Energy-Balanced Dispatch of Mobile Sensors in a Hybrid Wireless Sensor Network

You-Chiun Wang, Wen-Chih Peng, and Yu-Chee Tseng

Abstract—We consider a hybrid wireless sensor network with static and mobile nodes. Static sensors monitor the environment and report events occurring in the sensing field. Mobile sensors are then dispatched to visit these event locations to conduct more advanced analysis. A big challenge is how to schedule these mobile sensors' traveling paths in an energy-balanced way so that their overall lifetime is maximized. We formulate this problem as a *multi-round sensor dispatch problem* and show it to be NP-complete. Then, we propose a centralized and a distributed heuristics to schedule mobile sensors' traveling paths. Our heuristics allow arbitrary numbers of mobile sensors and event locations in each round and have an energy-balanced concept in mind. The centralized heuristic tries to minimize mobile sensors' moving energy while keeping their energy consumption balanced. The distributed heuristic utilizes a grid structure for event locations to bid for mobile sensors. Through simulations, we show the effectiveness of our schemes. This paper contributes in defining a more general multi-round sensor dispatch problem and proposing energy-efficient solutions to it.

Index Terms—energy saving, load balance, mobile sensor, robot, wireless sensor network.

1 Introduction

HYBRID sensor networks with static and mobile nodes open a new frontier of research in *wireless sensor networks (WSNs)*. Static sensors support environmental sensing and network communication. Mobile sensors are more resource-rich in sensing and computing capabilities and can move to particular locations to conduct more complicated missions such as repairing the network or providing in-depth analysis [1], [2]. Introducing mobility to a WSN not only reduces its deployment and maintenance costs but also enhances its capability. Applications of hybrid WSNs have been studied in [3]–[5].

In this paper, we focus on the problem of dispatching mobile sensors to the locations of events appearing in the sensing field. Static sensors serve as the backbone to identify where suspicious events may appear and report such events to mobile sensors so as to conduct more in-depth analysis. Assuming that events may appear anytime and anywhere, it is inefficient to dispatch one mobile sensor right after an event appears. We thus propose dividing time into multiple rounds and schedule mobile sensors' traveling paths in a round-by-round manner. The objective is to emphasize both path efficiency and balance of mobile sensors' energy consumption, because moving energy is critical for mobile sensors [6], [7]. Then, we measure the *system lifetime*, which is defined as the number of rounds until some event locations cannot be reached by any mobile sensor due to lack of energy.

Balancing energy consumption is important in the case of multiple mobile sensors. For example, when some mobile sensors exhaust their energy too early, the remaining mobile sensors may need to travel longer distances to serve some event locations, thus further shortening the system lifetime. Contrarily, if there are more mobile sensors, each mobile sensor may need to visit only local event locations. Reference [8] is such an example which targets at greedily minimizing the overall energy consumption of mobile sensors in each round without considering energy balance among mobile sensors. Take an example in Fig. 1. Initially, mobile sensors s_1 and s_2 are located at l_1 and l_2, respectively, each with energy of 600 units. The energy cost to move between any two locations is given in Fig. 1(a). Suppose that in each odd round, events appear at l_3 and l_4, and in each even round, events appear at l_1 and l_2. Fig. 1(b) shows that with the above greedy strategy, s_1 and s_2 move between (l_3, l_1) and (l_4, l_2), respectively. The system lifetime is 9 rounds. Fig. 1(c) shows that by balancing their energy consumption, s_1 and s_2 move between (l_4, l_1) and (l_3, l_2), respectively, and can survive 10 rounds, although in each round they consume more energy.

In this paper, we prove that even if all event locations in the future rounds are known in advance, the sensor dispatch problem is NP-complete. We then propose a centralized and a distributed heuristics to extend the system lifetime. The idea is to minimize the moving energy of mobile sensors while balancing their energy consumption in each round. Our centralized heuristic allows arbitrary numbers of event locations and mobile sensors in each round. When mobile sensors are more than event locations, we translate the dispatch problem to a maximum-matching problem in a weighted complete bipartite graph [9], where the vertex set contains mobile sensors and event locations and the edge set contains each edge from every mobile sensor to every event location. In addition, when matching two vertices, we adopt a *bound* concept to avoid choosing edges with too large weights, so the energy-balanced goal can be achieved. On the other hand, when mobile sensors are fewer than event locations, we group event locations into clusters where the number of clusters is equal to that of mobile sensors. Then, we adopt the above matching approach combined with the *traveling-salesman approximation algorithm (TSP)* [10]. Designed with a similar philosophy, we also propose a distributed heuristic using a grid structure. Specifically, each grid with event locations adopt grid-quorum [11] to obtain the information of mobile sensors. Then, the grids bid for mobile sensors with each other by using the bound concept in

The authors are with the Department of Computer Science, National Chiao-Tung University, Hsin-Chu, 30010, Taiwan.
E-mail: {wangyc, wcpeng, yctseng}@cs.nctu.edu.tw

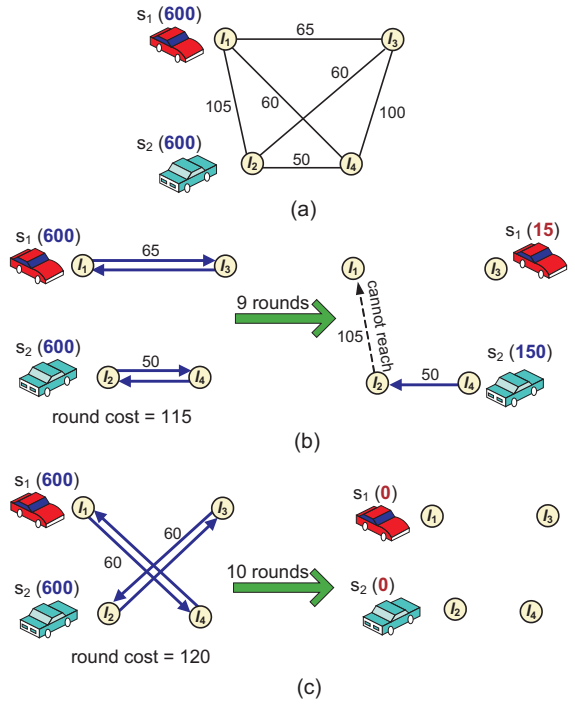


Fig. 1: Comparison of dispatch solutions: (a) the network configuration, (b) the greedy approach, and (c) the energy-balanced approach.

the centralized heuristic. After determining the winning grids, each mobile sensor visits these grids by the proposed *two-level TSP scheme*. In particular, the mobile sensor first calculates the shortest path to visit the winning grids, and then moves to the event locations inside each of these grids.

Although sharing the same bound concept, the centralized and distributed heuristics have two differences in essence. First, the centralized heuristic clusters event locations, from a global view, when there are fewer mobile sensors, while the distributed heuristic always clusters event locations into fixed grids. Second, the centralized heuristic requires a central node (e.g., the sink) to calculate the dispatch schedules, which incurs network flooding to gather/disseminate global information. In contrast, the distributed heuristic adopts grid-quorum to reduce the message complexity but lets event locations compete for mobile sensors using partial information. Our simulation results reflect that when there are more mobile sensors, the grid structure helps extend the system lifetime. On the other hand, when there are fewer mobile sensors, the centralized heuristic has a longer system lifetime due to its efficient clustering and global knowledge. Nevertheless, both heuristics result in a longer system lifetime compared to the greedy scheme. Also, simulation results show that the distributed heuristic is more message-efficient.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 defines the sensor dispatch problem and shows it to be NP-complete. Sections 4 and 5 propose our centralized and distributed heuristics. Simulation results are presented in Section 6. Conclusions and future research topics are drawn in Section 7.

2 Related Work

Mobility management has received considerable attention in *mobile ad hoc networks (MANETs)*. Most studies focus on the communication issue [12], [13] or topology control [14], [15] due to frequent node mobility. They consider that nodes move in an arbitrary manner or follow some mobility models [16]. Unlike MANETs, node mobility in hybrid WSNs is controllable and can even be coordinated. A *Multi-robot system (MRS)*, one topic in the field of robots, uses multiple cooperative robots to accomplish a task in an uncertain environment [17], [18]. *Multi-agent reinforcement learning* [19]–[21] is proposed to train robots to learn mappings from their statuses to their actions. However, these studies have different objectives from our work. *Multi-robot task allocation* [22] determines which robot should execute which task to cooperatively achieve the overall goal, where a *task* is viewed as an independent subgoal that is necessary for achieving the overall goal. However, [22] does not aim at the energy issue of robots.

Mobile sensors have been intensively researched to improve a WSN's topology. Moving sensors to approximate the event distribution, while maintaining complete coverage of the sensing field, is studied in [23]. The work in [24] moves nodes to keep a WSN biconnected. With a grid structure, the work in [25] moves sensors from high-density grids to low-density ones to generate a uniform topology. References [26], [27] use *virtual forces* to drive sensors' moving directions, while the work in [28] discusses moving sensors to fill uncovered holes. The studies [29], [30] also address the sensor dispatch problem, but they do not consider energy balance and only optimize energy consumption in one round.

Some studies deploy mobile sensors to track moving targets. The *purser-evader game* is studied in [31], where a pursuer needs to intercept an evader by the assistance of static sensors. The work in [32] discusses the problem of maneuvering mobile sensors for the optimal data acquisition from moving targets. Target tracking with the assistance of mobile sensors is discussed in [33] with concerns of energy consumption, network connectivity, and sensing coverage. They assume that the future trajectory of the moving target may be predicted, whereas our work allows events to arbitrarily appear.

Several variations of the sensor dispatch problem have been studied in the literature. In [34], static sensors detecting events will ask mobile sensors to move to their locations to conduct more in-depth analysis. The mobile sensor that has a shorter moving distance and more energy, and whose leaving will generate a smaller uncovered hole, is invited. The work in [35] dispatches mobile sensors to improve the sensing coverage of a hybrid WSN. Static sensors estimate the uncovered holes close to them and use the hole sizes to compete for mobile sensors. The concept of energy balance in dispatching mobile sensors is exploited in [36]. Once a mobile sensor s_i identifies a destination l_j, s_i tries to form a sequence of $s_i \rightarrow s_{k_1} \rightarrow s_{k_2} \rightarrow \cdots \rightarrow s_{k_m} \rightarrow l_j$, such that cascaded movements, $s_i \rightarrow s_{k_1}$, $s_{k_1} \rightarrow s_{k_2}$, $\cdots$, and $s_{k_m} \rightarrow l_j$ will happen. How event locations can find mobile sensors in a message-efficient manner is discussed in [37]. Compared with priori studies, this paper considers a more general dispatch problem, where events may appear in arbitrary locations, event locations in the future rounds are unpredictable, and the relationship between the number of event locations and the number of mobile sensors is arbitrary. We will develop centralized and distributed algorithms with energy-balanced property to prolong the system lifetime.

3 Problem Statement

We consider a hybrid WSN with both static and mobile sensors. Sensors are aware of their own locations, which can be achieved by GPS (global positioning system) or other localization techniques [38]. Static sensors are dense enough to form a connected network that fully covers the sensing field. They can cooperate to identify events that may appear in arbitrary locations in the sensing field (refer to [34] for possible solutions). We make no assumption on the event distribution, and the occurrence of any two events is independent. Mobile sensors are more resource-rich and can be dispatched to event locations to conduct more in-depth analysis. Both the moving speed of a mobile sensor and its energy consumption to move a unit distance are assumed to be constants. Also, we assume that the sensing field is obstacle-free, so mobile sensors can directly move to their destinations using the shortest distances.

