Design and Implement a Priority-Based Mobile Sensor Dispatch Scheme for Surveillance Applications

You-Chiun Wang, Wen-Chieh Wang, and Yu-Chee Tseng
Department of Computer Science
National Chiao-Tung University, Hsin-Chu, 30010, Taiwan
Email: {wangyc, wangwc, yctseng}@cs.nctu.edu.tw

Abstract

Wireless sensor networks provide a convenient way to monitor the physical environments. However, it is not always possible to have a wireless sensor network in the region of interest. In this case, how to obtain the environmental situations is a big challenge. In this paper, we propose to use mobile sensors to capture the events occurred in a region without any deployment of sensor networks. The idea is to first request mobile sensors to fully scan the environment to find out where events occur and then dispatch mobile sensors to event locations to conduct more advanced data collections. We propose a priority-based dispatch scheme for mobile sensors to visit events' locations based on priorities of these events. In particular, events with higher priorities and shorter moving distances will be visited first. In this way, mobile sensors can fast finish their jobs and the most important events can be analyzed first before they disappear. We have implemented a prototyping system to demonstrate our dispatch idea. Simulation results are also presented to verify the effectiveness of the proposed schemes.

Keywords: dispatch, mobile sensors, sensor applications, surveillance, traveling salesman problem.

1 Introduction

Recently, mobile sensors have attracted a lot of attentions by researchers. By mounting sophisticated sensing devices on mobile platforms [1–3], we can move these *mobile sensors* to conduct sensing missions at certain locations. With mobility, mobile sensors can significantly extend the capability of sensor networks.

In this paper, we consider how to quickly obtain the events occurred in a region without any pre-deployment of sensor networks. This issue is quite important for those surveillance applications in some emergent environments where it is difficult to deploy static sensors in advance. Under such a scenario, one possible solution is to dispatch mobile sensors to collect rough environmental information and then carefully analyze events later. In particular, we first request mobile sensors to fully scan the environment to collect basic information of events (e.g., locations and priorities). Then, we can dispatch mobile sensors to visit these events to conduct more advanced analysis. For example, one can imagine a surveillance application to detect gas leakage in an airtight factory without deployment of sensor networks. We can dispatch mobile sensors to fully scan the factory to collect the densities of gas in different locations and find out potential sources of leakage. Then, mobile sensors can be dispatched to these sources to obtain more detailed information.

Given the priorities of event locations, it is a critical issue to efficiently dispatch a mobile sensor to visit event locations such that the total moving time of the mobile sensor is minimized and the event locations with higher priorities can have a shorter waiting time, where the *waiting time* of an event location is defined as the duration until the mobile sensor reaches that location. Clearly, we should reduce the total moving time of the mobile sensor so that events can be quickly analyzed to satisfy the requirements of surveillance applications. On the other hand, since events may disappear in a short time, the mobile sensor should visit event locations with higher priorities earlier.

In the literature, a large amount of researches on mobile sensors have focused on using mobile sensors to deploy a sensor network [4–7], to enhance the network coverage [8], and to improve the network connectivity [9]. The work in [10] discusses how to move more mobile sensors close to event locations, but it focuses on maintaining complete coverage of the sensing field. Several studies [11,12] implement the pursuer-evader game by a sensor network, where a pursuer (i.e., a mobile sensor) needs to intercept an evader (i.e., a moving object) by the assistance of a static sensor network. However, these works focus on how to quickly tell the pursuer where the evader is through the sensor network.

Some works [13, 14] consider dispatching mobile sensors to visit events in a sensor network consisting of both static and mobile sensors. In [13], static sensors that have detected events will invite and navigate nearby mobile sensors to move to their locations. The mobile sensor that has a shorter moving distance and more energy, and whose leaving will not cause a large coverage hole, will be invited by the static sensors. The work in [14] addresses how to balance the energy consumption of mobile sensors when dispatching them to visit event locations, so that the lifetime of mobile sensors can be extended. However, these works assume that events have the same priority, so their results may not be directly applied in our sensor dispatch problem.

In this paper, we propose a priority-based dispatch scheme for mobile sensors to visit event locations. Given an environment with obstacles, we first request mobile sensors to fully scan the environment to obtain the locations and priorities of events. Then, we dispatch mobile sensors to visit and carefully analyze these events based on their locations and priorities. In particular, the dispatch problem can be viewed as a variance of the *traveling salesman problem (TSP)* that considers the priorities of visiting locations. Since TSP is NP-complete, we thus propose a heuristic approach to efficiently calculate the dispatch schedule of mobile sensors. We also develop a prototyping system to demonstrate our dispatch idea. This prototype is used to monitor the densities of carbon dioxide (CO_2) in an indoor environment. Specifically, we mount CO_2 sensing devices on a mobile platform and dispatch these mobile sensors to monitor CO_2 densities and to analyze potential CO_2 sources. Simulations are also conducted to validate the efficiency of our dispatch scheme in a large-scale scenario.

Major contributions of this paper are two-fold. First, our proposed dispatch scenario allows people to monitor and analyze important events occurred in a region with obstacles, even though there is no infrastructure of sensor networks inside that region. By dispatching mobile sensors to fully scan the region and visit event locations based on their locations and priorities, we can quickly assess the whole environment situation. This may be especially helpful for rescuing applications in some hazardous regions. Second, we implement a prototyping system to realize our dispatch idea. This prototype system demonstrates a surveillance application to monitor the CO_2 densities in an indoor environment.

The rest of this paper is organized as follows: Section 2 formally defines the dispatch problem and reviews some related work. Section 3 proposes our priority-based dispatch scheme. Section 4 gives our prototyping experiences. Section 5 presents some simulation results. Section 6 concludes this paper.

