Location Updating Scheme of Sink Node based on Topology Balance and Reinforcement Learning in WSN

XINDI WANG¹, QINGFENG ZHOU¹, CHUNXIAO QU¹, GAO CHEN¹, JUNJUAN XIA²

¹School of Electric Engineering and Intelligentization, Dongguan University of Technology, China.
²School of Computer Science, Guangzhou University, China.
Corresponding author: Qingfeng Zhou (e-mail: enqfzhou@ieee.org)

This work was supported by the Science and Technology Planning Project of Guangdong Province under Grant 2016B010108002, the NSFC under Grant 61471156, the Science and Technology Program of Guangzhou under Grant 201807010103, and the Starting Fund of Dongguan University of Technology under Grant DGUT-G200906-49.

ABSTRACT This paper proposes a scheme for updating the location of the sink node to balance the network topology when a Wireless Sensor Network (WSN) is scaled up. We divide the proposed location update scheme into two steps, namely searching the optimal location and designing the pathfinding algorithm. For the former, to find the optimal location of the sink node simply and efficiently, we only consider the information of the expanded longer paths and some key nodes instead of the global information of the entire network, which is easy to implement with a low computational load. Then, considering the general unattended application scenario, we propose an improved reinforcement learning (RL) algorithm for the sink node to calculate a feasible efficient path, and then the sink node follows the path to reach the optimal location. Finally, through simulations, we demonstrate the optimal position of the sink node in expanded scenarios, and successfully let the sink node learn the effective pathfinding method to reach the target position. A large number of simulation results verify the efficiency and effectiveness of our proposed scheme from the perspective of the efficiency of the pathfinding algorithm.

INDEX TERMS Network scaling; Location update; Pathfinding; Reinforcement learning; Wireless sensor network.

I. INTRODUCTION

In recent years, the wireless data rate has been explosively increasing, and many techniques have been proposed to meet the requirements [1]–[3]. Among these techniques, Wireless Sensor Network (WSN) [4], [5] provides an unattended manner to sense the status of the environment by using a large number of sensor nodes. Taking into account the practical application of WSN, more and more studies focus on the actual implementation of the WSN protocols [6], [7], which brings tough challenges in protocol design, such as the dynamic characteristic of the WSN scenario. In a typical WSN, the scale of the network is often dynamically adjusted according to changes in monitoring objectives, that is, when the monitoring area is scaled-up or increasing the monitoring quantity in some local areas [8], the number of nodes and the corresponding topology structure in the network may have to change synchronously.

However, since many existing studies only consider the static application scenarios, thus the dynamic characteristics of the actual WSN will pose a great challenge to the practical application of these proposed approaches. For example, the data aggregation schemes [9]–[12], which provide a uniform data gathering strategy within each node in the network, usually perform well under the pre-defined topology structure. Therefore, as the original network topology is scaled-up by adding some new nodes in the network, the data aggregation protocols which are designed according to the prior scenario may be inefficient or even fail completely. Wang [13] proposed a data aggregation scheme TADA under the tree-like topology structure, which can obtain an excellent performance as the paths in the topology are balanced. Also, Lan [14] proposed a hierarchical data aggregation protocol based on the cluster topology in WSN, the unbalanced structure between each cluster will cause partial nodes consuming too much energy and die prematurely. Wu [15] proposed a cluster grouping scheme to collect the data in the network, in which the balance property of the cluster size is directly related to the energy consumption of the sensor nodes. Apparently,
the changes in topology structure will seriously affect the performance of the applied protocols.

Since the primary objective of WSN is to pass the sensed data from nodes to the sink node, thus the location of the sink node can be treated as an important factor affecting the network performance. So, changing the location of the sink node in the scaled-up scenario can be treated as a good solution to improve the performance of the WSN schemes in the dynamic scenario. In fact, there already exist many studies considering the impact of the optimal location of the sink node on the network performance. The study in [16] proposed an efficient scheme for selecting the best location for the sink node, which limits the select range to the graph center or the centroid of the backbone connecting nodes. The approach in [17] proposed an optimization problem about the location of the sink node and develops an efficient grid-based algorithm to solve this problem. The results can prolong the life time in simulation and theory analysis validation. The approach in [18] studied the problem that the impact of the sink placement on the data transmission amount and lifetime of the network, and further proposes two data aggregation schemes based on existing placement of the sink node. The approach in [19] proposed to search the optimal location of the sink node to solve the energy hole problem in WSN. The optimal location of the sink node will help to save the energy consumption for the nearest nodes and optimize the cluster structure in the network. In [20], authors proposed to find out the energy efficiency locations of the sink node in the data aggregation scenario and design a schedule to move the sink node between the obtained optimal locations, which shows an energy-efficiency performance in a statics scenario. Therefore, for the scaled-up scenario, updating the location of the sink node according to the current scenario [16], [17] may be a good solution to address the problem caused by the dynamic changes of the WSN.

Besides on the above works, how to move the sink node to the new location is another challenge. Conventionally, many studies design the WSN scheme with a mobile sink node, which can easily solve this problem. In this regard, a mobile device can act as the sink node and update its traveling paths according to the scale of the scenario. Thus, for the protocols which use the mobile sink node in the scenario, the main consideration is on the path planning problem of sink node movement. In [21], the authors put forward a mobility-assisted data collection method, which considers the number of sink nodes and the routing path of each node at the same time. In [22], to accomplish the target that balances the energy consumption in wireless sensor network, the authors proposed to add the mobility to all nodes including the sink node and schedule the actions of each node correspondingly. Obviously, the task of the sink node location update can be solved by using a mobile sink node. Meanwhile, as we all known, the common used mobile sink node applied in the current WSN is an unmanned aerial vehicle (UAV) [23], [24], which is remotely controlled by ground operators. In this case, the sensor nodes in the scenario just need to store the sensed data in its memory temporarily and wait the UAV arriving at the nearby positions for gathering the data. The approach in [23] proposed to use a UAV to establish the wireless connectivity between each base stations from the perspective of the coverage, the main target is to minimize the required number of base stations in the fixed region. The approach in [24] proposed an energy-efficient communication scheme via optimizing the UAV’s trajectory, obtaining a good performance in communication throughput and energy consumption. However, the application of the UAV may not feasible in some actual scenarios, such as the monitoring task in the sensitive military areas or some remote regions which are out of the controlling range. Therefore, to realize the unattended monitoring task in the WSN with a mobile sink node, we consider to use a smart robot acted as the sink node to update its location periodically and automatically in the scaled-up network. The robot, which is employed in our scenario, can choose whether to update its location according to the timely status of the current scenario. As the scale of scenario expands beyond a certain threshold, the robot will figure out the optimal location and explore how to reach the location using the collected information of the scenario.

