Mobile Information Systems 6 (2010) 155–176
DOI 10.3233/MIS-2010-0097
IOS Press

Sensor relocation for emergent data acquisition in sparse mobile sensor networks

Wei Wu[a,*], Xiaohui Li[a], Shili Xiang[a], Hock Beng Lim[b] and Kian-Lee Tan[a]
[a]*School of Computing, National University of Singapore, Singapore*
[b]*Intelligent Systems Centre, Nanyang Technological University, Singapore*

Abstract. In this paper, we study the problem of sensor relocation for emergent data acquisition (initiated by a base station) in sparse mobile sensor networks. We propose a distributed scheme called BRIDGE that relocates mobile sensors to fulfill an emergent data acquisition task with the objective to minimize the task completion time. BRIDGE gradually finds a sensor that is close to the task location and relocates that sensor to the task location, and at the same time relocates some other sensors to connect that sensor to the base station. BRIDGE exploits the encountered sensors during relocation, and handles the challenges caused by intermittent connections. Our extensive performance study shows the effectiveness of our proposed scheme.

Keywords: Sensor network, data acquisition, sensor relocation

1. Introduction

Recently an increasing number of research activities are being carried out for Mobile Sensor Networks (MSNs) [1,5,8,12,15,20] where the sensors are capable of moving. The mobility of the sensors increases the sensors' coverage [13], and makes them possible to adapt to the environments, because the sensors are not constrained by the initial deployment and can be relocated to desirable locations when necessary.

We in this paper consider a class of MSNs applications which involve a base station and a number of mobile sensors. The sensors' basic task is to explore a large area and send collected information to the base station. The base station sometimes may have an emergent data acquisition task that requires one sensor to sense a specific location and send the data back to the base station as soon as possible.

For the sake of concreteness, let us look at an **application example**. In battlefield, a command center (CC) dispatches a small number of UAVs (Unmanned Aerial Vehicle) to scan a big area. These UAVs scan the area by taking pictures of the region that they fly over. They send back the pictures to the CC when they have wireless connection to it. Based on the pictures received or some other intelligence sources, the experts at the CC may find a region suspicious. When this happens, the CC issues an emergent task about that region and wants the UAVs to collect detailed information about that region and send the data back as soon as possible.

*Corresponding author: Wei Wu, School of Computing, National University of Singapore, Singapore 117417. Tel.: +65 6516 1424; Fax: +65 6779 4580; E-mail: wuw@nus.edu.sg.

We study the problem of relocating mobile sensors to carry out the emergent data acquisition tasks in *sparse* MSNs where the connections among the nodes (sensors and the base station) are intermittent. The aim is to make the sensors complete an emergent task as early as possible.

We study this problem in the context of sparse MSNs because of two reasons. First, MSNs are likely to be sparse. Mobile sensors are more expensive than stationary sensors, so it may not be feasible to deploy a large number of them. Moreover, dense MSNs can become sparse due to node failures caused by environmental hazards or even intentional demages (e.g. by adversaries in the battlefield). Second, solutions designed for sparse MSNs are more robust and versatile, because they will also work well in dense MSNs.

Although several works have studied the mobile sensor relocation problem [5,11,22,23,25], our problem is different in relocation objectives and in system settings and therefore existing approaches cannot be used to solve our problem. Existing works investigate the problem of relocating certain number of sensors to a region so that the region is covered by the sensors with a certain density. We are interested in relocating some sensors so that an emergent data acquisition task can be fulfilled in a short period of time. Existing works assume a system where all the sensors are connected and reachable. We look at a system where the sensor network is sparse and the connections in the system are intermittent.

Relocating sensors to fulfill an emergent data acquisition task in a sparse MSNs is challenging. If all the sensors are connected, then the following straightforward solution would be good enough: find a sensor near the task location, let it move to the task location to collect data, and use connected nodes to relay the data back to the base station. However, in sparse networks, this straightforward solution will not work well, because: 1) when the base station issues the task, most sensors are not reachable, therefore the base station does not know which sensor is near the task location and have no way to contact that sensor because the network is partitioned; 2) the base station therefore has to select a sensor that is connected to it to carry out the task; it may take much time for the sensor to move to the task location; 3) after collecting the data from the task location, the sensor may have no connection to the base station so it cannot send the acquired data directly back to the base station; it has to move towards the base station; again, it may need to move a long way before it is connected to the base station.

We propose a distributed sensor relocation scheme, called BRIDGE, by which the mobile sensors relocate themselves to carry out an emergent data acquisition task cooperatively. The main idea is to gradually find a sensor that is close to the task location and at the same time relocate some sensors to build a connection between the sensor at the task location and the base station. BRIDGE exploits the connected and encountered sensors to minimize the task completion time, by relocating proper sensors to proper locations.

The contributions of this paper are:

1. We identify an interesting problem: relocation of mobile sensors for emergent data acquisition in sparse mobile sensor networks.
2. We propose a distributed relocation scheme in which the mobile sensors relocate themselves to fulfill an emergent data acquisition task through collaboration. In the scheme we deal with the various problems caused by intermittent connections.
3. We show through an extensive simulation study that the proposed scheme is effective.

The rest of the paper is organized as follows. In Section 2 we briefly survey the related works. In Section 3 we describe the system model and the emergent data acquisition task, and define the associated sensor relocation problem. We present our distributed relocation scheme in Sections 4. Results of experimental study are shown in Section 5. We finally conclude the paper in Section 6.

2. Related works

In static wireless sensor networks, various techniques [3,7,16,26] have been proposed to organize sensors into logical structures to facilitate data collection and query processing. Unfortunately, these techniques cannot be applied to mobile sensor networks where the topology of the network is very dynamic.

Task execution using mobile sensors (e.g., UAVs) has attracted much research attention. In [21], multiple mobile UAVs cooperate to facilitate probabilistic information fusion to achieve high-accuracy environment perception and target tracking. The objective is to determine the actions a UAV should carry out so as to maximize the belief of the current information. This is different from our objective, which is to minimize the time for fulfilling an emergent data acquisition task. There are also works on multi-task allocation and path planning for cooperating UAVs, to minimize the task completion time, using market based approach [9] or mixed-integer linear programming [2]. However, these works did not consider the opportunity that we exploit in this paper, that is, some mobile sensors could be relocated to certain locations to relay information for a task, to further reduce the task completion time.

[6,14,18,24,28] propose methods for data collection using mobile elements in wireless sensor networks or mobile Ad-Hoc networks. The basic idea is to use mobile elements as message carriers. They collect information from sensors when they are in close range to the sensors, buffer the information when they move around, and pass the information to the base station when they become near to the base station. In these works, the time a mobile element takes to deliver the information is not critical. Our work differs from them in that we want to reduce the time the sensors take to fulfill an emergent data acquisition task.

The authors of [27] propose a data acquisition framework called SenseSwarm for mobile sensor networks. SenseSwarm partitions the sensors into perimeter and core nodes. Perimeter nodes are responsible for data acquisition while core nodes take care of storage and replication. The aim of the SenseSwarm framework is to improve data availability.

[5,11,22,23,25] study the mobile sensor relocation problem [23,25] focus on fine-grained relocation to deal with a coverage hole caused by a sensor failure [5,22] investigate event-based relocation where the sensor locations and density are adapted to properly sense and control a large event area [11] proposes self-deployment algorithms for sensors to achieve a focused coverage around a Point of Interest. In all these studies, the sensor networks under consideration are assumed to be fully connected, and the aim of relocation or self-deployment is to meet a certain coverage requirement. As mentioned in Section 1, the differences between our work and these existing studies are twofold: we look at the sensor relocation problem in sparse sensor networks; the objective of relocation in our work is to fulfill an emergent data acquisition task as soon as possible.

3. System model and problem definition

3.1. System model

The system consists of a stationary base station BS and n mobile sensors (s_1, s_2, ..., s_n) that are sparsely distributed in an area A. Each mobile sensor knows the BS's location and its own location. The mobile sensors and the BS use wireless technology (such as Wi-Fi) for communication and there is no direct long-range communication. Two sensors (or the BS and a sensor) can communicate *directly* only if the distance between them is smaller than the wireless technology's communication range r. The sensors and the BS form a mobile ad-hoc network (MANET) where one can communicate with another

if they are connected either directly or through other sensors. Since the sensors' communication range is limited and the sensors are sparse, the network formed by the sensors is not fully connected, and can even be severely partitioned. The topology of the network changes with time as the sensors move.

