

Collision-Free Motion Algorithms for Sensors Automated Deployment to Enable a Smart Environmental Sensing-Net

Ting-Yu Lin^{ID}, Member, IEEE, Kun-Ru Wu^{ID}, You-Shuo Chen^{ID}, and Yan-Syun Shen^{ID}

Abstract—As natural habitats protection has become a global priority, smart sensing-nets are ever-increasingly needed for effective environmental observation. In a practical monitoring network, it is critical to deploy sensors with sufficient automated intelligence and motion flexibility. Recent advances in robotics and sensors technology have enabled automated mobile sensors deployment in a smart sensing-net. Existing deployment algorithms can be employed to calculate adequate destinations (goals) for sensors to perform respective monitoring tasks. However, given the calculated goal positions, the problem of how to actually coordinate a fleet of robots and schedule moving paths from random initials to reach their goals safely, without collisions, remains largely unaddressed in the wireless sensor networking (WSN) literature. In this paper, we investigate this problem and propose polynomial-time collision-free motion algorithms based on batched movements to ensure all the mobile sensors reach their goals successfully without incurring collisions. We observe that the grouping (batching) strategy is similar to the coloring procedure in graph theory. By constructing a conflict graph, we model the collision-free path scheduling as the well-known k -coloring problem, from which we reduce to our k -batching problem (determining the minimum number of required batches for a successful deployment) and prove its NP completeness. Since the k -batching problem is intractable, we develop CFMA (collision-free motion algorithm), a simple yet effective batching (coloring) heuristic mechanism, to approximate the optimal solution. Performance results show that our motion algorithms outperform other existing path-scheduling mechanisms by producing 100% sensors reachability (success probability of goals reaching), time-bounded deployment latency with low computation complexity, and reduced energy consumption.

Note to Practitioners— This research was originally motivated by an oceanography project, which studied marine microbes by sending a team of tiny robots (sensors) randomly scattered on the ocean floor. For hard-to-access habitats like deserts or oceans, where manual placement of sensors is costly or impossible, auto-

matically scheduling robots movements to calculated positions from random initials is essential for an effective monitoring. Our contribution is unique in two ways. First, traditional path-planning research focuses more on independent robots navigating the environment, whereas we target on the network automation problem, coordinating a fleet of robots working together to perform an environmental sensing task. Second, we observe that existing movement methods are too complicated and energy-consuming due to the calculation on-the-go nature. To provide a practical solution, we suggest and design our motion algorithms from a new perspective: pre-scheduled batched movements. Our approach requires a central server for grouping (batching) calculations, but can effectively reduce computation complexity and save significant energy expenditure exerted on resource-limited sensors. The proposed collision-free motion algorithm (CFMA) is a suboptimal yet efficient batching solution, which can be easily applied in real-life open-space monitoring networks. Insights from this foundational research will facilitate future opportunities to implement and verify our method in operational monitoring testbeds.

Index Terms—Environmental observation and monitoring, autonomous sensors deployment, smart sensing network automation, collision-free motion algorithm, graph theory, vertex coloring.

I. INTRODUCTION

FOR decades, researchers have been seeking ways to deploy useful networks for realizing the goal of smart living. Wireless sensor networks (WSNs) have been thriving and attracting significant attention thanks to the advancements of micro-electromechanical system (MEMS), sensing technology, and wireless communication. A WSN is widely used for habitat and environmental monitoring, surveillance (camera) networks, medical application, agricultural assistance, and as solutions to military problems [1]–[4]. For monitoring and surveillance applications, sufficient sensing coverage is essential for a WSN to operate successfully. Traditionally maintaining such a network is labor-intensive, often requiring additional staff charged with monitoring the sites daily and responding to inquiries. We consider a smart sensing network which has the ability to self-deploy mobile sensors and react with proper responses in an automated manner. With the growing prevalence of wireless mobile sensors, automated sensors deployment (in lieu of manual placement) has become practical and feasible [5]. In our prior research [6], [7], we developed coverage-aware deployment protocols that calculate best monitoring positions (goals) for all sensors. However, the problem of actually coordinating a fleet of

Manuscript received 19 October 2021; accepted 17 December 2021. Date of publication 7 January 2022; date of current version 13 October 2022. This article was recommended for publication by Associate Editor M. Franceschelli and Editor B. Vogel-Heuser upon evaluation of the reviewers' comments. This work was supported by the Ministry of Science and Technology (MOST) of Taiwan under Grant 108-2221-E-009-028, Grant 108-2218-E-009-012 (Core technologies and application developments for M2M communication systems), and Grant 110-2634-F-009-019 (Universal core platform and vision enhancement based on AVIoT). (Corresponding author: Ting-Yu Lin.)

The authors are with the College of Electrical and Computer Engineering, National Yang Ming Chiao Tung University, Hsinchu 30010, Taiwan (e-mail: tingyoyo@nycu.edu.tw).

This article has supplementary material provided by the authors and color versions of one or more figures available at <https://doi.org/10.1109/TASE.2021.3138198>.

Digital Object Identifier 10.1109/TASE.2021.3138198

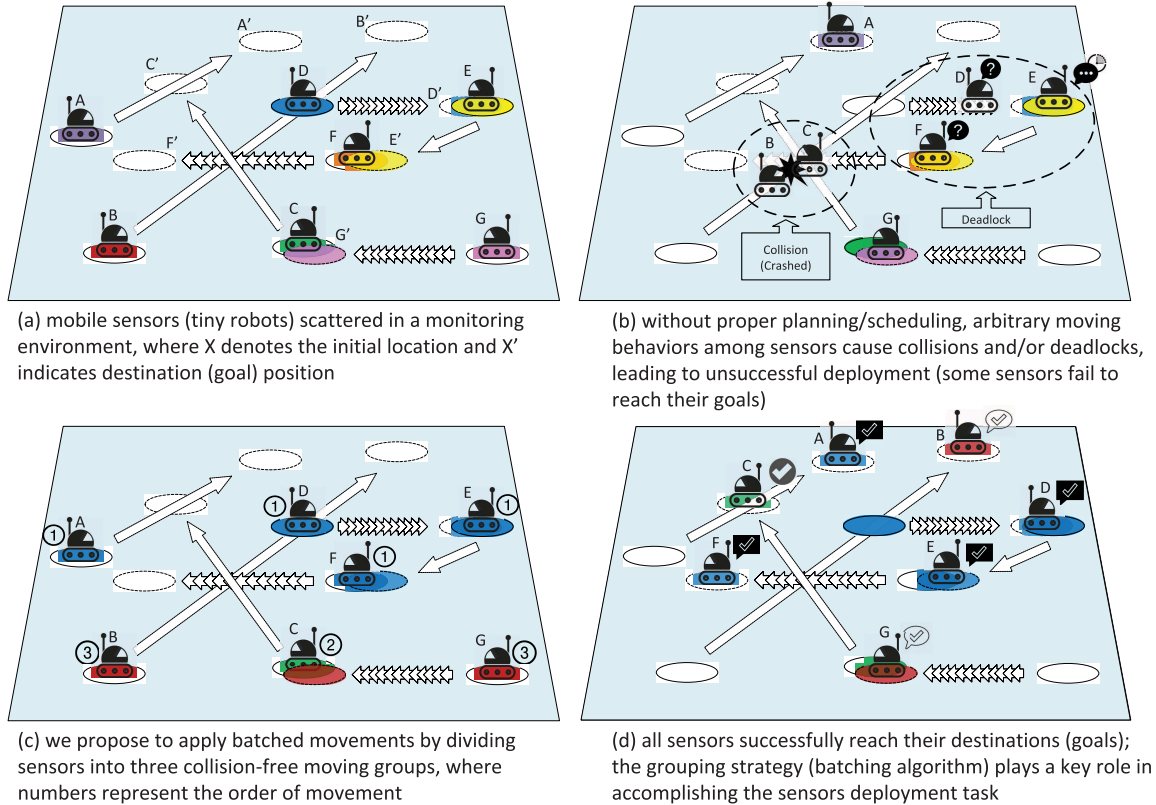


Fig. 1. Illustration of the sensors deployment problem in a smart sensing environment with mobile sensors each moving toward a specific designated (goal) position; grouping strategy (batching algorithm) is the key to a successful deployment.

autonomous sensors and directing them from their random initials to goals without collisions has been left unsolved. In this follow-up research, we emphasize on resolving this problem by designing actual collision-free motion strategies for realizing a practical sensing environment.

Fig. 1 (a) epitomizes an open-space sensing environment (such as an ocean floor), where each mobile sensor (tiny robot) has to move from its initial (departure) location to the goal (destination) position. Without careful path scheduling, those tiny robots are likely to run into each other, resulting in an unsuccessful deployment. As depicted in Fig. 1 (b), robots B and C collide at the intersection of their moving paths, further blocking the travel path of robot F to its goal position. Meanwhile, robots E and D are prevented from reaching their destinations because of the goal positions being occupied by robots F and E respectively. In fact, robots D, E, and F form a deadlock situation, which remains unsolved in some previous works [8]–[10]. Such collision and deadlock problems motivate us to design collision- and deadlock-free motion algorithms for ensuring sensors goal reachability.

We investigate the path scheduling problem and propose to apply *batched movements* by dividing robots into several moving groups. One possible grouping strategy, as illustrated in Fig. 1 (c), is to instruct robots A, D, E, F (group #1) move first, then robot C (group #2), followed by robots B and G (group #3). By assigning robots in proper moving groups (batches), Fig. 1 (d) displays a successful deployment with all robots now arriving at their correct goal positions.

Evidently, the grouping strategy (batching algorithm) plays a key role in helping robots reach their intended goals safely without collisions. Designing collision-free batching algorithms is non-trivial, which requires carefully analyzing all collision cases while considering computation and traveling latency (attempting to minimize the number of moving batches so as to accomplish the deployment task in a timely manner).

In this paper, we regard all sensors as dynamic objects (tiny robots) and develop our collision-free motion algorithms based on batched movements, to address the sensors deployment problem. Our proposed batching algorithms incur little computation latency, moderate energy consumption, and ensure 100% sensors goal reachability. The remainder of this paper is organized as follows. Section II prepares the readers with related work and summarizes our unique contributions. We present our CFPP, CFMA, and wCFMA protocol details in Section III, Section IV, and Section V respectively. Section VI provides performance comparisons in terms of goals reaching success probability, deployment latency, and energy consumption. Finally, we conclude the paper in Section VII. An illustrative video that shows the operations of our current prototype can be found at [11].

II. RELATED WORK

Traditional path-planning methods in the area of multi-robot systems can be generally classified into three categories: roadmap-based, performing cell decomposition, and applying the concept of artificial potential field, according to the authors

in [12] and [13]. The roadmap-based methods construct visibility graphs to assist in the path-planning process [14]–[16], in which the resultant moving paths may touch fixed obstacles at the vertices or even edges that are considered unsafe. To address this drawback, approaches based on cell decomposition by computing Voronoi diagrams have been introduced to perform path scheduling in a more precise way at the cost of increased computation latency [13]. The third category of path-planning approaches models the obstacles and targets as electrostatic charges interacting with each other, creating a potential field [17]–[19]. The obstacle exerts a repulsive force while the target location has an attractive effect on the robot position. This method enables path planning to be completed in real time. However, as only local properties are considered, the robots may get trapped at local minima or aimless oscillation without reaching their goals. The aforementioned approaches assume one or more fixed obstacles in a multi-robot system, which differs from our mobile sensors networking scene. In this paper, we **investigate a sensing system where all sensors are mobile and considered as dynamic objects** (rather than fixed obstacles).¹

Another approach in finding shortest moving paths is based on A* search [20]. A* works by expanding the vertices inside the map and searching for the nodes with lower estimated distance to the goal. The ability of this A* search algorithm to be manipulated in many ways leads to the development of many path planning techniques [9], [10], [21]. Approaches based on adapted GA (genetic algorithm) have also been proposed to address the problem [22], [23]. A comprehensive study of path-planning algorithms comparison can be found in [24]. In recent years, special-purpose research also puts interest in using autonomous underwater vehicles (AUVs) for marine geoscience studies in dangerous environments [25], [26]. In these settings, robots are deployed in an obstacle-free space moving/flying from initial to goal positions to perform certain tasks. However, those approaches require high computation complexity and deal with independent robots navigating an environment rather than coordinating a fleet of robots distributed to perform the monitoring task. In this paper, we consider a smart sensing environment² with **environmental monitoring application** to automatically coordinate a fleet of autonomous sensors (tiny robots) to move toward their designated goals (destinations). To enable a smart environment with sufficient sensing coverage, the **success rate of goals reachability is crucial** in our target application.

Three more closely related path-planning methods to our envisioned scenario that also deal with dynamic objects include ADO [8], Super A* [9], and M* [10]. In ADO, all robots are prioritized and equipped with omnidirectional cameras (visual sensors) to perform real-time path-planning computations. ADO suffers from the deadlock problem in which some robots are unable to reach their destinations (goal positions). In addition, extra (non-negligible) energy

consumption is required due to the constant usage of visual sensors for path calculations in ADO. On the other hand, both M* and Super A* modify the classical A* search algorithm [20] by taking environmental changes into account while robots move. M* additionally introduces subdimensional expansion, a framework to compute individual policy for each robot first (neglecting the presence of other robots) and then joint policy path is employed using an underlying planner (based on A* search) to find optimal paths in the search space. Another concept of backpropagation is introduced such that the search space is only expanded where necessary (when the planner encounters robot-robot collision). Super A* and M* successfully resolve some deadlock cases that ADO fails to handle. However, the triangular deadlock problem in ADO remains unsolved in Super A* and M*. Moreover, extensive computation latency is required in Super A* and M* due to the high time-complexity nature of the A* search algorithm. In our proposed motion mechanisms, **computation latency, execution (moving) time, and energy consumption are also evaluated** so that the deployment task can be accomplished in a timely manner while conserving energy resources on sensors.