Actually, this idea has potential to come true due to the development of the artificial intelligence (AI) algorithms [12], [25], [26], which assigns the robot a strong power to learn the path planning without manual assistance. The AI algorithms have been utilized in many aspects of the wireless communication technologies, such as MIMO [27], [29], cognitive radio [28] and so on. Among many AI algorithms, the reinforcement learning (RL) [30] algorithm has shown good performance for self-learning in many scenarios, such as the traffic signal control [31], [32] and resource allocation [33], [34]. The highlight of the RL algorithm is that it can help the agent to choose the better action in each step through learning the feedback from the environment without any prior training instances.

So, to accomplish our target, the popular and powerful algorithm, reinforcement learning (RL), is employed this work to help the sink node sense the environment and obtain an action list (i.e., turn left, turn right, forward, backward). Following the list, the sink node can reach the target location finally. Meanwhile, considering the practical application of the wireless sensor network which has limited time complexity and historical data, the more complicated algorithm, deep reinforcement learning [39], is not considered in our proposed scenario. In contrast, the two classic RL algorithms, Q-learning and Sarsa algorithm, are more applicable to the actual scenario of WSN. Among them, the Sarsa algorithm is more robust than Q-learning, which is helpful to avoid risks in practical applications. However, directly applying classical Sarsa algorithm to the pathfinding task may not achieve a good performance, this is because that the actual environment of the WSN scenario is much complicated, even for the improved algorithm, Sarsa(A) [30]. Thus, in recent years, some improved RL-based studies have been proposed to apply in the intelligent pathfinding application.
Amit [35] proposed an improved Q-learning algorithm to let the robot reach the destination as soon as possible. However, this algorithm may get into trouble in the scenario in which there exists a continuous obstacles in the direction from the current location to the destination. Also, the same problem will occur in the improved algorithm in [36]. Meanwhile, there are also some other researches [37], [38] studying on pathfinding methods for dynamic scenarios or multiple sink nodes, but they are beyond the scope of this article.

In this paper, taking both the sink placement update method and the RL algorithm into account, we aim to address the topological imbalance problems caused by the scenario scaling-up in WSN. First, we design a novel location update strategy, which makes full use of the key information of the scenario, to obtain the new location of the sink node efficiently. Then, combining with the obtained location of the sink node, we put forward a window-based directional Sarsa (λ) algorithm (WDS) to help the sink node find a feasible path in the pathfinding task.

The rest of the paper is organized as follows. In Section. II, we put forward the scenario model and the problem formulation in this paper. In Section. III, we elaborate the method of searching the optimal location of the sink node in the current scenario. In Section. IV, we elaborate the WDS algorithm. In Section. VI, we conclude this paper and suggest some further works.

II. SCENARIO MODEL AND PROBLEM FORMULATION

In this section, we introduce the scenario model and put forward the formulation of the problem.

A. SCENARIO MODEL

Assuming the monitoring scenario is a rectangular area with \( N \) randomly and uniformly distributed sensor nodes and the initial location of the sink node is set to the center of gravity of the network. Meanwhile, a balanced tree-like topology is generated based on BMST algorithm [13] in the existing scenario, which can evenly configure the links in the network by limiting the length of the longest path. The graphical description of the network is given as follows.

In Fig. 1, each node in the tree-like topology has a uniquely determined parent node, thus the data transmission direction of each node is also fixed and unique. Due to the fact that each data aggregation process starts at the leaf node of the path and terminates till the aggregated data sent to the sink node, thus we model the topology of the network as a directed acyclic graph, i.e., \( G(V, E) \), where the vertices \( V \) consists of \( N \) sensor nodes and a sink node, and \( E \) is the link between each pair of neighbor nodes. In addition, we do not consider the case where noise interference or data loss may occur during the data transmission process here, which is out of the scope of this paper.

B. PROBLEM FORMULATION

As described in Section. II, the existing scenario is scaled-up as a small number of new nodes join in the network, which make their choices to connect to the existing topology based on the same criterion as before. In this way, the added nodes will break the balance property of the topology because some paths are extended too much. In this paper, to balance the topology structure, the sink node should update its location to a new place and accordingly re-generates the topology structure. In order to achieve the new topology more efficiently, we get the sink node location by using the local topological feature and the locations of some key nodes in the existing scenario.

Here we give some definitions in this paper. The communication range of each sensor node is denoted as \( r \), and the length of the path \( i \) is denoted as \( L_i \), which means that the total number of hops on path \( i \) to reach the sink node is \( L_i \). Our target is to find a sink node location with the minimal value of function \( F(L_{max}, L_{var}) \), which is defined as

\[
F(L_{max}, L_{var}) = L_{max} + \lambda L_{var},
\]

where \( L_{max} \) is the length of the longest path, and \( \lambda \) is the weight coefficient. The variance \( L_{var} \) measures the difference of path lengths in the whole network. Here, assume that the notation \( T \) presents the collection of all the paths in the network.

\[
T = \{ L_1, L_2, \ldots, L_n \},
\]

where \( L_i \) in \( T \) denotes the length of the path which starts from the leaf node \( i \) and arrives at the sink node. Thus the variance of the path lengths of the topology structure can be expressed by

\[
L_{var} = \frac{\sum_{L_m \in T} (L_m - L_{ave})^2}{n},
\]

where \( n \) is the number of paths and \( L_{ave} \) is the average length of all the paths in the whole network. In order to achieve the balance property of the topology structure, we then define the objective function to minimize the variance of the path lengths.

![FIGURE 1. The topology structure of wireless sensor network.](image-url)
After obtaining the new location of the sink node, another problem in our paper is how to design an efficient pathfinding strategy for the sink node. Considering that the sink node has a simple mobility and knows all the information of the sensor nodes. After the sink node calculates the new location, it needs to obtain an action list (such as, turn left, turn right, forward, backward) and reaches the new location following the instructions in the list. Also, the accomplishment of above entire process is required to be autonomous and unattended in the sink node. Considering the timeliness and feasibility, only an efficient and simple pathfinding strategy is adopted here.

III. UPDATE STRATEGY OF SINK NODE LOCATION

In this section, we put forward an update strategy of the sink node placement in the scaled-up scenario.

Assume that a small number of nodes are added in a local area of the scenario and some prior paths are extended by these new nodes, thus the current topology is generated by expanding the previous topology with some new links, which are produced by the newly added nodes with the same metric as before. A simple example is shown in Fig. 2. The newly added links are in the lower right area of the topology in Fig. 2.