The general task of the mobile sensors is to explore (sense) the area A by moving in it following a certain mobility pattern. The mobile sensors' move speed is v. Each sensor senses the region that it passes by, carries the sensed data, and forwards the data to the BS when it is connected to the BS.

3.1.1. Emergent data acquisition task

An emergent data acquisition task (emergent task for short) $ET(L)$ specifies a location L (within A). Given an emergent task $ET(L)$, the mobile sensors shall carry out the task by having one sensor going to L, sensing for a period of time T_s, and sending the sensed data back to the BS. T_s models the time a sensor needs to collect enough information around L, and the length of T_s is determined by the applications. The BS would like the time from the moment an emergent task is issued to the moment relevant data is received to be as short as possible.

Note that when an emergent task arises, it is possible that only a small number of sensors are connected to the BS.

3.1.2. Assumptions

Since the focus of this work is on sensor relocation scheme and our optimization metric is the time the sensors take to fulfill an emergent task, for simplicity we in this paper do not consider energy consumption. In the class of applications that we are considering, all the sensors are moving to collect information. The energy spent on moving will be much more than the energy spent on communication. [17] shows that when a 0.5 kg UAV flies horizontally at a 10–12 m/s speed its minimum energy consumption is 10–25 J(joule). It is reported in [4] that a normal (Lucent) IEEE802.11 wireless network card consumes about 1.5 J per second when in active transmission mode. Although more powerful wireless transmitters may be used, they will be equipped only on larger UAVs. Clearly, larger UAVs will need much more energy for flying, or for simply staying in the air. It is also mentioned in [19,23] that a mobile sensor on ground spends much more energy on motion than on wireless communication when moving around.

Since all the sensors are moving most of the time, they will spend similar amount of energy no matter how they communicate with each other and how they move during a relocation. For this reason, we believe that: 1) trying to save energy by controlling the communication between the sensors will not be very helpful; 2) although controlling the move distances of the sensors during the relocations may help save some energy, the save will be marginal because the relocation only happens during the ad-hoc emergent tasks.

We will also neglect data transmission time, because we believe that in a sparse MSN, the relocation time will be much longer than the data transmission time so it does not affect our design of the relocation scheme.

3.2. Problem definition

We define the problem of sensor relocation for emergent data acquisition tasks in the system model described in Section 3.1 as the following problem: relocating mobile sensors to proper locations so that a given emergent data acquisition task is fulfilled in a minimum period of time, with the constraint that a sensor (or the BS) can communicate with another sensor only if they are connected in the sparse MANET formed by the BS and the sensors.

Let us call the sensor that is assigned to sense the task location L as the Scanner. The time for fulfilling the task $ET(L)$ can be divided into three parts:

– T_{go}: from the time the task is issued to the moment the Scanner arrives at L;
– T_s: the time for sensing the task location;
– T_{return}: from the Scanner finishes sensing the task location to the moment the BS receives the sensed data.

The goal is to minimize $T_{go} + T_s + T_{return}$. Since T_s is fixed for a given task (T_s is determined by the application), we would like to minimize $T_{go} + T_{return}$. Both T_{go} and T_{return} can be significant in sparse MSNs because: the Scanner could be far away from L; after sensing the task location the Scanner may be disconnected from the BS so it has to move to find a connection to the BS. We shall reduce both T_{go} and T_{return}.

4. BRIDGE: A distributed relocation scheme

We propose a distributed relocation scheme called BRIDGE for sparse mobile sensors to carry out emergent tasks. The main ideas are as follows:

1. relocate the sensor that is the nearest to the task location to sense the task location so that it can arrive at the task location as early as possible. We call the sensor relocated to the task location as the *Scanner*. This is to reduce T_{go}.
2. relocate some sensors to help connect the Scanner to the BS so that the Scanner can send the acquired data to the BS without moving towards the BS. We refer to these sensors as the *Connectors*. This is to reduce the T_{return} component of the task execution time.
3. adjust the relocation with the sensors encountered during relocation, see whether they can be better Scanner or better Connectors. This is to exploit the encountered sensors to reduce T_{go} and T_{return}.

In BRIDGE, we store the information about the sensors involved in the relocation for an emergent task in a *relocation plan*. It basically tells which sensors are involved and what are their relocation destinations. Details about relocation plan is presented in Section 4.3.

In Fig. 1 we use the processing of an emergent task as an example to illustrate these ideas. In the figures, the rectangle marked with L is the task location, the circles marked with numbers are the sensors, the arrows on the sensors indicate the sensors' moving directions, the sensor in dark (e.g. s_3 in (b)) is the Scanner, the sensors in light gray (e.g. s_1, s_2 in (b)) are the Connectors, and the ones in white (e.g. s_4, s_5, s_6 in (b)) are not involved in the relocation. A line between two sensors means that they are connected.

Figure 1(a) depicts the scenario when the emergent task is issued but before the initial relocation plan is executed. Figure 1(b) illustrates the initial relocation plan where s_3 is assigned as the Scanner and s_1 and s_2 are assigned as the Connectors. s_3 is assigned as the Scanner because it is the nearest to the task location among the sensors that are connected to the BS. s_1 and s_2 are relocated to the locations between the task location and the BS so that they will help connect the Scanner with the BS. Figure 1(c) depicts the event where s_3 encounters s_4 and s_5. s_3 adjusts the initial relocation plan to a new relocation plan that is depicted in Fig. 1(d). In the new relocation plan, s_5 is assigned as the Scanner, s_1, s_2, s_3 and s_4 work as Connectors. s_5 is selected as the new Scanner because it is nearer to the task location than the existing Scanner (s_3) is. s_3 and s_4 also work as Connectors because the existing Connectors (s_1 and s_2) are not enough to connect the Scanner to the BS. In Fig. 1(e), the Scanner is sensing the task location and the Connectors connect the Scanner with the BS. After the Scanner finishes sensing the task location and sending the data to the BS, the emergent task is done. Then all the participating sensors can move freely to continue their basic exploration task, as shown in Fig. 1(f).

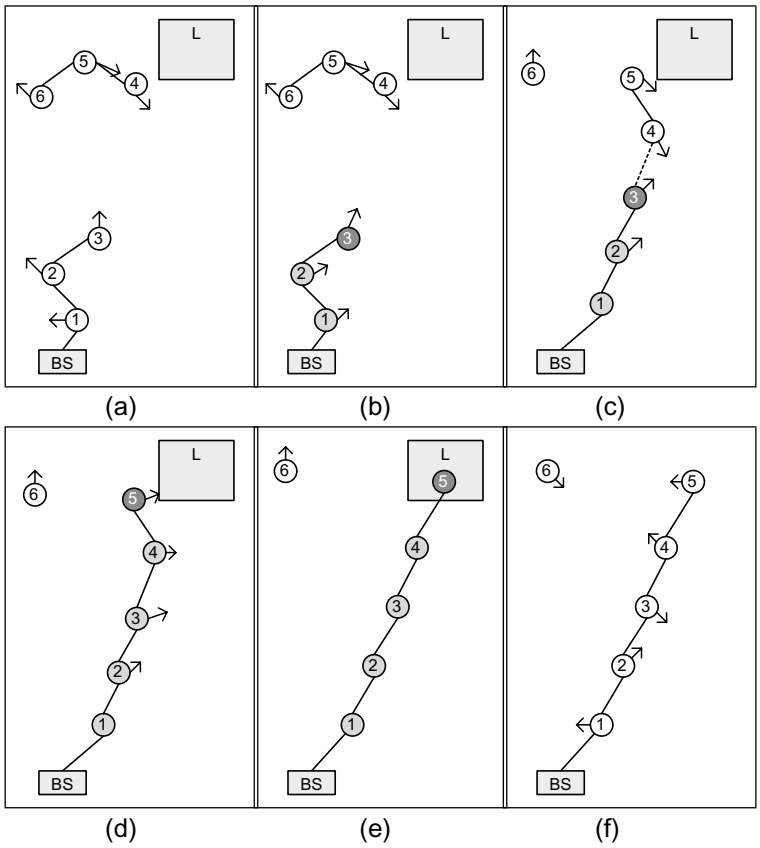

Fig. 1. A simple example illustrating the BRIDGE.

To realize the ideas of BRIDGE in a sparse MSNs, we have to deal with the problems caused by the intermittent connections.

First, when the BS has an emergent task, it is likely only few sensors are connected with it. BRIDGE must enable the BS to initiate the processing of the task by generating an initial relocation plan as long as at least one sensor is connected with the BS. This problem is solved in BRIDGE with an algorithm called `Init`.

