ACO-A*: Ant Colony Optimization Plus A* for 3-D Traveling in Environments With Dense Obstacles

Xue Yu, *Student Member, IEEE*, Wei-Neng Chen, *Senior Member, IEEE*, Tianlong Gu, Huaqiang Yuan, Huaxiang Zhang, and Jun Zhang, *Fellow, IEEE*

***Abstract*—Path planning is one of the most important problems in the development of autonomous underwater vehicles (AUVs). In some common AUV missions, e.g., wreckage search for rescue, an AUV is often required to traverse multiple targets in a complex environment with dense obstacles. In such case, the AUV path planning problem becomes even more challenging. In order to address the problem, this paper develops a two-layer algorithm, namely ACO-A*, by combining the ant colony optimization (ACO) with the A* search. Once a mission with a set of arbitrary targets is assigned, ACO is responsible to determine the traveling order of targets. But, prior to ACO, a cost graph indicating the necessary traveling costs among targets must be quickly established to facilitate traveling order evaluation. For this purpose, a coarse-grained modeling with a representative-based estimation (RBE) strategy is proposed. Following the order obtained by ACO, targets will be traversed one by one and the pairwise path planning to reach each target can be performed during vehicle driving. To deal with the dense obstacles, A* is adopted to plan paths based on a fine-grained modeling and an admissible heuristic function is designed for A* to guarantee its optimality. Experiments on both synthetic and realistic scenarios have been designed to validate the efficiency of the proposed ACO-A*, as well as the effectiveness of RBE and the necessity of A*.**

***Index Terms*—A* search, ant colony optimization (ACO), autonomous underwater vehicles (AUVs), dense obstacles, path planning.**

Manuscript received November 17, 2017; revised April 23, 2018 and August 14, 2018; accepted October 17, 2018. Date of publication October 26, 2018; date of current version July 30, 2019. This work was supported in part by the National Natural Science Foundation of China under Grant 61622206, Grant 61332002, and Grant 61873097, and in part by the Natural Science Foundation of Guangdong under Grant 2015A030306024. *(Corresponding authors: Wei-Neng Chen; Jun Zhang.)*

X. Yu is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China, and also with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China.

W.-N. Chen and J. Zhang are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China (e-mail: cwnraul634@aliyun.com; junzhang@ieee.org).

T. Gu is with the School of Computer Science and Engineering, Guilin University of Electronic Technology, Guilin 541004, China.

H. Yuan is with the School of Computer Science and Network Security, Dongguan University of Technology, Dongguan 523808, China.

H. Zhang is with the School of Information Science and Engineering, Shandong Normal University, Jinan 250014, China.

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the author.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEVC.2018.2878221

I. Introduction

The AUTONOMOUS underwater vehicle (AUV) is a robotic device equipped with an onboard computer, a propulsion system, a series of sensors and other units if necessary so that it is able to work autonomously in dangerous and complex underwater environments [1]. More and more efforts have been devoted to AUV-related researches focusing on sectors, such as energy, navigation, sensors, autonomy, and communication. Particularly, the AUV path planning [2] has always been an essential and challenging topic for AUV navigation and autonomy [3]. It aims to plan an optimal feasible path to reach one or more given target positions, i.e., targets, in terms of the optimization objectives or requirements, such as the operation cost minimization, risk minimization, and collision avoidance.

To meet the challenges of AUV path planning, many research efforts have been devoted to this field as detailed in Section II. However, most existing works [3] are merely aimed at the pairwise path planning from a position to a single target. Only a few of them have considered the mission requirements. Thus, in this paper, we focus on a common class of target traveling missions, e.g., air crash investigation and mine reconnaissance, which have the same requirement to traverse a number of targets orderly. For path planning of such missions, before planning the detailed path to reach each target, an algorithm must first determine the traveling order of all targets. However, the traveling order determination problem is quite challenging since it indeed belongs to the NP-hard permutation-based problems [4], [5]. Moreover, extra efforts must be devoted to build a target-to-target cost graph in preparation of traveling order evaluation and determination. To obtain the cost graph, it is intuitive to perform pairwise path planning between each pair of targets, but the necessary computing cost will be expensive. Instead, it should be more reasonable to design a sophisticated strategy for quick and accurate cost estimation.

Furthermore, in the literature, there are few AUV path planning algorithms specially designed for the very complex underwater environments with dense and irregular obstacles, e.g., wreck remains and reef, as discussed in Section II. However, such obstacles must be considered when AUVs are to conduct some common target traveling missions, such as wreckage search for rescue, and submarine exploration in reef zones. The dense and irregular obstacles bring new challenges to path planning. First, due to the ubiquitous obstacles, it is difficult to find feasible path segments to form a feasible path. Second, for many irregular obstacles, such as coral reefs, it is hard to find an appropriate model for each of them, but a good modeling for obstacles is essential to path evaluation and path planning.

In this paper, for path planning of target traveling missions in complex underwater environments with dense obstacles, we propose an ant colony optimization (ACO) algorithm armed with the A* search, namely ACO-A*, based on a two-layer environment modeling, i.e., the fine-grained modeling plus the coarse-grained modeling. As a result, the problem is solved from two respects: 1) the traveling order determination for targets and 2) the pairwise path planning to reach each target.

First, for traveling order determination, ACO is adopted due to its popularity and efficiency [6]–[8] for permutation-based optimization [9]. To accelerate the solution evaluation in ACO, a representative-based estimation (RBE) strategy is developed to quickly build a target-to-target cost graph based on the coarse grained modeling. In the strategy, a sufficient number of well distributed points, i.e., representatives, are located in the environment and then the optimal traveling costs among them are calculated to build a representative cost map. Both feasibility and connectivity have been considered during the representative location such that the resultant representatives are capable of representing the whole environment, in which each point may become a mission target. As a result, given a mission with a random set of targets, there is no need to compute the exact traveling costs among these targets. Instead, the best representative is selected for each target and then the optimal traveling costs among targets can be quickly estimated using the traveling costs among their representatives. Note that the representative cost map needs to be built only once for a specific environment and the map is reusable for different AUV missions.

Second, for pairwise path planning, considering the challenges brought by the dense obstacles, a fine-grained modeling is developed to decompose a 3-D environment into cubes and each cube is assigned a certain cube value for the convenience of path evaluation. Particularly, the cube value is set to be infinity or 0 if the cube is obstructive or safe; otherwise, it is set by the risk cost or intensity at the cube center. Based on the cube-based modeling, A* is performed to plan the detailed path to reach each target in the traveling order determined by ACO. An admissible heuristic function is carefully designed to guarantee the optimality of A*. As a result, the algorithm is always able to find the best possible path following the traveling order obtained by ACO.

Overall, the two-layer modeling is performed only when the working environment changes. The coarse-grained modeling locates representatives and computes the traveling costs among these representatives to build a representative cost map. The fine-grained modeling decomposes the feasible regions into cubes and assigns them cube values for subsequent path evaluation. Based on the two-layer modeling, ACO-A* can be invoked to plan a path for an arbitrary set of mission targets. The algorithm first selects a representative for each target and then builds an estimated cost graph of targets. Using the cost graph, ACO is performed to obtain an appropriate traveling order of all targets. In this order, A* is adopted to plan the detailed path to reach each target. In reality, once the path toward the first target has been planned, an AUV can start off for its mission since the path planning for subsequent targets can be performed during vehicle driving.

The rest of this paper is organized as follows. Section II reviews the related works. Section III gives the algorithm background, and the problem is stated in Section IV. Section V shows the environment modeling and Section VI presents the proposed algorithm. Experiments on synthetic and realistic scenarios are detailed in Sections VII and VIII. The conclusion is drawn in Section IX.

II. Related Works

A. *Path Planning Algorithms*

Up to now, there has been a great development in the field of autonomous path planning. Existing path planning algorithms can be roughly classified into two types, *the deterministic* ones and *the nondeterministic* ones. There are popular deterministic algorithms, such as fast marching [2], mixed integer linear programming [10], A* search algorithm [11], and A*-based dynamic algorithms, e.g., sparse A* search [12]. These algorithms are often effective to find optimal solutions on the premise of a good environment modeling. But they tend to be time-consuming when dealing with a large problem space since most of them are based on exhaustive space search [13]. As for nondeterministic algorithms, the evolutionary or swarm-based algorithms (EAs or SAs) [14], [15], e.g., genetic algorithm (GA) [16], differential evolution (DE) [17], [18], ACO [19], particle swarm optimization (PSO) [13], [16], predator-prey optimization [20], [21], are especially popular for path planning due to their ability to handle various constraints and multiple optimization objectives [22]. Generally, these algorithms plan paths by locating and connecting a fixed number of waypoints. Thus, their time consumption will not greatly increase as the problem space increases [13].

B. *AUV Path Planning Under Dense Obstacles*

In reality, AUVs often need to operate in complex ocean environments, e.g., underwater battlefields, which often bring up various difficulties, especially the commonly seen dense obstacles, e.g., coral reefs. However, the path planning problem will become much more challenging if dense obstacles must be considered. In such case, many recently developed algorithms, especially the nondeterministic algorithms [16]–[18], [22], such as EAs [15], [22] and SAs [13], [17], tend to become incapable since they are only designed for path planning in aerial or ground environments with a wide range of free space. Generally, these algorithms generate a path by locating a number of waypoints in the environment and then connecting adjacent waypoints by direct path segments. But the path is very likely to be infeasible if there are dense obstacles, since a random path segment indeed has a great chance to intersect with the dense obstacles. As exemplified in Fig. S1 in the supplementary material, if obstacles are added in Fig. S1(b) in the supplementary material, several path segments are found to intersect with the obstacles and the three originally feasible paths in Fig. S1(a) in the supplementary material all become infeasible.

