Indian Institute of Technology, Kanpur

Latency Bound Data Collection from Sparse Sensor Networks Using Data MULEs

Supervisors:
Dr. R. K. Ghosh (IIT Kanpur)
Dr. Maitreya Natu (TRDDC, TCS, Pune)

By:
Aman Kumar Singh

July 17, 2014

Contents

Chapter 1

Introduction

Wireless Sensor Networks consist of a large number of sensor nodes, which are battery-powered, low energy consuming devices. Wireless Sensor Networks (WSNs) are used for applications such as monitoring (e.g., pollution prevention [1], structures and buildings health), event detection (e.g. fire hazard [15]) and target tracking (e.g., surveillance [4]). These devices perform three basic tasks:

- Sample a physical quantity from the surrounding environment
- Process and buffer the acquired data
- Transfer the data through wireless communications to a data collection point called sink node or base station.

The sensors, as soon as deployed on the field, identify their adjacent sensors (neighbours), and are able to form an ad-hoc network. If this network is connected (each sensor is able to send data to another sensor on the field through one or multiple hops), then it is called a dense sensor network. Such a network is used as basic infrastructure for routing sensor-acquired data to the sink node (Base Station) using WSN routing protocols [3]. Occasionally, sink node may send data to all sensor nodes as well.

However, dense networks are seldom realizable in practice. Usually, sensors are spread (air dropped) over a large geographical area resulting in a sparse network. This kind of networks are useful for applications (e.g., environmental monitoring applications [22]) where fine-grained sensing is not required. In sparse sensor networks, islands of sensor sub-networks are formed (WSN islands [25]), which cannot communicate to each other without external help. Data collection from this kind of network, without deploying additional sensors can be done using:

- Long range relay nodes [33]

  Long range and high power nodes, whose main purpose is to establish connectivity among such WSN islands, and their connectivity to the base station by relaying/receiving data to/from sensors out of range of the local island. Their placement in a sensor network is a well studied problem.

- Mobile data collectors (MDCs) [8] or Mobile Ubiquitous LAN Extensions (MULEs) [29]:

  Controlled mobile robots acting as mobile sinks/Base stations/relay nodes. These can be programmed to tour any set of locations so as to collect data from the sensors at those positions,] and relay the same to base station directly or through another MULE. Since the sensors have to wait for the MULE to arrive in range

  1. Sensors may be required to have a battery powered passive wake-up radio device [5], which can be triggered by a nearby MULE. MULE will then be able to collect data from the sensor, and forward the collected data when it is in proximity of the base station (or another MULE assigned to relay to the base station [28]).
  2. Sensors wake-up and sleep operation can be controlled at the MAC layer [24] [18]. In this approach, sensors periodically sleep and wakeup to save power. Data transmission occurs only when the MULE is in range, and the sensor is awake.

The MULE approach naturally saves energy of the sensors leading to prolonging network lifetime. The use of MULEs limits sensor communication to one hop only, and relieves sensors from perform aggregation or forwarding of data of other sensors. However, the main disadvantage in this approach is increase in latency of data transfer from sensors to Base Station. This may have negative impact onperformance of some applications [11]. This is the reason that the studies on the problems of MULE path optimization and job scheduling are extensive in literature [12].

1.1 Main Contribution of the Work

In this thesis our investigation is centered around study of techniques and heuristics used for the use of MULEs in data collection from sensors. The main contribution of our work is to compare existing heuristics and propose new heuristics for data collection through MULEs. The usual approach in this direction is to make use of multiple MULEs by partitioning the network into roughly "equal" parts (in terms of TSP tour time of that partition, together with the time taken to collect data from sensors in that partition), and each partition is assigned to a MULE. Our work consisters the latency constraint of the sensor network application as input and computes the number of MULEs required along with their trajectories. We assume MULEs can move freely in the plane of the sensor network, and they have identical, constant speed.

1.2 Summary of the Work

This thesis has been organized into five chapters including the introductory chapter. In Chapter Preliminaries and Subproblems, we first deal with the components of the general problem of data MULE scheduling, and with the aspects that we are concerned, namely, *path selection*. Next, we introduce location nodes, location graphs and propose heuristics to compute them. We also provided justification of our choice of heuristic with performance measures. In Chapter Path selection, we describe our main heuristic for path selection and also provide pseudocode for the same. In Chapter **??**, we present the results of simulation studies conducted to examine the performance the proposed heuristics for data collection using multiple MULEs. Finally, Chapter 6 contains two anticipated directions in which this work may be continued. **Comments: Not happy with the sentence, what does it actually mean?**. The first is the addition of capability for avoidance of obstacles, and the second is the application of optimization of TSP tours in presence of neighbourhoods into the proposed heuristics.

Chapter 2

Preliminaries and Subproblems

2.1 Path Selection of Mobile elements

It is known that controlled MULEs can increase a WSN's lifetime by saving energy. But the problem of planning an optimal path and job schedule is a hard problem in general, known as the Data MULE scheduling problem [31]. It has three components:

1. Path selection: which trajectory the data mule follows
2. Speed control: how the data mule changes the speed while moving along the path
3. Job scheduling: from which sensor the data mule collects data at each time point

In this thesis we will focus only on Path selection of the MULEs.

2.2 Geometric Disc Covering and Location Nodes

In the interest of collecting data from sensors in one hop, we propose to cover the sensor field with circular discs of radius equal to the range of the sensors. The centers of these discs are going to be locations where MULEs will pause to collect data from the sensors. We call these positions *location nodes* for convinience. Our path selection heuristic uses location nodes as vertices of a graph embedded on the 2D plane. Ofcourse, we are approximating the communicable region of a sensor to a circular disc, and also assuming that the range of the MULE is greater than that of any sensor (we assume all the sensors are identical in communication range). The centres of these discs are called *location nodes*.

There are two kinds of geometric disc cover problems:

Geometric minimum disk cover (GMDC) In this problem, a field of nodes is given, and the aim is to cover all the nodes with minimum number of discs (of given radius) possible. The centers of the discs can lie anywhere on the field.

Geometric discreet unit disc cover (GDUDC) In this problem, in addition to a field of nodes, a set of points C is also given. The aim is to determine the smallest subset C' of C, such that the discs placed at the points in C' cover all the nodes in the field.

We use the problem GMDC here. Compared to GMDC, GDUDC is a much better studied problem. Therefore, we initially planned to get an intermediate solution of GMDC through some heuristic/PTAS, then apply GDUDC problem for a better solution. We tried and failed to implement one solution [6] for GDUDC. Therefore, we depend completely upon the following heuristics for disc covering.

There are many heuristics and approximation schemes [16] [14] for GMDC already known, any one of which can be used here. Following are four heuristics (JGRD,GRD,SEL,HEX) and one approximation scheme (SHFT) that we tested on sample fields.

2.2.1 Preliminaries

Let P be the set of points to be covered with discs of radius R, in the square field F. For any two $p_1, p_2 \in P$, there are two $c_1, c_2 \in F$ (called "crosses" for convinience) such that p_1 and p_2 will lie at the boundary of a disc of radius R placed at either c_1 or c_2. We consider points p_1 and p_2 as covered (barely) by any of the two discs above. For any two points p_1 and p_2 such that distance between p_1 and p_2 is less than $2R$, there exist two crosses c_1 and c_2 (c_1 and c_2 are said to "involve" p_1 and p_2 for convinience). Let C be the set of all possible crosses in the field F.

A set S of disc centres such that they cover all the points in the field F is called a solution of the MGDC problem in the field F. Let S_1 be such a solution. Then there exists a solution $S_2 \subset C$ such that $|S_1| = |S_2|$.

2.2.2 Heuristics

OPT This is the optimum solution algorithm, which uses brute-force to search for the minimum set cover to find the optimum solution. First we make a table of the information regarding which cross (refer 2.2.1) covers which of the sensors. These crosses act like the subsets of the set of all sensors, containing the sensors they cover. We then find the minimum set cover using brute-force. Needless to say, the running time of this algorithm blows up very fast.

HEX In this heuristic [23], first we tile the field with hexagons of sides equal to the radius of the discs to be used (as shown in figure 2.1). A sensor is said to be covered by a hexagon, if it is closer to its center than the center of any other hexagon.