A. Our Contributions

In this research, we provide an in-depth analysis on the sensors moving path scheduling problem and propose corresponding motion algorithms. Below we summarize our unique contributions.

First, we carefully classify all possible collision cases and devise a collision-free path planning (CFPP) algorithm (Section III) based on batched movements to guarantee all the mobile sensors reach their goals (destinations) with 100% success rate. The CFPP performance demonstrates the feasibility of resolving collisions and deadlocks by moving in batches.³ Second, to accelerate the deployment process (by possibly reducing the number of required moving groups), we observe the grouping (batching) strategy is similar to the coloring procedure in graph theory. By constructing a conflict graph, we model the collision-free path scheduling as the well-known *k-coloring* problem, from which we reduce to our *k-batching* problem (determining the minimum number of required batches for a successful deployment) and prove its NP completeness. Since the *k-batching* problem is intractable, we develop CFMA (collision-free motion algorithm), a simple yet effective batching (coloring) heuristic mechanism, to approximate the optimal solution (refer to Section IV for proof and algorithm details). Third, to further enhance CFMA

³The preliminary results of CFPP have been reported and published in a previous conference paper [27]. However, the prior work did not study the aspects of minimizing the number of moving groups (batches) or shortening the total deployment time, and lacks detailed results on sensors goal reachability with effectively reduced computation and execution (travel) latency. Also, the aspects of moving energy consumption and different initial/final sensors configurations were not extensively evaluated in the prior work. In this journal paper, we identify several inefficiencies of CFPP and propose two enhanced collision-free motion algorithms, namely CFMA and wCFMA, for accelerating the deployment process by possibly reducing the number of required moving groups (batches) and shortening the total travel (moving) time spent by all sensors. Though inspired from CFPP and also based on batched movements, CFMA and wCFMA are two new path scheduling approaches that operate quite differently from CFPP.

¹In our approach, instead of bypassing obstacles, we adopt batched movements to ensure those tiny robots can move along a straight line.

²We define the smart sensing environment as a sensing system which has the ability to sense the environment and react with proper responses in an automated manner.

performance, we propose wCFMA (weighted collision-free motion algorithm), which reduces sensors execution (moving) time by imposing smaller variance (difference) among travel distances within the same batch (moving group). We regard this as the *weighted-batching* problem and successfully prove its NP-hardness by reducing from the *weighted-coloring* problem. Consequently, we design wCFMA, a weight-ordered batching (coloring) heuristic mechanism, to approximate the min-weighted solution (refer to Section V for proof and algorithm details). Our wCFMA intends to generate batches with *shortened* total execution (moving) time in the example cases, thus accomplishing the sensors deployment task earlier (compared to CFPP and CFMA). The proposed CFMA and wCFMA grouping strategies can also be nicely applied to other similar research problems modeled in the form of graph coloring (using minimum number of colors) or weighted coloring (minimizing sum of total group weights). Furthermore, we have also implemented a proof-of-concept prototype based on moving robots (LEGO MINDSTORMS NXT 9797 [28]), network camera M30 [29], and Tibbo embedded systems [30] to demonstrate the operations of our proposed motion algorithm in a real system [11].

Specifically, in this work, we propose a new perspective of pre-scheduled batched movements to bring down the sensors deployment cost. There has been a growing recognition of the need to keep spending (cost spent on hardware and software) under control, even as environmental monitoring projects become increasingly complex. That is where our work comes in by introducing a practical solution. Existing path-planning methods for multi-robot systems are computation-intensive and/or energy-consuming. Another major concern is **those previous approaches do not ensure all robots reach their goal points** (as demonstrated in Section VI-A, Fig. 10 and Fig. 11). Sensors goal reachability is critical in realizing an effective environmental monitoring application.

Our theoretical contributions include interpretation of multi-robot path planning as the coloring problem on a conflict graph, proving NP-completeness of the *k-batching* problem (Theorem 2), and verifying NP-hardness of the *weighted-batching* problem (Theorem 3). Please note that we do not directly use graph theory to solve our problem. By proving NP-completeness/hardness, we show that there do not exist efficient (polynomial-time) algorithms to determine optimal moving sets. Therefore, we develop CFMA and wCFMA, suboptimal yet efficient batching solutions, to address the problems under study. **Existing coloring algorithms only take care of adjacent-coloring rule**, ensuring adjacent nodes colored differently, without considering coloring sequence (robots movement order). However, in our target application, movement order is important in resolving robot-robot collisions or deadlocks. To address these new challenges, in CFMA and wCFMA, we define collision Cases II and III (Fig. 5) to impose extra restrictions in addition to the valid adjacent-coloring rule. Adjusting colors for special conflict Cases II and III can be found in the last segment of our CFMA Algorithm. Unlike previous path-planning approaches, our motion algorithms advance related research by proposing

computation-efficient and robust path-scheduling methods that ensure all sensors reach their goals successfully.

III. PATH PLANNING STRATEGY

To enable actual sensors deployment, a collision-free moving path scheduling is essential, so that mobile sensors can reach their destinations without colliding with each other. However, the scheduling strategy is non-trivial since various collision cases need be systematically classified and handled/resolved in different ways.

A. Preliminaries

Assume that the sensor volume is neglected⁴ and each sensor (tiny robot) can be regarded as a *moving point* on a 2D plane, while every moving path (traveled by a sensor) formulated as a *line*. Suppose no two moving paths share the same line (i.e., no path lies in the sub-path of another).⁵ We identify the collision cases based on the following geometric theorem.

Theorem 1: With respect to the line $ax + by + c = 0$ on a 2D plane, points $Q_1(x_1, y_1)$ and $Q_2(x_2, y_2)$ fall on the same side if $(ax_1 + by_1 + c)(ax_2 + by_2 + c) > 0$, on different sides if $(ax_1 + by_1 + c)(ax_2 + by_2 + c) < 0$, while one or both reside(s) exactly on the line if $(ax_1 + by_1 + c)(ax_2 + by_2 + c) = 0$.

For an arbitrary sensor s_i departing from point p_i with coordinate (x_i, y_i) to point p'_i with coordinate (x'_i, y'_i) , the moving path can be formulated as a line, denoted as L_i . Similarly, the moving path of another sensor s_j is given as L_j . Define p_{ij} as the *intersection point* of lines L_i and L_j , which can be easily obtained by solving the two line equations. According to Theorem 1, we can now classify five possible intersection (collision) cases for any two sensors s_i and s_j , as illustrated in Fig. 2, where $d(p_i, p_{ij})$ and $d(p_j, p_{ij})$ represent the Euclidean distances from p_i to p_{ij} and from p_j to p_{ij} . **Case I** shows the case in which points p_i and p'_i fall on different sides of line L_j , whereas points p_j and p'_j fall on different sides of line L_i as well. In **Case II**, the departure point p_j of sensor s_j gets in the way of the moving path of s_i , while in **Case IV**, on the contrary, the departure point p_i of sensor s_i blocks the moving path of s_j . **Case III** draws the condition in which the destination point p'_j of sensor s_j lies on the moving path of s_i , whereas **Case V**, in contrast, displays the condition that destination point p'_i of sensor s_i falls on the moving path of s_j .

B. Collision-Free Path Planning (CFPP)

Given the five potential collision (intersection) cases caused by any two moving paths, we establish a colliding set C_i for each sensor, which includes all sensors whose moving paths intersect with that of s_i . Instead of performing one-time physical movements, we propose to use *batched movements*

⁴For collision-free path scheduling methods that consider the robot (sensor) volume, please refer to our previous work [31], [32].

⁵To ensure that the geometric formulation can work properly, our algorithm will first examine if any two moving paths share the same line. If this is really the case, one of the involved mobile sensors' initial (departure) location will be slightly adjusted to resolve the sub-path problem.

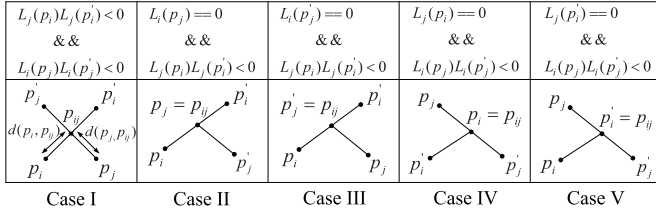


Fig. 2. Possible intersection (collision) cases generated by moving paths of any two sensors s_i and s_j , where p_i (p_j) denotes the original position of s_i (s_j) and p'_i (p'_j) indicates the physical movement destination for sensor s_i (s_j).

TABLE I
SUMMARY OF NOTATIONS USED IN CFPP

Notation	Description
C_i	Set of potential colliding sensors against s_i
$order_i$	Moving order of s_i where $order_i = C_i $
fix_order_i	Indicates the $order$ value of s_i has been henceforth fixed
$dirty_i$	Indicates s_i has been processed in the current round
$tflag_i$	Indicates s_i is allowed to move in the current round
$mflag_i$	Indicates s_i has moved from p_i to p'_i
M_{min_order}	Set of sensors with minimum $order$ value (being scheduled to move in the current round)

such that the scheduling complexity can be reduced at the expenses of increased moving latency. Define $order_i$ as the cardinality of set C_i ($order_i = |C_i|$) for sensor s_i , indicating its moving order. We start from performing movements for sensors with the smallest $order$ value. All sensors with the currently minimum (smallest) $order$ value are contained in set M_{min_order} . Intuitively, sensors with $order$ value of zero can move simultaneously since no other sensors pose potential colliding sources to them. For any sensor s_i with non-zero $order_i$ value, potential colliding conditions (on per node-pair basis) caused by all members in its C_i set should be analyzed and handled case by case. Specifically, all sensors are divided into moving groups (batches) based on their $order$ values and processed round by round (batch by batch). Sensors in set M_{min_order} are evaluated in the same round. The evaluation and processing details will be provided later in this section. After the evaluations, a subset of M_{min_order} (or probably the whole M_{min_order} set) is determined and all sensors included in the subset are allowed to move simultaneously in the current round. For sensor s_i that has been evaluated and permitted to move, the $tflag_i$ is set *true*, indicating its moving intention. Once the physical movement has been successfully performed by sensor s_i , moving flag $mflag_i$ is set *true* and s_i is removed from the consideration list. All $order$ values for the remaining sensors (physical movements not performed yet) should be refreshed, and the batched scheduling procedure starts over accordingly.

Now, we elaborate on the evaluation procedures for determining a set of movable sensors in a single round (batch). Based on the idea of batched movements, we regard all sensors with the currently minimum $order$ value as a potential moving batch and include them in set M_{min_order} . We then analyze all members in set M_{min_order} one by one to determine their moving possibilities. In our design, we start the evaluation from sensor with the smallest ID, say s_1 , and identify all possible collision cases caused by members in its colliding set C_1 . For any two sensors s_i and s_j with moving orders

Algorithm 1 Collision-free Path Planning (CFPP)

```

include all sensors in set  $S$ ;
establish set  $C_i$  for  $\forall s_i \in S$ ; //  $i = 1, \dots, n$ 
evaluate  $order_i$  for  $\forall s_i \in S$ ;
clear  $fix\_order_i, dirty_i, tflag_i, mflag_i$  for  $\forall s_i \in S$ ; // all set to false
while ( $S$  !empty) do
  re-establish set  $C_i$  for  $\forall s_i \in S$ ;
  re-evaluate  $order_i$  for  $\forall s_i \in S$  with  $fix\_order_i == false$ ;
  reset  $T_i = 0, V_i = V, dirty_i = false, tflag_i = false$  for  $\forall s_i \in S$ ;
  include all  $s_i$  with the minimum  $order_i$  value into the  $M_{min\_order}$  set;
  for (each  $s_i \in M_{min\_order}$ ) do
    set  $tflag_i = true$ ;
    for (each  $s_j \in C_i$ ) do
      classify the intersection (collision) case for  $s_i$  and  $s_j$ ;
      switch (case)
        Case D-II: do Action D-II;
        Case D-V: do Action D-V;
        Case S-I: do Action S-I;
        Case S-II: do Action S-II;
        Case S-III: do Action S-III;
        Case S-IV: do Action S-IV;
        Case S-V: do Action S-V;
      end for
    end for
  perform simultaneous physical movements for  $\forall s_i$  with  $tflag_i == true$ ;
  set  $mflag_i = true$  for such sensor  $s_i$ ; // physical movement performed
  remove all  $s_i$  with  $mflag_i == true$  from sensors set  $S$ ;
end while

```

(to be continued in next Algorithm segment)

Algorithm 1 Collision-Free Path Planning (CFPP) (cont.)

```

procedure Action D-II
  slightly adjust location of  $s_j$  from  $p_j$  (original) to  $p_j$  (adjusted);
procedure Action D-V
  set  $order_i = order_j + 1$ ; set  $fix\_order_i = true$ ;
  invoke Action Deferred ( $s_i$ );
procedure Action S-I
  if  $T_i + t_{p_i \rightarrow p_{ij}} = T_j + t_{p_j \rightarrow p_{ij}}$  then
    if  $dirty_j == false$  then set  $T_j = T_j + \Delta t$ ;  $dirty_j = true$ ;
    else set  $order_i = order_i + 1$ ; invoke Action Deferred ( $s_i$ );
procedure Action S-II
  if  $T_i + t_{p_i \rightarrow p_{ij}} \leq T_j$  then
    slightly adjust location of  $s_j$  from  $p_j$  (original) to  $p_j$  (adjusted);
procedure Action S-III
  if  $T_i + t_{p_i \rightarrow p_{ij}} \geq T_j + t_{p_j \rightarrow p_{ij}}$  then
    if  $dirty_j == false$  then
      set  $T_j = T_i + (t_{p_i \rightarrow p_{ij}} - t_{p_j \rightarrow p_{ij}}) + \Delta t$ ; set  $dirty_j = true$ ;
    else set  $order_j = order_j + 1$ ; invoke Action Deferred ( $s_j$ );
procedure Action S-IV
  if  $T_i \geq T_j + t_{p_j \rightarrow p_{ij}}$  then
    if  $dirty_j == false$  then
      set  $T_j = T_i - t_{p_j \rightarrow p_{ij}} + \Delta t$ ; set  $dirty_j = true$ ;
    else slightly adjust location of  $s_i$  from  $p_i$  (original) to  $p_i$  (adjusted);
procedure Action S-V
  if  $T_i + t_{p_i \rightarrow p_{ij}} \leq T_j + t_{p_j \rightarrow p_{ij}}$  then
    if  $dirty_j == false$  then
      set  $V_j = \frac{V_i d(p_i, p_{ij})}{d(p_i, p_{ij}) + V_i (T_i - T_j)} + \Delta v$ ; set  $dirty_j = true$ ;
    if  $V_j > V_{max}$  then
      set  $order_i = order_i + 1$ ; invoke Action Deferred ( $s_i$ );
    else set  $order_i = order_i + 1$ ; invoke Action Deferred ( $s_i$ );
procedure Action Deferred ( $s_i$ )
  set  $tflag_i = false$ ;
  do necessary slight adjustment of  $s_i$ 's departure location to resolve
  moving path blocking possibly caused by this not-moving decision;