Second, during the execution of a relocation plan, involved sensors may encounter sensors that were not connected. BRIDGE shall make use of the newly connected sensors to generate better relocation plans. This problem is solved in BRIDGE with an algorithm called `Adjust`.

Third, sensors involved in a same relocation plan can get disconnected during the relocation. They may encounter new neighbors. BRIDGE shall let them have enough information to generate better relocation plans even when being disconnected from other involved sensors. This problem is solved in BRIDGE by keeping enough information of the emergent task and involved sensors in a relocation plan and letting each involved sensor have a copy of the relocation plan.

Fourth, since involved sensors may generate different new relocation plans when they are disconnected, BRIDGE has to help them merge different relocation plans when they get connected again. This problem is solved in BRIDGE with an algorithm called `Merge`.

Fifth, after an emergent task is fulfilled, BRIDGE needs to make sure that all the sensors involved in the relocation finally know that the emergent task is done so that they can return to work on their general

task. This problem is solved in BRIDGE by letting the sensors keep the information about the emergent tasks (e.g. whether an emergent task is done) and synchronize the information when being connected.

Before presenting the algorithms, we will first describe the sensor's states (Section 4.1), the idea and definition of the *ConnectorPoints* (Section 4.2), what makes a relocation plan (Section 4.3), and the execution of a relocation plan (Section 4.4).

4.1. Sensor states

In BRIDGE, a mobile sensor is always in one of the following states: Free, Scanner, Connector, and Returner.

A sensor is in the Free state if it is not involved in the fulfillment of an emergent task. A Free sensor works on the general task of the application, e.g. explore the area.

A sensor is in the Scanner state if it is assigned to go to sense the task location. After being assigned as a Scanner, the sensor moves towards the task location.

A sensor is in the Connector state if its task is to help connect the Scanner with the BS. A Connector sensor will be given an ID (independent from its sensor ID) called the Connector-ID. A Connector sensor given a Connector-ID j will be called Connector-j. The Connector-ID tells the Connector where it shall move to. We will discuss this in Section 4.2.

A sensor is in the Returner state if it is carrying the acquired data of the emergent task. The Returner is responsible for sending the data to the BS. A Scanner becomes a Returner when it finishes sensing the task location.

The state of a sensor may change during the relocation because the sensors may adjust the relocation plan when they encounter new neighbors.

4.2. ConnectorPoints, Connector-ID

To connect the Scanner (at task location) and the BS with a minimum number of Connectors, we relocate the Connectors onto the line between the BS and the task location, and let them maintain a distance that is shorter than the sensors' communication range r from its neighboring Connectors. In this way, the number of Connectors, denoted as N_c, is minimized. N_c is computed as follows:

$$N_c = \lceil (Distance(BS, L)/r) \rceil - 1 \tag{1}$$

Here L is the task location, r is the sensors' communication range, $Distance(BS, L)$ means the distance between the BS and the task location.

These N_c Connector sensors can build a link on the line between the task location and the BS. For example, if the distance between the task location and the BS is 2400 meters and the sensors' communication range is 500 meters, we will need 4 sensors to work as Connectors. Figure 1 (e) shows the example.

We define the locations that the Connectors shall move to as the *ConnectorPoints* of the emergent task. Let $Line(BS, L)$ be the line segment between the BS and the task location L. There are N_c ConnectorPoints on the $Line(BS, L)$. We define $ConnectorPoint_j$ as the point on $Line(BS, L)$ whose distance to the BS is

$$j * Distance(BS, L)/(N_c + 1) \tag{2}$$

A Connector sensor given a Connector-ID j moves to the $ConnectorPoint_j$.

4.3. Relocation plan

A Relocation Plan of an emergent task $ET(L)$ is an assignment of a set of sensors to their roles in the processing of the task. The roles tell the sensors where they shall move to.

A relocation plan RP contains the following information:

- $ET(L)$: the emergent task, which includes the task location information.
- Sensor ID $\rightarrow$ Scanner: this specifies which sensor shall work as the Scanner. Later we will use RP.Scanner to refer to it.
- $\{$Sensor ID $\rightarrow$ Connector-$j\}$, $1 \leqslant j \leqslant N_c$: a map of sensor IDs to the Connector-IDs. This specifies which sensors shall be Connectors. We will use RP.Connector-j to refer to an entry in it. There could be fewer than N_c Connectors. If no sensor is assigned as Connector-j, RP.Connector-j is $null$.
- The time when the relocation plan is generated.
- The locations of the involved sensors (the Scanner and the Connectors) when the relocation plan is generated.

Note that in the relocation plan there is information about all the involved sensors, including their locations at the time the relocation plan is generated. By such, each involved sensor will have enough information to generate a better relocation plan when it encounters new neighbors as long as it has a copy of the current relocation plan. This design is important for the involved sensors to deal with the problems caused by intermittent connections.

4.4. Execution of a relocation plan

To execute a relocation plan, the sensor that computes the relocation plan simply disseminates the relocation plan to all the connected sensors. Upon receiving a relocation plan, a sensor checks whether it has a role in the plan. If yes, it saves a copy of the relocation plan and sets its state according to its role in the plan; otherwise it sets itself to the Free state. The Scanner moves towards the task location. The Connectors compute their ConnectorPoints based on their Connector-IDs, and then move to their ConnectorPoints.

If a new relocation plan is generated during the execution of a relocation plan (e.g. when new sensors are encountered), the new one will be executed in the connected sensors so that all the connected sensors follow the new plan.

4.5. Initiate a relocation for an emergent task

When the BS has a new emergent task, it uses the `Init` algorithm presented here to generate an initial relocation plan based on the Free sensors that are currently connected to it. `Init` assigns the Free sensor that is the nearest to the task location as the Scanner and assigns at most N_c Free sensors as the Connectors based on their vicinity to the ConnectorPoints.

The pseudocode of `Init` is listed in Algorithm 1. In the pseudocode, $Nearest(S, location)$ finds the sensor in set S that is the nearest to the $location$. $RP.Scanner$ denotes the information in the relocation plan while $Scanner$ denotes the corresponding sensor. This also applies to $RP.Connector_j$ and $Connector_j$. CP_j denotes $ConnectorPoint_j$.

Let $Frees$ be the set of Free sensors that are connected to the BS. If $Frees$ is empty, the BS has to wait until there is at least one sensor in Frees. The BS chooses the sensor that is the nearest to the task

Algorithm 1: `Init`

Input: an emergent task $ET(L)$
Output: a relocation plan RP for $ET(L)$
1 $Frees$ the Free sensors that are connected to BS;
2 $RP.Scanner$ $Nearest(Frees, L)$;
3 $Frees$ $Frees - \{Scanner\}$;
4 N_c $\lceil Distance(BS, L)/r \rceil - 1$;
5 m $Min(N_c, |Frees|)$;
6 **for** j 1 to m **do**
7 $RP.Connector_j$ $Nearest(Frees, CP_j)$;
8 $Frees$ $Frees - \{Connector_j\}$
9 **return** RP

location as the Scanner of the relocation plan (lines 2–3). Among the remaining Free sensors, the BS selects Connector sensors (lines 4–8) as follows. Let $m = Min(N_c, |Frees|)$. Here N_c is the number of Connectors that the relocation plan wants, and $|Frees|$ is the number of remaining Free sensors. m will be the number of Connectors in the initial relocation plan. For each $ConnectorPoint_j$ ($1 \leqslant j \leqslant m$), the BS finds the sensor in $Frees$ that is the nearest to it, and sets the sensor as the Connector-j of the relocation plan.

The time complexity of the `Init` algorithm is $O(n * N_c)$ where n is the number of Free sensors that are connected to the BS and N_c is the number of Connectors needed in the relocation plan. N_c typically is a small number. Since n will also be a small number in a sparse mobile sensor network, the initial plan of an emergent task can be computed efficiently.

Once the initial relocation plan for an emergent task is generated, it is executed among the connected sensors.

4.6. Adjust a relocation on meeting sensors

In BRIDGE, the participants of a relocation plan always try to find better relocation plans when they encounter new neighbors during the relocation. They do it using the `Adjust` algorithm listed in Algorithm 2. In this algorithm s_i denotes the involved sensor that encounters a set of new neighbors, and RP denotes the relocation plan that s_i currently has.

