國立交通大學

資訊科學與工程研究所

碩 士 論 文

無線感測網路中行動感測器派遣演算法

Energy-Efficient Algorithms for Dispatching Mobile Sensors in a Wireless Sensor Network

研 究 生：張民憲

指導教授：彭文志　教授

中 華 民 國　九 十 五　年 六 月

無線感測網路中行動感測器派遣演算法

Energy-Efficient Algorithms for Dispatching Mobile Sensors in a Wireless Sensor Network

研 究 生：張民憲　　Student：Min-Hsien Chang

指導教授：彭文志　　Advisor：Wen-Chih Peng

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年六月

無線感測網路中行動感測器派遣演算法

學生：張民憲　　指導教授：彭文志

國立交通大學資訊科學與工程研究所碩士班

摘　　要

在一混合式之無線感測網路中，固定式感測器負責偵測事件的發生，而行動感測器可以移動至事件發生點以便作進一步分析。行動感測器最耗能的操作即為移動，如何減少感測器之移動距離同時能讓其完成任務為一富有挑戰性之研究題目。若某些特定行動感測器一直被指派到不同的地點，他們的能量很快就會耗盡，並造成殘存感測器之工作負擔變大。在平衡各感測器之工作負擔的前提之下，本論文提出一有效派遣行動感測器之演算法，並且適用於任意數量之可動式感測器以及事件發生點。行動感測器的數量大於事件發生點時，本論文將其轉化為一最大配對數之配對問題(maximum matching problem)，而在事件發生點大於感測器的數量時，我們將事件先叢集化，並將行動感測器分配到各叢集完成任務。為了減少感測器之間的訊息傳輸量，本論文提出一分散式演算法。模擬實驗結果證明本論文所提出之演算法能夠有效的延長行動感測器之系統生命週期(system lifetime)。

Energy-Efficient Algorithms for Dispatching Mobile Sensors in a Wireless Sensor Network

Student：Min-Hsien Chang　　Advisor：Dr. Wen-Chih Peng

Institute of Computer Science
National Chiao Tung University

ABSTRACT

A hybrid sensor network consists of both static and mobile sensors, where the former is used to detect events while the latter can move to event locations for conducting more advanced analysis. By exploring the load balance of mobile sensors, we propose an algorithm *CentralSD* to efficiently dispatch mobile sensors. Our algorithm is general in which the numbers of mobile sensors and events are arbitrary. When the number of events is no larger than that of mobile sensors, we transform the dispatch problem to a maximum matching problem in a weighted bipartite graph. When there are only few mobile sensors to be dispatched to a large number of event locations, we propose an efficient clustering scheme to group event locations so that the maximum matching approach can be applied. To reduce messages incurred, we also develop a distributed algorithm *GridSD*. Extensive simulation results are presented to verify the effectiveness of our proposed algorithms.

誌　　謝

漫漫長路，這篇能夠呈獻出我這兩年來研究成果的碩士論文終於問世，在研究所的這段日子裡，有太多人與事都值得感謝，回憶起那些奮鬥到天明，掙扎著從實驗室騎摩托車回到宿舍馬上倒地不起的日子，現在倒是能夠微微地揚起嘴角懷念著那努力的過程。

首先要感謝指導教授彭文志老師，老師順利的引領我轉換到本論文的題目，並且在這個題目裡給我很多指導以及建議，如此才能有這篇論文的誕生。還需要感謝王友群學長在論文後期給予我的指正以及幫助，讓我對於自己的題目能有更深入且多面向的理解，洪智傑學長提供了我許多對於數學證明以及表示法的建議，而且常常充滿活力的鼓舞我們一度低迷的士氣。

實驗室的革命同志們：慧友、志劭以及向彥，有了你們各種方式的鼓勵與幫助我才能夠順利完成論文，回憶起大家一起吃飯一起打嘴炮的日子，讓我更能堅定的往人生的下段旅程前進，實驗室的各位學弟妹也提供我更廣闊的思維以及生活經驗。

最後我將感謝我的家人，在新竹的姊姊常常把我拉去吃飯，讓我免於吃泡麵過活的日子，在台中家裡的奶奶和爸媽的支持讓我全心全意投入本篇論文，你們的支持與鼓勵是本篇論文最重要的推手。感謝你們!

目　　錄

頁次

圖　目　錄

Chapter 1

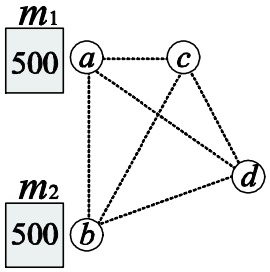
Introduction

Recent advances in micro-sensing MEMS and wireless communication technologies have promoted the development of *wireless sensor networks* (*WSN*). A WSN has many attractive characteristics including context-aware capability and fast ad-hoc networking configuration, so that it can be widely used in various applications such as border detection, environment monitoring, smart home, and surveillance. However, sensor nodes in a WSN are usually assumed to be simple and prone to error [1]. They may provide rough descriptions of events and even false information when sensors are broken. For example, in a military application, sensors that can detect the change of pressure may be deployed along the borders to determine if an enemy passes. However, these sensors can only report something passing but cannot describe what passes through them. Moreover, failure sensors or incorrect sensing reports will generate false alarms. In this case, we may prefer using more powerful but expensive sensors like cameras to recognize the passing object or to determine whether this is a false alarm. However, it is impossible to mount a camera on each sensor node due to their large number. An alternative and better way is to mount the expensive sensors on few mobile platforms [6, 11, 16], and then dispatch these *mobile sensors* to visit event locations when necessary.

In this paper, we consider a hybrid WSN consisting of static and mobile sensors. The static sensors are deployed to detect events, while the mobile sensors equipped with more resources such as sensing capability and computation power are dispatched to these event locations to conduct more

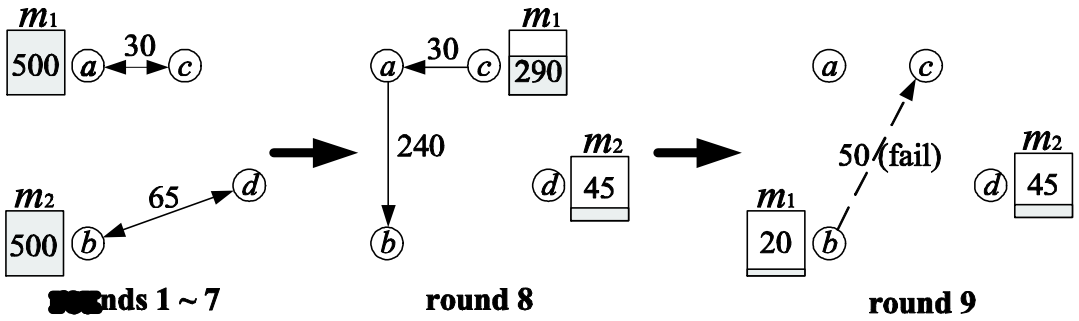
advanced analysis. Since mobile sensors use small batteries for their operations without any additional power source, one important issue of mobile sensors is to conserve their energies. In general, for a mobile sensor, the energy consumption of movement is larger than that of other operations such as sensing, computing and communication. The moving energy cost will be the dominated factor of the energy consumption of a mobile sensor. Therefore, in this paper, we investigate how to efficiently dispatch mobile sensors to visit event locations with the purpose of maximizing the *system lifetime*, which is defined as the time duration until there are some event locations that cannot be reached by any mobile sensor due to lack of energy.

As mentioned above, the problem we shall deal with is to dispatch mobile sensors in an energy-efficient manner so as to prolong the system lifetime. However, simply optimizing the solution in each one-round dispatch cannot guarantee to maximize the system lifetime. Specifically, we schedule each mobile sensor to visit certain of event locations in a way such that the total moving energy is minimized. Such procedure is repeated in each round until we cannot find any mobile sensor with enough energy to reach some event locations. Unfortunately, this iteratively-optimized method may cause some mobile sensors early to exhaust their energies, thereby reducing the system lifetime. Consider an illustrative example in Fig. 1.1, where there are two mobile sensors m_1 and m_2 at locations a and b, respectively. Both mobile sensors have an initial energy of 500 units. We consider only the energy consumption due to movement. Assume that two events occur at locations c and d (resp., a and b) during each odd (resp., even) round. Fig. 1.1(b) shows the execution of the above iteratively-optimized method. To minimize the total moving energy, m_1 and m_2 are assigned to move between the pair of locations (a, c) and (b, d), respectively, which results in a minimum cost of 95 units in each round. However, after seven rounds, m_2 almost runs out of its energy and has to stay at location d. As a result, m_1 should visit both event locations a and b and thus remains 20 units of energy in the eighth round. Finally, in the ninth round, no mobile sensor has enough energy to reach the event location c so that the system lifetime is eight rounds. In fact, when it comes to dispatching mobile sensors, we should not only attempt to reduce the total moving energy



cost	*a*	*b*	*c*	*d*
a	0	240	30	50
b	240	0	50	65
c	30	50	0	240
d	50	65	240	0

(a) the shortest paths and energy costs to reach event locations



(b) minimizing the total moving energy in each round

(c) balancing the loads of m_1 and m_2 in each round

Figure 1.1: Comparison of different dispatch methods.

but also *balance* the loads of mobile sensors. Fig. 1.1(c) illustrates the above method, where m_1 and m_2 are assigned to move between the pair of locations (a, d) and (b, c), respectively, resulting in a slightly higher cost (i.e., 100 units) in each round. Although spending more energy compared with the iteratively-optimized method in each round, the above method indeed extends the system lifetime (i.e., ten rounds). From Fig. 1.1, we can observe that simply optimizing the solution in each one-round dispatch could shorten the system lifetime because the early-exhausted mobile sensors will burden other still alive ones, which in turn early exhausts energy of alive mobile sensors. As the number of early-exhausted mobile sensors increases, the system lifetime will be shortened.