```

(end of Algorithm 1)

$order_i$ and $order_j$, the previously five collision cases can be further classified into ten cases according to the relationship

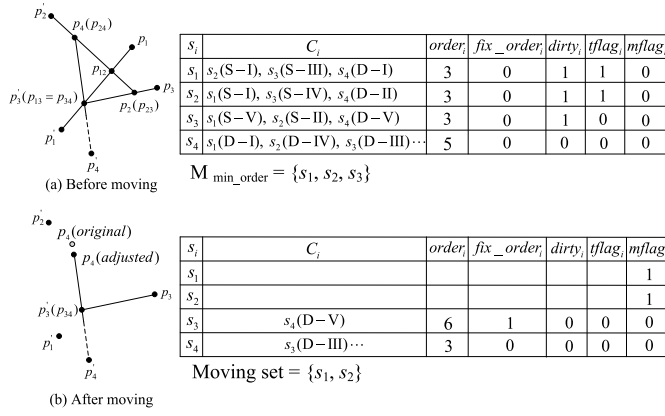


Fig. 3. Every sensor s_i in the potential moving set M_{min_order} should be analyzed by identifying its intersection (collision) relationship with each member in C_i , in which intersection cases D-II, D-V, S-I, S-II, S-III, S-IV, and S-V require further consideration/processing, before including s_i into the moving set (allowed to move in the current round).

of $order_i$ and $order_j$. Suppose $s_i \in M_{min_order}$, $s_j \in C_i$, and $order_i = order_j$, we term the five collision cases as **Case S-I**, **Case S-II**, **Case S-III**, **Case S-IV**, and **Case S-V**, where 'S' indicates that sensors s_i and s_j are potentially scheduled to move in the "same" round due to equal $order$ value. On the other hand, if $order_i < order_j$ (note that $order_i > order_j$ is not possible since $s_i \in M_{min_order}$), we define another five collision cases as **Case D-I**, **Case D-II**, **Case D-III**, **Case D-IV**, and **Case D-V**, where 'D' means s_i and s_j are potentially scheduled to move in "different" rounds due to their unequal $order$ values. In each potential collision case, on detecting a colliding possibility, s_i tries to resolve the collision by adjusting/prolonging the waiting time T_j or increasing the moving speed V_j of sensor s_j . Originally all waiting times are set to zero, and moving speeds all set at a constant velocity V . If the adjustment (on either waiting time or moving speed) is successful, the colliding possibility is eliminated and s_i moves on to evaluate collision cases with other members in C_i . To avoid repeated adjustments on a single sensor, in our design, each sensor is allowed to be adjusted (either on waiting time or moving velocity) *once*. In addition, s_i itself cannot be adjusted by other sensors in set M_{min_order} that are evaluated *after* it, if s_i is indeed scheduled to move in the current round. We keep track of the adjustment possibility for sensor s_i by the $dirty_i$ bit, implying adjustable if set *false* and not adjustable if set *true*. When s_i intends to resolve a collision by adjusting another sensor with $dirty$ bit set *true*, the adjustment is prohibited and s_i is not allowed to move in the current round ($tflag_i$ set to *false*), since the collision remains. Only when all members in C_i with various colliding possibilities are all resolved can sensor s_i be included into the movable set and perform physical movement. Upon receiving the moving instruction from the clusterhead, s_i waits for T_i (possibly adjusted) and then moves with speed V_i (possibly adjusted). In our route scheduling strategy, we try to include *as many sensors as possible* to move simultaneously in the same round (batch).

For each of the ten collision cases identified, we define corresponding actions (**Action D-I**, **Action D-II**, \dots , **Action**

S-I, **Action S-II**, \dots) to evaluate respective case and perform necessary adjustments. If colliding possibility remains due to unsuccessful adjustment, physical movement by sensor s_i is not allowed and should be deferred. Thus we additionally define **Action Deferred** to perform corresponding operations. Note that in **Case D-I**, **Case D-III**, and **Case D-IV**, no action is needed since s_i is scheduled to move before s_j in different rounds (no collision is likely to happen in the three cases despite intersection exists between the two moving paths). For the rest of seven cases, we describe the evaluation principles exercised by respective action as follows (detailed operations are available in Algorithm 2, Section III-C).

Action D-II In this case, since s_j gets in the way of s_i 's moving path, the clusterhead instructs s_j to slightly adjust its location along line $\overrightarrow{p_j p'_j}$ to avoid collision. Assume the location adjustment is small enough to have no effect on other moving paths.

Action D-V Sensor s_i is not allowed to move, for its destination point p'_i will block the moving path of s_j in a later round. In this case, the moving order of s_i should be set larger than that of s_j ($order_i = order_j + 1$) to postpone s_i 's physical movement after s_j . In addition, a fix_order_i flag should be set *true*, indicating no updates on $order_i$ will be performed in later rounds to ensure the delayed movement after s_j , and then **Action Deferred** is invoked for s_i .

Action S-I Define the traveling time from p_i to the intersection point p_{ij} as $t_{p_i \rightarrow p_{ij}}$ (obtained from available $d(p_i, p_{ij})$ and V_i), the clusterhead evaluates if $T_i + t_{p_i \rightarrow p_{ij}} = T_j + t_{p_j \rightarrow p_{ij}}$, where T_i and T_j are the waiting times of s_i and s_j as defined earlier. If equality holds, a collision at the intersection is expected, and the waiting time T_j of s_j should be increased by a small amount of Δt to avoid the collision. However, in case s_j has already been processed with $dirty_j$ set *true*, the adjustment is prohibited and s_i is not allowed to move in the current round. Consequently, moving order of s_i is increased ($order_i = order_i + 1$) and **Action Deferred** is invoked for s_i .

Action S-II If s_i reaches the intersection point p_{ij} no later than s_j 's departure time, the clusterhead should instruct s_j to slightly adjust its location along line $\overrightarrow{p_j p'_j}$ to avoid collision.

Action S-III If s_j reaches the intersection point p_{ij} no later than s_i , the destination point p'_j of s_j will block the moving path of s_i . In this case, the clusterhead should instruct s_j to increase its waiting time T_j by setting $T_j = T_i + (t_{p_i \rightarrow p_{ij}} - t_{p_j \rightarrow p_{ij}}) + \Delta t$ to ensure the delayed arrival of s_j at p_{ij} (p'_j). If the adjustment of T_j is not successful due to a *true* flag of $dirty_j$, then s_j is not allowed to move in the current round. Consequently, moving order of s_j is increased ($order_j = order_j + 1$) and **Action Deferred** is invoked for s_j .

Action S-IV If s_j reaches the intersection point p_{ij} no later than s_i 's departure time, the clusterhead should increase the waiting time of s_j by setting $T_j = T_i - t_{p_j \rightarrow p_{ij}} + \Delta t$. In case the adjustment is not allowed due to a *true* value of $dirty_j$, the clusterhead instructs s_i to slightly adjust its location along line $\overrightarrow{p_i p'_i}$ to avoid collision.

Action S-V If s_i reaches the intersection point p_{ij} no later than s_j , the destination point p'_i of s_i will block the

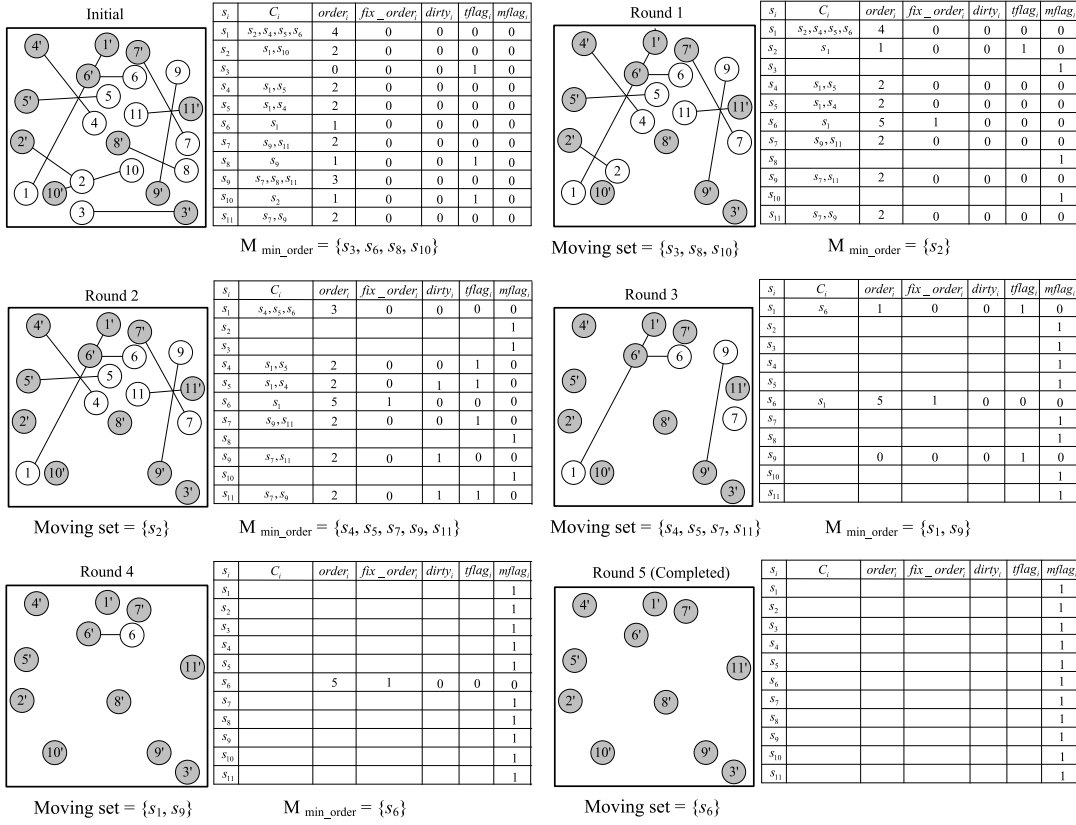


Fig. 4. Example illustrating the operations of CFPP grouping strategy for sensors batched movements.

moving path of s_j . In this case, the clusterhead should instruct s_j to increase its moving speed V_j by setting $V_j = \frac{V_i \cdot d(p_j, p_{ij})}{d(p_i, p_{ij}) + V_i(T_i - T_j)} + \Delta v$, where Δv is a small amount of speed increment to ensure s_j 's earlier arrival at p_{ij} (p_j') than s_i . However, if the adjusted V_j is larger than the maximum possible moving speed V_{max} or the adjustment of V_j is prohibited due to a *true* value of $dirty_j$, then s_i is not allowed to move in the current round. Moving order of s_i is increased ($order_i = order_i + 1$) and **Action Deferred** is invoked for s_i .

Action Deferred Since s_i (s_j) is not allowed to move in the current round, $tflag_i$ ($tflag_j$) is set *false*. In addition, the clusterhead should confirm if this not-moving decision leads to moving path blocking of any sensor in M_{min_order} set that is already allowed to move in the current round (with $tflag$ set *true*), and do necessary slight location adjustment to resolve the blocking.

Fig. 3 illustrates a snapshot of the CFPP operations. Note that s_4 has more intersections with other sensors, which are not shown in the figure (omitted for brevity). In the current round, potential moving set M_{min_order} includes s_1 , s_2 , and s_3 , all having the currently smallest *order* value of 3. For s_1 , colliding conditions caused by all members in C_1 are analyzed and handled case by case. In this example, since s_1 and s_2 are evaluated to reach intersection p_{12} simultaneously, the clusterhead adjusts the waiting time of s_2 by setting $T_2 = T_2 + \Delta t$ to resolve the collision. Next, since s_3 is found to reach intersection p_{13} earlier than s_1 , blocking s_1 's moving path, the clusterhead instructs s_3 to increase its waiting time by setting $T_3 = T_1 + (t_{p_1 \rightarrow p_{13}} - t_{p_3 \rightarrow p_{13}}) + \Delta t$. As to s_4

(scheduled to move in a later round), no action is required since no collision is likely to happen between s_1 and s_4 . Consequently, the clusterhead includes s_1 into the moving set. Similar operations apply to s_2 . In our example, s_2 has no colliding possibilities with s_1 and s_3 . However, since the departure location p_4 of s_4 blocks s_2 's moving path, the clusterhead instructs s_4 to slightly move from p_4 (original) to p_4 (adjusted), as shown in Fig. 3 (b). As a result, s_2 is also included into the moving set. For s_3 , in our example, both s_1 and s_2 do not pose colliding sources to s_3 . Unfortunately, since the destination point p_3' of s_3 will block the moving path of s_4 in a future round, s_3 is not allowed to move before s_4 (not included into the moving set), and $order_3$ should be updated to 6 ($order_4 + 1$) with fix_order_3 set *true*. After the evaluations, sensors included in the moving set (i.e., s_1 and s_2) perform physical movements simultaneously, and $order_4$ and set C_4 are updated accordingly.

