Mobile Element Scheduling for Efficient Data Collection in Wireless Sensor Networks with Dynamic Deadlines

Arun A Somasundara, Aditya Ramamoorthy, Mani B Srivastava
Department of Electrical Engineering, UCLA
Los Angeles, CA 90095-1594
{arun,adityar,mbs}@ee.ucla.edu

Abstract

Wireless networks have historically considered support for mobile elements as an extra overhead. However, recent research has provided means by which network can take advantage of mobile elements. Particularly, in the case of wireless sensor networks, mobile elements are deliberately built into the system to improve the lifetime of the network, and act as mechanical carriers of data. The mobile element, which is controlled, visits the nodes to collect their data before their buffers are full. It may happen that the sensor nodes are sampling at different rates, in which case some nodes need to be visited more frequently than others. We present this problem of scheduling the mobile element in the network, so that there is no data loss due to buffer overflow. We prove that the problem is NP-Complete and give an ILP formulation. We give some practical algorithms, and compare their performances.

1. Introduction

In recent years 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. In most cases the sensors are battery-constrained which makes the problem of energy-efficiency of paramount importance. Some practical deployments include Great Duck Island [9] and James Reserve [1]. Both 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. Naturally, there will be regions in which variation in pollution level will be more, 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. The sensor nodes in regions with higher variation in the phenomenon need to sample more frequently.

There are multiple ways in which the sensor readings are transferred from the sensors to a central location. Usually, the readings taken by the sensor nodes are relayed to a base station for processing using the ad-hoc multi-hop network formed by the sensor nodes. While this is surely a feasible technique for data transfer, it creates a bottleneck in the network. The nodes near the base station relay the data from nodes that are farther away. This leads to a non-uniform depletion of network resources and the nodes near the base station are the first to run out of batteries. If these nodes die, then the network is for all practical purposes disconnected. Periodically replacing the battery of the nodes for the large scale deployments is also infeasible.

A number of researchers have proposed mobility as a solution to this problem of data gathering. Mobile elements traversing the network can collect data from sensor nodes when they come near it. Existing mobility in the environment can be used [10, 11, 5, 3] or mobile elements can be added to the system [6, 14, 15], which have the luxury to be recharged. This naturally avoids multi-hop and removes the relaying overhead of nodes near the base station.

Various types of mobility have been considered for the mobile element. These can be broadly classified as random, predictable or controlled. In the work on Data Mules[10], the mobile element called "data mule" moves randomly. Humans and animals act as data mules, which collect data opportunistically from the sensor nodes when they come near it. The use of a predictable mobile element (mounted on a bus) was considered in [3]. The sensor nodes learn when the bus comes near them, and wake up accordingly to transfer their data. Controlled mobility has been considered in [6], where the robot acting as the mobile base station, moves on a predetermined path, but changes its speed depending on the goodness of the wireless channel and density of nodes. Thus the job of the mobile node is exclusively one of a data-gatherer. Controlled mobility is also used in Message Ferrying [14, 15], where the ferry is used to route

Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS 2004)
1052-8725/04 $20.00 © 2004 **IEEE**

messages between nodes in sparse networks.

In this paper we investigate some **scheduling problems** that naturally come up when trying to 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 has a finite buffer for storing the sensed values. The sensor network is equipped with a mobile element (acting as a base station) that does the job of the data gathering. Once the mobile element visits a sensor node, it transfers the data to its own memory and the sensor's memory is freed. A problem that naturally crops up is the scheduling of the visits of the mobile node so that none of the sensor nodes' buffer overflows. 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) [7]. 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 i.e. 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.

Related work is presented in Section 2. Section 3 provides a formal statement of our problem. We show that the problem is NP-Complete in Section 4. An integer LP formulation is presented in Section 5 followed by Section 6 that provides some heuristics that yield good results. Results are discussed in Section 7 and we conclude in Section 8 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 [14, 15] deals with using a message ferry to route data from one node to another in a sparse network. In particular, [14] gives schemes to find the route ferry has to take. Based on a given traffic matrix (expected traffic from any node to any other node), goal is to find the optimal route of ferry so that average delay from source to destination is minimized, 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

The difference from TSP has been outlined earlier. TSP is a special case of the Vehicle Routing Problem (VRP) [13]. In VRP, as in TSP, a set of nodes need to be visited, but unlike TSP, there can be more than one vehicle. Vehicles start and end at a special node ($node_0$), which is referred to as the depot. The goal is to find the number of vehicles and the Hamiltonian tour assigned to each, such that sum of the distances travelled by each vehicle is minimum.