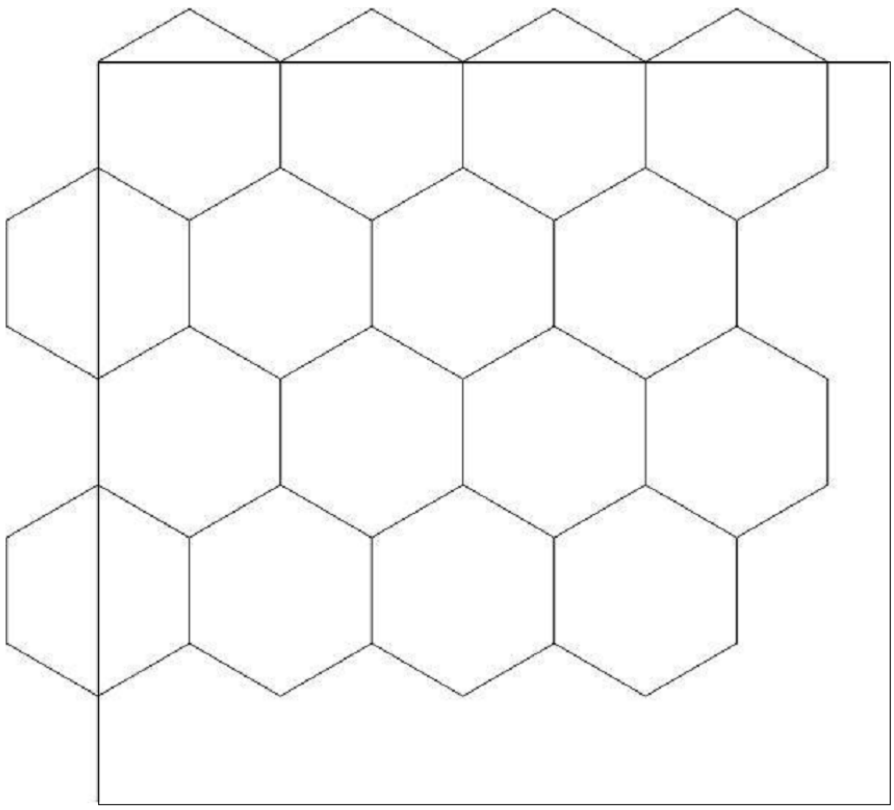


Figure 2.1: Hexagonal tiling

JGRD This is originally a set cover heuristic [10]. Consider the set P as the target set to be achieved from the union of smaller subsets, and the discs centered at the crosses in C represent the subsets of points covered by them. Our problem of MGDC then becomes a set cover problem. [10] gives a greedy heuristic for this.

GRD (We are not aware of any other work using this heuristic) Let the list of all sensor positions in the field F be L. First sort L according to their (x,y) position co-ordinates (first by x co-ordinate and then by y co-ordinate). Starting from the first sensor p in the list, pick the disc which covers maximum number of uncovered sensors and covers p too. To do this, compute all the crosses in C which involve p. Place a disc of radius R at each of the crosses, and choose the cross whose disc covers the most number of points. Choose the next uncovered sensor in L and repeat the covering procedure, until all sensors are covered.

SHFT The shifting strategy from [16]. This algorithm takes two input parameters: l (shifting parameter, chosen later) and D (diameter of discs to be used for covering). Let the field be a square of length W. First we divide the field into vertical strips of width $l \times D$, as shown in figure 2.2. We do disc cover on these vertical strips (explained later) individually (treating them as separate fields. The union of the solutions of all the vertical strips is clearly a solution to the problem. This solution is assigned to $current\_solution$. Now, the whole partition of strips is shifted horizontally, away from the origin, by the amount of D (as shown in figure 2.3). Again, all the strips are covered with discs (to be explained later how), and the union of the solutions of these shifted vertical strips another solution, assigned to $new\_solution$. If the size of $new\_solution$(the number of discs in the solution) is less than the size of $current\_solution$, the current solution now becomes $new\_solution$. We continue shifting the partition and measuring the solutions thus generated $l-1$ times.

To cover a strip of width w, we again apply the shifting strategy. We divide the strip into squares of side w (and maybe a remainder rectangle, in case W leaves a remainder upon dividing with w). To cover this sqaure (of size $w \times w$) with discs of diameter D, we use the strategy OPT described above. The shifting of this partition is done by the amount D just like in the original field, and the smallest solution is chosen. The shifting parameter l depends on the time constraint one has regarding the running time of this algorithm. It is chosen in such a way that the problem size in each sqaure (of size $w \times w$) is small enough for the OPT to run in required time constraint.

Figure 2.2: Original division of the field into vertical strips. l is 3 here.

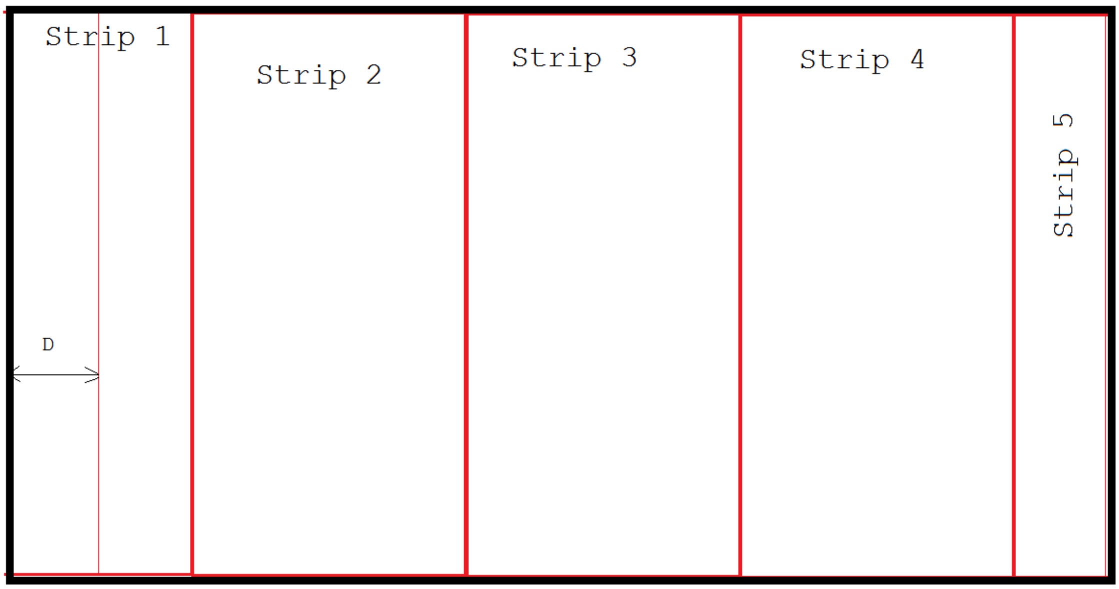


Figure 2.3: After the shift right by the amount D

SEL For any sensor s, let C_s be the set of crosses (refer 2.2.1) covering it. Pick the cross which covers the least number of sensors. Keep iterating on the still uncovered sensors, until all the sensors are covered.

RND Randomly pick crosses until all the sensors are covered. Just for sake of comparision with other heuristics.

2.2.3 Results

The above heuristics and the approximation scheme were tested on a 1000m×1000m field with discs of radius 50m. The number of sensors are increased form 50 to 100 in steps of 10. As we can see, GRD performed the best. Therefore we used GRD of for our implementation.

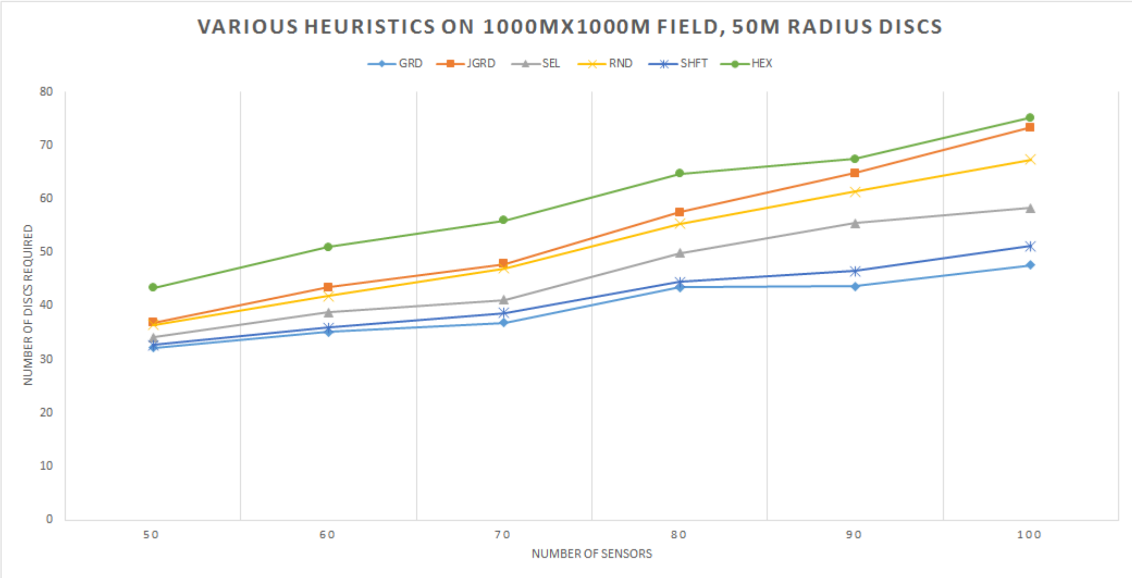


Figure 2.4: Results of testing above heuristics and the approximation scheme on a 1000m×1000m field

2.3 Euclidean minimum steiner trees

Definition (Euclidean minimum Steiner Tree problem). *Given a set P of points in a 2-D plane as input, the output is a network of line segements connecting all of the points in S, with the smallest total (Euclidean) length.*

The line segments making the Steiner Tree need just be incident on the points in S. This implies that the algorithm is free to use additional points from the plane, if necessary, to produce the smallest total length network. The additional points are called *Steiner points*

For computing Steiner tress, we use the exact solution finder software GeoSteiner [34] [35] [38]. Following figures show a disc cover of 100 sensors in a 1000m×1000m field with discs of radius 50m, and the Steiner trre of the disc centers thus computed.