2 Preliminary

2.1 Problem Statement

We consider a sensing field $\mathcal{A}$ possibly with obstacles, as shown in Fig. 1(a). We assume that these obstacles do not partition $\mathcal{A}$. Otherwise, full scanning wouldn't be possible. For convenience, we logically divide $\mathcal{A}$ into small squares (called *grids*). In this way, obstacles can be modeled by grids (marked by gray in Fig. 1(b)). Mobile sensors will patrol inside $\mathcal{A}$ along these grids to conduct full scanning and to visit event locations. Each mobile sensor has four unit mobility patterns: *east*, *west*, *north*, and *south*, where the mobile sensor will move toward the specified direction with one-grid length. Moving from one grid to another will take Δ_m time to finish. The other four diagonal directions can be combined by any two unit mobility patterns. However, it requires Δ_t time for a mobile sensor to make a 90-degree turn.

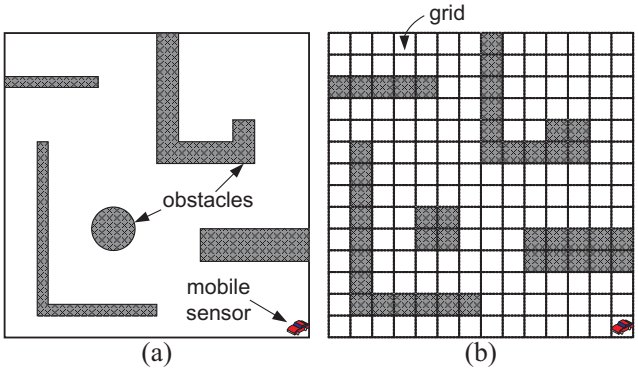


Figure 1: An example to model the environment: (a) sensing field $\mathcal{A}$ with obstacles and (b) logically dividing $\mathcal{A}$ into grids.

We assume that after a mobile sensor fully scans $\mathcal{A}$, we can obtain a set of event locations $\mathcal{L} = \{(l_1, p_1), (l_2, p_2), \cdots, (l_n, p_n)\}$, where $l_i = (x_i, y_i)$ and $p_i > 0$ are the location and priority of an event i, respectively. A smaller value of p_i means a higher priority. Note that two events may have the same priority. Our *sensor dispatch problem* is stated as follows: Given a mobile sensor initially located at $l_0 = (x_0, y_0)$ and the set $\mathcal{L}$, how can we dispatch the mobile sensor to visit all locations in $\mathcal{L}$ such that the total moving time of the mobile sensor is minimized and event locations with higher priorities can have a shorter waiting time?

Note that the above modeling does not consider the moving time to fully scan the sensing field $\mathcal{A}$ and the time to detect events. Since it requires the mobile sensor to visit every grid inside $\mathcal{A}$ to conduct the full scanning and the time to analyze an event depends on the sensing capability of the

mobile sensor, we thus ignore these two times. In addition, we aim at the dispatch problem of one mobile sensor. In the case of multiple mobile sensors, we can partition $\mathcal{A}$ into multiple non-overlapped subregions and then dispatch one mobile sensor to travel inside each subregion.

2.2 Related Work

Our dispatch problem can be viewed as a TSP variance that considers the priorities of visiting locations. TSP is one of the famous NP-complete problems and there have been many approximation solutions proposed to solve it and its variances [15]. Given an undirected weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, TSP asks how to find a Hamilton cycle $\mathcal{C}$ on $\mathcal{G}$ such that the total edge weight of $\mathcal{C}$ can be minimized. To solve TSP, Blaser [16] suggests that we can first find the minimum spanning tree on $\mathcal{G}$ and then visit all vertices in $\mathcal{V}$ by a preorder tree walk of the spanning tree. The visiting sequence of vertices is the approximate solution of TSP. Some works adopt more sophisticated methods to solve TSP. For example, Zhu and Chen [17] solve TSP by simulating ants' behaviors. In particular, each time when an ant passes through a path, it will leave pheromone along that path. Such pheromone will attract other ants to walk through this path. Thus, when a path is walked by more ants, it will have a larger amount of pheromone. By adopting this concept, we can first try some paths to visit vertices in the graph. When a good path is found, it will leave more pheromone and thus attracts more ants to walk through it. Finally, the path with the most pheromone will be the solution. Other works use simulated annealing [18], genetic algorithm [19], and tabu search method [20] to solve TSP. Obviously, these methods can approximate a better solution, but they may suffer from a higher computation cost.

On the other hand, several works consider the variances of TSP. The work in [21] introduces the concept of *time window* to TSP, where each location is associated with a time duration where we expect the salesman will visit that location within the specified time duration. This can be applied to some applications with timetables such as bus scheduling or consignment of goods. The work [22] considers a dynamic environment, where visiting locations may be disappeared or changed as time goes by. This can be used in some applications like realtime computing in satellite systems. Given a set of locations $\mathcal{L}$, the work [23] considers how to visit locations in $\mathcal{L}$ with a minimum cost, where a subset $\mathcal{L}' \subseteq \mathcal{L}$ of locations may follow some probability model. As can be seen, the solutions of these works could not be directly applied in our dispatch problem.

3 A Priority-Based Dispatch Scheme

Given the sensing field $\mathcal{A}$ and a mobile sensor initially located at l_0, our priority-based dispatch scheme involves the following steps:

1. Dispatch the mobile sensor to visit all grids in $\mathcal{A}$ to collect locations and rough information of events $\mathcal{L}$. To conduct such a full scanning, we can adopt the solution proposed in [24].

