Optimal $k$-Coverage Charging Problem

Xuan Li, and Miao Jin

Abstract—Wireless rechargeable sensor networks, consisting of sensor nodes with rechargeable batteries and mobile chargers to replenish their batteries, have gradually become a promising solution to the bottleneck of energy limitation that hinders the wide deployment of wireless sensor networks (WSN). In this paper, we focus on the mobile charger scheduling and path optimization scenario in which the $k$-coverage ability of a network system needs to be maintained. We formulate the optimal $k$-coverage charging problem of finding a feasible path for a mobile charger to charge a set of sensor nodes within their estimated charging time windows under the constraint of maintaining the $k$-coverage ability of the network system, with an objective of minimizing the energy consumption on traveling per tour. We show the hardness of the problem that even finding a feasible path for the trivial case of the problem is an NP-complete one with no polytime constant-factor approximation algorithm.

We model the problem and apply the color coding technique to design an algorithm that finds an exact solution to the optimal $k$-coverage charging problem. However, the computational complexity is still prohibitive for large size networks. We then introduce Deep Q-learning, an reinforcement learning algorithm to tackle the problem. Traditional heuristic or Mixed-Integer and Constraint Programming approaches to such combinatorial optimization problems need to identify domain-specific heuristics, while reinforcement learning algorithms could discover domain-specific control policies automatically. Specifically, Deep Q-Learning applies Deep Neural Network (DNN) to approximate the reward function that provides feedback to the control policy. It can handle complicated problems with huge size of states like the optimal $k$-coverage charging problem. It is also fast to learn the reward function. We implement three other heuristic algorithms for comparison and then conduct extensive simulations with experimental data included. Results demonstrate that the proposed Deep Q-Learning algorithm consistently produces optimal solutions or closed ones and significantly outperforms other methods.

Index Terms—Wireless rechargeable sensor networks, $k$-Coverage, mobile chargers.

1 INTRODUCTION

Wireless rechargeable sensor networks, consisting of sensor nodes with rechargeable batteries and mobile chargers to replenish their batteries, have gradually become a promising solution to the bottleneck of energy limitation that hinders the wide deployment of wireless sensor networks (WSN) [1], [2], [3], [4]. The mobile charger scheduling and path optimization problem optimizes the trajectory of a mobile charger (e.g., a mobile robot) to maintain the operation of a WSN system. Variants are studied by considering different optimization goals, application scenarios, and constraints. Some research works extend the mobile charger scheduling and path optimization problem from one charger to multiple ones [5], [6] and static sensor nodes to mobile ones [7], [8].

Many of the prior works focused on the problem of maximizing the number of nodes charged within a fixed time horizon or energy constraint with the assumption that each sensor node contributes equally to the sensing quality of a network. However, a full area coverage is a basic requirement of WSN deployment to monitor a certain area. Multiple coverage, where each point of the field of interest (FoI) is covered by at least $k$ different sensors with $k > 1$ ($k$-coverage), is often applied to increase the sensing accuracy of data fusion and enhance the fault tolerance in case of node failures [9], [10], [11], [12], [13], [14]. Existing approaches to achieve $k$-coverage deploy a set of sensor nodes over a FoI either in a randomized way [15], [16] or with a regular pattern [13], [17]. Regular deployments need less sensor nodes than randomized ones, but they require centralized coordination and a FoI with regular-shape. A common practice is a high density of sensor nodes randomly distributed over the monitored FoI.

In this paper, we focus on the mobile charger scheduling scenario in which the $k$-coverage ability of a network system needs to be maintained. A node sends a charging request with its position information and a charging time window estimated based on its current residual energy and battery consumption rate. A mobile charger seeks a path maximizing the energy usage efficiency, i.e., minimizing the energy consumption on traveling, to charge sensor nodes and maintain $k$-coverage of the monitored area.

We formulate the optimal $k$-coverage charging problem of finding a feasible path for a mobile charger to charge a set of sensor nodes within their estimated charging time windows under the constraint of maintaining the $k$-coverage ability of the network system, with an objective of minimizing the energy consumption on traveling per tour. We show the hardness of the problem that even finding a feasible path is an NP-complete problem as the traveling salesman problem with time window (TSPTW), a classical combinatorial optimization problem, can be converted to a trivial case of the optimal $k$-coverage charging problem, i.e., $k = 1$ with no redundant nodes, and vice versa. The TSPTW is well-known for no polytime constant-factor approximation algorithm unless $P \neq NP$ [18].

We model the problem using directed graphs and apply the color coding technique to reduce the computational complexity to design an algorithm with a provable optimality guarantee that finds an exact solution to the problem. However, the computational complexity is still prohibitive for large size networks. We then introduce Deep Q-learning, an reinforcement learning algorithm to tackle the problem. Traditional heuristic or Mixed-Integer and Constraint Programming approaches to such combi-
natorial optimization problems need to identify domain-specific heuristics, while reinforcement learning algorithms could discover domain-specific control policies automatically. Specifically, Deep Q-Learning applies Deep Neural Network (DNN) to approximate the reward function that provides feedback to the control policy. It can handle complicated problems with huge size of states like the optimal \( k \)-coverage charging problem. It is also fast to learn the reward function. We implement three other heuristic algorithms for comparison and then conduct extensive simulations with experimental data included. Results demonstrate that the proposed Deep Q-Learning algorithm consistently produces optimal solutions or closed ones and significantly outperforms other methods.

The main contributions of this work are as follows:

- We formulate the optimal \( k \)-coverage charging problem.
- We prove the NP-hardness of the optimal \( k \)-coverage charging problem.
- We model the optimal \( k \)-coverage charging problem and apply the color coding and reinforcement learning techniques to design algorithms to tackle the problem with extensive simulations conducted to verify their effectiveness.

The rest of the paper is organized as follows. We review the closely related works in Section 2. We formulate the optimal \( k \)-coverage charging problem and analyze its hardness in Section 3. We present our algorithms in Sections 4 and 5 respectively. Section 6 presents the simulation results. Section 7 concludes the paper.