Consequently, in this paper, by balancing the loads of mobile sensors, we propose an algorithm *CentralSD* (standing for *Centralized Sensor Dispatching*) to dispatch mobile sensors with the

purpose of maximizing the system lifetime. Assume that one central server is available to collect location of mobile sensors and events. In CentralSD, during each round of dispatching, we will determine which mobile sensors should visit which event locations with the objective of minimizing the moving distance (or maximizing the remaining energy) of mobile sensors. However, instead of simply optimizing the objective of dispatching, we exploit the load balance of mobile sensors, in terms of moving distance or remaining energy, during each round. Furthermore, balancing the loads of moving distance also implies that mobile sensors have similar energy costs and moving time to visit event locations, while balancing the loads of remaining energy can avoid a mobile sensor with less energy being dispatched to a long-distance event location. In CentralSD, a more general solution to the sensor dispatch problem is proposed in which the numbers of mobile sensors and event locations are arbitrary. In particular, two cases are considered in CentralSD. Explicitly, when the number of event locations is no larger than that of mobile sensors, we transform the dispatch problem to a maximum matching problem in a weighted bipartite graph, where the vertex set contains all mobile sensors and all event locations and the edge set contains the edge from each mobile sensor to each event location. However, instead of finding the matching with a minimum edge weight [12], we use a *preference list* and a *bound* to select the matching. Specifically, the preference list helps assign an event location with a suitable mobile sensor, while the bound is to avoid selecting edges with extreme weights so that the load-balance can be achieved. When the number of event locations is larger than that of mobile sensors, we propose an efficient clustering scheme to group event locations into clusters, where the number of clusters will be the same as that of mobile sensors. In this way, we can adopt the aforementioned matching approach to dispatch each mobile sensor to a cluster of event locations. Then the mobile sensor can use the traveling-salesman approximation algorithm [3] to reach all event locations in that cluster. However, as mentioned above, CentralSD requires a central server to collect information of mobile sensors and events, which incurs a considerable amount of message transmissions. To reduce messages incurred, we develop an algorithm *GridSD* (standing for *Grid-based architecture for Sensor Dispatching*) to dispatch mobile sensors in

a distributed manner. A comprehensive performance study is conducted and simulation results show that by exploring the load balancing of mobile sensors, our proposed algorithms can have more alive mobile sensors, thereby prolonging the system lifetime.

A significant amount of research [2, 8, 19, 20, 21] has elaborated on the issue of using mobile sensors to enhance the sensing coverage or the network connectivity in a WSN. We mention in passing that the authors in [5] considers how to move more sensors close to the locations of events predicted, while still maintaining complete coverage of the sensing field. However, the concept of sensor dispatch is still not addressed. The authors in [14] exploit the use of mobile sensors to track a moving object with mobile sensors. It is assumed that the object's trajectory can be predicted and the work discusses how to maneuver the mobile sensors to optimally acquire data from the object in real-time. However, the energy consumption of mobile sensors is not considered in [14]. Several research efforts [17, 18] have addressed how to select and move mobile sensors in a hybrid WSN. In [17], static sensors that detect events will invite and navigate nearby mobile sensors to move to their locations. The mobile sensor that have shorter moving distance and more energy, and whose leaving does not cause a large coverage hole, will be invited by the static sensors. In [18], static sensors estimate the coverage holes close to them and use the hole size to compete for mobile sensors. The mobile sensor then selects the largest one and moves to fill that coverage hole. To the best of our knowledge, the attention of prior works was mainly paid to the use of mobile sensors for coverage holes and object tracking, but not to the general dispatch problem explored in this paper.

In this work, we consider how to efficiently dispatch mobile sensors so that the system lifetime can be extended. In particular, during each one-round dispatch, we will schedule mobile sensors to visit event locations so that the moving distance or the remaining energy of mobile sensors can be minimized or maximized, respectively. However, instead of simply optimizing the solution, we consider to balance the loads of mobile sensors, in terms of moving distance or remaining energy, during each round. By balancing the loads can make each mobile sensor live longer, and thus can help prolong the system lifetime. In addition, balancing the loads of moving distance can make

mobile sensors have similar energy costs and moving time to visit event locations, while balancing the loads of remaining energy can avoid a mobile sensor with less energy being dispatched to a long-distance event location. In this work, we consider a more general solution to the sensor dispatch problem in which we do not restrict the numbers of mobile sensors and event locations. When the number of event locations is no larger than that of mobile sensors, we translate the dispatch problem to the problem of finding a maximum matching in a weighted bipartite graph, where the vertex set contains both the sets of mobile sensors and event locations, and the edge weights can be either the moving distance or the remaining energy, depending on the objective function being used. However, instead of finding the matching with a maximum or minimum edge weight [12], we use a *preference list* and a *bound* to select the matching. Specifically, the preference list can help each mobile sensor select a suitable event location so that it can have a shorter moving distance or remain more energy after movement, while the bound is to avoid selecting edges with extreme weights so that the load-balance can be achieved. When the number of event locations is larger than that of mobile sensors, we propose an efficient clustering mechanism to group event locations into clusters, where the number of clusters will be the same as that of mobile sensors. In this way, we can adopt the aforementioned solution to dispatch mobile sensors to the clusters of event locations. After we assign a mobile sensor to visit a cluster, it can use the traveling-salesman approximation algorithm [3] to reach all event locations in that cluster. Finally, we have also developed a distributed method based on the aforementioned dispatch solution to avoid the need of a centralized server when dispatching mobile sensors. In summary, our work has the following contributions:

1. We have addressed the importance of load-balance when dispatching mobile sensors and then propose a novel dispatch solution to prolong the system lifetime by balancing the loads of mobile sensors in terms of moving distance and remaining energy.

2. We have proposed an efficient clustering mechanism to group event locations so that our dispatch solution can be also suitably applied to the case when there are few mobile sensors that

have to be dispatched to a large number of event locations.

3. We have proposed a distributed version of our dispatch solution so that there is no need of a centralized server to execute our dispatch solution.

It is worth mentioning that we not only explore load-balancing in dispatching mobile sensors, but also develop one clustering mechanism to deal with the case that the number of event locations is larger than that of mobile sensors. These features distinguish this paper from others.

The rest of this paper is organized as follows. In Chapter 2, we formally define the sensor dispatch problem. Chapter 3 proposes our solutions to this problem. Simulation results are presented in Chapter 4. Chapter 5 concludes this paper.

Chapter 2

Sensor Dispatch Problem

We consider a hybrid WSN comprised of both static and mobile sensors. Static sensors can form a connected network and fully cover the area of interest to continuously monitor the environment. When events are detected by static sensors, there is a set of n mobile sensors $S = \{s_1, s_2, \ldots, s_n\}$, which are randomly distributed over the sensing region and can be dispatched to the event locations (as reported by static sensors) to provide higher quality of sensing results. To achieve this goal, we assume that sensors know their own locations, which can be achieved by the global positioning system (GPS) [9] or other localization techniques [4, 10].

The sensor dispatch problem is modeled as follows. We assume that there is a set of event locations $L = \{l_1, l_2, \ldots, l_m\}$, each to be visited by a mobile sensor. We allow an arbitrary relationship between the values of m and n. The goal is to compute a dispatch schedule SCH_{s_i} for each mobile sensor s_i such that each location in L is visited exactly once by one mobile sensor. Each schedule SCH_{s_i} is denoted by a sequence of event locations, and the jth location to be visited is written as $SCH_{s_i}[j]$. Let e_i be the current energy of s_i and $c(SCH_{s_i})$ be the energy required to complete s_i's visit schedule,

$$c(SCH_{s_i}) = \Delta_{move} \times (d(s_i, SCH_{s_i}[1]) + \sum_{j=1}^{|SCH_{s_i}|-1} d(SCH_{s_i}[j], SCH_{s_i}[j+1])),$$

where Δ_{move} is the energy required to move a sensor one-unit distance, $d(s_i, SCH_{s_i}[1])$ is the Euclidean distance from s_i's current location to $SCH_{s_i}[1]$, and $d(SCH_{s_i}[j], SCH_{s_i}[j+1])$ is the Euclidean distance between $SCH_{s_i}[j]$ and $SCH_{s_i}[j+1]$. Clearly, the schedule must satisfy $e_i \geq c(SCH_{s_i})$.

