J. Parallel Distrib. Comput. 104 (2017) 99–113

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Minimizing deep sea data collection delay with autonomous underwater vehicles

Huanyang Zheng *, Ning Wang, Jie Wu

Department of Computer and Information Sciences, Temple University, Philadelphia, PA, 19122, United States

HIGHLIGHTS

- Autonomous underwater vehicles are schedules to collect the data from underwater sensor networks.
- Optimal AUV resurfacing schedules are studied.
- Algorithms that schedule AUVs cooperatively are presented.

ARTICLE INFO

Article history:
Received 6 March 2016
Received in revised form
11 November 2016
Accepted 5 January 2017
Available online 18 January 2017

Keywords:
Deep sea data collection
Delay tolerant networks
Autonomous underwater vehicles
Euler circuit
Trajectory scheduling

ABSTRACT

As a special application of delay tolerant networks (DTNs), efficient data collection in the deep sea poses some unique challenges, due to the need for timely data reporting and the delay of acoustic transmission in the ocean. Autonomous underwater vehicles (AUVs) are deployed in the deep sea to surface frequently to transmit collected data from sensors (in a 2-dimensional or 3-dimensional search space) to the surface stations. However, additional delay occurs at each resurfacing. In this paper, we want to minimize the average data reporting delay, through optimizing the number and locations of AUV resurfacing events. We also study the AUV trajectory planning using an extended Euler circuit, where the search space is a set of segments (e.g., oil pipes) in the deep sea. To further reduce the data reporting delay, several schemes, which schedules multiple AUVs cooperatively, are also explored. Finally, experiments in both the synthetic and real traces validate the efficiency and effectiveness of the proposed algorithms.

1. Introduction

While the sea is the largest habitat on earth, it remains largely unexplored. The search efforts for the missing Malaysia flight MH370 demonstrated that it is extremely difficult to conduct an efficient search process in the deep sea for data collection. In addition to the vast area of search space, data reporting in the deep sea also poses a unique challenge that does not occur in regular land communications. Although several different types of media can be used under the sea, the acoustic transmission [10,3] was first used for underwater communications. However, it is well known that the acoustic transmission suffers from a very significant signal attenuation. Based on [27], the data rate of the acoustic technique is usually limited to 10 kbps with the maximum transmission range up to 100 m. Due to the limited data rate and the limited transmission range, acoustic techniques are not applicable to big data transmissions in the deep sea. Therefore, to report data in a search effort, autonomous underwater vehicles (AUVs) deployed in the deep sea are used to surface frequently and transmit collected data to the surface station. A motivational example could be the detection of oil pipe leaks through robotic submarines in the Gulf of Mexico [12].

This paper considers a special scheduling problem aiming to minimize the average data reporting delay. AUVs are used to search and collect data in a given 2-dimensional (2-D) search space, which is parallel to the water surface with a given depth. We also extend a scenario of a 3-dimensional (3-D) search space, through reducing it to a 2-D search space. In a given search space, the data reporting should be done in a timely manner; however, additional delay occurs at each AUV resurfacing. Fig. 1 shows such a scenario of data reporting from the deep sea. We consider the search space to be a set of segments (e.g., oil pipes), which is represented as a set of weighted edges in a graph. We propose an AUV trajectory planning using an extended Euler circuit, and then, we determine the number and locations of resurfacing events on the circuit (or simply *cycle*). Specifically, we study the following problems in sequence. (i) Given the circumference of a cycle of a search space at a given depth, we determine the number and locations of AUV

* Corresponding author.
E-mail address: huanyang.zheng@temple.edu (H. Zheng).

http://dx.doi.org/10.1016/j.jpdc.2017.01.006
0743-7315/

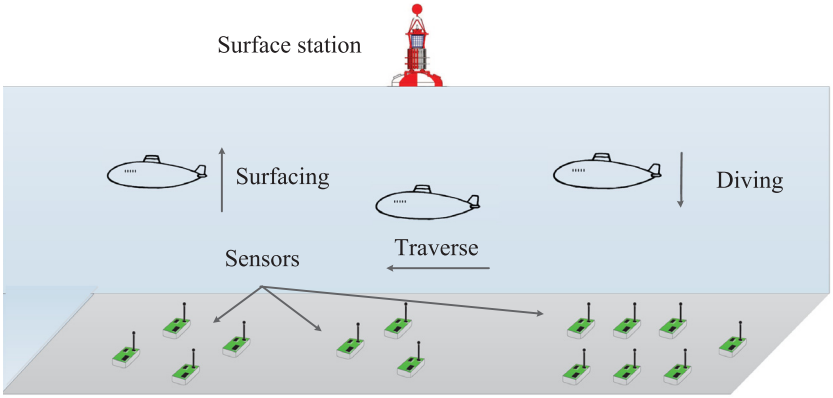


Fig. 1. Data reporting in the deep sea.

resurfacing events. (ii) We study a more general case where the search space is a collection of edges, called *sensing edges*. AUVs can collect the data from the sensing edges. We then determine the cycles that cover all sensing edges, where some edges may appear more than once. (iii) Using the geometric property, we replace some multiple-visited sensing edges with geometrically-shortest-distance links that are not sensing edges in the graph (called *non-sensing edges*), as to shorten the cycle circumference. Note that no data is collected from the non-sensing edges. We also adjust the number and locations of AUV resurfacing events for cycles with non-sensing edges. (iv) Given a search space of multiple cycles, we study a cooperative AUV trajectory planning, where the cycles are merged to further reduce the average data reporting delay.

The key difference between our approach and the classic ferry approaches [37,38] lies in the AUV resurfacing events that bring an extra delay. If the AUVs resurface frequently, then the uncollected data needs to wait longer to be reported, which leads to an increased average data reporting delay. On the other hand, if the AUVs resurface infrequently, then the collected data within the AUV needs to wait longer to be reported, which also leads to an increased average data reporting delay. This tradeoff poses some unique challenges of combining the design of AUV resurfacing events and trajectory planning in the deep sea, which have not been explored in existing works on underwater sensor networks [1,5,25,24] and corresponding protocols [2,6,7,33,18,8]. We also study AUV surfacing events in cycles with non-sensing edges, as well as the cooperative AUV trajectory planning.

2. Related work

Recently, underwater sensor networks [1,24,39] have become a very hot topic. Chandrasekhar et al. [5] surveyed different localization algorithms that are relevant to underwater sensor networks, including range-based and range-free algorithms. They also explored challenges in meeting the requirements posed by emerging applications for such networks (e.g. offshore engineering). Pompili et al. [25] studied the routing algorithms for delay-insensitive and delay-sensitive applications. An architecture for three-dimensional underwater sensor networks was considered, and a model characterizing the acoustic channel utilization efficiency was introduced. This model can adjust the optimal packet size for underwater communications, given monitored volume, density of the sensor network, and application requirements. More detailed surveys on underwater sensor networks were reported in [2,6]. Routing techniques were surveyed, including vector-based routing, sector-based routing, clustering-based routing, focused beam routing, reliable and energy balanced routing, multi-sink opportunistic routing, location-aware source routing, and so on. The monitoring problem has also been studied in underwater sensor networks. Jawhar et al. [18] proposed an efficient framework in AUV-extended sensor networks for pipeline monitoring. Linear sensor networks are used to monitor underwater pipelines. The data is collected from the sensor nodes, and then, transmitted to a surface sink using AUVs. This approach has a significantly smaller sensor transmission range than previous techniques, and can further reduce the interference between sensor nodes to mitigate hidden terminal and collision problems. Eichhorn et al. [8] designed a modular AUV system for the automated detection and analysis of water quality parameters. The AUV carrier platform is called "CWol", which is based on Fraunhofer IOSB-AST with a payload-sensitive design. The integration of the payload unit in the AUV carrier platform were studied. Compared to previous works, this paper uses underwater sensor networks to detect oil pipe leaks. We focus on AUV resurfacing decisions and the AUV trajectory planning to collect the data from underwater sensors.

Data collection has been considered as an important problem in sensor networks. Vasilescu et al. [31] built a sensor network consisting of static and mobile underwater sensor nodes. The mobile nodes can locate and hover above the static nodes for data collection, and they can perform network maintenance functions such as deployment, relocation, and recovery. Yao et al. [36] developed one data collection protocol called EDAL, which stands for Energy-efficient Delay-Aware Lifetime-balancing data collection. Their design leverages one result from the open vehicle routing problem to prove the problem hardness. Both centralized and distributed heuristics are proposed to reduce the computational overhead and improve the algorithm scalability. Liu et al. [21] proposed a novel compressive data collection scheme for wireless sensor networks, by leveraging empirical observations that sensory data possess strong spatiotemporal compressibility. Their scheme requires fewer compressed measurements, thus greatly reduces the energy consumption. It allows simple routing strategy without much computation and control overheads, which leads to strong robustness in practical applications. Incel et al. [15] studied a fundamental problem: how fast can information be collected from a wireless sensor network organized as tree? They first considered time scheduling on a single frequency channel with the aim of minimizing the number of time slots required (schedule length) to complete a convergecast. Next, they combined scheduling with transmission power control to mitigate the effects of interference, and show that while power control helps in reducing the schedule length under a single frequency, scheduling transmissions using multiple frequencies is more efficient.

In sensor networks that are on the ground, message ferries (or data mules) are also used to collect the data from different sensors [29]. For example, Zhao et al. [37] designed a set of special message ferries to carry data for sensors in the network. The main

idea behind this approach is to introduce non-randomness in the ferry movement and exploit such non-randomness to deliver data. As a result, the efficiency of the ferry route (i.e., ferry trajectory planning) is very important [16]. Sugihara and Gupta [26] focused on the trajectory scheduling problem for the data mule to achieve the smallest data delivery latency in the case of minimum energy consumption at each sensor. Since their problem is NP-hard, an approximation algorithm was presented and analyzed with respect to the approximation ratio. Tekdas et al. [28] explored synergies among mobile robots and wireless sensor networks in environmental monitoring through a system, in which robotic data mules collect measurements gathered by sensors. A proof-of-concept system was implemented to increase the system lifetime by conserving energy that the sensing nodes otherwise would use for communications. Klein et al. [19] considered the source localization using acoustic sensors dispersed over a large area, with the individual sensors located too far apart for direct connectivity. An unmanned aerial vehicle was employed for collecting sensor data, with its route adaptively adjusted based on data from sensors already visited, in order to minimize the time to localize events of interest. The key difference between this paper and the traditional message ferry (or data mule) approach lies in the AUV resurfacing process that brings an extra delay. While the routes of traditional message ferries are 2-dimensional [4,23,17,22,34], the routes of AUVs in this paper are 3-dimensional.

Traditional trajectory planning problems are usually formulated as traveling salesman problems (TSPs) or their extensions. The objective of the TSP is to find the shortest possible route that visits each node (sensor) exactly once and returns to the origin. For example, Moazzez et al. [23] used multiple message ferries to cooperatively visit sensors. Sensors are divided into several groups, while each sensor group has a message ferry to collect the data. Clearly, the trajectory of the message ferry in each sensor group is a TSP. Ma et al. [22] modeled trajectory planning problems as TSPs with additional distance and time constraints. A heuristic trajectory planning algorithm was presented for different scenario constraints. By comparison, this paper considers AUVs to go along pipes (i.e., edge traversal), which is formulated as an Eulerian cycle problem that finds the shortest possible route to visit each edge (instead of node) exactly once.