Algorithm 2: `Adjust`

Input: a relocation plan RP
Output: a new relocation plan
1 $Frees$ the Free sensors that are connected to s_i;
2 $Scanner'$ $Nearest(Frees, L)$;
3 **if** $Scanner'$ can reach L earlier than $RP.Scanner$ **then**
4 **if** $RP.Scanner$ is connected **then**
5 $Frees$ $Frees + \{Scanner\}$
6 $RP.Scanner$ $Scanner'$;
7 $Frees$ $Frees - \{Scanner'\}$;
8 RP `AdjustConnectors`$(RP, Frees)$;
9 **return** RP

On encountering new neighbors, s_i finds out all the Free sensors that it now can reach (line 1). It first checks whether it can find a better Scanner among the Free sensors (lines 2–7). If there is a better Scanner, s_i puts the current Scanner into the $Frees$ set if it is connected, and then updates the relocation plan with the new Scanner.

After that, s_i adjusts the Connectors with the remaining Free sensors using the algorithm `AdjustConnectors` which is listed in Algorithm 3. s_i divides the ConnectorPoints into two sets (lines 1–2): the ones for which no Connectors are assigned, and the ones with corresponding Connectors in current relocation plan. It first assigns Free sensors to the ConnectorPoints that have no Connectors (lines 3–6). Then it tries to find better Connectors for existing Connectors (lines 7–12).

Algorithm 3: `AdjustConnectors`

Input: a relocation plan RP
Input: a set of Free sensors $Frees$
Output: a new relocation plan
1 $EmptyCPs \quad \{j | j \leqslant N_c \wedge RP.Connector_j = null\}$;
2 $ExisingCPs \quad \{j | RP.Connector_j \neq null\}$;
3 **for** j in EmptyCPs **do**
4 **if** Frees is not empty **then**
5 $RP.Connector_j \quad Nearest(Frees, CP_j)$;
6 $Frees \quad Frees - \{Connector_j\}$;
7 **for** *j in ExisingCPs* **do**
8 **if** *Frees is not empty* **then**
9 $Connector'_j := Nearest(Frees, CP_j)$;
10 **if** $Connector'_j$ can reach CP_j earlier than $RP.Connector_j$ **then**
11 $RP.Connector_j \leftarrow Connector'_j$;
12 $Frees \leftarrow Frees - \{Connector_j\}$;
13 **return** RP

The complexity of both `Adjust` and `AdjustConnectors` is $O(n * N_c)$ where n is the number of Free sensors that are connected to s_i (the sensor that adjusts the relocation plan) and N_c is the number of Connectors needed in the relocation plan.

Note that when s_i encounters new neighbors, it is possible that some other participants of the relocation plan are disconnected from s_i. For example, when there are fewer than N_c Connectors, the Scanner will get disconnected from the Connectors when it moves to the task location. This kind of disconnection between the participants of a same relocation plan has two influences on the design of BRIDGE.

First, at the time a participating sensor meets new neighbors it may not be able to connect to other participating sensors and get up-to-date information from them. However, to find out whether the new neighbors can be helpful it needs information about current participants. For example, it needs the location information of the current Scanner to determine whether a new neighbor is nearer to the task location (as in line 3 of the `Adjust` algorithm). To resolve this problem, as described in Section 4.3, in the relocation plan we keep information about each participant's role and location at the time the relocation plan was generated. In this way, each participating sensor will have enough information to compute other participating sensors' locations using the following information: their starting locations, their relocation destinations (determined by roles), the elapsed time, and moving speed (all sensors have the same speed).

Second, disconnected participants will not receive the new relocation plans that are generated by other participants. In this case, some will have the old relocation plan while the others have a new one. Furthermore, partitioned groups of participants may independently adjust the relocation plan to new ones. Therefore, there can be more than one relocation plan for an emergent task being executed. In such cases, different relocation plans have to be merged when sensors having different relocation plans get connected again.

Table 1
Relocation Plans

Relocation Plan	Scanner	Connectors
RP1	s_2	s_1:1
RP2	s_4	s_1:1; s_2:2
RP3	s_2	s_1:1; s_3:2; s_5:3
RP4	s_4	s_1:1; s_3:2; s_2:3

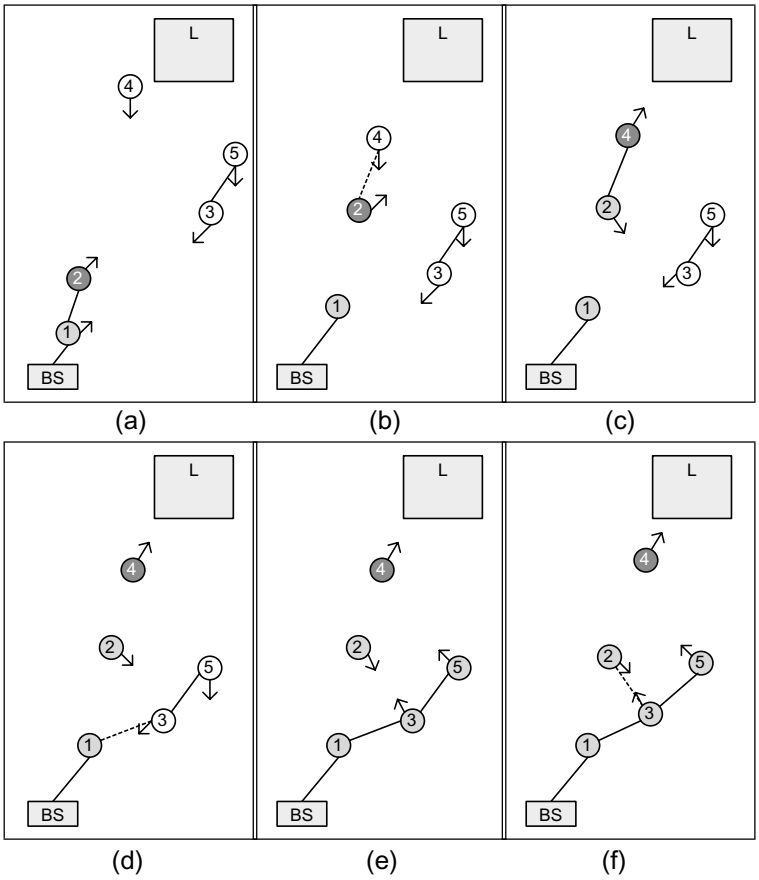

Fig. 2. Example for `Merge`.

4.7. Merge relocation plans

Figure 2 shows an example where participants of a relocation plan get disconnected, generate new relocation plans, and the participants having different relocation plans meet. The relocation plans are sketched in Table 1. In the table, "$s_i : j$" means that sensor s_i works as Connector-j.

– Figure 2(a) shows the scene after the initial relocation plan RP1 is executed. As the Scanner (s_2) moves towards the task location, it gets disconnected from s_1.
– When the Scanner (s_2) encounters s_4 (a Free sensor) as shown in Fig. 2(b), s_2 adjusts the initial relocation plan to a new relocation plan RP2.

– Figure 2(c) shows the scene after s_2 executes RP2. At this time s_2 and s_4 are disconnected from s_1 therefore they cannot update s_1 about the new relocation plan.
– As s_3 and s_5 (two Free sensor) move, they encounter s_1 as shown in Fig. 2(d). s_1 adjusts its relocation plan RP1 to a new relocation plan RP3.
– Figure 2(e) shows the scene after s_1 executes RP3. Note that when s_1 meets s_3 and s_5 it has the initial relocation plan RP1. So s_1 thinks that s_2 is still the best Scanner because it is not aware of s_4 and RP2.
– As the sensors move, s_2 encounters s_3. Figure 2(f) depicts this event. s_2 and s_3 have two different relocation plans. RP4 in Table 1 is the resultant relocation plan after s_2 and s_3 merge their relocation plans.

In BRIDGE, when two sensors having different relocation plans meet, one of them will merge the relocation plans to a new relocation plan and execute the new one among the connected sensors. The algorithm for merging two relocation plans is called `Merge`. Algorithm 4 lists its pseudocode.