For performance reason, we define two objective functions for the sensor dispatch problem. The first one is to minimize the total energy consumption to move sensors, i.e.,

$$\min \sum_{s_i \in S} c(SCH_{s_i}). \tag{2.1}$$

The second one is to maximize the total remaining energy of mobile sensors after movement, i.e.,

$$\max \sum_{s_i \in S} (e_i - c(SCH_{s_i})). \tag{2.2}$$

To balance the energy consumption of mobile sensors, we will also measure the standard deviations of sensors' energy consumption and remaining energies. When we assign the dispatching schedules of mobile sensors in each round. However, as mentioned in Chapter 1, we should also balance the loads of mobile sensors. In this work, we use the *standard deviation* to measure how balance a dispatching method can achieve. In particular, the standard deviation of the energy consumption among mobile sensors is

$$\left(\frac{1}{n} \sum_{s_i \in S} (c_{avg} - c(SCH_{s_i}))^2 \right)^{\frac{1}{2}}, \tag{2.3}$$

and the standard derivation of the remaining energy among mobile sensors is

$$\left(\frac{1}{n} \sum_{s_i \in S} (e_{avg} - (e_{s_i} - c(SCH_{s_i})))^2 \right)^{\frac{1}{2}}, \tag{2.4}$$

where c_{avg} and e_{avg} are the average of the total energy consumption and the total remaining energy of mobile sensors, respectively. To balance the loads of mobile sensors, we should also minimize either Eq. (2.3) or Eq. (2.4) when assigning their dispatching schedules, depending on the objective function Eq. (2.1) or Eq. (2.2) that we adopt in each round, respectively.

Note that the above modeling is only concerned about one round of sensors' dispatch schedules. In general, multiple rounds of dispatch schedules need to be determined, where each round contains

those events being detected over a fixed amount of time, and the goal is to extend mobile sensors' lifetimes to the maximum number of rounds. The length of a round depends on users' real-time constraint. Since event locations are unexpected, we will only focus on the optimization of one round in our work.

Chapter 3

Algorithms for Dispatching Mobile Sensors

We first propose a centralized solution, where there is a central server which will collect the sets L and S and compute sensors' schedules. Then, a distributed solution will be developed.

3.1 Algorithm CentralSD: A Centralized Dispatch Method

When $|S| \geq |L|$, we will transform the dispatch problem to a maximum matching problem in a weighted bipartite graph. When $|S| < |L|$, we partition L into $|S|$ clusters such that each mobile sensor only needs to visit one cluster of event locations. Then a maximum matching approach will be applied again.

3.1.1 Case of $|S| \geq |L|$

For each mobile sensor $s_i \in S$, we first determine the energy cost $c(s_i, l_j)$ to move s_i to each event location $l_j \in L$. We define a cost function $c(s_i, l_j) = \Delta_{move} \times d(s_i, l_j)$. We then construct a weighted complete bipartite graph $G = (S \cup L, S \times L)$ such that the vertex set contains all mobile sensors and all event locations and the edge set contains the edge (s_i, l_j) from each $s_i \in S$ to each $l_j \in L$. The weight of (s_i, l_j) is defined as

$$w(s_i, l_j) = c(s_i, l_j),$$

if Eq. (2.1) is the objective function, or as

$$w(s_i, l_j) = \begin{cases} e_{max} - (e_{s_i} - c(s_i, l_j)), & \text{if } e_{s_i} \geq c(s_i, l_j) \\ 0, & \text{otherwise} \end{cases},$$

if Eq. (2.2) is the objective function, where e_{max} is a large value no less than $\max_{s_i \in S}\{e_{s_i}\}$. For simplicity, we can set $e_{max} = \max_{s_i \in S}\{e_{s_i}\}$. It is not hard to see that the objective functions Eqs. (2.1) and (2.2) can both be reduced to the same goal of minimizing the total weight of a matching. Therefore, with G, our goal is to find a matching P such that (1) the number of matches in P is the largest, (2) the total edge weight of P is minimized, and (3) the standard deviation of edge weights of P is minimized.

To find P, we first associate a *preference list* $Plist(s_i)$ to each s_i, which ranks each event location $l_j \in L$ by its weight $w(s_i, l_j)$ in an increasing order. When edge weights are equal, events' IDs are used to break the tie. Similarly, for each l_j, we associate it with a preference list $Plist(l_j)$, which ranks each $s_i \in S$ by its weight $w(s_i, l_j)$ in an increasing order.

To reduce the standard deviation of the matching P, we introduce a *bound* B_{l_j} for each $l_j \in L$ to restrict the candidate mobile sensors with which l_j can match. Specifically, l_j can only consider a mobile sensor s_i such that $w(s_i, l_j) \leq B_{l_j}$.

To find P, we set the initial value of each B_{l_j} to the average of the minimum of the weights of all edges incident to each event location, i.e.,

$$B_{l_1} = \cdots = B_{l_m} = \frac{\sum_{j=1}^{m} \min_{(s_i, l_j) \in S \times L}\{w(s_i, l_j)\}}{m}.$$

Then, for each $l_j \in L$, we try to find a match $s_i \in Plist(l_j)$ with l_j such that $w(s_i, l_j)$ is minimized and $w(s_i, l_j) \leq B_{l_j}$. If we cannot find such s_i, we will continue extending B_{l_j} with an increasing level Δ_B until l_j can find available mobile sensors. Note that the value of Δ_B should be carefully designed so that the weight of each pair will not increase sharply while the number of extending the bound operations for the next available mobile sensor can be reduced. In view of the above design guide, for each event, we should derive the distance interval of the farthest mobile sensor

and the nearest mobile sensor. Those mobile sensors staying in the distance interval should be taken into consideration for dispatching. Furthermore, when more mobile sensors are given, an event can easily find a mobile sensor in its neighborhood. Otherwise, one should use larger increasing level to increase the possibility of finding available mobile sensors. Therefore, the increasing level, denoted as Δ_B, is formulated as follows:

$$\Delta_B = \frac{\delta}{mn} \times (\sum_{j=1}^{m} \max_{(s_i,l_j)\in S\times L} \{w(s_i,l_j)\} - \sum_{j=1}^{m} \min_{(s_i,l_j)\in S\times L} \{w(s_i,l_j)\}), \tag{3.1}$$

where δ is an adjustable coefficient Δ_B.

As an unmatched event location l_j expands its bound B_{l_j}, more candidates will be included in its $Plist(l_j)$. If the first unvisited candidate s_i in $Plist(l_j)$ is also unmatched, the pair (s_i, l_j) is added into P. Otherwise, s_i must be matched with another event location (e.g., l_o). From the bounds B_{l_j} and B_{l_o}, we can determine which event location s_i should be matched. This is referred to as a competition between l_j and l_o. In particular, s_i is matched to l_j if one of the following cases is satisfied.

1. $B_{l_j} > B_{l_o}$. Since enlarging the bound will increase the risk of including an edge with an extreme weight into P, we will prefer matching s_i with l_j to avoid expanding the larger bound B_{l_j}.

2. $B_{l_j} = B_{l_o}$ and l_j is prior to l_o in s_i's preference list. As s_i prefers l_j to l_o, we thus match s_i with l_j to reduce the total weight of P.

3. $B_{l_j} = B_{l_o}$ and s_i is the last candidate in $Plist(l_j)$ but not in $Plist(l_o)$. Since l_j cannot have another candidate to pick in $Plist(l_j)$, s_i should be matched with l_j.

Once s_i is matched with l_j, the pair (s_i, l_o) should be replaced by the new pair (s_i, l_j) in P, and l_o should search for another mobile sensor to match with. It is possible that l_o will compete with other event locations for mobile sensors.