2. Calculate the shortest distance $d(l_i, l_j)$ between any two locations l_i and l_j, where $l_i, l_j \in \mathcal{L} \cup \{l_0\}$. Note that the above calculation should consider the existence of obstacles. How to calculate $d(l_i, l_j)$ will be discussed in Section 3.1.

3. Construct a weighted complete graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{L} \cup \{l_0\}$. For each vertex $l_i \in \mathcal{L}$, we associate it with a priority value p_i. How to assign the priority depends on the application requirement. One possible assignment is based on events' strengths (e.g., gas's densities). The weight $w_{i,j}$ of each edge $(l_i, l_j) \in \mathcal{E}$ is defined as the moving time for a mobile sensor to move along the shortest distance $d(l_i, l_j)$.

4. Find a Hamilton cycle $(l_0, l_{v_1}, l_{v_2}, \cdots, l_{v_n}, l_0)$ on $\mathcal{G}$ to minimize

$$w_{0,v_1} + \sum_{i=1}^{n-1} w_{v_i,v_{i+1}} + w_{v_n,0}, \tag{1}$$

such that

$$\sum_{i=1}^{x-1} w_{v_i,v_{i+1}} \leq \sum_{i=1}^{y-1} w_{v_i,v_{i+1}}, \forall p_{v_x} \leq p_{v_y}. \tag{2}$$

Eq. (1) means that we should minimize the total moving time for the mobile sensor to visit all event locations, and Eq. (2) indicates that an event location l_{v_x} with a higher priority should have a smaller waiting time compared with another event location l_{v_y} with a lower priority. Note that we eliminate the term w_{0,v_1} from both the left and right parts of Eq. (2). How to find such a Hamilton cycle will be discussed in Section 3.2.

5. Dispatch the mobile sensor to visit event locations following the sequence of $l_{v_1}, l_{v_2}, \cdots, l_{v_n}$ and then come back to l_0. The mobile sensor will move along the shortest path calculated in step 2 when visiting an event location.

6. Go to step 1 to conduct the full scanning again after a predefined period of time.

3.1 Calculating the Shortest Distance between Two Locations

In this section, we discuss how to calculate the shortest distance between any two locations, considering the existence of obstacles. Given the grid partition of the sensing field and two locations l_S and l_D, the calculation of the shortest distance between l_S and l_D involves the following steps:

1. For each grid i that is not an obstacle, we associate it with a grade g_i. Initially $g_i = \infty$ for all i.
2. We start from the grid with l_D and set $g_D = 0$. Then, for each adjacent grid j (in the directions of north, south, east, and west) of a grid i that has already been assigned with a non-infinite grade, we set $g_j = g_i + 1$. If such a grid j is an obstacle, we just ignore it.
3. Repeat step 2 until the grid with l_S has been assigned with a non-infinite grade.
4. We then start from the grid with l_S. For the current grid i, we always select the adjacent grid j that is not an obstacle and $g_j = g_i - 1$ as the next grid to move. If two or more such grids are found, we select the one without changing the current moving direction. In the case that all candidates for the next grid will change the current moving direction, we randomly select one grid to move.
5. Repeat step 4 until we arrive at the grid with l_D.

Fig. 2 gives an example, where the sensing field is modeled by a 5×5 grids. Two locations l_S and l_D are located at grids $(0, 0)$ and $(1, 4)$, respectively. We start from the grid $(1, 4)$ and assign a grade to each grid, until to the grid $(0, 0)$, as shown by the numbers in Fig. 2(b). Then, we start from grid $(0, 0)$ and find the shortest path to grid $(1, 4)$ following the decreasing order of grades, as shown in Fig. 2(c). Note that when we arrive at grid $(0, 1)$, there are two grids $(0, 2)$ and $(1, 1)$ that can be selected as candidates to move. Since the current moving direction is east (from $(0, 0)$ to $(0, 1)$), we will select grid $(0, 2)$ as the next grid because it does not need to change the moving direction. The total moving time for a mobile sensor to move along this shortest distance is $5\Delta_m + 2\Delta_t$.

3.2 Finding a Suitable Hamilton Cycle

Recall that given a weighted complete graph $\mathcal{G}$, our goal is to find a Hamilton cycle starting from l_0 such that both Eqs. (1) and (2) can be satisfied. However, minimizing the total moving time of the mobile sensor (that is, to satisfy Eq. (1)) may not always guarantee that event locations with

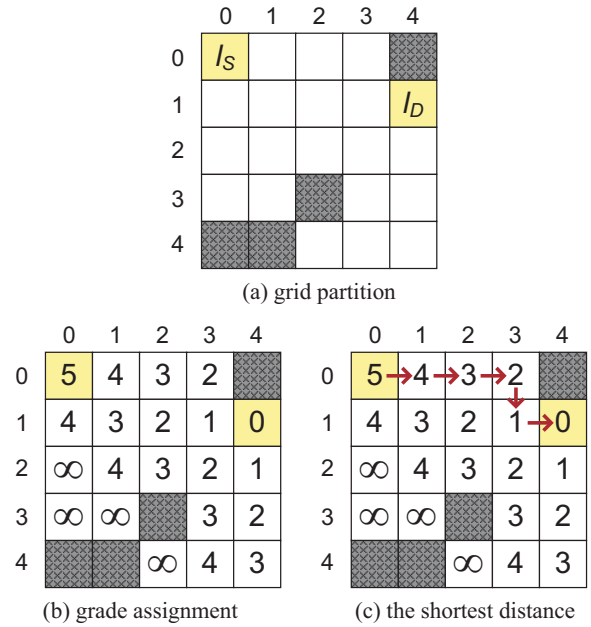


Figure 2: An example to calculate the shortest distance between two locations l_S and l_D.

