansal, A. Blum, S. Chawla, and A. Meyerson, "Approximation Algorithms for Deadline-TSP and Vehicle Routing with Time-Windows," *Proc. 36th ACM Symp. Theory of Computing (STOC '04),* June 2004.

[24] H.N. Psaraftis, "Dynamic Vehicle Routing: Status and Prospects," *Annals of Operations Research,* vol. 61, pp. 143-164, 1995.

[25] M. Gendreau, F. Guertin, J.Y. Potvin, and E. Taillard, "Parallel Tabu Search for Real-Time Vehicle Routing and Dispatching," *Transportation Science,* vol. 33, no. 4, pp. 381-390, Nov. 1999.

[26] E. Frazzoli and F. Bullo, "Decentralized Algorithms for Vehicle Routing in a Stochastic Time-Varying Environment," *Proc. IEEE Conf. Decision and Control,* Dec. 2004.

[27] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM,* vol. 20, no. 1, Jan. 1973.

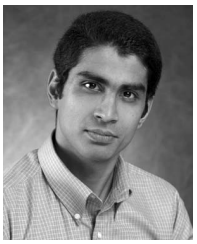
[28] M.R. Garey and D.S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.

[29] M.W.P. Savelsbergh, "Local Search in Routing Problems with Time Windows," *Annals of Operations Research,* vol. 4, pp. 285-305, 1985.

[30] "Packbot, the Next Step in Unmanned Tactical Mobile Robots," http://www.packbot.com, 2007.

[31] "Mica2 Wireless Measurement System," datasheet, http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf, 2007.

Arun Somasundara received the BE degree in computer engineering from the Karnataka Regional Engineering College, Surathkal, India, followed by the ME degree in computer science and engineering from the Indian Institute of Science, Bangalore, India, and the PhD degree from the University of California, Los Angeles. His research was focused on wireless sensor networks, in particular, the effect of mobility on improving the system performance and the scheduling problems arising in such a system. He is currently with Broadcom Corporation in San Jose, California.



Aditya Ramamoorthy received the BTech degree in electrical engineering from the Indian Institute of Technology (IIT), Delhi, in 1999. He was a systems engineer at Biomorphic VLSI Inc. in Thousand Oaks, California, till September 2001. He received the MS and PhD degrees in electrical engineering from the University of California, Los Angeles (UCLA) in 2002 and 2005, respectively. From August 2005 to July 2006, he was a senior design engineer in the Data Storage Signal Processing Group at Marvell Semiconductor Inc., in Santa Clara, California. In August 2006, he joined the faculty of Iowa State University in Ames, Iowa, and is currently an assistant professor in the Department of Electrical and Computer Engineering. His research interests are in the areas of network information theory, channel coding, and their applications to problems in networking and data storage. He is a member of the IEEE.

Mani Srivastava received the PhD degree in electrical engineering and computer science from the University of California, Berkeley in 1992. Currently, he is a professor in the Electrical Engineering Department at the University of California, Los Angeles (UCLA). He is also associated with UCLA's Center for Embedded Networked Sensing (CENS), a US National Science Foundation Science and Technology Center. Prior to joining UCLA, he worked at Bell Labs Research. His current interests are in embedded sensor and actuator networks, wireless and mobile systems, embedded systems, power-aware computing and communications, and pervasive computing. More information about him and his research group is available at his Networked and Embedded Systems Lab's Web site, http://nesl.ee.ucla.edu. He is a senior member of the IEEE and a member of the IEEE Computer Society.