There are many variants to VRP. Those relevant to our problem are VRP with time Windows (VRPTW), and Periodic VRP (PVRP). In VRPTW [12, 13], 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 [2], 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 apriori. Only the current time window is known, and the next time window is decided based on the current visit time. In PVRP [13], each node has to be visited a pre-specified number of times, but there are no time window constraints on the visits. Our problem can be considered as "Periodic VRP with Dynamic Time Windows".

2.2. Processor Scheduling

Our problem can also be looked at from the view of processor scheduling. Tasks come periodically, have a execution time, and need to be finished before their deadline. We can 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 is that the travel time is the execution time (Worst Case Execution Time (WCET) will be time to reach from its farthest neighbor). A very important distinction in our problem is that the next instance of the task is released as soon as the previous instance is serviced. This automatically rules out schemes based on static priority assignment, such as Rate Monotonic scheduling [8]. Dynamic schemes such as Earliest Deadline First [8] (called deadline-driven scheduling in the reference) are feasible approaches, and are discussed in future sections. Also, there is no notion of preemption in our problem.

3. Problem Formulation

We are given the following:

- A fully connected graph of n nodes: $node[1..n]$
- A matrix $cost[1..n][1..n]$ that denotes the time taken to go from one node to another
- A vector that contains buffer overflow times, $overflow\_time[1..n]$. The i^{th} element of this vector determines the time at which the buffer in the

i^{th} node will overflow. This can be computed using the buffer size and sensing rate.

- A starting node $node_0$.

We make the following assumptions,

- The matrix, $cost[1..n][1..n]$ and the vector $overflow\_time[1..n]$ 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 **Mobile-Element-Schedule** ($cost[1..n][1..n]$, $overflow\_time[1..n]$, $node_0$) problem is the problem of finding a sequence of visits to nodes from $node[1..n]$ 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 $t_k + overflow\_time[k]$.

Note that the problem outlined in Section 1 can be cast in this form. For example if one considers the example of a mobile element gathering pollution level data from sensor nodes, we can set the sensing rates of different nodes based on expected pollution level dynamics. Since we know the specifications of the sensor nodes deployed, we can also compute the overflow times for each sensor node.

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 we are given an instance of Mobile-Element-Schedule($cost[1..n][1..n]$, $overflow\_time[1..n]$, $node_0$) that has a solution S, i.e. a schedule such that none of the buffers at any node 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, $T_O = \max_{k \in [1..n]} overflow\_time[k]$

S is some sequence of numbers and letters $x_1, x_2, ...$ where each $x_i \in \{1, 2, ..., n\} \cup M$ 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 towards another sensor node).

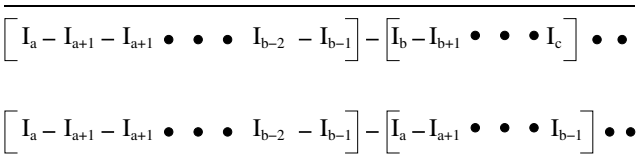


Figure 1. If $I_a = I_b$, and the schedule is valid until the interval I_{b-1}, then we can replace the portion of the schedule after I_b by $I_{a+1}...I_{b-1}$ and repeat it.

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 $[t_1, t_1 + T_O], [t_1 + T_O + 1, t_1 + 2T_O + 1], ...$ The maximum number 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 labelled with a number from $\{1, 2, ..., n\}$ 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. i.e. we can let the new sequence to be I_a, I_{a+1}, ..., I_{b-2}, I_{b-1}, I_b, I_{a+1}, I_{a+2} .... 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 $I_{a+1}, I_{a+2}, ..., I_{b-1}$ and thus by repeating them after I_b we can be sure that none of the buffers overflow. Thus the schedule is periodic.

We now show that the Mobile-Element-Schedule problem is NP-Complete. We state this as a theorem.