a higher priority can have a smaller waiting time (that is, to satisfy Eq. (2)), and vice versa. Therefore, we use a *cost* $\mathcal{C}$ to take care of the effects of both Eqs. (1) and (2):

$$\begin{aligned}\mathcal{C} =& f(v_1) \cdot w_{0,v_1} + f(v_2) \cdot w_{v_1,v_2} + \cdots \\ &+ f(v_{n-1}) \cdot w_{v_{n-2},v_{n-1}} + w_{v_{n-1},v_n} + w_{v_n,0}, \end{aligned} \quad (3)$$

where $f(\cdot) \geq 1$. In particular, we should find a Hamilton cycle with a minimum cost $\mathcal{C}$. Here, the cost should take both the total moving time of the mobile sensor and the priorities of event locations into consideration. Thus, we use a *penalty function* $f(\cdot)$ to reflect the effect of priority. Intuitively, $f(v_i)$ should return a larger value if an event location l_{v_i} with a lower priority is selected. By $f(\cdot)$, those event locations with higher priorities could be selected first. However, to avoid the case that some nearer event locations are not selected due to their lower priorities, which may cause the mobile sensor to move in a too long distance, the returning value of $f(\cdot)$ cannot be too large. Therefore, we suggest setting the penalty function $f(\cdot)$ as follows:

$$f(v_i) = 1 + \frac{p_{v_i} - p_{\min}}{p_{\max} - p_{\min}}, \quad (4)$$

where $p_{\min} = \min_{\forall l_j \in \mathcal{L} - \mathcal{L}_V} \{p_j\}$ and $p_{\max} = \max_{\forall l_j \in \mathcal{L} - \mathcal{L}_V} \{p_j\}$, where $\mathcal{L}_V$ is the set of event locations that have already been visited. In Eq. (4), we can observe that when an event locations l_{v_i} with a larger value of p_{v_i} (i.e., lower priority) is selected, the value of $f(v_i)$ will be increased and thus the cost $\mathcal{C}$ is also increased. On the other hand, because

$1 \leq f(\cdot) \leq 2$, the effect of penalty function will not be too violent. In this way, a nearer event location with a lower priority could be selected to minimize the total moving time. Note that in Eq. (3), the term w_{v_{n-1},v_n} is not multiplied by the factor $f(v_n)$ because l_{v_n} is the only candidate to be selected as the next visiting location.

It can be observed that finding a Hamilton cycle with a minimum cost $\mathcal{C}$ is NP-hard because we can reduce TSP to this problem by setting all $f(\cdot) = 1$ in Eq. (3). Thus, we propose a heuristic solution by adopting a simple greedy idea. Specifically, we start finding the Hamilton cycle from the vertex l_0. Given the current location l_i, we select an unvisited vertex l_j as the next visiting location such that the value of $f(j) \cdot w_{i,j}$ can be minimized. We repeat the above operation until all event locations are visited. With such a greedy selection, we can obtain a Hamilton cycle with a smaller cost.

Fig. 3 gives an example to show how to find the Hamilton cycle with the minimum cost. By adopting the TSP solution, we will obtain a Hamilton cycle $0 \rightarrow 7 \rightarrow 9 \rightarrow 2 \rightarrow 5 \rightarrow 0$, which has a total edge weight of $8+11+9+9+9 = 46$. However, it will have a cost of

$$(1 + \frac{7-2}{9-2}) \times 8 + (1 + \frac{9-2}{9-2}) \times 11 + (1 + \frac{2-2}{5-2}) \times 9 + 9 + 9 = 62.71.$$

On the other hand, our greedy approach will obtain a Hamilton cycle of $0 \rightarrow 2 \rightarrow 5 \rightarrow 7 \rightarrow 9 \rightarrow 0$, which has a total edge weight of $9 + 9 + 8 + 11 + 12 = 49$. This greedy approach will result in a cost of

$$(1 + \frac{2-2}{9-2}) \times 9 + (1 + \frac{5-5}{9-5}) \times 9 + (1 + \frac{7-7}{9-7}) \times 8 + 11 + 12 = 49.$$

As can be seen, although our greedy approach will find a Hamilton cycle with a total edge weight larger than that of the TSP solution, it can result in a quite smaller cost. In this way, event locations with higher priorities can be visited first.

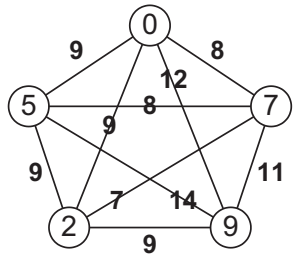


Figure 3: An example to show how to find the Hamilton cycle with the minimum cost.

3.3 Time Complexity Analysis

Next, we analyze the time complexity of our priority-based dispatch scheme. Let n be the number of event locations, and k be the number of grids in the sensing field, excluding those grids used to model obstacles. In step 1, conducting a full scanning takes $O(k)$ time because the mobile sensor has to travel all grids. In steps 2 and 3, we need to calculate the shortest distance between any two locations. This operation takes $O(k)$ time to assign grade to each grid and requires at most $O(k)$ time to calculate the shortest path since the worst case is to pass through all grids. The above operation will be repeated $\binom{n+1}{2}$ times. So, it takes totally $\binom{n+1}{2} \cdot O(k) = O(n^2 \cdot k)$ to conduct steps 2 and 3. Step 4 takes $O(n)$ time due to the greedy selection, and step 5 requires at most $O(k)$ time for the mobile sensor to visit all event locations (because the worst case is to travel all grids). Therefore, the total time complexity of our priority-based dispatch scheme is

$$O(k) + O(n^2 \cdot k) + O(n) + O(k) = O(n^2 \cdot k).$$

4 Prototyping Experiences

We have implemented a prototyping system to monitor CO_2 densities in an indoor environment. This system consists of a *control server* and a *mobile sensor*. The mobile sensor will fully scan the environment to collect CO_2 densities inside the room. Then, the control server will identify some locations with higher CO_2 densities (that is, potential CO_2 sources) and request the mobile sensor to visit these locations, according to our priority-based dispatching scheme. Below, we give our prototyping experiences, including the implementation details and the user interface at the control server.