Thus, recently, it has attracted increasing research interests to develop specialized AUV path planning considering obstacle avoidance. A survey of path planning for AUVs can be found in [3]. Petillot *et al.* [23] modeled obstacles in ellipses and then adopted numerical programming techniques for path planning. Braginsky and Guterman [24] developed a two-layer obstacle avoidance algorithm by combining the preplanning technique with the reactive method, in which obstacles are modeled in rectangles. Aghababa [25] formalized the 3-D path planning with static obstacles as a nonlinear optimal control problem and applied five EAs to solve the problem. Zhang *et al.* [26] improved the wolf pack algorithm to solve the 3-D underwater path planning considering terrain obstacles in the peak shape. Yan *et al.* [27] classified irregular obstacles into four types and accordingly designed obstacle avoidance rules.

However, despite the rich achievements as reviewed above, some deficiencies still exist. On the one hand, some works merely consider 2-D environments [23], [24]; on the other hand, many algorithms are designed on the hypothesis that the obstacles are of a small number and in a certain regular shape [26], but the hypothesis actually not always holds in reality. Even among the few works designed for irregular obstacles, most of them such as [27] lack a versatility due to the use of some sophisticated classification strategies. Thus, there is still a great need to develop AUV path planning considering the very complex 3-D environments with dense and irregular obstacles.

C. AUV Path Planning Under Mission Requirements

In the literature, many efforts have been devoted to designing more efficient algorithms for pairwise path planning. These algorithms often ignore the specific mission requirements. However, in reality, different AUV missions usually have different requirements for path planning.

Recently, researchers have gradually realized the importance of mission requirements to path planning. Zhu *et al.* [28] have achieved path planning based on an improved self-organizing map for a multi-AUV system with the requirement of dynamic task assignment. Englot and Hover [29] accounted for the inspection coverage need of a ship hull and developed a comprehensive method for sampling-based design of inspection routes. Also, in [30], both path planning and replanning are designed to satisfy the inspection coverage requirement of AUV missions.

From the above, we find that existing mission planning often focus on a class of missions with the same requirement, such as task assignment and inspection coverage, so as to design an algorithm with higher versatility. Thus, in this paper, we also concentrate on a class of commonly seen AUV missions, e.g., air crash investigation, with the same requirement for target traveling. That is, an AUV needs to visit a set of targets instead of a single target during a voyage. It is encouraging to find that such target traveling missions have attracted some attentions. For example, Han *et al.* [31] developed an ethology-based hybrid control architecture for an AUV to travel to multiple target positions; Mcmahon and Plaku [32] focused on motion planning for such AUV missions in spatially and temporally environments. But few works have been specially designed for target traveling in complex environments with dense obstacles. In such case, the traveling order of targets must be first determined before the detailed path planning to reach each target. For preparation of traveling order determination, the necessary traveling costs among targets should be obtained to build a target-to-target cost graph so as to facilitate the evaluation of traveling orders. However, many time-efficient path planning algorithms are indeed unavailable in dense obstacles as mentioned above. Thus, more efficient strategies must be developed for quick cost graph building.

III. Algorithm Background

A. A* Search Algorithm

A* is a best-first search algorithm [33] to find an optimal path from a start node to a target node in terms of the given cost function. A* is developed by importing a heuristic function into the Dijkstra's algorithm so as to improve the computational efficiency, especially for pairwise path planning.

The search of A* is graph-based. Given a graph $G = (V, E)$, in order to find the path from start node s to target node t, A* starts at s to search the graph until the target node t is reached. A* maintains a priority queue Q of candidate nodes to be searched. The priority of each node x is inversely proportional to

$$f(x) = g(x) + h(x).$$

In fact, $f(x)$ estimates the cost of an optimal path from s to t through x, where $g(x)$ is the best-so-far cost from s to x, $h(x)$ is the heuristic function to estimate the cost from x to t. A node with a smaller f-value is thought to have a higher probability or priority to occur in an optimal path from s to t. Thus, by always choosing the node with minimal f-value from Q, the graph search is directed toward the target.

The optimality of A* can be guaranteed by the admissibility of heuristic function as Definition 1. If the heuristic function is *admissible*, A* must be able to find an optimal path; otherwise, A* may only find a suboptimal path. Thus, to apply A* to solve a specific problem, it should be important to design an appropriate heuristic function corresponding to the problem objective function.

The general framework of the A* algorithm for pairwise path planning from s to t can be described as follows.

Step 1 (Initialization): Set $f(i)$ as positive infinity for each possible node i; $f(s) = 0$; initialize "open list" Q using s.

Step 2 (Waypoint Selection): Node n with minimal f-value in Q is selected as the next waypoint; delete n from Q.

Step 3 (Stop Criteria): Algorithm stops if the destination t is reached in step 2.

Step 4 (Search Extension): For each node m adjacent to n by edge (n,m) in the graph, if $g(m) > g(n)+\text{cost}(n, m)$, update $g(m) = g(n) + \text{cost}(n, m)$, set $f(m) = g(m) + h(m)$, add m into Q if it is not in Q; then, go to step 2.

Definition 1 (Admissibility): Assume $h^*(x)$ is the truly optimal cost from x to t. If $h(x) \leq h{*}(x)$ holds for every x, function $h(x)$ is said to be *admissible*.

B. Ant Colony Optimization

ACO is a class of nature-inspired algorithms for solving combinatorial optimization problems [34]. There are popular ACO variants, such as the ant colony system (ACS) [6] and the elitist ant system [35]. The efficiency of ACO was originally verified on the traveling salesman problem. Up till now, ACO has been successfully applied to solve various problems, e.g., data mining [36] and project scheduling [37].

ACO is mainly characterized by its pheromone model and probabilistic solution construction. In addition, a local search procedure is often utilized for solution refinement. Generally, ACO solves a problem by performing the following steps repeatedly.

1) *Solution Construction:* Candidate solutions are constructed according to a parameterized probability distribution, which is composed by heuristic information and pheromone values.
2) *Pheromone Update:* Pheromone values are updated using the previously constructed solutions such that they can direct the future evolution toward the high-quality solution space.

To solve a specific problem using ACO, the solution construction process and pheromone model should be carefully designed in terms of the specific problem characteristics, such as decision variables and solution structures.

IV. Problem Statement

A. Target Traveling Problem

In this paper, we focus on the target traveling problem caused by a class of target traveling missions [28], [38], such as air crash investigation, with the requirement to traverse a number of targets in underwater environments. Such missions are quite common especially when the environment is so wide and complex that an AUV often only has time to traverse and examine the several most suspicious positions, i.e., targets. The problem mainly includes the following aspects.

AUV: To achieve autonomy, AUVs are typically equipped with sensors, controllers, propellers, on-board computers, and so on. Since there are more communication difficulties within water than air, AUVs usually follow a preprogrammed course with robust navigation [39] and path following [40].

Environment: The problem considers the complex underwater environments with rough bathymetrics, risky areas, and dense obstacles of various irregular shapes [41]. During mission execution, AUVs should have no collision with obstacles and try to keep away from the risky areas. It is worth noting that in reality, there are often not sufficient computing resources to handle a very large underwater scenario, e.g., the wide ocean space. Thus, like most existing works [28], [38], the path planning problem must be solved in a local environment of a limited size instead of the whole underwater scenario. However, in a complex local environment with dense obstacles, two random targets may be disconnected and no feasible path exists between them. For ease of description, the *feasibility* and *connectivity* of point(s) and path(s) are defined as Definitions 2 and 3.

AUV Target Traveling: A typical mission consists of a set of targets to be traversed by an AUV. That is, the AUV needs to be released from a certain target to visit the other targets one by one, and finally return to the start position to be recovered. The AUV path planner is expected to find an optimal path to traverse these targets in a certain order. The optimization objective should consider collision, risk, as well as cost, which will be detailed in the following section.

AUV Path: For AUV path generation, one simple but popular category of existing methodologies is based on straight lines [3], which is also adopted in this paper. In this method, to generate a path, an ordered set of intermediate waypoints is located in the environment such that each direct path segment connecting two adjacent waypoints is feasible. The resultant path is indeed a polygonal line formed by these path segments.

Notably, in reality, it is quite common to release AUVs into the same underwater environment to perform different missions with different sets of targets. Therefore, a generic algorithm for the target traveling problem should not highly depend on the target information that will change according to AUV missions. Instead, the algorithm is expected to extract more environment information that will be useful for the path planning of a set of arbitrary targets. Moreover, since an operator may carelessly design an unreasonable mission with disconnected targets, an algorithm should be able to quickly identify the disconnected targets and only plan paths for connected targets.

Definition 2 (Feasible and Feasibility): A point in the environment is said to be *feasible* if it is above the bathymetry and has no collision with any obstacle. A *feasible path* only passes through feasible points. A *feasible region* only consists of feasible points.

Definition 3 (Connected and Connectivity): A feasible point is *connected* to itself. Two feasible points are said to be *connected* if there is a feasible path between them. A feasible region is said to be *connected* if any point within the region is connected to the others, i.e., *connected region*.

B. Path Optimization Objective

To plan paths in complex environments with obstacles and risky areas, the optimization objective should at least consider three requirements: 1) collision avoidance; 2) risk minimization; and 3) AUV operation cost minimization.

Collision Avoidance: The collision avoidance is viewed as a hard constraint. That is, an AUV path is infeasible once it has an intersection with any obstacle in the environment.

Risk Minimization: In this paper, we consider global risky areas with descending risk, which are quite common in reality, e.g., areas within the range of sonar detection. Once an AUV enters a risky globe, it is at risk and the risk increases when the AUV approaches the globe center. Consider the rth risky area in the environment with area center pr, the maximal risk intensity RI, and the maximal risk radius R. The *risk intensity* of a random point p caused by this risky area is inversely proportional to its distance to pr, which is defined as

$$\text{risk}(p, r) = \text{RI} * \max\{0, (R - \text{distance}(p, pr))\}/R. \quad (1)$$

Operation Cost Minimization: Referring to existing works such as [42], the AUV operation cost considers traveling length cost, turning cost, and height cost. The height cost, also denoted as climbing/diving cost, is defined by the absolute heights of path segments. The turning cost is defined by the turning angles between adjacent path segments.