2 Related Works

The problem we study in the paper is closely related with the mobile charger scheduling and traveling salesman problem with time window. We give a brief review of the related works.

2.1 Mobile Charger Scheduling

The mobile charger scheduling problem optimizes the trajectory of a mobile charger (e.g., a mobile robot) to maintain the operation of a network system. There are many research works in this area with variants of the problem. Here we only list some of the most recent and representative works.

Shi et al. [1] consider the scenario of a wireless vehicle charger periodically traveling inside a sensor network and charging sensor nodes. They aim to minimize the time spent on path in each cycle. Multi-node wireless energy transfer technology is considered in [19]. The authors propose a cellular structure that partitions a two-dimensional plane into adjacent hexagonal cells such that a wireless vehicle charger visits the center of each cell and charges several sensor nodes at the same time. Xie et al. [20], [21] consider the scenario of co-locating a mobile base station in a wireless charging vehicle and investigate the optimization problems of entire system. Liang et al. [22] seek a charging tour that maximizes the total energy replenished to sensor nodes by a mobile charger with a limit energy capacity. Dai et al. [22] considers the scenario that both chargers and rechargeable devices are static. They study the optimization problem to maximize the overall effective charging energy of all rechargeable devices and minimize the total charging time without violating the electromagnetic radiation (EMR) safety.

Energy replenishment in robotic sensor networks is discussed in [7], [8]. Specifically, He et al. [7] aim to minimize the traveling distance of a mobile charger and keep all the mobile robots charged before their deadlines. Chen et al. [8] seek a charging path maximizing the number of mobile robots charged within a limited time or energy budget.

Multiple mobile chargers in a wireless rechargeable network raise new challenges but can work more efficiently. Collaborative mobile charging, where mobile chargers are allowed to intentionally transfer energy between themselves, is proposed in [23] to optimize energy usage effectiveness. Liang et al. [5] minimize the number of mobile charging vehicles to charge sensors in a large-scale WSN so that none of the sensors will run out of energy. Lin et al. [6] propose a real-time temporal and spatial-collaborative charging scheme for mobile chargers by combining temporal requirements as well as spatial features into a single priority metric to sort real-time charging requests.

2.2 Traveling Salesman Problem with Time Window

The Traveling Salesman Problem (TSP), a class of combinatorial optimization problems, has been extensively studied with many approximation algorithms proposed [24]. Deadline-TSP and TSP with time windows (TSPTW) are two relevant extensions of TSP, where all the nodes need to be visited before their deadlines or within their time windows with a minimum traveling cost. The added time constraint seems restrict the search space of solution, but it actually renders the problem even more difficult. Even finding a feasible path for such problems is NP-complete [25]. Some exact algorithms for TSPTW have been proposed in [25], [26], [27].

3 Network Model and Problem Formulation

Before giving a formal definition of the \( k \)-coverage charging problem studied in the paper, we first introduce the wireless sensor network model employed in this research.

3.1 Model of Wireless Sensor Network

We assume a set of stationary sensor nodes, \( V = \{v_i| 1 \leq i \leq n\} \), deployed over a planar FoI with locations, \( P = \{p_i| 1 \leq i \leq n\} \). For each sensor node \( v_i \), we assume a disk sensing model with sensing range \( r \). Specifically, denote a FoI, if the Euclidean distance between a point \( q \in A \) and node position \( p_i \) is within distance \( r \), i.e., \( \|p_i - q\|_2 \leq r \), then the point \( q \) is covered by the sensor \( v_i \), and we use \( v_i(p) = 1 \) to mark it, as shown in equation (1):

\[
    v_i(q) = \begin{cases} 
        1 & \text{if } \|p_i - q\|_2 \leq r \\
        0 & \text{otherwise} 
    \end{cases}
\]

(1)

Definition 1 (Full Coverage). If for any point \( q \in A \), there exists at least one sensor node covering it, i.e., \( \sum v_i(q) \geq 1 \), then area \( A \) is full covered.

Definition 2 (\( k \)-Coverage). If for any point \( q \in A \), there exist at least \( k \geq 1 \) sensor nodes covering it, i.e., \( \sum v_i(q) \geq k \), then area \( A \) is \( k \)-covered.

It is obvious that full coverage is a special case of \( k \)-coverage with \( k = 1 \).
3.2 Problem Statement

A sensor node \(v_i\) is equipped with a battery that needs to be recharged periodically with capacity \(B_i\). \(v_i\) decides to send a charging request before the leaving of a mobile charger from a service station. \(v_i\) will include in the request an estimated release time denoted as \(R_i\) and energy exhausted time denoted as \(D_i\) based on its current residual energy and energy consumption rate. Specifically, \(D_i\) is estimated as \(R_i + B_i/\beta_i\), where \(\beta_i\) represents the maximum battery consumption rate of \(v_i\). Note that nodes may have different energy consumption rates. Sensor \(v_i\) then sends a charging request \((id, p_i, R_i, D_i)\), where \(id\) is the sensor ID.

A mobile charger is responsible for charging sensors sending requests with an average moving speed \(s\). Considering that the charging time of a sensor can be reduced to less than 0.1s with current charging techniques that support charging power up to 120kW \(\text{[28]}\) and the charging time can be ignored compared to the time spent on path, we assume that charging of a sensor node to its full energy capacity is immediate. We also assume that the time spent on charging path is less than the operation time of sensors, so a sensor node only needs to be charged once in each tour. Unless an extreme dense sensor deployment, we consider that the charger charges sensor nodes one by one because the energy efficiency reduces dramatically with distance, e.g., the energy efficiency drops to 45 percent when the charging distance is 2m \(\text{[29]}\).