The time is divided into *rounds*. Each round is led by a *collecting phase* followed by a *dispatching phase*. Static sensors report those events that have been detected but not yet processed in the collecting phase. Mobile sensors then visit these event locations in the dispatching phase. In each round, an event only needs to be visited by one mobile sensor. Our discussion focuses on the dispatch problem in one round. Therefore, given a set of m event locations $\mathcal{L} = \{l_1, l_2, \cdots, l_m\}$ and a set of n mobile sensors $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ in a round, our objective is to assign each s_i a *dispatch schedule* DS_i, $i = 1..n$, which contains a sequence of event locations. The union of locations in all dispatch schedules should be equal to $\mathcal{L}$. In case that DS_i is too long for s_i to complete in the current round, the remaining unvisited locations are deleted from DS_i and will be put into the next round for scheduling (thus these deleted locations may be visited by other mobile sensors). The jth location of DS_i is denoted by $DS_i[j]$, and the current energy of s_i is expressed by e_i. Consequently, the energy required to complete s_i's dispatch schedule is formulated as $f(DS_i) = e_{\text{move}} \times \left(d(s_i, DS_i[1]) + \sum_{j=1}^{|DS_i|-1} d(DS_i[j], DS_i[j+1])\right)$, where e_{move} is the energy cost for a mobile sensor to move one unit distance, $|DS_i|$ is the number of event locations in DS_i, and $d(\cdot,\cdot)$ is the distance between two locations. Clearly, any dispatch schedule of a mobile sensor should satisfy $e_i \geq f(DS_i)$. Also, we are given the initial energy e_i^{init} of each s_i, $i = 1..n$, in round zero. Assuming that mobile sensors are not rechargeable, the objective is to schedule their traveling paths such that the system lifetime is maximized. Table 1 summarizes the notations used in this paper.

Next, we show that the above sensor dispatch problem is NP-complete even if the event locations are known in advance in each round. We first formulate the sensor dispatch problem as a decision problem:

Definition 1. Given a set of mobile sensors $\mathcal{S}$ and a sequence of k event location sets $\mathcal{L}_1, \mathcal{L}_2, \cdots, \mathcal{L}_k$, the sensor dispatch decision problem is to determine whether there is a feasible schedule for $\mathcal{S}$ to visit $\mathcal{L}_1, \mathcal{L}_2, \cdots, \mathcal{L}_k$ in that order.

Theorem 1. The sensor dispatch decision problem is NP-complete.

Proof: We first show that this problem belongs to NP. Given a problem instance and a solution containing the dispatch schedules in k rounds, it can be verified whether the solution is valid in polynomial time. So, this part is proved.

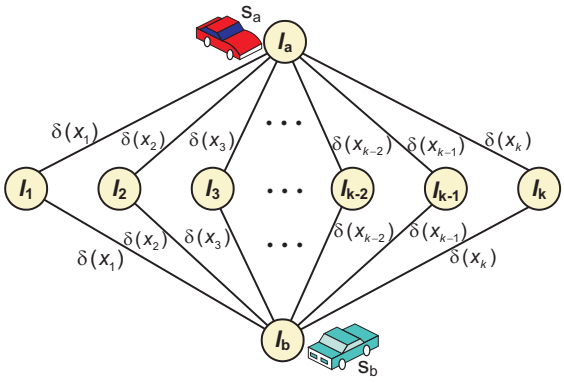


Fig. 2: Reduction of the partition problem to the sensor dispatch decision problem.

We then reduce the *partition problem* [39], which is known to be NP-complete, to our problem. Given a finite set $\mathcal{X}$ in which each element $x_i \in \mathcal{X}$ is associated with a number $\delta(x_i)$, the partition problem is to determine whether we can partition $\mathcal{X}$ into two subsets such that the sums of their associated numbers are equal.

Let $\mathcal{X} = \{x_1, x_2, \cdots, x_k\}$ be an instance of the partition problem. We construct an instance of the sensor dispatch decision problem as shown in Fig. 2. Initially, mobile sensors s_a and s_b are located at l_a and l_b, respectively, each with initial energy of $\sum_{i=1}^{k} \delta(x_i)$. For each x_i, $i = 1..k$, we construct a location l_i such that the energy required to move from both l_a and l_b to l_i is $\delta(x_i)$. Now, consider a $2k$-round problem such that in the $(2i-1)$th round an event appears at l_i and in the $(2i)$th round two events appear at l_a and l_b, $i = 1..k$. We show that $\mathcal{X}$ has a solution if and only if the dispatch problem has a solution.

Suppose that we have a solution to the sensor dispatch decision problem. The solution must dispatch one mobile sensor to one event location in each odd round and then dispatch the same mobile sensor back to its original location in the subsequent even round. That is, either s_a or s_b will move to l_i and move back in the $(2i-1)$th and the $(2i)$th rounds, respectively. The total energy consumption of s_a and s_b is thus $2 \cdot \sum_{i=1}^{k} \delta(x_i)$. Since the initial energy of s_a and s_b is $\sum_{i=1}^{k} \delta(x_i)$, both of s_a and s_b exhaust their energy after $2k$ rounds. This implies that s_a and s_b have moved the same distance. Therefore, the sets of locations visited by s_a and s_b all constitute a solution to the partition problem. This proves the *if* part.

Conversely, suppose that subsets $\mathcal{X}_a$ and $\mathcal{X}_b$ are a solution to the partition problem. Then, in the $(2i-1)$th round, $i = 1..k$, we can dispatch s_a (respectively, s_b) to visit l_i if $\mathcal{X}_a$ (respectively, $\mathcal{X}_b$) has an element associated with a number $\delta(x_i)$, and move it back in the $(2i)$th round. Clearly, both of s_a and s_b move the same distance and exhaust their energy (i.e., $\sum_{i=1}^{k} \delta(x_i)$) in the $(2k)$th round. This constitutes a solution to the sensor dispatch decision problem, thus proving the *only if* part. □

4 A Centralized Dispatch Algorithm

Here, we propose a centralized algorithm to dispatch mobile sensors. The idea is to minimize their moving energy while keeping their energy consumption balanced after each round. Without loss of generality, we delete those mobile sensors

TABLE 1: Summary of notations.

notation	definition
$\mathcal{L}$	the set of event locations ($\|\mathcal{L}\| = m$)
$\mathcal{S}$	the set of mobile sensors ($\|\mathcal{S}\| = n$)
DS_i	the dispatch schedule of a mobile sensor s_i
e_i^{init}	the initial energy of s_i
e_{move}	the energy cost for a mobile sensor to move one unit distance
$\mathcal{P}_j$	the preference list of an event location l_j (or an event grid g_j)
B_j	the bound of l_j (or g_j)
β	a system parameter to determine the bound B_j
$\hat{\mathcal{L}}_i$	a cluster of event locations
$\phi(\hat{\mathcal{L}}_i)$	the estimated cost of $\hat{\mathcal{L}}_i$
α_j	the iteration counter of an event grid g_j

that do not have sufficient energy to reach any location in $\mathcal{L}$ from $\mathcal{S}$. Considering the values of $|\mathcal{S}|$ and $|\mathcal{L}|$, there are two cases to be discussed. When $|\mathcal{S}| \geq |\mathcal{L}|$, we translate the dispatch problem to a maximum-matching problem in a weighted complete bipartite graph. On the other hand, when $|\mathcal{S}| < |\mathcal{L}|$, we partition $\mathcal{L}$ into $|\mathcal{S}|$ clusters so that each mobile sensor only needs to visit one cluster of event locations. Then, the maximum-matching approach is applied again.

4.1 Case of $|\mathcal{S}| \geq |\mathcal{L}|$

We first construct a weighted complete bipartite graph $\mathcal{G} = (\mathcal{S} \cup \mathcal{L}, \mathcal{S} \times \mathcal{L})$. Each mobile sensor and each event location is converted into a vertex. Edges only connect vertices between $\mathcal{S}$ and $\mathcal{L}$. For each $s_i \in \mathcal{S}$ and each $l_j \in \mathcal{L}$, its weight is defined as $w(s_i, l_j) = e_{\text{move}} \times d(s_i, l_j)$. Then, the sensor dispatch problem is formulated as the problem of finding a matching $\mathcal{M}$ in $\mathcal{G}$ such that

1. The number of matches is maximum.
2. The sum of the weights associated with all matches is as small as possible.
3. The standard deviation of the weights associated with all matches is as small as possible.

Note that objective 1 is a necessary condition in our algorithm. However, minimum values for both objectives 2 and 3 may not always be achieved, but keeping them small is our goal.

Below, we propose a heuristic to find $\mathcal{M}$:

1. For each location $l_j \in \mathcal{L}$, we associate with it a *preference list* $\mathcal{P}_j$, which contains all mobile sensors ranked by their weights in correspondence with l_i in an ascending order. In case of tie, sensors' IDs are used to break the tie.
2. Construct a queue $\mathcal{Q}$ containing all locations in $\mathcal{L}$[1].
3. To achieve objective 3, we create a *bound* B_j for each location $l_j \in \mathcal{L}$ to restrict the mobile sensors that l_j can match with. Initially, we set $B_j = w(s_i, l_j)$ such that s_i is the βth element in l_j's preference list $\mathcal{P}_j$, where β is a system parameter.
4. Dequeue an event location, say, l_j from $\mathcal{Q}$.
5. To achieve objective 2, we select the first candidate mobile sensor, say, s_i from $\mathcal{P}_j$ and try to match s_i with l_j. If s_i is also unmatched, we add the match (s_i, l_j) into $\mathcal{M}$ and remove s_i from $\mathcal{P}_j$. Otherwise, s_i must have matched with another location, say, l_o. Then, l_j and l_o will compete by their bounds B_j and B_o. Location l_j wins the competition if one of these conditions is true:
   - $B_j > B_o$: Since l_j has raised to a higher bound, we match s_i with l_j.
   - $B_j = B_o$ and $w(s_i, l_j) < w(s_i, l_o)$: Since moving s_i to l_j is more energy-efficient, we match s_i with l_j.
   - $B_j = B_o$, s_i is the only candidate of l_j, and l_o has more than one candidate: In this case, if s_i is not matched with l_j, l_j has to increase its bound B_j. However, l_o may not increase its bound B_o if s_i is not matched with l_o. Thus, we match s_i with l_j.

   If l_j wins the competition, we replace the pair (s_i, l_o) in $\mathcal{M}$ by (s_i, l_j), remove s_i from $\mathcal{P}_j$, enqueue l_o into $\mathcal{Q}$, and go to step 7. Otherwise, we remove s_i from $\mathcal{P}_j$ (since l_j will not consider s_i any more) and go to step 6.
6. If l_j still has candidates in $\mathcal{P}_j$ (under bound B_j), go to step 5 directly. Otherwise, we increase l_j's bound to $B_j = w(s_k, l_j)$ such that s_k is the βth element in the *current* $\mathcal{P}_j$ and then go to step 5. (Note that since $\mathcal{P}_j$ is sorted in an ascending order and the first mobile sensor s_i is always removed from $\mathcal{P}_j$ after step 5, we will obtain a new larger bound $B_j = w(s_k, l_j) > w(s_i, l_j)$.)
7. If $\mathcal{Q}$ is empty, the algorithm terminates; otherwise, go to step 4.