Theorem 1 *The Mobile-Element-Schedule($cost[1..n][1..n]$, $overflow\_time[1..n]$, $node_0$) problem is NP-Complete.*

Proof :- The proof is in two parts,
(a) 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.

(b) 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 [4].

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

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

$$\text{overflow\_time}[i] = n \quad (2)$$

$$\text{node}_0 = 1 \quad (3)$$

This reduction is clearly in P.

If G contains a Hamiltonian Cycle, then we also have a solution to Mobile-Element-Schedule since all edges that participate in the Hamiltonian Cycle have weight 1 which means that the total cycle time $= n$ and consequently none of the buffers would overflow.

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 [t+1, ..., t+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, $(t+1), ..., (t+n)$ and the minimum cost between any two nodes is 1.

b) If we let x_i denote the state of the mobile element when $i \in \{(t+1), (t+2), ..., (t+n)\}$, 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 $(t+1), (t+2), ..., (t+n-1)$, otherwise it's buffer would overflow, but this is a contradiction since we know that each node appears exactly once in $(t+1), ..., (t+n)$ from part (b).

But then, the sequence $x_t, x_{t+1}, ..., x_{t+n}$ 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, we have shown that our problem is NP-Complete.

5. ILP formulation

We proved in the last section that the schedule is periodic. Let the period be T. In this section we pose our scheduling problem as an Integer-Linear-Programming (ILP) problem.

Variables:

x_{ij}: $i \in \{1...T\}$, $j \in \{1...n\}$

$x_{ij} = 1$ if at time i, mobile element is at node j, 0 otherwise.

y_i: $i \in \{1..T\}$

$y_i = 1$ if at time i, 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 variable 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,

$$\sum_{j=1}^{n} x_{ij} + y_i = 1, \forall i \quad (4)$$

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

$$\left.\begin{array}{c} \sum_{i=0}^{overflow\_time[j]} x_{ij} \geq 1 \\ \vdots \\ \sum_{i=T-overflow\_time[j]}^{T} x_{ij} \geq 1 \end{array}\right\} \forall j \quad (5)$$

- 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,

$$\sum_{t=i}^{k} y_t \geq [x_{ij} + x_{kl} - C] \times \frac{cost[j][l]}{2 - C} \quad (6)$$

$\forall i, k \in \{1...T\}$, $i < k$ and $j, l \in \{1...n\}$, $j \neq l$ 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, 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 atleast 1 time unit when the mobile element is moving. We can handle the pathological case of cost between two nodes $= 1$ by appropriate discretization of time.)

– Otherwise, i.e. if either or both of x_{ij}, y_{kl} are 0, then there is no constraint that needs to be enforced. Note that in this case the RHS of the constraint is negative and since $y_i \in \{0, 1\}$, it is trivially satisfied.

Given the **Variables** and **Constraints**, we can see that the scheduling problem reduces to finding the existence of a feasible set.

In general we will not know the period T beforehand, and some amount of experimentation will be required. If the period T is large, the total number of constraints for the ILP would be large. In practice it might be hard to solve the problem in this way. Nevertheless, the formulation is presented to provide some insight into the problem.

6. Practical Solutions

The problem has been proved to be NP-complete. In this section, we present some heuristic algorithms. For the kind of dynamic scheduling we need to do, Earliest Deadline First (EDF) would seem to be the first choice to consider, where the node with closest deadline is visited first. This algorithm can be summarized as below:

ALGORITHM: Earliest Deadline First (EDF)

- Input:
  $cost[1..n][1..n]$, $overflow\_time[1..n]$, $start\_node$
- 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 Figure 2 with the values on edges indicating the $cost$ (which are symmetric) and those near nodes indicating their $overflow_time$s. Suppose $start\_node$ is A. First part of Table 1 shows the sequence of visits, ending with A missing its deadline. If we did not stay at D, even though it had the earliest deadline, we would get the 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 $A, D, B, D, A, D, C, D, ...$, and none of the nodes miss their deadlines. On the contrary, it is not possible that not staying at a node caused to miss its deadline, whereas staying would have prevented it. This is because, it has to leave sometime, if not now, to service other nodes; in which case deadline will miss then.