Procedure 1 PairMatching(L, S)

Input: sets of event locations L and mobile sensors S

Output: maximum matching P between L and S

1: construct a weighted bipartite graph $G = (S \cup L, S \times L)$;

2: generate a preference list $Plist(k)$ for each $k \in \{S \cup L\}$;

3: assign an initial value for each bound B_{l_j};

4: **for** each $l_j \in L$ that is not matched **do**

5: **while** $Plist(l_j)$ contains no unvisited candidate **do**

6: $B_{l_j} = B_{l_j} + \Delta_B$;

7: **end while**

8: let $s_i \in S$ be the first unvisited candidate in $Plist(l_j)$;

9: **if** s_i is unmatched **then**

10: add the pair (s_i, l_j) to P;

11: **else**

12: let $l_o \in L$ be the location originally matched with s_i;

13: **if** $B_{l_j} > B_{l_o}$ **then**

14: replace (s_i, l_o) by (s_i, l_j) in P;

15: **else if** $B_{l_j} = B_{l_o}$ **then**

16: **if** l_j is prior to l_o in $Plist(s_i)$ **then**

17: replace (s_i, l_o) by (s_i, l_j);

18: **else if** s_i is the last candidate in $Plist(l_j)$ but not in $Plist(l_o)$ **then**

19: replace (s_i, l_o) by (s_i, l_j) in P;

20: **end for**

Consider an illustrative example in Fig. 3.1 , where δ is set to 2. The initial bound is $\frac{101+77+92}{3} = 90$ and the increasing level $\Delta_B = \frac{(213+234+229)-(101+77+92)}{4\times3} \times 2 = 67.7$. We start with the event location l_1. Since there is no available mobile sensor in $Plist(l_1)$ with the initial bound, B_{l_1} will be expanded by Δ_B. Accordingly, B_{l_1} is updated to 90+67.7=157.7. As a result, we will have three candidates (i.e., s_1, s_2, and s_3). Since l_1 finds that the first unvisited candidate s_1 is unmatched, we add (s_1, l_1) to the matching P. Following the same operation, the pair (s_3, l_2) is determined, as shown in Fig. 3.1(b). However, after expanding B_{l_3}, l_3 finds that the first candidate s_1 has been matched with l_1. Therefore, l_3 will compete with l_1 for s_1. It can be verified that case 2 is satisfied (i.e., $B_{l_3} = B_{l_1} = 157.7$ and l_3 is prior to l_1 in $Plist(s_1)$). Consequently, (s_1, l_1) is replaced by (s_1, l_3) in Fig. 3.1(c). Following the same procedure, l_1 will obtain s_3 from l_2 and then l_2 has to find an unmatched mobile sensor s_4 to pair with. Fig. 3.1(e) shows the final result.

In the above example, we follow the sequence of l_1, l_2, and l_3 to decide the matching. Note that due to the competition of event locations, the PairMatching procedure can generate the same result no matter what processing sequence we use. This will be proved in Theorem 1. Theorem 2 analyzes the time complexity of PairMatching.

Theorem 1. *The result of PairMatching is irrelevant to the processing sequence of event locations.*

Proof. Suppose that PairMatching generates two different matchings P and P' when different processing sequences are used. Then there must be an event location $l_x \in L$ matched with different mobile sensors s_a and s_b in P and P', respectively. Without loss of generality, we assume that $w(l_x, s_b) < w(l_x, s_a)$, so s_b will be prior to s_a in $Plist(l_x)$. In the matching P, since l_x has been matched with s_a, s_b must be matched with another event location (e.g., l_y). Thus, we have $\{(l_x, s_a), (l_y, s_b)\} \subseteq P$. Recall the three cases that l_y can win the competition to get s_b. Then exactly one of the following two conditions must be satisfied:

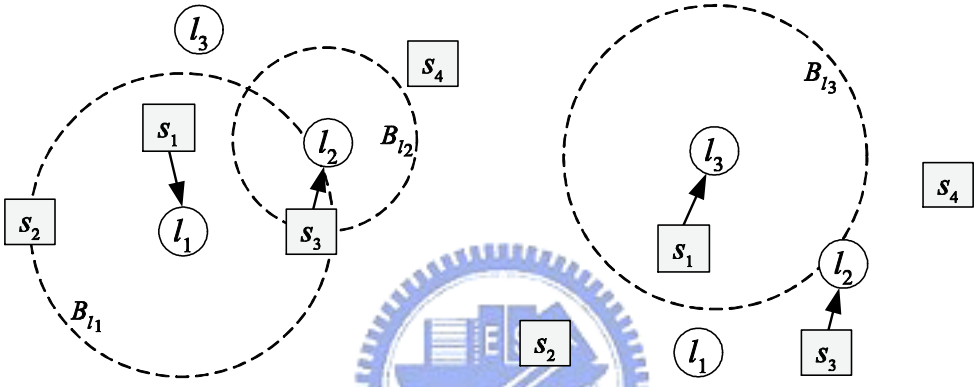
1. l_y has the advantage over l_x because either $B_{l_y} > B_{l_x}$ (i.e., case 1) or s_b is the last candidate with B_{l_y} (i.e., case 3).

cost	s_1	s_2	s_3	s_4
l_1	101	152	131	213
l_2	142	234	67	101
l_3	92	175	229	179

$Plist(l_1) = \{s_1, s_3, s_2, s_4\}$
$Plist(l_2) = \{s_3, s_4, s_1, s_2\}$
$Plist(l_3) = \{s_1, s_2, s_4, s_3\}$

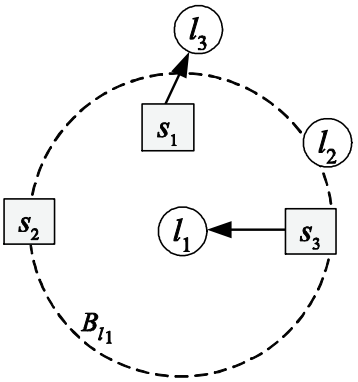
$Plist(s_1) = \{l_3, l_1, l_2\}$
$Plist(s_2) = \{l_1, l_3, l_2\}$
$Plist(s_3) = \{l_2, l_1, l_3\}$
$Plist(s_4) = \{l_2, l_3, l_1\}$

(a) energy costs and preference lists

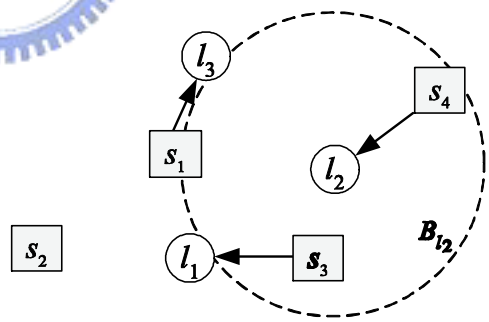


(b) $P = \{(s_1, l_1), (s_3, l_2)\}$

(c) $P = \{(s_3, l_2), (s_1, l_3)\}$



(d) $P = \{(s_3, l_1), (s_1, l_3)\}$



(e) $P = \{(s_3, l_1), (s_4, l_2), (s_1, l_3)\}$

Figure 3.1: An example to show how PairMatching works.

2. l_y is prior to l_x in $Plist(s_b)$ (i.e., case 2).

However, in the matching P', since s_b is matched with l_x, l_y will be matched with another mobile sensor (e.g., s_c). As such, we have $\{(l_x, s_b), (l_y, s_c)\} \subseteq P'$. In this case, if the first condition is satisfied, a contradiction will be occurred because l_y must be also matched with s_c in the matching P. If the second condition is satisfied, we have $B_{l_x} = B_{l_y}$. However, since l_y is prior to l_x in $Plist(s_b)$, s_b must be matched with l_y rather than with l_x in P', which also causes a contradiction. Therefore, the result of PairMatching is unique, no matter what processing sequence of event locations we use. □

Theorem 2. *The time complexity of PairMatching is* $O(mn \lg mn)$.

Proof. In PairMatching, we first construct the graph G, which takes $O(mn)$ time since we have to assign the weight of each edge. Then generating the preference lists for all elements in $S \cup L$ needs $O(mn \lg n + nm \lg m)$ time since we have to sort the elements in S and L. The worst case for an event location to find its matched mobile sensor is $O(n)$ since it has to go through its preference list. Thus, the time to compute the maximum matching P will be $O(mn)$. Therefore, the total time complexity of PairMatching is $O(mn + mn \lg n + nm \lg m + mn) = O(mn \lg mn)$. □

3.1.2 Case of $|S| < |L|$

When the number of event locations is larger than that of mobile sensors, we can group those event locations whose distance are close to each other into a cluster. This can be achieved by using K-means [7]. Consequently, in light of PairMatching, each mobile sensor is dispatched to one cluster and then travels the event locations within the assigned cluster. To facilitate the presentation of this paper, we briefly describe how K-means works. In K-means, event locations are randomly partitioned into n non-empty clusters. Then for each cluster, we determine the mean from the event locations assigned to the same cluster. Then, each event location should join the cluster whose mean is the closest one to it. After all event locations decide their corresponding clusters, we should re-

calculate the mean for each cluster. The above procedure is repeated until no event relocation is needed.