Since $|\mathcal{S}| \geq |\mathcal{L}|$, each event location will eventually find a mobile sensor to match with. Thus, the above algorithm must terminate and return a maximum matching. Here, the bound of an event location implicitly indicates that if the current candidate mobile sensor cannot match with the event location, the event location may possibly match with another mobile sensor of a distance equal to that bound. Since we want to balance the energy consumption among mobile sensors by preventing some mobile sensors from moving too long distances, we should avoid raising the bounds of event locations. The system parameter β determines the number of candidate mobile sensors and the bound. We suggest setting $\beta > 1$ and will discuss the effect of β in Section 6.6.

Fig. 3 gives an example, where $\beta = 2$. The energy cost to move each mobile sensor to each event location is given in Fig. 3(a). Let $\mathcal{Q} = \{l_1, l_2, l_3\}$. Fig. 3(b) shows the running iterations. In iteration 1, l_1 has two candidates s_1 and s_2 in its $\mathcal{P}_1$ and sets bound $B_1 = w(s_2, l_1) = 99$. Since s_1 is l_1's first candidate and s_1 is also unmatched, we match s_1 with l_1. Similarly, s_2 is matched with l_2. In iteration 2, l_3 finds that its first candidate s_1 has already matched with l_1, it thus competes with l_1 by their bounds. Since $B_3 = 111 > B_1 = 99$, we thus replace the pair (s_1, l_1) by (s_1, l_3) in $\mathcal{M}$. In iteration 3, l_1 checks its remaining candidates and competes with l_2

1. It can be verified that any order of the locations in $\mathcal{Q}$ can lead to the same result in our scheme. So, we do not specify the order in this step.

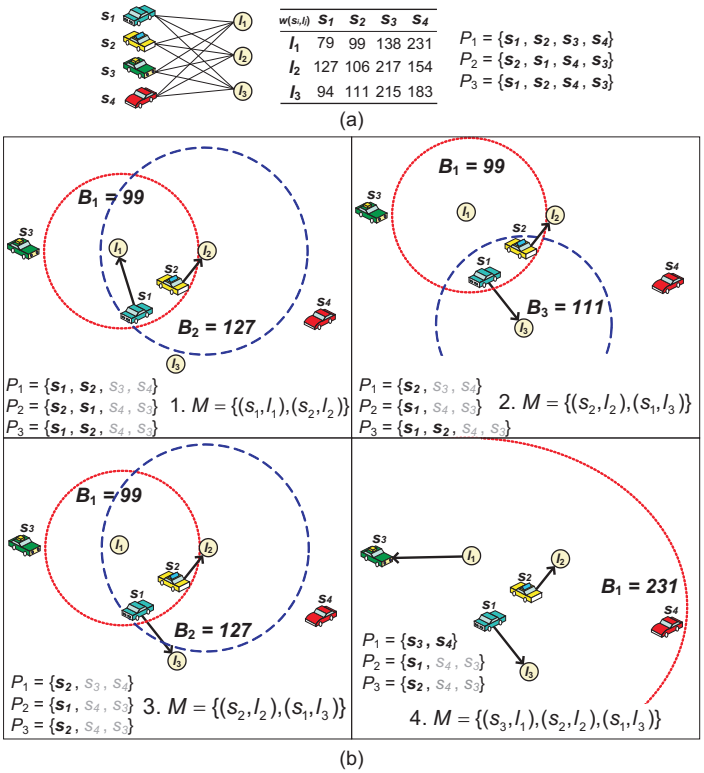


Fig. 3: An example of finding $\mathcal{M}$: (a) the energy consumption to move each mobile sensor to each event location and (b) the running iterations.

for s_2. However, because $B_1 = 99 < B_2 = 127$, l_1 loses the competition. Therefore, in iteration 4, l_1 expands its bound as $B_1 = w(s_4, l_1) = 231$ and matches with its first candidate s_3. The final result is $\mathcal{M} = \{(s_3, l_1), (s_2, l_2), (s_1, l_3)\}$.

We then analyze the time complexity of the above algorithm. Recall that $|\mathcal{L}| = m$ and $|\mathcal{S}| = n$. Calculating the preference lists of all event locations takes $O(mn \lg n)$ time. The worst case to match an event location with a mobile sensor is $O(n)$ because the event location has to go through its whole preference list. Thus, computing the maximum matching $\mathcal{M}$ takes $O(mn)$ time. Therefore, the total time complexity is $O(mn \lg n + mn) = O(mn \lg n)$.

4.2 Case of $|\mathcal{S}| < |\mathcal{L}|$

When mobile sensors are fewer than event locations, we divide $\mathcal{L}$ into n ($= |\mathcal{S}|$) clusters $\hat{\mathcal{L}}_1, \hat{\mathcal{L}}_2, \cdots, \hat{\mathcal{L}}_n$ and dispatch one mobile sensor to visit one cluster of event locations. The algorithm is outlined as follows. Let $\tilde{\mathcal{L}} = \{\hat{\mathcal{L}}_1, \hat{\mathcal{L}}_2, \cdots, \hat{\mathcal{L}}_n\}$. We construct a weighted complete bipartite graph $\mathcal{G}' = (\mathcal{S} \cup \tilde{\mathcal{L}}, \mathcal{S} \times \tilde{\mathcal{L}})$ such that the vertex set contains $\mathcal{S}$ (all mobile sensors) and $\tilde{\mathcal{L}}$ (all clusters), and the edge set contains all $(s_i, \hat{\mathcal{L}}_j)$ such that $s_i \in \mathcal{S}$ and $\hat{\mathcal{L}}_j \in \tilde{\mathcal{L}}$. The weight of $(s_i, \hat{\mathcal{L}}_j)$ is defined as $w(s_i, \hat{\mathcal{L}}_j) = e_{\text{move}} \times (d(s_i, \hat{\mathcal{L}}_j) + \phi(\hat{\mathcal{L}}_j))$, where $d(s_i, \hat{\mathcal{L}}_j)$ is the distance from s_i to the nearest event location in $\hat{\mathcal{L}}_j$ and $\phi(\hat{\mathcal{L}}_j)$ is the moving distance for s_i to visit all event locations in $\hat{\mathcal{L}}_j$ (below, we call $\phi(\hat{\mathcal{L}}_j)$ the *cluster cost*). Then, we adopt the algorithm in Section 4.1 to find a maximum matching $\mathcal{M}$ on $\mathcal{G}'$. For each $(s_i, \hat{\mathcal{L}}_j) \in \mathcal{M}$, we dispatch s_i to the nearest event location in $\hat{\mathcal{L}}_j$ and then apply any TSP solution for s_i to visit other event locations in $\hat{\mathcal{L}}_j$.

There are two remaining issues in the above algorithm: i) how to estimate cluster costs and ii) how to cluster event locations. For issue (i), since the TSP problem is NP-complete

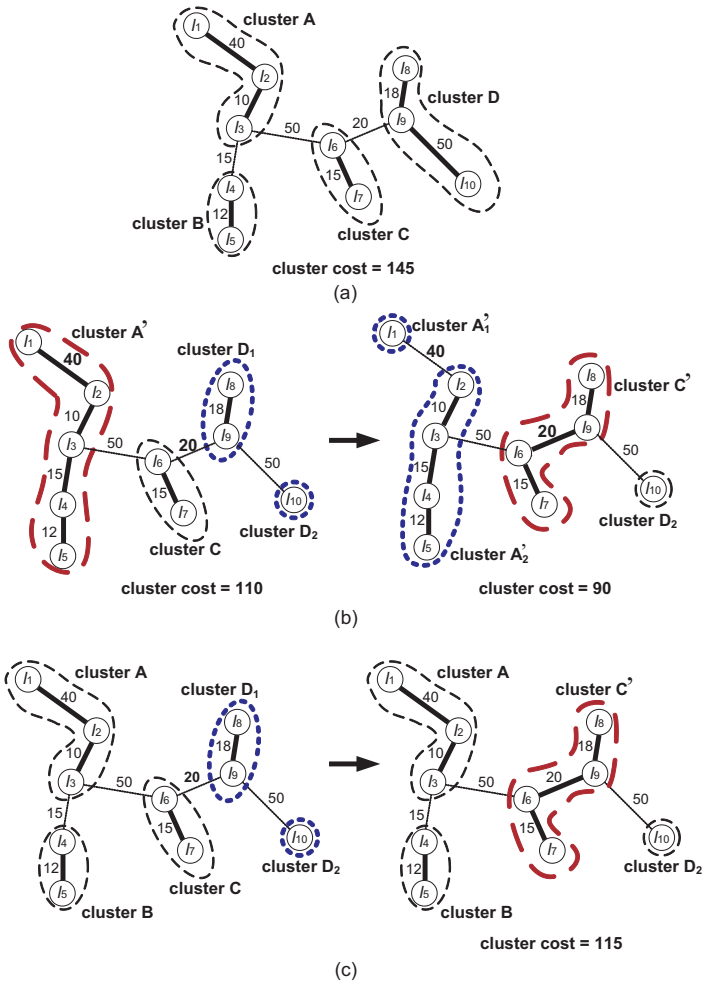


Fig. 4: Examples of clustering results: (a) the K-means clustering scheme, (b) the MaxMin clustering scheme, and (c) the balanced clustering scheme.

and it is not yet known which mobile sensor will visit which cluster (different mobile sensors may land at different initial locations of a cluster), we need to estimate a value for each $\phi(\hat{\mathcal{L}}_j)$. Since some TSP heuristics [10] are based on constructing a minimum spanning tree from the nodes to be visited, we propose letting $\phi(\hat{\mathcal{L}}_j)$ be the sum of all edge weights[2] in the minimum spanning tree of $\hat{\mathcal{L}}_j$. Fig. 4(a) shows an example, where $\phi(A) = 50$, $\phi(B) = 12$, $\phi(C) = 15$, and $\phi(D) = 68$. For issue (ii), we introduce three schemes below.

K-means clustering scheme [40]: This scheme groups event locations based on their relative distances in the sensing field such that those locations close to each other are grouped together. Initially, $\mathcal{L}$ is randomly partitioned into $|\mathcal{S}|$ nonempty clusters. Then, an iterative process is conducted. In each iteration, the *central point* of each cluster is computed. Explicitly, given a cluster of locations $\{(x_1, y_1), (x_2, y_2), \cdots, (x_p, y_p)\}$, its central point is calculated as $\left(\frac{1}{p}\sum_{i=1}^{p} x_i, \frac{1}{p}\sum_{i=1}^{p} y_i\right)$. Then, $\mathcal{L}$ is repartitioned such that locations closest to the same central point are put into the same cluster. The process is repeated until no cluster is changed. Fig. 4(a) shows an example by assuming $|\mathcal{S}| = 4$.

The time complexity of the K-means clustering scheme is $O(|\mathcal{S}| \cdot |\mathcal{L}| \cdot \rho) = O(mn\rho)$, where ρ is the number of iterations to perform K-means for clustering. According to [40], ρ usually ranges from tens to hundreds.