To optimize more than one cost as aforementioned, there are two popular approaches: 1) optimize the multiple costs simultaneously using specialized multiobjective algorithms and 2) integrate the costs into a single optimization objective function using techniques, such as a weighted sum of the objectives [43], and then optimize it by single-objective algorithms. In the field of path planning, weight-based single optimization is quite popular, and easy to use [16], [26], [44]–[46]. For the setting of cost weights, previous works generally believe that the weights can be reasonably set according to some professional knowledge, e.g., the specific task type or priority [45] or the user preference [46]. Notably, such professional knowledge is often also necessary in multiobjective optimization. For example, in [22], a multiobjective EA is designed for multi-UAV path planning but prior knowledge is required to preset the priority level and the acceptable interval of each objective for solution evaluation and the final solution selection; in [47], after a path set is obtained by the multiobjective path planning, a weight vector is needed to show the user preference so as to select the ideal path with the minimum preferred value.

Thus, in this paper, the weighted approach is also adopted. In particular, four cost weights, i.e., $\alpha_1, \alpha_2, \alpha_3, \alpha_4$, are introduced for risk cost, length cost, height cost, and turning cost, respectively. Thereby, the traveling cost of a path can be defined by the weighted summation of these four kinds of cost.

Mathematically, given a path from start point st to end point ed with n waypoints, i.e., $\text{path}(st, ed) = (x_0, \ldots, x_i, \ldots, x_n, x_{n+1})$ with $x_0 = st$ and $x_{n+1} = ed$, the traveling cost of this path, i.e., path cost, can be computed using

$$\cos t(st, ed) = \sum_{i=0}^{n}(lc(x_i, x_{i+1}) + hc(x_i, x_{i+1})) + \sum_{i=1}^{n}(rc(x_i) + tc(x_{i-1}, x_i, x_{i+1})) \tag{2}$$

where $lc(x_i, x_{i+1})$ gets the length cost of the path segment from x_i to x_{i+1} as

$$lc(x_i, x_{i+1}) = \alpha_2 \cdot |x_i - x_{i+1}| \tag{3}$$

$hc(x_i, x_{i+1})$ gets the height cost of this path segment as

$$hc(x_i, x_{i+1}) = \alpha_3 \cdot |Z(x_i) - Z(x_{i+1})|. \tag{4}$$

$Z(x_i)$ is Z-coordinate value of point x_i, $rc(x_i)$ computes the risk intensity at point x_i caused by the total rn risky areas as

$$rc(x_i) = \alpha_1 \cdot \sum_{r=1}^{rn} \text{risk}(x_i, r) \tag{5}$$

and $tc(x_{i-1}, x_i, x_{i+1})$ obtains the turning cost between the two path segments that join at x_i as

$$tc(x_{i-1}, x_i, x_{i+1}) = \alpha_4 \cdot \left(1 - \frac{\boldsymbol{p}_i \cdot \boldsymbol{p}_{i+1}}{|\boldsymbol{p}_i| \cdot |\boldsymbol{p}_{i+1}|}\right), \quad \begin{cases} \boldsymbol{p}_i = x_i - x_{i-1} \\ \boldsymbol{p}_{i+1} = x_{i+1} - x_i. \end{cases} \tag{6}$$

V. Two-Layer Modeling

To model the environment in preparation for path planning of target traveling missions, a two-layer environment modeling is proposed consisting of a fine-grained modeling and a coarse-grained modeling. The fine-grained modeling helps to build a search graph for pairwise path planning, and the coarse-grained modeling defines a RBE strategy so as to quickly build an estimated cost graph of targets before target order determination. To facilitate understanding, we have given an exemplification of the two-layer modeling in Fig. S2 in the supplementary material, where the fine-grained modeling is shown in black thin lines and the coarse-grained modeling is shown in blue thick lines. Also, an explanation of the key symbols used in this paper is provided in Table SI in the supplementary material.

A. Representation (Environment and Path)

To prepare for the following modeling, we first represent the environment, the AUV, and the path in a coordinate system, as done in some existing works of AUV path planning [2], [3], [28].

Environment Coordinate System: Given a rectangular 3-D environment Ω with (length, width, and height) $= (l, w, h)$, a 3-D coordinate system will be established where the x-axis, y-axis, and z-axis correspond to the length, width, and height, respectively. Then, each position in the environment uniquely corresponds to a point (x, y, z) in the system satisfying

$$x \in [0, l], y \in [0, w], z \in [0, h].$$

AUV Point: Since the AUV size is often ignorable compared with the environment size, an AUV is often simplified as a point that flies in the coordinate system [2].

Target Point: As some existing works on target-search path planning [28], [38], we also use points in the coordinate system to simplify the suspicious or candidate targets to be traversed.

AUV Path: Based on the above abstractions, to plan a path for a given set of m target points $T = \{t_1, \ldots, t_m\}$, an algorithm should first determine a target order or permutation P that starts from some target P_1 and finally return P_1 after traversing each target once and only once, defined as

$$P = (P_1, \ldots, P_m, P_1), P_i = P_j \leftrightarrow i = j, P_i \in T. \tag{7}$$

Then, between each pair of successive targets in P, an ordered set of intermediate waypoints should be located to constitute a complete path, i.e., solution, as follows:

$$\text{path} = \Big(P_1, \ldots, W_1^j, \ldots, P_2, \ldots, P_i, \ldots, W_i^j, \ldots P_{i+1}, \ldots, P_m, \ldots, W_m^j, \ldots, P_1\Big) \tag{8}$$

Algorithm 1 (cubeConnectivityRecord)

Input: cubes obtained by cube-based decomposition
Output: connectivity tag of each cube, i.e., *connect*[]

```
1.  $tag = 0$;
2.  While there is an unvisited feasible cube $sc$
3.      $tag += 1$;//$tag$ of a new connected region
4.      put $sc$ into an empty queue $q$;
5.      While $q$ is not empty
6.          $cur = q$.front();//get the front cube in $q$
7.          $connect[cur] = tag$;//mark cube using $tag$
8.          For each unvisited feasible cube $i$ adjacent to $cur$
9.              put cube $i$ into $q$;
10.         End For
11.         flag $cur$ as visited and delete it from $q$;
12.     End While
13. End While
```

Algorithm 2 (representativeLocation)

Input: cubes of a block, connected region tag *tag*
Output: representative point *pr* of the region

```
1. $fcB = 0$; $pr =$ Null;
2. For each unvisited feasible block facet cube $i$, $connect[i] == tag$
3.     $(p,fc) =$ connectedTraverse($i$);//Algorithm 3
4.     If $fc > fcB$
5.         $pr = p$; $fcB = fc$;
6.     End If
7. End For
```

Algorithm 3 (connectedTraverse)

Input: start cube *sc*
Output: region center point *p* and connectivity metric *fc*

```
1.  put start cube $sc$ in queue $q$
2.  $Ncube = Nface = 0$;//number of feasible cubes and facets
3.  $p =$ the center point of $sc$;
4.  While $q$ is not empty
5.      $cur$=$q$.front();//get the front cube of the queue
6.      For each unvisited feasible cube $i$ around $cur$ in block
7.          $q$.push(cube $i$);//add this cube into the queue
8.          $Ncube += 1$;//a new feasible cube
9.          If cube $i$ is in a unvisited block facet
10.             $Nface += 1$; //a new feasible facet
11.             flag the facet as visited;
12.         End If
13.         If cube $i$'s center is closer to the block center than $p$
14.             $p =$ the center point of cube $i$;
15.         End If
16.     End For
17.     flag cube $cur$ as visited;
18.     $q$.pop();//delete the front element of the queue
19. End While
20. calculate $fc$ using $Ncube$ and $Nface$;
```

where W_i^j is the jth waypoint of the *subpath* from target P_i to target $P_{i\%m+1}$, and each *path segment* that connects two adjacent points in *path* must be feasible.

Definition 4 (Cube Feasibility): A cube is said to be feasible if each point within the cube is feasible.

Definition 5 (Cube Connectivity): A feasible cube is *connected*. Two cubes are *connected* if they are both feasible and their center points are connected. The *connectivity* of a cube is determined by the number of cubes that are connected to this cube. The *connectivity* of a point equals the *connectivity* of the cube containing the point.

Lemma 1 (Connectivity Affirmation): Two points must be connected if they exist in connected cube(s).

B. Fine-Grained Modeling

Given an environment, to find a path for a set of mission targets using A*, the environment should be first transformed into a search graph to indicate all candidate waypoints and the adjacent relationship among these waypoints. Moreover, considering the possible existence of disconnected targets as mentioned in Section IV, a strategy should be designed to quickly examine the connectivity of targets since it is meaningless to plan paths for disconnected targets.

To this end, we adopt a cube-based decomposition to help build the search graph for preparation of A*. Also, a fast connectivity examination strategy is designed to quickly obtain the targets' connectivity from the cubes' connectivity. The cube-based feasibility and connectivity have been defined in Definitions 4 and 5, respectively.

Specifically, a rectangular environment is decomposed into exclusive cubes so that each point exists in one and only one *cube*. It is worth noting that we are actually using "cube" to denote "cuboid." That is, the cube size is determined by three parameters, i.e., length L, width W, and height H, which are often related to the environment size. After decomposition, each cube with a center point $p = (x, y, z)$ has a unique triple index (i, j, k) where i, j, k are integers, and they satisfy

$$x = (2i + 1)L, y = (2j + 1)W, z = (2k + 1)H.$$

Supposing there are at most N cubes along each axis direction, each triple index corresponds to a unique index $(ixN^2 + jxN + k)$.

To model an environment with dense obstacles, the cube-based decomposition should be performed in a quite fine-grained scheme to obtain more feasible cubes and hence to increase the probability of finding a feasible path. However, as the cube size decreases, the search graph will become larger and hence the search complexity will increase. Thus, an appropriate cube size is expected to be set in consideration of realistic factors, such as environment size, density of obstacles, and available computing resources.

1) Search Graph Building: To build a search graph $G = (V, E)$ based on the above decomposition, the center point of each feasible cube is viewed as a candidate waypoint for path planning. Two waypoints are said to be adjacent only when their corresponding cubes are adjacent. As a result, the vertex set V is composed by the center points of all feasible cubes, and an edge is created for each pair of adjacent vertexes to form the edge set E.