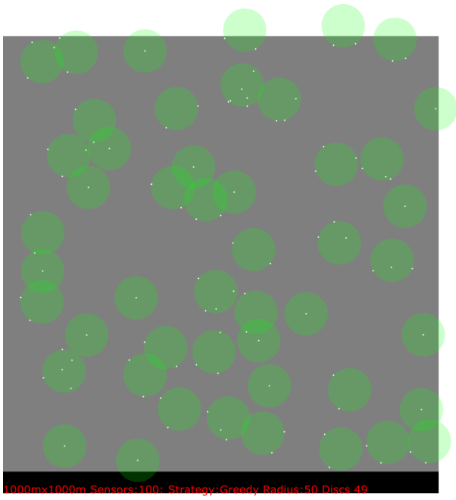


Figure 2.5: 50m radius disc cover of 100 sensors in 1000x1000 field

The white dots are sensor positions

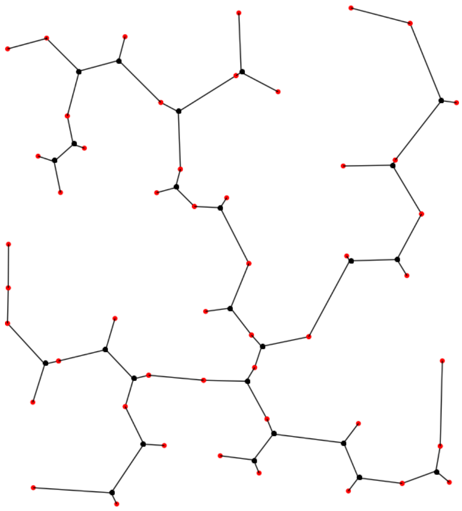


Figure 2.6: The Steiner tree of disc centers of figure 2.5

The black dots are Steiner points and the red dots are disc centers

Chapter 3

Related Works

Most notable related work is [19]. Their goal is to find the trajectories of k MULEs to collect data from the N wireless sensor nodes deployed on 2-D Euclidean space such that the data collection latency (the length of the longest trajectory among k trajectories) is minimized. They consider two cases:

Case 1: Each MULE is connected to the sink only from their original positions (base stations).

Case 2: Each MULE is connected to the sink directly at any time from any location.

Intuitively, it may be better to utilize MULE to MULE communication too for data transfer, which is absent from their approach, and is essential in our approach. They solve the problem by formulating k-traveling salesperson problem with neighborhood (k-TSPN) and k-rooted path cover problem with neighborhood (k-PCPN) problems. Their algorithm has a constant factor approximation, however, as they say in their section 5, their algorithm itself is based on another approximation algorithm [13], which is a very complicated rounding-based algorithm and difficult to use in some occasions. Therefore, they propose heuristics to solve it.

A similar problem of covering a set of points in a plane using multiple robots [7] also seemed promising, but, we were unable to accomodate the additional constraints listed above (they use Clique cover problem [17], which has its own application in data collection algorithms for dense sensor networks).

Another related work [21] is on SenCars [20].Their algorithm takes as input m the number of MULEs (SenCars), and the sensor network (sensor positions in the field). They first divide the sensor network into "polling points", then compute its Eulcidean MST. They put weights on polling points which estimates the time a SenCar will take to cover that point. The weight of a subtree is the sum of weights of its vertices and the sum of weights of its edges. Their aim is to divide the Euclidean MST into m more or less equal weight subtrees, which, they argue, is a good estimate of the time a Sencar will take on a data collection tour of that subtree. They solve using ILP and also provide a greedy heuristic.

The problem with graph partitioning approach is that the rendezvous of two neighbouring MULEs is not taken in account in the TSP evaluation of a partition. In [21], they simply assume that each SenCar can forward the gathered data to one of the nearby SenCars when they move close enough. Which is reasonable in a relatively denser sensor network (where MULEs' tours can be close enough to communicate to each other) because MULEs transfer data at high speed and they aggregate data before transmitting. This facilitates a MULE to transfer/recieve required data within the brief contact time. But in our case of a sparse sensor network, this approach (Euclidean MST partitioning) may lead to an isolated partition, whose MULE cannot communicate with any other partition, because its tour never gets close enough to another MULE's. Our approach ensures connectivity through ensuring MULE rendezvous at some point in the plane.

[27] is a connectivity restoration algorithm for a sparse sensor network. Its authors use both Stationary Relay nodes and Mobile data collectors, or mobile relay nodes (similar to MULEs) to restore connectivity in a set of sensor islands as their sensor network. They call each island a partition. Although RNs are best for providing stable link from cluster to the sink, there is only a limited number of stationary RNs they can use. For each partition they choose a representative sensor node, which has the responsilbility to collect and store the data from its partition. For these representative nodes, they compute the Steiner tree, for the determination of positions of relay nodes. Since stationary RNs are preferable to MDC's, initially all the relay nodes are mobile (i.e. are MDCs). Each of them visits a subset of representative nodes under bounded latency L. One by one they start connecting each partition to the sink through relay nodes, and at each step they check whether the remaining MDCs are still able to cover the remaining clusters under the bound latency.

Another work that uses Steiner tree to plan MULE path is [36]. Their work is applicable only on dense sensor networks, because they use RPs (rendezvous points) situated at steiner points to collect and buffer data from nodes farther away from the bases station for the MULE to visit and collect later. Ofcourse this means they have to be in range of their adjacent snesor in the Steiner tree, which can not be guarantedd in a sparse WSN.

Chapter 4

Path selection

Our approach to a better heuristic is driven by the concept of location graph and using the same for construction of low latency tours of the MULEs in data collection phase. The two critical subproblems encountered in this context are: (i) Euclidean Minimum Stiener Tree (EMST), and (ii) Euclidean Travelling Saleman Problem (ETSP). The EMST problem has already been introduced in chapter 2. We will briefly restate these here for sake of completeness and convenience in description of our heuristic.

Definition (Euclidean Travelling Salesman Problem). *Given a set of n 2D points in a plane find a minimum weight length tour of all the points, visiting every point exactly once.*

We use Christofides algorithm [9] as the approximation heuristic for computing TSP route of a set of points.

Definition (Euclidean minimum Steiner Tree problem). *Given a set P of points in a 2-D plane as input, the output is a network of line segements connecting all of the points in S, with the smallest total (Euclidean) length.*

The line segments making the Steiner Tree need just be incident on the points in S. This implies that the algorithm is free to use additional points from the plane, if necessary, to produce the smallest total length network. The additional points are called *Steiner Points*.

The location nodes (the nodes in location graph) computed earlier, form the set P, and the EMST is generated using the set P, and is called the location graph T. The pseudo code for the heuristic which appear later in section 4.1.1 uses the EMST as input. The choice of EMST as data structure over MST is guided by the fact that the weight (weight here means total of lengths of all the edges in a graph) of EMST is always atmost the weight of the MST, and we use the weight of a tree as an approximation of the weight of the TSP tour of its vertices.

The TSP time unit is the time taken by a MULE to visit all points on EMST. The cost of ETSP on a set is atmost twice the cost of the EMST on it. Therefore, it should be reasonable to use the Steiner tree traversal as a guide for partitioning. Furthermore, in the case where the field also includes obstacles, Steiner trees lend themselves naturally to cover all the points due the properties of Steiner points [39] as explained in section **??** of chapter 2. Though we plan not to cover the obstacle avoidance case, we briefly sketch the underlying ideas in Chapter 6.

4.1 Path Selection Heuristic

The aim of path selection heuristic is to find a minimum partitioning of the set of location nodes of a location graph by addition of extra Steiner points such that following conditions are satisfied.

- Each of the subsets has a TSP tour length (in units of time) less than a per-specified value L, and for any two sets S_i and S_j, $S_i \cap S_j \leq 1$.
- Let V be the set of all subsets S_i. Let E be the set of pairs of subsets (S_i, S_j) such that $S_i \cap S_j = 1$. Then the graph $G(V, E)$ should be a connected graph.

It assumed that the time a MULE spends in a network while collecting data consists of three main components: (i) travelling from one location node to another t_{TSP} (ii) Talking to sensors belonging to a location node t_{LS}, called MULE's pause time, at a location node (iii) Talking to other MULEs/Base station t_{MBS}.

We assume that MULE to MULE data transfer times are shorterdue to two reasons, namely, (i) MULEs may use data aggregation while sending data to fellow a MULE or BS, and (ii) MULEs being relative expensive and more robust than sensor node, typically have higher bandwidth for inter MULE data transfer. This is the reason, we ignore the contribution of t_{MBS}. The component t_{LS} for each location node can be given as an input (observed before running the heuristic, by simply sending one MULE on a tour of all the location nodes in the field to measure and record such times beforehand), or computed using sensor parameters, such as: Sensor data throughput (SDT, effective data rate, after taking into account the overhead introduced due to protocol headers) and Sensor data sampling rate (SSR, data sampling rate of the sensor, the speed of data acquisition of the sensor from its environment in bytes/sec).