![Figure 2](image1)

FIGURE 2. Illustration of an expanded network.

Algorithm 1: Update Strategy of Sink Node Location

1. Initialize:
   - $L_{max_p}$: the length of the longest path in the previous scenario;
   - $L_{max_c}$: the length of the longest path in the expanded scenario;
2. if $|L_{max_c} - L_{max_p}| >= \delta$ then
   3. Partitioning the scenario.
   4. Searching the direction of the updated placement of the sink node.
   5. Searching the activity area of the sink node.
   6. Finding the optimal location and rebuild the topology.
   7. end if

in the expanded scenario increases beyond a certain threshold $\delta$, we consider that the balance of the current topology structure is broken. Since the location informations of all the nodes in the scenario are known, we can test all possible locations for the sink node and then obtain the corresponding topology structures. Then, the location giving the best performance of the topology structure will be assigned to the sink node. Obviously, the method is feasible but not efficient in practice.

So, we consider another method to update the location of the sink node in combination with the quantity and the directions of the changes in paths length. The former, which is given in Step 5 of the algorithm, would be detailed in Section III-C and the latter, which is given in Step 4 of the algorithm, would be detailed in Section III-B. In addition, the step 3 and step 6 respectively corresponding to Section III-A and Section III-B are designed to quantify the performance of the new location of the sink node. The corresponding details will be given next.

A. NETWORK PARTITION

To better quantify the the update process in the scenario, the sink node can first divide the expanded scenario into equal-sized grids [17] as small as possible by utilizing the collected network information (such as the topology structure and the positions of sensor nodes). The corresponding graphical description is shown as follows.

![Figure 3](image2)

FIGURE 3. Illustration of a scenario which is partitioned into equal-size grids.
In the partitioned scenario, each square grid contains at most one sensor node and the sink node cannot update its location to the grid in which there already exists a sensor node. Furthermore, the location of each sensor node is represented by the central position of the occupied grid. By gridding the scenario, the sink node can get the relative position of sensor nodes in the network and design the location update scheme based on these information.

**B. MOVING DIRECTION OF LOCATION UPDATE**

In this subsection, we consider how to select the moving direction of the sink node so that the longer paths in the expanded network can be shortened.

Assume that the sink node moves in a certain direction, the nodes in the moving direction will be more closer to the sink node, resulting in a shorter length of the corresponding path. So, moving the sink node toward the leaf nodes which are on the longer paths may shorten the path length. The graphical description of the impact of the sink node movement on the path length is given as follows.

![Figure 4. The influence of the sink node movement.](image)

As shown in Fig. 4, if the sink node moves to the new location which is in the direction of the longer path, the two paths, with lengths 3 and 7 respectively, become more balanced in length.

We denote the length of the longest path in the previous scenario as \( L_{\text{max}} \), and the path started from the leaf node \( i \) to the sink node is denoted as path \( i \) with length \( L_i \). Suppose that the length of the current longest path in the scaled-up scenario denotes as \( L_{\text{max}} \). Here, we define the balanced path length \( L_b \) as follows.

\[
L_b = \left[ \frac{L_{\text{max}} + L_{\text{max}}}{2} \right].
\]

Then, we give the definition of the direction of any path \( i \) as follows

\[
\vec{D}_i = \frac{(x_i - x_{\text{sink}}, y_i - y_{\text{sink}})}{\| (x_i - x_{\text{sink}}, y_i - y_{\text{sink}}) \|^2},
\]

where \( x_i \) and \( y_i \) are the horizontal coordinate and the vertical coordinate of the sensor node \( i \), and \( x_{\text{sink}} \) and \( y_{\text{sink}} \) are the horizontal coordinate and the vertical coordinate of the sink node.

Therefore, for the current scenario, if the sink node moves toward the paths whose length are no less than \( L_b \), the lengths of these longer paths will be shorter, which will be more helpful to the balance of the topology. Thus, the direction of the sink node movement can be further given by

\[
D_{\text{sink}} = \sum_{i=1}^{n} \vec{D}_i \times f(L_i),
\]

in which \( n \) is the number of the longer paths in the expanded scenario and \( f(L_i) \) is a function which measures the impact of the path length on the direction choice.

Apparently, different path lengths have different effects on the movement direction of the sink node. Assuming a path \( j \) is much longer than other paths, to balance the topology, the final movement direction will definitely be more biased towards the path \( j \). So, the longer the path length, the greater the influence on the moving direction of the sink node. Therefore, the definition of function \( f(L_i) \) is given as follow

\[
f(L_i) = (L_i - L_b + 1)^\alpha,
\]

where \( \alpha \) is a positive constant to measure the strength of the impact of each path.

**C. FEASIBLE RANGE OF SINK NODE MOVEMENT**

As shown in Fig. 4, as the sink node moves, the previous longer path will become shorter, and the previous shorter path will become longer accordingly. Therefore, control the moving distance of the sink node to avoid the previous shorter path being lengthened too much is the main consideration in this subsection.

For each path, the movement of the sink node will directly impact the connectivity of the child-node. Assume that if the sink node moves out of the communication range, then the corresponding child-node has to re-generate the link connecting to the sink node through a relay node, which results in lengthening some original paths. Here, to balance the topology structure, we limit the moving scope of the sink node by using the information of the shorter paths whose lengths are less than \( L_b \) in the network.

Furthermore, for the paths which transmit the data through the child-node \( i \), the one which will be most affected by the movement of the sink node is the longest path under the child-node \( i \). Therefore, as the sink node moves, we only need to consider the impact on the longest path under each child-node.

Here, The symbol \( L_i \) denotes the length of the transmission path which starts from the leaf node \( i \) to the sink node. The symbol \( L_{ci} \) denotes the length of the longest transmission path through the child-node \( i \) of the sink node. If \( L_{ci} \) is less than \( L_b \), it means that this path have redundant length with \( (L_b - L_i) \) hops, that is, even the sink node move \( (L_b - L_i) \) hops further, the longest path length through child-node \( i \) will be still no more than \( L_b \), which guarantees the balance property as the sink node moves. Thus, we define the activity area of the child-node \( i \), called here \( S_{ci} \), as a circular area centered on the location of child-node \( i \) of fixed radius.
$L_b - L_i + 1 \right) \text{dis}_{\text{hop}}$, among them, \( \text{dis}_{\text{hop}} \) is the distance of a single hop. The formulation is given by

$$S_{c_i} = \{(x, y) | \|(x - x_i, y - y_i)\|^2 < (L_b - L_i + 1) \text{dis}_{\text{hop}} \},$$

where \( x_i \) and \( y_i \) are the coordinates of the child-node \( i \). Obviously, the minimal value of the radius of activity area \( S_{c_i} \) is \( 2 \text{dis}_{\text{hop}} \) and the corresponding path length is \( (L_b - 1) \). To explain the idea clearly, we give an example of the scenario as follows.