Furthermore, for the convenience of path evaluation, cubes are assigned different cube values so as to distinguish the cubes in safe zones, risky zones, and obstructive zones, i.e., safe cubes, risky cubes, and obstructive cubes, as illustrated in Fig. S2 in the supplementary material. Specifically, if a cube intersects with obstructive areas, it is an obstructive cube; otherwise, if the cube intersects with risky areas, it is a risky cube; otherwise, the cube is a safe cube. The cube value of a safe cube is set as 0. The cube value of an obstructive cube is set as positive infinity such that a path planning algorithm will try to avoid the cube. The cube value of a risky cube is set by the risk cost at the cube's center point. Thereby, the risk cost of a path segment from x_i to x_{i+1} in (5) can be simply computed by the summation of the involved cube values as follows:

$$rc(x_i, x_{i+1}) = \sum_{b \in S} cv(b) \tag{9}$$

where $cv(b)$ is the cube value of cube b and S is the set of cubes that intersect with the path segment.

2) Fast Connectivity Examination: Based on the cube-based definitions, the point connectivity can be implied by the cube connectivity as Lemma 1. That is, two points existing in one or two connected cubes must be connected. Lemma 1 can be proven as follows.

1) Two points within a connected cube must be connected by the direct line connecting them. The line is feasible since each point in a connected, i.e., feasible, cube must be feasible (by Definition 4).
2) If points a and b exist in two connected cubes whose center points are c_a and c_b, we can construct a feasible path between a and b using the direct line connecting a and c_a, the direct line connecting c_b and b, and the feasible path between c_a and c_b (by Definition 5). Thus, two points existing in two connected cubes must be connected.

In this paper, we further constrain Lemma 1 so as to design a fast point connectivity examination strategy based on the cube connectivity. That is, two points are viewed as connected only if the cube(s) containing the points is/are connected. As a result, whenever to check the connectivity of two arbitrary targets, we can quickly query the connectivity of the cube(s) containing the two targets.

To obtain the cube connectivity, we traverse the cubes and record their connectivity during the fine-grained modeling as Algorithm 1. In each loop of Algorithm 1, an exhaustive search is started at an unvisited feasible cube *sc* to traverse all cubes that are connected to *sc*, and these cubes as well as *sc* are marked by an identical *tag* value. In this way, two feasible cubes are connected if and only if they have the same *tag* value. Indeed, all cubes with the same *tag* value constitute a maximal connected region (MCR) as Definition 6. The maximal *tag* value will be equal to the number of different MCRs in the environment.

Definition 6 (Maximal Connected Region): An MCR is a connected region formed by cubes such that each cube outside the region is not connected to any cube in the region.

Definition 7 (Local Connectivity): Two points in a region are *locally connected* if they are connected by a feasible path that entirely exists in the region. Two cubes in a region are locally connected if their centers are locally connected. A locally connected region (LCR) is a region in which any two points are locally connected.

C. Coarse-Grained Modeling

In order to quickly build an estimated target-to-target cost graph prior to target order determination, we locate a number of points as representatives in the environment and then calculate the optimal traveling costs among these representatives to build a reusable representative cost map. In this way, given a random set of targets, we can select a representative for each target and then consult the representative cost map for quick cost estimation among targets. However, several issues must be considered during representative location to achieve high estimation accuracy and efficiency. First, considering that there may be more than one MCR in the environment, each MCR must be guaranteed to have a sufficient number of representatives. Second, since each feasible point may become a target in the future, representatives should be evenly distributed in each MCR so as to better represent all feasible points therein. Third, the number of representatives should be within a reasonable range to avoid a heavy computational burden. To address these issues, we further develop a coarse-grained modeling, in which the cubes in fine-grained modeling are merged into larger-size blocks. Then, a block-based approach is carefully designed for representative location.

In the coarse-grained modeling, provided that the block size is B, every B^3 cubes will constitute a large-size cubic *block* such that each cube belongs to one and only one block. In particular, each cube indexed by (i, j, k) belongs to the block labeled by (I, J, K) with

$$I = i/B, J = j/B, K = k/B.$$

A block is *feasibly connected* only if there are at least one feasible cube in the block's facets since a block's connection to other blocks must use the feasible cubes in the block's facets.

For example, as shown in blue thick lines in Fig. S2 in the supplementary material, B is 5, every 125 cubes constitute a block. There are totally two blocks, and they are both feasibly connected.

1) Representative Location: In order to locate a reasonable number of representatives for each MCR in the environment, we locate a representative in each block that intersects with the MCR. As a result, once a block intersects with an MCR, it will contribute a representative to this region. For ease of demonstration, we give a 2-D example of environment modeling in Fig. S3 in the supplementary material, which consists of 36 cubes, four blocks, and two MCRs (tagged by 1 and 2). Since the left bottom block contains cubes in both MCRs, two representatives will be located in the block. In contrast, one representative will be located in the right bottom block since the block contains only cubes with tag = 2.

To locate a representative in a block for an MCR with a certain *tag* value, we first identify all feasible cubes with the same *tag* value in the block, which indeed constitute the common connected region (CCR) of the MCR and the block. Then, a representative will be located in the CCR in consideration of two factors, i.e., centrality and connectivity. Herein, the two factors are considered during representative location for two reasons: 1) a centered representative tends to have a short average distance to all points it represents and 2) a representative of higher connectivity is likely to be able to represent more points since a point must be connected to its representative. Particularly, the local connectivity in the block as defined in

Definition 7 is considered since two points in a CCR may be not locally connected. That is, a CCR may consist of more than one LCR. For example, in the right bottom block in Fig. S3 in the supplementary material, a cube in yellow is not locally connected to a cube in green since a feasible path between them must go through the cube(s) outside the block. To measure the local connectivity of an LCR indexed by R, the connectivity metric fc is defined as

$$fc(R) = \begin{cases} w \cdot \mathrm{FR}_1 + (1-w) \cdot \mathrm{FR}_2, & \text{if } \mathrm{FR}_2 > 0 \\ 0, & \text{otherwise} \end{cases} \quad (10)$$

where parameter w balances FR_1 and FR_2, FR_1 measures the region's cube feasibility ratio to the total number of cubes in a block as follows:

$$\mathrm{FR}_1 = ncube(R)/B^3$$

where $ncube(R)$ gets the total number of cubes in region R, and FR_2 measures the region's facet feasibility ratio to the total number of facets of a block as follows:

$$\mathrm{FR}_2 = nface(R)/6$$

where $nface(R)$ gets the number of block facets that intersect with region R.

Therefore, to locate a representative in the CCR of an MCR and a block, we first identify the LCR with the best connectivity in the CCR and then select the most central cube center in the LCR as a representative. The specific procedure is shown in Algorithm 2. Especially, each LCR is obtained by starting a traversal procedure (Algorithm 3) from an unvisited feasible cube sc in the block facet, and the representative pr is set by the center of the LCR with the largest fc value (lines 4–6). In detail, to obtain the LCR containing cube sc, as shown in Algorithm 3, we perform a breadth-first-search (BFS) to identify all feasible cubes connected to sc. These cubes plus cube sc constitute an LCR. In each iteration of BFS, the search extends to get the queue front cube cur and put all unvisited feasible cubes adjacent to cur into the queue (lines 5–16), and then cube cur is flagged as visited and deleted from the queue. During BFS search, the necessary statistics are kept (lines 8–12) for the final calculation of fc (line 19). Meanwhile, the candidate representative point p is updated to approximate the LCR's center (lines 13–15).

2) Representative Cost Map (R-Map): After representative location, we build a representative cost map for the representatives in each connected region, where the optimal traveling costs among representatives can be obtained by a path planning algorithm. On this basis, given a random set of target nodes, we can quickly find the best representative for each target and estimate the traveling costs among these targets using the costs among their representatives. In this way, an estimated cost graph can be quickly built for a random set of targets by consulting the representative cost map.

VI. ACO-A*

In order to solve the target traveling problem in complex environments, based on the two-layer environment modeling, a two-layer algorithm is proposed by combining ACO and the A* search algorithm, namely ACO-A*. The overall framework of environment modeling and ACO-A* is given in Fig. 1, where the modeling in blue extracts useful environmental information in preparation for ACO-A* in yellow. As shown in Fig. 1, in fine-grained modeling, the environment is modeled to obtain the search graph for A* path planning, and in coarse-grained modeling, representatives, and a representative cost map are obtained to facilitate ACO's traveling order determination. Once the environment modeling is completed, ACO-A* can be invoked by different AUVs to plan paths for sets of arbitrary targets. Given a set of targets, ACO optimizes the traveling order of these targets according to an estimated cost graph of targets, which is built by selecting representatives for targets and consulting the representative cost map. After a target order is determined by ACO, A* is performed for pairwise path planning to reach each target following the order.

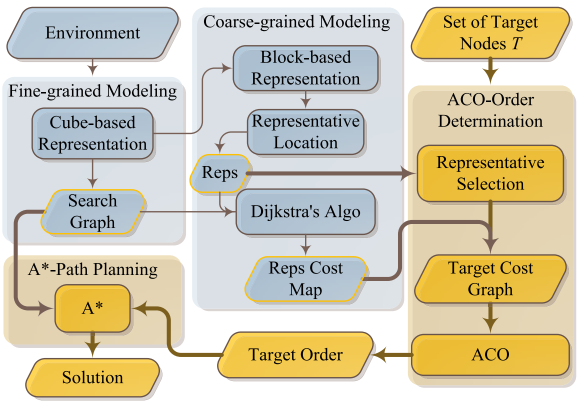

Fig. 1. Overall framework of environment modeling and ACO-A*, where the parallelogram means data, the rectangle means operation or procedure. The blue part is the two-layer environment modeling. The yellow part is the proposed ACO-A*. The yellow blue parallelograms are modeling results, which are important input to ACO-A*.

A. Target Connectivity and Target Grouping

As mentioned in Section IV, in reality, a limited local environment may include more than one connected region, and a mission may include disconnected targets. In such case, the traveling costs among disconnected targets are thought to be much larger than the costs among connected targets. Thus, in this paper, targets are first grouped according to their connectivity such that the disconnected targets are grouped into different sets. Thereafter, ACO-A* is performed to plan a path for each set of connected targets.