3. Framework and problem formulation

This paper studies the data collection in the deep sea with delay minimization. Our research is motivated by the detection of oil pipe leaks through robotic submarines in the Gulf of Mexico [12]. As shown in Fig. 2(a), we study a search space that is a set of oil pipes deployed in the seabed. Nodes are sources or destinations of oil pipes, which are not necessarily linear. Sensors are densely and uniformly deployed along pipes to detect the leakages. Another application scenario is the seabed settlement monitoring [13], where the sensors are deployed to monitor the seabed environmental change.

Underwater acoustic communications suffer from a significant signal attenuation. Based on [27], the data rate of the acoustic technique is usually limited to 10 kbps with the maximum transmission range up to 100 m. Therefore, AUVs are used to go along pipes to collect the data from the sensors, and then surface to report the data. The above data collection and AUV resurfacing are periodic. Our objective is to collect and report the data with a minimal average delay. If the AUVs resurface frequently, then the uncollected data needs to wait longer to be reported, which leads to an increased average data reporting delay. On the other hand, if the AUVs resurface infrequently, then the collected data within the AUV needs to wait longer to be reported, which also leads to an increased average data reporting delay. For further processing, the search space is converted to a given graph with a certain depth in the sea, as shown in Fig. 2(b). The lengths of the pipes are the edge weights in the given graph. The edges in the given graph are also called sensing edges, since AUVs need to traverse these edges to collect the data.

In Section 4, we will start with an ideal case, where the given graph is composed of only one cycle. As shown in Fig. 2(c), we would like to determine the number and locations of resurfacing events that minimize the average data delay. However, the assumption that the given graph is cyclic may not be very practical. Therefore, in Section 5, we discuss how to construct cycles from the given graph, based on the extended Eulerian cycles. An example is shown in Fig. 2(d). Each connected component in the given graph of Fig. 2(b) is converted to a cycle (i.e., cycles ABDBACA and EGHFHGE). The constructed cycles are only composed of sensing edges, where some edges may appear more than once, as the given graph is not necessarily Eulerian. At this time, we could use the results in Section 4 to schedule the AUV resurfacing events for each constructed cycle.

In Section 6, we would improve the cycle construction, through replacing some multiple-visited sensing edges with geometrically-shortest-distance links that are not sensing edges in the graph, as to shorten the circumference of the resultant cycle. These geometrically-shortest-distance links are called non-sensing edges, since no data is collected from them. An example is shown in Fig. 2(e), where we use the non-sensing edges of DC and EF to shorten the circumferences of the cycles in Fig. 2(d). Smaller circumferences of the constructed cycles can result in smaller average data reporting delays. Furthermore, in Section 7, we observe that cycles can be merged with a cooperative AUV scheduling. As shown in Fig. 2(f), the two smaller cycles in Fig. 2(e) are merged, leading to a bigger cycle of ABDFHGECA. The cycle merge can also reduce the average data reporting delay [32]. Several cycle merge criteria are discussed with respect to the cycle circumferences and the geographical distances among different cycles.

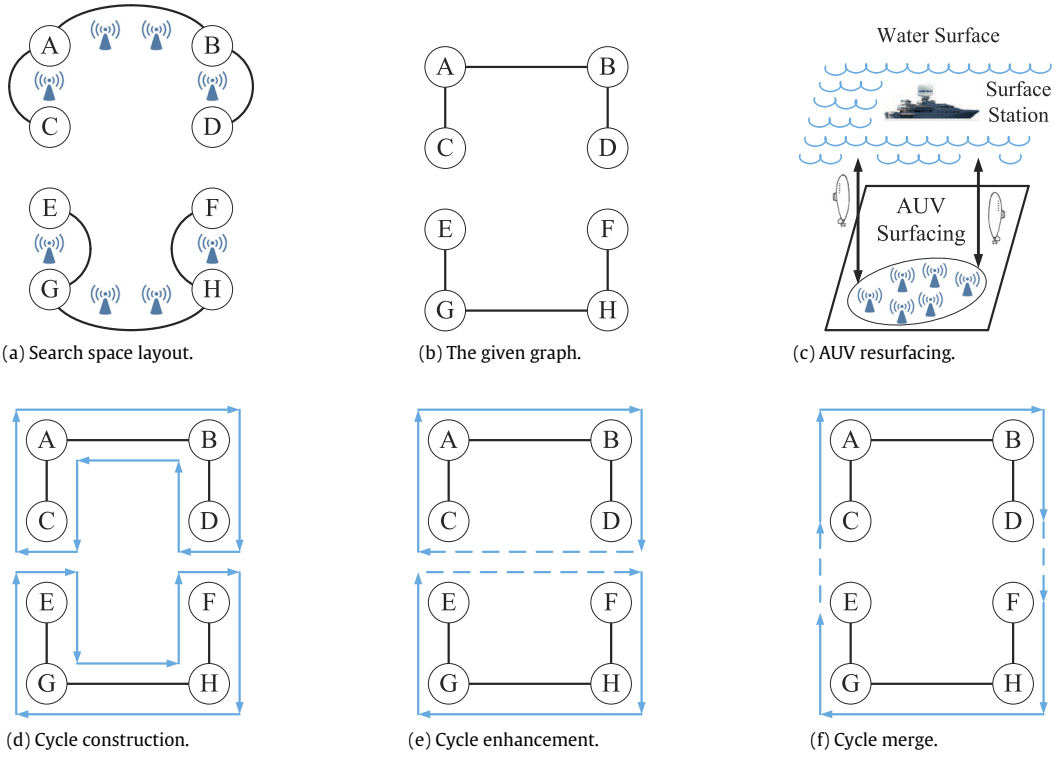
Finally, all notations are shown in Table 1.

4. Resurfacing frequency

4.1. Basic scenario

This subsection focuses on a basic scenario, where the search space is completely parallel to the water surface. We start with a cycle of search space in a given depth with several AUVs, as shown in Fig. 2(c). We determine the AUV resurfacing frequency. In the search space, sensors are uniformly distributed along the cycle, while the data has a constant generation rate. Let us consider the scenario with only one AUV, which has a unit speed. Let C denote the circumference of the cycle. We first consider that the depth from the search space to the water surface is fixed and is denoted by L. Note that the cruising speed and the diving/surfacing speed of the AUV may not be the same. However, they can be converted to the unit speed through the distance scaling. For simplicity, AUVs are assumed to have unit speeds. This paper assumes that the perturbation of ocean currents is limited, and thus, can be relatively ignored. According to [30], the speed of ocean currents is usually less than 5 km/h (though an occasionally strong ocean current can exceed this speed). On the other hand, according to [14], the cruising speed of AUVs are 37 km/h, and the diving/surfacing speed of AUVs are 26 km/h. Since the speed of AUVs is much larger the speed of ocean currents, it is reasonable to relatively ignore the perturbation of ocean currents.

Let k denote the surfacing frequency per circulation of the cycle. The locations for surfacing are uniformly distributed along the cycle. We consider the data generation rate of the sensor to be



(a) Search space layout. (b) The given graph. (c) AUV resurfacing.

(d) Cycle construction. (e) Cycle enhancement. (f) Cycle merge.

Fig. 2. An illustration for the background and problem formulation.

Table 1
All notations used in this paper.

C	A cycle and its circumference. C_i is used for multiple cycles.
	For 3-dimensional cycles, we have $C = \oint \sqrt{f'(\lambda)^2 + g'(\lambda)^2 + h'(\lambda)^2} d\lambda$, in which $x = f(\lambda)$, $y = g(\lambda)$, and $z = h(\lambda)$ are parametric equations.
L	Depth from the search space to the water surface.
	For 3-dimensional cycles, $\bar{L}$ denotes the average cycle depth.
	$\lambda_0, \lambda_1, \ldots$ are sampling points on a cycle to compute $\bar{L}$.
k	Surfacing frequency per circulation of the cycle for an AUV.
D_n	Optimal average data reporting delay for n AUVs in a cycle C.
G	The given graph $G = (V, E)$. V is the set of vertices, and E is the set of sensing edges. V' is the set of vertices with odd degrees.
	G' is constructed by G and matching in V' via sensing edges.
	G'' is constructed by G and matching in V' via non-sensing edges.
	v, v', u, u', w, and w' are some vertices used in the proof.
S_i	Length of ith sensing edge in a cycle with non-sensing edges.
S'_i	Length of ith non-sensing edge in a cycle with non-sensing edges.
C^*	Total length of sensing edges in $C = \sum_i (S_i + S'_i)$, i.e., $C^* = \sum_i S_i$.
d	Distance between two closest points of two cycles. For example, $d(C_1, C_2)$ is the distance between C_1 and C_2.

larger than $\frac{1}{C}$, which implies that an AUV can always collect new data when it re-circulates the cycle. The objective is to minimize the average data reporting delay, from the time that the data is generated to the time that the data arrives at the water surface. It is assumed that the data can then be quickly transmitted in the air to a base station (and this part of delay is neglected). Therefore, the overall data reporting delay includes three parts as follows:

- For each AUV, its actual travel length is $C+2kL$ per circulation of the cycle. Here, $2kL$ results from k times of surfacing from depth L, counting AUV both coming up and going down. Consequently, each data item needs to wait an average time of $\frac{C+2kL}{2}$ before being transmitted from the sensor to the AUV.
- It can be seen that the cycle has been partitioned into k intervals by the surface points. The average delay, from the time that the data is received by the AUV to the time that the AUV arrives the surface point, is clearly $\frac{C}{2k}$.
- Finally, the surfacing process takes a time of L.

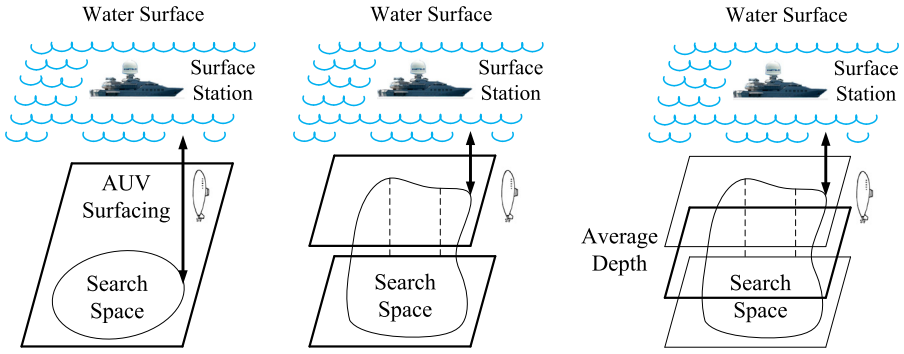
In total, the average data reporting delay for one AUV (denoted by D_1) is

$$D_1 = \frac{C + 2kL}{2} + \frac{C}{2k} + L. \tag{1}$$

Eq. (1) is minimized to $\frac{C}{2} + \sqrt{2LC} + L$, when $k = \sqrt{\frac{C}{2L}}$ (the surfacing frequency). This analysis is summarized in the following theorem:

Theorem 1. *Optimally, the AUV resurfaces after traveling a distance of $\sqrt{2LC}$ in a cyclic search space with a circumference, C, and a depth, L.*

We define the length of $\sqrt{2LC}$ as an *optimal interval*. When $L = 2C$, the traveling distance before resurfacing is $2C$, i.e., once every two circulations of the cycle. The insight behind optimal resurfacing is a tradeoff: As k increases, the data item needs to wait AUV for a longer time, but its time spent on AUV before resurfacing



(a) Basic scenario. (b) General scenario. (c) Average depth, $\bar{L}$.

Fig. 3. An illustration for the extended scenario.

reduces. Note that $k = \sqrt{\frac{C}{2L}}$ is the optimal value that balances the above tradeoff.