![FIGURE 5. Illustration of the activity area of a child-node.](image)

For example, assume that the balanced path length in the scenario is \( L_b = 7 \). As Fig. 5 shows, the length of the path through child-node \( i \) is \( L_{c_i} = 6 \). Therefore, the activity area of the child-node \( i \) is a circular area centered on child-node \( i \) with fixed radius \((7 - 6 + 1) \text{dis}_{\text{hop}}\). It means that even if the sink node moves to the farthest position (i.e., the hollow square), the distance from node \( i \) to it only increases one more hop, which guarantees the path length through child-node \( i \) is no more than \( L_b \) as the sink node moves.

Therefore, the activity area of the sink node movement can be treated as the intersection of the activity areas of the child-nodes which have path length less than \( L_b \). The exact definition can be given by

$$S_s = S_{c_1} \cap S_{c_2} \cap S_{c_3} \cap \cdots \cap S_{c_n},$$

where \( n \) is the number of child-nodes which satisfies the length condition, and \( S_{c_i} \) is the activity area of child-node \( i \).

For instance, in the scenario of Fig. 2 the child-nodes (no.1, no.3, no.5, no.6, no.7, and no.8) are located on the paths whose length are less than \( L_b = 7 \). Among them, for the child-node 6 with path length \( L_{c_6} = 6 \), its activity area is centered on node 6 with fixed radius \( 2 \text{dis}_{\text{hop}} \). The graphical description of the activity areas in Fig. 2 is shown in Fig. 6.

![FIGURE 6. Intersection of the activity areas in the example scenario.](image)

In Fig. 6 the shaded region \( S_s \) is the intersection of the activity areas of all conditioned child-nodes. So, we set it as the obtained activity area of the sink node movement. In region \( S_s \), regardless of the sink node moves to anywhere, the length of the extended paths through the original child-nodes will not exceed \( L_b \). Thus, by constraining the scope of the activity of the sink node, the lengths of the previous shorter paths will not increase significantly or even beyond the longest path.

In addition, the intersection of the activity areas is always existed and the relevant proving process is given as follows.

**Theorem 1.** The activity areas of the child-nodes always have an intersection area, that is, the region \( S_s \) always exits.

Proof. Assume that the activity area \( S_{c_i} \) of the child-node \( i \) has no intersection with another child-node \( j \) with activity area \( S_{c_j} \). As mentioned in Eq. (8), the minimal radius of the activity area \( S_{c_i} \) is 2 hops. In other words, the distance from the child-node \( j \) to the child-node \( i \) is greater than 2 hops, which contradicts the assumption that the nodes \( i \) and \( j \) are the child-nodes of the sink node, so it is proved.

**Remark 1.** To guarantee the connectivity of the network, we consider a simple case here, in which a new sensor node will be deployed to the prior location of the sink node as the sink node moves to the new location.

**D. OPTIMAL LOCATION OF SINK NODE**

Jointly consider the obtained moving direction in Section III-B and activity area \( S_s \) in Section III-C, we will get the feasible region of the sink node movement, which can reduce the length of the longer paths and prevent the shorter paths from being lengthen too much. Thus, to find the optimal location of the sink node, we should solve the following problem,

$$\arg \min_i F(L_{\text{max}}, L_{\text{var}}),$$

s.t. \( i \in S_s \cap \{(x_{\text{sink}}, y_{\text{sink}}) + k \overrightarrow{D_{\text{sink}}} \},$$

where the function \( F() \) is defined in Eq. (1) and \( k \) is an positive constant value, and \( i \) is the coordinates of the grid which is located along the moving direction \( k \overrightarrow{D_{\text{sink}}} \) within the activity area \( S_s \). The conventional method to solve above
problem is traversing all the possible value of \(i\) in the feasible region and generating the update topology structure with each value of \(i\), then we set the value of \(i\) with the minimal function value as the optimal location of the sink node. Obviously, the above method brings huge computational complexity, which is not applicable in practical scenario.

To meet the demand of low computational complexity in WSN application, it is prefer to find an easy-gained solution that is approximate to the optimal solution of the original problem. Therefore, we use a straightforward method \(^{(1)}\) to solve this problem, which provides an alternative way to obtain the trade-off between performance and complexity.

Due to the fact that the scenario is partitioned into multiple grids with equal size, we can approximately solve the above problem by searching a grid whose central point has a minimal value of \(F(L_{max}, L_{var})\) in the scenario. So, the approximate problem is given by

\[
\arg \min_i F(L_{max}, L_{var}), \quad \text{s.t. } i \in G_d,
\]

where \(G_d\) is the collection of the center coordinates of the grids, which are picked from the activity area \(S_a\) and located on the moving direction of the sink node (i.e., \(D_{sink}\)) at the same time.

The objective function of the original problem Eq. (10) is not a continuous function, and each function value can only be obtained by counting the features of the generated topology structure based on the formula Eq. (1).

Assume that the scenario has been partitioned into equal-size grids and the number of grids on the direction of the sink node movement \(D_{sink}\) is \(n\), and the number of sensor nodes in the network is \(N\). We generate the topology structure with BMST algorithm, which costs the computational complexity \(O(N^2)\). Therefore, to get the grid with the optimal performance, the computational complexity is \(O(n \times N^2)\). As proven in \(^{(17)}\), if the number of grids, \(n\), approaches infinite, the result of the approximate problem (9) approaches the optimal solution. Meanwhile, since the complexity of our scheme is \(O(n \times N^2)\), the larger the value of \(n\), the more complexity to solve the problem. So, the proposed approximate problem Eq. (11) provides an alternative way to obtain the trade-off between performance and complexity.

**IV. RL-BASED PATHFINDING STRATEGY**

As mentioned in previous section, an approximate optimal location of the sink node is obtained in the expanded scenario. By partitioning the network, the sink node gets the relative coordinates of the final location and all sensor nodes in the scenario. However, consider that the distributed sensor nodes in the scenario will act as the obstacles which hinder the movement of the sink node. To address this problem, we put forward a reinforcement learning (RL) based pathfinding strategy for the sink node to reach the optimal location. To overcome the shortcomings of the traditional algorithms, we propose a window-based directional Sarsa(\(\lambda\)) algorithm (WDS), which is able to obtain a more efficient performance in the pathfinding scenario.