Table I summarizes the notations used in CFPP, and Algorithm 2 provides the pseudocode for CFPP operations.

C. CFPP Summary

A running example illustrating the route scheduling procedures is available in Fig. 4. Note that in Round 3 of this example, s_9 is excluded from the moving set due to an unsuccessful adjustment of s_{11} 's waiting time (since T_{11} has been adjusted by the clusterhead to resolve collision with s_7 and can only be adjusted *once* according to the scheduling principles adopted by CFPP). After the clusterhead decides that s_9 is not allowed to move in the current round, s_9 no

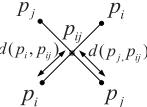
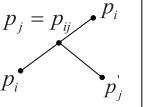
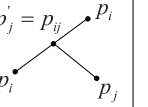
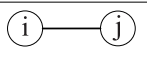
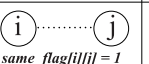
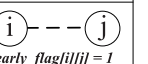
$L_j(p_i)L_j(p_i') < 0$ && $L_i(p_j)L_i(p_j') < 0$	$L_i(p_j) = 0$ && $L_j(p_i)L_j(p_i') < 0$	$L_i(p_i') = 0$ && $L_j(p_i)L_j(p_i') < 0$
		
Case I	Case II	Case III
	 $same_flag[i][j] = 1$	 $early_flag[i][j] = 1$

Fig. 5. Possible intersection (collision) cases and corresponding conflict links (edges to be used in the conflict graph).

longer poses as a colliding source to s_{11} . Consequently, s_{11} can be included into the current moving set (Round 3).

IV. ENHANCED PATH SCHEDULING

While CFPP operates correctly, yet the collision cases classification and corresponding actions pose as a tedious work. Nonetheless, we gain useful insights from the design process, which prompts us to wonder whether or not CFPP produces the minimum number of moving sets (batches). In other words, does there exist an efficient path scheduling algorithm that guarantees *the fewest number of moving batches* to minimize the overall deployment time? We attempt to answer this intriguing question and possibly enhance our path scheduling (grouping) strategy as follows.

A. Problem Formulation

Observing the example in Fig. 4, we identify two main problems in CFPP operations. First, scheduling *min_order* sensors to move early may not be a good idea as those sensors actually bear lesser colliding conflict to others and could be handled later. Second, sensors with same *order* value are not necessarily able to move in the same round. For instance, s_4 and s_5 are on the collision course with each other and should not be scheduled in same moving set despite having the same *order* value ($order_4 = order_5 = 2$). On the other hand, sensors with different *order* values are possible to move in the same round without collisions. For instance, s_2 , s_6 and s_9 are able to deploy simultaneously despite having different *order* values ($order_2 = 2$, $order_6 = 1$, and $order_9 = 3$). However, CFPP conservatively schedules s_2 (Round 2) and s_6 (Round 5) to move alone respectively, as shown in Fig. 4. For a better scheduling, whether or not two sensors can be assigned in the same moving set should depend on their mutual conflict relationship, rather than inspecting their *order* values alone and separately.

Inspired from this observation, we aim to resolve the CFPP inefficiency by first constructing a *conflict graph* to specifically identify the *conflict relationship* between each pair of sensors in their moving paths.

1) *Construction of a Conflict Graph*: We re-organize the collision cases in CFPP into three possible intersection categories (collision types), as shown in Fig. 5. **Case I** plots the

moving paths of s_i and s_j collide at the path intersection.⁶ **Case II** illustrates the departure (initial) location of s_j gets in the way of s_i 's moving path whereas **Case III** draws the condition in which the destination (goal) position of s_j lies on the moving path of s_i . The intersection cases sufficiently reveal three *collision types*, allowing us to explicitly define *conflict links* between sensors.

For a given set of n sensors and moving paths, we construct an undirected conflict graph $G' = (S, L)$ where $S = \{s_1, s_2, \dots, s_n\}$ includes all sensor nodes and edge set L contains all *conflict links* between any two sensors. That is, $(s_i, s_j) \in L$ iff moving paths of s_i and s_j encounter one of the three collision types (**Case I**, **Case II**, or **Case III**).

Furthermore, back to Fig. 5, since **Case I** indicates the two sensors will reach the intersection at the same time, this sensor pair should move in different rounds. We denote this conflict type by a **solid link** in our conflict graph. For **Case II**, denoted by a **dotted link**, s_j should move *no later than* s_i , meaning that s_j must move earlier than s_i or be scheduled in the same round as s_i to resolve the conflict. By setting up $same_flag[i][j]$ we are able to identify this special case and possibly schedule s_i and s_j in the same group to reduce the number of moving sets (batches). In **Case III**, we should instruct s_i to move earlier than s_j , as such $early_flag[i][j]$ must be set, drawn as a **dashed link** to imply this conflict type in our conflict graph. Algorithm 5 (the last segment) provides a detailed description on how to handle special **Cases II** and **III**. At this moment, we consider a conflict graph containing conflict links with *no distinction* between different collision types.

A constructed conflict graph concisely defines the collision relationship between each pair of sensors' moving paths, which helps in both theoretically analyzing the path scheduling problem and our algorithm design in Section IV-B.

2) *NP Completeness Proof*: Now, we return to the interesting question of how many moving sets (batches) are just sufficient for a successful collision-free sensors deployment with minimum execution time. For a given constructed conflict graph, our path scheduling (grouping) strategy intends to divide the sensors into different groups (batches) such that all sensors scheduled in the same group (moving round) do not collide with each other. Specifically, our grouping strategy attempts to make sure *adjacent* nodes (connected with one of the three conflict types defined in Section IV-A1) in the conflict graph *do not move simultaneously*, so collisions can be effectively eliminated. This design rationale reminds us of the well-known vertex-coloring problem in graph theory, which deals with *coloring adjacent vertices using different colors*. A valid coloring of an undirected graph $G = (V, E)$ is an assignment of colors to the vertices such that each vertex is assigned one color and no two adjacent vertices share the same color. The *k-coloring* problem has developed from the *3-coloring* decision (yes/no) problem to one that determines the minimum number of colors needed to color a graph [33], [34], as formally defined below.

⁶We consider practical sensors of different moving speeds. For sensors s_i and s_j with travel velocity v_i and v_j , **Case I** collision case holds iff $d(p_i, p_{ij})/v_i = d(p_j, p_{ij})/v_j$.

Definition 1: Given an undirected graph $G = (V, E)$ with vertex set $V = \{v_1, v_2, \dots, v_n\}$ and edge set $E \subseteq V \times V$. A k -coloring of G is a mapping (valid coloring) function $f : V \rightarrow \{1, \dots, k\}$ where k is a positive integer, with each vertex v_i corresponding to an integer color value c_i ($1 \leq c_i \leq k$) such that $c_i \neq c_j$ for every edge $(v_i, v_j) \in E$. In other words, the numbers $1, 2, \dots, k$ represent the k colors, and adjacent vertices must have different colors. The k -coloring problem is to determine the minimum k .

The k -coloring problem has proven to be NP-complete (NPC), from which we prove our path scheduling (k -batching) problem is also NPC. Given a conflict graph, the k -batching problem is to determine the minimum number of batches (moving groups) required to accomplish a collision-free sensors deployment, formally defined as follows.

Definition 2: Given a conflict graph $G' = (S, L)$ with sensor set $S = \{s_1, s_2, \dots, s_n\}$ and conflict link set $L \subseteq S \times S$. A k -batching of G' is a mapping (valid grouping) function $f' : S \rightarrow \{1, \dots, k\}$ where k is a positive integer, with each sensor node s_i corresponding to an integer group number g_i ($1 \leq g_i \leq k$) such that $g_i \neq g_j$ for every conflict link $(s_i, s_j) \in L$. In other words, the numbers $1, 2, \dots, k$ denote the k groups (batches), and adjacent sensor nodes must be assigned in different moving groups (batches). The k -batching problem is to determine the minimum k .

Theorem 2: Our k -batching problem is NP-complete.

Proof: The k -batching problem belongs to NP since we can verify a solution easily in polynomial time. To prove k -batching is NP-complete, we reduce k -coloring to k -batching by showing k -coloring $\leq_p k$ -batching. In other words, any instance of k -coloring can be reduced in polynomial time to an instance of k -batching. Let $G = (V, E)$ represent an arbitrary instance of k -coloring. We can transform G with coloring function f to an instance of the k -batching $G' = (S, L)$ with grouping function f' by taking $S = V$, $L = E$ and group number g_i corresponding to the color value c_i in polynomial time. We claim that we can find a valid k -coloring of G if and only if we can discover a collision-free k -batching of G' . For the **if** part, suppose that G can be validly colored using a minimum number of k colors. By taking $G = G'$ and $f = f'$, we can find a solution in G' with each sensor node s_i being assigned group number $g_i = c_i$ resulting in a minimum of k batches (moving groups). Conversely, we prove the **only if** part. Suppose that we can obtain a minimum k batches for grouping all sensors without collisions in G' . By taking $G' = G$ and $f' = f$, there must exist a valid coloring of G with each vertex v_i being colored $c_i = g_i$ using a minimum number of k colors, which completes the proof. \square

B. Collision-Free Motion Algorithm (CFMA)

Since our k -batching problem is intractable, the answer to determining the minimum required number of moving sets (batches) cannot be easily obtained. Specifically, there does *not* exist an efficient (polynomial-time) path scheduling algorithm that guarantees *the fewest number of moving batches* to minimize the overall deployment execution time. Therefore, we devise CFMA (collision-free motion algorithm), a simple

TABLE II
SUMMARY OF NOTATIONS USED IN CFMA

Notation	Description
$conflict_type[i][j]$	Collision type between s_i and s_j
U_i	Used (not allowed) color set for s_i
$degree_i$	Node degree of s_i in the conflict graph
max_degree	Highest (max) node degree in the conflict graph
$same_flag[i][j]$	Indicates s_i and s_j must be colored the same (in same moving round) or s_j must move earlier (in different rounds)
$early_flag[i][j]$	Indicates s_i must move earlier than s_j (in different moving rounds) or s_j must reach intersection $p_{i,j}$ later than s_i (in same round)
$chain_i$	Set of ordered sensors in the i^{th} chain
$chain_count$	Number of discovered sensors chains
$loop_i$	Indicates loop (circular deadlock) ID with which sensor s_i is involved
$loop_count$	Number of discovered loops (circular deadlocks)
C_i	Set of potential colliding sensors against s_i
S_{order}	Set of all sensors sorted in decreasing degrees (coloring performed in order from high to low node degrees)
$color_i$	Color (moving group) assigned to s_i
$color_count$	Number of colors already assigned (being used)
$mflag_i$	Indicates s_i has moved from p_i to p_i'

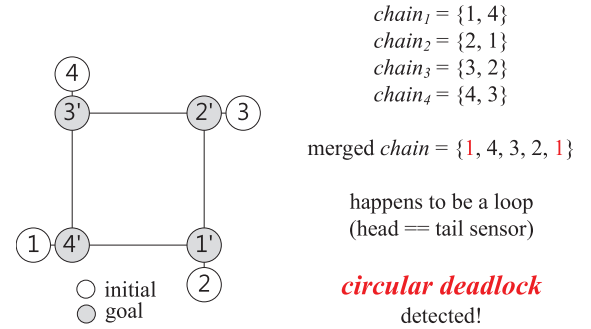


Fig. 6. Detection of a circular deadlock.

yet effective batching (coloring) strategy, to approximate the optimal solution.

1) *Algorithm Description:* In this proposed heuristic algorithm, we attempt to address the two main drawbacks in CFPP (identified in Section IV-A) by first constructing a conflict graph to reflect mutual conflict (collision) relationship between sensors and then assigning batches (colors) from highest- to lowest-degree nodes in a greedy manner. Here we define the **node degree** as the number of conflict links a sensor is connected with in a conflict graph. The strategy of considering sensors with high conflict degrees *before* sensors with low conflict degrees intends to *take care of sensors with the largest number of conflicts as early as possible*.⁷

CFMA starts out with an established conflict graph. For each pair of sensors (s_i, s_j) connected with a conflict link, a specific $conflict_type[i][j]$ is set values 1, 2, or 3 to respectively indicate the three collision cases **Case I**, **Case II**, and **Case III** described in Section IV-A1 (illustrated in Fig. 5). Before performing the coloring (grouping) process, we need to identify potential circular deadlocks. First, CFMA examines all sensor pairs and constructs *chains*, set of ordered sensors based on the $early_flag[i][j]$ defined in Section IV-A1. Specifically, $chain_x = \{s_i, s_j\}$ indicates that s_i should move

⁷In other words, we attempt to perform the vertex-coloring (grouping) process in a greedy manner, starting from the highest-degree sensors.