Next, we discuss the schedule with n AUVs for one cycle. They traverse the cycle with identical directions (i.e., clockwise or counter-clockwise). Initially, they are equally distributed on the cycle. Using a calculation that is analogous to Eq. (1), the average data reporting delay for n AUVs (denoted by D_n) is

$$D_n = \frac{C + 2kL}{2n} + \frac{C}{2k} + L. \tag{2}$$

Eq. (2) can be minimized to $\frac{C}{2n} + \sqrt{\frac{2LC}{n}} + L$, when we have $k = \sqrt{\frac{nC}{2L}}$. The optimal scheduling is that n AUVs start being uniformly distributed on the cycle, and each AUV resurfaces after traveling a distance of $\frac{C}{k} = \sqrt{\frac{2LC}{n}}$. Note that the length of the optimal interval decreases logarithmically with respect to the number of AUVs for one cycle.

4.2. General scenario

The previous subsection discusses a basic scenario, where the search space is completely parallel to the water surface. An example of a basic scenario is shown in Fig. 3(a). As an extension, this subsection studies a general scenario, where the search space may not be parallel to the water surface, as shown in Fig. 3(b). Mathematically, the search space can be described by a closed curve, which can be in turn represented by parametric equations. Let $x = f(\lambda)$, $y = g(\lambda)$, and $z = h(\lambda)$ denote the corresponding parametric equations. Since sensors in the search space can have different depths to the water surface, we use $\bar{L}$ to denote the average search space depth, as shown in Fig. 3(c). Meanwhile, C remains to denote the circumference of the cycle. By definition, we have the following equation:

$$C = \oint \sqrt{f'(\lambda)^2 + g'(\lambda)^2 + h'(\lambda)^2}\ \mathrm{d}\lambda. \tag{3}$$

Since points on the search space may have heterogeneous heights, the optimal AUV resurfacing schedule becomes more complex. Consequently, we propose a two-stage resurfacing scheme. In the first stage, we determine the travel distance between two consecutive resurfacing events. According to Theorem 1. We consider the AUV to resurface after traveling a distance of $\sqrt{2\bar{L}C}$ as an approximation. In the second stage, we determine the surface points to minimize the average data reporting delay. To reduce the delay brought by the resurfacing process, we would like to choose surface points that are closer to the water surface. Let (x_0, y_0, z_0), (x_1, y_1, z_1), $\ldots$, (x_m, y_m, z_m) denote the surfacing points, which correspond to $\lambda_0, \lambda_1, \ldots, \lambda_m$ in parametric equations. They satisfy the following constraint:

$$\int_{\lambda_i}^{\lambda_{i+1}} \sqrt{f'(\lambda)^2 + g'(\lambda)^2 + h'(\lambda)^2}\ \mathrm{d}\lambda = \sqrt{2\bar{L}C}$$
$$i = 0, 1, \ldots, m. \tag{4}$$

Eq. (4) means that the distance between consecutive surface points is $\sqrt{2\bar{L}C}$. Once the initial surface point of (x_0, y_0, z_0) is determined, the subsequent surface points can be calculated by Eq. (4). We choose the surface point that is closer to the water surface. Consequently, the initial surface point can be optimally determined as (x_0, y_0, z_0) with $\lambda_0 = \arg\max \frac{1}{m} \sum_i h(\lambda_i)$.

5. Cycle construction

In the previous section, it is assumed that the traveling cycle for AUVs is given. However, this may not be true for real-world applications. Therefore, in this section, we focus on constructing such a cycle in a given search space, aiming to minimize the circumference of the cycle. We assume that the search space is a set of segments (oil pipes), represented by a weighted given graph G. The cost associated with each edge in G is the length of the corresponding segment (the length of the oil pipe). An example of the search space is shown in Fig. 2(a), while the corresponding given graph is shown in Fig. 2(b).

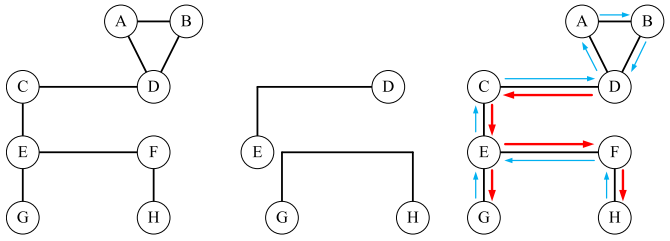
In graph theory, an Eulerian trail in a graph is a trail which visits every edge exactly once. Similarly, an Eulerian circuit or Eulerian cycle is an Eulerian trail which starts and ends on the same vertex. An Eulerian cycle exists, if and only if each vertex in the given graph has an even degree. Given an Eulerian graph, we can construct such a cycle in a linear time proposed by Hierholzer [9]: Choose any starting vertex v in G, and follow a trail of edges from that vertex until it returns to v. It is not possible to get stuck at any vertex other than v. This is because the even degrees of all vertices ensure that, when the trail enters another vertex u, there must be an unvisited edge leaving u. The trial formed in this way may not visit all the edges of the given graph. As long as there exists a vertex v that belongs to the current trail and v has adjacent edges that are unvisited, we can start another trail from v, following unvisited edges until they return to v. This new tour starting at v can join the previous tour. If we repeat the above process, then all edges can be eventually visited by the tour.

Let us consider a general graph G with odd-degree vertices (or simply odd vertices). Since the total degree of all vertices must be even (each edge is counted twice), there must exist an even number of odd vertices in G. We then pair odd vertices using *minimum weight perfect matching* [20], aiming to reduce added costs to paired odd vertices, where the cost of a pair (u, v) is the

Algorithm 1 Extended Eulerian cycle

In: A given graph G;
Out: An extended Eulerian cycle;

1: Consider subset V' of all odd vertices in G;
2: Set the cost between pairs of vertices in V' as their shortest path distances in G;
3: Find a minimum weight perfect matching in V';
4: Construct a new weighted graph G' with vertex set V' and edge set of matching pairs;
5: Combine G' and G to obtain a new weighted graph G'';
6: Return an Eulerian cycle in G'' by applying Hierholzer's algorithm.



(a) Given graph. (b) Matching. (c) Eulerian cycle.

Fig. 4. An example of Algorithm 1 with a cycle of ABDCEFHFEGECDA.

shortest path cost of u and v in G. Finally, we add a *virtual edge* between each matching pair to make all odd vertices even-degree vertices (or simply even vertices), leading to a new generated graph G''. The linear Hierholzer's algorithm is then applied to derive the Eulerian cycle.

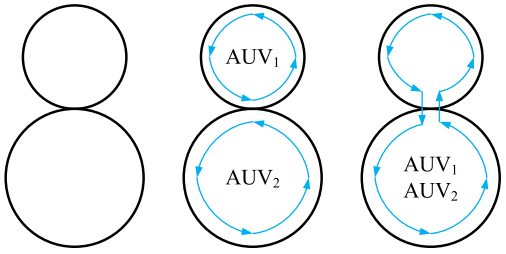
Note that the Eulerian cycle in G'' is no longer an Eulerian cycle in G, as each virtual edge G' is mapped to a set of edges in G. Therefore, several edges will be visited more than once (i.e., it is no longer a tour, but a closed walk). Hence, we call it an 'extended Eulerian cycle' for convenience. The proposed algorithm is shown in Algorithm 1 with an example in Fig. 4. The given graph is shown in Fig. 4(a), while the corresponding odd-degree vertex matching is shown in Fig. 4(b). G'' can be obtained through combining Fig. 4(a) and (b). The resultant Eulerian cycle is shown in Fig. 4(c).

To illustrate the reason for using only one large cycle instead of multiple small cycles to cover the search space, a motivational example is provided. Let us consider the scheduling of two AUVs for the search space of two neighboring cycles connected by one vertex, as shown in Fig. 5(a). Then, we have two scheduling policies as follows. Scheduling 1 assigns one AUV for each of the two neighboring cycles. The two AUVs operate independently, as shown in Fig. 5(b). Scheduling 2 considers the two neighboring cycles as one large cycle. The two AUVs operate cooperatively in the combined cycle, as shown in Fig. 5(c). We have the following theorem:

Theorem 2. *The average data reporting delay of Scheduling* 2 *is no larger than that of Scheduling* 1 *for the given search space.*

Proof. Suppose the circumferences of the two neighboring cycles are C_1 and C_2, respectively. Then, their delays are $\frac{C_1}{2}+\sqrt{2LC_1}+L$ and $\frac{C_2}{2}+\sqrt{2LC_2}+L$, respectively. Their weighted average delay for Scheduling 1 is

$$\frac{C_1\times\left(\frac{C_1}{2}+\sqrt{2LC_1}+L\right)+C_2\times\left(\frac{C_2}{2}+\sqrt{2LC_2}+L\right)}{C_1+C_2}. \tag{5}$$

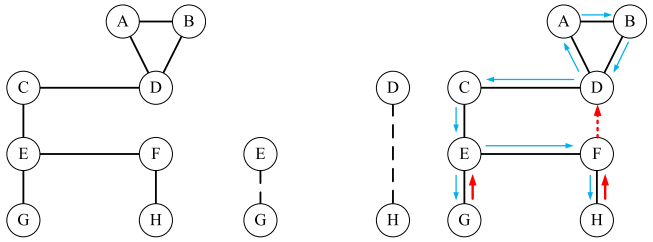


(a) Search space. (b) Scheduling 1. (c) Scheduling 2.

Fig. 5. Two scheduling policies for two neighboring cycles.

Algorithm 2 Extended cycle with non-sensing edges

In: A given graph G;
Out: A cycle with all edges in G plus some links not in G;
Same as Algorithm 1, except the change of step 2: Set the cost between each pair of vertices in V' as their geometric distance.



(a) Given graph. (b) Matching. (c) Eulerian cycle.

Fig. 6. An example of Algorithm 2 with a cycle of ABDCEGEFHFDA.

For Scheduling 2, the circumference of the combined cycle is C_1+C_2. As shown in Eq. (2), its average data reporting delay is

$$\frac{C_1+C_2}{4}+\sqrt{L(C_1+C_2)}+L. \tag{6}$$

Note that we have $\frac{(C_1+C_2)^2}{4}\le\frac{C_1^2+C_2^2}{2}$. It can also be proved that $C_1\sqrt{2LC_1}+C_2\sqrt{2LC_2}\ge(C_1+C_2)\sqrt{L(C_1+C_2)}$, or $\sqrt{2}C_1^{1.5}+\sqrt{2}C_2^{1.5}\ge(C_1+C_2)^{1.5}$. This is because derivations show that the function $\sqrt{2}+\sqrt{2}(\frac{C_2}{C_1})^{1.5}-(1+\frac{C_2}{C_1})^{1.5}$ is non-negative with respect to positive $\frac{C_2}{C_1}$. Therefore, the average data reporting delay in Eq. (5) is always no less than that in Eq. (6), meaning that Scheduling 2 is no worse than Scheduling 1. The key insight behind this theorem is that these two AUVs have balanced traversals in Scheduling 2, instead of relatively-unbalanced traversals in Scheduling 1. ■

Assuming that the given graph is connected, Theorem 2 shows that independent schedules for several cycles with small circumferences are not better than a joint schedule that combines those small cycles to a larger one. Therefore, we favor the scheduling policy that constructs one extended Eulerian cycle for AUVs to traverse all the sensing edges, rather than scheduling policies that assign the AUVs to traverse small cycles independently. If the given graph is not connected, then Algorithm 1 would obtain multiple cycles, as shown in Fig. 2(d). This case will be further explored in Section 7. The next section will introduce non-sensing edges to shorten the cycle circumference.