To evaluate the energy cost of the clustering result, the cost $\phi(\hat{c}_k)$ of each cluster $\hat{c}_k$ is formulated as the total edge weight of the minimum spanning tree in that cluster, where the weight of an edge (l_i, l_j) is the Euclidean distance of the two event locations l_i and l_j. For example, in Fig. 3.2(a), $\phi(A) = 50$, $\phi(B) = 12$, $\phi(C) = 15$, and $\phi(D) = 68$. Unfortunately, K-means cannot guarantee to minimize the total cost of the clusters derived, especially when some "sparse" event locations are far away from others. In this case, K-means will group these sparse event locations into the same cluster, thereby increasing the total cost of clusters. Consider an example in Fig. 3.2(a), where four clusters are determined by K-means. Since both clusters A and D consist of sparse event locations (i.e., l_1 and l_{10}), the total cost of clusters is thus increased. By properly splitting and merging some clusters, we could adjust the result of K-mean so as to reduce the total cost of clusters. Intuitively, those clusters containing sparse event locations should be split. However, in order not to change the number of clusters, we have to merge two clusters when splitting a large one. In particular, let *max_intra_cost* be the maximum edge weight among edges in all clusters and *min_inter_cost* be the minimum distance of two clusters, where the distance between two clusters $\hat{c}_a$ and $\hat{c}_b$ is the Euclidean distance of the two closest event locations l_i and l_j, where $l_i \in \hat{c}_a$ and $l_j \in \hat{c}_b$. If max_intra_cost is larger than min_inter_cost, we can split the cluster with the edge whose weight is max_intra_cost (by removing that edge) and then merge two clusters whose distance is min_inter_cost (by connecting them with the shortest edge). We can repeat the above procedure until max_intra_cost is not larger than min_inter_cost. In this way, we can avoid some clusters having too large costs and thus reduce the total cost of clusters. Fig. 3.2 illustrates an example. In Fig. 3.2(a), max_intra_cost is 50 (in cluster D) and min_inter_cost is 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 into the same one, as shown in Fig. 3.2(b). Similarly, we can further split cluster A and then merge clusters C and D_1 to reduce the total cost of clusters. The final result will be shown in Fig. 3.2(c).

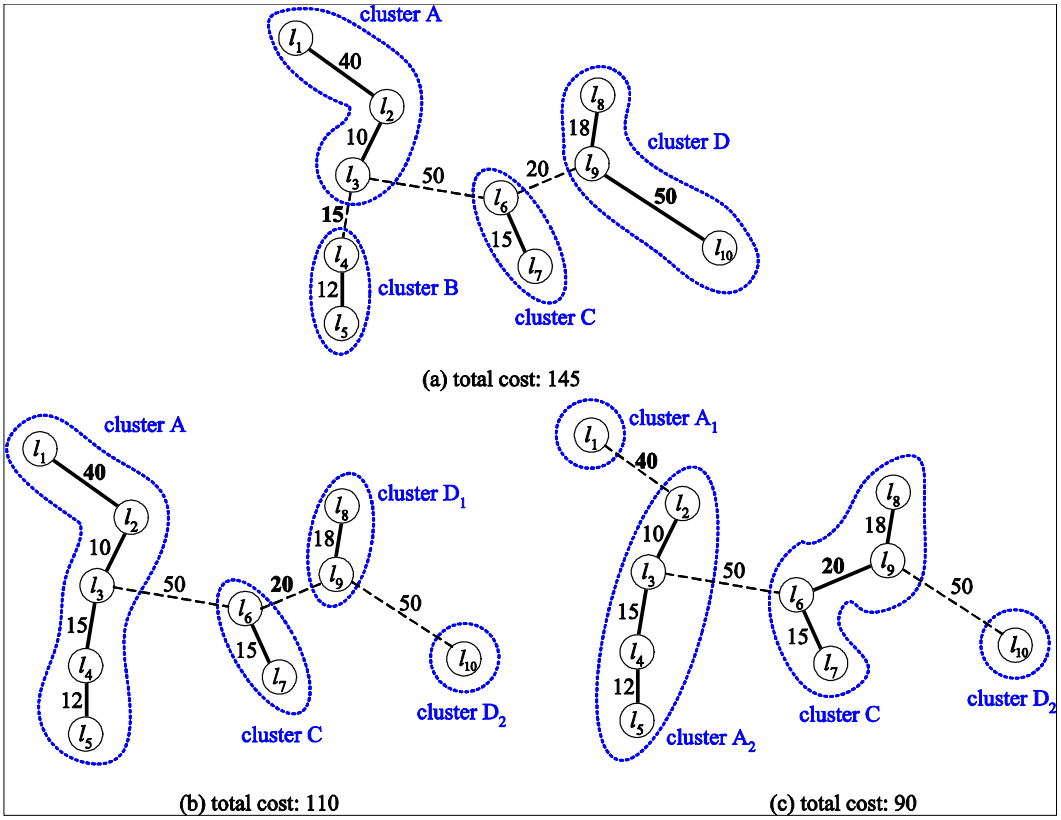


Figure 3.2: An example to cluster event locations.

After grouping event locations into n clusters $\hat{C} = \{\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_n\}$, we can use PairMatching to dispatch mobile sensors to these clusters. To assign edge weights of the graph $G = (S \cup \hat{C}, S \times \hat{C})$, the energy cost function should be re-formulated as follows:

$$c(s_i, \hat{c}_k) = \Delta_{move} \times (d(s_i, l_j) + \phi(\hat{c}_k)), \forall s_i \in S \text{ and } \hat{c}_k \in \hat{C},$$

where $l_j \in \hat{c}_k$ is the closest event location to s_i. Specifically, the total energy energy consumption for s_i to visit $\hat{c}_k$ includes the energy to move to the closest event location l_j in $\hat{c}_k$ and the energy to reach all event locations in $\hat{c}_k$. Once the graph G is constructed, we can adopt PairMatching to decide which mobile sensor should be dispatched which cluster. When s_i is dispatched to a cluster $\hat{c}_k$, it first moves to the closest event location l_j in $\hat{c}_k$ and then exploits the solution of the *traveling salesman problem* (*TSP*) [3] to reach other event locations with a minimum cost.

Algorithm 2 CentralSD

Input: sets of event locations L and mobile sensors S

Output: dispatch schedules $\{SCH_{s_1}, SCH_{s_2}, \ldots, SCH_{s_n}\}$

1: **if** $|S| \geq |L|$ **then**

2: P = PairMatching(L, S);

3: **while** $(s_i, l_j) \in P$ **do**

4: $SCH_{s_i} = \{l_j\}$;

5: **end while**

6: **else** /* event locations are more than mobile sensors */

7: group locations in L into n clusters by K-means;

8: **repeat**

9: compute max_intra_cost and min_inter_cost;

10: **if** max_intra_cost $>$ min_inter_cost **then**

11: split the cluster with max_intra_cost;

12: merge the two clusters with min_inter_cost;

13: **until** max_intra_cost $\leq$ min_inter_cost

14: let $\hat{C} = \{\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_n\}$ be the set of clusters;

15: P = PairMatching($\hat{C}$, S);

16: **while** $(s_i, \hat{c}_k) \in P$ **do**

17: travel all locations in $\hat{c}_k$ by TSP and add these locations into SCH_{s_i} in sequence;

18: **end while**

3.2 Algorithm GridSD: A Distributed Dispatch Method

In CentralSD, a central server is needed to collect the information of mobile sensors and events, which results in a large amount of message transmissions. To reduce the messages incurred, we propose a distributed algorithm GridSD that explores a grid-based architecture, in which each grid head will collect the information of mobile sensors and events and performs CentralSD locally. Therefore, both the computation complexity and message transmissions can be greatly reduced.

In GridSD, the sensing field is divided into grids, where each grid is an $\alpha \times \alpha$ square, as shown in Fig. 3.3. For each grid, we select a *grid head* [13] to collect the information such as the numbers of mobile sensors and event locations within its territory. Specifically, each mobile sensor will inform its location and remaining energy to its corresponding grid head. When detecting events, static sensors will report to their grid head. As pointed out earlier, once obtaining such information, a grid head will perform CentralSD to dispatch mobile sensors to those events occurred in its grid. However, if no mobile sensors available in this grid, the grid head will search available mobile sensors in other grids.

To reduce the number of message transmissions when a grid head searches for mobile sensors in other grids, we propose a modified approach of the *grid-quorum* [20]. Specifically, each grid head will send *advertisement* (ADV) messages containing the number of mobile sensors in its grid to the same column of grids. In this way, each grid head will have the information of mobile sensors in other grids located in the same column. When a grid head wants to search for available mobile sensors in other grids, it will send a *request* (REQ) message to the grid head in the same row. Clearly, due to the grid structure, there must be a grid head that receives both the ADV and REQ messages. Consider Fig. 3.3 as an example, where the grid head in $(0, 0)$ sends an ADV message to inform the grids $(0, 1)$, $(0, 2)$, and $(0, 3)$ that a mobile sensor is available in grid $(0, 0)$. Since there is no available mobile sensor in grid $(1, 2)$, its grid head will send an REQ messages to the grids $(0, 2)$, $(2, 2)$, and $(3, 2)$ to search available mobile sensors. It can be seen that the grid head in $(0, 2)$ will

receive both the ADV and REQ messages and then can reply the available mobile sensors in grid $(0, 0)$ to the grid head of $(1, 2)$.