We consider a charging path scheduling and optimization problem. The mobile charger charges a number of sensor nodes to guarantee the \(k\)-coverage of area \(A\), and it seeks a path with a minimum energy consumption on traveling. In the meantime, each node needs to be charged within its time window. Specifically, the charging time is defined as the following:

**Definition 3 (Charging Time).** Denote \(P\) a charging path and \(tp(v_i)\) the charging time along \(P\) at node \(v_i\). If \(P\) goes from nodes \(v_i\) to \(v_j\), the charging time begins at node \(v_j\) is

\[
 tp(v_j) = \begin{cases} 
 \max\{tp(v_i) + \frac{d_{ij}}{s}, R_j\} & \text{if} \quad \frac{d_{ij}}{s} \leq D_j \\
 tp(v_i) + \frac{d_{ij}}{s} & \text{otherwise},
\end{cases}
\]

where \(d_{ij}\) the Euclidean distance between nodes \(v_i\) and \(v_j\) and \(s\) is the average speed of a charger.

3.3 Problem Formulation

The optimal \(k\)-coverage charging problem can be formulated as follows.

**Definition 4 (Optimal \(k\)-coverage charging problem).** Given a set of sensor nodes \(V = \{v_i\}_{1 \leq i \leq n}\), randomly deployed over a planar region \(A\) with locations \(P = \{p_i\}_{1 \leq i \leq n}\), the optimal \(k\)-coverage charging problem is to schedule a charging path \(P\)

\[
 \min |P| \quad \text{(3a)} \\
 \text{s.t.} \quad \sum_{i=1}^{n} r_i(v_i) \geq k, \forall q \in A. \quad \text{(3b)} \\
 tp(v_i) \in [R_i, D_i], \forall v_i \in P. \quad \text{(3c)}
\]

3.4 Problem Hardness

We prove the NP-hardness of optimal \(k\)-coverage charging problem below.

**Theorem 1.** The optimal \(k\)-coverage charging problem is NP-hard.

**Proof.** To prove the NP-hardness of the optimal \(k\)-coverage charging problem, we prove that the NP-hard problem: Traveling Salesman Problem with Time Window (TSPTW) can be reduced to the optimal \(k\)-coverage charging problem in polynomial time. The TSPTW is similar to the Traveling Salesman Problem (TSP) except that cities must be visited within a given time window.

We consider a trivial case of the optimal \(k\)-coverage charging problem where an initially full-covered network has no redundant nodes with \(k = 1\). It is straightforward to see that the solution of the TSPTW is also the solution of the optimal \(k\)-coverage charging problem and vice versa. Since the TSPTW is NP-hard, the optimal \(k\)-coverage charging problem is also NP-hard.

We then prove no Polynomial Time Approximation Solution of optimal \(k\)-coverage charging problem below.

**Theorem 2.** The optimal \(k\)-coverage charging problem has no Polynomial Time Approximation Solution (PTAS) unless \(P = NP\).

**Proof.** We have proved that the TSPTW can be converted to a trivial case of the optimal \(k\)-coverage charging problem. The solution of the TSPTW is also the solution of the optimal \(k\)-coverage charging problem and vice versa. Considering that TSPTW has no PTAS, i.e., even finding a feasible path for the TSPTW is an NP-complete problem \(\text{[25]}\), the optimal full-coverage charging problem has no PTAS either unless \(P = NP\).

4 Color-coding Algorithm

Given a sensor network with \(n\) sensor nodes randomly deployed over a FoI, we assume the network with a reasonable density such that the area is \(k\)-coverage initially. We design an algorithm based on the color coding technique introduced in \(\text{[30]}\) to find the optimal solution of the \(k\)-coverage charging problem.

4.1 Area Segmentation

The sensing range of a sensor node \(v_i\) is a disk-shape region centered at \(p_i\) with radius \(r_i\). These disk-shape sensing regions of a network divide a planar FoI \(A\) into a set of subregions, marked as \(A = \{a_i\}_{1 \leq i \leq m}\). Then \(\sum_{i=1}^{m} v_i(a_i)\) is the number of sensors with \(a_i\) within their sensing ranges. We assume \(\sum_{i=1}^{n} v_i(a_i) \geq k\) in the initial deployment of a network. Denote \(r(a_i)\) the number of sensor nodes sending charging requests among the \(\sum_{i=1}^{n} v_i(a_i)\) ones. Three cases exist for subregion \(a_i\):

- **Case I:** \(\sum_{i=1}^{n} v_i(a_i) - r(a_i) \geq k; a_i\) is still \(k\)-covered even if a charger ignores all the requests.
- **Case II:** \(\sum_{i=1}^{n} v_i(a_i) = k\) and \(\sum_{i=1}^{n} v_i(a_i) - r(a_i) < k; a_i\) charger needs to charge all the sensor nodes sending requests with area \(a_i\) within their sensing ranges to satisfy \(k\)-coverage.
- **Case III:** \(\sum_{i=1}^{n} v_i(a_i) > k\) and \(\sum_{i=1}^{n} v_i(a_i) - r(a_i) < k; a_i\) charger needs to choose at least \(k - \sum_{i=1}^{n} v_i(a_i) + r(a_i)\) sensor nodes from \(r(a_i)\) ones to charge to satisfy \(k\)-coverage.

An array denoted as \(T\) with size \(m\), is constructed to store the minimum number of sensors to charge for each \(a_i\). Specifically, \(T[i] = k - \sum_{i=1}^{n} v_i(a_i) + r(a_i)\). If the value is negative, we simply set \(T[i]\) to zero.

4.2 Time Discretization and Graph Construction

We divided the time window \([R_i, D_i]\) of a sensor node \(v_i\) into a set of time units \(\{t^i_k\}_{0 \leq k \leq D_i - R_i}\), where \(t^i_k = R_i \text{ and } t^i_{D_i-R_i} = D_i\). We then represent a node \(v_i\) with a set of discretized nodes \(\{v_i(t^i_k)\}_{0 \leq k \leq D_i - R_i}\), where \(v_i(t^i_k)\) represents node \(v_i\) at time
Note that we do not need to discretize sensor nodes without charging requests.

We construct a directed graph denoted as $G$ with vertices and edges defined as follows.

**Vertices.** The vertex set $V(G)$ includes all the discretized sensor nodes, i.e., $\{v_i(t^*_i)|1 \leq i \leq n, 0 \leq k \leq D_s - R_s\}$.