**A. SCENARIO MODEL**

As designed in Section. III-A, the scenario is partitioned into equal-sized grids and each sensor node occupies a grid, and the sink node knows all the center coordinates of the grids and the locations of the sensor nodes. However, in the proposed scenario, the sink node cannot cross the grid in which there already exists a sensor node. Therefore, using the proposed WDS algorithm, the sink node gets an action list based on the above information, and reaches the target location by performing the actions of the list in sequence. Notice that the sink node here dose not need to get the feedback from the environment through actual moving, the whole learning process can be carried out in its cpu and finally the calculated action list will be output. An example of the scenario is shown as follows.

\[
\text{FIGURE 7. The model of the pathfinding scenario.}
\]

In Fig. 7 the sink node partitions the scenario into 20 \(\times\) 16 grids, and the red point is the calculated final location of the sink node. So, our target is letting the sink node find a feasible path from the black square to the five-pointed star, which avoids to collide the sensor nodes.

**B. DESIGN OF WDS ALGORITHM**

Here we introduce the general idea of the proposed WDS algorithm, which consists of three basic components, that is, state, action, and reward. The state, denoted as \(s_i\), means that the sink node is located in the \(i\)-th grid (with coordinate \((x_i, y_i)\)) of the network. In WDS, the scope of the search action is limited by the window size \(w\), which will be detailed in next subsection. For each state, the sink node has four possible actions, which correspond to different moving directions (i.e., up, down, left, right). So, the activities of the sink node in the scenario can be modeled as a state-action table \(T(w) \in \mathbb{R}^{N_s \times N_a}\) with window size \(w\), in which \(N_s\) is the number of possible states in the application scenario and \(N_a\) is the possible actions at each state. Therefore, in our proposed scenario, \(N_s\) is equal to the total number of grids and \(N_a\) is equal to 4 corresponding to the four moving direction.

In \(T(w)\), the value of each state-action pair, namely \(Q(s_i, a_{ij})\), is used to measure the long-run cumulative reward in the pathfinding process, which is set to 0 initially.
In pathfinding process, if the agent selects the action \( a_{i,j} \) for transferring its current state \( s_i \) to the next state \( s_j \), it will obtain the immediate reward \( r(s_j) \) which is pre-assigned in state \( s_j \). Then, the immediate reward \( r(s_j) \) will be added into the accumulated reward \( Q(s_i, a_{i,j}) \) of state \( s_i \). Then, the value of reward \( Q(s_i, a_{i,j}) \) will increase. The formulation of this process is given as follows.

\[
Q(s_i, a_{i,j}) = Q(s_i, a_{i,j}) - \alpha Q(s_i, a_{i,j}) + \alpha (E_{i,j} \times (r(s_j) + \gamma \max Q(s_j, a_{j,k}))), (12)
\]

where the \( \alpha \) is the learning rate and \( \gamma \) is the decay rate. \( s_i \) is the next state obtained by selecting the action \( a_{i,j} \) at current state \( s_i \) and \( a_{j,k} \) is selected action at state \( s_j \). \( E \) is the trace table which record the paths taken. The immediate rewards \( r(s_j) \) in this work are briefly divided into two categories in the initialization phase, that is, the reward of the state which denotes the final location of the sink node is set to 1 and the rewards of other states are set to 0. From the perspective of the path-finding task, this work is to find a feasible path without finding an optimal path. So, only the reward of the final location state requires a predetermined determined non-zero value to feedback the successful task. In addition, the value of immediate rewards of all states are fixed and pre-defined based on the current topology information of the expanded scenario. As the features of the topology structure are broken beyond a certain threshold \( \delta \) as described in Algorithm,\(^\dagger\) the immediate rewards of all the states will be reset to 0 and only the reward of the new obtained final location will be set to 1 as before. The referred parameters are presented in Table.\(^\dagger\)

<table>
<thead>
<tr>
<th>Notations</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M, N )</td>
<td>The scenario consists of ( M \times N ) grids.</td>
</tr>
<tr>
<td>( s_i )</td>
<td>The ( i )-th state of the sink node.</td>
</tr>
<tr>
<td>( a_{i,j} )</td>
<td>The ( j )-th action of the ( i )-th state.</td>
</tr>
<tr>
<td>( r(s_i) )</td>
<td>The immediate reward that the sink node obtains due to the state ( s_i ).</td>
</tr>
<tr>
<td>( Q(s_i, a_{i,j}) )</td>
<td>The accumulated reward that the sink node obtains by executing the ( j )-th action at state ( i ).</td>
</tr>
<tr>
<td>( \gamma )</td>
<td>The decay rate.</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>The learning rate.</td>
</tr>
<tr>
<td>( E )</td>
<td>The trace table of the state transformation.</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>The fading factor of the impact of trace table.</td>
</tr>
<tr>
<td>( w )</td>
<td>The window size.</td>
</tr>
<tr>
<td>( T(w) )</td>
<td>The state-action table generated based on the window size ( w ), and its entry ( T_{i,j} ) is equal to ( Q(s_i, a_{i,j}) ).</td>
</tr>
</tbody>
</table>

By multiple rounds of training, the value of each state-action pair \( Q(s_i, a_{i,j}) \) will eventually coverage. Finally, the agent can reaches to the destination by choosing each state-action pair with the maximum value in each step. The pseudo-code of the WDS algorithm is given in Algorithm.\(^\dagger\)

**Algorithm 2 Window-Based Directional Sarsa(\( \lambda \)) Algorithm**

1. **Initialize:**
   - \( s_i \): the initialization state; \( w=0 \); Set all initial state-action pairs to 0; Set the reward of the final state to 1, otherwise 0.
2. Generate the corresponding state-action table \( T(w) \) with window size \( w \).
3. **Repeat** (for each episode):
   - Initialize the state of the sink node as \( s_i \).
4. **Repeat** (for each step of episode):
   - Select an action at each state.
   - Update the reward \( Q \) and trace table \( E \).
   - Update the state of the sink node to the next state.
   - If the sink node falls into a trap, **then**:
     - Go back and update the step.
   - If the sink node cannot find a feasible path and the number of tried pathfinding episodes is greater than \( I_{\text{max}} \), **then**:
     - Go back to step 2 and let \( w = w + 1 \).
5. **Terminal:** Output \( T(w) \).

In Algorithm,\(^\dagger\) Step 2, Step 6 and Step 9 are our designed novel steps in WDS, which improve the performance of the pathfinding task in our proposed scenario, and are elaborated as follows.