To further reduce the message transmission for searching available mobile sensors, we exploit *search length* to limit the number of grids to be searched. Explicitly, each REQ message is associated with two integers β and M_{grid}, where β is the search length and M_{grid} records the number of available mobile sensors found. Due to the load-balance features, one would like to get as many available mobile sensors as possible. That is why we use β to restrict searching length and within the search lengths, all available mobile sensors rather than the nearest one will be considered for dispatching. Initially, $\beta > 0$ and $M_{grid} = 0$ in each REQ message. When receiving the REQ message, a grid head will increase M_{grid} by the number of mobile sensors in this column since the ADV messages from the grid heads in the same column will publish the available mobile sensors in this column. If $\beta > 1$, the grid head will decrease β by one and then propagate the REQ message to the next grid in the same row. However, if $\beta = 1$ and the value of M_{grid} is still zero, which means that there is no mobile sensor found yet, the grid head will send the REQ message with $\beta = 1$ to the next grid until at least one mobile sensors are available. Fig. 3.3 illustrates an example, where $\beta = 1$ in the initial REQ message. When receiving the REQ message, the grid head in $(0, 2)$ increases M_{grid} by one and decreases β by one. Since the value of β becomes zero, the REQ message will not be propagated toward the left-hand side. When the grid head in $(2, 2)$ gets the REQ message, it finds that $\beta = 1$ and $M_{grid} = 0$. So the grid head in $(2, 2)$ propagates the REQ message with $\beta = 1$ to grid $(3, 2)$ for searching mobile sensors. By exploring the search length, GridSD can reduce not only the message complexity but also the competition of mobile sensors from grid heads. Once obtaining the information of mobile sensors and events, a grid head is able to perform CentralSD locally.

Algorithm 3 GridSD

AT EACH GRID HEAD WITH EVENT LOCATIONS

Notations:

L_{grid} is the set of event locations in this grid.

S_{grid} is the set of mobile sensors found.

Procedure:

1: send REQ with $\beta > 1$ and $M_{grid} = 0$ to the same row;

2: collect the information of S_{grid} from neighboring grid;

3: execute CentralSD with L_{grid} and S_{grid};

4: send dispatch schedules to the mobile sensors in S_{grid};

AT EACH GRID HEAD WITH MOBILE SENSORS

1: send ADV to the same column;

2: **if** receive a dispatch schedule **then**

3: dispatch the mobile sensor according to the schedule;

4: remove the departing mobile sensor;

AT EACH GRID HEAD

Notation:

$M_{current}$ is the number of mobile sensors in the column.

Procedure:

1: **If** receive an REQ **then**

2: $M_{grid} \leftarrow M_{grid} + M_{current}$;

3: **If** $\beta = 1$ **and** $M_{grid} = 0$ **then**

4: $\beta \leftarrow 1$;

5: **else**

6: $\beta \leftarrow \beta - 1$;

7: reply the information to the REQ's originator;

8: **If** $\beta > 0$ **then**

9: propagate the REQ to the next grid;

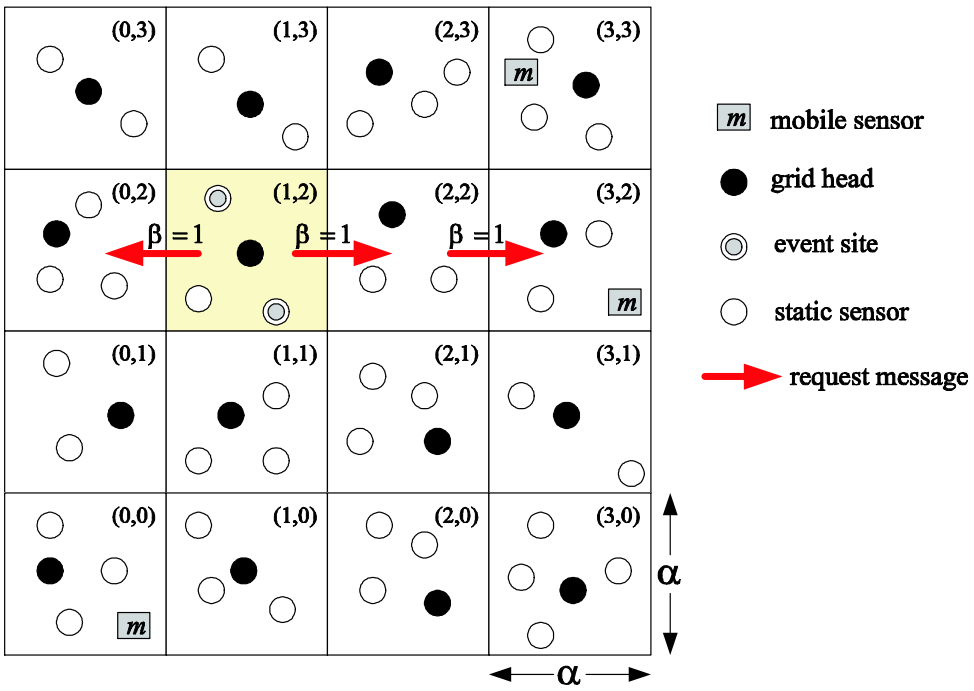


Figure 3.3: An example to show how GridSD works.

Chapter 4

Experimental Results

In this chapter, we evaluate the performances of our proposed algorithms by simulations. We set up a sensing field as a 450m $\times$ 300m rectangle, on which there are 400 static sensors and several mobile sensors uniformly and randomly deployed, respectively. Each mobile sensor has an initial energy of 3960J (joule) reserved for movement and the moving energy consumption per meter is set to 8.27J [15]. In this chapter, the *energy consumption* refers to the one caused by the movements of mobile sensors. The communication distances of static and mobile sensors are set to 150m and 80m, respectively, so that all sensors can form a connected network.

4.1 Effectiveness of CentralSD and GridSD

In the first experiment, we investigate the system lifetime under various dispatching algorithms. We dispatch 50 mobile sensors to visit event locations round by round. During each round, there are 10 to 15 static sensors randomly selected as the event locations. Mobile sensors will then move to these event locations according to the dispatch algorithms and stay at their last-visiting locations to wait for the next dispatch schedule. We mainly observe the percentage of alive mobile sensors during each round. When the number of alive mobile sensors is fewer than that of event locations, the proposed clustering scheme is adopted to group event locations. The system lifetime is referred as

the round when the percentage decreases to zero (i.e., all mobile sensors exhaust their energies). We compare our proposed CentralSD and GridSD against the iteratively-optimized algorithm mentioned in Chapter 1.

Fig. 4.1 shows the system lifetimes when two objective functions Eqs. (2.1) and (2.2) are used (i.e., minimizing the moving cost and maximizing the remaining energy). As can be seen, when the remaining energy of mobile sensors is considered, all the three algorithms will have a longer system lifetime. The iteratively-optimized algorithm always has the shortest system lifetime, although dispatching mobile sensors with the minimal energy cost during each round. This is because the iteratively-optimized algorithm does not balance the loads of mobile sensors, which causes some mobile sensors early to exhaust their energies. The situation becomes worse because these early-exhausted mobile sensors will burden the remaining alive ones with heavy loads. Our proposed algorithms have a longer system lifetime since they not only try to satisfy the objective function but also balance the loads of mobile sensors. Note that CentralSD outperforms GridSD because it has the global knowledge of the network.

4.2 The Impact of Load-Balance

We further evaluate these three dispatch algorithms in terms of the objective function and the load-balance metric (i.e., the standard deviation) among mobile sensors. We use Eq. (2.1) as the objective function and evaluate the average of energy consumption when different dispatch algorithms are adopted. The event locations are randomly selected from 5% to 40% of static sensors. To fairly compare the standard deviation, we set the number of mobile sensors as equal to that of event locations, so that each mobile sensor will be dispatched to exactly one event location.

Fig. 4.2(a) illustrates the average of energy consumption. Since the iteratively-optimized algorithm always finds the optimal solution in each round, it will have the smallest average of energy consumption. By adopting the preference lists, the averages of our proposed algorithms will be

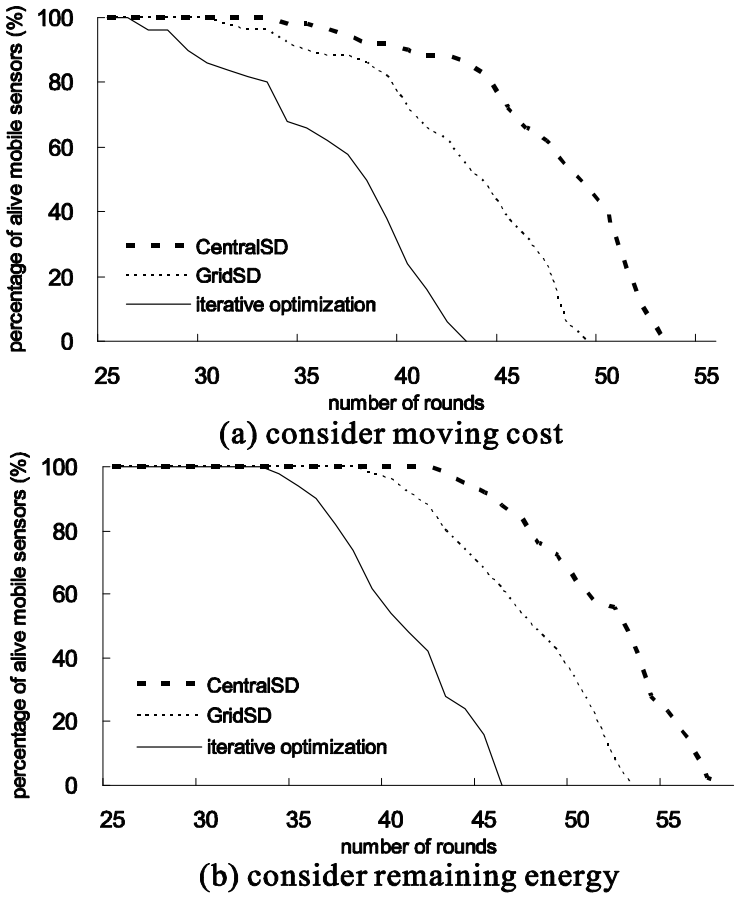


Figure 4.1: Comparison on system lifetime.