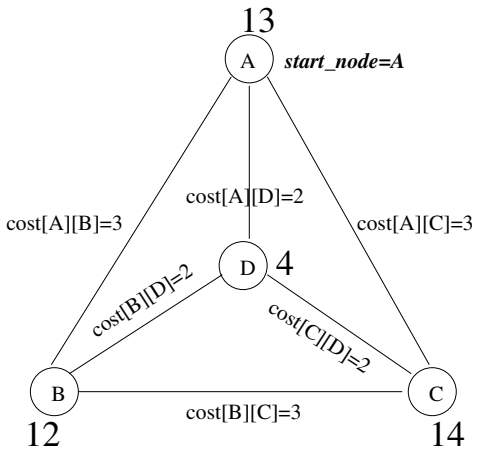


Figure 2. An example to explain choosing next node different from $current\_node$

$current\_time$	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

Table 1. Analysis of the example in Figure 2

One obvious shortcoming in this algorithm is that it does not take into account the $cost$ values, and relies only on deadlines. For instance, consider Figure 3, which shows part of a network. Suppose the mobile element has just visited

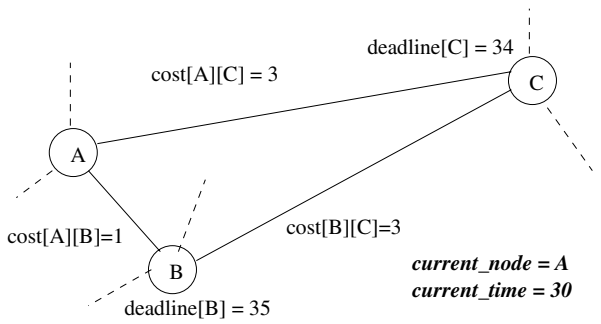


Figure 3. An example to illustrate that EDF type of scheduling is not the best

node A, and 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, deadline of B is missed. On the contrary, had it visited node B first, and then node C, both the deadlines would have been met. This example suggests that one way to account for the $costs$ 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 deadlines of both the nodes are met. Generalizing this, in k-lookahead, we can consider the $k!$ permutations of the k least deadline nodes. Suppose we are at node x_0, and the next k least deadline nodes are $x_1, x_2, ..., x_k$. 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

ALGORITHM: EDF with k-lookahead

- Input:
  k, $cost[1..n][1..n]$, $overflow\_time[1..n]$, $start\_node$
- Initialize $current\_time$, $current\_node$, $deadline[1..n]$ as before.
- Main: Repeat the following
  1. Sort $deadline[1..n]$ 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) Arrival time at the node at $(k+1)^{th}$ entry is minimum, 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

A point to be noted is 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 $x_a, x_i, x_b, x_i, x_c, x_i$. The node x_i is revisited after visiting only one other node. Now, if we schedule k nodes at a time, we would not be able to achieve this result, and deadline of x_i will be surely missed.

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

6.2. Weighted sum heuristic

The lookahead algorithm, in addition to deadline, indirectly 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 values 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 $\alpha \in [0, 1]$, $cost[1..n][1..n]$, $overflow\_time[1..n]$, $start\_node$
- Initialize $current\_time$, $current\_node$, $deadline[1..n]$ as before.
- Main: Repeat the following
  1. $\forall i$, calculate, $weighted\_sum[i] = \alpha * (deadline[i] - current\_time) + (1 - \alpha) * cost[current\_node][i]$
  2. Choose the node $i \neq current\_node$ whose $weighted\_sum[i]$ is minimum
     - if $deadline[i] < current\_time + cost[current\_node][i]$
       * declare failure and stop

$\alpha \in [0,1]$	Result
1	Weight to deadlines only. Same as EDF.
0	Weight to cost only. Results in back and forth motion between closest-distance nod es. All other nodes miss their deadlines.
small	Higher priority to closer nodes
big	Higher priority to closer deadlines

Table 2. Effect of α values on Minimum Weighted Sum First heuristic

- Else
  - $current\_time+ = cost[current\_node][i]$
  - $current\_node = i$
  - $deadline[i] = current\_time+overflow\_time[i]$