Recall that two targets are thought to be connected only if their corresponding cubes are connected. Since the cube connectivity has been obtained in fine-grained modeling, we can query the connectivity of the cubes containing the targets so as to quickly determine the targets' connectivity.

B. ACO for Order Determination

To determine the traveling order of a set of targets a set of targets, we first select a representative for each target, and then utilize the representative cost map to quickly build an estimated cost graph. The cost graph is expected to indicate the necessary traveling costs among targets so as to facilitate the evaluation of an arbitrary traveling order of targets. Based on the cost graph, ACO is performed to find an optimal or suboptimal traveling order. Notably, in this paper, we mainly adopt the specific ACO variant, the ACS [6].

1) Representative Selection: To select a reasonable representative for each target, the following requirements are necessary: 1) the target should be closely connected to its representative and 2) the traveling cost from the target to its representative should be available and sufficiently small. To satisfy the requirements, given a target t, we perform a breath-first-search from t. Once a representative p is reached, it will be selected as target t's representative, and the traveling cost from t to p is also obtained during the search. For efficiency, the search is limited in a depth of double block size ($2*B$). If a representative is not found when the search stops, supposing target t is in block b, we will first identify all representatives that have the same *tag* value as target t in block b and b's adjacent blocks. Thereafter, the traveling cost from each of these representatives to target t is estimated by the direct line cost between them, and then the representative with the minimum traveling cost will be selected. Herein, the *direct line cost* between two points is computed by viewing the direct line segment between them as a feasible path.

2) Cost Graph and Data Structure: Given a set of connected targets, to build an estimated cost graph $G = (V, E)$, each target acts as a vertex in the vertex set V and we generate an edge between any pair of targets to form the edge set E. The edge cost of each edge $e_{i,j}$ from target i to target j is set by a representative-based estimated value of the optimal traveling cost from i to j. Particularly, to estimate the optimal traveling cost from i to j, we construct an estimated optimal path through the two targets' representatives. As shown in Fig. S4 in the supplementary material, to simulate the truly optimal path path($i \to j$) from target i to target j as marked in solid lines, we utilize an estimated optimal path that passes through target i, i's representative r_i, target j's representative r_j, and target j sequentially, i.e., path($i \to r_i \to r_j \to j$) as marked in dotted lines. Since a target is always close to its representative, the estimated path($i \to r_i \to r_j \to j$) from target i to j tends to be quite close to the optimal path($i \to j$). Thus, the traveling cost of path($i \to r_i \to r_j \to j$) is indeed a quite reasonable estimation for the traveling cost of the optimal path($i \to j$), and the estimated cost is hence used to set the cost of edge e_{ij} from i to j in this paper. However, as shown in Fig. S4(b) in the supplementary material, if the length between two targets is shorter than the length between their representatives, an estimated path through the representatives is more likely to take a detour. Thus, we design a length-based ratio rl to reduce the estimated cost in such case so as to better estimate the optimal path cost. Mathematically, the edge cost of e_{ij} is set as

$$|e_{ij}| = \text{cost}(r_i, r_j) + rl * \text{cost}(i, r_i) + rl * \text{cost}(j, r_j) \quad (11)$$

where cost(r_i, r_j) is the optimal traveling cost from r_i to r_j which can be obtained from the representative cost map, cost(i, r_i), and cost(r_j, j) have been obtained during representative selection and they are generally equal to the traveling costs of the optimal path($i \to r_i$) and path($r_j \to j$), respectively. Thus, the sum of these three cost values approximates the optimal traveling cost of path($i \to r_i \to r_j \to j$). The ratio rl is defined as

$$rl = \min\left\{1.0, \text{length}(i, j)/\text{length}(r_i, r_j)\right\} \quad (12)$$

such that rl only takes effect when the length between i and j is smaller than the length between their representatives.

Based on the cost graph of targets, the solution structure, pheromone model, and heuristic model of ACS are defined by the vertex set V and the edge set E. Particularly, a solution is defined as a permutation of all vertexes in V, and each vertex or target hence acts as a candidate solution component. The cost of a solution can be easily obtained by adding the costs of the edges that connect all adjacent vertexes in the solution. As for pheromone model and heuristic model, each edge $e_{i,j}$ in E corresponds to a pheromone trail parameter τ_i^j and a heuristic parameter η_i^j. For heuristic setting, each heuristic parameter$\eta_i^j$is defined by the inverse of the edge cost of $e_{i,j}$ referring to [6].

3) Solution Construction: To construct a solution in ACS, targets are selected one by one to obtain a permutation of targets, which indicates a target traveling order. The construction process starts with a partial solution sol = $<s_0>$ where the first target s_0 is selected randomly. Then, the solution is completed step by step. In each ith step, an unvisited target s_{i+1} is selected after the current target s_i. Specifically, all unvisited targets together constitute the set of candidate nodes C_i. Then, the probability of each target j in C_i to be selected as s_{i+1} is calculated combining pheromone values and heuristic values according to

$$P(j) = \frac{\left(\tau_{s_i}^j\right)^\alpha \left(\eta_{s_i}^j\right)^\beta}{\sum_{j \in C_i} \left(\tau_{s_i}^j\right)^\alpha \left(\eta_{s_i}^j\right)^\beta}, \forall j \in C_i \quad (13)$$

where α and β are parameters that determine the relative importance of pheromone values versus heuristic values.

To select a target out of C_i as s_{i+1}, a random real number q within [0, 1] is generated and compared with a preset parameter q_0. If q is smaller than q_0, the target with the largest probability value in C_i will be selected; otherwise, Roulette wheel selection according to above probability distribution is applied.

4) Pheromone Initialization and Update: The pheromone initialization is often essential to the ACS's performance. In fact, given a cost graph of targets, the problem of traveling order determination closely resembles the classic traveling salesman problem, for which ACS [6] is originally proposed. Thus, referring to [6], we also utilize a solution sol generated by a greedy method for pheromone initialization. To generate sol, the greedy method starts from a random target s_0 and then always selects an unvisited target which has the least edge cost from the last selected target s_i so as to form a target traveling order. Then, each pheromone trail parameter is initialized by the cost of solution sol and the number of targets m as follows:

$$\tau_i^j = \tau_0 = (\text{cost(sol)} \cdot m)^{-1}. \quad (14)$$

The pheromone update is performed globally and locally in ACS. The global update is performed after each iteration of ACS according to

$$\tau_i^j = (1-\rho)\cdot\tau_i^j + \rho\cdot\Delta\tau_i^j \tag{15}$$

where ρ is a pheromone decay parameter within [0, 1]

$$\Delta\tau_i^j = \begin{cases} (\text{cost}(b\text{sol}))^{-1}, & \text{if } i \text{ is followed by } j \text{ in } b\text{sol} \\ 0, & \text{otherwise} \end{cases}$$

cost($b$sol) is the traveling cost of the best-so-far solution $b$sol.

The local update is performed after the solution construction of each ant k as follows:

$$\tau_i^j = (1-\rho')\cdot\tau_i^j + (\rho')\cdot\Delta\ \tau_i^j \tag{16}$$

where ρ' is a parameter within [0, 1]

$$\Delta\ \tau_i^j = \begin{cases} \tau_0, & \text{if } i \text{ is followed by } j \text{ in some sol}_k \\ 0, & \text{otherwise} \end{cases}$$

sol$_k$ is the latest solution constructed by ant k.

C. A* for Path Planning

To apply the A* search for pairwise path planning, two crucial issues must be carefully addressed: 1) define candidate waypoints and their adjacent relationship for the search extension of A* and 2) define the cost functions required by A*, especially the heuristic function, according to the problem characteristics.

1) Search Extension: Based on the fine-grained modeling, we consider all feasible cube centers as candidate waypoints and extend the A* search in a 26-neighbor graph structure. That is, once the center of a certain cube has been processed, A* will check all 26 cubes adjacent to the cube so as to add the unvisited feasible cubes into the open list for subsequent waypoint selection and flag them as visited.

2) Cost Functions: Corresponding to the problem definition, the cost functions should be defined to reflect the path traveling cost in terms of three path optimization requirements, i.e., collision avoidance, risk minimization, and operation cost minimization, where the operation cost includes traveling length cost, AUV turning cost, and height cost. Indeed, collision avoidance has already been guaranteed by the above feasible-cube-based search extension. Thus, in the following, the cost functions, i.e., actual cost function $g(x)$ and heuristic cost function $h(x)$, are defined to combine the risk cost, length cost, height cost, and turning cost corresponding to the weighted-sum cost definition as in (2).

As for the actual cost function $g(x)$, it should be defined to reflect the actual cost of a so-far-constructed path. In A*, a path is composed by the path segments connecting waypoints and the path cost should be obtained by adding the costs of all these path segments. Particularly, during the A* search, given a so-far-constructed path till waypoint x, the path cost till each candidate point y adjacent to x will be

$$g(y) = g(x) + rc(x,y) + lc(x,y) + hc(x,y) + tc(px,x,y) \tag{17}$$

where $g(x)$ is the path cost till x, px is the waypoint prior to x, $rc(x, y)$, $lc(x, y)$, and $hc(x, y)$ compute the risk cost, length cost, and height cost of the path segment from x to y by (9), (3), and (4), respectively, and $tc(px, x, y)$ computes the turning cost at turning point x by (6).

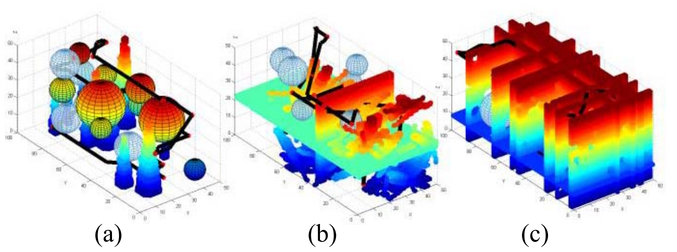
Fig. 2. Examples of different types of synthetic problem instances. (a) G-type is regularly clustered. (b) I-type is irregular. (c) D-type is space-segmented.