**Edges.** There exists a directed edge $v_i(t^*_i)v_j(t^*_j)$ from $v_i(t^*_i)$ to $v_j(t^*_j)$ in the edge set $E(G)$ if and only if the inequalities below hold:

$$t^*_i + \frac{d_{ij}}{s} \leq t^*_j,$$

where $k' = 0$, or

$$t^*_i + \frac{d_{ij}}{s} > t^*_j, t^*_i + \frac{d_{ij}}{s} \leq t^*_j,$$

where $k' > 0$.

From Def. 3 of charging time, Eqn. 4 means a charger comes to sensor node $v_j$ from $v_i$ before the release time of $v_j$, so the charging time is the release time $R_i$ of $v_j$. Similarly, Eqn. 5 means a charger comes to sensor node $v_j$ during its window time, so the charging time is the coming time.

**Theorem 3.** $G$ is a directed acyclic graph (DAG).

**Proof.** Suppose there exists a cycle in $G$. Assume vertices $v_i(t^*_i)$ and $v_j(t^*_j)$ are on the cycle. Along the directed path from $v_i(t^*_i)$ to $v_j(t^*_j)$, it is obvious that $t^*_i < t^*_j$. However, along the directed path from $v_j(t^*_j)$ to $v_i(t^*_i)$, we have $t^*_j < t^*_i$. Contradiction, so $G$ is a directed acyclic graph.

**Definition 5 (Feasible Path).** A path $P$ in $G$ is a feasible one if it passes no more than one discretized vertex of a sensor node and the charging time at each vertex is not inf. At the same time, charging along $P$ satisfies the k-coverage requirement of the given network.

### 4.3 The Color-coding algorithm

To guarantee the computed charging path passes no more than one discretized vertex of a sensor node, we apply the color coding technique introduced in [30] to assign each vertex a color. Specifically, we generate a coloring function $c_v : V \rightarrow \{1, \ldots, n\}$ that assigns each sensor node a unique node color. Each sensor node then assigns its node color to its discretized nodes. A path in $G$ is said to be colorful if each vertex on it is colored by a distinct node color. It is obvious that a colorful path in $G$ passes no more than one discretized vertex of a sensor node.

We add an extra vertex denoted as $v_0$ and connect it with directed edges to vertices in $G$, i.e., $\{v_i(t^*_i)|1 \leq i \leq n\}$. The length of edge $v_0v_i(t^*_i)$ is the Euclidean distance between the service station and sensor node $v_i$. We look for all colorful paths that start from vertex $v_0$ and charge a necessary number of sensor nodes to keep k-coverage.

$v_0$ checks neighbors connected with outgoing edges and sends $T$ to those contributing to the decrease of at least one entry array. Once a vertex $v_i(t^*_i)$ receives $T$, $v_i(t^*_i)$ checks the subregions within its sensing range and updates the corresponding entries of $T$. $v_i(t^*_i)$ also generates a color set $C = \{c(v_i(t^*_i))\}$ and stores with $T$, which indicates a colorful path of length $|C| = 1$ that connects $v_0$ and $v_i(t^*_i)$.

Suppose for each vertex, we have found a set of color sets representing colorful paths of length $i$ that connects $v_0$ and the vertex. For a vertex $v_i(t^*_i)$, we check each color set $C$ and each outgoing edge $v_j(t^*_j)$. If $c(v_j(t^*_j)) \not\in C$ and charging $v_j(t^*_j)$ helps decrease at least one entry of the $T$ associated with $C$, we add the color set $C = \{C + c(v_j(t^*_j))\}$ along with the updated $T$ to the collection of $v_j(t^*_j)$, which corresponds to a colorful path of lengths $i + 1$.

When there is no update of any node in the graph, we check the stored $T$s in each node and identify those with all zero entries. A color set associated with a $T$ with all zero entries represents a colorful path that is a feasible solution of the k-coverage problem.

We can easily recovered a path from a color set. The basic idea is to start from vertex $v_i(t^*_i)$ with a color set $C$ and associated all zero-entry $T$. We check the stored color sets of vertices connected to $v_i(t^*_i)$ with incoming edges. Assume we identify a neighbor node $v_j(t^*_j)$ storing a color set $C - c(v_j(t^*_j))$, then we continue to trace back the path from $v_j(t^*_j)$ with a color set $C - c(v_j(t^*_j))$.

When we trace back to $v_0$, we have recovered a feasible charging path. Among all feasible charging paths, the one with a minimal traveling distance is the optimal one.

Algorithm 1 summarizes the major steps of the algorithm.

### 4.4 Analysis

**Lemma 1.** The color-coding algorithm returns an optimal solution of the k-coverage charging problem, i.e., a feasible path maximizing the energy usage efficiency, if it exists.

**Proof.** We first prove that the color-coding algorithm returns a feasible path. The path returned by the algorithm passes a sensor node no more than once, otherwise the color set corresponding to the path must contain redundant colors. In the meantime, charging time is within the time window of each sensor node along the path, otherwise the directed edges along the path won’t exist. The k-coverage is also maintained, otherwise the array $T$ won’t have all zero entries. We can also show that the color-coding algorithm returns all feasible paths. In the $i$th iteration, each path with length $i$ will check all outgoing edges from current ending vertex and the opportunity to extend length to $i + 1$. Considering that the maximal length of a feasible path, if exists, is no more than $n$, all feasible paths will be found after $n$ iterations. The one with a minimal traveling distance is the optimal charging path.

**Lemma 2.** The optimal solution can be found in $O(2^{O(n)}E)$ worst-case time, where $n$ is the size of sensor nodes, and $E$ is the number of edges in $G$.