END

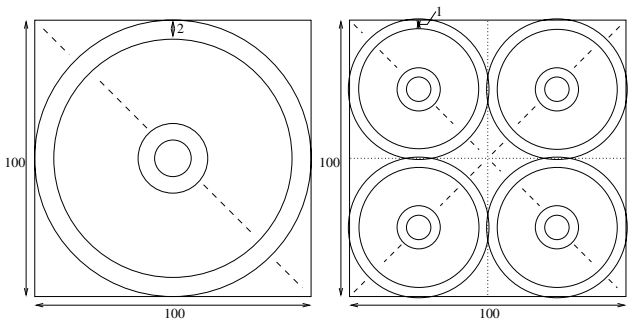
Table 2 shows the effect of different α values. To illustrate the contents of the table, suppose we are at node x_i, and two nodes x_a and x_b have costs 25 and 50 respectively, and relative deadlines 200 and 175 respectively. Clearly x_a is the closer node and x_b has an earlier deadline. When $\alpha < 0.5$, x_a will be chosen, and when $\alpha > 0.5$, x_b will be chosen.

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

At each step, finding the next node takes $O(n \log n + k.k!)$ in the EDF with k-lookahead algorithm. First term is for sorting, and second for finding the best permutation. There are $k!$ permutations, and testing each takes k time. Each step of MWSF takes $O(n)$ for calculating the weighted sum for all nodes, and choosing the minimum. Optimally, ability to find a schedule if one exists, requires n-lookahead. This has a $n.n!$ complexity and is infeasible.

It would be easier if there were some necessary and sufficient conditions to check the existence of a schedule. From Lemma 1 in Section 4, a necessary condition for a schedule to exist is the existence of a solution for TSP(T_O), where T_O was defined to be $\max_i(overflow\_time[i])$). TSP(C) is the decision version of TSP, and has a solution if there is a hamiltonian cycle of length atmost 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, which means a even tighter constraint. Similarly, a sufficient condition for a schedule to exist is the existence of a solution for TSP($\min_i(overflow\_time[i])$). It may be argued that for checking necessary and sufficient condition for existence of 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 which can be leveraged for this purpose.



(a) Topology of type A (b) Topology of type B

Figure 4. The two types of topologies considered in simulation

7. Experimental Methodology and Results

The previous section presented few heuristics. We try to evaluate them in this section through simulation. As the parameter space is huge, we fixed some of them:

- Grid Size: We fix the grid size to 100x100. The speed is taken to be 1, so the *cost* values are same as distance between nodes. They were rounded to the nearest integer.
- Simulation time: We simulate for 100000 time units.
- Location and number of nodes n: The nodes are randomly placed on the grid. We use 50, 100 nodes.
- $overflow\_time$ values. One option was to assign these values randomly to the nodes. But to simulate real world situation, we assumed that the point of interest is located in the center of grid, and the nodes closer to the center have smaller $overflow\_time$ values, as they are sampling more frequently. We placed concentric circles, smallest one being of radius 2. Also, the radius increased by 2 from one circle to next. This is shown in Figure 4(a). The innermost region had smallest $overflow\_time$, called $basic\_overflow\_time$, and the regions radially outwards were a constant factor of it, i.e. $\{1, 1.2, 1.3, ...\} * basic\_overflow\_time$. Similarly we considered the grid with four points of interest as shown in Figure 4(b). Here the smallest circle had radius 1, and it was increasing by 1 unit from one circle to next. The $overflow\_time$ was assigned based on its distance from the center of the quadrant in which the point was present.

Label	Figure #	n	$basic\_overflow\_time$
A1	4(a)	50	500
A2	4(a)	50	600
A3	4(a)	50	700
A4	4(a)	100	1000
A5	4(a)	100	1100
A6	4(a)	100	1200
B1	4(b)	50	500
B2	4(b)	50	600
B3	4(b)	50	700
B4	4(b)	100	1000
B5	4(b)	100	1100
B6	4(b)	100	1200

Table 3. Set of Experiments

Putting everything together, we ran the experiments shown in Table 3. We will refer to the labels shown in the table. For each of these, results presented are average of 100 runs. The parameter which changed from run to run was the location of the nodes, and correspondingly the $cost$ and $overflow\_time$ values.