4.1.1 Heuristic

The overall strategy is to create an EMST of the location nodes and then divide this tree into subgraphs, using the tree edges as the guide. Each subgraph's set of nodes will be covered by one MULE (henceforth, this set of nodes will be called a subtour). The term "weight" is equivalent to "time interval" in this algorithm, and is measured in seconds. An edge of the tree is said to have weight equal to its length divided by the speed of the MULE (time taken by the MULE to cover that edge). The weight of a location node is equal to the time a MULE has to wait there for data collection (called pause time). It depends on the latency bound and the number of sensor nodes covered by that location node. Steiner points have zero weight. The weight of the TSP tour of a set of nodes is equal to the smallest amount of time it tales for a travelling salesman to visit each node exactly once.

Consider any euclidean spanning tree of a set of 2D points in a plane (none of the points have any weight). Clearly, one way to visit all points would be to start from the root, and visit the nodes in the depth first search order, always travelling along the edges. This would take time equal to twise the weight of the tree (each edge travelled twice, once for going from parent to child, and once for coming back to the parent from the child). Thus, the optimal travelling salesman tour must be bounded by twice the weight of the tree.

Now consider the case, when the points have weights too (i.e. the travelling salesman has to wait at the point for a time equal to its weight). Then, the TSP approximation needs to be modified just by adding the total weight of all points in the tree.

Given any tree T, we start from the given root node $root$. $root$ then becomes the current node $curr$. Then following steps of Prim's algorithm [**?**], we first mark $curr$ as visited, then we insert all the incident edges of the current node with unvisited nodes to the min-heap $edgeHeap$. Computation of a tour for a MULE consists of two stages coded in two inner while loops.

Note: Since we use a 3/2 approximation algorithm [9] for computing the TSP tour of a subtour, the bound that we will use in the implementation will be actually $3\times$ weight of the tree. In general, if an x approximation algorithm for euclidean TSP is used, then the bound for TSP weight used should be $2 \times x \times$ weight of the tree.

Now we describe the algorithm. Let $boundary$ be a set of nodes of the steiner tree, initially containing any one node from the Steiner tree. The algorithm picks any one node from the $boundary$ and calls it $root$. The algorithm then computes a connected subgraph of the Steiner tree (containing that root node) with bounded TSP time. The nodes of this subgraph form a subtour, covered by one MULE. The boundary nodes of a subgraph are those nodes of the steiner tree (regardless of whether they are location nodes or Steiner points), which belong to the subgraph, and do not have all their adjacent nodes in the subgraph. All the boundary nodes from this subgraph are inserted into $boundary$, and the algorithm is called again. This continues until all the nodes are part of exactly one subtour.

Preliminaries

Some data structures and inputs used in the pseudocode are as follows. T is the Steiner tree of the location nodes, given as input to the algorithm. ASP is the average sensor pause time, calculated using L (latency bound), SDT (sensor data throughput) and SSR (sensor data sampling rate). These three are given as input to the algorithm. The map w, taken as input by the algorithm, is the map from the nodes in the Steiner tree T to the number of sensors they cover. Each location nodes will have a non zero entry in the map w, whereas all the Steiner points will have zero entry in the map. For instance, for a node v, $w[v]$ is the number of sensors covered by v. $edgeHeap$ is a min-heap of the pairs $(tWeight, edge)$, where $edge$ is a pair of nodes (v_1, v_2), and $tWeight = 3 \times weight\_of\_edge(v_1, v_2) + ASP \times w[v_1]$. The ordering in the min-heap is according to its first argument $tWeight$. To push an edge into the heap $edgeHeap$ means to calculate the $tWeight$ of the edge, then inserting the pair $(tWeight, edge)$ in the heap.

Computation of subtours

Computation of subtours begins from the picking of a node form the set $boundary$. This node will be called $root$ for this subtour. Before the computation of the subtour, we mark $root$ as visited, and for all nodes v adjacent to $root$, we push the edges $(v, root)$ into $edgeHeap$.

The subtour is computed in two stages. In the first stage, represented by the first inner while loop, $currSet$ contains the nodes currently included in the subtour being computed. $currT$ is the upper bound of the TSP tour of the nodes in $currSet$; it is updated every time we insert a node into $currSet$. Before popping a pair $(tWeight, edge)$ from $edgeHeap$, we first check for the condition, whether $currT + tWeight$ is greater than L; if it is, first stage ends here, and we break from the while loop. Otherwise, we pop the pair $(tWeight, (v1, v2))$ from $edgeHeap$, all the edges incident to $v1$ which have unvisited end nodes are inserted into the heap, $v1$ is inserted to $currSet$, and $currT+ = tWeight$.

We keep popping edges from $edgeHeap$ until either $edgeHeap$ becomes empty or, adding any more nodes to $currSet$, leads to $currT$, our current estimate of the weight of the TSP tour of $currSet$, becoming greater than L, our given latency bound. By this time, we are sure that the TSP tour of the nodes in $currSet$ will not exceed L.

Observe that any Steiner point whose all adjacent nodes belong to same subtour is useless. Because, such a Steiner point neither serves as a connecting node between different subtours, nor represents a location node. So, such a Steiner point should be eliminated form the subtour. The function $cleanTour$ is used on $currT$ for this purpose, and the first stage ends here.

In the beginning of the second stage, although we are sure that the TSP tour of the nodes in $currSet$ will not exceed L, but the actual weight of the TSP tour of the nodes in $currSet$ might be low enough to add still more nodes into $currSet$. For testing whether this is possible or not, first we compute the actual weight of the TSP tour of $currSet$. Then, before adding a node to $currSet$, we test whether its inclusion will make $currT$ exceed L or not. If yes, then $currSet$ is the final subtour for this MULE. Otherwise, the second stage repeats.

The edge records still left in the Heap, after completion of one full iteration of the second inner while loop, form the boundary of the nodes in $currSet$. These nodes are pushed in the $boundary$, from where the next $root$ for the next MULE's subtour computation is chosen.

Pseudocode

Algorithm 1 Dividing the set nodes with weights of a given Steiner tree into subsets of bounded TSP time

function GREEDYSTEINER(A Steiner tree T of location in a plane, An array w of the number of sensors under a location node, Starting vertex $root$, desired upper bound on latency L) ▷ this function returns S: Set of tours, each with touring time $\leq$ L

    Set S
    $N \leftarrow$ number of vertices in T
    $hApprox \leftarrow 3.0$
    $MSPEED \leftarrow$ speed of the MULE used
    $SDT \leftarrow$ sensor data throughput
    $SSR \leftarrow$ sensor data sampling rate
    $ASP \leftarrow \frac{(L \times SSR)}{SDT}$ (average sensor pause time for one sensor)
    $queue$ boundary.push($root$)
    $bool\ visited[N]$
    $vertex$ curr

```
for i ← 1, N do
    visited[i] ← false
end for
while 1 do
    if boundary.empty() then
        break
    end if
    currT ← 0.0
    cycleWeight ← 0.0
    Min_heap edgeHeap
    curr ← root
    Set currSet, tempSet
    visited[curr] ← true
    Set U ← all unvisited vertices adjacent to cur
    for all vertex v in U do
        edgeHeap.push((hApprox × dist(v, curr)) + (ASP × w[curr]), (v, curr))
    end for
    currSet.insert(curr)
    while ¬ edgeHeap.empty() do
        nextWeight ← edgeHeap.top().first
        currT ← currT + nextWeight
        if currT ≥ hApprox × L then break
        end if
        curr ← edgeHeap.top().second.first
        currSet.insert(curr)
        visited[curr] ← true
        edgeHeap.pop()
        U.clear()
        U ← all unvisited vertices adjacent to cur
        for all vertex v in U do
            edgeHeap.push((hApprox × dist(v, curr)) + (ASP × w[curr]), (v, curr))
        end for
    end while
    cleanTour(currSet)
    Tour currTour
    tempSet ← currSet
    cycleWeight, currTour ← TSPCircuit(tempSet)
    for all sensor s in currTour do cycleWeight ← cycleWeight + ASP * w[s]
    end for
    while ¬ edgeHeap.empty() and cycleWeight < L do
        curr ← edgeHeap.top().second.first
        tempSet.insert(curr);
        cleanTour(tempSet)
        cycleWeight, currTour ← TSPCircuit(tempSet)
```

```
            for all  sensor $s$ in $currTour$ do $cycleWeight \leftarrow cycleWeight + ASP * w[s]$
            end for
            if $cycleWeight > L$ then
                break
            end if
            currSet.insert($curr$);
            $visited[curr] \leftarrow true$
            edgeHeap.pop()
            $U$.clear()
            $U \leftarrow$ all unvisited vertices adjacent to $cur$
            for all  vertex $v$ in U do
                $edgeHeap.push((hApprox \times dist(v, curr)) + (ASP \times w[curr]), (v, curr))$
            end for
        end while
        $S$.insert($currTour$)
        while ¬ heap.empty() do
            $edge\ e = edgeHeap$.top().second;
            $boundary$.push($e$.second);
            $edgeHeap$.pop();
        end while
        if $boundary$.empty() then
            return false
        end if
    end while
end function
function CLEANTOUR($vSet$ : the set of vertices in the current tour)   ▷ Delete all Steiner
vertices from $vSet$, whose all adjacent verices are in $vSet$ itself.
end function
```