Algorithm 4: `Merge`

```
    Input: two relocation plans RP1 and RP2
    Output: a relocation plan
1   if RP1.Scanner ≠ RP2.Scanner then
2       Scanners ← {RP1.Scanner, RP2.Scanner};
3       RP.Scanner ← Nearest(Scanners, R);
4       if the other Scanner is connected then
5           put it to Frees;
6   else
7       RP.Scanner = RP1.Scanner;
8   for Connector_j in RP1 and RP2 do
9       if Connector_j is connected then
10          Frees ← Frees + {Connector_j};
11  for For j ← 1 to N_c do
12      if RP1.Connector_j ≠ null || RP2.Connector_j ≠ null then
13          Connectors ← {RP1.Connector_j, RP2.Connector_j};
14          RP.Connector_j ← Nearest(Connectors, CP_j);
15  RP ← AdjustConnectors(RP, Frees);
16  return RP
```

Let s_i be the sensor that merges two relocation plans. In `Merge`, s_i first checks whether the two relocation plans have the same Scanner. If not, s_i chooses the one that can arrive at the task location earlier as the Scanner in the new plan, and put the other Scanner into the $Frees$ set if it is connected. Then for all the Connectors in the two relocation plans, if a Connector is connected s_i puts it into the $Frees$ set and clears the corresponding information in the relocation plan (lines 8–10). After this, all the available sensors will be in the $Frees$ set, and the two input relocation plans only contain information about Connectors that are currently disconnected. For each Connector slot in the new relocation plan, if the input relocation plans have disconnected Connector(s) for it, s_i picks the one that is nearer to the corresponding ConnectorPoint (lines 11–14). s_i finally uses the `AdjustConnectors` algorithm to: (1) assign Free sensors to empty Connector slots; (2) find better Connectors for disconnected existing Connectors.

The complexity of the `Merge` algorithm is $O(n * N_c)$ where n is the number of sensors that are connected to s_i (the sensor that merges the relocation plans) and N_c is the number of Connectors needed in the relocation plan.

4.8. Relocation after acquiring data

A Scanner becomes a Returner when it finishes sensing the task location. The Returner behaves according to the following rules.

- If the Returner is connected to the BS through the Connectors, it sends the data to the BS and the task is done. The Returner and all the connected Connectors are set to the Free state.
- Otherwise, if the Returner is connected to a Connector, it passes the data to that Connector and changes itself to the Free state. The Connector that receives the data becomes a Returner.
- Otherwise, the Returner moves towards the BS.

4.9. Handle obstacles

In the previous sections, we have implicitly assumed that there are no obstacles in the area. Here we discuss how BRIDGE handles obstacles.

Obstacles only affects how BRIDGE computes the ConnectorPoints between the BS and the task location L. If an obstacle is not on the line between BS and L, ConnectorPoints are computed on $Line(BS, L)$ as described in Section 4.2. If an obstacle is on $Line(BS, L)$, however, then we cannot relocate sensors onto $Line(BS, L)$ because the obstacle can be a region where the sensors cannot move into or a region where wireless communication is jammed by adversaries. Figure 3(a) shows an example where an obstacle intersects the line between BS and L.

The basic idea for handling an obstacle (that intersects $Line(BS, L)$) is to relocate the Connectors onto the shortest simple polyline between BS and L that does not intersect with the obstacle. Such a polyline is computed by first computing the convex hull using BS, L and the vertices of the obstacle, and then taking the shorter path from BS to L on the convex hull perimeter. Figure 3(b) shows the convex hull computed for the scenario shown in Fig. 3(a). The polyline BS-v_2-L is shorter than the other polyline between BS and L on the convex hull perimeter, so it is taken as the polyline on which the ConnectorPoints will be computed.

To make sure that two Connectors at consecutive ConnectorPoints can communicate with each other, a line of sight between them is necessary (in particular when the obstacle blocks the wireless communication). This requirement is satisfied as follows. BRIDGE first computes the ConnectorPoints on the selected polyline (shown in Fig. 3(c)). If the line between any two consecutive ConnectorPoints intersects the obstacle, the polyline is adjusted by adding a line segment that does not intersect the obstacle, and then the ConnectorPoints are re-computed on the adjusted polyline. Figures 3(d-f) show how the polyline is adjusted. In this example (see Figures 3(d)), the line between CP_2 and CP_3 intersects the obstacle. Because of this, a line segment near vertex v_2 is added and the polyline is adjusted by incorporating the new line segment (shown in Fig. 3(e)). The length of the new line segment is no longer than the communication range of the mobile sensors so that the two Connectors relocated on to the segment can communicate with each other. The ConnectorPoints are re-computed on the adjusted polyline. Figure 3(f) shows the final polyline and the ConnectorPoints on it.

Note that the obstacles only affect the locations of the ConnectorPoints. The polyline and the ConnectorPoints are computed (on BS) in the initial relocation plan, and they are fixed during the relocation. Therefore only the `Init` algorithm needs to be slightly modified to handle the obstacles in the field.

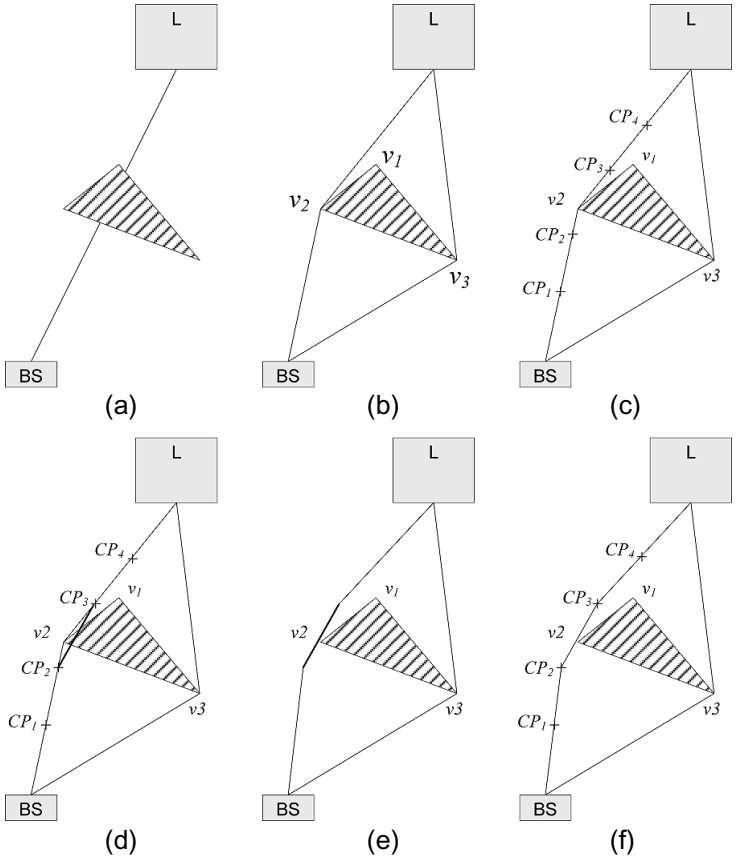

Fig. 3. Example for handling an obstacle.

5. Experimental study

We use simulation to study the performance of our proposed BRIDGE scheme. Since this is the first work on relocation of mobile sensors for emergent data acquisition tasks, there is no existing solution to compare with. We compare the proposed scheme to two variants: a simpler version (Simple-BRIDGE) and a motion-aware version (Motion-Aware-BRIDGE).

In the Simple-BRIDGE scheme, we set the number of Connectors (N_c) to 0. By this, a relocation plan only contains a Scanner and no sensors will be relocated to help connect the Scanner with the BS. The goal of comparing BRIDGE with this simpler version is to study the effect of relocating some sensors to connect the Scanner to the BS.

The Motion-Aware-BRIDGE solution is an ideal scheme with perfect information. We assume that the Scanner know all other sensors' planned trajectories. Rather than moving directly to the task location, the Scanner in this version moves to meet a currently disconnected sensor that can reach the task location earlier. The Scanner finds such a sensor by computing the sensor that minimizes the expression ($T_{meet} + T_{move}$) where T_{meet} is the minimal time the Scanner needs to move to meet that sensor and T_{move} is the time that sensor needs to move to the task location. The goal of comparing BRIDGE with this version is to see how much the BRIDGE algorithm will improve if all sensors' future trajectories are given.

Table 2
Parameter Settings

Parameter	Default value	Value range
$FieldWidth$	30 km	10–50 km
$FieldHeight$	30 km	
Number of Sensors n	20	5–50
Sensor speed v	0.05 km/s	0.02–0.2 km/s
Communication range r	5 km	1–10 km
T_s	40 (s)	20–100 (s)

The simulation model follows the System Model that we describe in Section 3. The system parameters and their values are listed in Table 2. The parameter values are chosen based on the setting used in [10]. This simulates a system where a set of UAVs are carrying out a reconnaissance task and the base station sometimes issues emergent data acquisition tasks to the UAVs.