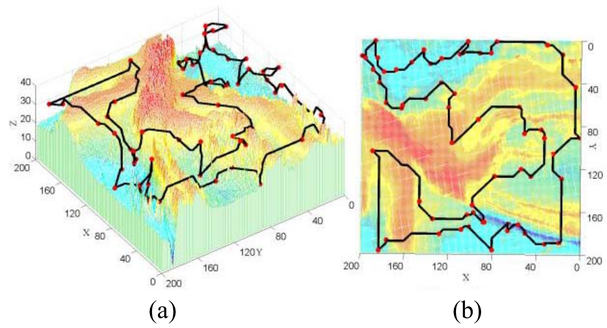
Fig. 3. Path planned by ACO-A* to traverse 60 targets in New Zealand bathymetry. (a) 3-D view. (b) Top view.

As for the heuristic function $h(x)$, the virtual segment $p(x,t)$ from a waypoint x to the destination t is utilized for cost estimation. Particularly, $h(x)$ is defined as

$$h(x) = lc(x,t) + hc(x,t) + tc(px,x,t). \tag{18}$$

Such a heuristic function can be proven to be admissible so as to guarantee A*'s optimality. First, the traveling length and height caused by the virtual path segment $p(x, t)$ are obviously no larger than the actual ones. Second, the turning cost in $h(x)$ only considers the virtual turning angle caused by $p(x, t)$, and the angle must be smaller than the sum of all necessary turning angles if the truly optimal path from x to t requires two or more path segments. For example, in Fig. S5 in the supplementary material, angle c is the virtual turning angle, angles b and d are the actual turning angles, then $c \le (d+b)$ must hold since $c = (a+b)$ and a is smaller than d. Since a larger turning angle corresponds to a larger turning cost, the turning cost in $h(x)$ will not exceed the actual one. Third, the risk cost in $h(x)$ is always 0, which is hence guaranteed to be no larger than the true risk cost. Thus, the estimated cost obtained by $h(x)$ is a lower bound of the actual cost required to reach the destination and $h(x)$ is admissible by Definition 1.

VII. Synthetic Experiments

A. Synthetic Problem Instances

To simulate the 3-D underwater environments with dense obstacles and risky areas, synthetic problem instances are carefully constructed in consideration of obstacle scenarios and risky scenarios. The 3-D environment of fixed size

50*100*50 is considered. To simulate risky scenarios, a number of risky global zones are generated in terms of risky degree *fr*, i.e., the ratio of feasible risky space volume to the whole space volume. To simulate obstructive scenarios, a method is designed to combine a set of 3-D obstacle units of various sizes and shapes, e.g., ring, cylinder, cone, globe, cuboid, hollow plane, spike ball, and random tree. In this way, the method is able to construct a complex scenario of a certain obstructive degree *fo*, i.e., the ratio of obstructive space volume to the whole space volume. In this paper, mainly three types of obstacle scenarios are constructed.

1) *G*-type is regularly clustered, which includes only clustered obstacles of global or coned shape, as exemplified in Fig. 2(a) where the blue hollow spheres are risky zones, and the other colorful things are obstacles.
2) *I*-type is irregular and branchy, which is filled by obstacles in spike ball or random tree, plus several planes with ring hollowness, as Fig. 2(b).
3) *D*-type is space-segmented and the free space is almost segmented into several parts by planes with holes of random number and size, as Fig. 2(c).

B. Parameters

In reality, the four cost weights $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ should be given by a decision maker according to the mission requirements. In the experiments, the same setting of cost weights is guaranteed for all compared algorithms for fairness. We mainly take the setting of (5.0, 1.0, 2.0, and 5.0) for an example and some other settings will also be tested for completeness.

As for the parameters introduced for environment modeling, the cube size is set as 1, i.e., $L = W = H = 1$, the block size B is set as 10, and a balance weight w within [0.4, 0.6] generally performs well for representative location according to our trials and it is set as 0.5. The ACO parameters $q_0, \rho, \rho', \alpha, \beta$ are set as 0.9, 0.1, 0.1, 1.0, 2.0, referring to [6].

C. Performance of ACO-A*

For the lack of ready-made research into 3-D target traveling in dense obstacles, we have not found available comparative algorithms to validate the effectiveness of our proposed ACO-A*. Thus, we design an inverse proof technique and a trial-based proof technique to build a potentially optimal solution and a baseline solution for each problem instance, respectively, and then we compare the solutions obtained by ACO-A* with these bound solutions. Especially, given an upper-bound solution with cost U and a baseline solution with cost L, we define the optimization degree opt of an algorithm as follows:

$$\text{opt(sol)} = (L - \text{cost(sol)})/(L - U) \tag{19}$$

where sol is the final solution obtained by the algorithm and a larger opt value means a larger optimization capability. If the value equals 1, the algorithm finds the upper-bound solution.

The *inverse proof technique* builds a problem instance from an optimal solution as follows.

1) Generate a set of targets in the free space and then construct a potentially optimal solution sol* to traverse these targets in the free space. Then, the optimal target cost graph is obtained by computing the direct line costs among targets. According to the cost graph, an optimal or suboptimal target order is obtained using ACO and then a potentially optimal path sol* is constructed by sequentially connecting all targets in this order.
2) An environment scenario that satisfies the preset *fo* and *fr* is generated by randomly deploying obstacles and risky zones in avoidance of any path segment in sol*. Since sol* is at least a suboptimal solution in the free space, it is very likely to be an optimal solution after risks and obstacles are deployed. Thus, sol* is viewed as an upper-bound or optimal solution in the newly generated scenario.

The *trial-based proof technique* generates a baseline solution through a number of greedy trials. In each trial, the greedy algorithm is performed to construct a target traveling order step by step by always selecting the target that has the least direct line cost from the last selected target, and then a path is planned by the Dijkstra's algorithm following this order. The best path out of all trials is viewed as a baseline solution.

Following the inverse proof technique, we inversely construct 45 instances of *G*-type, *I*-type, or *D*-type with different target number *m*, obstructive degree *fo* and risky degree *fr* (we name each instance by "type-*m*-*fo*"). ACO-A* is run 20 times independently for each instance and the experimental results are detailed in Table I, where column "*m*" lists target number, column "*U*" lists the upper-bound solution costs obtained by the inverse proof technique, column "*L*" lists the baseline solution costs obtained by the trial-based proof technique, columns "*fo*" and "*fr*" correspond to obstructive degree and risky degree, column "opt" lists the optimization degree of ACO-A* computed by (19).

From Table I, the optimization degree opt of ACO-A* is larger than 0.3 on each instance, and opt is not less than 0.5 on 42 out of 45 instances as marked in bold. Particularly, ACO-A* finds the upper-bound solutions on 3 instances, i.e., G-20-0.1, I-20-0.3, I-20-0.4, and on 9 out of 45 instances as marked in italic, the solutions obtained by ACO-A* are quite close to the upper-bound solutions, since ACO-A*'s opt is not less than 0.9. Moreover, the Wilcoxon test has been applied at 5% significance level to compare ACO-A* and the baseline greedy algorithm. According to the test results as shown in column "*W*," ACO-A* performs significantly better than the baseline greedy algorithm. In conclusion, ACO-A* generally has good performance in terms of both the optimization degree and the Wilcoxon test, and ACO-A*'s optimization capability is hence verified to some degree. Additionally, the standard deviation values are almost negligible on each instance. That is, ACO-A* is quite stable and robust.

Moreover, to investigate ACO-A*'s general performance for a random instance, we normally construct 45 instances of *G*-type, *I*-type, or *D*-type with target number *m* within {20, 40, 60}, and obstructive degree *fo* within $\{0.1, 0.2, \ldots, 0.5\}$, and $fr = 0$. Contrary to the inverse instance generation, the normal instance generation first constructs environment scenarios and

TABLE I
COMPARED RESULTS ON INVERSED INSTANCES

m	*fo*	*fr*	G-type					I-type					D-type				
			U	*L*	ACO-A*	*opt*	W	*U*	*L*	ACO-A*	*opt*	W	*U*	*L*	ACO-A*	*opt*	W
20	0.1	0.04	926.4	1019.11	926.4(0.00)	***1.00***	+	746.07	848.14	748.05(0.00)	***0.98***	+	808.89	1014.73	841.51(0.00)	**0.84**	+
20	0.2	0.03	790.69	877.05	806.16(13.52)	**0.82**	+	805.27	968.54	822.86(0.00)	**0.89**	+	787.23	874.91	804.73(6.23)	**0.80**	+
20	0.3	0.02	794.40	853.76	815.56(0.00)	**0.64**	+	795.71	887.68	795.71(0.00)	***1.00***	+	867.16	1105.25	891.79(0.00)	***0.90***	+
20	0.4	0.01	815.62	958.20	883.92(0.00)	**0.52**	+	727.74	962.94	727.74(0.00)	***1.00***	+	730.65	907.66	777.19(0.00)	**0.74**	+
20	0.5	0	779.29	873.09	806.39(0.00)	**0.71**	+	811.55	1040.96	925.41(0.00)	**0.50**	+	769.50	1068.15	859.96(10.50)	**0.70**	+
40	0.1	0.04	1203.17	1405.14	1335.09(1.93)	0.35	+	1148.12	1364.16	1178.73(2.02)	**0.86**	+	1180.03	1507.67	1277.95(7.80)	**0.70**	+
40	0.2	0.03	1228.51	1482.33	1323.89(0.00)	**0.62**	+	1275.34	1570.61	1326.95(1.02)	**0.83**	+	1220.69	1661.94	1264.95(1.11)	***0.90***	+
40	0.3	0.02	1237.52	1360.85	1260.29(0.00)	**0.82**	+	1319.26	1576.97	1412.14(9.75)	**0.64**	+	1128.65	1435.05	1142.99(0.00)	***0.95***	+
40	0.4	0.01	1203.01	1506.59	1238.44(0.00)	**0.88**	+	1226.10	1508.85	1354.36(4.18)	**0.55**	+	1133.91	1445.90	1201.39(9.72)	**0.78**	+
40	0.5	0	1211.40	1494.07	1213.91(0.00)	***0.99***	+	1276.12	1675.51	1366.59(0.00)	**0.77**	+	1223.68	1934.74	1389.85(0.00)	**0.77**	+
60	0.1	0.04	1591.31	1937.27	1675.95(1.00)	**0.76**	+	1623.61	2013.63	1698.62(11.00)	**0.81**	+	1686.66	2048.79	1708.06(3.44)	***0.94***	+
60	0.2	0.03	1552.19	1812.03	1615.72(2.39)	**0.76**	+	1553.70	1832.28	1710.83(10.17)	0.44	+	1678.37	2196.12	1839.10(0.00)	**0.69**	+
60	0.3	0.02	1607.02	1978.82	1767.46(16.07)	**0.57**	+	1592.72	2109.34	1717.83(0.45)	**0.76**	+	1539.74	1880.94	1719.54(16.59)	0.47	+
60	0.4	0.01	1469.45	1709.17	1529.04(0.42)	**0.75**	+	1609.38	1938.50	1674.08(7.23)	**0.80**	+	1573.88	1939.78	1653.64(0.00)	**0.78**	+
60	0.5	0	1501.38	1860.45	1559.97(6.94)	**0.84**	+	1681.51	2136.74	1910.76(2.77)	**0.50**	+	1561.41	2157.72	1634.27(8.35)	**0.88**	+