2. The weight of an edge (l_i, l_j) is the distance between l_i and l_j.

MaxMin clustering scheme: The K-means clustering scheme could be inefficient when the distribution of event locations is irregular or sparse. For example, l_1 and l_{10} in Fig. 4(a) are far away from other locations and thus considered *sparse*. Therefore, we propose a MaxMin clustering scheme that is based on the result of K-means and then iteratively splits and merges some clusters to obtain better clustering. Intuitively, clusters with sparse locations should be split. In each iteration, we first construct the minimum spanning tree of each cluster. Let $w_{\max}^{\text{intra}}$ be the maximum of the maximum edge weight in each cluster among all clusters and $w_{\min}^{\text{inter}}$ be the minimum of the distances between all cluster pairs, where the distance between two clusters is the distance between the two closest locations in the two clusters. We split the cluster which contains the edge with weight $w_{\max}^{\text{intra}}$ by removing that edge. Then, among all $|\mathcal{S}|+1$ clusters, we merge two clusters with distance $w_{\min}^{\text{inter}}$. The above operation is repeated until $w_{\max}^{\text{intra}} \leq w_{\min}^{\text{inter}}$. This scheme can avoid clusters containing very long edges, thereby reducing the energy costs for mobile sensors to visit clusters. Fig. 4(a) and (b) give an example. In Fig. 4(a), $w_{\max}^{\text{intra}} = 50$ (in cluster D) and $w_{\min}^{\text{inter}} = 15$ (between clusters A and B). We thus split cluster D into two clusters D_1 and D_2, and then merge clusters A and B, as shown in Fig. 4(b). Now, we have $w_{\max}^{\text{intra}} = 40$ and $w_{\min}^{\text{inter}} = 20$, enforcing us to further split A' and then to merge C and D_1. After this operation, we have $w_{\max}^{\text{intra}} = 20$ and $w_{\min}^{\text{inter}} = 40$. Thus, the scheme terminates and the final result is shown in Fig. 4(b).

The time complexity of the MaxMin clustering scheme is analyzed as follows. Since there are $|\mathcal{L}| = m$ event locations, we need $\frac{m(m-1)}{2}$ edges to connect each pair of locations (in a complete graph). The worst case occurs when one half of these $\frac{m(m-1)}{2}$ edges belong to intra-cluster edges (after executing K-means) and the other half of them belong to inter-cluster edges. We then build a maximum binary heap $\Psi_{\max}$ and a minimum binary heap $\Psi_{\min}$ to maintain all intra-cluster and inter-cluster edges, respectively. Building a heap requires $O(\frac{m(m-1)}{4}) = O(m^2)$ time. Recall that in each iteration of MaxMin, we find the edges with weights $w_{\max}^{\text{intra}}$ and $w_{\min}^{\text{inter}}$ and then exchange them (i.e., let the edge with weight $w_{\max}^{\text{intra}}$ become an inter-cluster edge and the edge with weight $w_{\min}^{\text{inter}}$ become an intra-cluster edge) if $w_{\max}^{\text{intra}} \leq w_{\min}^{\text{inter}}$. Thus, one iteration involves four operations in the two heaps: deletion of the maximum in $\Psi_{\max}$, deletion of the minimum in $\Psi_{\min}$, insertion of $w_{\max}^{\text{intra}}$ into $\Psi_{\min}$, and insertion of $w_{\min}^{\text{inter}}$ into $\Psi_{\max}$. Each of these four operations takes $O(\lg m^2) = O(2\lg m)$ time. Thus, the total time to complete one iteration in MaxMin is $4 \cdot O(2\lg m) = O(8\lg m)$. Now, we calculate how many iterations are executed in MaxMin. The worst case occurs when $\Psi_{\max}$ contains the edges with the smallest $\frac{m(m-1)}{4}$ weights and $\Psi_{\min}$ contains the edges with the largest $\frac{m(m-1)}{4}$ weights. In this case, MaxMin is executed until $\Psi_{\max}$ and $\Psi_{\min}$ exchange all of their edges. So, there are totally $\frac{m(m-1)}{4}$ iterations in the worst case. Thus, the total time complexity of MaxMin is (time to execute K-means) + (time to build $\Psi_{\max}$ and $\Psi_{\min}$) + (time to execute $\frac{m(m-1)}{4}$ iterations) = $O(mn\rho)+2O(m^2)+\frac{m(m-1)}{4}\cdot O(8\lg m) = O(mn\rho+m^2\lg m)$.

Balanced clustering scheme: The MaxMin clustering scheme can minimize the total cost of clusters, but it may lead to unbalanced clusters. Thus, we should try to reduce both the total cost of clusters and the standard deviation of clusters' costs. To do so, we first cluster event locations by K-means. Then, we iteratively split and merge some clusters. In each iteration, we split the cluster with the maximum cost into two new clusters, say, $\hat{c}_i$ and $\hat{c}_j$ such that $|\phi(\hat{c}_i) - \phi(\hat{c}_j)|$ is minimized. Then, among all $|\mathcal{S}|+1$ clusters, we merge the two clusters into one new cluster, say, $\hat{c}_k$ such that $\phi(\hat{c}_k)$ is minimized. This operation is repeated until the total cluster cost is no longer reduced. Fig. 4(a) and (c) show an example. In Fig. 4(a), D has the maximum cost of $\phi(D) = 68$, so we split it. There are two ways to split D. One is to split D into $D_1 = \{l_8, l_9\}$ and $D_2 = \{l_{10}\}$. The other way is to split D into $D_3 = \{l_8\}$ and $D_4 = \{l_9, l_{10}\}$. Since $|\phi(D_1) - \phi(D_2)| = 18 < |\phi(D_3) - \phi(D_4)| = 50$, we split D into D_1 and D_2, as shown in Fig. 4(c). Then, we merge C and D_1 such that the new cluster C' has the minimum cost of 53. In the next iteration, since C' has the maximum cost, we split it into $C'_1 = \{l_6, l_7\}$ and $C'_2 = \{l_8, l_9\}$. However, among the five clusters A, B, C'_1, C'_2 and D_2, we have to merge C'_1 and C'_2 because the cost of the merged cluster is the smallest. Since the total cluster cost is still 115, the balanced clustering scheme terminates. The final result is shown in Fig. 4(c).

The time complexity of the balanced clustering scheme is analyzed as follows. Clustering event locations (by K-means) takes $O(mn\rho)$ time. Calculating the cost of each cluster by constructing its minimum spanning tree can be done by the Prime's algorithm [41]. The worst case occurs when one cluster contains $(m-n+1)$ locations and each of other $(n-1)$ clusters only contains one location. In this case, constructing the minimum spanning tree in the largest cluster takes $O((m-n+1)^2)$ time. Then, we build a maximum heap $\Psi_{\max}^C$ to maintain these n clusters, which takes $O(n)$ time. An iteration of the balanced clustering scheme involves the following four operations: 1) Delete the maximum in $\Psi_{\max}^C$ to find the cluster with the maximum cost, which takes $O(\lg n)$ time. 2) Split that cluster into two balanced clusters. The worst case occurs when we have to search all tree edges in the cluster with $(m-n+1)$ locations. This takes $O(m-n)$ time. 3) Merge two clusters such that the cost of the new cluster derived by merging these two clusters is minimized. Since there are $(n+1)$ clusters, this operation takes $O(C_2^{n+1}) = O(n^2)$ time. 4) Insert the two new clusters into $\Psi_{\max}^C$, which takes $2O(\lg n)$ time. Thus, an iteration takes $O(\lg n)+O(m-n)+O(n^2)+2O(\lg n) = O(n^2)$ time. Since the balanced clustering scheme stops when the total cluster cost is no longer reduced, the number of iterations should be no more than that of MaxMin. Therefore, the time complexity of the balanced clustering scheme is $O(mn\rho)+O((m-n+1)^2)+\frac{m(m-1)}{4}O(n^2) = O(mn\rho+m^2n^2)$.

We then analyze the total time complexity when $|\mathcal{S}| < |\mathcal{L}|$. Recall that it spends $O((m-n+1)^2)$ time to calculate the costs of all clusters. Since $|\tilde{\mathcal{L}}| = n$ and $|\mathcal{S}| = n$, it takes $O(n^2\lg n)$ time to execute the algorithm in Section 4.1. Therefore, if the K-means clustering scheme is adopted, the total time complexity is $O(mn\rho)+O((m-n+1)^2)+O(n^2\lg n) = O(mn\rho+(m-n+1)^2+n^2\lg n)$. If the MaxMin clustering scheme is adopted, the total time complexity is $O(mn\rho+m^2\lg m)+O((m-n+1)^2)+O(n^2\lg n) = O(mn\rho+m^2\lg m)$. Finally, since the balanced clustering scheme has already calculated the cost of each cluster, the total time complexity is $O(mn\rho+m^2n^2)+O(n^2\lg n) = O(mn\rho+m^2n^2)$.

5 A Distributed Dispatch Algorithm

Next, we propose a distributed heuristic based on a grid structure. There are three challenges. First, how can we efficiently exchange the messages between event locations and mobile sensors so that they can know each other? Second, how can

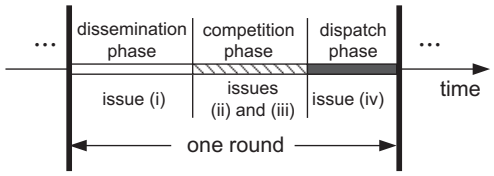


Fig. 5: Phases of a round in the distributed dispatch algorithm.

one event location compete for a mobile sensor when the event location is not aware of other event locations? Third, since it is difficult to cluster event locations, how can we handle the case when $|\mathcal{S}| < |\mathcal{L}|$? To address the above challenges, we propose a distributed algorithm outlined as follows. We partition the sensing field into grids and elect one static sensor as the *grid head* in each grid. Mobile sensors report their locations and remaining energy to their grid heads. On detecting events, static sensors notify their grid heads of the events. A grid with events is called an *event grid*. We assume that all sensors are roughly time-synchronized (how to do so is beyond the scope of this work). The time is divided into multiple *rounds* and each round is further divided into three phases (refer to Fig. 5). In the *dissemination phase*, each grid head collects the locations of events and mobile sensors in that grid. Then, the existence of mobile sensors is advertised (respectively, queried) by grids with mobile sensors (respectively, event grids). In the *competition phase*, event grids bid for mobile sensors by sending invitation messages. Then, mobile sensors determine their target grids and compute their dispatch schedules. Finally, in the *dispatch phase*, mobile sensors travel according to their dispatch schedules and visit all event locations in these grids. Lengths of these phases depend on situations. Note that strict time synchronization of these phases is not necessary.

To balance the energy consumption among mobile sensors, event grids adopt the bound concept in Section 4.1 to bid for mobile sensors. Specifically, each event grid maintains a preference list that ranks all mobile sensors by their energy costs. For competition purpose, it also maintains a bound. Then, event grids send *invitation (INV)* messages containing their bounds to bid for mobile sensors. On the other hand, each mobile sensor s_i maintains a dispatch schedule DS_i to record its target grids. Each s_i limits the size of its DS_i to $\lceil \frac{m}{n} \rceil$, where m is the number of event grids and n is the number of mobile sensors (how to obtain m and n will be discussed later). On receiving an INV, s_i does not immediately reply its decision but will continuously collect more INVs for a small duration Δ_t (it can be a few of average packet round-trip time). Then, s_i determines the winners (based on event grids' bounds), inserts them into its DS_i, and replies a *confirmation (CFM)* message to each of them. For each non-winning grid, s_i replies a *reject (RJT)* message containing the remaining number of free entries in its DS_i. An event grid removes those mobile sensors with no remaining capacity from its preference list and tries to invite other mobile sensors until a CFM is received or all mobile sensors run out of their capacities in this round. Note that a mobile sensor may scan its preference list more than once (explained later).

There are four remaining issues in the above discussion: i) how event grids collect locations of mobile sensors in a message-efficient way, ii) how event grids bid for mobile sensors, iii) how mobile sensors accept bids and determine their dispatch schedules, and iv) how mobile sensors visit event locations in an energy-efficient way. Fig. 5 relates these issues to our three phases.