**Proof.** Suppose for each vertex, the color-coding algorithm has found a set of color sets representing feasible paths with partial length $i$ that connects $v_0$ and the vertex. Note that the algorithm does not record the feasible paths but the color sets appearing on paths. Each vertex has a collection of at most $\binom{n}{2}$ color sets where $n$ is the number of colors in $G$, i.e., the size of sensor nodes. The algorithm checks every color set that belongs to the collection of the vertex and every outgoing edge whether the current path with length $i$ can be extended to $i + 1$. Since the length of a feasible path is bounded by $n$, the number of operations is bounded by $\sum_{x=0}^{n-1}(\binom{n}{2})x^{x-1}y = (x + y)^n$, the number of operations is bounded by $2^{O(n)}E$.

### 4.5 Weaknesses of the Color-coding Algorithm

The weaknesses of the Color-coding algorithm is its high computational complexity. It needs to list all feasible paths to find the optimal one so it is prohibitive for large instances.
Algorithm 1 The Color-coding Algorithm

Input:
- $c_v: V \rightarrow \{1, ..., n\}$: a coloring function assigns a unique color to each sensor node,
- $T$: an array constructed in Sec. 4.1,
- $G$: a DAG constructed in Sec. 4.2.

Output:
- $P$: A charging path satisfying Def. 3 of optimal $k$-coverage charging problem.
1. for all $v_i(t_i^t) \in G(V)$ do
2. $c(v_i(t_i^t)) = c_v(v_i)$.
3. end for
4. Add vertex $v_0$ to $G(V)$.
5. for $1 \leq i \leq n$ do
6. Connect $v_0$ with a directed edge to $v_i(t_0^0)$.
7. end for
8. for $1 \leq i \leq n$ do
9. if Charging $v_i$ updates $T$ then
10. $v_i$ copies $T$ to $v_i(t_i^0)$.
11. $v_i(t_i^0)$ updates $T$.
12. $v_i(t_i^0)$ generates a color set $C = \{c(v_i(t_i^0))\}$ and stores with $T$.
13. end if
14. end for
15. while updated do
16. updated = false.
17. for all $v_i(t_i^t) \in G(V)$ do
18. for all $C$ stored at $v_i(t_i^t)$ do
19. for all $v_i(t_i^t)v_j(t_j^t) \in G(E)$ do
20. if $c(v_j(t_j^t)) \notin C$ then
21. if Charging $v_j$ updates $T$ then
22. Store the color set $C = \{C + c(v_j(t_j^t))\}$ and the updated $T$ in $v_i(t_i^t)$.
23. updated = true.
24. end if
25. end if
26. end for
27. end for
28. end while.
29. for all $v_i(t_i^t) \in G(V)$ do
30. for all $T$ stored in $v_i(t_i^t)$ do
31. if $T$ all zero-entry then
32. $P = \text{TracePath}(C, v_i(t_i^t)) \{C$ the color set stored with $T\}$
33. end if
34. end for
35. end for
36. end for
37. return $\min\{P\}$.

Algorithm 2 TracePath Algorithm

Input:
- $C$: a color set;
- $v$: $v \in G(V)$;

Output:
- $P$: a feasible charging path satisfying $k$-coverage charging problem.
1. Add $v$ to $P$.
2. while $v \neq v_0$ do
3. for all $\bar{v} \in$ incoming neighbors of $v$ do
4. if $\{C - c(v)\} \in$ colors sets stored in $\bar{v}$ then
5. $C = \{C - c(v)\}$
6. $v = \bar{v}$
7. Add $v$ to $P$.
8. end if
9. end for
10. end while

5 Deep Q-Learning Algorithm

5.1 Motivation

We have showed in Sec. 3.4 that the TSPTW, a classical combinatorial optimization problem, can be converted to a trivial case of the optimal $k$-coverage charging problem and vice versa. They have no polynomial-time constant-factor approximation algorithm unless $P = NP$.

A combinatorial optimization problem searches for an optimal solution of an objective function under a set of constraints. The domain of the objective function is discrete, but prohibitive for an exhaustive search. A feasible solution of the problem satisfies the set of constraints. An optimal solution, also a feasible one, minimizes the value of the objective function.

Heuristic or Mixed-Integer and Constraint Programming approaches to combinatorial optimization problems need to identify domain-specific heuristics, while reinforcement learning algorithms could discover domain-specific control policies automatically. Specifically, an reinforcement learning agent explores different solutions and evaluates the qualities as rewards. Such rewards are then provided as feedback to improve the control policy.

We introduce Deep Q-learning, an reinforcement learning algorithm to tackle the optimal $k$-coverage charging problem. “Q” stands for the quality or reward of an action taken in a given state. In Q-learning, an agent maintains a state-action pair function stored as a Q-table $Q(S,A)$ where $S$ is a set of states and $A$ is a set of actions. A Q-value of the table estimates how good a particular action will be in a given state, or what reward the action is expected to bring.

Q-Learning is a model-free reinforcement learning algorithm, so there is no need to find all the combinations to check the existence of a solution. An agent will choose next state based on current one and stored state-action rewards, requiring less computation and storage space compared with model-based reinforcement learning algorithms. Q-Learning is a also temporal-difference reinforcement learning algorithm. An agent can learn online after every step, and even from an incomplete sequences (a situation that leads to unfeasible solution). In theory, Q-learning has been proven to converge to the optimal Q-function for an arbitrary target policy given sufficient training.

Deep Q-Learning applies Deep Neural Network (DNN) to approximate the Q-function, i.e., a deep neural network that takes
a state and approximates the Q-value for each state-action pair. It can handle the situation when the number of states of a problem is huge. It is also faster to learn the reward value of each state-action pair than Q-learning. Therefore, deep Q-learning can handle more complicated problems compared with Q-Learning.

An end-to-end deep Q-learning framework introduced in [33] automatically learns greedy heuristics for hard combinatorial optimization problems on graphs. We modify the framework in [33] and combine with deep Q-learning in [35] to tackle the optimal k-coverage charging problem.

5.2 Graph Construction

We first construct a directed graph denoted as $G$ as input for the Deep Q-Learning algorithm. The vertices and edges of $G$ are defined as follows:

**Vertices.** The vertex set $V(G)$ includes all the sensor nodes, i.e., $\{v_i | 1 \leq i \leq n\}$ and a start vertex denoted as $v_0$. Each vertex $v_i$ has a time window $[R_i, D_i]$. The location of $v_0$ is the service station and the time window of $v_0$ is $[0, \infty)$. Note that the time window of sensor nodes without sending any request is set as $[\infty, 0]$. 