Chapter 5

Results

5.1 Experiment setup

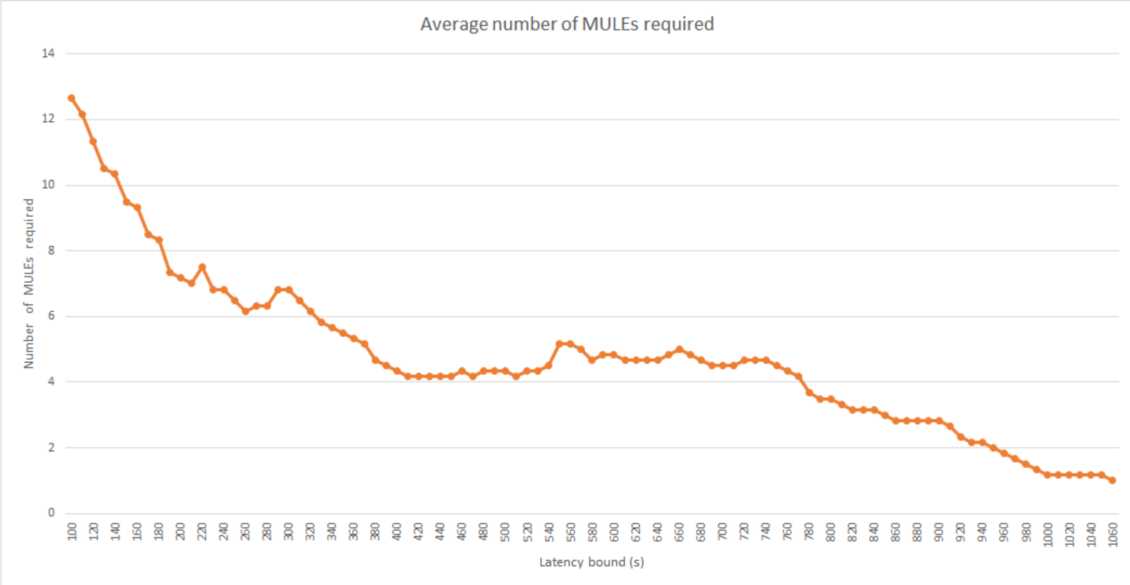
We use [32] for sensor specification. The sensors are ZigBee temperature sensors, collecting data at 1 reading per second. [32] have used 12 bits for 1 temperature reading, therefore our SSR (sensor data sampling rate) should be 1.25 byte/second. The SDT (sensor data throughput) is taken to be 13.4 kbps [26]. The MULEs' speed is taken to be 10m/s [30]. The sensor positions are chosen randomly in a field of 1000mx1000m, and the sensor range is 50m (Zigbee sensor ranges can lie between 10m to 100m).

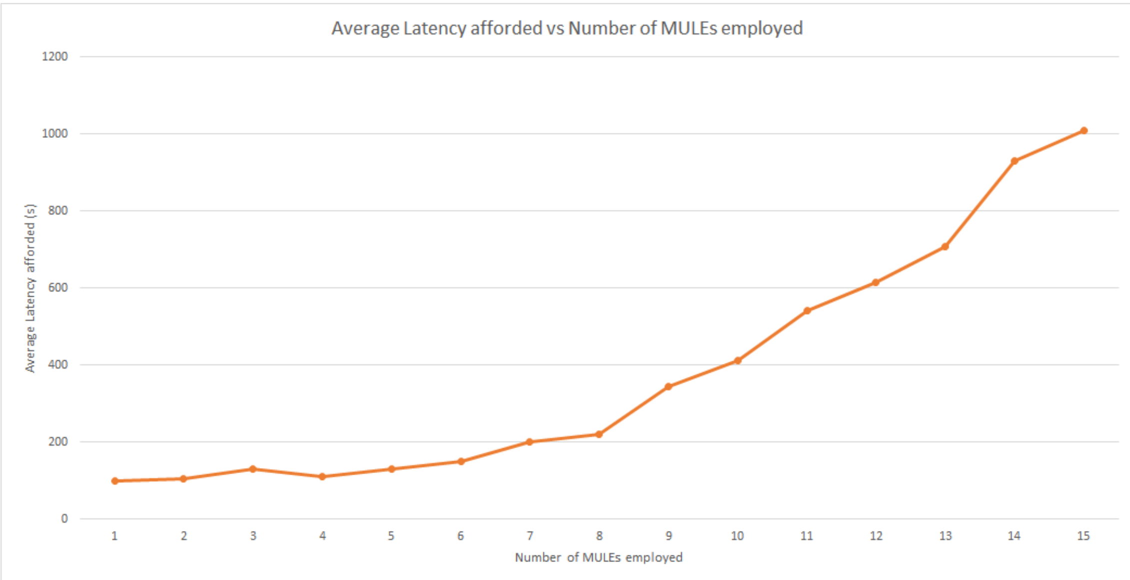
5.2 Simulation results

We now present simulation results of our heuristic applied on the following cases:

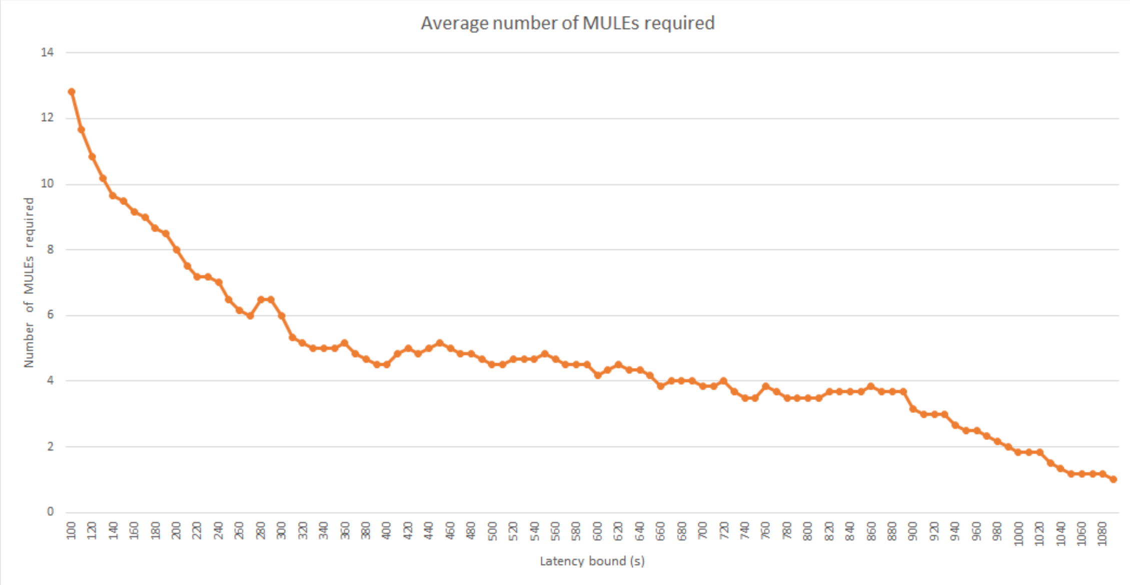
- Field size 1000m $\times$ 1000m
- Sensor Range: 50 m
- Number of sensors: 50, 60, 70, 80, 90, 100