6. Cycle enhancement

6.1. Extended cycles

The previous section derives an extended Eulerian cycle to minimize the circumference of the cycle. In contrast, this section

will further shorten the cycle by visiting shorter non-sensing edges (edges not in G), instead of visiting redundant sensing edges (sensing edges that appear more than once in the cycle). As we recall that, a virtual edge is added between every two matching odd vertices. The cost of the virtual edge is the shortest path cost of these two vertices. However, multiple appearances of an edge do not contribute to the reduction of the data reporting delay, since data is generated at a given rate. On the other hand, odd vertices can be connected via non-sensing edges with costs measured as geographic distances in straight lines.

Algorithm 2 describes such an extension of Algorithm 1, through replacing some multiple-visited sensing edges with geometrically-shortest-distance links that are not sensing edges in the graph. An example of Algorithm 2 is shown in Fig. 6. It further reduces the circumference of the cycle by using non-sensing edges. Moreover, we have the following theorem.

Theorem 3. *In the resultant cycle constructed by Algorithm 2, the total length of non-sensing edges is no larger than the total length of sensing edges.*

Proof. We first show that no single edge (w, w') will appear in the shortest paths of two matching pairs $\{v, v'\}$ and $\{u, u'\}$. We prove this fact by contradiction. Suppose the shortest path from v to v' is $(v, \ldots, w, w', \ldots, v')$. Similarly, the shortest path from u to u' is $(u, \ldots, w, w', \ldots, v')$. Then, we will have two better matching pairs $\{v, u\}$ with paths $(v, \ldots, w, \ldots, u)$, and $\{v', u'\}$ with paths $(v', \ldots, w', \ldots, u')$. That is, edge (w, w') can be removed in the new pairings. This contradicts the goal of minimum cost perfect matching. Therefore, the total length of virtual edges generated from Algorithm 1 for G' is no larger than the total length of edges in G (i.e., the total length of sensing edges). Since Algorithm 2 is an enhancement of Algorithm 1 for matching in G', and not all virtual edges are non-sensing edges, we conclude that Theorem 3 clearly holds. ■

In general, only a subset of virtual edges in Algorithm 1 are replaced by non-sensing edges in Algorithm 2. Some sensing edges may still appear twice in the resultant cycle, as shown in Fig. 6 (e.g., sensing edges of GE and HF).

6.2. Shifting the surface point

In Section 5, we have discussed the surfacing frequency for a cycle of sensing edges with a given number of AUVs. The cycles are constructed based on an extended Eulerian cycle through multiple visits of some sensing edges. Then, the methodology in Section 4 can be used to determine the number and locations of resurfacing events in such cycles. However, this section constructs cycles with both sensing and non-sensing edges. It is meaningless for AUVs to resurface in the middle of non-sensing edges, since the data is only collected from sensing edges. If a schedule assigns an AUV to resurface at a non-sensing edge, then a better schedule can be obtained through shifting that resurface time to an earlier time when the AUV enters that non-sensing edge.

To better illustrate the idea of shifting, an example is shown in Fig. 7. If an AUV plans to resurface at a non-sensing edge, then this surface point is shifted to the end of the last sensing edge it traverses. The current shifting will not alter the next surface point. In Fig. 7, the first portion of the interval between AUV surfacing and next AUV surfacing belongs to a non-sensing edge, on which the AUV surfacing is shifted. This shifting scheme can always get a smaller delay, since it removes the unnecessary delay at a non-sensing edge, during which no data is collected. However, can we totally remove the effect of non-sensing edges by adjusting both surfacing frequency and location? The next subsection gives a definite answer, but with a stringent constraint.

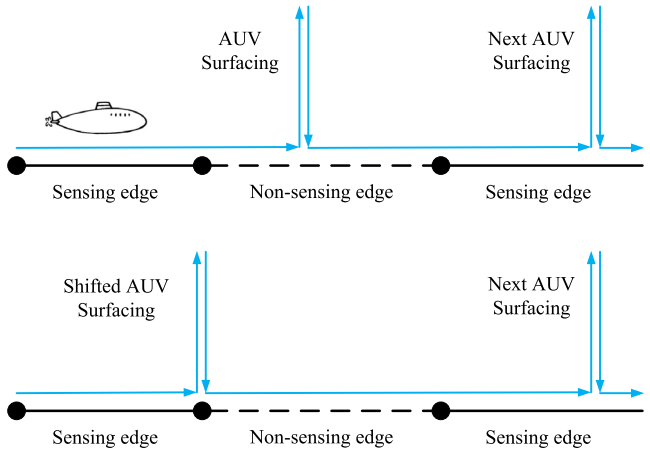


Fig. 7. An illustration of the shifting.

6.3. Exploring the optimal scheduling

In Section 4, we have explored the optimal AUV surfacing frequency for the search space of a cycle, which is composed of only sensing edges. Since Algorithm 2 constructs cycles with non-sensing edges, in this subsection, we re-explore the AUV surfacing frequency for such kinds of cycles. For simplicity, we only consider the scheduling with one AUV. Suppose the cycle is composed of alternating sensing edges (denoted by S_i, with its length as C_i) and non-sensing edges (denoted by S'_i, with its length as C'_i). In other words, the cycle of C is composed of S_1, S'_1, S_2, S'_2, and so on. Its circumference is $C = \sum_i (S_i + S'_i)$.

Here, we give out a new solution to determine the number and locations of resurfacing events for cycles with non-sensing edges. Let us remove all non-sensing edges from C to form a new cycle C^* : $S_1, S_2, \ldots, S_m$. Based on Theorem 1, we can calculate the optimal frequency and corresponding surface points within C^*, which can then be mapped back to the original cycle C as a solution. An example is shown in Fig. 8, where the original cycle C is in the left part and the new cycle C^* is in the right part. In C^*, we use the methodology stated in Section 4 to calculate the surface points (the AUV resurfaces after traveling a distance of $\sqrt{2LC^*}$, based on Theorem 1). Four surface points are determined and then mapped back to the original cycle C as the final solution. In C^*, if the interval between adjacent surfacing points (with interval length of $\sqrt{2LC^*}$) never goes across two sensing edges in C^*, then this solution is optimal. In other words, the optimality prerequisite is that the length of each sensing edge should be an integer multiple of optimal interval length (i.e., $\sqrt{2LC^*}$). If an interval goes across two sensing edges in C^*, it will intersect with a non-sensing edge in C, leading to a non-optimal result. This is because AUVs should not surface at a non-sensing edge.

Note that the optimality prerequisite for the above solution is very stringent and is not likely to be satisfied in real traces. If the length of S_i is not an integer multiple of optimal interval length, then the amount of resurfacing on S_i (calculated by $C_i/\sqrt{2LC^*}$) should be rounded off to the closest integer (except when it is less than one, then one should be used). For each sensing edge S_i, the surface points are equally distributed, so that all intervals within the sensing edge S_i have the same length and no interval goes across to a non-sensing edge. This scheme should work well, particularly when the length of each sensing edge is close to an integer multiple of $\sqrt{2LC^*}$.

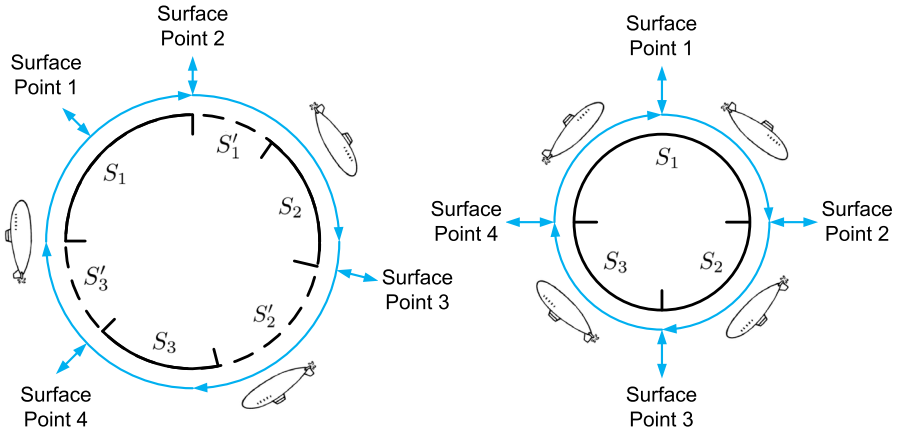
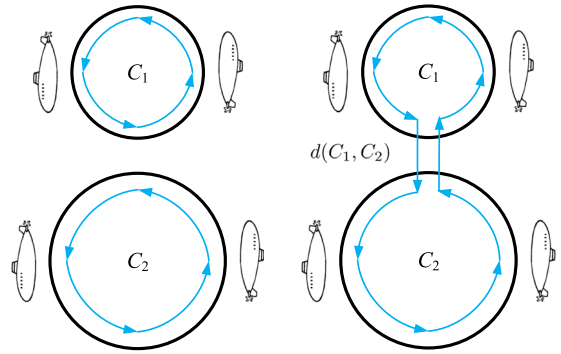


Fig. 8. The algorithm optimality. The sensing edges are in the solid line, while the non-sensing edges are in the dotted line.



(a) Two cycles. (b) Merge result.

Fig. 9. An illustration for the two-way cycle merge.

Algorithm 3 Cycle merge (minimum delay criterion)

In: The resulting cycles by Algorithm 2;
Out: The cycle merge result;
1: **while** there exists more than one cycle **do**
2: **for** each pair of cycles **do**
3: Calculate the merge benefit as the resulting value difference between Eqs. 7 and 8;
4: **if** the largest merge benefit is positive **then**
5: Merge that pair of cycles as a bigger cycle;
6: **else**
7: Break the while loop;
8: **return** The cycle merge result;

7. Cycle merge

7.1. Two-way merge

In Sections 5 and 6, we have discussed how to construct the cycles from the given graph of the search space. However, the given graph is not necessarily connected. Therefore, multiple cycles may be obtained, as shown in Fig. 2(d) and (e). However, we observe that cycles can be further merged with a cooperative AUV scheduling. As previously shown in Fig. 2(f), the two smaller cycles of ABDCA and EFHGE in Fig. 2(e) are merged to a bigger cycle of ABDFHGECA. The cycle merge can reduce the average data reporting delay [32], by balancing the AUV traversals in different cycles. This subsection discusses the two-way cycle merge.

As shown in Fig. 9, suppose we have two cycles, C_1 and C_2. The distance between C_1 and C_2 is defined as the smallest distance between two points that are located in C_1 and C_2, respectively. Let $d(C_1, C_2)$ denote this distance. Suppose there are n_1 AUVs assigned to the cycle C_1, while there are n_2 AUVs assigned to the cycle C_2. According to Eq. (2), the average data reporting delay for these two cycles is

$$\frac{C_1 \times \left(\frac{C_1}{2n_1} + \sqrt{\frac{2LC_1}{n_1}} + L\right) + C_2 \times \left(\frac{C_2}{2n_2} + \sqrt{\frac{2LC_2}{n_2}} + L\right)}{C_1 + C_2}. \tag{7}$$

If C_1 and C_2 are merged, then we can obtain a bigger cycle with a circumference of $C_1 + C_2 + 2d(C_1, C_2)$. Meanwhile, $n_1 + n_2$ AUVs can be assigned to this merged cycle. The merged cycles include both sensing edges and non-sensing edges. If we use the shifting strategy in Section 6.2 to schedule these AUVs, then the average data reporting delay for the merged cycle should be no larger than

$$\frac{C_1 + C_2 + 2d(C_1, C_2)}{2(n_1 + n_2)} + \sqrt{\frac{2L[C_1 + C_2 + 2d(C_1, C_2)]}{(n_1 + n_2)}} + L. \tag{8}$$

If we compare the resulting values in Eqs. (7) and (8), then we can determine whether C_1 and C_2 should be merged. The insight behind the cycle merge is similar to that in Theorem 2. If the traversals of AUVs in C_1 and C_2 are more unbalanced, then we are more likely to merge C_1 and C_2, in order to balance AUV traversals. For example, if C_1 is large, C_2 is small, n_1 is small, and n_2 is large, then we should merge C_1 and C_2, assuming that $d(C_1, C_2)$ is not too large. This is because C_2 has too many AUVs that can be re-balanced to collect the data from C_1, in which the AUVs are not sufficient.