1) Window-based searching method

The step 2 of the Algorithm,\(^\dagger\) is to limit the search scope of the states in the scenario, called the window-based searching method. In the scaled-up WSN scenario which is partitioned to \( M \times N \) grids, the corresponding number of the states in the network is about \( M \times N \). For the classical Sarsa(\( \lambda \)) algorithm, the sink node has to cost huge computational complexity and time complexity to obtain the feasible path from the initial state to the final state by almost traversing all the feasible state-action pairs in the state-action table, which is unpractical in an actual WSN application.

By contrast, our proposed WDS algorithm contains the prior knowledge, namely the final location of the sink node. Through utilizing the coordinates of the final location and the initial location of the sink node, the state-action table with a smaller size will be obtained, which is used to find the feasible path with low computational overhead.

Here, assume that the current window size is \( w \), thus the corresponding state-action table \( T(w) \) is give by

\[
T(w) = \{(s_i, a_{i,1}, a_{i,2}, a_{i,3}, a_{i,4}) | i \in S(w)\}, (13)
\]

where the \( S(w) \) is the set of states included the window.
range, which is further given by
\[
S(w) = \{(x, y)| x \in \max(R_{x_{min}}, \min(x_{sink}, x_{des}) - w), \ldots, \min(\max(x_{sink}, x_{des}) + w, R_{x_{max}})\},
\]
\[
y \in \{\max(R_{y_{min}}, \min(y_{sink}, y_{des}) - w), \ldots, \min(\max(y_{sink}, y_{des}) + w, R_{y_{max}})\},
\]
where \((x_{sink}, y_{sink})\) and \((x_{des}, y_{des})\) are the coordinates of the initial location and the final location of the sink node, respectively, and \(w\) is the window size. \(R_{x_{min}}\) and \(R_{y_{max}}\) is the coordinates of the left edge and the upper edge of the monitoring area, individually.

Thus, the searching range of the states in WDS algorithm decreases from \(M \times N\) to \((2w + |x_{sink} - x_{des}| + 1) \times (2w + |y_{sink} - y_{des}| + 1)\) approximately. With the current window size \(w\), if the sink node cannot reach the final location under a certain number of pathfinding episodes. It considers that there is no valid path which can be obtained from the current state-action table \(T(w)\). At this time, the window size \(w\) will be updated to \(w+1\) and the pathfinding task is re-performed with a larger state-action table \(T(w+1)\). Obviously, by reducing the number of states that requires to search, the sink node can find a feasible path to the final location more efficiently.

2) Action Selection Mechanism

The step 6 of the Algorithm. 2 is a novel action selection mechanism, which is helpful to efficiently search the feasible path from the current state-action table \(T(w)\).

As mentioned in previous sections, the selection metric is the key issue of the pathfinding task. Generally, as designed in the classical Sarsa(\(\lambda\)) algorithm, the sink node will select the state-action pair which owns the maximum reward value \(Q(s_i, a_{i,j})\) at current state \(s_i\) with high probability \(\varepsilon\), and select a random action with the probability of \(1 - \varepsilon\). So, in the initialization stage, the sink node has to randomly select an action at any states because all the corresponding state-action values are all set to 0 initially, which poses a potential challenge in practical applications. Therefore, to eliminate the random interference in the first pathfinding task, the prior knowledge is taken into account here.

Assume that the moving direction of the sink node is obtained by Eq. [6]. Let the actions (i.e., turn left, turn right, forward, backward) at state \(s_i\) correspond to the directions (i.e., \(\vec{y}\), \(-\vec{y}\), \(-\vec{x}\), and \(\vec{x}\)), respectively. \(\vec{x}\) and \(\vec{y}\) are the unit vectors in the positive horizontal direction and the positive vertical direction. So, the moving direction of the sink node can be decomposed into
\[
D_{sink} = m \vec{x} + n \vec{y},
\]
where \(m\) and \(n\) are the mapping of \(D_{sink}\) on the positive horizontal direction and positive vertical direction. The graphical description of compositing the \(D_{sink}\) is shown as follows.

For example, if the mappings of the direction \(D_{sink}\) on \(\vec{x}\) and \(\vec{y}\) are \(m > 0\) and \(n > 0\), respectively, it means that the preferred actions are up and right. In this case, if right-action \(a_{i,j}\) and the up-action \(a_{i,k}\) both have the maximum value of state-action pair at state \(s_i\), the sink node prefers to choose the action from these two actions instead of all from feasible actions, the corresponding possibility of choosing the action \(a_{i,j}\) and \(a_{i,k}\) can be further given by \(\frac{m}{m+n}\) and \(\frac{n}{m+n}\), respectively. If either action \(a_{i,j}\) or the action \(a_{i,k}\) has the maximum value of state-action pair at state \(s_i\), it will be specified as the fixed next action. Otherwise the next action is selected as designed in classical Sarsa (\(\lambda\)). The pseudo-code of the improved action selection mechanism is given as follows.

Algorithm 3 Improved Action Selection Mechanism.

1: \(D_{sink} = m \vec{x} + n \vec{y}\)
2: Generate the preference actions based on \(m\) and \(n\), denoted as \(a_{i,j}\) and \(a_{i,k}\) here.
3: if either \(a_{i,j}\) or \(a_{i,k}\) is equal to \(\max Q(s_i, : )\) then
4: Select the action \(j\) with probability \(\frac{|m|}{|m|+|n|}\) or \(k\) with probability \(\frac{|n|}{|m|+|n|}\).
5: end if

In this way, the sink node prefers to select a better action at current state according to the moving direction of the sink node.

3) Risk Avoidance Mechanism

The step 9 of the Algorithm. 2 is to put forward a risk avoidance mechanism during pathfinding process.

As assumed, the actual WSN application scenario is complicated, thus the movement of the sink node may be obstructed by the deployed sensor nodes in the network. In our proposed scenario, it considers that the grids occupied by several nodes in the local area may happen to connect to each other, and thus form a trap-like area which is shown in Fig. 9.
In Fig. 9 the red five-pointed star in the lower right corner is the final location of the sink node and the black triangle in the upper left corner is the initial location of the sink node. The blue squares are the deployed sensor nodes, which are treated as obstacles during the sink node movement.

If we directly adopt the action selection mechanism as described in Algorithm 3 during each action selection round, the sink will move down with probability $\frac{8}{15}$ and move right with probability $\frac{7}{15}$. As shown in the figure above, the sink node may move into the area which is called a "trap". In the trap area, if the sink node gets stuck in a certain location with coordinates $(x_{trap}, y_{trap})$, we consider that this pathfinding episode fails. Unfortunately, due to the fact that the sink node only select the two actions with the fixed probabilities, it may be stuck into the location denoted by the dotted square. As the predefined steps in a path-finding episode exhaust, the program of the path-finding scheme will be reset. However, it still doesn’t work on preventing the sink node from falling into the trap in the next path-finding episode because the action selection probability are unchanged. So, once the sink node falls into the trap, we will jointly consider the trap area, if the sink node gets stuck in a certain location with coordinates $(x_{trap}, y_{trap})$, we will jointly consider the trap area, if the sink node gets stuck in a certain location with coordinates $(x_{trap}, y_{trap})$. We consider that this pathfinding episode fails.