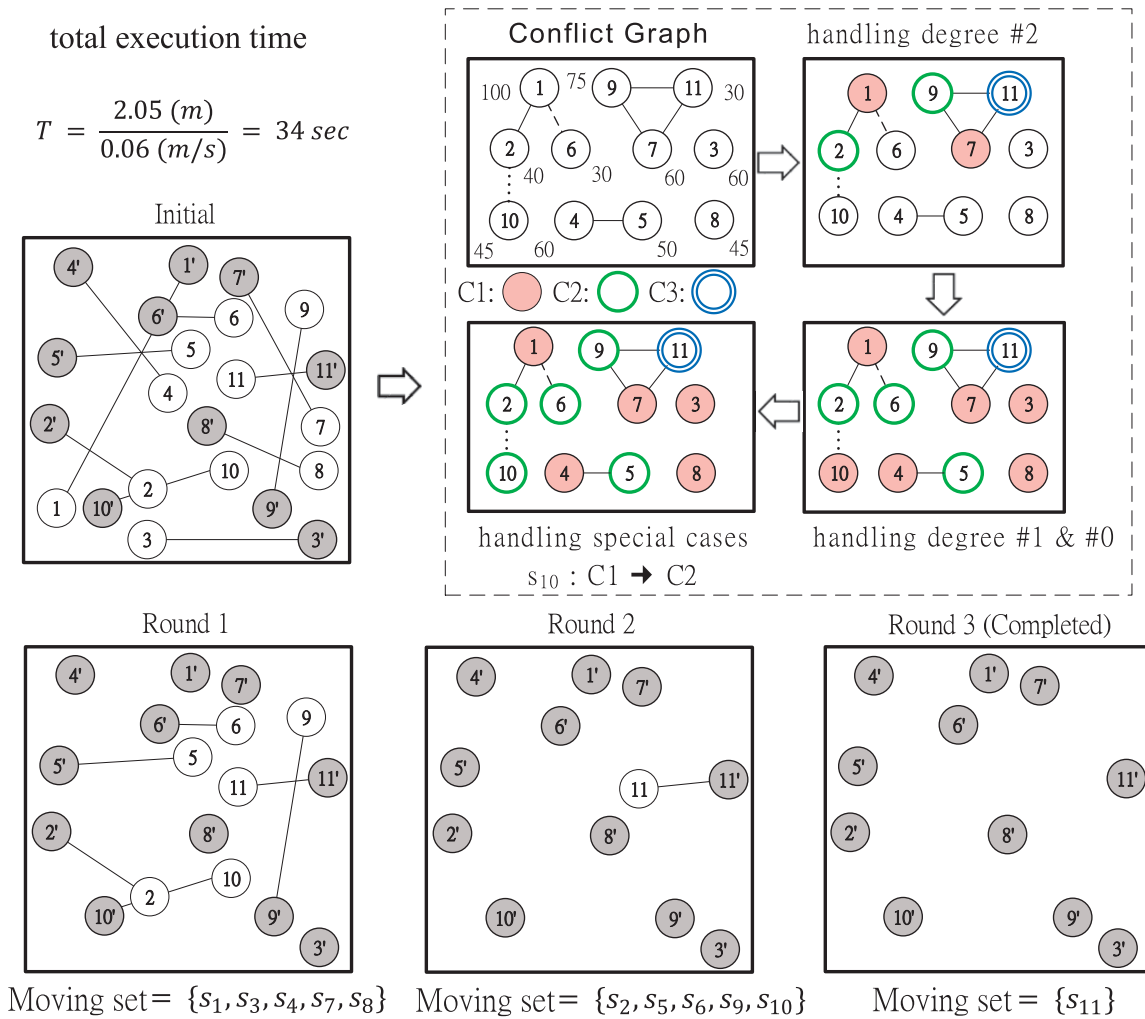


Fig. 7. Example illustrating the operations of CFMA grouping strategy for sensors batched movements.

earlier to avoid being blocked by s_j . Take Fig. 6 as an example, four chains are established: $chain_1 = \{1, 4\}$, $chain_2 = \{2, 1\}$, $chain_3 = \{3, 2\}$, and $chain_4 = \{4, 3\}$. Second, all chains are examined to produce merged chains by repeatedly connecting any two chains $chain_x$ and $chain_y$ if tail sensor in $chain_x$ equals head sensor in $chain_y$. In Fig. 6, the four chains are merged into one large $chain = \{1, 4, 3, 2, 1\}$. Finally, our algorithm checks the head and tail sensors in every merged chain. A circular deadlock (loop) is detected if head sensor equals tail sensor within the chain. In our example, the merged $chain = \{1, 4, 3, 2, 1\}$ happens to be a loop (circular deadlock) with the same head and tail sensor s_1 . Once a circular deadlock is detected, CFMA will apply the same color to all involved sensors in the loop. In other words, those sensors involved in the circular deadlock will be *scheduled* in the *same moving batch* and act together (colored as a group) thereafter. Consequently, in this example, all four sensors s_1, s_2, s_3 , and s_4 will *move simultaneously*, hence resolving the circular deadlock problem.

Our goal is to assign as few collision-free moving groups as possible to all sensors (associated with valid coloring for the conflict graph using as few colors as possible). The coloring (grouping) process is applied in the order of decreasing

node degrees, meaning that s_i with max_degree will be colored first. In case of sensors with the same degree, s_i will be considered before s_j if $i < j$. A used color set U_i keeps track of colors *not allowed* for s_i because these colors have already been allocated to neighboring nodes. Once producing a successfully colored conflict graph,⁸ CFMA returns to handle special collision cases **Case II** and **Case III** where some colors adjustment may be necessary. Finally, sensors perform physical movements accordingly in the order of their assigned colors (moving groups).

Table II summarizes the notations used in CFMA, and Algorithm 5 provides the CFMA pseudocode, presented in three segments. The first Algorithm 5 segment describes the procedure of classifying conflict types for every sensor pair (s_i, s_j) and constructing corresponding conflict graph. The second segment details on how to detect and resolve potential deadlocks, followed by a complete coloring process. Finally, the last segment illustrates the process of handling special

⁸Other existing vertex-coloring schemes are also applicable in scheduling collision-free groups (batches). However, since the k -batching problem is NP-complete, like our CFMA coloring algorithm, the number of scheduled moving batches cannot be guaranteed to be minimum.

Algorithm 2 Collision-Free Motion Algorithm (CFMA)

```

include all sensors in set  $S$ ; //  $i = 1, \dots, n$ 
initiate  $conflict\_type[i][j] = 0 \forall s_i, s_j \in S$ ;
for each sensor pair  $(s_i, s_j)$  where  $i \neq j$  do
  classify the conflict (collision)  $type$ 
  switch ( $type$ )
    Case I: set  $conflict\_type[i][j] = 1$ ;
    //  $s_i$  and  $s_j$  reach the intersection simultaneously
    Case II: set  $conflict\_type[i][j] = 2$ ;
    //  $s_j$ 's start position blocks  $s_i$ 's moving path
    Case III: set  $conflict\_type[i][j] = 3$ ;
    //  $s_j$ 's goal position blocks  $s_i$ 's moving path
  end for
clear  $degree_i$ ,  $max\_degree$ ,  $same\_flag[i][j]$ ,
 $early\_flag[i][j]$ ;
for  $\forall s_i, s_j \in S$  do
  initiate conflict set  $C_i = \emptyset$ ;
  for  $(i=1, i++, i \leq n)$  do
    for  $(j=1, j++, j \leq n)$  do
      if  $conflict\_type[i][j] \parallel conflict\_type[j][i] \neq 0$ 
then
         $degree_i++$ ;
        include  $s_j$  into conflict set  $C_i$ ;
      if  $conflict\_type[i][j] == 2$  then
         $same\_flag[i][j] = 1$ ;
        //  $s_i, s_j$  must be in same moving round or  $s_j$  moves earlier
      else if  $conflict\_type[i][j] == 3$  then
         $early\_flag[i][j] = 1$ ;
        //  $s_i$  must move earlier or  $s_j$  reaches intersection  $p_{ij}$  later
      if  $degree_i > max\_degree$  then
         $max\_degree = degree_i$ ;
      end for
    end for
  end for
// conflict graph successfully established with node degree information
sort the coloring order for  $\forall s_i \in S$  in decreasing degrees;

```

(to be continued in next Algorithm segment)

conflict **Cases II** and **III**. Now, the number of required moving sets is indicated by the value of $color_count$.

2) *An Example*: Using the same configuration with our CFPP example, Fig. 7 displays the constructed conflict graph and corresponding coloring process inside the dashed box. Starting from the $max_degree = 2$ with smallest sensor ID, CFMA colors s_1 red (c_1), followed by s_2 colored green (c_2). Next, sensors s_7, s_9, s_{11} are colored red (c_1), green (c_2), and blue (c_3) respectively according to the coloring rules. Then CFMA proceeds to color remaining sensors with lower degrees until all sensors have been validly colored. Finally, CFMA inspects the special collision cases (**Cases II and III**) and discovers s_{10} should move simultaneously with s_2 ($same_flag[10][2] = 1$), thus changing color from c_1 to c_2 for s_{10} . So far a total of three colors have been applied, which happens to be the minimum required number of colors because there is a triangle in the graph (formed by sensors s_7, s_9, s_{11}). In any undirected graph containing a triangle, at least three colors are needed for a valid coloring. Consequently, CFMA effectively improves the total execution time by reducing from 5 rounds in CFPP to 3 rounds, which

Algorithm 2 CFMA (Second Segment Continued)

```

initiate  $loop\_count = 0$ ,  $chain\_count = 0$ ; clear  $chain = \emptyset$ ;
clear  $loop_i = 0$  for  $\forall s_i \in S$ ;
for  $(i=1, i++, i \leq n)$  do
  for  $(j=1, j++, j \leq n)$  do
    if  $early\_flag[i][j] == 1$  then
       $chain\_count++$ ;
      include sensors  $s_i$  and  $s_j$  (orderly) into  $chain_{chain\_count}$ ;
    end for
  merge  $chain_x$  and  $chain_y$  if tail sensor in  $chain_x ==$  head sensor
  in  $chain_y$ ;
  update  $chain\_count$  after merging operations;
  for  $(j=1, j++, j \leq chain\_count)$  do
    if head sensor  $==$  tail sensor in  $chain_j$  then
       $loop\_count++$ ; // circular deadlock detected
       $loop_k = loop\_count \forall s_k \in chain_j$ ;
    end for
  end for
//  $s_i$  with  $max\_degree$  colored first
establish ordered set  $S_{order}$ ;
// for  $s_i, s_j$  with same degree,  $s_i$  will be considered before  $s_j$  if  $i < j$ 
initiate used color set  $U_i = \emptyset$  for  $\forall s_i \in S_{order}$ ;
clear  $color_i = 0$ ,  $color\_count = 0$ ;
for  $(i=1, i++, i \leq n)$  do
  if  $color_i \neq 0$  then continue; // skip coloring
  if  $U_i == \emptyset$  then  $color_i = 1$ ;
  else choose a color  $c \notin U_i$ ; set  $color_i = c$ ;
  for each  $s_j \in$  conflict set  $C_i$  do
    add  $color_i$  into used color set  $U_j$ ;
  end for
  for each  $s_j$  with  $loop_j = loop_i$  &&  $loop_i \neq 0$  do
    set  $color_j = color_i$ ; // all sensors involved in same loop
    (circular deadlock) applied same color
  for each  $s_k \in$  conflict set  $C_j$  do
    add  $color_j$  into used color set  $U_k$ ;
  end for
  end for
  if  $color_i > color\_count$  then  $color\_count = color_i$ ;
end for // coloring completed

```

(to be continued in next Algorithm segment)

Algorithm 2 CFMA (Final Segment)

```

for each sensor pair  $(s_i, s_j)$  where  $i \neq j$  do
  if  $same\_flag[i][j] == 1$  then
    if feasible then set  $color_i = color_j$  or  $color_j = color_i$ ;
    else slightly adjust location of  $s_j$  from  $p_j$  (original) to  $p_j'$  (adjusted);
  if  $early\_flag[i][j] == 1$  &&  $color_i \geq color_j$  then
    if  $color_i == color_j$  &&  $t_{p_j \rightarrow p_{ij}} > t_{p_i \rightarrow p_{ij}}$  then no action;
    else if feasible then switch  $color_i$  and  $color_j$ ;
    else set  $color_j = color\_count++$ ; check  $C_j$ ;
  end for // adjust colors for special conflict Cases II and III
clear moving flag  $mflag_i$  for  $\forall s_i \in S$ ;
while ( $S$  !empty) do
  for  $(c = 1, c++, c \leq color\_count)$  do
    perform simultaneous physical movements for  $\forall s_i$  with  $color_i = c$ ;
    set  $mflag_i = true$  for such sensor  $s_i$ ;
    // physical movements performed
    remove all  $s_i$  with  $mflag_i == true$  from sensors set  $S$ ;
  end for
end while

```

(end of Algorithm 5)

turns out to be the optimal (smallest) number of required moving sets (batches) for a collision-free deployment in this

configuration.⁹ Suppose $w(B_r)$ denotes the *longest travel time* among all sensors in moving batch B_r , here $r = 1, 2, 3$, then the corresponding total execution time T can be obtained by summing up $w(B_r)$. That is, $T = \sum_{r=1}^3 w(B_r)$ in this CFMA example.

V. WEIGHTED PATH SCHEDULING

Now that CFMA successfully reduces moving sets (batches) from 5 rounds in CFPP to 3 rounds (as shown in Fig. 7), the total execution time has been effectively decreased. If all robots require the same or similar travel times, then minimizing the number of moving batches naturally translates to minimized overall execution time. However, in a real deployment, robots (mobile sensors) typically take different travel times to their respective destinations (goal positions). For a moving set (group) containing robots with mixed long and short travel times, the execution time to complete physical movements within this group is determined by the *longest* travel time. From this perspective, a solution using fewer batches does not always perform better than another solution requiring more batches in terms of total execution time. Specifically, *minimizing the number of moving sets does not necessarily translate to minimized total execution time*, due to the fact that the number of batches cannot solely determine the aggregate required deployment time.

In light of this, we associate each sensor s_i with a *weight* defined by its *travel time* (moving from initial to goal positions) and investigate the weighted path scheduling problem. One can easily obtain the required travel time $w(s_i)$ for sensor s_i by computing $w(s_i) = \frac{\text{distance}(s_i)}{\text{velocity}(s_i)}$. For each batch (moving set) B_r , the weight of B_r is defined as $w(B_r) = \max\{w(s_i) | s_i \in B_r\}$ since the elapsed time to complete physical movements depends on the longest travel time among all sensors within this batch. Our idea is to design an enhanced grouping strategy that aptly arranges sensors with similar travel times in one group, combining with the goal of employing as few moving sets as possible, to further improve (accelerate) the total deployment time.