A greedy cycle merge method is proposed to further reduce the average data reporting delay, as shown in Algorithm 3. At each step, it greedily merges the pair of cycles that yields the largest merge benefit (the value difference between Eqs. (7) and (8)). The cycle merge only happens when the given graph is not connected.

Algorithm 4 Cycle merge (most unbalanced criterion)

In: The resulting cycles by Algorithm 2;
Out: The cycle merge result;
1: **while** there exists more than one cycle **do**
2: **for** each pair of cycles **do**
3: Calculate their cycle circumference difference;
4: **if** the merge of the pair of cycles with the largest cycle circumference difference reduces the average data reporting delay **then**
5: Merge that pair of cycles as a bigger cycle;
6: **else** break the while loop;
7: **return** The cycle merge result;

Algorithm 5 Cycle merge (geographically closest criterion)

In: The resulting cycles by Algorithm 2;
Out: The cycle merge result;
1: **while** there exists more than one cycle **do**
2: **for** each pair of cycles **do**
3: Calculate their geographical distance;
4: **if** the merge of the pair of cycles with the closest geographical distance reduces the average data reporting delay **then**
5: Merge that pair of cycles as a bigger cycle;
6: **else** break the while loop;
7: **return** The cycle merge result;

7.2. Different merge criteria and three-way merge

The last subsection demonstrates that AUVs in adjacent cycles should be scheduled collaboratively. This subsection discusses two different merge criteria: the most unbalanced criterion merges the most unbalanced pairs of cycles first, and the geographically closest criterion merges the geographically closest pairs of cycles first. The merging process is described as follows: first, we calculate the average data reporting delay of each cycle in the monitoring area. Then, we scan all the merging combinations, and merge the two cycles determined by the criterion. This merge process is repeated, until there exists no merge that can minimize the average data reporting delay. Algorithms 4 and 5 are proposed to implement the most unbalanced criterion and the geographically closest criterion, respectively. Different merge criteria can have different advantages under certain scenarios, as shown in the experiments.

Moreover, the cycle merge can be implemented in a three-way manner. In each greedy iteration, we can additionally try to merge three cycles. Three-way merge naturally extends two-way merge by additionally considering merging three cycles in a greedy iteration. However, it is different from consecutive two-way merges. An example is shown in Fig. 10, where Fig. 10(a) includes the scenario of three cycles that are close to each other. Fig. 10(b) shows the result of consecutive two-way merges, which takes four inter-cycle traverses. Meanwhile, Fig. 10(c) shows the result of the three-way merge, which only takes three inter-cycle traverses. It can be seen that, through reducing AUV inter-cycle traverses, a three-way merge can further optimize the multiple-cycle AUV trajectory planning.

7.3. Parallel cycle merge implementation

This subsection shows that the proposed cycle merge algorithm can be implemented in parallel. The key observation is that we are not likely to merge cycles, which are geographically remote. Consequently, the key idea is to cut the scenario into several small regions, and then conduct the cycle merge algorithm for each small region in parallel. Note that the parallel merge algorithm may degrade the merge performance, since inter-regional merges are no longer considered. Therefore, there exists a tradeoff between the parallelism speedup and the performance.

Experiments are conducted to verify the above tradeoff in Algorithm 3. We generate search spaces of 500 m $\times$ 500 m with 10 and 25 circles (sparsely and densely distributed circles), respectively. The circle circumference is uniform-randomly selected from 40, 60, and 80 m. Circles are not nested or intersected. There exists a sensor in every meter for each circle. The data generation speed is 10 Mb/s for each sensor. 20 AUVs are used and the AUV speed is 1 m/s. The sea depth of the search space is 100 m. The number of AUVs in each circle is proportional to its circumference. The experimental result is shown in Fig. 11. It can be seen that, a larger degree of parallelism leads to a larger performance degradation (i.e., a larger average data reporting delay), as well as a smaller running time. In Fig. 11(a), when the degree of the parallelism is 4, the average data reporting delay increases by about 11%, but the program running time decreases by about 60%. Another notable point is that, the parallel merge algorithm performs better in the scenario with sparsely distributed circles than the scenario with densely distributed circles. This is because the scenario with sparsely distributed circles has less inter-regional merges than the scenario with densely distributed circles.

8. Experiments

Experiments are conducted to evaluate the performances of the proposed algorithms. After presenting the basic settings, the evaluation results are shown from different perspectives to provide insightful conclusions.

8.1. Basic settings

Our experiments denote the shifting algorithm in Section 6.2 as Algorithm 2s, and the approximated optimal algorithm with round-off in Section 6.3 as Algorithm 2r. For Algorithms 1, 2, 2s, and 2r, if the given graph is not connected, they will obtain multiple cycles. For this case, the number of AUVs distributed to each cycle is proportional to the cycle circumference. Algorithms 3, 4, and 5 are cycle merge algorithms with different merging criteria. One baseline is used for comparison:

- Baseline distributes AUVs according to the lengths of the sensing edges. The number of AUVs assigned to a sensing edge is proportional to its length (i.e., edge weight). For each sensing edges, the corresponding AUVs also uniformly go back and forth along that sensing edges.

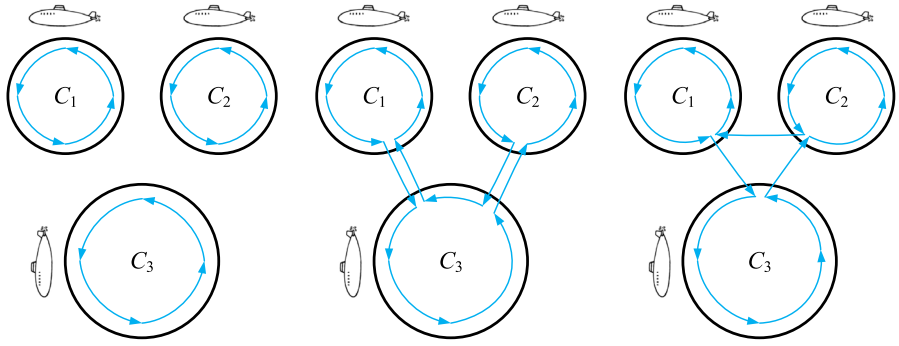
Three related works are used for comparison:

- Sugihara [26] focused on the trajectory scheduling problem for the AUV to achieve the smallest data delivery delay under energy constrains (sensors have identical energy consumptions in our experiments). An approximation algorithm is provided.
- Moazzez [23] used multiple AUVs to cooperatively visit sensors. Sensors are divided into several groups, while each sensor group has an AUV to collect the data by solving a traveling salesman problem.
- Ma [22] modeled trajectory planning problems as traveling salesman problems with additional distance and time constraints for multiple AUVs. A heuristic trajectory planning algorithm was provided.

Acoustic techniques [10,3] are not compared, since they suffer from a significant signal attenuation. Based on [27], the data rate of the acoustic technique is usually limited to 10 kbps with the maximum transmission range up to 100 m. Due to the limited data rate and the limited transmission range, acoustic techniques are not applicable in our scenario.

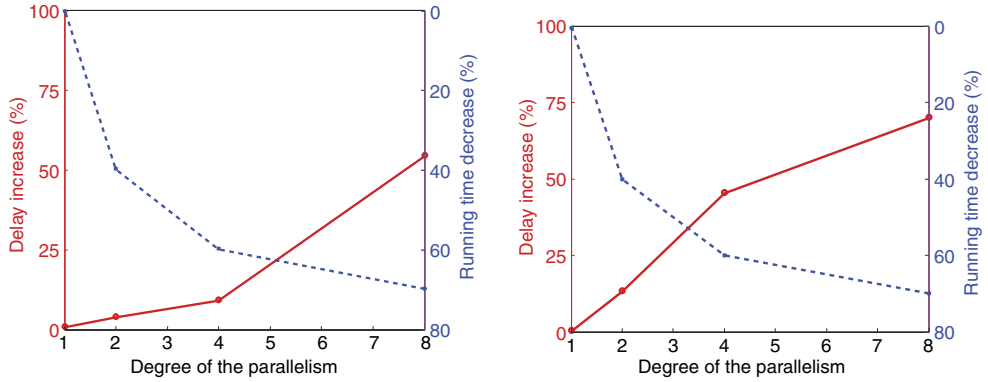
The average data reporting delay is the key performance metric in the experiments. We are interested in how the average data reporting delay is impacted by the settings (e.g., the sea depth, the graph density, the percentage of non-sensing edges in the cycle, the number of AUVs, and so on). Experiments are organized as follows:

- Section 8.2 will test the performance gap between Algorithms 1 and 2 (cycle construction methods based on sensing and non-sensing edges in Sections 5 and 6.1).
- Section 8.3 will test the performance gap between Algorithms 2, 2s, and 2r (resurfacing points adjustments for cycles with non-sensing edges in Section 6).
- Section 8.4 will test the performance gap between Algorithms 3, 4, and 5 (different cycle merge criteria in Section 7).



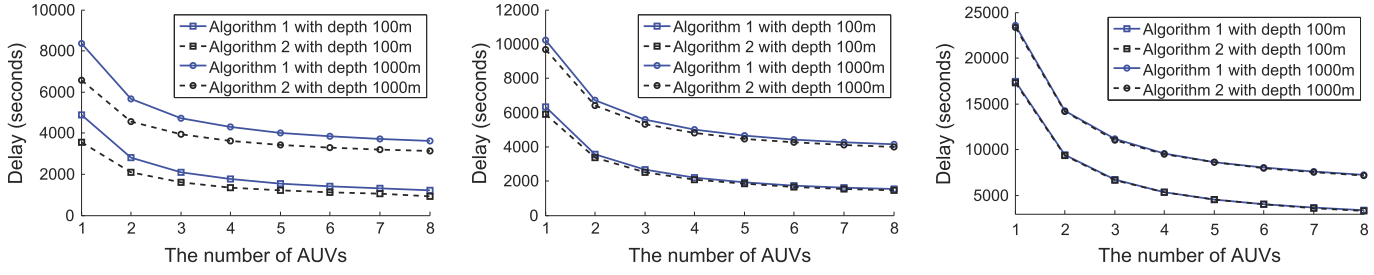
(a) Three cycles. (b) Two-way merge. (c) Three-way merge.

Fig. 10. An illustration for the three-way cycle merge.



(a) Sparsely distributed cycles. (b) Densely distributed cycles.

Fig. 11. The tradeoff between the parallelism speedup and the performance.



(a) Given 20 sensing edges. (b) Given 100 sensing edges. (c) Given 500 sensing edges.