4.1 Implementation Details

Fig. 4 shows our mobile sensor, which consists of the following components:

- **Stargate processing board:** The Stargate [25] is the processing platform of the mobile sensor. It is composed of a 32-bits, 400-MHz Intel PXA-255 XScale RISC processor with 64 MB main memory and 32 MB extended flash memory. It also has a daughter board with an RS-232 serial port, a PCMCIA slot, a USB port, and a 51-pin extension connector. It drives the CO_2 sensor through a COM port, and an IEEE 802.11 WLAN card through its PCMCIA slot. The Stargate controls the LEGO car via a USB port connected to a LEGO infrared (IR) tower, as shown in Fig. 4.

- **LEGO car:** The LEGO car [26] supports the mobility of mobile sensor. It has an infrared ray receiver in the front to receive commands from the tower (which are passed from the Stargate processing board) and two motors on the bottom to drive wheels. It also has several light sensors for the navigation purpose, which will be described later.
- **CO_2 sensor:** The CO_2 sensor is used to collect the CO_2 densities inside the room. Here, we adopt TGS 4161 [27] as our CO_2 sensor. TGS 4161 is a solid electrolyte CO_2 sensor that provides miniaturization and low power consumption. It can detect a range of 350 to 5000 ppm of CO_2 density.
- **IEEE 802.11 WLAN card:** The control server will pass commands (e.g., full scanning or dispatching) to the mobile sensor via the IEEE 802.11 WLAN card. When collecting CO_2 readings from the CO_2 sensor, the mobile sensor can also report the data to the control server through the WLAN card.

Note that some more sophisticated sensing devices can be attached to the mobile sensor to increase its sensing capability. For example, we can attach a webcam on the mobile sensor so that it can take snapshot at the event locations to provide image information. The size of the mobile sensor is approximately $20\,\mathrm{cm} \times 15\,\mathrm{cm}$.

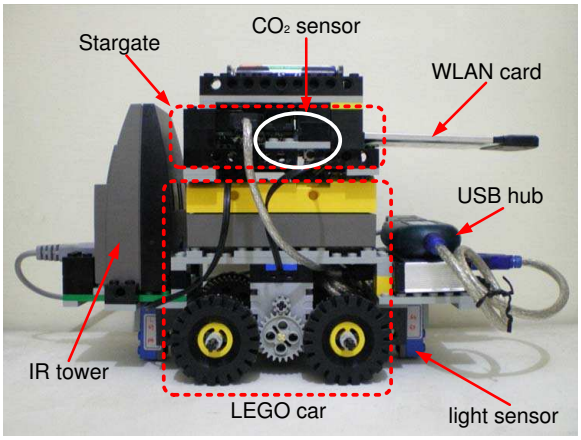


Figure 4: The mobile sensor.

Fig. 5 gives the operating flowchart of the mobile sensor. Once powering on, the Stargate processing board will configure all corresponding hardwares, including the drivers of the IEEE 802.11 WLAN card, USB ports, and the COM port. Then, it will communicate with the control server to obtain a dynamic IP (this can be done by setting a DHCP server at the control server) and set up all necessary network configurations. After establishing the communication link with the control server, the Stargate processing board will notify the CO_2 sensor to start collecting data and then wait commands from the control server (via the WLAN card). When receiving a dispatching command from the control server, the mobile sensor will move to the specified locations and report the CO_2 densities of these locations to the control server.

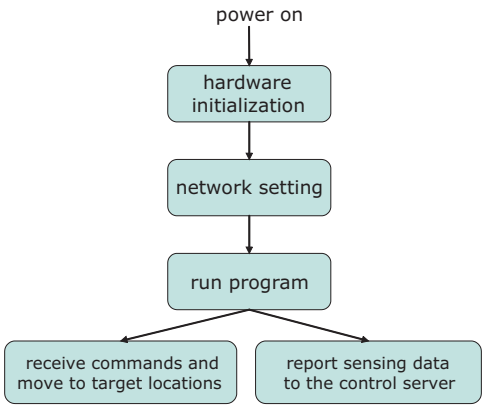


Figure 5: The operating flowchart of the mobile sensor.

In our prototyping, an experimental 9×5 grid-like sensing field is implemented, as shown in Fig. 6. Black tapes represent roads and golden tapes represent intersections. One mobile sensor is placed on the sensing field. We use some boxes to model obstacles. Red crosses on the sensing field indicate the event locations, where we can put some small piece of dry ice to simulate CO_2 sources.