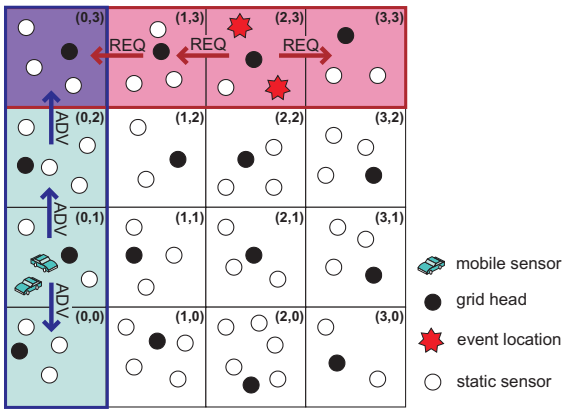


Fig. 6: An example of the grid-quorum scheme.

To address issue (i), we adopt the *grid-quorum* scheme in [11]. Each grid with mobile sensors periodically sends an *advertisement (ADV)* message containing the locations and remaining energy of the mobile sensors inside its grid to other grids in the same column. On the other hand, each event grid sends a *request (REQ)* message to other grids in the same row. This behavior ensures that each ADV and each REQ will intersect. Fig. 6 gives as an example. Grid $(0, 1)$ with two mobile sensors sends an ADV along its column, while event grid $(2, 3)$ sends an REQ along its row to search for mobile sensors. When an REQ meets an ADV, a *reply (RPY)* message containing the mobile sensors in the ADV is sent back to the REQ-initiating grid. Also, an RPY containing the event locations in the REQ is sent back to the ADV-initiating grid. In this way, both ADV-initiating and REQ-initiating grids can obtain each other's information. Note that when some grids are empty, the intersection property of the grid-quorum scheme may not exist. In this case, some recovery mechanism may be applied [11].

We then analyze the message complexity of the above scheme. Assume that the sensing field is partitioned into $\hat{M} \times \hat{N}$ grids. For each event grid, it takes $(\hat{M} - 1)$ REQs to make other grids in the same row to obtain its information. Similarly, for each grid containing mobile sensors, it takes $(\hat{N} - 1)$ ADVs to make other grids in the same column to obtain its information. Supposing that there are m event grids and n mobile sensors, the worst case occurs when n grids contain mobile sensors. Since a grid that hears both ADV and REQ has to send RYPs to the ADV-initiating and REQ-initiating grids, the message complexity of this scheme is $O(2(m(\hat{M} - 1) + n(\hat{N} - 1))) = O(m\hat{M} + n\hat{N})$.

To deal with issue (ii), event grids bid for mobile sensors as follows:

1. Each event grid g_j calculates the energy cost for each mobile sensor s_i to visit its grid. The cost is formulated by $w(s_i, g_j) = e_{\text{move}} \times (d(s_i, \gamma_j) + \phi(\mathcal{L}_j))$, where γ_j is the center of all events in g_j and $\mathcal{L}_j$ is the set of event locations in g_j. Then, g_j sorts all available mobile sensors into a preference list $\mathcal{P}_j$ according to their costs in an ascending order. Note that if the remaining energy of a mobile sensor cannot afford to visit g_j, it is removed from $\mathcal{P}_j$. If $\mathcal{P}_j$ is empty, g_j does not participate in this round but may participate in future rounds.

2. Each event grid g_j maintains an iteration counter α_j and a bound B_j. Initially, $\alpha_j = 1$ and $B_j = w(s_i, g_j)$, where s_i is the βth mobile sensor in $\mathcal{P}_j$. Also, each mobile sensor in $\mathcal{P}_j$ is marked as *unsolicited*. It is changed to *solicited* if g_j has ever sent an INV to the mobile sensor in the current iteration. A mobile sensor s_i in $\mathcal{P}_j$ is called a *candidate* for g_j if it is unsolicited and $w(s_i, g_j) \leq B_j$.
3. Each event grid g_j selects the first candidate mobile sensor, say, s_i from $\mathcal{P}_j$ and sends s_i an INV$(g_j, \alpha_j, w(s_i, g_j), B_j, c_j, n)$, where c_j is the remaining number of candidates in $\mathcal{P}_j$ under bound B_j and n is the number of mobile sensors that g_j learns in the dissemination phase. Then, g_j waits for s_i's response. If g_j receives a CFM from s_i, this algorithm terminates. Otherwise, g_j should receive an RJT containing s_i's remaining capacity. If g_j finds that s_i's remaining capacity is zero, it removes s_i from its $\mathcal{P}_j$; otherwise, g_j marks s_i as solicited and one of three cases will happen:
   a) If g_j still has candidates under bound B_j, it repeats step 3 again.
   b) If g_j has no candidate under bound B_j and there still exist unsolicited mobile sensors in $\mathcal{P}_j$, g_j increases its bound to $B_j = w(s_k, g_j)$ such that s_k is the βth unsolicited mobile sensor in the current $\mathcal{P}_j$ and repeat step 3 again.
   c) Otherwise, g_j has reached the end of its $\mathcal{P}_j$. If $\mathcal{P}_j$ is empty, the algorithm terminates; otherwise, g_j clears all entries in $\mathcal{P}_j$ as unsolicited, increases its iteration counter α_j by one, resets its bound as $B_j = w(s_i, g_j)$ such that s_i is the βth unsolicited mobile sensor in $\mathcal{P}_j$, and repeats step 3 again.

   The above case (c) occurs when g_j fails to invite any mobile sensor after examining all elements in its $\mathcal{P}_j$. In this case, g_j enters the next iteration and re-examine its $\mathcal{P}_j$ to check those mobile sensors still with remaining capacities. As will be seen later, the counter α_j may help g_j win a mobile sensor.

To deal with issue (iii), each mobile sensor s_i maintains a dispatch schedule DS_i to record event girds that it has to visit. The capacity of DS_i is set to $\lceil \frac{m}{n} \rceil$ (m can be learned from the dissemination phase and n can be obtained from INVs). It may happen that the received values of n from INVs are inconsistent (the differences should be minor). If so, the largest value is taken. INVs are processed in a batch mode after every Δ_t interval. Multiple INVs may be accepted. For all INVs with the same iteration counter, only one event grid is accepted and the others are rejected. The following rules are enforced: If s_i has no remaining capability, an RJT indicating that it has no capability is sent. Otherwise, all requesting event grids with the same iteration counter will compete for one position. Specifically, for two INV$(g_j, \alpha_j, w(s_i, g_j), B_j, c_j, n)$ and INV$(g_k, \alpha_k = \alpha_j, w(s_i, g_k), B_k, c_k, n)$, g_j wins if one of the following conditions is true: 1) $B_j > B_k$, 2) $B_j = B_k$ and $w(s_i, g_j) < w(s_i, g_k)$, and 3) $B_j = B_k$, $c_j = 1$, and $c_k > 1$. (These conditions are the same as those in the centralized algorithm.) Then, s_i sends out CFMs and RJTs, updates its DS_i, and deducts its remaining capability accordingly.

To deal with issue (iv), we propose a *two-level TSP scheme*. Given the dispatch schedule DS_i, s_i first applies any TSP solution on DS_i by regarding each event grid in DS_i as one node (for example, this node can be the center of all event locations in the corresponding event grid). This TSP solution forms the first-level solution. For the second level, for each event grid in DS_i, s_i can apply any TSP solution again to visit all event locations inside that grid. Many TSP heuristics already exist, so we omit the details.

We then analyze the message complexity of our distributed algorithm. For the competition phase, we consider the worst case as follows. Suppose that there are m event grids $\{g_1, g_2, \cdots, g_m\}$ and n mobile sensors $\{s_1, s_2, \cdots, s_n\}$, where $m > n$. Then, the DS_i of each s_i has a capacity of $\frac{m}{n}$ (for simplicity, we assume that m is divisible by n). In the first iteration, m event grids compete for one mobile sensor, say, s_1 and then only one event grid, say, g_1 wins. In this case, m event grids send m INVs and s_1 replies 1 CFM and $(m-1)$ RJTs. Thus, the total number of messages is $2m$. In the second iteration, $(m-1)$ event grids $\{g_2, g_3, \cdots, g_m\}$ compete for one mobile sensor, say, s_2 and then only one event grid, say, g_2 wins. Thus, the total number of messages is $2(m-1)$ (because $(m-1)$ event grids send $(m-1)$ INVs and s_2 replies 1 CFM and $(m-2)$ RJTs). Similarly, in the ith iteration, $(m-i+1)$ event grids compete for one mobile sensor, say, s_i by sending $(m-i+1)$ INVs, and then s_i replies 1 CFM and $(m-i)$ RJTs. So, the total number of messages is $2(m-i+1)$. Since there are at most m iterations, the overall number of messages in the competition phase is $2m+2(m-1)+\cdots+2 = 2\cdot\sum_{i=1}^{m} i = m(m+1)$. In the dispatch phase, mobile sensors move to event locations to conduct event analysis. However, there is no need to exchange messages for determining the dispatch schedules of mobile sensors. Therefore, the total message complexity of our distributed algorithm is $O(m\hat{M} + n\hat{N}) + O(m(m+1)\cdot h) = O(m\hat{M} + n\hat{N} + m^2 h)$, where the sensing field is partitioned into $\hat{M} \times \hat{N}$ grids and h is the network diameter (i.e., the average hop count between sources and destinations).

Remark 1. The length of the dissemination phase depends on the applications. Specifically, when events occur frequently, a shorter dissemination phase should be used. Otherwise, a longer dissemination phase can be adopted. One possible way to determine the length of the dissemination phase is to measure the frequency of events from historical statistics. For the competition phase, the worst case occurs when one event grid has to query all mobile sensors, each with $\lceil \frac{m}{n} \rceil$ times, where m is the number of event grids and n is the number of mobile sensors. Let RTT be the round trip time for an event grid to query and get a response from a mobile sensor. The length of the competition phase can be set as $\lceil \frac{m}{n} \rceil \cdot n \cdot RTT (\approx m \cdot RTT)$. Finally, for the dispatch phase, its length depends on the longest traveling path for a mobile sensor to visit all event grids in its dispatch schedule. We can initially allocate a longer dispatch phase and then calculate the average time for mobile sensors to finish their dispatch schedules in a few rounds. This average time can help adjust the length of the dispatch phase in the following rounds.

Remark 2. There are three different designs between the distributed algorithm and the centralized one. First, since both event grids and mobile sensors have no global knowledge, messages such as ADVs and REQs needed to be exchanged in the dissemination phase between event grids and mobile sensors. We adopt grid-quorum to facilitate such message exchanges. Second, unlike the centralized algorithm, an

event grid does not know the existence of other event grids, so it has to send INVs to compete for mobile sensors. With INVs, we can realize the competition in a distributed manner. Third, since it is difficult to cluster event grids in a distributed manner, we adopt the iteration counter α_i to handle the case when event grids are more than mobile sensors. In this way, multiple event grids can choose the same mobile sensor to visit them.