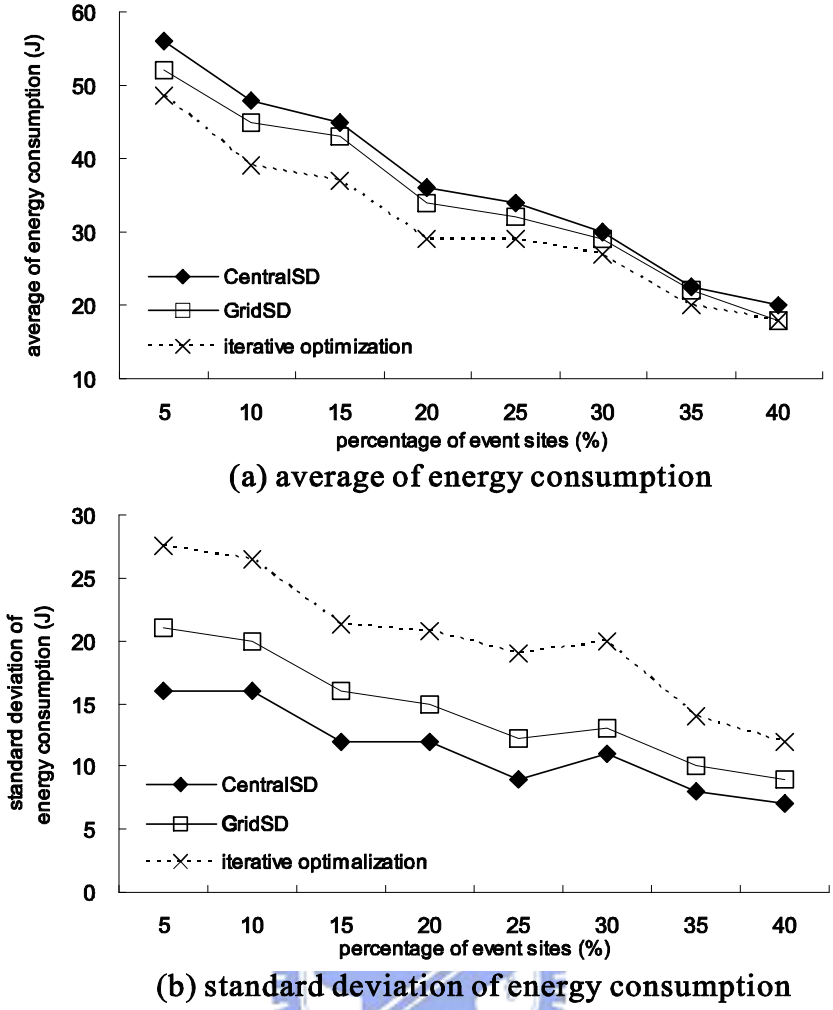


Figure 4.2: Comparisons on energy consumption.

slightly higher than that of the optimal solution. Note that in GridSD, the use of search length will prevent the grid heads with event locations from searching those mobile sensors far away from them, thereby having a smaller average compared with CentralSD. Fig. 4.2(b) shows the standard deviation of energy consumption. We can observe that the standard deviation of the iteratively-optimized algorithm is twice than that of CentralSD, which indicates that the former results in seriously unbalance loads among mobile sensors. Note that GridSD has a larger standard deviation compared with CentralSD since each grid head only has partial information of mobile sensors.

Although CentralSD outperforms GridSD in terms of system lifetime and load-balance, CentralSD incurs a large amount of message transmissions. Fig. 4.3 illustrates the number of packet delivery of CentralSD and GridSD with the number of events varied. We can observe that the num-

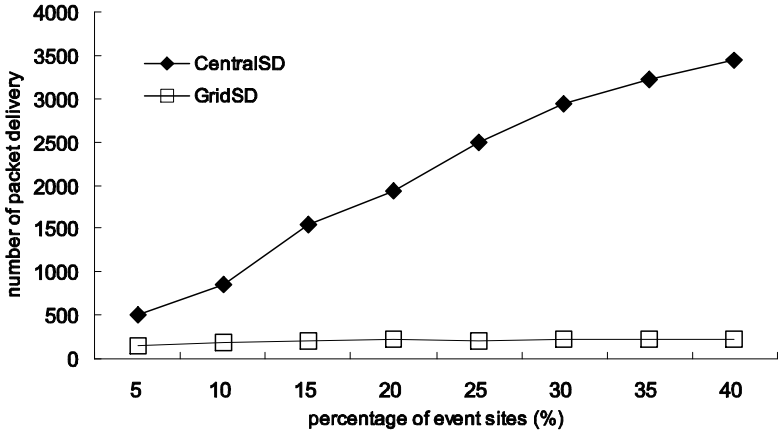


Figure 4.3: Comparison on number of packet delivery.

ber of message transmissions in CentralSD grows very fast as the event locations increase, while that in GridSD remains constant because the loads of message exchange are distributed among grid heads.

4.3 The Impact of Clustering

When the number of event locations is larger than that of mobile sensors, we will group event locations into clusters and each mobile sensor will be dispatched to one cluster. Fig. 4.4 shows the effect of clustering scheme on the average of energy consumption. As can be seen, when the clustering scheme is adopted, mobile sensors can have a lower energy consumption because they do not need to travel around event locations far from each other.

4.4 Sensitivity Analysis on the Coefficient δ

We finally examine the impact of the coefficient δ on the increasing level Δ_B in Eq. (3.1). The value of δ affects both the computation time and result of PairMatching, as shown in Fig. 4.5. Specifically, we use the number of *redundant iterations* that an event location has to repeat Eq. (3.1) to find an available mobile sensor as the metric to measure the computation time. To evaluate the result of PairMatching, we use the product of average and standard deviation of energy consumption. Clearly,

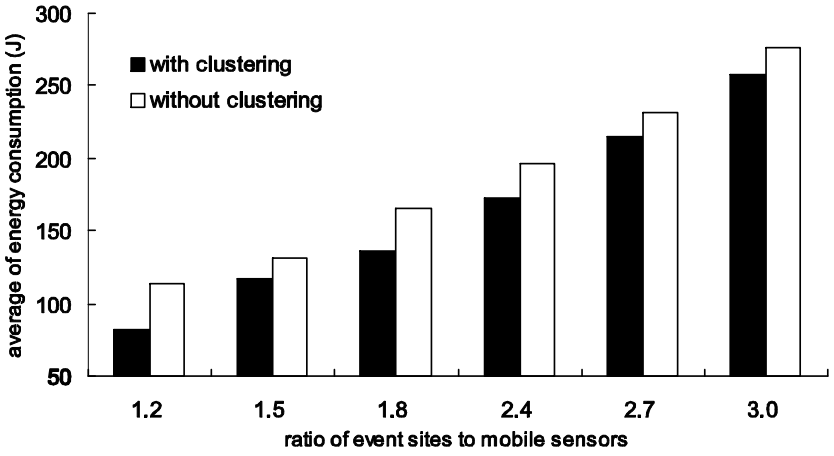


Figure 4.4: The effect of clustering on energy consumption.