**Edges.** For any sensor nodes $v_i$ and $v_j$ in $V(G)$, $d_{ij}$ is the euclidean distance between $v_i$ and $v_j$. There exists an edge $v_iv_j$ in $V(G)$ if and only if the inequality below holds

$$R_i + \frac{d_{ij}}{s} \leq D_j$$

where $s$ is the average speed of a charger.

If a charging path $P$ goes from node $v_i$ to node $v_j$, the charging time beginning at node $v_i$ is given by Def.[3]

**Definition 6 (Feasible Path).** A path $P$ in $G$ is a feasible one if it starts from and ends at $v_0$, and has no repeated vertex and the charging time at each vertex is not inf. At the same time, charging along $P$ satisfies the k-coverage requirement of the given network.

5.3 Deep Q-Learning Formulation

We define the states, actions, rewards, and stop function in the deep Q-learning framework as follows:

**States:** A state $S$ is a partial solution $S \subseteq V(G)$, an ordered list of visited vertices. The first vertex in $S$ is $v_0$.

**Actions:** Let $\bar{S}$ contains vertices not in $S$ and has at least one edge from vertices in $S$. An action is a vertex $v$ from $\bar{S}$ returning the maximum reward. After taking the action $v$, the partial solution $S$ is updated as

$$(S, v), \text{where } v = \arg \max_{v \in \bar{S}} Q(S, v)$$

$(S, v)$ denotes appending $v$ to the best position after $v_0$ in $S$ that introduces the least traveling distance and maintains all vertices in the new list a valid charging time.

**Rewards:** The reward function $R(S, v)$ is defined as the change of the traveling distance when taking the action $v$ and transitioning from the state $S$ to a new one $S'$. Assume $v_i, v_j$ are two adjacent vertex in $S$, $v_0$ is the first vertex in the $S$, and $v_j$ is the last vertex in the $S$.

The reward function $R(S, v_k)$ is defined as follows:

$$R(S, v_k) = \begin{cases} -\min(d_{ik} + d_{kj} - d_{ij}, d_{ik} + d_{jk} - d_{ij}) & i_{S}(v) \neq \inf \\ -\inf & \text{otherwise} \end{cases}$$

5.4 Deep Q-Learning Algorithm

Algorithm [3] summarizes the major steps of our algorithm. Briefly, $Q(S, v; \Theta)$ is parameterized by a deep network with a parameter $\Theta$ and learned by one-step Q-learning. At each step of an episode, $\Theta$ is updated by a stochastic gradient descent to minimize the squared loss:

$$f(y_l - Q(S_i, v_l; \Theta))^2$$

where

$$y_l = \begin{cases} R(S_l, v_l) + \gamma \max_{v' \in \bar{S}_m} Q(S_{l+1}, v'; \Theta) & S_{l+1} \text{is non-terminal} \\ R(S_l, v_l) & \text{otherwise} \end{cases}$$

We use experience replay method in [35] to update $\Theta$, where the agent’s experience in each step is stored into a dataset. When we update $\Theta$, we sample random batch from dataset that is populated from previous episodes. The benefits include increasing data efficiency, reducing correlations between samples, and avoiding oscillations with the parameters.

**Algorithm 3 Deep Q-learning Algorithm**

1: Initialize replay memory $H$ to capacity $C$
2: for each episode do
3: Initialize state $S_1 = (v_0)$
4: for step $m = 1$ to $M$ do
5: Select $v_m = \arg \max_{v \in \bar{S}_m} Q(S_m, v; \Theta)$ with probability $1 - \epsilon$
6: Otherwise select a random vertex $v_m \in \bar{S}_m$
7: Add $v_m$ to partial solution: $S_{m+1} := (S_m, v_m)$
8: Calculate reward $R(S_m, v_m)$ by (8)
9: Store tuple $(S_m, v_m, R(S_m, v_m), S_{m+1})$ to $H$
10: Sample random batch $(S_l, v_l, R(S_l, v_l), S_{l+1})$ from $H$
11: Update the network parameter $\Theta$ by (9)
12: if $S_{m+1}$ satisfy the stop function then
13: Break
14: end if
15: end for
16: end for

6 PERFORMANCE EVALUATION

6.1 Simulation Settings

We set up an Euclidean square $[500, 500] m^2$ as a simulation area and randomly deploy sensor nodes ranging from 32 to 75 in the square such that the area is $k$-coverage initially where $k$ varies from 2 to 4. The sensing range $r$ is 135m. The base station and service station of charger are co-located in the center of the square. A charger with a starting point from the service station has an average traveling speed $5 m/s$ and consumes energy $600 J/m$ [4]. The battery capacity of each sensor is 10.8KJ [4]. A sensor sends a charging request before the leaving of the charger from the service station. The sensor will include in the request the estimated release and energy exhausted times based on its current residual
energy and energy consumption rate. To simulate such request, we consider the release time of a sensor is a uniform random variable in $[0, 1000]$ s, and the window width $W_i$ is also a uniform random variable between $[50, 100]$ s. The energy exhausted time $D_i = R_i + W_i$. In Sec. 6.6, we choose the energy consumption rate from the historical record of real sensors in [36] where the rate is varying according to the remaining energy and generate time windows of charging requests.

We add three heuristic algorithms for comparison. Sec. 6.2 explains the implementation details.

Computing time and traveling energy are important metrics to evaluate the performances of our algorithms and others. Different parameters of a network setting may affect the performance. Therefore, we study the impact of the three parameters: the coverage requirement $k$, the size of sensor network $n$, and the percentage of sensor nodes sending requests $\alpha$ in Secs. 6.3, 6.4, and 6.5, respectively.

Considering that a feasible solution does not always exist giving a set of charging requests with a restrictive rule of time window, Sec. 6.7 discusses the relaxation of requirement of time window to maintain the $k$-coverage of a given network.