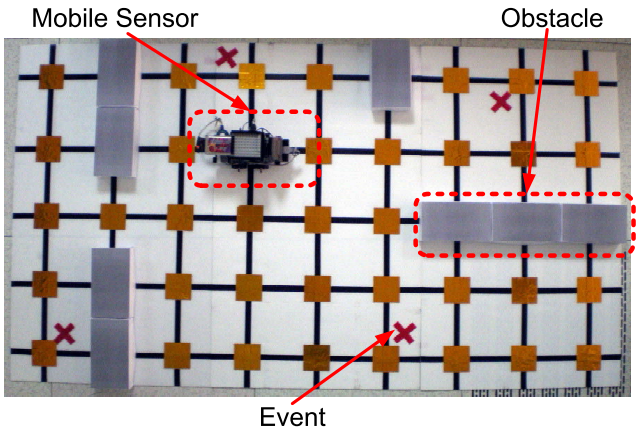


Figure 6: A 9×5 grid-like sensing field in our experiment.

4.2 User Interface at the Control Server

We have also designed a user interface at the control server for users to monitor the statuses of the sensing field and to control the actions of the mobile sensor, as shown in Fig. 7. It has four areas: *information*, *monitor*, *status*,

and *control* areas. The information area shows the current status of the mobile sensor, including the readings of the CO_2 density, the location and direction of the mobile sensor, and so forth. The monitoring area illustrates the state of the sensing field, where the blue rectangle represents the mobile sensor, the small yellow rectangles indicate event locations, and the grids marked with oblique lines are obstacles. The status area gives the server's status, including the contents of packets exchanged with the mobile sensor. Finally, the control area provides an interface for users to control the motion of the mobile sensors. Users can issue a high-level command such as taking full scanning of the sensing field, or a low-level command such as moving one-grid length or making a 90-degree turning.

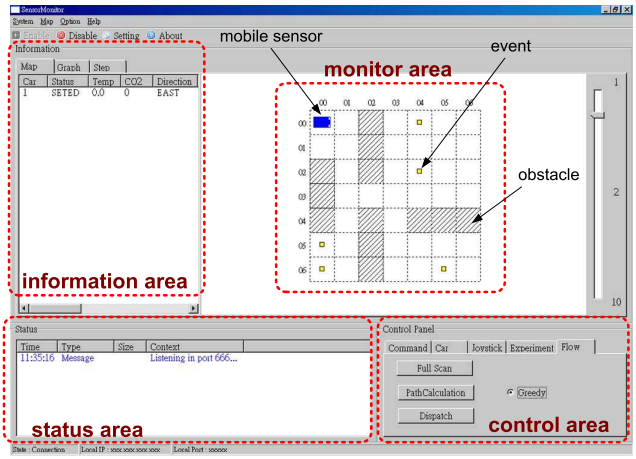


Figure 7: User interface at the control server.

At the control server, we also set up a DHCP server to assign dynamic IPs to mobile sensors. Here, we use DHCPD32 [28] (version 1.10) as our DHCP server, as shown in Fig. 8. In addition, we also provide an interface for users to edit the map, as shown in Fig. 9. Users can specify the size of the map, where the minimum and maximum map sizes are 2×2 and 100×100 grids, respectively. Obstacles can be specified by clicking corresponding grids. With this map editor, users can easily model the sensing field.

Fig. 10 shows some snapshots of the user interface when executing our priority-based dispatch scheme. Fig. 10 (a) illustrates the shortest path between two locations. Fig. 10 (b) gives the final cost matrix (with six locations), where the number in each grid indicates the length of the shortest path between two locations. Fig. 10 (c) shows the basic motion operations of mobile sensor. For example, it takes four steps for the mobile sensor to move to the event location with priority 1. The mobile sensor should move to east with one grid-length, move to south with three grid-lengths, move to east with three grid-lengths, and finally move to north with three grid-lengths. Fig. 10 (d) shows the final result of the

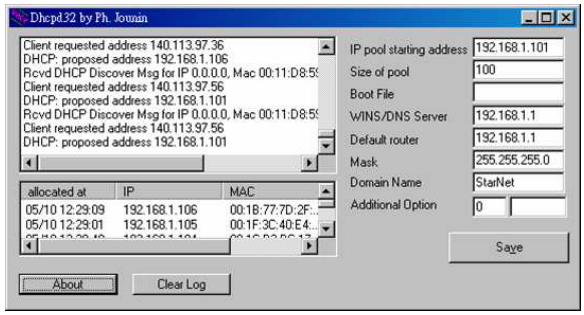


Figure 8: The DHCP server.

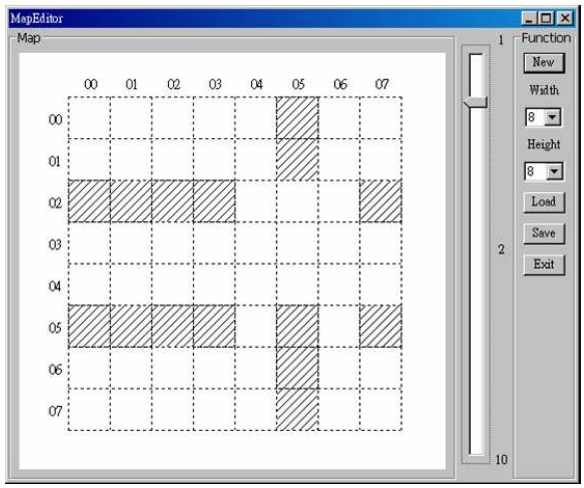


Figure 9: The map editor.

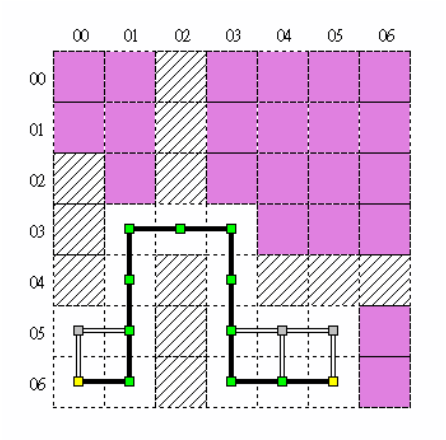
patrolling path of the mobile sensor.