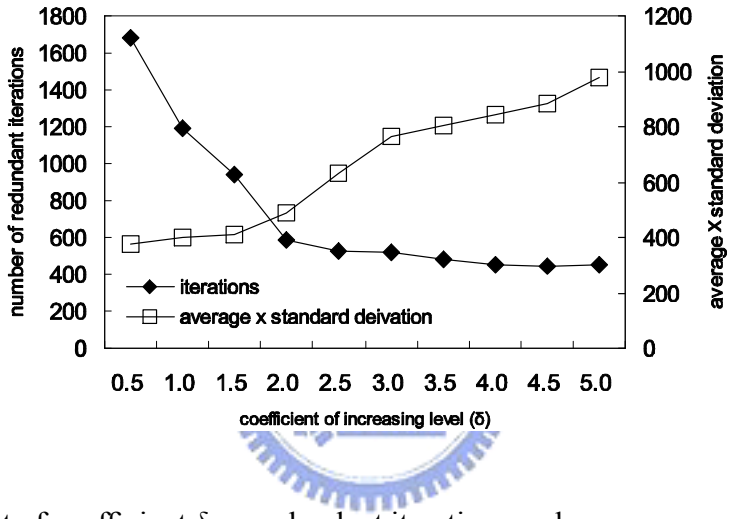


Figure 4.5: The effect of coefficient δ on redundant iterations and energy consumption, where $|L| = |S| = 50$.

a smaller product means a better result since mobile sensors can have a lower energy consumption and a more balanced load. Fig. 4.5 shows the effect of coefficient δ on redundant iterations and energy consumption. We can observe that a smaller δ will cause more redundant iterations while a larger δ will make mobile sensors consume more energy and become unbalanced. From Fig. 4.5, we suggest the optimal value of coefficient δ as 2.0 because both the redundant iterations and the product can be minimized.

4.5 Effect of grid size α and search length β

In Section 3.2, there are two parameters, grid size α and search length β, used in the GridSD scheme. Fig. 4.6 shows the effects of these two parameters on the number of packet delivery and the discovery ratio. Here we define the *discovery ratio* as $\frac{1}{N}\sum_{i=1}^{N}\frac{n_i}{m_i}$, where N is the number of grids, m_i is the number of event locations in a grid i, and n_i is the total number of mobile sensors found by the head of grid i with the search length β. Intuitively, the larger the discovery ratio is, the more mobile sensors a grid head can find. From Fig. 4.6, we can observe that both the message transmissions and the discovery ratio increase as the grid size α and the search length β increase. This is because the searching range (to find available mobile sensors) expands and thus grid heads have to exchange more messages.

Although we can increase the number of mobile sensors found by a grid head by increasing both the values of α and β, grid heads have to exchange more messages. To obtain a reasonable discovery ratio while not to increase too many message transmissions, we suggest to use $\alpha = 30$ and $\beta = 2$ in this experiment since it has a discovery ratio slightly higher than one. In this case, the number of found mobile sensors will approximate to the number of event locations in a grid.

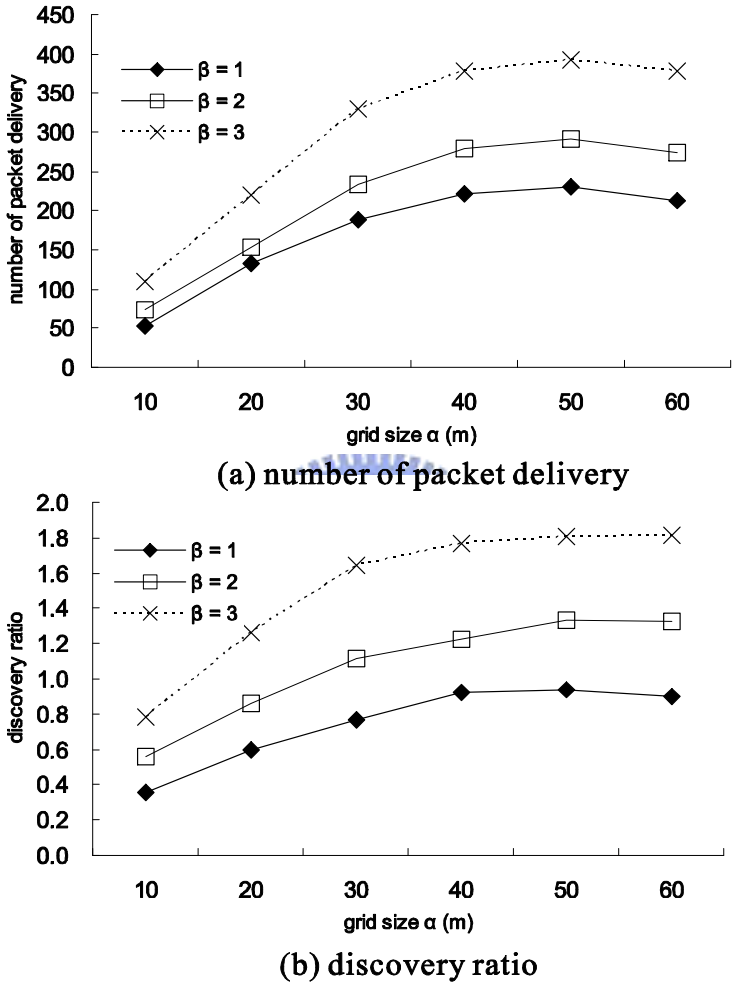


Figure 4.6: The effects of grid size α and search length β on the number of packet delivery and the discover ratio, where $|L| = |S| = 50$.

Chapter 5

Conclusions

In this paper, we have considered how to efficiently dispatch mobile sensors in a hybrid sensor network. By exploring the load-balance of mobile sensors, we have proposed an algorithm CentralSD to dispatch mobile sensors. When the locations are more than the mobile sensors, we developed a clustering scheme to group event locations so that our proposed algorithm is able to apply. In order to reduce the message transmissions, we have also proposed a distributed algorithm GridSD. Simulation results showed that our proposed algorithms can have a longer system lifetime compared with the iteratively-optimized algorithm.

Bibliography

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Commun. Mag.*, vol. 40, no. 8, pp. 102–114, 2002.

[2] 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.

[3] M. Bläser, "A new approximation algorithm for the asymmetric TSP with triangle inequality," in *ACM-SIAM Symp. on Discrete Algorithms*, 2003, pp. 638–645.

[4] N. Bulusu, J. Heidemann, and D. Estrin, "GPS-less low-cost outdoor localization for very small devices," *IEEE Personal Commun. Mag.*, vol. 7, no. 5, pp. 28–34, 2000.

[5] Z. Butler and D. Rus, "Event-based motion control for mobile-sensor networks," *IEEE Pervasive Computing*, vol. 2, no. 4, pp. 34–42, 2003.

[6] T. A. Dahlberg, A. Nasipuri, and C. Taylor, "Explorebots: a mobile network experimentation testbed," in *ACM SIGCOMM Workshop on Experimental Approaches to Wireless Network Design and Analysis*, 2005, pp. 76–81.

[7] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, D. D. Cerra, Ed. Academic Press, 2001.

[8] N. Heo and P. K. Varshney, "Energy-efficient deployment of intelligent mobile sensor networks," *IEEE Trans. on Syst., Man and Cybern. A*, vol. 35, no. 1, pp. 78–92, 2005.

[9] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins, *Global Positioning System: Theory and Practice*. 4th ed., Springer Verlag, 1997.

[10] L. Hu and D. Evans, "Localization for mobile sensor networks," in *Int'l Conf. on Mobile Computing and Networking*, 2004, pp. 45–57.

[11] D. Johnson, T. Stack, R. Fish, D. M. Flickinger, L. Stoller, R. Ricci, and J. Lepreau, "Mobile Emulab: a robotic wireless and sensor network testbed," in *IEEE INFOCOM*, 2006.

[12] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955.

[13] H. Luo, F. Ye, J. Cheng, S. Lu, and L. Zhang, "TTDD: Two-tier data dissemination in large-scale wireless sensor networks," *Wireless Networks*, vol. 11, pp. 161–175, 2005.

[14] M. D. Naish, E. A. Croft, and B. Benhabib, "Dynamic dispatching of coordinated sensors," in *IEEE Int'l Conf. on Systems, Man, and Cybernetics*, 2000, pp. 3318–3323.

[15] M. Rahimi, H. Shah, G. S. Sukhatme, J. Heideman, and D. Estrin, "Studying the feasibility of energy harvesting in a mobile sensor network," in *IEEE Int'l Conf. on Robotics and Automation*, 2003, pp. 19–24.

[16] Y. C. Tseng, Y. C. Wang, and K. Y. Cheng, "An integrated mobile surveillance and wireless sensor (iMouse) system and its detection delay analysis," in *ACM Int'l Symp. on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2005, pp. 178–181.

[17] A. Verma, H. Sawant, and J. Tan, "Selection and navigation of mobile sensor nodes using a sensor network," in *IEEE Int'l Conf. on Pervasive Computing and Communications*, 2005, pp. 41–50.

[18] G. Wang, G. Cao, and T. L. Porta, "A bidding protocol for deploying mobile sensors," in *IEEE Int'l Conf. on Network Protocols*, 2003, pp. 315–324.

[19] G. Wang, G. Cao, and T. L. Porta, “Movement-assisted sensor deployment,” in *IEEE INFOCOM*, 2004, pp. 2469–2479.

[20] G. Wang, G. Cao, T. L. Porta, and W. Zhang, “Sensor relocation in mobile sensor networks,” in *IEEE INFOCOM*, 2005, pp. 2302–2312.

[21] Y. Zou and K. Chakrabarty, “Sensor deployment and target localization based on virtual forces,” in *IEEE INFOCOM*, 2003, pp. 1293–1303.