As described in Eq. (15), the moving direction of the sink node can be decomposed on two base unit directions, $\vec{x}$ and $\vec{y}$, and the corresponding mappings are $m$ and $n$, respectively. Therefore, based on the obtained mappings $m$ and $n$, we set the probability of the corresponding actions selection as $\frac{|m|}{|m|+|n|}$ and $\frac{|n|}{|m|+|n|}$ in the initialization stage. Here, assume that the sink node falls into the trap with coordination $(x_{trap}, y_{trap})$ in the scenario, we can update the possibility of each action selection accordingly to avoid the accident happened repeatedly, the pseudo-code are given as follows.

**Algorithm 4 Risk Avoidance Mechanism.**

1. if the pathfinding episode fails and the sink node is finally stuck in a certain location for more than $q$ steps then
2. The coordinate of the trap location is $(x_{trap}, y_{trap})$.
3. if $\frac{|n|}{|m|+|n|} < \frac{y_{trap}}{x_{trap}}$ then
4. Update the corresponding selection possibility from $\frac{|m|}{|m|+|n|}$ to $\frac{|m|+|n|}{|m|+|n|}$ and $\frac{|n|}{|m|+|n|}$.
5. else
6. Update the corresponding selection probability from $\frac{|m|}{|m|+|n|}$ to $\frac{|n|+|m|}{|m|+|n|}$ and $\frac{|n|}{|m|+|n|}$.
7. end if
8. end if

**V. COMPUTER SIMULATION**

In this section, we put forward some simulation tests to validate our proposed scheme in terms of the property of the newly generated topology structure and the performance of the proposed pathfinding algorithm, which are accomplished in PYTHON language.

**A. LOCATION UPDATE STRATEGY OF THE SINK NODE**

In our computer simulations, we consider an original WSN scenario with 60 sensor nodes distributed randomly and uniformly. The original scenario is a rectangle area with size $18 \times 13$, and the topology is generated based on BMST algorithm [13].

At the network scaling-up phase, 26 new sensor nodes are deployed around the original network. Then, the scale of the scenario is expanded to $20 \times 16$, and the original topology with the longest path $L_{max} = 5$, is scaled to an unbalanced structure with longest path $L_{max} = 8$. The graphical description of the scaled-up scenario is given in Fig. 10.

To balance the topology structure, a new location of the sink node is obtained in this scenario. In order to exhibit the calculation process clearly, we firstly partition the scenario with $20 \times 16$ grids as described in Sec. III-A thus the size of each grid is $1 \times 1$. In addition, we set the bottom left...
corner of the scenario as the origin coordinates (0, 0), and the coordinates of each grid are represented by the coordinate of its central point.

Initially, the maximum communication radius of each sensor node is set to 2.4, which can also be measured as the distance of a single hop. The initial location of the sink node is given by (8, 10) and the balanced path length $L_b$ is given by $\left\lfloor \frac{9+2}{2} \right\rfloor$. The coordinates of the leaf nodes which are on the paths with length no less than $L_b$ are (19, 2), (19, 3), and the corresponding path length is 8 and 8, respectively. Meanwhile, the strength factor $\alpha$ in Eq. (6) is set to 2 here. Therefore, the moving direction $D_{sink}$ of the sink node can be obtained as $D_{sink} = (8.515, -8.921)$. Then, we generate the activity area $S_v$ of the sink node movement in our scenario. The graphical description of the results of our proposed location update strategy is given in Fig. 11.

In Fig. 11, the line with black arrow is the direction $D_{sink}$ and the green squares are the grids which are located inside the activity area $S_v$ and on the direction $D_{sink}$ simultaneously. As a result, there are four grids meet the above conditions, which have the center coordinates (9, 9), (10, 8), (11, 7), and (12, 6), respectively. For each possible optimal location obtained above, the sink node moves and generates the topology structure based on BMST algorithm, which is shown in Fig. 12. Also, the longest path length and the variance of the path length in the four newly generated topology structures are further given in Table. 2.

Table 2: Comparison of the longest path length and the variance of the path length on different sink node locations.

<table>
<thead>
<tr>
<th>Location</th>
<th>Maximum length</th>
<th>Variance of length</th>
</tr>
</thead>
<tbody>
<tr>
<td>(9, 9)</td>
<td>7</td>
<td>1.964</td>
</tr>
<tr>
<td>(10, 8)</td>
<td>6</td>
<td>1.8</td>
</tr>
<tr>
<td>(11, 7)</td>
<td>7</td>
<td>1.712</td>
</tr>
<tr>
<td>(12, 6)</td>
<td>7</td>
<td>2.45</td>
</tr>
</tbody>
</table>

In Table. 2, the variance is significantly larger for a sink node located at (10, 8) and (11, 7) compared to the other locations, which indicates that the sink node at these locations will move more for a longer time.

The optimal location of the sink node in this scaled-up scenario is the grid with coordinate (10, 8).

B. PERFORMANCE OF WDS ALGORITHM

In this subsection, we perform the RL-based pathfinding algorithm to teach the sink node how to get the optimal location obtained above. We propose two different scenarios to compare the performance of the proposed WDS algorithm with the classical Sarsa (\(\lambda\)) algorithm, and the improved Q-Learning (IQL) algorithm [36].

1) Scenario 1

The first simulation test is carried out on the scenario which is generated in above subsection. We have executed the location update strategy and obtained the optimal location of the sink node as coordinates (10, 8), thus the pathfinding task here is to find a feasible path from the initial point (8, 10) to the destination (10, 8). The graphical description of the scenario is given in Fig. 13.

As shown in Fig. 13, the black line with arrow is the shortest routing path with 4 hops from the previous location (8, 10) to the final location (10, 8). Here, we initialize the windows size $w$ of the WDS algorithm as 0. The rest parameters in the simulation test are given in Table. 3.

Table 3: Simulation parameters in scenario 1.