### 6.2 Comparison Algorithms

We add three more heuristic algorithms for comparison: Ant Colony System (ACS) based algorithm, Random algorithm, and Greedy algorithm.

ACS algorithm solves the traveling salesmen problem with an approach similar to the foraging behavior of real ants [37], [38], [39]. Ants seek path from their nest to food sources and leave a chemical substance called pheromone along the paths they traverse. Later ants sense the pheromone left by earlier ones and tend to follow a trail with a stronger pheromone. Over a period of time, the shorter paths between the nest and food sources are likely to be traveled more often than the longer ones. Therefore, shorter paths accumulate more pheromone, reinforcing these paths.

Similarly, ACS algorithm places agents at some vertices of a graph. Each agent performs a series of random moves from current vertex to a neighboring one based on the transition probability of the connecting edge. After an agent has finished its tour, the length of tour is calculated and the local pheromone amounts of edges along the tour are updated based on the quality of the tour. After all agents have finished their tours, the shortest one is chosen and the global pheromone amounts of edges along the tour are updated. The procedure continues until certain criteria are satisfied. When applying ACS algorithm to solve the traveling salesmen problem with time window, two local heuristic functions are introduced in [34] to exclude paths that violate the time-window constraints.

We modify ACS algorithm introduced in [34] for the optimal $k$-coverage charging problem. Agents start from and end at $v_0$. Denote $\tau_{ij}(t)$ the amount of global pheromone deposited on edge $v_iv_j$ and $\Delta\tau_{ij}(t)$ the increased amount at the $t^{th}$ iteration. $\Delta\tau_{ij}(t)$ is defined as

$$\Delta\tau_{ij}(t) = \begin{cases} \frac{1}{k} & \text{if } v_iv_j \in P^* \\ 0 & \text{otherwise} \end{cases}$$

(10)

where $L^*$ is the traveling distance of the shortest feasible tour $P^*$ at the $t^{th}$ iteration. Global pheromone $\tau_{ij}(t)$ is updated according to the following equation:

$$\tau_{ij}(t) = (1 - \theta)\tau_{ij}(t-1) + \theta\Delta\tau_{ij}(t),$$

(11)

where $\theta$ is the global pheromone decay parameter. The local pheromone is updated in a similar way, where $\theta$ is replaced by a local pheromone decay parameter and $\Delta\tau_{ij}(t)$ is set as the initial pheromone value.

We also modify the stop criteria of one agent such that the traveling path satisfies the requirement of $k$-coverage, or the traveling time of current path is inf, or the agent is stuck at a vertex based on the transition rule.

Random and Greedy algorithms work much more straightforward. The Random algorithm randomly chooses a next node with an outgoing edge from current one to charge each time. The Greedy algorithm always chooses the nearest node with an outgoing edge from current one. Random and Greedy algorithms terminate either they find a feasible path or they are locally stuck. Note that for ACS and Random algorithms, we always run multiple times and choose the best solution.

### 6.3 Coverage Requirement

We set the number of sensor nodes $n = 64$ and the percentage of nodes sending requests $\alpha = 0.45$. The coverage requirement $k$ varies from 2 to 4. Table I gives the performances of different algorithms. The higher the coverage requirement $k$ is, the more the sensor nodes need to be charged. The trend is obvious in Table I that the traveling energy increases with the increase of the coverage requirement $k$. The Random and Greedy algorithms can only find a feasible path when $k$ is small. The ACS algorithm, better than the Random and Greedy algorithms, can find a feasible path, even optimal one when $k$ is small. The DQN algorithm can not only find a feasible path, but also an optimal one for all $k$s. However, the computing time of DQN, including the training time, increases slightly with the increase of $k$. By contrast, the computing time of the exact algorithm increases exponentially. Overall, the performance of DQN significantly outperforms all other algorithms.

| Algorithm | $k$ | Computation Time (s) | Feasible Path Found | Traveling Energy (kJ) |
|-----------|----|----------------------|---------------------|----------------------|
| Exact     | 4  | 0.0032               | Yes                 | 477                  |
| DQN       | 3  | 15                   | Yes                 | 477                  |
| ACS       | 2  | 8                    | Yes                 | 477                  |
| Random    | 4  | 0.0004               | No                  | –                    |
| Greedy    | 3  | 0.0004               | No                  | –                    |

Table 1
### 6.4 Size of Sensor Network

We set the coverage requirement \( k = 3 \) and the percentage of nodes sending requests \( \alpha = 0.45 \). The size of a sensor network varies from 48 to 75. There are more redundant sensor nodes in a network with the increase of network size under the same \( k \) and \( \alpha \). Therefore, the traveling energy in Table 2 decreases with the increase of the network size \( n \). Both the Random and Greedy algorithms fail to detect a feasible path. The DQN algorithm can not only find a feasible path, but also an optimal one for all \( n \). The ACS algorithm can also detect a feasible path close or equivalent to the optimal one with a very stable computing time.

### 6.5 Percentage of Nodes with Requests

Table 3 and Table 4 give the performances of different algorithms with the percentage of nodes sending requests \( \alpha \) varying from 0.2 to 0.8 under two network settings: \( n = 32 \) and \( k = 2 \), and \( n = 48 \) and \( k = 3 \). With the increased percentage of nodes sending requests, the traveling energy increases in both network settings. The Random and Greedy algorithm fail to detect a feasible path. The DQN algorithm can detect a feasible path very close or equivalent to the optimal one with a reasonable computing time. However, the Exact algorithm runs out of the memory when \( n = 48 \), \( k = 3 \), and \( \alpha = 0.8 \). It is worth mentioning that the Random algorithm performs better than the Greedy one in some case because the nearest node may not be a good choice and a randomly chosen one may lead the searching of feasible path out of stuck.