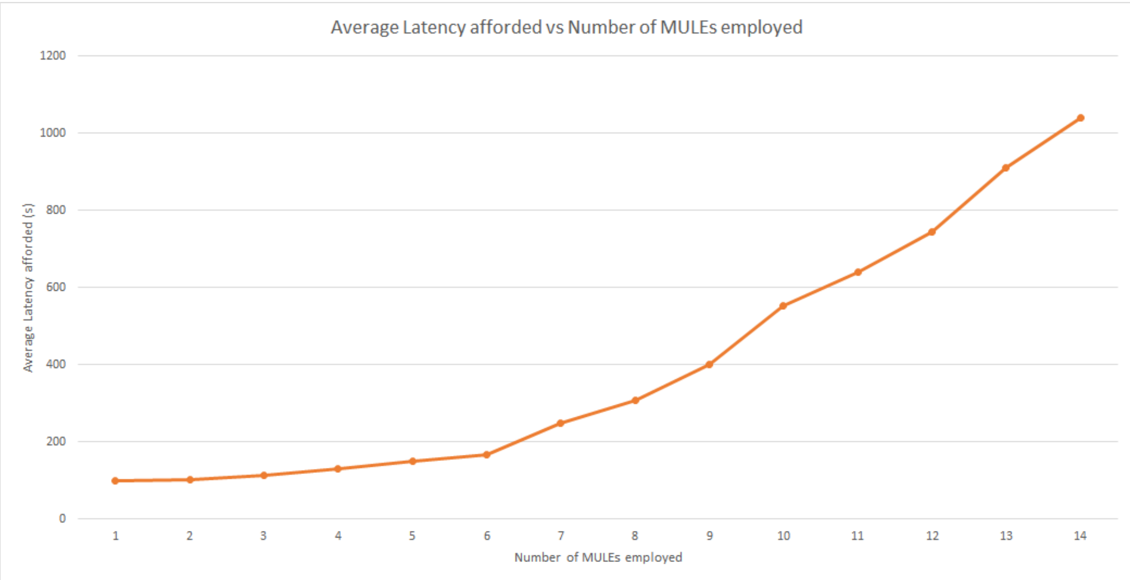
5.2.1 50 sensors



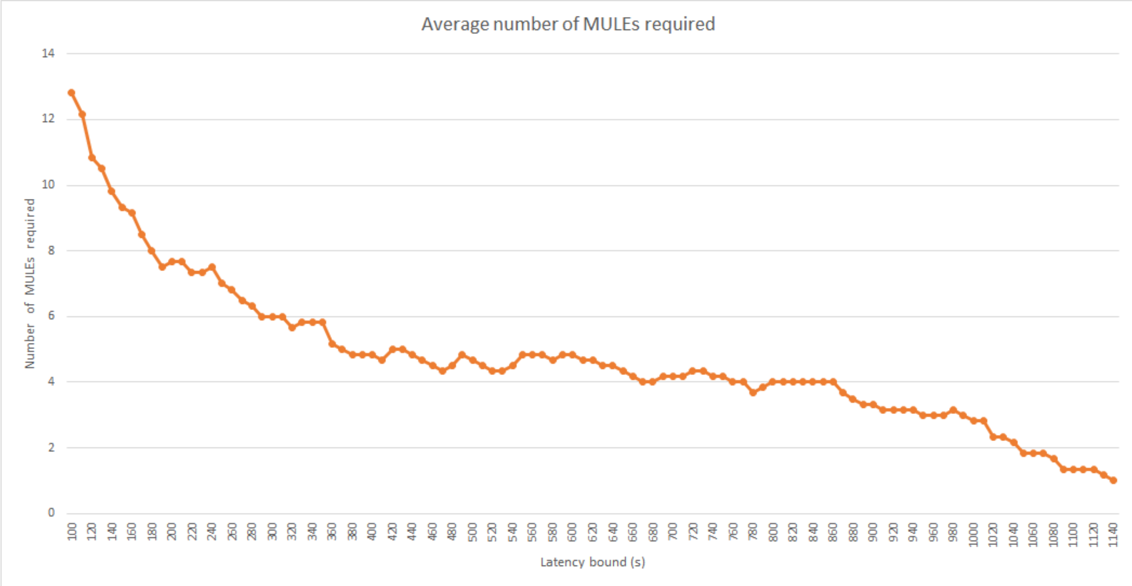


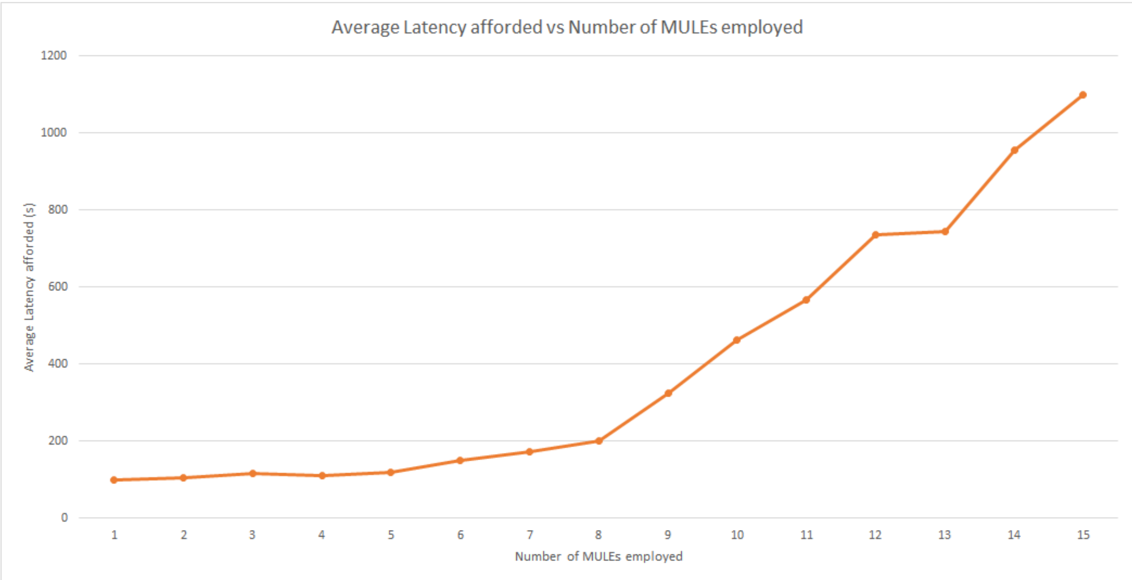
5.2.2 60 sensors



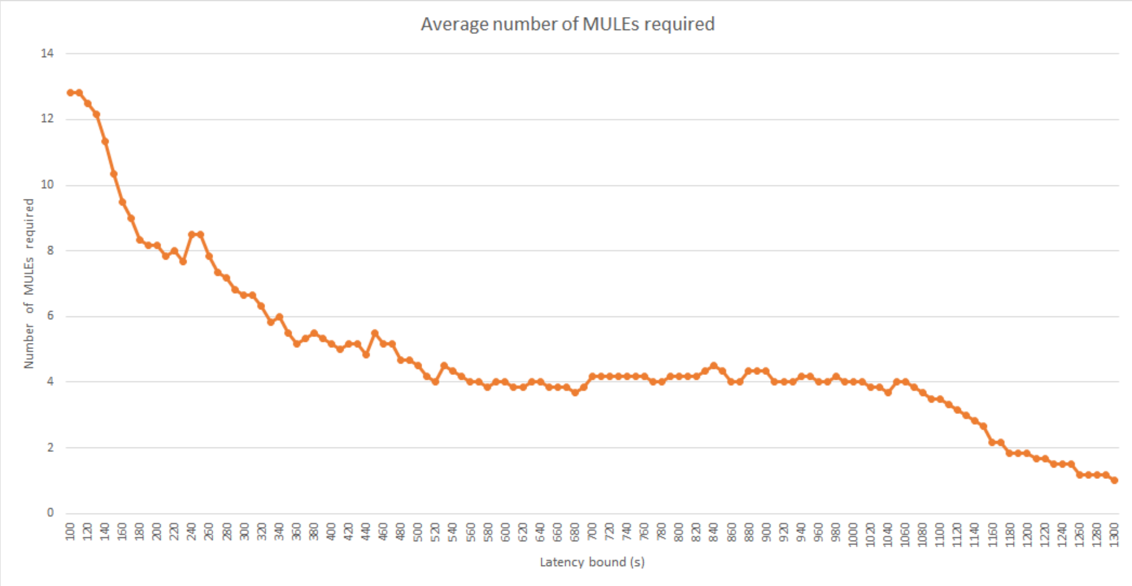


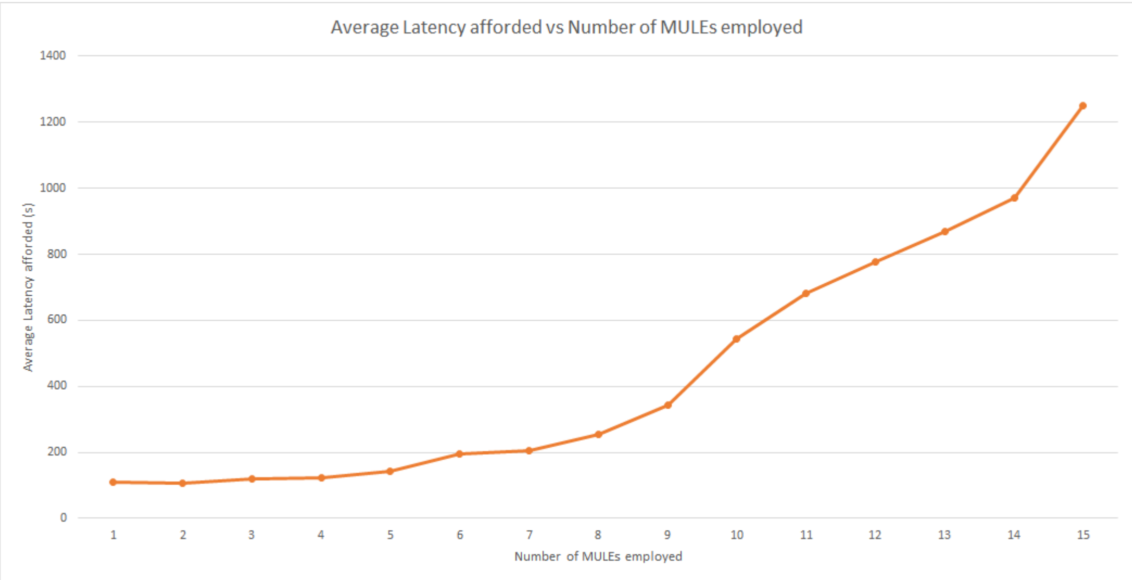
5.2.3 70 sensors



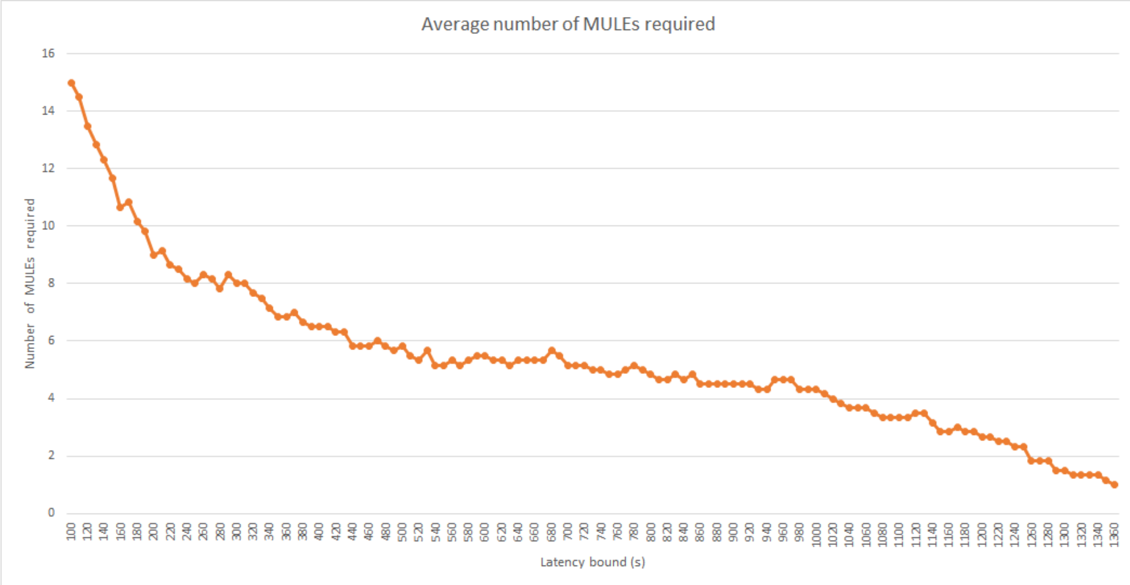


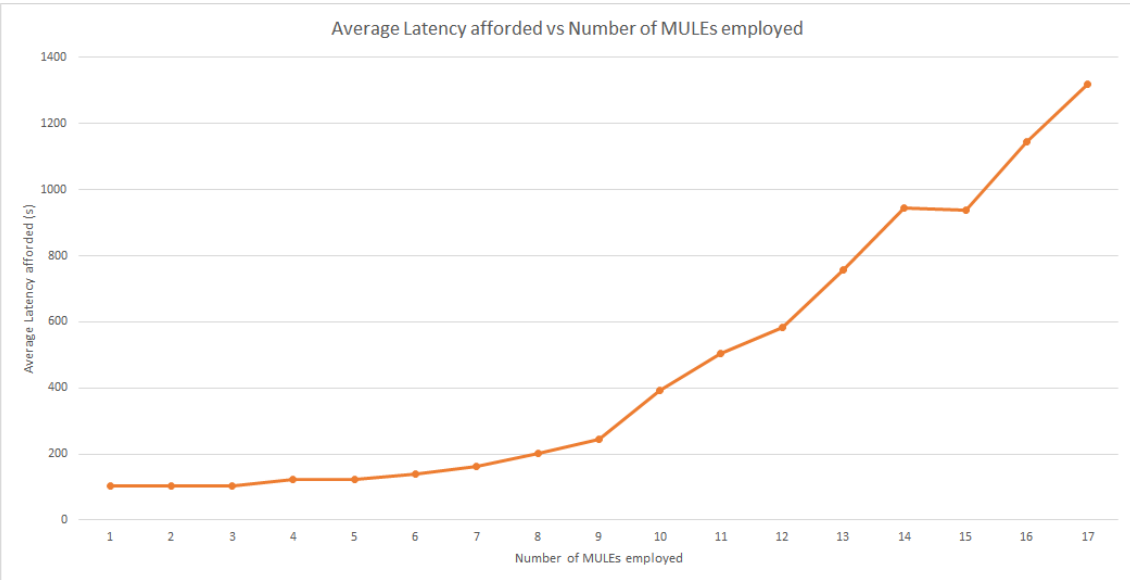
5.2.4 80 sensors



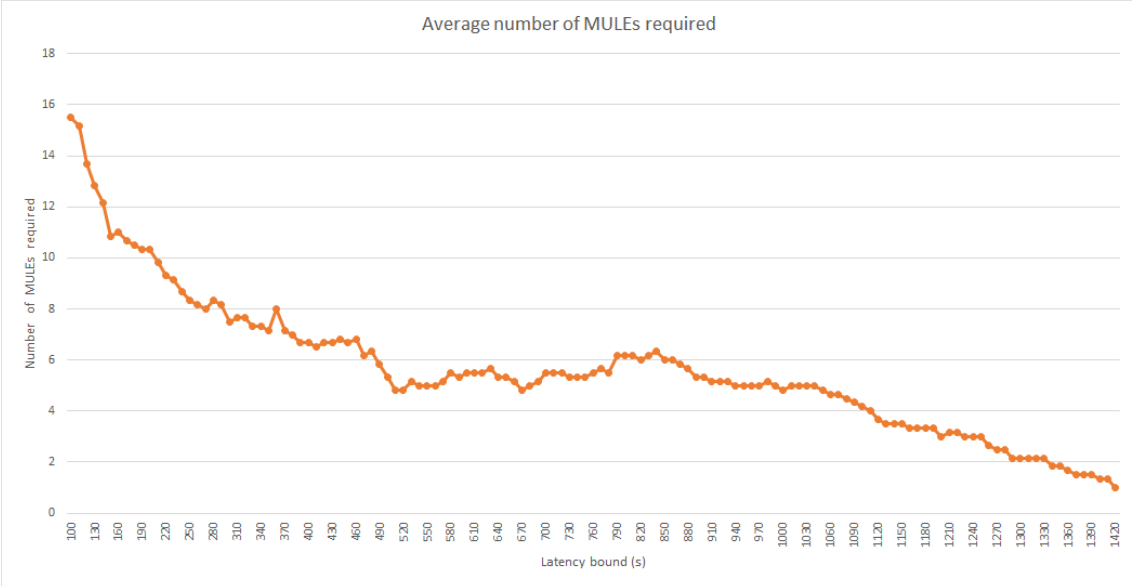


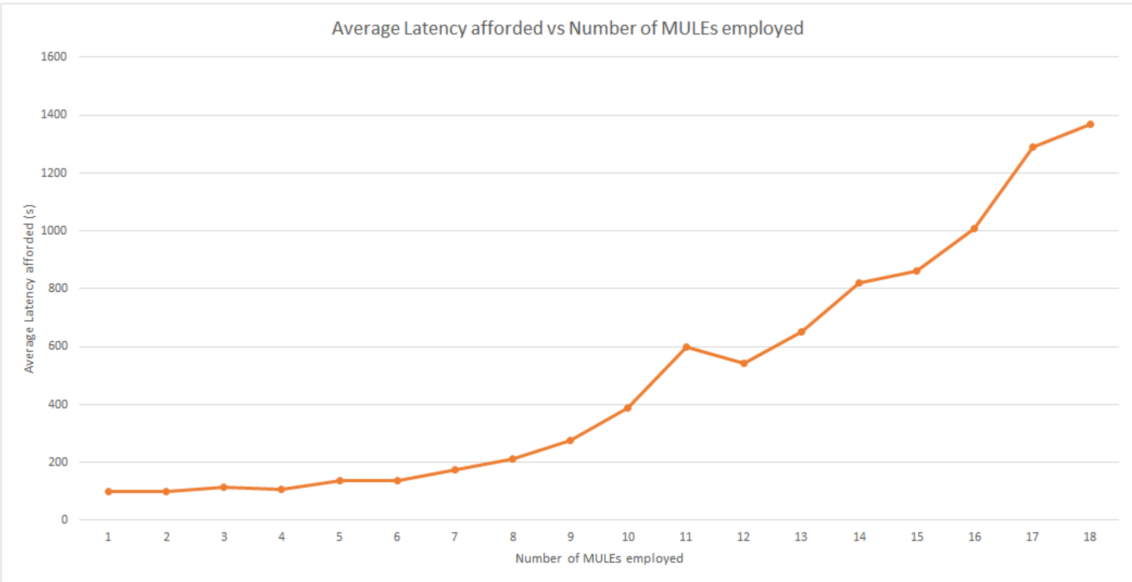
5.2.5 90 sensors





5.2.6 100 sensors





Chapter 6

Conclusions and Future work

In this thesis, we have provided a heuristic to collect data from a given sparse sensor network field and a latency bound. The heuristic outputs the number of MULEs required for achieving the latency goal. We have also deduced the minimum latency supportable for any sensor network field which uses this heuristic for data collection.