TABLE 2: Comparison on the system lifetime under different dispatching algorithms.

dispatch algorithm	$\|\mathcal{L}\|$		
	20	80	140
greedy	140.60	27.89	20.59
centralized (K-means)	163.26	29.51	24.20
centralized (MaxMin)	163.26	33.25	24.83
centralized (balanced)	163.26	32.37	25.19
distributed	203.65	33.86	21.04

6 Experimental Results

To evaluate the performances of our proposed algorithms, we have developed a simulator in Java. In our simulator, the sensing field is set as a 450 m $\times$ 300 m rectangle, on which there are 400 static sensors randomly[3] deployed. The communication distance of each sensor is set to 80 m, and static sensors can form a connected network. In each simulation, we randomly select a number of static sensors as event locations (i.e., $\mathcal{L}$) and randomly deploy a number of mobile sensors (i.e., $\mathcal{S}$). The energy consumption of mobile sensors is the one caused by their movement. Since we focus on evaluating the energy efficiency of dispatch algorithms, we ignore the effect of communication impairments. In other words, we assume that communications are reliable and each mobile sensor can obtain the exact value of n (i.e., the total number of mobile sensors) from INVs. We set $\beta = 4$ in our proposed algorithms. (We will discuss the effect of β on the system performance in Section 6.6.) The grid size is 15 m $\times$ 15 m in the distributed algorithm, so the sensing field is partitioned into 30 $\times$ 20 grids. For each simulation, at least 100 experiments (with different random seeds) are repeated, and we take their average.

6.1 System Lifetime

We first investigate the system lifetime under different dispatch algorithms. The number of mobile sensors is 40. According to [42], the moving energy consumption of a mobile sensor is 0.21 J (joule) per inch (i.e., 8.27 J per meter). We assume that each mobile sensor is equipped with two batteries, each of energy capacity 1350 mAh (milliampere-hour), for its moving energy, so we have $e_i^{\text{init}} = 29160\,\text{J}$ for $i = 1..n$. During each round, we randomly select 20, 80, and 140 static sensors as event locations. Mobile sensors then move to event locations based on the dispatch algorithms and stay at their last-visiting locations to wait for their next dispatch schedules. We compare the centralized and distributed algorithms against the greedy algorithm discussed in Section 1. When mobile sensors are fewer than event locations, the three clustering schemes in Section 4.2 are adopted to group event locations in the centralized algorithm. For the greedy algorithm, we dispatch mobile sensors to visit a subset $\mathcal{L}' \subseteq \mathcal{L}$ of event locations, where $|\mathcal{L}'| \leq |\mathcal{S}|$, and then make $\mathcal{L} = \mathcal{L} - \mathcal{L}'$. We iteratively repeat the above procedure, until $\mathcal{L}$ is empty.

Table 2 shows the system lifetime under different dispatch algorithms. The greedy algorithm has the shortest system lifetime due to two reasons. First, the greedy algorithm does not balance the energy consumption among mobile sensors, which may lead some mobile sensors early to exhaust their energy. Second, the greedy algorithm does not cluster event locations when $|\mathcal{L}| > |\mathcal{S}|$, leading mobile sensors to move a longer distance. For the centralized algorithm, different clustering schemes affect the system lifetime when $|\mathcal{L}| > |\mathcal{S}|$. In particular, both the MaxMin and balanced clustering schemes help the centralized algorithm result in a longer system lifetime, because they adjust the clustering result of K-means to reduce the total cluster cost. When $|\mathcal{L}| = 140$, the balanced clustering scheme has a longer system lifetime than the MaxMin clustering scheme, because the balanced clustering scheme not only reduces the total cluster cost but also tries to balance the costs among clusters. When $|\mathcal{L}| \leq 80$, the distributed algorithm has a longer system lifetime than the centralized algorithm. The reason is that the distributed algorithm clusters event locations into grids and thus reduces the energy consumption of mobile sensors to visit them. However, when $|\mathcal{L}| = 140$, since the number of event girds is large, mobile sensors may be asked to visit multiple event grids and thus move longer distances. We will further discuss the energy consumption of mobile sensors under different dispatching algorithms in Section 6.3.

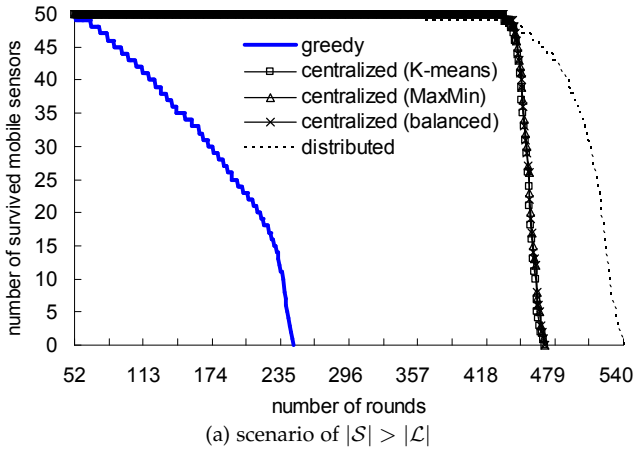
3. In our simulations, the term 'random' means uniformly random.

6.2 Survived Mobile Sensors

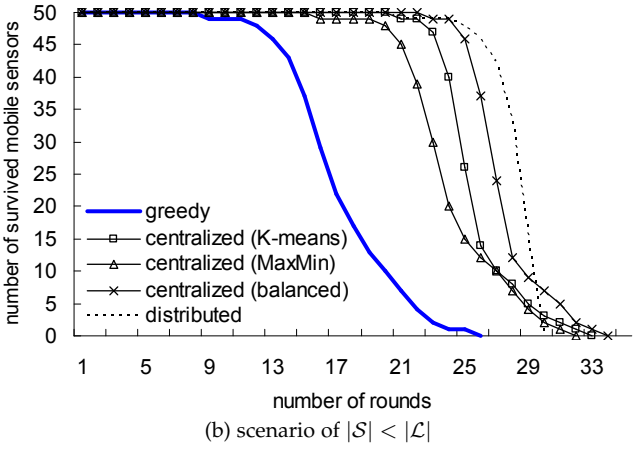
We then examine the number of survived mobile sensors in each round under different dispatching algorithms. We consider two scenarios. In the scenario of $|\mathcal{S}| > |\mathcal{L}|$, we randomly select [10, 15] static sensors as event locations in each round. On the other hand, in the scenario of $|\mathcal{S}| < |\mathcal{L}|$, we randomly select [120, 160] static sensors as event locations in each round. The number of mobile sensors is 50.

Fig. 7(a) shows the result in the scenario of $|\mathcal{S}| > |\mathcal{L}|$. For the greedy algorithm, the first mobile sensor exhausts its energy very early in the 53rd round. After 248 rounds, all mobile sensors drain out their energy. For the centralized algorithm, since we have $|\mathcal{S}| > |\mathcal{L}|$ in most rounds, the effect of different clustering schemes is not significant. The first mobile sensor exhausts its energy in the 440th round, and the system lifetime is 475 rounds. For the distributed algorithm, the first mobile sensor exhausts its energy in the 367th round, and the system lifetime is 546 rounds. With a grid structure, the distributed algorithm has a longer system lifetime than the centralized algorithm when $|\mathcal{S}| > |\mathcal{L}|$.

Fig. 7(b) shows the result in the scenario of $|\mathcal{S}| < |\mathcal{L}|$. For the greedy algorithm, the first mobile sensor exhausts its energy very early in the 9th round and the system lifetime is 25 rounds. For the centralized algorithm, different clustering schemes affect the number of survived mobile sensors. Specifically, the first mobile sensor exhausts its energy in the 21st, 16th, and 23rd rounds under the K-means, MaxMin, and balanced clustering schemes, respectively. Note that since MaxMin may generate unbalanced clusters, it spends the fewest rounds to let the first mobile sensor exhaust the energy. The system lifetimes are 32, 31, and 33 rounds under the K-means, MaxMin, and balanced clustering schemes, respectively. These results show the benefit of both balancing energy consumption of mobile sensors and clustering event locations to extend the system lifetime. For the distributed algorithm, the



(a) scenario of $|\mathcal{S}| > |\mathcal{L}|$



(b) scenario of $|\mathcal{S}| < |\mathcal{L}|$

Fig. 7: Comparison on the number of survived mobile sensors under different dispatch algorithms.

first mobile sensor exhausts its energy in the 21st round and the system lifetime is 29 rounds. Without global information, the distributed algorithm has a shorter system lifetime than the centralized algorithm when $|\mathcal{S}| < |\mathcal{L}|$.

6.3 Energy Consumption

We then evaluate the energy consumption of mobile sensors under different dispatch algorithms. The number of mobile sensors is 20, and the number of events is ranged from 10 to 80. In the simulation, mobile sensors have infinite energy. We observe the average and standard deviation of energy consumption per mobile sensor per round.

Fig. 8(a) shows the average energy consumption of mobile sensors. The averages of energy consumption under all schemes increase when the number of event locations increases, because each mobile sensor has to visit more event locations. For the greedy algorithm, when $|\mathcal{L}| \leq 70$, it has a smaller average than the centralized algorithm, because the greedy algorithm always tries to minimize the total energy consumption of mobile sensors. However, when $|\mathcal{L}| = 80$, the greedy algorithm has a larger average than the centralized algorithm. This shows the necessity of clustering event locations, especially when the number of event locations is much larger than that of mobile sensors (around $|\mathcal{L}| \geq 4 \cdot |\mathcal{S}|$ as shown in Fig. 8(a)). For the centralized algorithm, different clustering schemes have impacts on the average of energy consumption when $|\mathcal{L}| \geq 30$. Both of the MaxMin and balanced clustering schemes help the centralized algorithm to reduce the average of energy consumption, because they try to adjust the clustering result of K-means by reducing the total cluster cost. For the distributed algorithm, it has a smaller average than the centralized algorithm when $|\mathcal{L}| \leq 50$. Interestingly, the distributed algorithm even incurs a smaller average than the greedy algorithm when $|\mathcal{L}| \leq 20$. This is due to two reasons. First, in the distributed algorithm, some mobile sensors might be asked to visit their current grids, thereby reducing their energy consumption. Second, each event grid automatically *clusters* those event locations inside itself. That is, it naturally clusters event locations when $|\mathcal{S}| \geq |\mathcal{L}|$. However, when $|\mathcal{L}|$ increases, the number of event grids may become larger and thus a mobile sensor may need to visit multiple event grids. In this case, since the distributed algorithm does not have global information, mobile sensors may need to move longer distances as compared to the centralized algorithm.

Fig. 8(b) shows the standard deviation of energy consumption of mobile sensors. The greedy algorithm has a larger standard deviation than the centralized algorithm (under the K-means and balanced clustering schemes). The reason is that the greedy algorithm does not balance the energy consumption among mobile sensors, which causes some mobile sensors to exhaust their energy earlier. For the centralized algorithm, different clustering schemes result in different standard deviations. The balanced clustering scheme has the smallest standard deviation because it tries to balance the costs among clusters. On the other hand, the standard deviation of the MaxMin clustering scheme significantly increases when $|\mathcal{L}|$ increases. The reason is that the MaxMin clustering scheme tries to reduce the overall cluster cost, resulting in unbalanced clusters. (We will further discuss this issue in Section 6.4.) For the distributed algorithm, since event grids are not clustered and they do not have global knowledge, some mobile sensors may be asked to visit more event grids, thereby increasing the standard deviation.