In the experiments the mobile sensors are placed randomly in the simulated area and they follow the Random WayPoint mobility model. Emergent tasks are also placed randomly in the simulated area. When we generate an emergent task we always generate a new scenario (placement of the sensors) so that all the solutions have the same start point.

Because obstacles only affect the number of sensors needed to connect BS and a task location, its effect is the same as shortening the communication range of the mobile sensors. For this reason, we do not generate obstacles in the experiments.

The performance metric is the average completion time of a large number of emergent data acquisition tasks.

In the figures that show the experimental results, "Simple", "BRIDGE", and "MA" ("S", "G", "M" in some bar figures) denote the Simple-BRIDGE, the BRIDGE, and the Motion-Aware-BRIDGE schemes respectively. In the figures that show the breakdown of task completion times, the number under each group of three bars denotes the value of the parameter under investigation. Each bar has three parts: Go, Scan, and Return. They correspond to the T_{go}, T_s, and T_{return} times defined in Section 3.

5.1. Basic performance study

Here we study the solutions' performance under the default parameters setting. Figure 4 shows the breakdowns of the solutions' average processing times. We see that BRIDGE and MA have similar performance and they perform much better than Simple. In particular, BRIDGE and MA spend much less time in the Return phase. MA performs a little better than BRIDGE because MA spends a little less time in the Go phase than BRIDGE does. However, the performance difference between BRIDGE and MA is minor.

We learn two things from this basic performance study. 1) The idea of relocating some sensors to connect the Scanner with the base station is very effective in reducing task completion time. 2) A more complex solution (MA) that makes use of sensors' motion information does not improve BRIDGE much. They show the merits of the BRIDGE scheme: it is effective and widely applicable (since it does not make any assumption of the sensors' mobility pattern).

5.2. Effect of sensors density

We use the average number of sensors that are within each sensor's communication range as the measurement of the sensor density. Three parameters affect the sensor density: the number of sensors n,

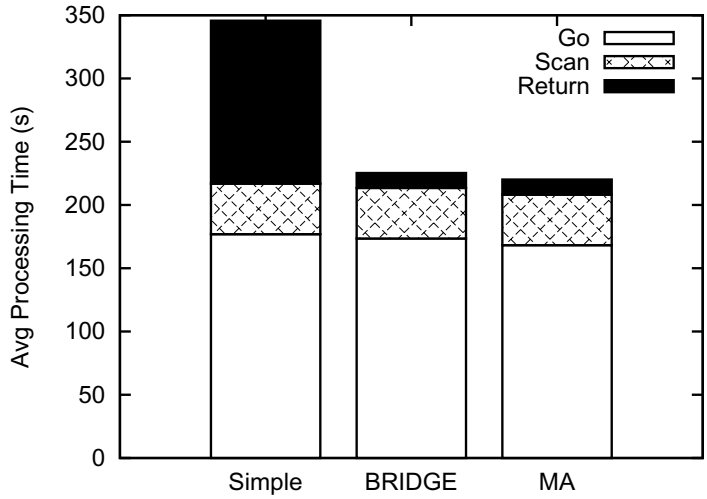

Fig. 4. Default setting.

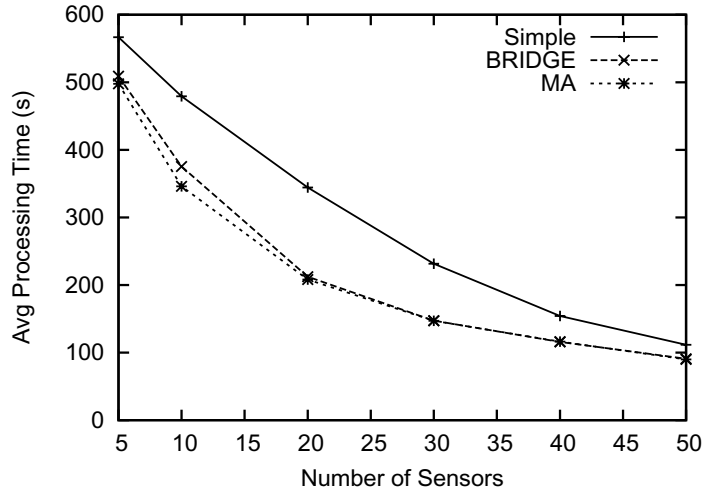

Fig. 5. Effect of n.

$FieldWidth$ that controls the field size, and the sensors' communication range r. As n or r increases, the sensor density increases; as $FieldWidth$ increases, the sensor density decreases. Figures 5, 6, and 8 show the effect of the three parameters on the solutions' performance. We have several observations here.

1) As the sensor density increases, all schemes perform better. This is because when sensor density increases, more sensors are connected, then it becomes easier to find a sensor that is near to the task location, and it is more likely that the acquired data can be sent back to the BS directly.

2) BRIDGE's performance is very close to MA's. Only in very sparse networks, MA performs a little better than BRIDGE does. In very sparse networks, the chance of encountering new neighbors during the relocation is small. In this circumstance, having the motion information of other sensors (in MA) helps the Scanner to meet more sensors.

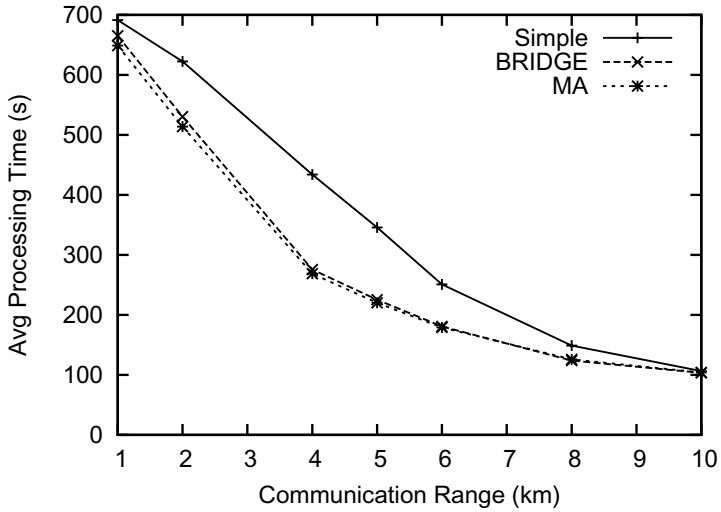

Fig. 6. Effect of r.

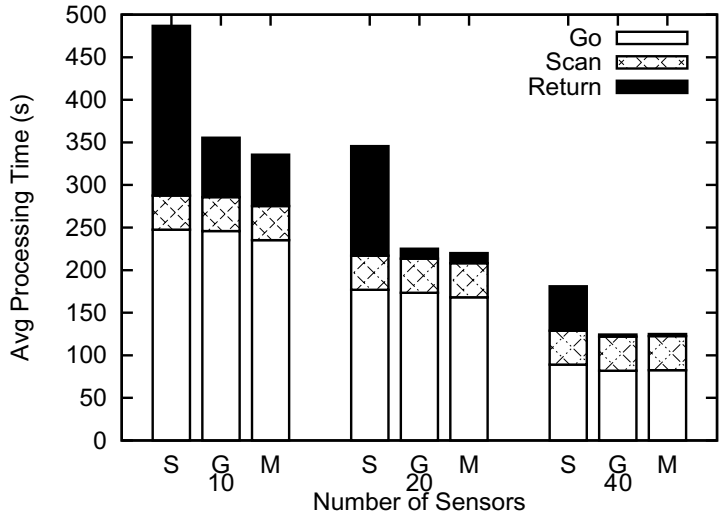

Fig. 7. Breakdowns, n.

3) When the number of sensors or the communication range increases from small to large (sensor density increases from very sparse to very dense) in Figs 5 and 6, the performance gap between Simple and BRIDGE first increases and then decreases. This is explained by Fig. 7 which shows the breakdowns of the task completion times when the number of sensors is 10, 20 and 40. Please notice the difference between Simple and BRIDGE's Return times first increases and then decreases. The gap increases first because as sensor density increases from very sparse to medium, BRIDGE can find more sensors to work as Connectors, and this reduces the time of the Return phase greatly. Then, when the sensor density increases from medium to dense, it is more likely that the Scanner at the task location is connected to the BS so that it is less necessary to relocate some sensors to work as Connectors.

4) When the field size increases (in Fig. 8), the gap between the Simple and BRIDGE always increases.