A. Motivation for Further Enhancement

We contemplate the example in Fig. 8 where all sensors are assumed to travel with the same moving speed, so the weight parameter (time cost) is proportional to the travel distance. Here all sensors are associated with weights $w(s_1) = 100$, $w(s_2) = w(s_3) = 40$, $w(s_4) = 20$, $w(s_5) = 40$, and $w(s_6) = 100$ (distance) units. Fig. 8 (a) shows the grouping result from applying CFMA employing three colors (moving sets). The first moving set B_1 contains s_1, s_4 with weight $w(B_1) = \max\{w(s_1), w(s_4)\} = \max\{100, 20\} = 100$, whereas the second and third moving sets imposing weights $w(B_2) = \max\{w(s_2), w(s_3), w(s_5)\} = \max\{40, 40, 40\} = 40$ and $w(B_3) = \max\{w(s_6)\} = \max\{100\} = 100$. As a result, CFMA produces a total moving cost of 240 units.

⁹From the proof in Theorem 4.3, in general, the number of moving batches calculated by CFMA cannot be guaranteed to be optimal (minimum).

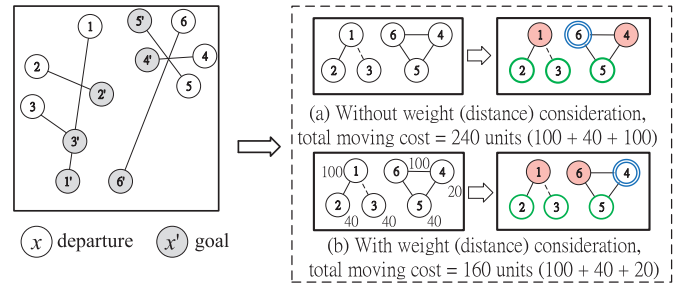


Fig. 8. Example showing the benefit of considering weight (traveling distance) in our grouping strategy (assuming the same moving speed); total deployment cost is reduced from 240 units (40 sec) to 160 units (26 sec).

To further enhance the performance with reduced moving cost, we consider *weight* (traveling distance) in the grouping strategy by starting group arrangement from sensors associated with the *highest weight*. Thus s_1, s_6 are assigned to the first moving set B_1 producing weight $w(B_1) = \max\{w(s_1), w(s_6)\} = \max\{100, 100\} = 100$, whereas the second and third moving sets imposing weights $w(B_2) = \max\{w(s_2), w(s_3), w(s_5)\} = \max\{40, 40, 40\} = 40$ and $w(B_3) = \max\{w(s_4)\} = \max\{20\} = 20$, as illustrated in Fig. 8 (b). Consequently, this new arrangement results in a total moving (travel) cost of 160 units, which shows a noticeable improvement in cost reduction by 33%. The enhancement brought by incorporating *weights* (time costs) into the grouping strategy motivates us to study the problem complexity of weighted path scheduling.

B. Problem Formulation

In Section IV-A2 we regard our path scheduling to minimize moving sets as a reduced case from the k -coloring problem in graph theory. As investigating the problem further, we discover, as mentioned in the beginning of Section V, minimizing the number of moving batches does not always yield minimized total travel cost. When we consider *weights* (time costs) in the grouping strategy, the overall execution time can be effectively reduced, as demonstrated in Fig. 8(b). For the goal of speeding up deployment time, *weights* should be taken into consideration when we perform the batching (coloring) process, referred to as the *weighted-batching* problem. This reminds us of another coloring problem, known as *weighted-coloring* in graph theory. The *weighted-coloring* is a weighted version of the coloring problem which consists in finding a color partition $P = (P_1, \dots, P_k)$ (k is the number of used colors) in the vertex set V of an undirected graph G into stable sets and minimizing $\sum_{r=1}^k w(P_r)$ where the partition weight $w(P_r)$ is defined as $\max\{w(v_i) | v_i \in P_r\}$. Note that the *weighted-coloring* problem attempts to *minimize the sum of total weights* imposed by all partitions, rather than optimize (minimize) the number of partitions k . Similarly, our *weighted-batching* problem intends to find a batch arrangement producing a *minimum sum of weights* (time costs), instead of minimizing the number of batches.

1) *NP Hardness Proof*: We formally define the *weighted-coloring* problem as follows.

Definition 3: Given an undirected graph $G = (V, E, w)$ with vertex set $V = \{v_1, v_2, \dots, v_n\}$ and edge set $E \subseteq V \times V$, the vertex-weighted function w assigns a positive real number $w(v_i) \geq 0$ for any vertex $v_i \in V$. The weighted-coloring problem is to find a valid coloring that contains a color partition $P = (P_1, \dots, P_k)$ of the vertex set V minimizing $\sum_{r=1}^k w(P_r)$ where the color partition weight $w(P_r) = \max\{w(v_i) | v_i \in P_r\}$.

The weighted-coloring problem was originally proposed in [35] and proven to be NP-hard [36], from which we prove our weighted path scheduling (*weighted-batching*) problem is also NP-hard. Given a conflict graph, the *weighted-batching* problem is to determine a batch arrangement with a minimized sum of total weights (travel time costs) to optimize the collision-free deployment time, formally defined below.

Definition 4: Given a conflict graph $G' = (S, L, w')$ with sensor set $S = \{s_1, s_2, \dots, s_n\}$ and conflict link set $L \subseteq S \times S$, the sensor-weighted function w' assigns a positive real number $w'(s_i) \geq 0$ for any sensor $s_i \in S$. The *weighted-batching* problem is to find a collision-free batching that contains a batch partition $B = (B_1, \dots, B_k)$ of the sensor set S minimizing $\sum_{r=1}^k w'(B_r)$ where the batch partition weight $w'(B_r) = \max\{w'(s_i) | s_i \in B_r\}$.

Theorem 3: Our *weighted-batching* problem is NP-hard.

Proof: To prove *weighted-batching* is NP-hard, we reduce *weighted-coloring* to *weighted-batching* by showing *weighted-coloring* \leq_p *weighted-batching*. In other words, any instance of *weighted-coloring* can be reduced in polynomial time to an instance of *weighted-batching*. Let $G = (V, E, w)$ with coloring function f and color partition P represent an arbitrary instance of *weighted-coloring*. We can transform G with coloring function f and color partition P to an instance of the *weighted-batching* $G' = (S, L, w')$ with grouping function f' and batch partition B by taking $S = V$, $L = E$, $w' = w$, group number g_i corresponding to the color value c_i and batch partition B_r corresponding to the color partition P_r in polynomial time. We claim that we can find a valid *weighted-coloring* of G if and only if we can discover a collision-free *weighted-batching* of G' . For the **if** part, suppose that G can be validly colored with minimized $\sum_{r=1}^k w(P_r)$ where $w(P_r) = \max\{w(v_i) | v_i \in P_r\}$. By taking $G = G'$, $f = f'$ and $P_r = B_r$, we can find a solution in G' with each sensor node s_i being assigned group number $g_i = c_i$ resulting in a minimized $\sum_{r=1}^k w'(B_r)$ (sum of batch partition weights) where $w'(B_r) = \max\{w'(s_i) | s_i \in B_r\}$. Conversely, we prove the **only if** part. Suppose that we can obtain a collision-free (valid) sensors grouping arrangement with minimized total travel time in G' . By taking $G' = G$, $f' = f$ and $B_r = P_r$, there must exist a valid coloring of G with each vertex v_i being colored $c_i = g_i$ producing a minimized $\sum_{r=1}^k w(P_r)$ (sum of color partition weights), which completes the proof. \square

C. Weighted CFMA (wCFMA)

Since the *weighted-batching* problem belongs to NP-hard, we propose wCFMA, a weight-ordered heuristic algorithm, to approximate the min-weighted solution. Our idea is to impose smaller variance (difference) among travel time costs

TABLE III
SUMMARY OF NOTATIONS USED IN wCFMA

Notation	Description
$conflict_type[i][j]$	Collision type between s_i and s_j
U_i	Used (not allowed) color set for s_i
$weight_i$	Weight (estimated moving time) associated with sensor s_i
max_weight	Highest (max) weight among all sensors
$same_flag[i][j]$	Indicates s_i and s_j must be colored the same (in same moving round) or s_j must move earlier (in different rounds)
$early_flag[i][j]$	Indicates s_i must move earlier than s_j (in different moving rounds) or s_j must reach intersection p_{ij} later than s_i (in same round)
C_i	Set of potential colliding sensors against s_i
S_{order}	Set of all sensors sorted in decreasing weights (coloring performed in order from high to low node weights)
$color_i$	Color (moving group) assigned to s_i
$color_count$	Number of colors already assigned (being used)
$mflag_i$	Indicates s_i has moved from p_i to p_i

Algorithm 3 Weighted CFMA (wCFMA)

```

initiate and classify conflict type (same as CFMA)
calculate  $weight_i$  for all sensors
clear  $max\_weight$ ,  $same\_flag[i][j]$ ,  $early\_flag[i][j]$ ;
for  $\forall s_i, s_j \in S$  do
  initiate conflict set  $C_i = \emptyset$ ;
  for ( $i=1, i++, i \leq n$ ) do
    for ( $j=1, j++, j \leq n$ ) do
      if  $conflict\_type[i][j] \parallel conflict\_type[j][i] \neq 0$  then
        include  $s_j$  into conflict set  $C_i$ ;
      if  $conflict\_type[i][j] == 2$  then
         $same\_flag[i][j] = 1$ ;
        //  $s_i, s_j$  must be in same moving round or  $s_j$  moves earlier
      else if  $conflict\_type[i][j] == 3$  then
         $early\_flag[i][j] = 1$ ;
        //  $s_i$  must move earlier or  $s_j$  reaches intersection  $p_{ij}$  later
      if  $weight_i > max\_weight$  then
         $max\_weight = weight_i$ ;
    end for
  end for
end for
// conflict graph successfully established with node weight information
sort the coloring order for  $\forall s_i \in S$  in decreasing weights;
//  $s_i$  with  $max\_weight$  colored first
establish ordered set  $S_{order}$ ;
// for  $s_i, s_j$  with same weight,  $s_i$  will be considered before  $s_j$  if  $i < j$ 
initiate used color set  $U_i = \emptyset$  for  $\forall s_i \in S_{order}$ ;
clear  $color_i = 0$ ,  $color\_count = 0$ ;
perform coloring process (same as CFMA)
adjust colors for special conflict Cases II and III
perform physical movements (same as CFMA)

```

within the same moving batch. In contrast to CFMA intending to reduce the number of batches, wCFMA attempts to reduce the sum of batch weights, where *weight* of a moving batch is the maximum (longest) travel time among sensors in the same batch.

1) *Algorithm Description:* wCFMA mostly follows the previously outlined CFMA method. The key difference lies in the coloring process, in which wCFMA colors sensors in the order of *decreasing weights*. Table III summarizes the notations used in wCFMA, and Algorithm 6 provides the wCFMA pseudocode.

2) *An Example:* Our wCFMA intends to generate batches with *shortened* total execution (moving) time, thus

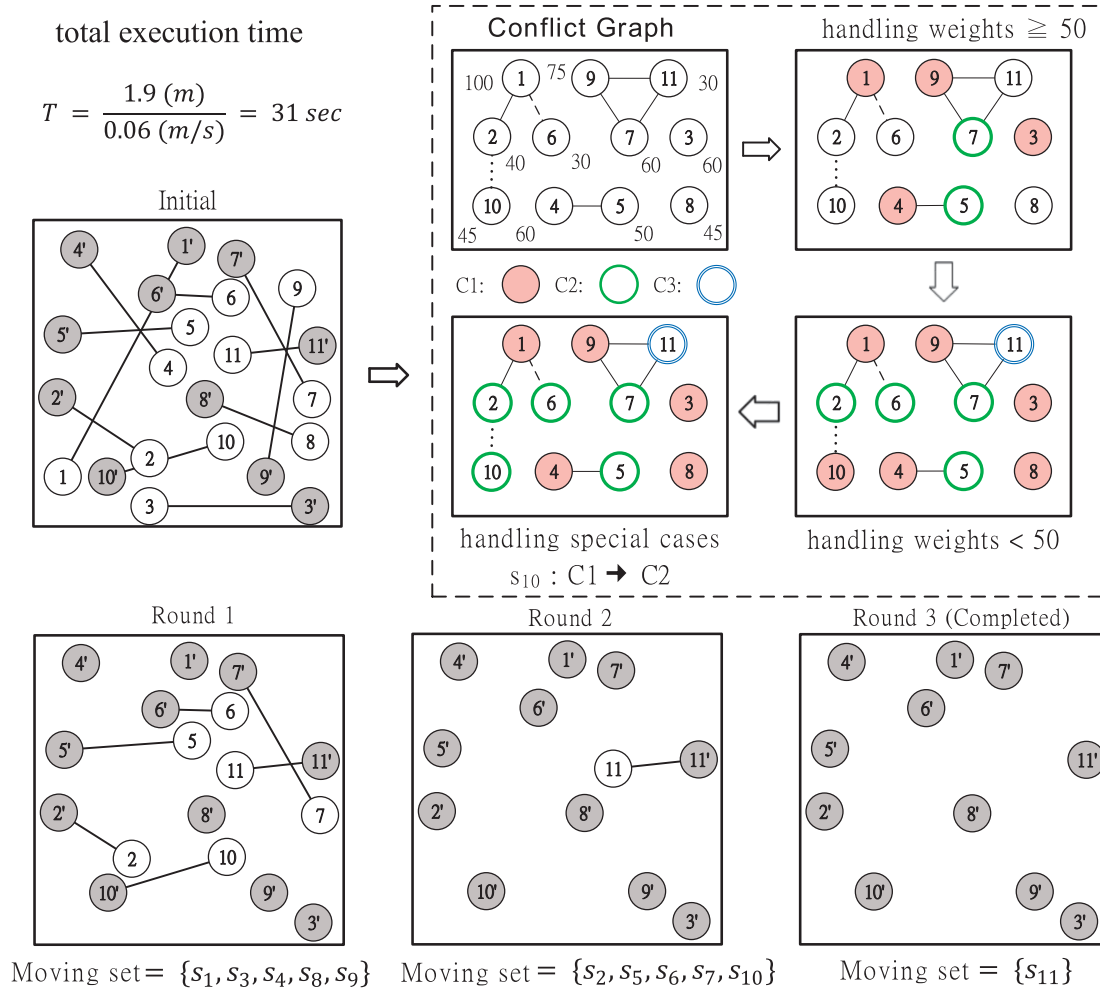


Fig. 9. Example illustrating the operations of our wCFMA grouping strategy for sensors batched movements.

accomplishing the sensors deployment task earlier (compared to CFPP and CFMA). Fig. 9 illustrates the wCFMA coloring process which results in 3 rounds (same as CFMA) with total moving cost of 190 units. Compared to 290 units in CFPP and 205 units in CFMA, the overall execution cost in wCFMA improves (reduces) by 35% (improved from CFPP) and 7.3% (further improved from CFMA) respectively. In our prototype, one grid unit equals 1 cm, and the average moving speed of our robots is measured at 0.06 m/sec. Hence the total execution time can be obtained at 48 sec (CFPP), 34 sec (CFMA), and 31 sec (wCFMA), respectively.