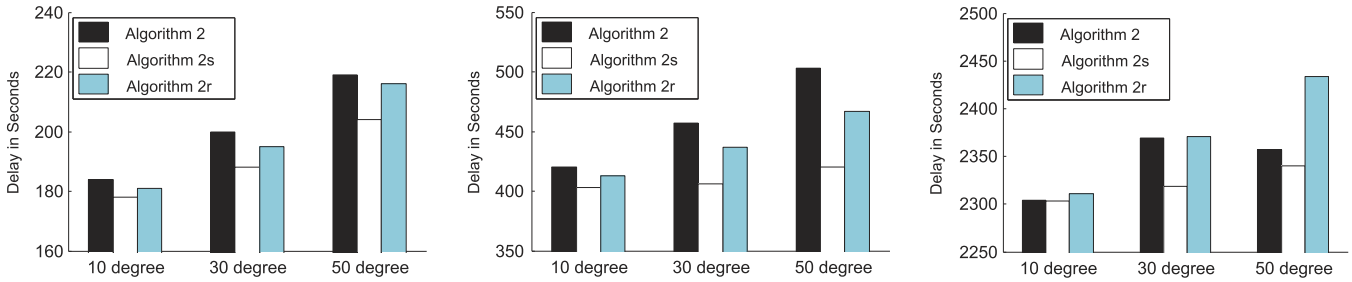
Fig. 12. Performance gap between Algorithms 1 and 2.

- Section 8.5 will test the performance gap under basic and general scenarios (2-D and 3-D search spaces described in Section 4), with respect to all the algorithms (including comparison algorithms).
- Section 8.6 will test the performance of all algorithms in real traces to demonstrate their applicability in the real world.

Sections 8.2, 8.3, 8.4, and 8.5 are simulations based on synthetic traces to capture the performance gap and the insight. The AUV speed is set to be 1 m/s in these subsections. Section 8.6 includes simulations based on several real oil pipe traces to evaluate the algorithm applicabilities. In real traces, the cruising speed of AUVs are set to be 37 km/h, and the diving/surfacing speed of AUVs are set to be 26 km/h, according to [14]. The data generation speed is simulated to be 10 Mb/s for each sensor.

8.2. Performance gap between Algorithms 1 and 2

This subsection tests the performance gap between Algorithms 1 and 2. Note that Algorithm 1 constructs the cycle through shortest paths, while Algorithm 2 constructs the cycle through geometrically-shortest-distance links. For these two algorithms, we would like to verify the impact of the graph density, in terms of the average data reporting delay. Our experiments are based on a special synthetic trace, which is generated through a uniformly-random placement of 100 nodes on a 100 m $\times$ 100 m square. To guarantee the graph connectivity, a minimum spanning tree is constructed. Then, sensing edges, with given total numbers of 20, 100, and 500, are introduced to uniform-randomly connect these nodes. Note that the given number of sensing edges represents the



(a) Given depth $L = 10$ m. (b) Given depth $L = 100$ m. (c) Given depth $L = 1000$ m.

Fig. 13. Performance gap between Algorithms 2, 2s and 2r.

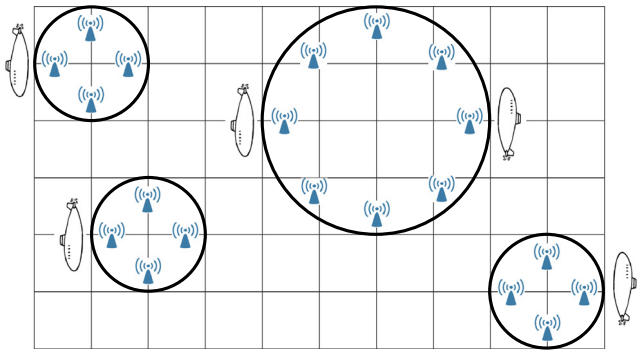


Fig. 14. An example of the synthetic trace in Section 8.4. It has a 10 m × 6 m grid and four circles. Three circles have radiuses of 2 m, and one circle have a radius of 4 m.

graph density. For each edge in the synthetic trace, there exists a sensor in every meter. The given depth of the search space is set as 100 m and 1000 m, respectively. Since this synthetic trace is randomly generated, experiments on this synthetic trace are repeated to determine the average, until the confidence interval of the average result is sufficiently small (1% for a 90% probability).

The experimental results are shown in Fig. 12. Different subfigures have different graph densities. The sensing edges in Fig. 12(a) are sparser than the sensing edges in Fig. 12(c). The performance gap between Algorithm 1 and Algorithm 2 is significant in Fig. 12(a), which represents the results for the sparsest trace. However, the performance gap between these two algorithms decreases when the trace becomes denser, as shown in Fig. 12(b) and (c). This is because the gap between (i) pairwising odd vertices through the shortest path, and (ii) that through the geometrically-shortest-distance links, becomes smaller as the trace gets denser. If the trace is sparse, pairwising odd vertices through the shortest path could be very costly, since the geometrical distances among these vertices could be much smaller. Another observation is that a larger sea depth brings a larger delay. This is very intuitive, since AUVs need more time to resurface. Finally, the last observation is that the reduction of the average data reporting delay brought by one more AUV decreases with respect to the current number of AUVs. This effect follows the law of the diminishing return. In Fig. 12(a), if the sea depth is 1000 m, the delay brought by Algorithm 1 with one AUV is about 8000s. Meanwhile, if 8 AUVs are used, then the delay reduces to about 4000s (about 50% reduction). Generally speaking, a denser and larger trace should use more AUVs to achieve a small average data reporting delay.

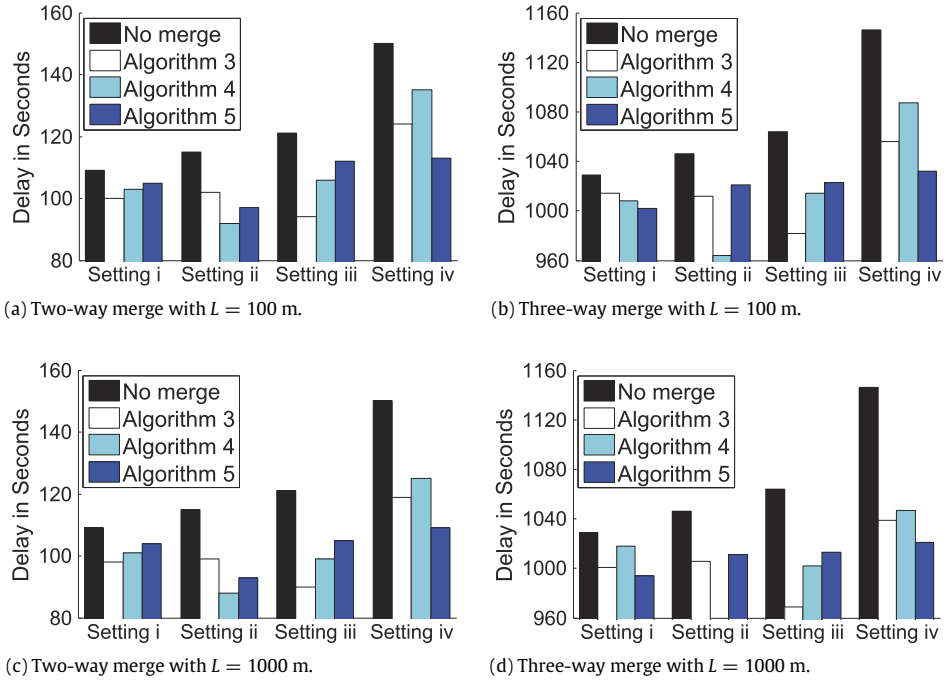
8.3. Performance gap between Algorithms 2, 2s, and 2r

This subsection tests the performance gap between Algorithms 2, 2s, and 2r. These algorithms are introduced in Section 6, where we adjust the number and locations of resurfacing events for cycles with non-sensing edges. Our experiments are based on another synthetic trace. It includes 100 nodes and has a shape of "V", which corresponds to the layout of the sensing edges in the search space. Each side of the V-trace has a length of 100 m and there exists a sensor in every meter for it. The intersection angle between the two sides of the V-trace is given as 10°, 30°, and 50°, respectively. A smaller intersection angle brings a shorter geometrical distance between the two ends of the V-trace. Only one AUV is used. The given depth of the search space is set as 10 m, 100 m, and 1000 m, respectively. The above parameter settings are used to tune the graph density for the proposed algorithms.

The experimental results are shown in Fig. 13, in terms of the average data reporting delay. Since we have adjusted the number and locations of resurfacing events in Section 6 for cycles with non-sensing edges, these traces are used to validate the improvements of those adjustments. As previously mentioned, this synthetic trace has a shape of V. The intersection angles between the two sides of the V-trace are given as 10°, 30°, and 50°, respectively. A smaller intersection angle means that the corresponding non-sensing edges are shorter, and thus, the adjustment strategy should be less efficient. It can be seen that the Algorithm 2s (shifting scheme) is very effective, especially when the trace has a cycle of long non-sensing edges (i.e., the trace with an intersection angle of 50°). On the other hand, if the total length of non-sensing edges is very small, then the performance improvement brought by Algorithm 2s is limited. The delay reduction brought by Algorithm 2s ranges from about 5%–20%, compared to Algorithm 2. This is because a longer non-sensing edge means that AUVs are more likely to surface on that non-sensing edge, which should be adjusted by Algorithm 2s. In contrast, Algorithm 2r has a limited reduction on the average data reporting delay. Although Algorithm 2r could be optimal under a stringent constraint, that constraint is uncommon (at least in this synthetic trace). Therefore, the shifting scheme is recommended for its simplicity and effectiveness.

8.4. Performance gap between Algorithms 3, 4, and 5

This subsection tests the performance gap between Algorithms 3, 4, and 5. These algorithms are introduced in Section 7, where we use different cycle merge criteria to reduce the average data reporting delay. We also design a synthetic trace for evaluations. This synthetic trace generates a search space of 500 m × 500 m grid with a given number of circles. Each circle serves as a cyclic search space (i.e., a cycle for AUVs). Circle centers are randomly-located on grid intersections. The radius of a circle is chosen from 10 m and 50 m. Circles are not nested or intersected. There exists a sensor in every meter for each circle. An example of such a synthetic trace is shown in Fig. 14. This subsection involves 500 AUVs, which are evenly distributed to circles according to their radiuses. The sea depth of the search space is set as 100 m and 1000 m, respectively. We have four different circle settings:



(a) Two-way merge with $L = 100$ m.

(b) Three-way merge with $L = 100$ m.

(c) Two-way merge with $L = 1000$ m.

(d) Three-way merge with $L = 1000$ m.

Fig. 15. Performance gap between Algorithms 3, 4 and 5.

- Setting i generates 20 circles of radiuses 10 m.
- Setting ii also generates 20 circles. 10 circles have radiuses of 10 m and 10 circles have radiuses of 50 m.
- Setting iii generates 20 circles of radiuses 50 m.
- Setting iv generates 500 circles of radiuses 10 m.

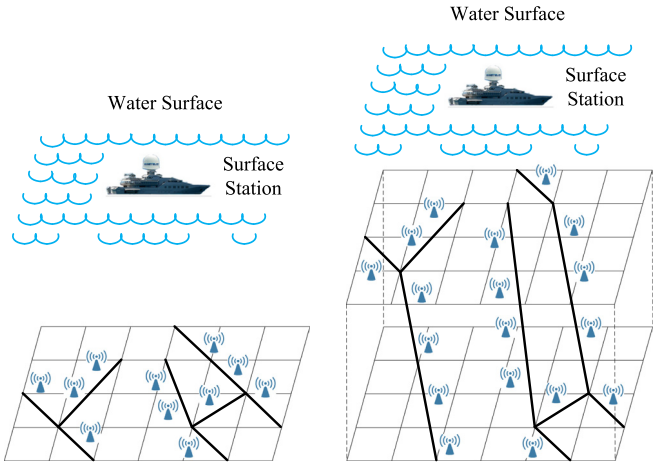
Settings i, ii, and iii have the same number of circles with different radiuses. Settings i and iv have different number of circles, but have the same circle radiuses. Settings iii and iv have different number of circles and different circle radiuses, but have the same circle areas in total ($500 \times \pi 10^2 = 20 \times \pi 50^2$). Since circles are randomly-located, experiments on this synthetic trace are repeated to determine the average, until the confidence interval of the average result is sufficiently small (1% for a 90% probability).