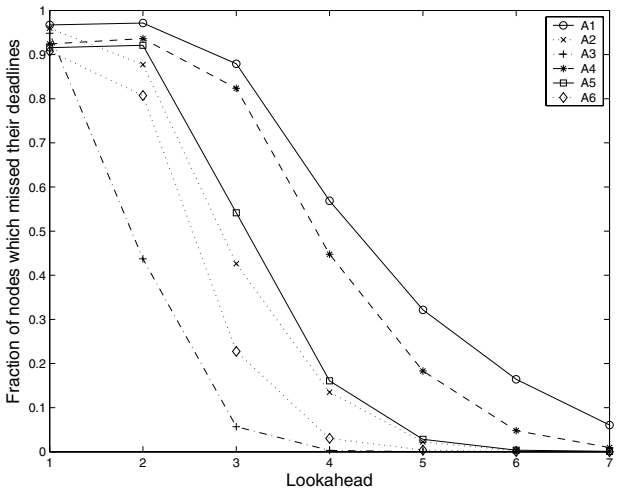
For the purposes of evaluation, instead of stopping the algorithm when a node missed its deadline, we continue, noting this fact, and updating its deadline. Thus, the metric used is the fraction of nodes which missed their deadlines, out of the number of visits made to them in the simulation time of 100000.

7.1. EDF with lookahead

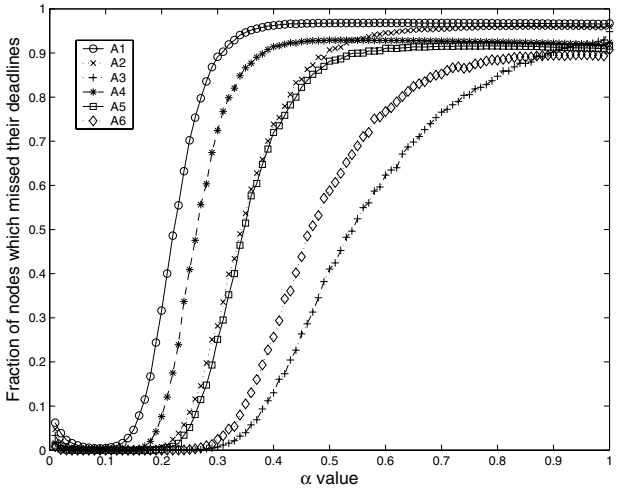
Figure 5(a) and 6(a) show the result of running the k-lookahead EDF algorithm on the two sets of topologies. k ranged from 1 to 7. If no permutation of k nodes were available such that none of the k nodes missed their deadlines, we would choose the permutation which had the minimum overflow. The fraction of nodes missing their deadlines decreased with increasing lookahead. Also performance on A3 was better than A2 which was better than A1. This is obvious because these have $basic\_overflow\_time$ of 700, 600, 500 respectively, and higher this quantity, lesser nodes miss their deadlines. Similar trends are observed in other sets.

7.2. Minimum Weighted Sum First

Figure 5(b) and 6(b) show the result of running the MWSF algorithm on the two topologies. α ranged from 0.01 to 1 with steps of 0.01. $\alpha = 1$ is same as EDF. We did not use $\alpha = 0$, because, with no weight to deadlines, the algorithm is not guaranteed to visit all nodes, as it would be stuck going to and fro between nodes on a locally minimum weighted edge. We see that lower α values perform



(a) EDF with lookahead on set A: A1-A6



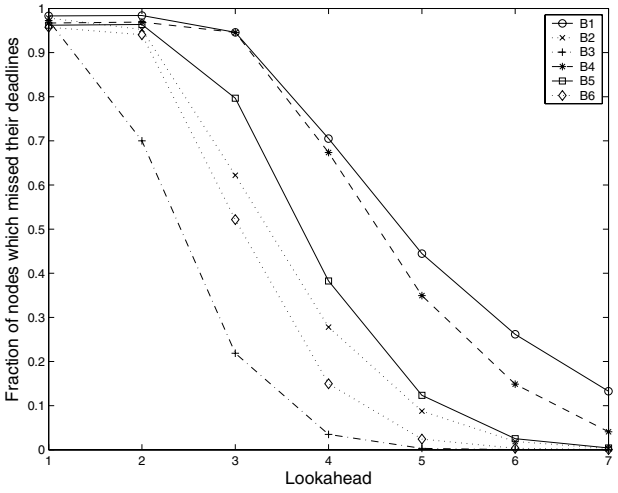
(b) MWSF on set A: A1-A6

Figure 5. Results of EDF with lookahead and Minimum Weighted Sum First algorithms on first set of topologies A1-A6