### 6.6 Experimental Simulation

we use the energy consumption rate from the historical record of real sensors in [36] where the rate is varying according to the remaining energy. Table 5 gives the performances of different algorithms with an increasing percentage of nodes sending requests \( \alpha \) under the network setting \( k = 3 \) and \( n = 48 \). Since the energy consumption rate in [36] is small, the generated time window is wide. A wide time window increases the possibility of finding a feasible path for heuristic algorithms, but increases the computing time for the exact algorithm with a larger size of edges in \( G \). The DQN algorithm consistently outperforms all other heuristic ones.

### 6.7 Relax the Release Time when Feasible Path not exists

A feasible solution does not always exist giving a set of charging requests with a restrictive rule of time window. In that case, we have to relax requirement of time window to maintain the \( K \) coverage of a given network. A straightforward solution is to ignore the release time and charge sensor nodes before their energy exhausted time. Therefore we do not need to modify the proposed algorithms except setting all the release times to 0.

We study 3 various cases to evaluate the performances of different algorithms. The settings of the three cases are \( n = 16, k = 1, \alpha = 0.45, n = 32, k = 2, \alpha = 0.45 \) and \( n = 48, k = 3, \alpha = 0.45, \)
TABLE 4
Performance under different percentage of nodes sending requests $\alpha$ when $k = 3, n = 48$

| Algorithm | $\alpha$ | Computation | Feasible | Traveling |
|-----------|----------|--------------|----------|-----------|
|           |         | Time (s)     | Path Found | Energy (kJ) |
| Exact     | 0.02    | Yes          | 486      |
| DQN       | 0.2     | Yes          | 486      |
| ACS       | 0.0003  | No           | –        |
| Greedy    | 0.0003  | No           | –        |
| Exact     | 30      | Yes          | 1068     |
| DQN       | 24      | Yes          | 1068     |
| ACS       | 18      | Yes          | 1233     |
| Random    | 0.0005  | No           | –        |
| Greedy    | 0.0004  | No           | –        |
| Exact     | 20858   | Yes          | 1626     |
| DQN       | 410     | Yes          | 1836     |
| ACS       | 57      | No           | –        |
| Random    | 0.001   | No           | –        |
| Greedy    | 0.001   | No           | –        |

TABLE 5
Performance under different percentage of nodes sending requests $\alpha$ with real energy consumption rate

| Algorithm | $\alpha$ | Computation | Feasible | Traveling |
|-----------|----------|--------------|----------|-----------|
|           |         | Time (s)     | Path Found | Energy (kJ) |
| Exact     | 2       | Yes          | 486      |
| DQN       | 0.2     | Yes          | 486      |
| ACS       | 0.0003  | Yes          | 501      |
| Greedy    | 0.0003  | Yes          | 516      |
| Exact     | 1736000 | Yes          | 846      |
| DQN       | 32      | Yes          | 846      |
| ACS       | 0.4     | Yes          | 846      |
| Random    | 0.0004  | No           | –        |
| Greedy    | 0.0006  | Yes          | 1134     |
| Exact     | –       | –            | –        |
| DQN       | 0.6     | Yes          | 912      |
| ACS       | 316     | Yes          | 1275     |
| Random    | 0.001   | Yes          | 2514     |
| Greedy    | 0.002   | Yes          | 1551     |
| Exact     | –       | –            | –        |
| DQN       | 0.8     | Yes          | 1320     |
| ACS       | 1550    | Yes          | 2106     |
| Random    | 0.01    | Yes          | 4146     |
| Greedy    | 0.002   | Yes          | 2616     |

TABLE 6
Three cases when feasible path does not exist

| Algorithm | Computation | Feasible | Traveling |
|-----------|--------------|----------|-----------|
|           | Time (s)     | Path Found | Energy (kJ) |
| Exact     | 0.013        | Yes       | 501      |
| DQN       | 2            | Yes       | 501      |
| ACS       | 6            | Yes       | 501      |
| Random    | 0.0003       | Yes       | 618      |
| Greedy    | 0.0002       | Yes       | 648      |
| Exact     | 0.014        | Yes       | 732      |
| DQN       | 2            | Yes       | 732      |
| ACS       | 7            | Yes       | 732      |
| Random    | 0.0003       | Yes       | 777      |
| Greedy    | 0.0002       | Yes       | 801      |
| Exact     | 1780000      | Yes       | 900      |
| DQN       | 381          | Yes       | 900      |
| ACS       | 125          | Yes       | 1038     |
| Random    | 0.0005       | No        | –        |
| Greedy    | 0.0006       | No        | –        |

7 CONCLUSIONS
We explore the mobile charger scheduling and path optimization problem that the $k$-coverage ability of a wireless rechargeable sensor network system needs to be maintained. We formulate the problem and show the hardness that even finding a feasible path for the trivial case of the problem is an NP-complete one with no polytime constant-factor approximation algorithm. We model the problem and apply the color coding and reinforcement learning techniques to design algorithms to tackle the optimal $k$-coverage charging problem. We also implement three other heuristic algorithms for comparison. Extensive simulations with experimental data included demonstrate that the proposed Deep Q-Learning algorithm consistently produces optimal solutions or closed ones and significantly outperforms other methods.

ACKNOWLEDGMENTS
X. Li and M. Jin are partially supported by NSF CNS 1018306.

REFERENCES
[1] Y. Shi, L. Xie, Y. T. Hou, and H. D. Sherali, “On renewable sensor networks with wireless energy transfer,” in INFOCOM, 2011 Proceedings IEEE, pp. 1350–1358, IEEE, 2011.
[2] Y. Yang and C. Wang, Wireless Rechargeable Sensor Networks. Springer, 2015.
[3] F. Y. C. Wang, J. Li and Y. Yang, “A mobile data gathering framework for wireless rechargeable sensor networks with vehicle movement costs and capacity constraints,” IEEE Trans. Comput., vol. 65, no. 8, p. 2411–2427, 2016.
[4] W. Liang, Z. Xu, W. Xu, J. Shi, G. Mao, and S. K. Das, “Approximation algorithms for charging reward maximization in rechargeable sensor networks via a mobile charger,” IEEE/ACM Transactions on Networking, vol. 25, no. 5, pp. 3161–3174, 2017.