VI. PERFORMANCE EVALUATION

In this section, we validate our motion algorithms by comparing the performance with three other path-planning approaches: ADO (introduced in [8]), Super A* (introduced in [9]) and M* algorithm (introduced in [10]).

A. Sensors Goal Reachability

Fig. 10 depicts a configuration consisting of 20 random moving paths with three deadlock situations. Potential deadlocks #1 and #2 simulate the situation when a goal position blocks another sensor's moving path, whereas potential

deadlock #3 demonstrates a triangular deadlock situation, in which all involved sensors are prevented from moving. This configuration is arranged in order to observe the capability of path-planning algorithms on resolving deadlocks, as mentioned in [37].¹⁰

Fig. 11 shows the sensors goal reachability accomplished by CFPP, CFMA, wCFMA, ADO, Super A*, and M* as time advances. Samples are taken every 20 seconds to record the reachability rate within a set time. We observe that ADO suffers from deadlocks #1 and #3 at time 20, which stop sensors $s_{11}, s_{16}, s_{17}, s_{18}$ from moving to their destinations (goals). At time 40, ADO encounters deadlock #2, which prevents sensor s_{14} from reaching its goal position. After time 40, ADO is unable to make any further progress with eventually 75% final goal reachability. Super A* is capable of resolving deadlocks #1 and #2, but unable to handle the triangular deadlock #3, which occurs at time 20 and traps sensors s_{16}, s_{17}, s_{18} from departing toward their destinations. Super A* stops making progress after time 40, leading to 85%

¹⁰Here we define that deadlocks among two or more robots (mobile sensors) occur if these robots block each other in a way such that any or all of them is/are unable to continue along its/their trajectory (traveling path) without causing a collision.

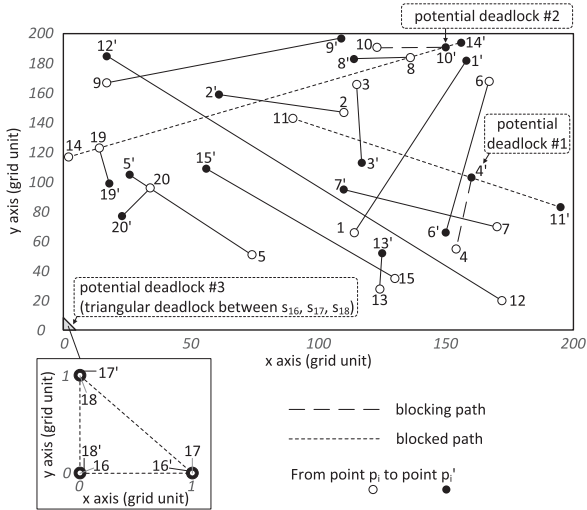


Fig. 10. Random path configuration of 20 mobile sensors in a monitored 200×200 area with potential deadlocks #1, #2, and #3.

final goal reachability. Similar to Super A*, M* also suffers from deadlock #3, leading to 85% final goal reachability.

In contrast, our CFPP is capable of resolving all deadlock situations. For deadlocks #1 and #2 (classified as Case D-V in our CFPP algorithm), sensors s_4 and s_{10} execute **Case D-V** by deferring their movements (scheduled in a later batch after s_{11} and s_{14} reach their goal positions). For deadlock #3 (a triangular deadlock), since sensors s_{16} , s_{17} , s_{18} will be scheduled in the same moving order, CFPP naturally resolves this deadlock situation by allowing three sensors to move simultaneously without blocking each other. Interestingly, CFPP reachability grows slowly (due to the batched movements applied by CFPP) and outperforms the other three approaches after time passes 80, eventually leading to 100% goal reachability.

CFMA also defers the movement of sensors s_4 and s_{10} after s_{11} and s_{14} reach their goals, resolving deadlocks #1 and #2, whereas the triangular deadlock #3 is handled by allowing three sensors to move in the same round (batch). CFMA achieves 100% goal reachability at time 60 (earlier than CFPP) due to the reduced number of moving batches employed. wCFMA further improves the overall deployment time by grouping sensors with longer travel distances (s_1 , s_{11} , s_{12} , s_{14}) in the same batch, while CFMA arranges s_1 and s_{14} (two sensors with longer paths) to move in different batches. As a result, wCFMA achieves 100% goal reachability before time 40 (fastest among all approaches).

We conduct extensive experiments against various sensor populations (creating various sensor densities) to observe how each mechanism performs in different sensor configurations. Fig. 12 illustrates a monitored 200×200 area with up to 35 sensors under three different configurations: (a) balanced initial/goal positions, (b) crowded initial positions, and (c) crowded goal positions.

B. Computation Latency and Execution Time

Fig. 12 (d)(e)(f) show the results of our experiments with three different configurations. ADO suffers from the deadlock problems in which some robots are unable to reach

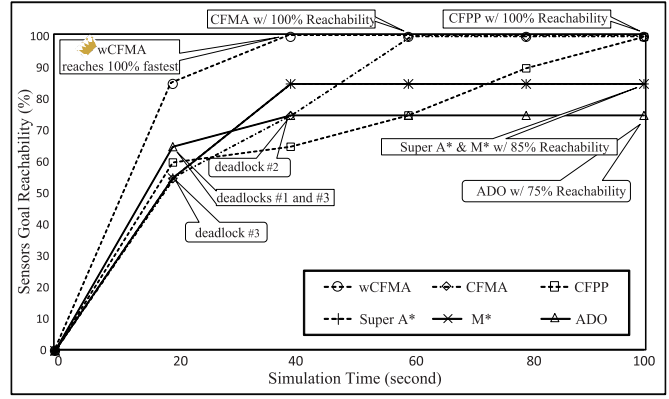


Fig. 11. Sensors goal reachability accomplished by our CFPP, CFMA, wCFMA, also Super A*, M*, and ADO path-planning strategies with the existence of potential deadlocks #1, #2, and #3 in a monitored 200×200 area.

their destinations (goal positions) with reachability weakening into 80% as the population of sensors grows. Super A* is capable of approaching 100% reachability under a balanced configuration, yet with crowded initials and crowded goals, the reachability of Super A* drops, indicating this mechanism is vulnerable to uneven distribution of sensors. Although M* performs relatively well with small numbers of sensors, its performance weakens significantly as the population grows because the algorithm fails to generate safe moving paths for most sensors. In fact, M* reachability drops drastically to almost 0% in the second configuration (crowded initials). When the initial points of sensors are very crowded (very close to each others), the tree expansion in M* encounters failures very early, and the mechanism eventually fails to generate moving paths for most/all of the sensors as the population grows. In contrast, our CFPP, CFMA, and wCFMA can safely generate collision-free paths and deliver all sensors to their destinations with 100% goal reachability for various sensor populations under all three configurations.

Fig. 12 (d)(e)(f) also display the sensors deployment time incurred by all six approaches. The total deployment time consists of computation latency (shaded parts) and execution (moving) time (unshaded/light parts). We define the moving time as the time required for all robots to travel to their goals (from robots start moving until no robot moves anymore, even though in some cases, not all robots can successfully reach their goals). CFPP, CFMA, wCFMA and ADO manifest little computation latency (so that it might be difficult to observe in the figure, as the value is very small, close to zero second), while M* and Super A* require significantly longer time to compute path-planning solutions (can be observed in the figure clearly). Apparently, computation latency with M* increases drastically as the number of mobile sensors (moving paths) grows, due to extensive computation required by the tree expansion method in M* algorithm. In the second (crowded initials) and third configurations (crowded goals), the generated paths encounter failures (collisions) and M* needs to retrace backward (initiating backpropagation sets), leading to growing dimensions of the search space/tree, consuming even more computational time. For execution (moving) time, we can observe that CFPP spends the most time for travel due to its

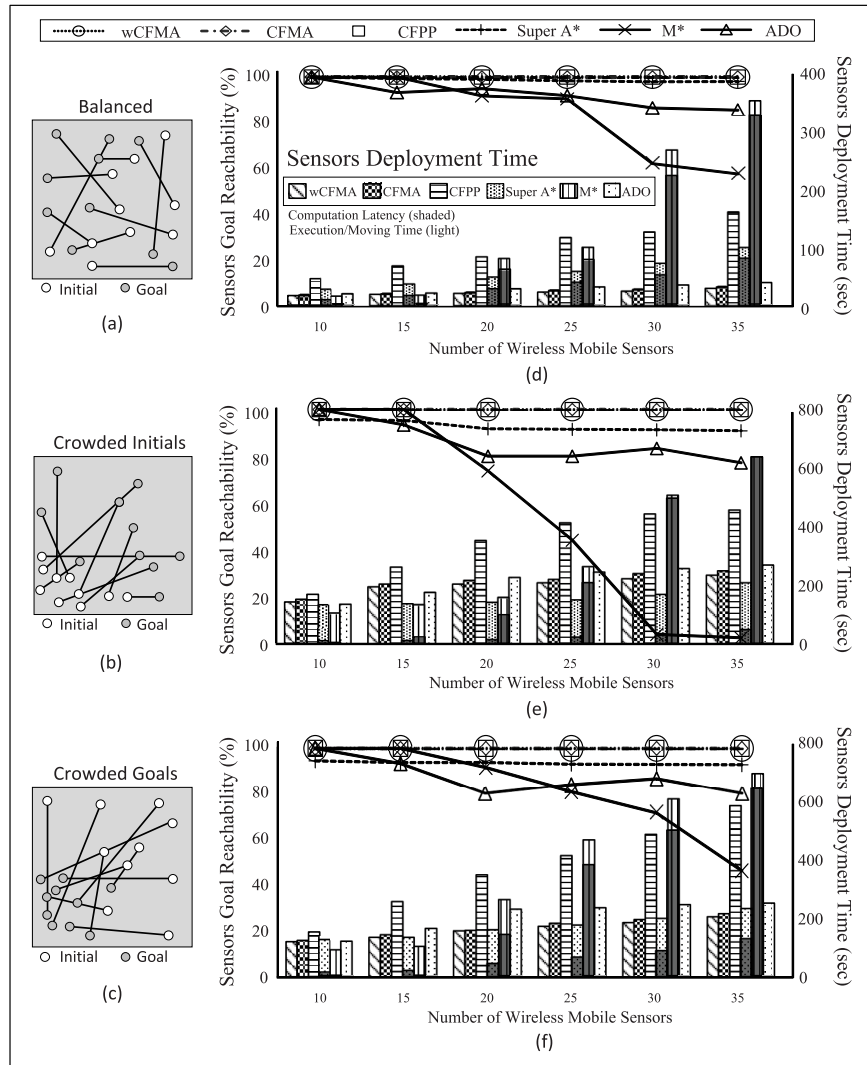


Fig. 12. Sensors goal reachability and sensors deployment time accomplished by different path-planning strategies in a monitored 200×200 area under three deployment scenarios with (a) balanced initial/goal positions, (b) crowded initial positions, and (c) crowded goal positions; performance results are shown in (d), (e), and (f) respectively.

batched movements in a conservative way, whereas CFMA and wCFMA improve (decrease) the travel time noticeably because of minimized number of batches and minimized sum of total time costs. The travel (execution) time for M* drops as the population grows, because most robots are unable to reach their intended destinations (algorithm fails to generate paths for those robots). Despite that CFPP consumes more execution time, the total deployment time required by M* is longer than CFPP when sensor population increases (over 30 sensors in the figure); in fact, M* fails to deliver most sensors whereas CFPP still achieves a 100% reachability. Overall, this set of experiments demonstrates our CFMA and wCFMA outperform other approaches in terms of sensors deployment latency while maintaining 100% goal reachability.