<table>
<thead>
<tr>
<th>Notation</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>0.01</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>0.9</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>0.9</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 4: Simulation results in scenario 1.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Average acquisition rate of the optimal path</th>
<th>Average searching steps in first episode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sarsa($\lambda$)</td>
<td>76%</td>
<td>157</td>
</tr>
<tr>
<td>WDS</td>
<td>100%</td>
<td>9.7</td>
</tr>
<tr>
<td>IQL</td>
<td>100%</td>
<td>4.9</td>
</tr>
</tbody>
</table>

We test the performance of WDS algorithm, classical Sarsa($\lambda$) and IQL algorithm independently. Considering the requirement of the low computational complexity in practical applications, for each algorithm, we execute 120 times of the pathfinding task independently. Each time, we test 100 episodes and the number of tried steps in each episode is 400. Here, the corresponding average results of the 120 times pathfinding tasks are shown in Table. 4.

In Table. 4 the average acquisition rate of the optimal path $\eta$ indicates the proportion of the optimal route path that can be obtained during 120 times pathfinding tasks. The average searching steps in first episode refers to the number of required steps to reach the final location (the red point with coordinate (10, 8)) for the first time during 120 times pathfinding tasks. From the results, it can be seen that WDS
algorithm and IQL algorithm have better guarantee because of the adopted prior knowledge. In contrast, the search scope of Sarsa(\(\lambda\)) algorithm in pathfinding task is the global range. Once a feasible path is found, it will continue to follow this path with great probability, so there is no guarantee that it will get the optimal path.

Due to the fact that the first path searching process naturally exhausts the most steps, the average searching steps in first episode is also a very important performance indicator. Also, the Sarsa(\(\lambda\)) algorithm, which has no prior knowledge, is naturally inefficient in this case. Comparing IQL algorithm with WDS algorithm, we can see that the performance of IQL algorithm takes fewer steps in searching the optimal path. However, it does not mean that the IQL algorithm is more efficient and practical because it costs more computational load during pathfinding process. In each step of IQL algorithm, it has to calculate the distances from all the feasible next states to the final location of the sink node, and then sets the one with the minimum value as the fixed next state. In contrast, the WDS algorithm and Sarsa(\(\lambda\)) algorithm only need to choose action with maximum reward during the action selection phase, which have a huge energy consumption advantage in the unattended monitoring applications.

2) Scenario 2
To further compare the actual performance of above three algorithms, we establish a more complex scenario, which has a longer distance between the initial location and the final location of the sink node, and some traps are sited on moving direction of the sink node. The graphical description is shown in Fig. 14.

As shown in Fig. 14, the red five-pointed star in the lower right corner is the final location of the sink node and the black triangle in the upper left corner is the initial location of the sink node. Furthermore, there are two traps-like obstacles (consisted of distributed sensor nodes) locate on the moving direction of the sink node, which may seriously impact the performance of the sink node pathfinding task. Considering the requirement of the low computational complexity in practical applications, for each algorithm, we execute 120 times of the pathfinding task. Each time, we test 100 episodes and the number of tried steps in each episode is 400. The rest parameters in simulation tests are set to the same as in scenario 1, which has given in Table. 3. The average results of the 120 times independent pathfinding tasks of the scenario 2 are given in Table. 5.

We first explain the performance indicator appeared in Table. 5. The pathfinding task success rate refers to the probability that a feasible path can be obtained finally in limited pathfinding episodes during 120 times of the pathfinding
FIGURE 13. Illustration of the pathfinding task in scenario 1.


TABLE 5. Simulation results in scenario 2.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Pathfinding task success rate</th>
<th>Pathfinding success rate in first episode</th>
<th>Number of episodes required for pathfinding success</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sarsa(λ)</td>
<td>100%</td>
<td>17%</td>
<td>6.46</td>
</tr>
<tr>
<td>WDS</td>
<td>100%</td>
<td>52%</td>
<td>1.67</td>
</tr>
<tr>
<td>IQL</td>
<td>0%</td>
<td>0%</td>
<td>∞</td>
</tr>
</tbody>
</table>

tasks. The pathfinding success rate in first episode refers to the probability that a feasible path can be found under the specified number of steps (400) in the first pathfinding episode during 120 times independent pathfinding tasks. The number of episodes required for pathfinding success refers to the number of pathfinding episodes that are required to find a feasible path during 120 times independent pathfinding tasks. From the results, it is obviously that the IQL algorithm can not complete the task of pathfinding in this proposed scenario because its action selection principle only considers the distance from the sink node. Then, the sink node executed the IQL algorithm is easy to be locked into the trap-like obstacles and can not get out under the limited steps range. Even if the sink node restarts a new pathfinding episode, it will still fall into the same trap because the adopted action selection mechanism keeps unchanged.

The classical Sarsa(λ) algorithm finds the optimal path based on the random search method. So, in the case of a finite number of steps, the success rate in the first search episode is very low. This also limits the application of the classical Sarsa(λ) algorithm in the actual scenario.

In contrast, the performance of WDS algorithm is obviously better than others. In the proposed scenario, the \( D_{\text{sink}} \) is given by \( 10 \overrightarrow{x} + (-10) \overrightarrow{y} \), thus the sink node has a probability of 50% to select the down action and a probability of 50% to select the right action. In the first pathfinding test episode, WDS algorithm has a probability of nearly 52% to get the feasible path from the initial location to the final location, and has a probability of nearly 48% to fall into the trap with coordinate \((7, -6)\). To avoid falling into the trap again, based on the risk avoidance mechanism proposed in Section. [IV-B3] we update the probabilities of selecting the down action and the right action to 75% and 25%, respectively. As a result, the WDS algorithm get a more guaranteed performance in the next pathfinding episode, that is, the number of episodes required for pathfinding success is 1.67.
VI. CONCLUSION

This paper proposes the location update scheme of the sink node to address the problems caused by the network scaling-up. We divide the scheme into two aspects, namely searching the optimal update location and designing an intelligent pathfinding algorithm. For the first point, by collecting the information of some key nodes and topology structure, we can get the update direction and the scope of the new location of the sink node. Then, the best performing location is selected and set as the final location of the sink node. After that, we generate a reinforcement learning algorithm (WDS), which is designed jointly considering the classical Sarsa (λ) algorithm and some prior knowledge in the application scenario. Finally, we firstly get the optimal location of the sink node and further put forward some simulation tests to exhibit the performance of WDS algorithm in two different scenario. The results show the superior performance of WDS algorithm. However, the actual application scenario of the WSN is more complicated than the assumptions in our paper. So, it poses a great challenge to execute the location update scheme in practical application, which will be tackled in our future work. Moreover, we will investigate how to safeguard the security for the considered systems, by using some physical-layer secure techniques [40–44].

REFERENCES

Xindi Wang, Qingfeng Zhou, Chunxiao Qu, Gao Chen, Junjuan Xia: Location Updating Scheme of Sink Node based on Topology Balance and Reinforcement Learning in WSN.


***