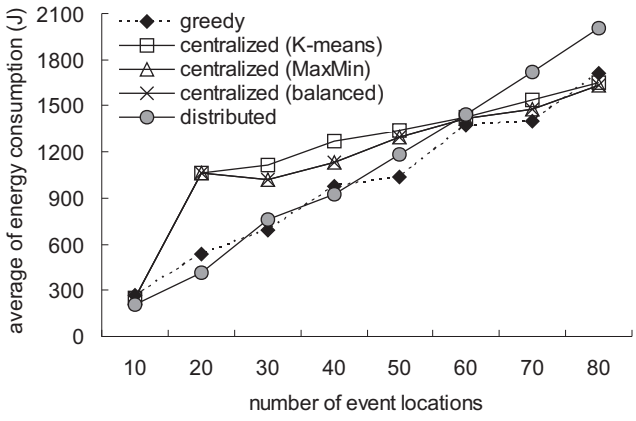
From Fig. 8, we conclude that the distributed algorithm can reduce the average energy consumption of mobile sensors when there are fewer event locations, but increase the standard deviation of energy consumption of mobile sensors. With the K-means and MaxMin clustering schemes, the centralized algorithm can reduce both the average and standard deviation of energy consumption of mobile sensors.

6.4 Impact of Clustering

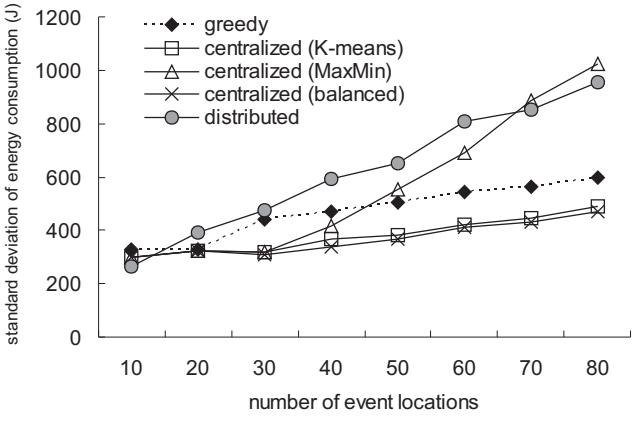
We then evaluate the clustering results derived by different schemes. We set the number of mobile sensors as 20 and randomly select 30, 40, 50, 60, 70, and 80 static sensors as event locations. We compare the K-means, MaxMin, and balanced clustering schemes.

Fig. 9(a) shows the average cluster costs under different schemes. The MaxMin clustering scheme has the minimum average cluster cost, because it always splits the cluster with the longest intra-cluster edge and then merges two clusters with the shortest inter-cluster edge. In this way, the total cluster cost is greatly reduced. On the other hand, the balanced clustering scheme also adjusts the clustering result of K-means by splitting the cluster with the maximum cost, thus incurring a smaller average cluster cost.

Fig. 9(b) shows the standard deviation of cluster costs under different schemes. Since the MaxMin clustering scheme has a goal of reducing the total cluster cost, it may generate unbalanced clusters, thus increasing the standard deviation. This

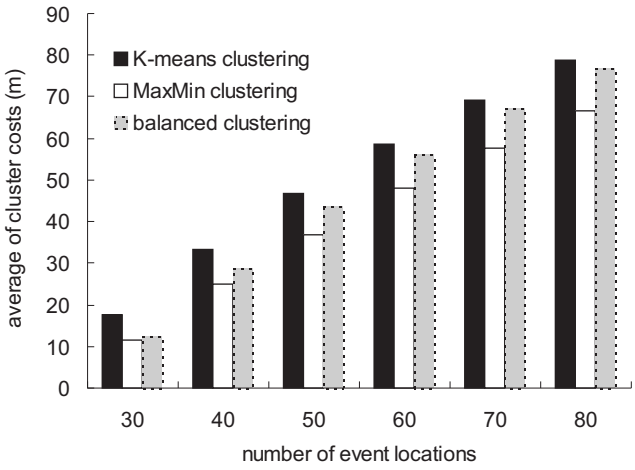


(a) average of energy consumption

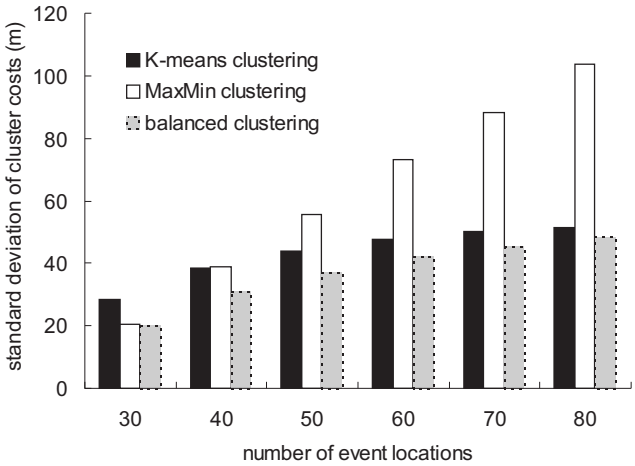


(b) standard deviation of energy consumption

Fig. 8: Comparison on the energy consumption of mobile sensors under different dispatch algorithms.



(a) average of cluster costs



(b) standard deviation of cluster costs

Fig. 9: Comparison on the cluster costs under different schemes.

effect is more prominent when the number of event locations increases. In Fig. 9(b), when $|\mathcal{L}| = 30$, the MaxMin clustering scheme has a smaller standard deviation than K-means. The reason is that the average number of event locations in each cluster is only 1.5. By adjusting the clustering result of K-means, the MaxMin clustering scheme could have more balanced clusters. On the other hand, the balanced clustering scheme has the minimum standard deviation, because it tries to balance the costs between two split clusters.

From Fig. 9(b), the MaxMin clustering scheme has the largest standard deviation. This is because MaxMin always groups sparse event locations into zero-cost *one-node clusters*. Fig. 10 shows the ratio of one-node clusters under different schemes. When $|\mathcal{L}| = 30$, the ratios of all schemes are more than 60%, because the average number of event locations in each cluster is smaller than 2. When $|\mathcal{L}|$ increases, the ratio tends to decrease. Such an effect is more significant under the K-means and balanced clustering schemes. The ratio of MaxMin is always larger than 37.8%, which makes its standard deviation increase (as shown in Fig. 9(b)). On the other hand, since the balanced clustering scheme has the smallest ratio, it can result in more balanced clusters.

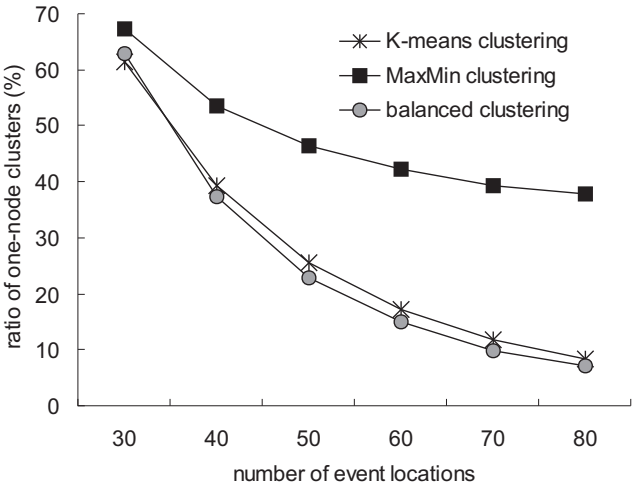


Fig. 10: Comparison on the ratio of one-node clusters under different schemes.

6.5 Communication Cost

We then measure the number of messages incurred by the centralized and distributed algorithms. We set $10 \leq |\mathcal{L}| \leq 80$ and let $|\mathcal{S}| = |\mathcal{L}|$. The centralized algorithm may incur the following message costs: 1) The sink (or the server) broadcasts commands to all static and mobile sensors to ask them to report their current states. 2) Static sensors report any detected event. 3) Mobile sensors report their locations and remaining energy to the sink. 4) The sink transmits its dispatch schedules to all mobile sensors.

Fig. 11 shows the numbers of messages sent by both algorithms. The centralized algorithm incurs more message

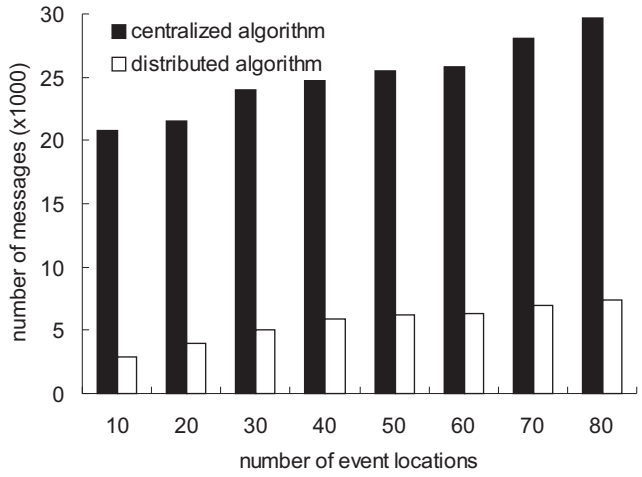


Fig. 11: Comparison on the number of messages incurred the centralized and distributed algorithms.

exchanges because the sink needs to notify all sensors to report their information through network flooding[4]. On the other hand, the grid structure makes the distributed algorithm more message-efficient for two reasons. First, no costly flooding is incurred. Second, only grid heads need to conduct inter-grid message exchanges.

6.6 Effect of β value

We finally investigate the effect of system parameter β on the performance of the centralized algorithm. To eliminate the effect of clustering, we set the numbers of both event locations and mobile sensors as 20 and 80. The value of β is increased from 1 to 10. We observe the average and standard deviation of energy consumption of mobile sensors, as shown in Fig. 12. The average energy consumption decreases while the standard deviation of energy consumption increases when β grows. The reason is that each event location can have more candidate mobile sensors and its bound value will increase faster. In this case, some event locations could match mobile sensors closer to them, thereby reducing the total energy consumption of mobile sensors. On the other hand, some other event locations may be asked to match mobile sensors relatively farther from them, and thus makes the energy consumption of mobile sensors unbalanced. From Fig. 12, we find the best value of β to be around 4 or 5 since both the average and standard deviation of energy consumption can be kept quite small.

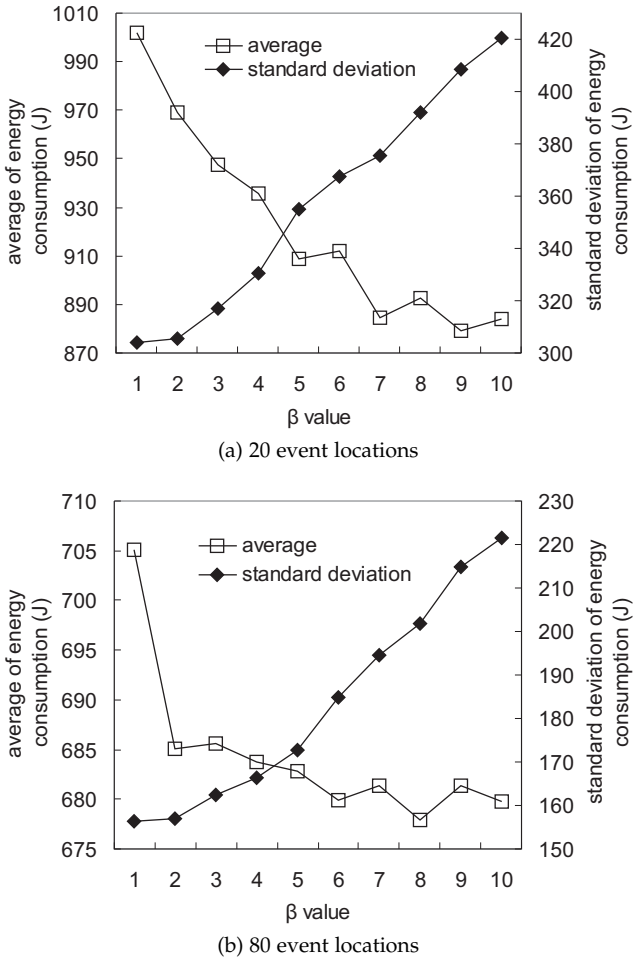


Fig. 12: Effect of β value on the average and standard deviation of energy consumption of mobile sensors.