C. Energy Consumption

To model the moving energy, we estimate the energy consumed by the motion device moving for one grid unit by performing real measurements on the sensor robot used in our

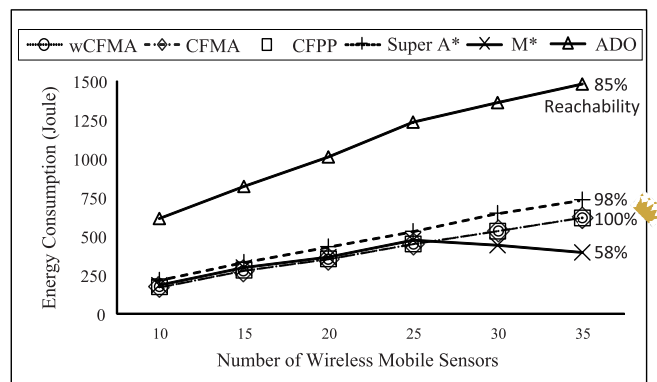


Fig. 13. Total energy consumption incurred by different path-planning strategies under various numbers of wireless mobile sensors in a monitored 200×200 area.

implementation testbed [7], based on moving robots (LEGO MINDSTORMS NXT 9797 [28]), with grid size equal to

1 cm. The robot assembles six 1.2 V 2000 mAh rechargeable NiMH batteries with measured 200 ~ 290 mA moving current and average moving speed at 0.06 m/sec. Consequently, the average moving energy consumption per grid (unit distance) can be obtained by $0.29 \times 7.2 \times \left(\frac{0.01}{0.06}\right) = 0.348$ Joule. We then compute the total moving energy consumption based on the traveling distance accordingly. To model the energy consumed by visual sensors (cameras) in ADO (recall that ADO algorithm depends on the presence of omnidirectional cameras for calculating moving paths, as described in Section II), we perform measurements on a commercial video camera M30 Series [29]. An estimated 3.4 Watt on average (ranging from 2.2 to 4.6 Watt) facilitates our calculation of the total camera energy required by an individual sensor in ADO, which can be obtained by $3.4 \times \text{moving time (sec)}$ Joule.

Fig. 13 displays the aggregate energy consumed by six approaches under a balanced sensors configuration. Our proposed mechanisms (CFPP, CFMA and wCFMA) consume the least aggregate energy while achieving 100% goal reachability, whereas ADO produces the most power cost due to extra energy consumed by visual sensors (omnidirectional cameras). Meanwhile, since the generated Super A* paths are not always in straight lines (shortest paths), Super A* consumes relatively more moving energy as compared to our approaches. Note that, as sensor population grows, M* deployment consumes less energy, not because of improved efficiency, but due to its inability to deliver most of the robots to their intended destinations (most robots fail to move with only 58% goal reachability). Consequently, this set of experiments further demonstrates our proposed motion algorithms are practical in terms of energy efficiency.

VII. CONCLUSION AND FUTURE APPLICATIONS

In this paper, we devised three collision-free motion algorithms (CFPP, CFMA, wCFMA) to schedule moving paths for the mobile sensors deployment problem. When sensors move around to self-deploy, the motion algorithm comes into play by systematically classifying colliding cases and employing batched movements. Our proposed mechanisms guarantee 100% sensors goal reachability due to the capability of judiciously grouping all sensors and guiding them to their goals without causing collisions nor deadlocks. With the assistance of an IoT application development tool [38], we have implemented a proof-of-concept prototype¹¹ based on moving robots (LEGO MINDSTORMS NXT 9797 [28]), network camera M30 [29], and Tibbo embedded systems [30] to further corroborate our CFMA protocol feasibility in a real-life environment. A brief demonstration video on our experiments is available at [11]. Among a great number of practical IoT testbeds that have been set up during recent years, potential application targets include roof/bridge monitoring networks, agricultural plant watering schedules to enable smart farming, underwater sensor systems [25], [39], etc. We observe

¹¹In our system, robots are equipped with collision-detection sensors to prevent occasional bumps in case of path prediction inaccuracy. When that happens, involved robots will stop and wait for central server's instructions before proceeding further.

that IoT testbeds operated in open space are in need of effective, intelligent yet not too complicated, collision-free motion scheduling algorithms. We expect the concept and deployment of our proposed approaches can facilitate the realization of various meaningful IoT applications in the near future.

REFERENCES

- [1] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proc. ACM WSNA*, Sep. 2002, pp. 88–97.
- [2] E. S. Biagioni and K. W. Bridges, "The application of remote sensor technology to assist the recovery of rare and endangered species," *Int. J. High Perform. Comput. Appl.*, vol. 16, no. 3, pp. 315–324, Aug. 2002.
- [3] T.-Y. Lin, H. A. Santoso, W.-T. Liu, and H.-T. Liu, "An enhanced sensor deployment scheme for automated smart environments," in *Proc. IEEE Int. Conf. Adv. Netw. Telecommun. Syst. (ANTS)*, Dec. 2013, pp. 1–6.
- [4] C. Panditharathne, T. Y. Lin, and K. H. Chen, "Heterogeneous directional sensors self-deployment problem in a bounded monitoring area," in *Proc. IEEE Wireless Commun. Netw. Conf.*, Mar. 2011, pp. 623–628.
- [5] K.-H. Peng, T.-Y. Lin, and K.-H. Chen, "On seamless wireless sensor deployment and system implementation," *Int. J. Adv. Inf. Technol.*, vol. 3, no. 2, pp. 72–92, Dec. 2009.
- [6] T.-Y. Lin, H. A. Santoso, K.-R. Wu, and G.-L. Wang, "Enhanced deployment algorithms for heterogeneous directional mobile sensors in a bounded monitoring area," *IEEE Trans. Mobile Comput.*, vol. 16, no. 3, pp. 744–758, Mar. 2017.
- [7] T.-Y. Lin, H. A. Santoso, and K.-R. Wu, "Global sensor deployment and local coverage-aware recovery schemes for smart environments," *IEEE Trans. Mobile Comput.*, vol. 14, no. 7, pp. 1382–1396, Jul. 2015.
- [8] C. Cai, C. Yang, Q. Zhu, and Y. Liang, "Collision avoidance in multi-robot systems," in *Proc. Int. Conf. Mechatronics Autom.*, Aug. 2007, pp. 2795–2800.
- [9] F. Liu and A. Narayanan, "Real time replanning based on A* for collision avoidance in multi-robot systems," in *Proc. 8th Int. Conf. Ubiquitous Robot. Ambient Intell. (URAI)*, Nov. 2011, pp. 473–479.
- [10] G. Wagner and H. Choset, "Subdimensional expansion for multirobot path planning," *Artif. Intell.*, vol. 219, pp. 1–24, Feb. 2015.
- [11] T.-Y. Lin, K.-R. Wu, Y.-S. Chen, and Y.-S. Shen. (2021). *Demo: Collision-Free Motion Algorithms for Sensors Automated Deployment*. [Online]. Available: <http://bun.cm.nctu.edu.tw/demo/CFMA.mp4>
- [12] B. Mahajan and P. Marbate, "Literature review on path planning in dynamic environment," *Int. J. Comput. Sci. Netw.*, vol. 2, no. 1, pp. 115–118, Feb. 2013.
- [13] P. Bhattacharya and M. L. Gavrilova, "Roadmap-based path planning using the Voronoi diagram for clearance-based shortest path," *IEEE Robot. Autom. Mag.*, vol. 15, no. 2, pp. 58–66, Jun. 2008.
- [14] A. N. Nazif, A. Davoodi, and P. Pasquier, "Multi-agent area coverage using a single Query roadmap: A swarm intelligence approach," *Advances Practical Multi-Agent Syst.*, vol. 325, no. 1, pp. 95–112, Sep. 2011.
- [15] M. T. Rantanen, "A connectivity-based method for enhancing sampling in probabilistic roadmap planners," *J. Intell. Robot. Syst.*, vol. 64, no. 2, pp. 161–178, Nov. 2011.
- [16] M. T. Rantanen and M. Juhola, "A configuration deactivation algorithm for boosting probabilistic roadmap planning of robots," *Int. J. Autom. Comput.*, vol. 9, no. 2, pp. 155–164, Apr. 2012.
- [17] P. Song and V. Kumar, "A potential field based approach to multi-robot manipulation," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2002, pp. 1217–1222.
- [18] Y. Wang and G. S. Chirikjian, "A new potential field method for robot path planning," in *Proc. IEEE Int. Conf. Robot. Automat. Symp.*, Apr. 2000, pp. 977–982.
- [19] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Trans. Robot. Autom.*, vol. 8, no. 5, pp. 501–518, Aug. 1992.
- [20] V. Kunchev, L. Jain, V. Ivancevic, and A. Finn, "Path planning and obstacle avoidance for autonomous mobile robots: A review," in *Proc. Int. Conf. Knowl.-Based Intell. Inf. Eng. Syst.*, Oct. 2006, pp. 537–544.

- [21] J. Wang, W. Chi, C. Li, C. Wang, and M. Q.-H. Meng, "Neural RRT*: Learning-based optimal path planning," *IEEE Trans. Autom. Sci. Eng.*, vol. 17, no. 4, pp. 1748–1758, Oct. 2020.
- [22] K. Li, Q. Hu, and J. Liu, "Path planning of mobile robot based on improved multiobjective genetic algorithm," *Wireless Commun. Mobile Comput.*, vol. 2021, pp. 1–12, Apr. 2021.
- [23] K. Hao, J. Zhao, B. Wang, Y. Liu, and C. Wang, "The application of an adaptive genetic algorithm based on collision detection in path planning of mobile robots," *Comput. Intell. Neurosci.*, vol. 2021, pp. 1–20, May 2021.
- [24] I. Noreen, A. Khan, K. Asghar, and Z. Habib, "A path-planning performance comparison of RRT*-AB with MEA* in a 2-dimensional environment," *MDPI Symmetry*, vol. 11, no. 7, pp. 945–960, 2019.
- [25] R. B. Wynn *et al.*, "Autonomous underwater vehicles (AUVs): Their past, present and future contributions to the advancement of marine geoscience," *Marine Geol.*, vol. 352, pp. 451–468, Jun. 2014.
- [26] V. Pshikhopov, M. Medvedev, and B. V. Gurenko, "Homing and docking autopilot design for autonomous underwater vehicle," *Appl. Mech. Mater.*, vols. 490–491, pp. 700–707, Jan. 2014.
- [27] T.-Y. Lin, H. A. Santoso, C.-A. Lin, and G.-L. Wang, "CFPP: Collision-free path planning for wireless mobile sensors deployment," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2015, pp. 6412–6417.
- [28] (2016). *LEGO Mindstorms NXT 9797*. [Online]. Available: <http://www.lego.com>
- [29] (2020). *Axis Communication M30 series*. [Online]. Available: http://tw.axis.com/download/ds_m30series_51497_tw_1304.pdf
- [30] (2020). *Tibbo Technology*. [Online]. Available: <http://tibbo.com>
- [31] T.-Y. Lin, K.-R. Wu, Y.-S. Chen, W.-H. Huang, and Y.-T. Chen, "Takeout service automation with trained robots in the pandemic-transformed catering business," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 903–910, Apr. 2021.
- [32] T.-Y. Lin and *et al.*, "Keeping social distance during the pandemic: Contactless meal order and takeout service via assisted smart robots," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Jun. 2021.
- [33] R. M. Karp, "Reducibility among combinatorial problems," *Complex. Comput. Comput.*, vol. 4, pp. 85–103, Oct. 1972.
- [34] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W.H. Freeman, 1979.
- [35] D. Guan and X. Zhu, "A coloring problem for weighted graphs," *Inf. Process. Lett.*, vol. 61, pp. 77–81, Jan. 1997.
- [36] A. Mishra, S. Banerjee, and W. Arbaugh, "Weighted coloring based channel assignment for WLANs," *ACM SIGMOBILE Mobile Comput. Commun. Rev.*, vol. 9, no. 3, pp. 19–31, 2005.
- [37] M. Jager and B. Nebel, "Decentralized collision avoidance, deadlock detection, and deadlock resolution for multiple mobile robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Nov. 2001, pp. 1213–1219.
- [38] Y.-B. Lin, Y.-W. Lin, C.-M. Huang, C.-Y. Chih, and P. Lin, "IoTtalk: A management platform for reconfigurable sensor devices," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1552–1562, Oct. 2017.
- [39] I. Vasilescu, K. Kotay, D. Rus, M. Dunbabin, and P. Corke, "Data collection, storage, and retrieval with an underwater sensor network," in *Proc. Int. Conf. Embedded Netw. Sensor Syst.*, Nov. 2005, pp. 154–165.



Ting-Yu Lin (Member, IEEE) received the Ph.D. degree in computer science from the National Chiao Tung University, Taiwan. From June 2003 to February 2004, she was affiliated with the Massachusetts Institute of Technology (MIT) as a Research Scientist. She worked with the Industrial Technology Research Institute, Taiwan, from March 2004 to August 2005, as a Software Engineer. Afterwards, she joined the University of Illinois at Urbana-Champaign (UIUC), as a Post-Doctoral Research Associate, from September 2005 to August 2006, under both the government and university sponsorship. She is currently an Associate Professor with the Department of Electrical and Computer Engineering, Institute of Communications Engineering, National Yang Ming Chiao Tung University, Taiwan. Her research interests include wireless communications, mobile computing, WLANs/WPANs, wireless sensor/mesh networking, and performance optimization. She is a member of ACM. When she graduated, she was recipient of the Phi Tau Phi Scholastic Honor Award.



Kun-Ru Wu received the B.S. degree in electrical engineering from the National Chung Hsing University, Taiwan, and the Ph.D. degree from the Institute of Communications Engineering, National Chiao Tung University, Taiwan, in September 2015. He is currently a Post-Doctoral Research Associate with the Department of Computer Science, National Yang Ming Chiao Tung University. His research interests include system programming, sensor fusion algorithms, wireless mesh networks, and the IoT applications.



You-Shuo Chen received the B.S. degree in electrical engineering and the M.S. degree from the Institute of Communications Engineering, National Chiao Tung University, in June 2018 and September 2020, respectively. His research interests include wireless network protocols and mobile sensor networks.



Yan-Syun Shen received the B.S. degree in electrical and computer engineering from the National Chiao Tung University in June 2020. He is currently pursuing the M.S. degree with the Institute of Communications Engineering, National Yang Ming Chiao Tung University, Taiwan. His research interests include autonomous sensor networks, embedded systems, and robotics network automation technologies.