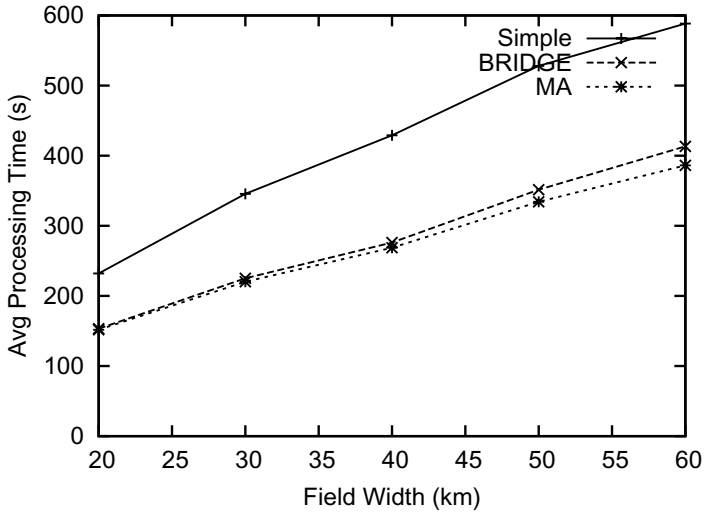

Fig. 8. Effect of field size.

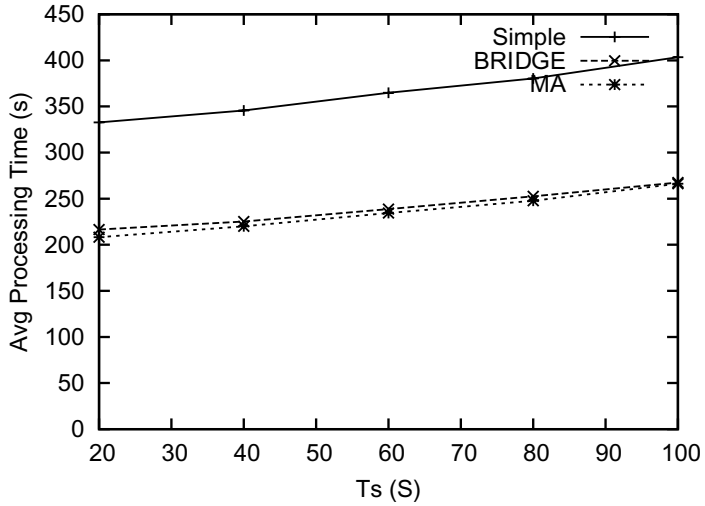

Fig. 9. Effect of T_s.

This is expected as the increase in field size not only makes the sensor network sparser, but also makes the task location farther from the base station (in the experiments the emergent task locations are randomly generated in the area). When an emergent task is farther from the BS, it is more important to have some sensors working as Connectors.

5.3. Other sensitivity study

5.3.1. Effect of T_s

Figure 9 shows the effect of T_s on the schemes' performance. Figure 10 shows the breakdowns of processing times when T_s is 40 and 80 (s). From the figures, we see that the task completion times increase as T_s increases.

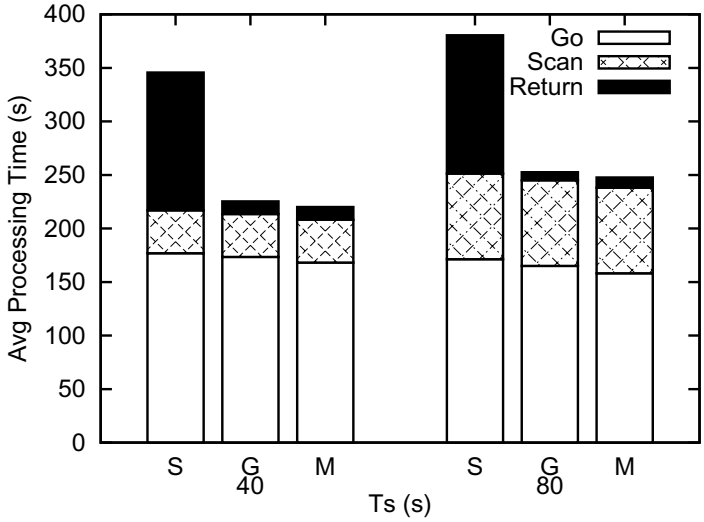

Fig. 10. Breakdowns, T_s.

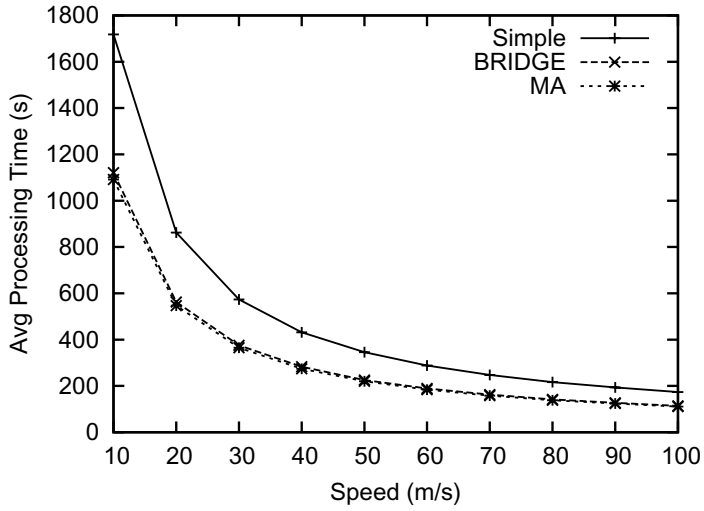

Fig. 11. Effect of move speed.

5.3.2. Effect of sensor speed

The sensors' move speed affect the time for a sensor to relocate to a location and the dynamism of the sensor network's topology. When sensors move faster, it takes shorter time for a sensor to meet new neighbors. Figures 11 and 12 show the effect of sensors' move speed on the solutions' performance. It is clear, and as expected, that as sensors' move speed increases, the processing times.

6. Conclusion and future work

In this paper, we have identified the problem of sensor relocation for emergent data acquisition in sparse mobile sensor networks. Mobile sensors are tasked to go to a given location to gather information

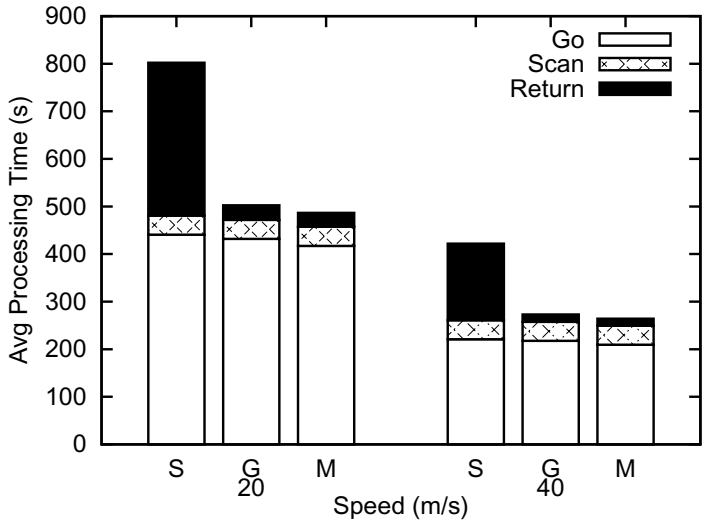

Fig. 12. Breakdowns, v.

and to return the sensed data to the base station as early as possible. We proposed a BRIDGE scheme in which the sensors collaborate to fulfill an emergent task by gradually finding a sensor that is near the task location and relocating some sensors to connect the base station to the sensor that is responsible for sensing the task location. Our extensive performance evaluation showed the effectiveness of the proposed solution.

There are a number of directions in which we would like to expand this research. We are interested, for example, in how the BRIDGE scheme can be extended to handle multiple emergent data acquisition tasks, and how to adapt BRIDGE to deal with obstacles that are not known in advance. For multiple emergent data acquisition tasks, the challenge is to compute a relocation plan so that the average completion time of several concurrent emergent data acquisition tasks is minimized. In scenarios where not all the obstacles are known in advance, the requirement for the relocation is dynamic and the relocation plan may have to be adjusted dramatically during relocation. For example, not only the involved sensors, but also the locations of the ConnectorPoints will be different after encountering an un-anticipated obstacle. The challenge here is to coordinate the mobile sensors (whose connections are intermittent) so that they will be informed when a relocation plan has been changed significantly.

Acknowledgements

This project is partially supported by a research grant R-252-000-352-232 (TDSI/08-001/1A) from the Temasek Defense Systems Institute.