7 Conclusions and Future Work

In this paper, we have proposed energy-balanced centralized and distributed algorithms to efficiently dispatch mobile sensors in a hybrid WSN. We formulate a general sensor dispatch problem that allows an arbitrary relationship between the numbers of mobile sensors and event locations. Our dispatch algorithms can extend the system lifetime by reducing and balancing the energy consumption of mobile sensors. In the centralized algorithm, when there are more mobile sensors, we translate the sensor dispatch problem to a maximum-matching problem in a weighted complete bipartite graph. On the other hand, when there are more event locations, they are grouped into clusters and then each mobile sensor is assigned to one cluster of event locations. In the distributed algorithm, a grid-quorum approach is adopted to reduce the message complexity for event grids to obtain the information of mobile sensors. Then, by applying the bound concept in the centralized algorithm, event grids can bid for mobile sensors. Mobile sensors then apply the proposed two-level TSP scheme to visit event locations. Simulation results have shown that our proposed dispatch algorithms can extend the system lifetime compared to the greedy algorithm.

Next, we give several future research topics. In this work, we consider dispatching mobile sensors in a sensing field without obstacles. Several studies [43]–[45] have discussed how to find the shortest path for a mobile sensor to reach its destination without colliding with any obstacle. These results can help extend our dispatch solutions. In addition, we make several assumptions on the movement model of mobile sensors such as uniform moving speed and uniform energy consumption. How to relax these assumptions deserves further investigation. Finally, for some critical event applications, events have their time constraints. Thus, a more challenging issue is how to dispatch mobile sensors to event locations with delay constraints.

4. Here, we adopt the naive flooding scheme. Although a more efficient broadcast scheme can be used, it is out of the scope of this work.

References

[1] G. Cao, G. Kesidis, T.F.L. Porta, B. Yao, and S. Phoha, "Purposeful mobility in tactical sensor networks," *Sensor Network Operations*, Wiley-IEEE Press, 2006.

[2] Y.C. Wang and Y.C. Tseng, "Intentional mobility in wireless sensor networks," *Wireless Networks: Research, Technology and Applications*, Nova Science Publishers, 2009.
[3] M.A. Batalin, M. Rahimi, Y. Yu, D. Liu, A. Kansal, G.S. Sukhatme, W.J. Kaiser, M. Hansen, G.J. Pottie, M. Srivastava, and D. Estrin, "Call and response: experiments in sampling the environment," *Proc. ACM Int'l Conf. Embedded Networked Sensor Systems*, pp. 25–38, 2004.
[4] T. Wark, P. Corke, P. Sikka, L. Klingbeil, G. Ying, C. Crossman, P. Valencia, D. Swain, and G. Bishop-Hurley, "Transforming agriculture through pervasive wireless sensor networks," *IEEE Pervasive Computing*, vol. 6, no. 2, pp. 50–57, 2007.
[5] Y.C. Tseng, Y.C. Wang, K.Y. Cheng, and Y.Y. Hsieh, "iMouse: An integrated mobile surveillance and wireless sensor system," *IEEE Computer*, vol. 40, no. 6, pp. 60–66, 2007.
[6] R. Rao and G. Kesidis, "Purposeful mobility for relaying and surveillance in mobile ad hoc sensor networks," *IEEE Trans. Mobile Computing*, vol. 3, no. 3, pp. 225–231, 2004.
[7] Y.G. Mei, Y.H. Lu, Y.C. Hu, and C.S.G. Lee, "Deployment strategy for mobile robots with energy and timing constraints," *Proc. IEEE Int'l Conf. Robotics and Automation*, pp. 2816–2821, 2005.
[8] B.I. Kim, J. Shin, S. Jeong, and J. Koo, "Effective overhead hoist transport dispatching based on the Hungarian algorithm for a large semiconductor FAB," *Int'l J. Production Research*, vol. 47, no. 10, pp. 2823–2834, 2009.
[9] H.W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955.
[10] M. Blaser, "A new approximation algorithm for the asymmetric TSP with triangle inequality," in *Proc. ACM-SIAM Symp. Discrete Algorithms*, pp. 638–645, 2003.
[11] I. Stojmenovi, "A routing strategy and quorum based location update scheme for ad hoc wireless networks," *Technical Report TR-99-09, Computer Science, SITE, University of Ottawa*, Sep. 1999.
[12] Z.J. Haas and B. Liang, "Ad hoc mobility management with uniform quorum systems," *IEEE/ACM Trans. Networking*, vol. 7, no. 2, pp. 228–240, 1999.
[13] Y.G. Mei, Y.H. Lu, Y.C. Hu, and C.S.G. Lee, "A mobility management and routing protocol using tree architecture for Internet connectivity of mobile ad hoc networks," *Proc. IEEE Int'l Conf. Computer Comm. and Networks*, pp. 967–972, 2007.
[14] N. Li, J.C. Hou, and L. Sha, "Design and analysis of an MST-based topology control algorithm," *IEEE Trans. Wireless Comm.*, vol. 4, no. 3, pp. 1195–1206, 2005.
[15] J. Wu and F. Dai, "Mobility-sensitive topology control in mobile ad hoc networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 17, no. 6, pp. 522–535, 2006.
[16] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless Comm. and Mobile Computing*, vol. 2, no. 5, pp. 483–502, 2002.
[17] T. Balch and R.C. Arkin, "Behavior-based formation control for multirobot teams," *IEEE Trans. Robotics and Automation*, vol. 14, no. 6, pp. 926–939, 1998.
[18] M.J. Mataric, G.S. Sukhatme, and E.H. Ostergaard, "Multi-robot task allocation in uncertain environments," *Autonomous Robots*, vol. 14, pp. 255–263, 2003.
[19] M. Asada, E. Uchibe, and K. Hosoda, "Co-operative behaviour acquisiton for mobile robots in dynamically changing real worlds via vision-based reinforcement learning and development," *Artificial Intelligence*, vol. 110, pp. 275–292, 1999.
[20] K.H. Park, Y.J. Kim, and J.H. Kim, "Modular Q-learning based multiagent cooperation for robot soccer," *Robotics and Autonomous Systems*, vol. 35, pp. 109–122, 2001.
[21] C.F. Touzet, "Distributed lazy Q-learning for cooperative mobile robots," *Int'l J. Advanced Robotic Systems*, vol. 1, no. 1, pp. 5–13, 2004.
[22] B.P. Gerkey and M.J. Mataric, "A formal analysis and taxonomy of task allocation in multi-robot systems," *Int'l J. Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.
[23] Z. Butler and D. Rus, "Event-based motion control for mobile-sensor networks," *IEEE Pervasive Computing*, vol. 2, no. 4, pp. 34–42, 2003.
[24] P. Basu and J. Redi, "Movement control algorithms for realization of fault-tolerant ad hoc robot networks," *IEEE Network*, vol. 18, no. 4, pp. 36–44, 2004.
[25] J. Wu and S. Yang, "SMART: a scan-based movement-assisted sensor deployment method in wireless sensor networks," *Proc. IEEE INFOCOM*, pp. 2313–2324, 2005.
[26] Y. Zou and K. Chakrabarty, "Sensor deployment and target localization in distributed sensor networks," *ACM Trans. Embedded Computing Systems*, vol. 3, no. 1, pp. 61–91, 2004.
[27] N. Heo and P.K. Varshney, "Energy-efficient deployment of intelligent mobile sensor networks," *IEEE Trans. Systems, Man and Cybernetics—Part A: Systems and Humans*, vol. 35, no. 1, pp. 78–92, 2005.
[28] G. Wang, G. Cao, and T.F.L. Porta, "Movement-assisted sensor deployment," *IEEE Trans. Mobile Computing*, vol. 5, no. 6, pp. 640–652, 2006.
[29] Y.C. Wang, C.C. Hu, and Y.C. Tseng, "Efficient placement and dispatch of sensors in a wireless sensor network," *IEEE Trans. Mobile Computing*, vol. 7, no. 2, pp. 262–274, 2008.
[30] Y.C. Wang and Y.C. Tseng, "Distributed deployment schemes for mobile wireless sensor networks to ensure multilevel coverage," *IEEE Trans. Parallel and Distributed Systems*, vol. 19, no. 9, pp. 1280–1294, 2008.
[31] C. Sharp, S. Schaffert, A. Woo, N. Sastry, C. Karlof, S. Sastry, and D. Culler, "Design and implementation of a sensor network system for vehicle tracking and autonomous interception," *Proc. IEEE European Workshop on Wireless Sensor Networks*, pp. 93–107, 2005.
[32] M.D. Naish, E.A. Croft, and B. Benhabib, "Dynamic dispatching of coordinated sensors," *Proc. IEEE Int'l Conf. Systems, Man, and Cybernetics*, pp. 3318–3323, 2000.
[33] Y. Zou and K. Chakrabarty, "Distributed mobility management for target tracking in mobile sensor networks," *IEEE Trans. Mobile Computing*, vol. 6, no. 8, pp. 872–887, 2007.
[34] A. Verma, H. Sawant, and J. Tan, "Selection and navigation of mobile sensor nodes using a sensor network," *Pervasive and Mobile Computing*, vol. 2, no. 1, pp. 65–84, 2006.
[35] G. Wang, G. Cao, P. Berman, and T.F.L. Porta, "Bidding protocols for deploying mobile sensors," *IEEE Trans. Mobile Computing*, vol. 6, no. 5, pp. 515–528, 2007.
[36] G. Wang, G. Cao, T.F.L. Porta, and W. Zhang, "Sensor relocation in mobile sensor networks," *Proc. IEEE INFOCOM*, pp. 2302–2312, 2005.
[37] X. Li, N. Santoro, and I. Stojmenovic, "Localized distance-sensitive service discovery in wireless sensor and actor networks," *IEEE Trans. Computers*, vol. 58, no. 9, pp. 1275–1288, 2009.
[38] N. Bulusu, J. Heidemann, and D. Estrin, "GPS-less low-cost outdoor localization for very small devices," *IEEE Personal Comm.*, vol. 7, no. 5, pp. 28–34, 2000.
[39] M. Udi, *Introduction to Algorithms: A Creative Approach*, Addison-Wesley Publishing Company, 1989.
[40] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Academic Press, 2001.
[41] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms*, The MIT Press, 2001.
[42] M. Rahimi, H. Shah, G.S. Sukhatme, J. Heideman, and D. Estrin, "Studying the feasibility of energy harvesting in a mobile sensor network," *Proc. IEEE Int'l Conf. Robotics and Automation*, pp. 19–24, 2003.
[43] G.M. Dai, A.H. Du, Q.H. Li, and M.C. Wang, "Planning of moving path based on simplified terrain," *Proc. Int'l Conf. Machine Learning and Cybernetics*, pp. 1915–1918, 2003.
[44] Y.H. Liu and S. Arimoto, "Finding the shortest path of a disc among polygonal obstacles using a radius-independent graph," *IEEE Trans. Robots and Automation*, vol. 11, pp. 682–691, 1995.
[45] S.Q. Zheng, J.S. Lim, and S.S. Iyengar, "Finding obstacle-avoiding shortest paths using implicit connection graphs," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, pp. 103–110, 1996.