5 Simulation Results

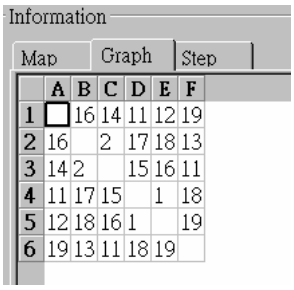
We also develop a simulator to evaluate the performance of the proposed dispatch scheme. We set up a sensing field as a 100×100 grids, on which there may be several obstacles. We randomly pick up 50, 100, 150, 200, 250, and 300 grids (excluding those grids representing obstacles) as event locations. The values of Δ_m and Δ_t are set to 1 and 2, respectively. We mainly compare our priority-based dispatch scheme against the TSP approximate solution proposed in [16].

Fig. 11 shows the comparison of the total moving time of the mobile sensor under our priority-based dispatch scheme and the TSP approximate solution. We can observe that the TSP solution has a larger total moving time, because it is only an approximate solution. Our priority-based dispatch scheme adopts a greedy approach, so it can have a smaller total moving time.

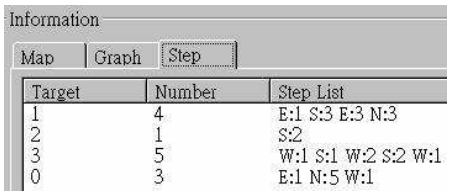
Fig. 12 shows the comparison of the costs in Eq. (3) of



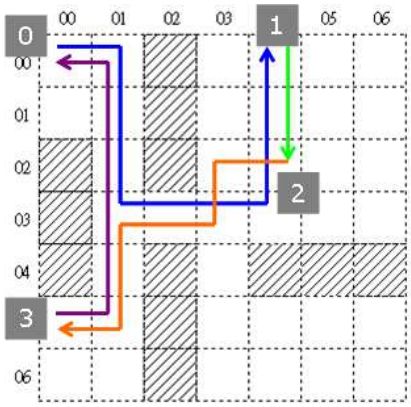
(a) calculation of one path



(b) the cost matrix



(c) the basic motion operations of the mobile sensor



(d) the patrolling path of the mobile sensor

Figure 10: Snapshots of the user interface when executing the priority-based dispatch scheme.

the Hamilton cycles found by our dispatch scheme and the TSP solution. As can be seen, our priority-based dispatch scheme can find a Hamilton cycle with a cost smaller than that of the TSP solution. This indicates that event locations with higher priorities could be visited first.

Fig. 13 shows the waiting time of event locations by adopting our priority-based dispatch scheme. In this experiment, we randomly select 100 locations as event locations. In Fig. 13, we can observe that event locations with higher priorities can have a shorter waiting time, which satisfy our goal.

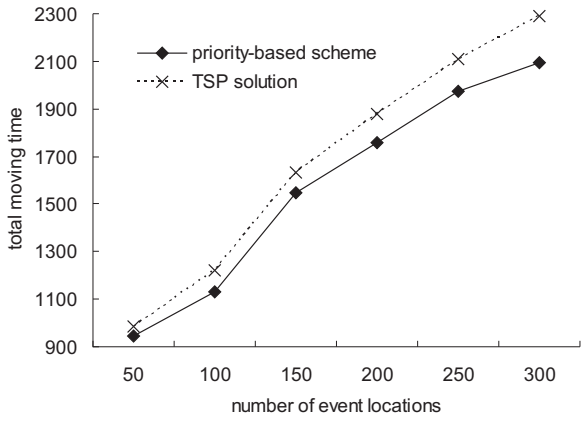


Figure 11: Comparison on the total moving time of the mobile sensor.

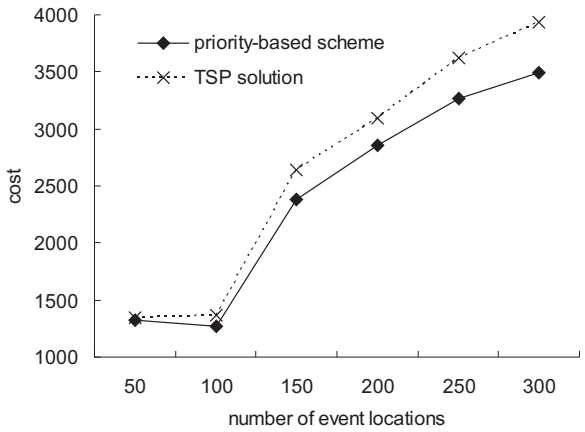
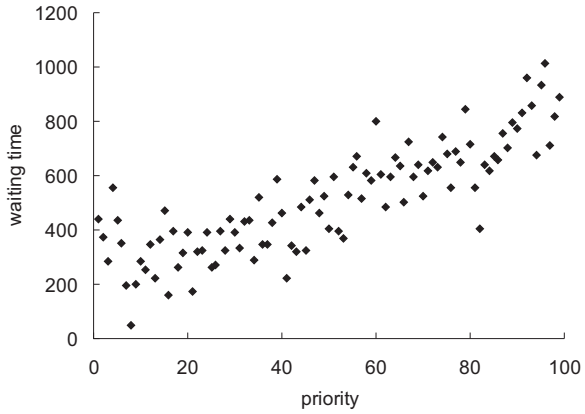


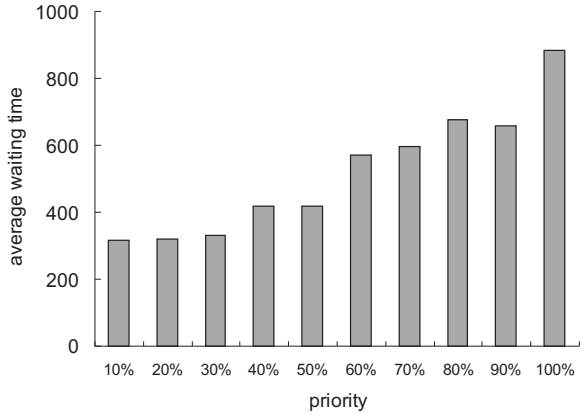
Figure 12: Comparison on the costs of the Hamilton cycles.