We start with two-way cycle merges, which merges two cycles (i.e., circles) in each greedy iteration. The evaluation results are shown in Fig. 15(a) and (b). Theorem 1 is used to schedule the AUV resurfacing in each circle. In Fig. 15, "no merge" means that AUVs in different circles surface independently, i.e., AUVs will not cooperate with each other. It can be seen that cycle merge algorithms (Algorithms 3, 4, and 5) can always reduce the average data reporting delay, compared to the "no merge" algorithm. This is because cycles are merged only if the average data reporting delay can be reduced. In the setting i, the delay reduction brought by the cycle merge algorithms is very limited. For example, the delay reduction brought by Algorithm 5 with $L = 100$ is only 4s (109s − 105s = 4s), which is only about 4% performance improvement. This is because circles are overly sparse in the setting i (20 circles of radiuses 10 m only take 2.5% area of the 500 m × 500 m grid). As a result, circles are relatively far from each other, and every few circles are eventually merged in the setting i. In contrast, in the setting ii, the delay reduction brought by the cycle merge algorithms is significant. The delay reduction brought by Algorithm 4 with $L = 100$ is 23s (115s − 92s = 23s), which is 20% performance improvement. Algorithm 4 outperforms Algorithms 3 and 5 in the setting ii (for both $L = 100$ and $L = 1000$ m), since it balances the merge between cycles with different circumferences (circles with different radiuses of 10 m and 50 m). However, Algorithm 3 outperforms Algorithms 4 and 5 in the setting iii, which has 20 circles of radiuses 50 m. It means that we mainly consider the merge delay reduction when cycles have large circumferences. In contrast, Algorithm 5 outperforms Algorithms 3 and 4 in the setting iv, which has 500 circles of radiuses 10 m. The geographical distance is the most important factor, when cycles have small circumferences. Different cycle merge criteria have advantages under different scenarios.

We also evaluate three-way cycle merges, which merges three cycles (i.e., circles) in each greedy iteration. Three-way merge is described in Section 7.2, and is illustrated in Fig. 10. It naturally extends two-way merge by additionally considering merge three cycles in a greedy iteration. The evaluation results for three-way merges are shown in Fig. 15(c) and (d). Compared to two-way merges, three-way merges can further reduce the data reporting delay (however, less than 10% performance improvements are brought). For example, in setting ii and $L = 100$ m, Algorithm 4 has average data reporting delays of 92s for two-way merge and 88s for three-way merge. In setting iv and $L = 100$ m, Algorithm 5 has average data reporting delays of 113s for two-way merge and 109s for three-way merge. Clearly, three-way merge outperforms two-way merge through reducing AUV inter-cycle traverses. As a tradeoff, three-way merge has a higher time complexity than two-way merge, since it scans tuples of cycles instead of pairs of cycles in each greedy iteration.

8.5. Experiments for basic and general scenarios

This subsection tests all the algorithms (including comparison algorithms), under both basic and general scenarios (2-D and 3-D search spaces described in Section 4). Our experiments are based on a synthetic trace that is similar to the one in Section 8.2. We also generate the synthetic trace through a uniformly-random placement of 100 nodes on a 100 m × 100 m square. Sensing edges, with given total numbers of 100 and 500, are introduced to uniform-randomly connect these nodes. The given number of sensing edges represents the graph density. Note that the given graph may not be connected. There exists a sensor in every meter for each edge. Ten AUVs are used. For the basic scenario, the given



(a) Basic scenario (2-D search space). (b) General scenario (3-D search space).

Fig. 16. Examples for basic and general scenarios (10 nodes and 8 edges).

Table 2
Results for basic and general scenarios.

Trace setting	Comparison algorithms	The scenario	
		Basic scenario	General scenario
100 sensing edges	Algorithm 1	1853s	1954s
	Algorithms 2, 2s, 2r	1805s, 1724s, 1796s	1865s, 1794s, 1843s
	Algorithms 3, 4, 5	1643s, 1704s, 1694s	1734s, 1783s, 1755s
	Baseline	2563s	2801s
	Sugihara [26]	2216s	2418s
	Moazzez [23]	1756s	1814s
	Ma [22]	1945s	2074s
500 sensing edges	Algorithm 1	4175s	4421s
	Algorithms 2, 2s, 2r	4104s, 4053s, 4089s	4363s, 4312s, 4355s
	Algorithms 3, 4, 5	3861s, 3978s, 3942s	4144s, 4301s, 4234s
	Baseline	5236s	5812s
	Sugihara [26]	4632s	4893s
	Moazzez [23]	4005s	4288s
	Ma [22]	4154s	4341s

depth of the search space is set as 100 m, as shown in Fig. 16(a). For the general scenario, the depth of each node is uniform-randomly chosen from 50 m and 150 m (the average search space depth remains 100 m). An example is shown in Fig. 16(b). Since this synthetic trace is randomly generated, experiments on this synthetic trace are also repeated to determine the average results.

The evaluation results for all the algorithms are shown in Table 2. For the synthetic trace with 100 sensing edges, Algorithm 3 outperforms all the other algorithms under basic and general scenarios. Compared to Baseline, Algorithm 3 brings more than 20% delay reductions. Sugihara [26] performs poorly, since it sacrifices the average performance to guarantee a bounded result. Although Moazzez [23] outperforms Algorithms 1 and 2, it still has a larger data reporting delay than Algorithm 3. This is because it is based on traveling salesman problems that visit sensors instead of sensing edges. Ma [22] fails to outperform Algorithms 3, 4, and 5, since it do not cooperate AUVs among different cycles. Algorithm 2 outperforms Algorithm 1 through constructing cycles with non-sensing edges. Algorithms 2s and 2r improve Algorithm 2 by adjusting the AUV resurfacing points on non-sensing edges. Algorithms 3, 4, and 5 improve Algorithm 2s by cooperatively scheduling AUVs in different cycles. Another important observation is that the result in the general scenario is close to (is slightly larger than) the result in the basic scenario. This is because the average cycle circumference in the general scenario is larger than that in the basic scenario (a larger cycle circumference). For the synthetic trace with 500 sensing edges, the results are similar. Therefore, in a general scenario with a 3-D search space, we can use the average search space depth to approximate the AUV resurfacing scheduling.

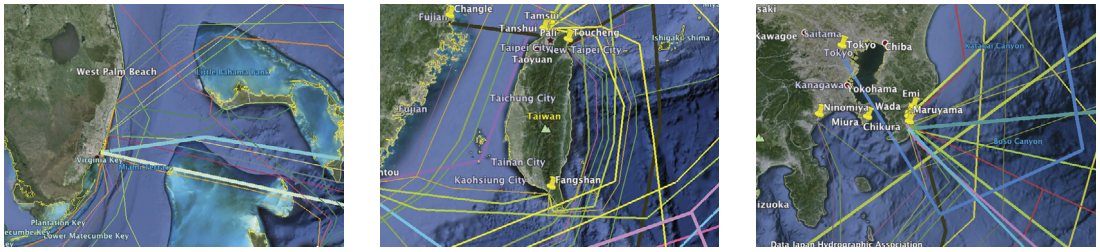
8.6. Experiments in the real trace

This subsection tests the performance of all algorithms in real traces to demonstrate their applicabilities in the real world. The real traces are published in [11], which includes three parts. The first part includes the oil pipe layout near Florida, including SAm-1, COLUMBUS I to III, Mid-Atlantic Crossing (MAC), BAHAMAS-1, BAHAMAS-2, GlobeNet, BDNSi, and so on. The second part includes the oil pipe layout near Taiwan, including CUCN, EAC, FNAL, APCN, SEA-ME-WE, APCN, RNAL, ASE, and so on. The third part includes the oil pipe layout near Japan, including TOKYO, Australia–Japan, Cable, ROTACS, RNAL, APG, ASE, SJC, Trans-Pacific, Express, EAC, and so on. These oil pipe layouts are shown in Fig. 17. Meanwhile, the sea depth is set to be the average sea depth in the real world of 3790 m [35]. Sensors are uniformly placed along each pipe, while the distance between two adjacent sensors on a pipe is 1 km.

The experimental results for the real trace are shown in Table 3, where "h" represents hours. We use ten and twenty AUVs to collect the data, respectively. Baseline has the worst performance, since Baseline schedules AUVs independently. AUVs on different oil pipes do not cooperate with each other for the data collection. Sugihara [26] performs poorly, since it sacrifices the average performance to guarantee a bounded result. Moazzez [23] performs poorly, since it does not visit sensors on a sensing edge consecutively. Ma [22] performs poorly, since it does not schedule AUVs cooperatively.

There exists a significant performance gap between Algorithms 1 and 2. This is because the real trace is sparse, leading to a large gap between pairwising odd vertices through the shortest path and that through the geometrically-shortest-distance links. Since the Japan oil pipe trace is the sparsest, Algorithms 1 and 2 have the largest performance gap (more than 10%) in the Japan oil pipe trace than the other traces. Algorithm 2s can further reduce the average data reporting delay of Algorithm 2 by about 5% in the Taiwan and Japan oil pipe traces. This is because AUVs should not resurface in the middle of non-sensing edges, since the data is only collected from sensing edges. The performance gap between Algorithms 2 and 2s is small in the Florida oil pipe trace, since there are few cycles with long non-sensing edges. Another notable point is that Algorithm 2r may not outperform Algorithm 2s, depending on the trace. The optimality prerequisite of Algorithm 2r is very stringent and may not be satisfied in real applications, leading to performance degradations. Algorithms 3, 4, and 5 bring further reductions on the average data reporting delay through cycle merges, based on different criteria (minimum delay, most unbalanced, and geographically closest). When we have 10 AUVs, they have about 5%–15% less delay than Algorithm 2s. When we have 20 AUVs, they have 10%–20% less delay than Algorithm 2s, depending on the real trace. Different cycle merge criteria have different performances with respect the real traces. In summary, the proposed algorithms are applicable in the real traces.

The major advantages of the proposed techniques over existing techniques are their smaller average data reporting delays. It can be seen that the average data reporting delay is less than half an hour in the Florida oil pipe trace, and is less than ten hours in the Taiwan and Japan oil pipe traces. Our major advantages come from cooperative AUV schedules through combinational



(a) Florida oil pipes. (b) Taiwan oil pipes. (c) Japan oil pipes.

Fig. 17. Oil pipe layouts in the real traces.

Table 3
Average data reporting delay (in hours) for the real traces.

Trace setting	Comparison algorithms	The number of AUVs	
		10 AUVs	20 AUVs
Florida oil pipe trace	Algorithm 1	0.59 h	0.36 h
	Algorithms 2, 2s, 2r	0.52 h, 0.49 h, 0.52 h	0.34 h, 0.32 h, 0.34 h
	Algorithms 3, 4, 5	0.45 h, 0.43 h, 0.40 h	0.26 h, 0.23 h, 0.21 h
	Baseline	0.85 h	0.58 h
	Sugihara	0.71 h	0.43 h
	Moazzez	0.46 h	0.29 h
	Ma	0.49 h	0.31 h
Taiwan oil pipe trace	Algorithm 1	7.87 h	7.51 h
	Algorithms 2, 2s, 2r	7.49 h, 7.26 h, 7.29 h	7.24 h, 7.05 h, 7.04 h
	Algorithms 3, 4, 5	6.76 h, 6.95 h, 6.86 h	6.47 h, 6.67 h, 6.61 h
	Baseline	9.24 h	8.43 h
	Sugihara	8.57 h	7.94 h
	Moazzez	7.25 h	7.15 h
	Ma	7.31 h	7.22 h
Japan oil pipe trace	Algorithm 1	9.84 h	8.92 h
	Algorithms 2, 2s, 2r	8.65 h, 8.22 h, 8.17 h	8.13 h, 7.85 h, 7.85 h
	Algorithms 3, 4, 5	7.75 h, 7.56 h, 7.81 h	7.43 h, 7.29 h, 7.38 h
	Baseline	11.43 h	10.28 h
	Sugihara	10.36 h	9.75 h
	Moazzez	8.71 h	8.31 h
	Ma	7.93 h	7.78 h

approaches. Given a graph, cycles with sensing edges and non-sensing are constructed to reduce the AUV traversing distance. Based on resultant cycles, AUV resurfacing frequencies and points are optimized. Cycles are also merged to reduce their total circumferences.