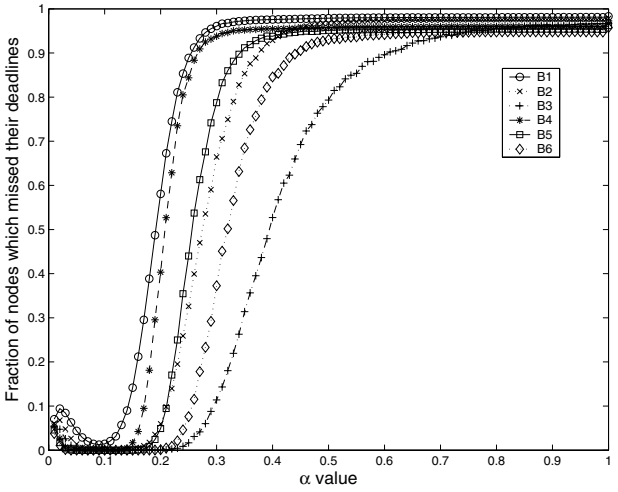
better. To get a better insight into working of this algorithm, we examine part of 5(b). Figure 7 shows the result for $0.01 \leq \alpha \leq 0.24$ for topology types A1, A2, A3. We see that as the topology becomes less constrained, the range of α values giving the optimum solution increases.

7.3. Comparison and Discussion

From the graphs, we can see, especially for the most constrained topologies A1, A4, B1, B4, that MWSF performs better than EDF with 7-lookahead. These were the most constrained as the $basic\_overflow\_time$ was least (500 for



(a) EDF with lookahead on set B: B1-B6



(b) MWSF on set B: B1-B6

Figure 6. Results of EDF with lookahead and Minimum Weighted Sum First on the second set of topologies B1-B6

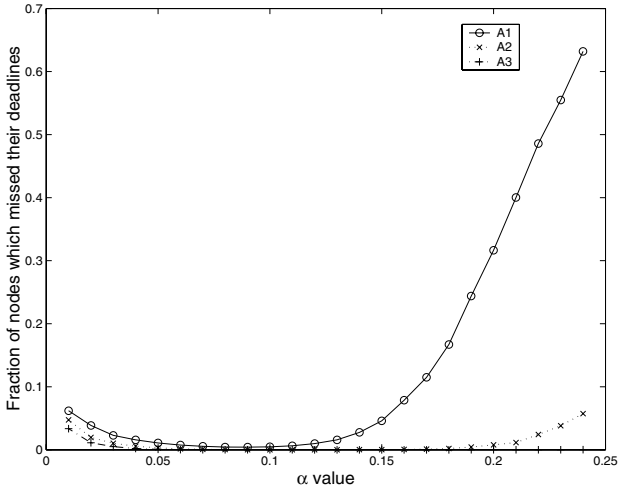


Figure 7. Results of Minimum Weighted Sum First on smaller range of α values for A1-A3. This is part of Figure 5(b)

A1, B1 and 1000 for A4, B4). It may be noted that for a particular topology type, say A1, the set of 100 topologies over which results are averaged, were same for the two algorithms.

Obviously, EDF will continue to perform better with increasing lookahead, and at some point will be better than MWSF. But the goal is to get a better solution with lesser computation complexity.

One thing to be mentioned in case of MWSF is that each topology would have a α value, at which it would perform best. As the results presented are average of 100 runs, we investigated the α value which was performing best for each topology. Figure 8 shows a histogram which shows the number of topologies (out of 100) which had best performance at each α value. The results are shown for A1. α around 0.09 seems to be performing well for most cases (for this topology type), as was also evident previously from Figure 7. Now, with the best α value for a given topology, we tried applying lookahead, but did not notice any performance gains.

To summarize, given a topology, we can run the EDF with k-lookahead algorithm, choosing k depending on the computational budget. Also we can run MWSF with different α values, and finally choose the best.