However, we believe that the potential of application of Steiner trees for data collection may go further.

6.1 Data collection with convex obstacle avoidance

Consider the current problem of data collection from a sparse sensor network, inside a simple convex polygon as the containing field. If we place 2D obstacles in the field in the shape of convex polygons, the problem becomes data collection with convex obstacle avoidance.

The Obstacle Avoiding Euclidean Steiner tree (OAEST) problem [39] is already known; Given a set P of points in a 2D plane contained within a polygon, with polygon holes inside it as obstacles, compute an EMST, such that no edge of the required graph may intersect with either the polygon boundary or the obstacle polygons inside it.

If there is only single polygon obstacle, then [2] can be used. Solving the problem for multiple convex polygonal obstacles is suggested as future work here.

6.2 Further optimization on TSP tour of a MULE

We can apply the work of [37] here. First, we have to cover the sensor field with discs of radius *half* the range of the sensors. This way, the MULE need not travel to the center of a location node disc for data collection, and the location node can be treated as a communication area [37]. Then after applying their algorithm on our set of location nodes, we can reduce our TSP time by 15-20%.

Bibliography

[1] Underwater robots track oil and ocean life. http://spectrum.ieee.org/podcast/robotics/industrial-robots/underwater-robots-track-oil-and-ocean-life. Accessed: 09/07/2014.

[2] Jeff Abrahamson, Ali Shokoufandeh, and Pawel Winter. Euclidean tsp between two nested convex obstacles. *Inf. Process. Lett.*, 95(2):370–375, July 2005.

[3] Kemal Akkaya and Mohamed Younis. A survey on routing protocols for wireless sensor networks. *Ad Hoc Networks*, 3:325–349, 2005.

[4] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. *Comput. Netw.*, 38(4):393–422, March 2002.

[5] He Ba, Ilker Demirkol, and Wendi Heinzelman. Passive wake-up radios: From devices to applications. *Ad Hoc Netw.*, 11(8):2605–2621, November 2013.

[6] Paz Carmi, Matthew J. Katz, and Nissan Lev-Tov. Covering points by unit disks of fixed location. In *Proceedings of the 18th International Conference on Algorithms and Computation*, ISAAC'07, pages 644–655, Berlin, Heidelberg, 2007. Springer-Verlag.

[7] N. Chakraborty, S. Akella, and J.T. Wen. Coverage of a planar point set with multiple robots subject to geometric constraints. *Automation Science and Engineering, IEEE Transactions on*, 7(1):111–122, Jan 2010.

[8] Tzung-Cheng Chen, Tzung-Shi Chen, and Ping-Wen Wu. Data collection in wireless sensor networks assisted by mobile collector. In *Wireless Days, 2008. WD '08. 1st IFIP*, pages 1–5, Nov 2008.

[9] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[10] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):pp. 233–235, 1979.

[11] Jun-Hong Cui, Jiejun Kong, M. Gerla, and Shengli Zhou. The challenges of building mobile underwater wireless networks for aquatic applications. *Network, IEEE*, 20(3):12–18, May 2006.

[12] Mario Di Francesco, Sajal K. Das, and Giuseppe Anastasi. Data collection in wireless sensor networks with mobile elements: A survey. *ACM Trans. Sen. Netw.*, 8(1):7:1–7:31, August 2011.

[13] G. Even, N. Garg, J. KöNemann, R. Ravi, and A. Sinha. Min-max tree covers of graphs. *Oper. Res. Lett.*, 32(4):309–315, July 2004.

[14] Massimo Franceschetti, Matthew Cook, and Jehoshua Bruck. A geometric theorem for approximate disk covering algorithms. Technical report, 2001.

[15] M. Hefeeda and M. Bagheri. Wireless sensor networks for early detection of forest fires. In *Mobile Adhoc and Sensor Systems, 2007. MASS 2007. IEEE Internatonal Conference on*, pages 1–6, Oct 2007.

[16] Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *J. ACM*, 32(1):130–136, January 1985.

[17] Xiaoyun Ji and John E. Mitchell. The clique partition problem with minimum clique size requirement. 2005.

[18] Raja Jurdak, Pierre Baldi, and Cristina Videira Lopes. Adaptive low power listening for wireless sensor networks. *IEEE Transactions on Mobile Computing*, 6(8):988–1004, August 2007.

[19] Donghyun Kim, R.N. Uma, B.H. Abay, Weili Wu, Wei Wang, and AO. Tokuta. Minimum latency multiple data mule trajectory planning in wireless sensor networks. *Mobile Computing, IEEE Transactions on*, 13(4):838–851, April 2014.

[20] Ming Ma and Yuanyuan Yang. Sencar: An energy-efficient data gathering mechanism for large-scale multihop sensor networks. *Parallel and Distributed Systems, IEEE Transactions on*, 18(10):1476–1488, Oct 2007.

[21] Ming Ma and Yuanyuan Yang. Data gathering in wireless sensor networks with mobile collectors. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–9, April 2008.

[22] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, WSNA '02, pages 88–97, New York, NY, USA, 2002. ACM.

[23] Adam Martinez, Timothy Norfolk, and Kathy Liszka. A geometric tiling algorithm for wireless sensor networks. 2012.

[24] Xu Ning and Christos G. Cassandras. Dynamic sleep time control in wireless sensor networks. *ACM Trans. Sen. Netw.*, 6(3):21:1–21:37, June 2010.

[25] Grammati Pantziou, Aristides Mpitziopoulos, Damianos Gavalas, Charalampos Konstantopoulos, and Basilis Mamalis. Mobile sinks for information retrieval from cluster-based wsn islands. In *Proceedings of the 8th International Conference on Ad-Hoc, Mobile and Wireless Networks*, ADHOC-NOW '09, pages 213–226, Berlin, Heidelberg, 2009. Springer-Verlag.

[26] E.D. Pinedo-Frausto and J.A Garcia-Macias. An experimental analysis of zigbee networks. In *Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on*, pages 723–729, Oct 2008.

[27] IF. Senturk, K. Akkaya, F. Senel, and M. Younis. Connectivity restoration in disjoint wireless sensor networks using limited number of mobile relays. In *Communications (ICC), 2013 IEEE International Conference on*, pages 1630–1634, June 2013.

[28] Izzet F. Senturk and Kemal Akkaya. Mobile data collector assignment and scheduling for minimizing data delay in partitioned wireless sensor networks. In *ADHOCNETS*, volume 129 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 15–31. Springer, 2013.

[29] R.C. Shah, S. Roy, S. Jain, and W. Brunette. Data mules: modeling a three-tier architecture for sparse sensor networks. In *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*, pages 30–41, May 2003.

[30] Ryo Sugihara. Controlled mobility in sensor networks. 2009.

[31] Ryo Sugihara and Rajesh K. Gupta. Speed control and scheduling of data mules in sensor networks. *ACM Trans. Sen. Netw.*, 7(1):4:1–4:29, August 2010.

[32] Mitsugu Terada. Application of zigbee sensor network to data acquisition and monitoring. *MEASUREMENT SCIENCE REVIEW*, 9(6):2–3, 2009.

[33] A Vallimayil, V.R.S. Dhulipala, K.M.K. Raghunath, and R. M. Chandrasekaran. Role of relay node in wireless sensor network: A survey. In *Electronics Computer Technology (ICECT), 2011 3rd International Conference on*, volume 5, pages 160–167, April 2011.

[34] David Michael Warme. *Spanning Trees in Hypergraphs with Applications to Steiner Trees*. PhD thesis, Charlottesville, VA, USA, 1998. AAI9840474.

[35] Pawel Winter and Martin Zachariasen. Euclidean steiner minimum trees: An improved exact algorithm. *Networks*, 30(3):149–166, 1997.

[36] Guoliang Xing, Tian Wang, Weijia Jia, and Minming Li. Rendezvous design algorithms for wireless sensor networks with a mobile base station. In *Proceedings of the 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '08, pages 231–240, New York, NY, USA, 2008. ACM.

[37] Ronghua Xu, Hongjun Dai, FengYu Wang, and Zhiping Jia. A convex hull based optimization to reduce the data delivery latency of the mobile elements in wireless sensor networks. In *High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC EUC), 2013 IEEE 10th International Conference on*, pages 2245–2252, Nov 2013.

[38] Martin Zachariasen. Rectilinear full steiner tree generation. *NETWORKS*, 33(2):125–143, 1999.

[39] Martin Zachariasen and Pawel Winter. Obstacle-avoiding euclidean steiner trees in the plane: An exact algorithm. In MichaelT. Goodrich and CatherineC. McGeoch, editors, *Algorithm Engineering and Experimentation*, volume 1619 of *Lecture Notes in Computer Science*, pages 286–299. Springer Berlin Heidelberg, 1999.