6 Conclusions

In this paper, we have proposed a scenario to use mobile sensors to detect events in a region without any deployment of wireless sensor networks. Mobile sensors will be first requested to conduct full scanning of the region to collect rough information of the environment and to identify



(a) waiting time of each event location



(b) average waiting time of each 10 event locations

Figure 13: Waiting time of event locations.

potential event locations. Then, mobile sensors will be dispatched to visit these event locations according to their priorities. We have proposed a priority-based dispatch scheme for mobile sensors to visit event locations. In particular, our dispatch scheme can reduce the total time for the mobile sensor to visit all event locations, while event locations with higher priorities can be visited earlier. In this way, event locations with higher priorities can have a smaller waiting time. Simulation results have shown that our proposed dispatch scheme outperforms the approximated solution of TSP, on both the total moving time of the mobile sensor and the average waiting time of event locations with higher priorities. In this paper, we have also implemented a prototyping system to realize our dispatch idea. Such a prototyping system can be used to monitor CO_2 densities in an indoor environment. The prototyping experience is also reported in this paper.

Acknowledgment

Y.-C. Tseng's research is co-sponsored by Taiwan MoE ATU plan, by NSC grants 93-2752-E-007-001-PAE, 96-2623-7-009-002-ET, 95-2221-E-009-058-MY3, 95-2221-E-009-060-MY3, 95-2219-E-009-007, 95-2218-E-009-209, and 94-2219-E-007-009, by Realtek Semiconductor Corp., by MOEA under grant number 94-EC-17-A-04-S1-044, by ITRI, Taiwan, by Microsoft Corp., and by Intel Corp.

References

[1] 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.

[2] 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.

[3] 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.

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

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

[6] N. Heo and P. K. Varshney, "Energy-efficient deployment of intelligent mobile sensor networks," *IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans*, vol. 35, no. 1, pp. 78–92, 2005.

[7] Y. C. Wang, C. C. Hu, and Y. C. Tseng, "Efficient placement and dispatch of sensors in a wireless sensor network," *IEEE Transactions on Mobile Computing*, vol. 7, no. 2, pp. 262–274, 2008.

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

[9] 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.

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

[11] M. Demirbas, A. Arora, and M. Gouda, "A pursuer-evader game for sensor networks," in *Sixth Symposium on Self-Stabilizing Systems*, 2003, pp. 1–16.

[12] 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," in *IEEE European Workshop on Wireless Sensor Networks*, 2005, pp. 93–107.

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

[14] Y. C. Wang, W. C. Peng, M. H. Chang, and Y. C. Tseng, "Exploring load-balance to dispatch mobile sensors in wireless sensor networks," in *IEEE International Conference on Computer Communications and Networks*, 2007, pp. 669–674.

[15] M. Bellmore and G. L. Nemhauser, "The traveling salesman problem: A survey," *Operations Research*, vol. 17, no. 3, pp. 538–557, 1968.

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

[17] Q. B. Zhu and S. Y. Chen, "A new ant evolution algorithm to resolve tsp problem," in *International Conference on Machine Learning and Applications*, 2007, pp. 62–66.

[18] C. S. Jeong and M. Kim, "Fast parallel simulated annealing for traveling salesman problem," *Neural Network*, vol. 3, pp. 947–953, 1990.

[19] A. Mohebifar, "New binary representation in genetic algorithms for solving tsp by mapping permutations to a list of ordered numbers," in *WSEAS International Conference on Computational Intelligence, Man-Machine Systems and Cybernetics*, 2006, pp. 363–367.

[20] N. Yang, P. Li, and B. Mei, "An angle-based crossover tabu search for the traveling salesman problem," in *International Conference on Natural Computation*, 2007, pp. 512–516.

[21] E. Baker, "An extra algorithm for time constrained traveling salesman problem," *Operations Research*, vol. 31, pp. 938–945, 1983.

[22] X. S. Yan, A. M. Zhou, L. S. Kang, and Y. P. Chen, "Tsp problem based on dynamic environment," in *Intelligent Control and Automation Fifth World Congress*, 2004, pp. 2271–2274.

[23] A. M. Campbell, "Aggregation for the probabilistic traveling salesman problem," *Computers & Operations Research*, vol. 33, no. 9, pp. 2703–2724, 2006.

[24] C. Y. Chang, H. R. Chang, C. C. Hsieh, and C. T. Chang, "OFRD: obstacle-free robot deployment algorithms for wireless sensor networks," in *IEEE Wireless Communications and Networking Conference*, 2007, pp. 4371–4376.

[25] Crossbow, "SPB400 - Stargate Gateway," http://www.xbow.com.

[26] MINDSTORM, "Robotics Invention System," http://mindstorms.lego.com/eng/default.asp.

[27] TGS 4161, "The detection of Carbon Dioxide," http://www.tashika.co.jp.

[28] TFTPD32, "A free TFTP and DHCP server for windows," http://tftpd32.jounin.net.