Column '*m*' lists the target number, '*fo*' and '*fr*' list obstructive degree and risky degree; in each type of scenario, column '*U*' lists upper-bound solution costs, column '*L*' lists baseline solution costs, '*opt*' gives the optimization degree of ACO-A* compared with the upper-bound and baseline solutions, 'W' lists the Wilcoxon test results. W-test (Wilcoxon signed-rank test) has been applied at 5% significance level to compare the traveling costs of ACO-A* against at the greedy algorithm (the baseline algorithm). In each test, the 10 best runs of the greedy algorithm are selected. "+" denotes that the traveling cost of ACO-A* is significantly better than that of the greedy algorithm, and " − " denotes that the cost value of ACO-A* is worse, "*" denotes that the cost values to be compared are not significantly different.

then generates targets in the feasible space. After instance generation, the baseline solutions are found using the trial-based proof technique. To build an upper-bound solution for each instance, we first build a real cost graph of targets by computing the truly optimal traveling cost between any two targets using the A* search, and then find an optimal or suboptimal target traveling order according to the real cost graph using ACO.

The final results of ACO-A* on normal instances are given in Table SII in the supplementary material. From the Wilcoxon test, ACO-A* still achieves significantly better performance than the greedy algorithm on normal instances. Particularly, the optimization degree opt of ACO-A* is generally larger than 0.2. As marked in bold, ACO-A*'s opt is not less than 0.5 on most of instances, i.e., 37 out of 45 instances. On two instances I-20-0.3, D-20-0.3, ACO-A* even finds the upper-bound solutions. In addition, the optimization capability of ACO-A* is more obvious on *D*-type and *I*-type scenarios than *G*-type scenarios, since ACO-A*'s opt is relatively smaller on *G*-type scenarios. In conclusion, ACO-A* is able to find a quite good solution for a random problem instance. Especially, ACO-A*'s advantage becomes more obvious on irregular and space-segmented scenarios.

Finally, in order to investigate whether ACO-A* is able to retain its superiority under different cost weight settings, we further conduct a comparison on normal instances under another weight setting, i.e., (1.0, 1.0, 1.0, and 1.0). From Table SIII, in the supplementary material, we find that ACO-A* still achieves significantly better performance than the greedy algorithm. Typically, ACO-A*'s opt is larger than 0.5 on 41 out of 45 instances. To some degree, the consistent results on different cost weight settings show that the specific cost weight setting has no obvious influence on the ACO-A*'s effectiveness and the weight setting should be more problem-related rather than algorithm-related.

D. Performance of Estimation

In ACO-A*, a good estimated cost graph of targets plays an important role in the subsequent target order determination and path planning. Thus, the estimation strategy should play an important role in the performance of ACO-A*.

In practice, given an environment, it is quite intuitive to estimate the traveling costs between targets by the line segment connecting them. The line-based estimation (LBE) is very simple and needs no special environment modeling. However, if there are dense obstacles, even if the line segment connecting two targets is short, an optimal path between them may be quite winding and long and the LBE hence becomes ineffective. Additionally, the LBE is unable to consider some special kinds of path cost, e.g., the turning cost.

Herein, to investigate whether the proposed RBE has an advantage over the LBE, we construct 24 scenarios of *G*-type, *I*-type, or *D*-type, with obstructive degree equals to $0.1, 0.2, \ldots, 0.8$. Particularly, to investigate the possible negative impact brought by LBE's neglect of turning cost, we compare RBE and LBE under three different settings of turning cost weight, i.e., 5, 50, and 100, resulting in a total of 24×3 comparisons. For each comparison, there are five competition circles and in each circle, a random set of targets is generated for test. In each comparison, the number of victory circles of RBE is counted, and RBE is thought as a winner if its victory number is larger than a half of five. From Table SIV, in the supplementary material, we observe that:

1) RBE wins on more comparisons, i.e., 48 out of 72, and when *fo* is larger than 0.4, RBE wins almost all comparisons, i.e., 33 out of 36. The fact shows that RBE has a great superiority for dealing with densely obstructive cases;
2) RBE performs better in *I*-type and *D*-type scenarios than *G*-type scenarios. Specifically, RBE wins 22 out of 24 *D*-type comparisons, and wins 15 out of 24 *I*-type comparisons. But for *G*-type scenarios, RBE only wins when the obstructive degree is larger than 0.4;
3) as the weight of turning cost increases, LBE's performance becomes worse due to its neglect of turning cost. In contrast, RBE's performance is less affected since it always considers all kinds of path cost.

In conclusion, the compared results show that the proposed representative based estimation generally outperforms the

LBE, especially on instances with more complex obstacles. Also, it is a great advantage for the RBE to be able to consider all kinds of path cost.

E. Necessity of A*

Due to the enormous continuous search space of 3-D path planning, a number of nondeterministic algorithms, especially EAs, have been developed and they have shown time efficiency and good performance. Particularly, in [42], a self-adapted DE (AoDE) is proposed and compared with other EAs, including PSO, jDE, and DEs with different parameter settings for 3-D underwater path planning. The results shows that two EAs, i.e., AoDE and DE with parameters $F = \text{CR} = 0.7$, generally achieve better performance. In this section, we compare these two EAs with A* so as to validate the necessity of A* for 3-D path planning considering dense obstacles. We construct 14 G-type or I-type obstacle scenarios with obstructive degree fo in the range of [0.1, 0.5]. In each scenario, 60 pairs of distant points are randomly generated and then, EAs and A* are adopted for the pairwise path planning between each pair of points.

From the results as shown in Table SV in the supplementary material, the performance of both EAs is obviously worse than that of A*. Apart from this, EAs' performance deteriorates as the obstructive degree fo increases. When fo is larger than 0.4 for G-type and I-type instances, it becomes difficult for EAs to find a feasible path for the targets. When fo is smaller than 0.2, although EAs manages to find feasible paths, the corresponding path costs are far larger than the path costs obtained by A*. Thus, compared with the popular EAs, the traditional graph-search algorithms, such as A* should be more capable for the densely obstructive environments considered in this paper.

F. ACO-A* Versus (GA-AoDE and GA-A*)

In order to further investigate the performance of ACO-A* compared with EAs, we build a hybrid EA, namely GA-AoDE based on our two-layer environment modeling, in which a permutation-based GA [48] is adopted to determine traveling order and AoDE [42] is applied for pairwise path planning. The recommended parameter settings in the original works are adopted. Meanwhile, to investigate the relative importance of ACO and A* to the performance of ACO-A*, we also introduce GA-A* into comparison. GA-A* differs from ACO-A* only in that GA-A* adopts GA rather than ACO for traveling order determination. The comparison results of GA-AoDE, GA-A*, and ACO-A* are shown in Table SVI in the supplementary material. Moreover, we have shown the detailed path information found by these algorithms on three G-type, I-type, and D-type instances with $m = 20$, $fo = 0.1$ in Table SVII in the supplementary material.

First, from Table SVI, in the supplementary material, ACO-A* shows a great superiority over GA-AoDE in terms of both solution quality and time efficiency. ACO-A* is always able to find feasible paths for target traveling while GA-AoDE fails on almost all instances. In fact, the results are consistent with the above finding that AoDE may fail to find a feasible path in a densely obstructive environment. As shown in Table SVII in the supplementary material, although the obstructive degree of 0.1 is quite small, AoDE still fails to find the feasible path(s) to reach one or more targets out of the total 20 targets, especially on I-type or D-type instances which include more complex obstacles.

Second, it is observed that GA-A* also significantly outperforms GA-AoDE. This further shows that the necessity of A* for solving the problem considered in this paper. In fact, comparing GA-A* with ACO-A*, GA-A* is only slightly worse than ACO-A* as shown in bold in Table SVI in the supplementary material. The results show that the great advantage of ACO-A* over GA-AoDE should be mainly attributed to the adoption of A* in ACO-A* for pairwise path planning in densely obstructive environments.

G. Time Complexity

Herein, we investigate the time complexity of the proposed ACO-A* from two respects, i.e., ACO for traveling order determination and A* for pairwise path planning.

1) Comparing ACO with the greedy algorithm as in Table SVIII in the supplementary material, we find that the time consumed by both of them is less than a tenth of second, which should be negligible in reality. But note that ACO is able to achieve a significantly better performance.
2) By comparing A* with the Dijkstra's algorithm as in Table SIX in the supplementary material, we verify that proposed heuristic function for A* does help to accelerate A*'s search process.
3) Comparing A* with two EAs, i.e., AoDE and DE as shown in Table SX in the supplementary material, A* is observed to be more time-efficient, especially on instances with dense and irregular obstacles. On the whole, the proposed ACO-A* is quite efficient. For more details of the time comparison and analysis, please refer to Appendix A in the supplementary material.