9. Conclusions and future directions

This paper studies a data collection problem in the deep sea. The scenario is based on a search space that is a set of oil pipes deployed in the seabed. Sensors are deployed along the oil pipes for leak detection, while AUVs are used to collect the data from the sensors and then resurface to report the data. We focus on the scheduling of the AUV trajectory planning, as well as the AUV resurfacing frequencies and their locations. An optimization problem is formulated by minimizing the average data reporting delay. The AUV trajectory planning is simplified to an extended Euler cycle problem, where we construct cycles through both sensing edges and non-sensing edges. We also discuss the cycle merge with different merge criteria, where AUVs in different cycles can operate cooperatively. The cost-effectiveness of the proposed approach is validated by extensive experiments.

Our work has several future directions. The first future direction is to consider a more realistic speed model of AUVs. The current model assumes that the cruising speed and the diving/surfacing speed of AUVs are fixed. This AUV speed model can be improved by considering the acceleration, when AUVs switch among cruising, diving, and surfacing. The speed of ocean currents should also be included to justify the AUV trajectory planning. The second future direction is to deeply explore the AUV resurfacing schedule, when the given cycle includes non-sensing edges. Schedule optimality or approximation ratio is desired to guarantee the system performance. The third direction is to implement real AUVs to test their applicabilities. Although theoretical analysis is conducted, the practical performance needs to be validated. The financial budget of AUV implementations should be evaluated, in terms of its cost-effectiveness.

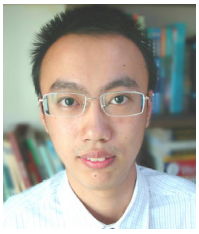
Acknowledgment

This research was supported in part by NSF Grants ECCS 1231461, ECCS 1128209, CNS 1065444, and CCF 1028167.

References

[1] I.F. Akyildiz, D. Pompili, T. Melodia, Underwater acoustic sensor networks: research challenges, Ad Hoc Networks 3 (3) (2005) 257–279.
[2] M. Ayaz, I. Baig, A. Abdullah, I. Faye, A survey on routing techniques in underwater wireless sensor networks, J. Netw. Comput. Appl. 34 (6) (2011) 1908–1927.
[3] N. Baccour, A. Koubâa, L. Mottola, M.A. Zúñiga, H. Youssef, C.A. Boano, M. Alves, Radio link quality estimation in wireless sensor networks: A survey, ACM Trans. Sens. Netw. 8 (4) (2012) 34:1–34:33.
[4] M.M. Bin Tariq, M. Ammar, E. Zegura, Message ferry route design for sparse ad hoc networks with mobile nodes, in: Proceedings of ACM MobiHoc, 2006, pp. 37–48.
[5] V. Chandrasekhar, W.K. Seah, Y.S. Choo, H.V. Ee, Localization in underwater sensor networks: survey and challenges, in: Proceedings of ACM WUWNet, 2006, pp. 33–40.
[6] H.-H. Cho, C.-Y. Chen, T.K. Shih, H.-C. Chao, Survey on underwater delay/disruption tolerant wireless sensor network routing, IET Wirel. Sens. Syst. (2014) 1–10.
[7] J.-H. Cui, J. Kong, M. Gerla, S. Zhou, The challenges of building mobile underwater wireless networks for aquatic applications, IEEE Netw. 20 (3) (2006) 12–18.
[8] M. Eichhorn, R. Taubert, C. Ament, M. Jacobi, T. Pfuetzenreuter, Modular AUV system for sea water quality monitoring and management, in: Proceedings of MTS/IEEE OCEANS-Bergen, pp. 1–7, 2014.
[9] H. Fleischner, X.1 algorithms for Eulerian trails, Eulerian Graph. Relat. Top. Part 1 (Ann. Discrete Math.) 2 (50) (1991) 1–13.
[10] J. Heidemann, M. Stojanovic, M. Zorzi, Underwater sensor networks: applications, advances and challenges, Phil. Trans. R. Soc. A 370 (1958) (2012) 158–175.
[11] http://www.cablemap.info/.
[12] http://www.scientificamerican.com/article/deepwater-robot-sub/.
[13] http://www.sonardyne.com/products/monitoring-a-control/autonomous-monitoring-system.html.
[14] T. Hyakudome, T. Aoki, T. Murashima, S. Tsukioka, H. Yoshida, H. Nakajoh, T. Ida, S. Ishibashi, R. Sasamoto, Key technologies for AUV" URASHIMA", in: Proceedings of MTS/IEEE OCEANS, 2002, pp. 162–166.
[15] O.D. Incel, A. Ghosh, B. Krishnamachari, K. Chintalapudi, Fast data collection in tree-based wireless sensor networks, IEEE Trans. Mob. comput. 11 (1) (2012) 86–99.
[16] S. Jain, K. Fall, R. Patra, Routing in a delay tolerant network, in: Proceedings of ACM SIGCOMM, 2004, pp. 145–158.
[17] I. Jawhar, M. Ammar, S. Zhang, J. Wu, N. Mohamed, Ferry-based linear wireless sensor networks, in: Proceedings of IEEE GLOBECOM, 2013b, pp. 304–309.

[18] I. Jawhar, N. Mohamed, J. Al-Jaroodi, S. Zhang, An efficient framework for autonomous underwater vehicle extended sensor networks for pipeline monitoring, in: Proceedings of IEEE ROSE, 2013a, pp. 124–129.
[19] D.J. Klein, S. Venkateswaran, J.T. Isaacs, J. Burman, T. Pham, J. Hespanha, U. Madhow, Localization with sparse acoustic sensor network using UAVs as information-seeking data mules, ACM Trans. Sens. Netw. 9 (3) (2013) 30.
[20] V. Kolmogorov, Blossom V: a new implementation of a minimum cost perfect matching algorithm, Math. Program. Comput. 1 (1) (2009) 43–67.
[21] X.-Y. Liu, Y. Zhu, L. Kong, C. Liu, Y. Gu, A.V. Vasilakos, M.-Y. Wu, CDC: Compressive data collection for wireless sensor networks, IEEE Trans. Parallel Distrib. Syst. 26 (8) (2015) 2188–2197.
[22] M. Ma, Y. Yang, M. Zhao, Tour planning for mobile data-gathering mechanisms in wireless sensor networks, IEEE Trans. Veh. Technol. 62 (4) (2013) 1472–1483.
[23] R. Moazzez-Estanjini, J. Wang, I.C. Paschalidis, Scheduling mobile nodes for cooperative data transport in sensor networks, IEEE/ACM Trans. Netw. 21 (3) (2013) 974–989.
[24] J. Partan, J. Kurose, B.N. Levine, A survey of practical issues in underwater networks, ACM SIGMOBILE Mob. Comput. Commun. Rev. 11 (4) (2007) 23–33.
[25] D. Pompili, T. Melodia, I.F. Akyildiz, Routing algorithms for delay-insensitive and delay-sensitive applications in underwater sensor networks, in: Proceedings of ACM MobiCom, 2006, pp. 298–309.
[26] R. Sugihara, R.K. Gupta, Improving the data delivery latency in sensor networks with controlled mobility, in: Proceedings of IEEE DCOSS, 2008, pp. 386–399.
[27] D.D. Tan, T.T. Le, D.-S. Kim, Distributed cooperative transmission for underwater acoustic sensor networks, in: Proceedings of IEEE WCNC, 2013, pp. 205–210.
[28] O. Tekdas, V. Isler, J.H. Lim, A. Terzis, Using mobile robots to harvest data from sensor fields, IEEE Wirel. Commun. 16 (1) (2009) 22.
[29] L. Tong, Q. Zhao, S. Adireddy, Sensor networks with mobile agents, in: Proceedings of IEEE MILCOM, 2003, pp. 688–693.
[30] J.E. Ullgren, E. André, T. Gammelsrød, A.M. Hoguane, Observations of strong ocean current events offshore Pemba, Northern Mozambique, J. Oper. Oceanogr. 9 (1) (2016) 55–66.
[31] I. Vasilescu, K. Kotay, D. Rus, M. Dunbabin, P. Corke, Data collection, storage, and retrieval with an underwater sensor network, in: Proceedings of ACM SenSys, 2005, pp. 154–165.
[32] J. Wu, H. Zheng, On efficient data collection and event detection with delay minimization in deep sea, in: Proceedings of ACM CHANTS, 2014, pp. 77–80.
[33] P. Xie, J.-H. Cui, L. Lao, VBF: Vector-based forwarding protocol for underwater sensor networks, in: Proceedings of IEEE NETWORKING, 2006, pp. 1216–1221.
[34] X. Xue, X. Hou, B. Tang, R. Bagai, Data preservation in intermittently connected sensor networks with data priority, in: Proceedings of IEEE SECON, 2013, pp. 122–130.
[35] Z. Yang, M. Li, Y. Liu, Sea depth measurement with restricted floating sensors, in: Proceedings of IEEE RTSS, 2007, pp. 469–478.
[36] Y. Yao, Q. Cao, A.V. Vasilakos, EDAL: An energy-efficient, delay-aware, and lifetime-balancing data collection protocol for wireless sensor networks, in: Proceedings of IEEE MASS, 2013, pp. 182–190.
[37] W. Zhao, M. Ammar, E. Zegura, A message ferrying approach for data delivery in sparse mobile ad hoc networks, in: Proceedings of ACM MobiHoc, 2004, pp. 187–198.
[38] W. Zhao, M. Ammar, E. Zegura, Controlling the mobility of multiple data transport ferries in a delay-tolerant network, in: Proceedings of IEEE INFOCOM, 2005, pp. 1407–1418.
[39] H. Zheng, J. Wu, Data collection and event detection in the deep sea with delay minimization, in: Proceedings of IEEE SECON, 2015, pp. 1–9.



Huanyang Zheng received his B.Eng. degree in Telecommunication Engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2012. He is currently a Ph.D. candidate in the Department of Computer and Information Sciences, Temple University, Philadelphia, Pennsylvania, USA. His current research focuses on mobile networks, social networks, and cloud systems.

Ning Wang received his B.Eng. in Electrical Engineering from University of Electronic Science and Technology of China, Chengdu, China in 2013. He is currently a Ph.D. candidate in the Department of Computer and Information Sciences, Temple University, Philadelphia, PA, US. His current research focuses on delay tolerant networks, vehicular networks, and content delivery networks.

Jie Wu is the chair and a Laura H. Carnell professor in the Department of Computer and Information Sciences at Temple University. He is also an Intellectual Ventures endowed visiting chair professor at the National Laboratory for Information Science and Technology, Tsinghua University. Prior to joining Temple University, he was a program director at the National Science Foundation and was a Distinguished Professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Service Computing and the Journal of Parallel and Distributed Computing. Dr. Wu was general co-chair/chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, and ACM MobiHoc 2014, as well as program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a CCF Distinguished Speaker and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.