References

[1] J. Allred, A.B. Hasan, S. Panichsakul, W. Pisano, P. Gray, J. Huang, R. Han, D. Lawrence and K. Mohseni, Sensorflock: an airborne wireless sensor network of micro-air vehicles, in: *SenSys*, New York, NY, USA, ACM, 2007, pp. 117–129.

[2] J. Bellingham, M. Tillerson, A. Richards and J.P. How, Multi-task allocation and path planning for cooperating uavs, in: *Conference on Cooperative Control and Optimization*, 2001.

[3] J.Y. Chen, G. Pandurangan and D. Xu, Robust computation of aggregates in wireless sensor networks: Distributed randomized algorithms and analysis, *Transactions on Parallel and Distributed Systems* **17**(9) (2006), 987–1000.
[4] L.M. Feeney and M. Nilsson, Investigating the energy consumption of a wireless network interface in an ad hoc networking environment, in: *INFOCOM*, 2001.
[5] M. Garetto, M. Gribaudo, C.F. Chiasserini and E. Leonardi, A distributed sensor relocatlon scheme for environmental control, in: *MASS*, 8–11 Oct. 2007, pages 1–10.
[6] Y.Y. Gu, D. Bozdağ, R.W. Brewer and E. Ekici, Data harvesting with mobile elements in wireless sensor networks, *Comput Netw* **50**(17) (2006), 3449–3465.
[7] H. Gupta, Z.H. Zhou, S.R. Das and Q. Gu, Connected sensor cover: self-organization of sensor networks for efficient query execution, *IEEE/ACM Transactions on Networking* **14**(1) (Feb 2006), 55–67.
[8] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan and S. Madden, Cartel: a distributed mobile sensor computing system, in: *SenSys*, New York, NY, USA, ACM, 2006, pages 125–138.
[9] Y. Jin, A.A. Minai and M.M. Polycarpou, Cooperative real-time search and task allocation in uav teams, in: *IEEE Conference on Decision and Control*, (Vol. 1), 9–12 Dec. 2003, pp. 7–12.
[10] E. Kuiper and S. Nadjm-Tehrani, Mobility models for uav group reconnaissance applications, in: *ICWMC*, 29–31 July 2006, pages 33–33.
[11] X. Li, H. Frey, N. Santoro and I. Stojmenovic, Focused-coverage by mobile sensor networks, in: *Proc IEEE 6th International Conference on Mobile Adhoc and Sensor Systems MASS '09*, October 12–15, 2009, pp. 466–475.
[12] X. Li and N. Santoro, An integrated self-deployment and coverage maintenance scheme for mobile sensor networks, in: *Mobile Ad-hoc and Sensor Networks, Second International Conference*, 2006, pp. 847–860.
[13] B.Y. Liu, P. Brass, O. Dousse, P. Nain and D. Towsley, Mobility improves coverage of sensor networks, in: *MobiHoc*, New York, NY, USA, 2005. ACM, pp. 300–308.
[14] W. Liu, J.P. Wang, G.L. Xing and L.S. Huang, Throughput capacity of mobility-assisted data collection in wireless sensor networks, in: *Proc IEEE 6th International Conference on Mobile Adhoc and Sensor Systems MASS '09*, October 12–15, 2009, pp. 70–79.
[15] J. Luo, D. Wang and Q. Zhang, Double mobility: Coverage of the sea surface with mobile sensor networks, in: *Proc INFOCOM 2009. The 28th Conference on Computer Communications. IEEE*, April 19–25, 2009, pp. 118–126.
[16] S. Madden, M.J. Franklin, J.M. Hellerstein and W. Hong, Tinydb: An acquisitional query processing system for sensor networks, *ACM TODS* **30**(1) (November 2005).
[17] E.D. Margerie, Jean baptiste Mouret, Stphane Doncieux, Jean arcady Meyer, Thomas Ravasi, Pascal Martinelli, and Christophe Gr. Flapping-wing flight in bird-sized uavs for the robur project: from an evolutionary optimization to a real flapping-wing mechanism, in: *3rd US-European Competition and Workshop on Micro Air Vehicle Systems* (*MAV07*), 2007.
[18] R.C. Shah, S. Roy, S. Jain and W. Brunette, Data mules: modeling a three-tier architecture for sparse sensor networks, in: *IEEE SNPA Workshop*, pages 30–41, 11 May 2003.
[19] G.T. Sibley, M.H. Rahimi and G.S. Sukhatme, Robomote: a tiny mobile robot platform for large-scale ad-hoc sensor networks, in: *Proc. IEEE International Conference on Robotics and Automation ICRA '02*, (Vol. 2), 11–15 May 2002, pp. 1143–1148.
[20] W. Wang, V. Srinivasan and K.-C. Chua, Trade-offs between mobility and density for coverage in wireless sensor networks, in: *MobiCom*, New York, NY, USA, 2007. ACM, pp. 39–50.
[21] S. Sukkarieh, E. Nettleton, J.H. Kim, A. Goktogan and H. Durrant-Whyte, The anser project: data fusion across multiple uninhabited air vehicles, *The International Journal of Robotics Research* **22**(7–8) (2003), 505–539.
[22] G. Trajcevski, P. Scheuermann and H. Brönnimann, Mission-critical management of mobile sensors: or, how to guide a flock of sensors, in: *DMSN*, New York, NY, USA, 2004. ACM, pp. 111–118.
[23] G. Wang, G. Cao, T.L. Porta and W. Zhang, Sensor relocation in mobile sensor networks, in: *INFOCOM*, (Vol. 4), 13–17 March 2005, pp. 2302–2312.
[24] G.L. Xing, T. Wang, W.J. Jia and M.M. Li, Rendezvous design algorithms for wireless sensor networks with a mobile base station, in: *MobiHoc '08: Proceedings of the 9th ACM international symposium on Mobile ad hoc networking and computing*, New York, NY, USA, 2008. ACM, pp. 231–240.
[25] N. Santoro, X. Li and I. Stojmenovic, Mesh-based sensor relocation for coverage maintenance in mobile sensor networks, in: *Ubiquitous Intelligence and Computing*, 2007.
[26] O. Younis and S. Fahmy, Heed: A hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks, *IEEE Transactions On Mobile Computing* **3**(4) (December 2004), 366–379.
[27] D. Zeinalipour-Yazti, P. Andreou, P.K. Chrysanthis and G. Samaras, Senseswarm: a perimeter-based data acquisition framework for mobile sensor networks, in: *DMSN*, New York, NY, USA, 2007. ACM, pp. 13–18.
[28] W. Zhao, M. Ammar and E. Zegura, A message ferrying approach for data delivery in sparse mobile ad hoc networks, in: *MobiHoc*, New York, NY, USA, 2004. ACM, pp. 187–198.

Wei Wu is a research fellow at the School of Computing, National University of Singapore (NUS). He completed his PhD in computer science in 2009 at Singapore-MIT Alliance, NUS, and received his Bachelor and Master from Nanjing University in 2002 and 2005 respectively. His research interests include data management, mobile computing, and sensor networks.

Xiaohui Li is currently a PHD candidate in School of Computing (SoC), National University of Singapore (NUS). His current research interest is moving objects data management. He received a B.Comp (Hons.) and a B.S. from NUS. Prior to joining SoC, he worked as IT Analyst in Center for Life Science (CeLS).

Shili Xiang is a Ph.D. student in Computer Science department at National University of Singapore, under supervision of Professor Kian-Lee Tan. Her research interests include query processing in sensor networks, streaming processing and pervasive computing.

Hock Beng Lim is program director of the Intelligent Systems Center at Nanyang Technological University, Singapore. He received his BS in Computer Engineering, MS in Electrical Engineering, and PhD in Electrical and Computer Engineering from the University of Illinois at Urbana-Champaign, and his MS in Management Science and Engineering from Stanford University. His research interests include sensor networks and sensor grids, cyber-physical systems, cloud computing, parallel and distributed computing, wireless and mobile networks, computer architecture, embedded systems, performance evaluation, e-Science and high-performance computing.

Kian-Lee Tan is a Professor of Computer Science at the School of Computing, National University of Singapore (NUS). He received his Ph.D. in computer science in 1994 from NUS. His current research interests include multimedia information retrieval, query processing and optimization in multiprocessor and distributed systems, database performance, and database security. He has published numerous papers in conferences such as SIGMOD, VLDB, ICDE and EDBT, and journals such as TODS, TKDE, and VLDBJ. Kian-Lee is a member of ACM.