In addition to the algorithms and results presented here, we also tried the Minimum Slack First (MSF) algorithm. Slack is defined as the time remaining if a node is processed now, i.e. $deadline[i] - current\_time - cost[current\_node][i]$ for a node i. In MSF, the node with minimum slack is visited first. We implemented MSF, and its lookahead version. It performed better with increasing values of lookahead, but performance was poorer than the corresponding EDF with lookahead.

All the algorithms assumed that the mobile element travels at a constant speed, say s. As the energy varies with $voltage^2$ for processors, energy for the mobile element varies as s^2. So lower the s, more energy efficient the system is. One simple optimization that can be done is global slowdown, where the mobile element travels at speed $x * s, 0 < x < 1$, and still the deadlines are met. Decreasing s increases the $cost$ matrix, the internode travel time.

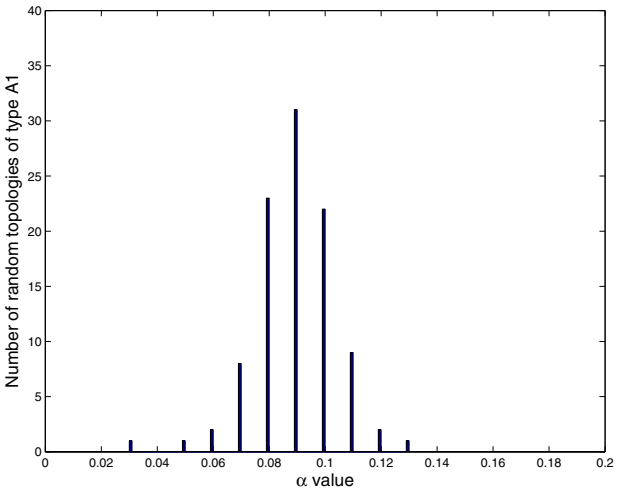


Figure 8. Histogram of number of topologies (out of 100) which had best performance at each α value for A1

Hence we can try with lower values of s, and use the minimum, which still meets the constraints.

8. Conclusions and Future Work

Deployments of sensor networks are taking place to sense the environment. Using a controlled mobile element is a promising approach to collect data from these sensor nodes. The sensor nodes may be sampling at different rates. In this context, we have introduced a scheduling problem, where the mobile element needs to visit the nodes so that none of their buffers overflow. We showed that it is NP-complete and gave an ILP formulation for it. We gave some heuristics, and showed that Minimum Weighted Sum First algorithm performs well and is computationally inexpensive. There are many directions in which this work may be pursued further. We can try to find an approximation algorithm for this problem. We can formulate the problem to adapt to node addition or failures. Another direction to be explored is the case with multiple mobile elements.

References

[1] www.jamesreserve.edu (james san jacinto mountains reserve).

[2] N. Bansal, A. Blum, S. Chawla, and A. Meyerson. Approximation algorithms for deadline-tsp and vehicle routing with time-windows. In *36th ACM Symposium on Theory of Computing (STOC)*, 2004.

[3] A. Chakrabarti, A. Sabharwal, and B. Aazhang. Using predictable observer mobility for power efficient design of sensor networks. In *The second International Workshop on Information Processing in Sensor Networks (IPSN)*, 2003.

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

[5] 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. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2002.

[6] A. Kansal, A. Somasundara, D. Jea, M. Srivastava, and D. Estrin. Intelligent fluid infrastructure for embedded networks. In *The Second International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2004.

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

[8] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1), Jan 1973.

[9] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2002.

[10] R. C. Shah, S. Roy, S. Jain, and W. Brunette. Data mules: Modeling a three-tier architecture for sparse sensor networks. In *IEEE Workshop on Sensor Network Protocols and Applications (SNPA)*, 2003.

[11] 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). In *The Fourth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2003.

[12] M. Solomon. Algorithms for the vehicle routing and scheduling problem with time window constarints. *Operations Research*, 35(2), Mar-Apr 1985.

[13] P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*. Society for Industrial & Applied Mathematics (SIAM), 2001.

[14] W. Zhao and M. Ammar. Message ferrying: Proactive routing in highly-partitioned wireless ad hoc networks. In *The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS)*, 2003.

[15] W. Zhao, M. Ammar, and E. Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *The fifth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2004.