VIII. Realistic Experiment

In order to further validate the efficiency of the proposed environment modeling and the proposed ACO-A* to deal with realistic submarine scenarios, we conduct experiments on the high-resolution, i.e., 250 m resolution, gridded New Zealand bathymetry data.[1] The bathymetry belongs to one of the largest deep-water areas under national jurisdiction. For experiments, we utilize a part of the bathymetry as shown in Fig. 3, in which the maximum depth is close to 10 km.

For environment modeling, since the environment height is significantly smaller than the environment length or width, we set the cube size as $(L, W, H) = (12.5, 12.5, 0.25)$. Therefore, there are $200 \times 200 \times 40$ cubes after fine-grained modeling. Furthermore, we set block size B as 10, resulting in a total of $20 \times 20 \times 4$ blocks. For path cost evaluation, we set the cost weights $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ as 50, 1, 10, and 50, respectively,

[1]The New Zealand Region bathymetric datasets (2016) are free available online: https://www.niwa.co.nz/our-science/oceans/bathymetry [2017-10-19].

where the unit of traveling length is kilometer. The parameter setting of ACO-A* keeps unchanged.

For completeness, we perform experiments in three kinds of scenarios: 1) the *above-safe scenario* includes only the New Zealand bathymetry, and the space above the bathymetry is safe; 2) the *risky scenario* includes bathymetry and randomly generated risky zones, which occupy 10% of the space above the bathymetry; and 3) the *obstructive scenario* includes bathymetry and randomly generated obstructive zones, which occupy 10% of the space above the bathymetry. In each scenario, we randomly generate 20, 40, and 60 targets and perform ACO-A* for their path planning. The results are given in Table SXI in the supplementary material, which presents not only the total traveling cost of each path, but also the four kinds of path cost constituting the traveling cost. Since the weight of length cost equals 1.0, the "length cost" column indeed lists the total traveling length of each path and length unit is kilometer. Fig. 3 illustrates the path planned by ACO-A* to traverse 60 targets in the above-free scenario and the path seems quite reasonable.

Furthermore, in order to validate the efficiency of the proposed ACO-A* as well as the efficiency of the proposed RBE for quick cost graph building in realistic scenarios, we compare the performance of ACO-A* using representative-based estimated cost graph, line-based estimated cost graph, and the real cost graph in each scenario, respectively. To some degree, the line-based and real-based ACO-A* indeed provides baseline solutions and upper-bound solutions, respectively. According to these bound solutions, the optimization ratio of the RBE is computed and listed in column opt in Table SXII in the supplementary material. In each scenario, opt value is found to be not less than 0.85. The results show that the proposed ACO-A* and its RBE still perform well in realistic environments.

IX. Conclusion

In this paper, to solve the target traveling problem in densely obstructive environments, an algorithm, namely ACO-A* is proposed by combining ACO and the A* search algorithm. In ACO-A*, ACO is responsible to determine a traveling order of targets according to an estimated cost graph, which is built using a RBE strategy based on the coarse-grained modeling. Following the traveling order obtained by ACO, A* performs pairwise path planning based on a search graph obtained by the fine-grained modeling. The optimization capability of ACO-A* has been verified on both synthetic scenarios and realistic scenarios. Particularly, the effectiveness of RBE and the necessity of A* have been demonstrated by thorough comparison experiments. The time efficiency of the proposed ACO-A* has also been validated in experiments.

In the future study, there are several potential directions: 1) consider multiobjective optimization for multiple kinds of traveling cost so as to avoid the use of cost weights; 2) develop distributed algorithm versions so as to deal with larger-scale problems; and 3) consider more realistic mission requirements to improve the algorithm practicality.

References

[1] M. Chyba, "Autonomous underwater vehicles," *Ocean Eng.*, vol. 36, no. 1, p. 1, 2009.

[2] C. Petres *et al.*, "Path planning for autonomous underwater vehicles," *IEEE Trans. Robot.*, vol. 23, no. 2, pp. 331–341, Apr. 2007.

[3] Z. Zeng *et al.*, "A survey on path planning for persistent autonomy of autonomous underwater vehicles," *Ocean Eng.*, vol. 110, pp. 303–313, Dec. 2015.

[4] L. Hernando, A. Mendiburu, and J. A. Lozano, "A tunable generator of instances of permutation-based combinatorial optimization problems," *IEEE Trans. Evol. Comput.*, vol. 20, no. 2, pp. 165–179, Apr. 2016.

[5] V. Santucci, M. Baioletti, and A. Milani, "Algebraic differential evolution algorithm for the permutation flowshop scheduling problem with total flowtime criterion," *IEEE Trans. Evol. Comput.*, vol. 20, no. 5, pp. 682–694, Oct. 2016.

[6] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53–66, Apr. 1997.

[7] Y. Zhou, "Runtime analysis of an ant colony optimization algorithm for TSP instances," *IEEE Trans. Evol. Comput.*, vol. 13, no. 5, pp. 1083–1092, Oct. 2009.

[8] Q. Yang *et al.*, "Adaptive multimodal continuous ant colony optimization," *IEEE Trans. Evol. Comput.*, vol. 21, no. 2, pp. 191–205, Apr. 2017.

[9] X. Yu *et al.*, "Set-based discrete particle swarm optimization based on decomposition for permutation-based multiobjective combinatorial optimization problems," *IEEE Trans. Cybern.*, vol. 48, no. 7, pp. 2139–2153, Jul. 2018.

[10] N. K. Yilmaz, C. Evangelinos, P. F. J. Lermusiaux, and N. M. Patrikalakis, "Path planning of autonomous underwater vehicles for adaptive sampling using mixed integer linear programming," *IEEE J. Ocean. Eng.*, vol. 33, no. 4, pp. 522–537, Oct. 2008.

[11] T. Phanthong, "Real time underwater obstacle avoidance and path re-planning using simulated multi-beam forward looking sonar images for autonomous surface vehicle," *Eng. J.*, vol. 19, no. 1, pp. 107–123, 2015.

[12] R. J. Szczerba, P. Galkowski, I. S. Glicktein, and N. Ternullo, "Robust algorithm for real-time route planning," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 36, no. 3, pp. 869–878, Jul. 2000.

[13] Y. Fu, M. Ding, and C. Zhou, "Phase angle-encoded and quantum-behaved particle swarm optimization applied to three-dimensional route planning for UAV," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 42, no. 2, pp. 511–526, Mar. 2011.

[14] P. Yang, K. Tang, J. A. Lozano, and X. Cao, "Path planning for single unmanned aerial vehicle by separately evolving waypoints," *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1130–1146, Oct. 2015.

[15] J. Kok, L. F. Gonzalez, and N. A. Kelson, "FPGA implementation of an evolutionary algorithm for autonomous unmanned aerial vehicle on-board path planning," *IEEE Trans. Evol. Comput.*, vol. 17, no. 2, pp. 272–281, Apr. 2013.

[16] V. Roberge, M. Tarbouchi, and G. Labonte, "Comparison of parallel genetic algorithm and particle swarm optimization for real-time UAV path planning," *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 132–141, Feb. 2013.

[17] Y. Fu, M. Ding, C. Zhou, and H. Hu, "Route planning for unmanned aerial vehicle (UAV) on the sea using hybrid differential evolution and quantum-behaved particle swarm optimization," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 43, no. 6, pp. 1451–1465, Nov. 2013.

[18] Z. Sun *et al.*, "Path planning for GEO-UAV bistatic SAR using constrained adaptive multiobjective differential evolution," *IEEE Trans. Geosci. Remote Sens.*, vol. 54, no. 11, pp. 6444–6457, Nov. 2016.

[19] K. K. Lim, Y. Ong, M. Lim, X. Chen, and A. Agarwal, "Hybrid ant colony algorithms for path planning in sparse graphs," *Soft Comput.*, vol. 12, no. 10, pp. 981–994, 2008.

[20] B. Zhang and H. Duan, "Three-dimensional path planning for uninhabited combat aerial vehicle based on predator-prey pigeon-inspired optimization in dynamic environment," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 14, no. 1, pp. 97–107, Jan./Feb. 2017.

[21] W. Zhu and H. Duan, "Chaotic predator–prey biogeography-based optimization approach for UCAV path planning," *Aerosp. Sci. Technol.*, vol. 32, no. 1, pp. 153–161, 2014.

[22] E. Besada-Portas, L. D. La Torre, J. M. D. La Cruz, and B. de Andrés-Toro, "Evolutionary trajectory planner for multiple UAVs in realistic scenarios," *IEEE Trans. Robot.*, vol. 26, no. 4, pp. 619–634, Aug. 2010.

[23] Y. Petillot, I. T. Ruiz, and D. M. Lane, "Underwater vehicle obstacle avoidance and path planning using a multi-beam forward looking sonar," *IEEE J. Ocean. Eng.*, vol. 26, no. 2, pp. 240–251, Apr. 2001.

[24] B. Braginsky and H. Guterman, "Obstacle avoidance approaches for autonomous underwater vehicle: Simulation and experimental results," *IEEE J. Ocean. Eng.*, vol. 41, no. 4, pp. 882–892, Oct. 2016.

[25] M. P. Aghababa, "3D path planning for underwater vehicles using five evolutionary optimization algorithms avoiding static and energetic obstacles," *Appl. Ocean Res.*, vol. 38, pp. 48–62, Oct. 2012.

[26] L. Zhang, L. Zhang, S. Liu, J. Zhou, and C. Papavassiliou, "Three-dimensional underwater path planning based on modified wolf pack algorithm," *IEEE Access*, vol. 5, pp. 22783–22795, 2017.
[27] Z. Yan, J. Li, G. Zhang, and Y. Wu, "A real-time reaction obstacle avoidance algorithm for autonomous underwater vehicles in unknown environments," *Sensors Basel*, vol. 18, p. E438, Feb. 2018.
[28] D. Zhu, H. Huang, and S. X. Yang, "Dynamic task assignment and path planning of multi-AUV system based on an improved self-organizing map and velocity synthesis method in three-dimensional underwater workspace," *IEEE Trans. Cybern.*, vol. 43, no. 2, pp. 504–514, Apr. 2013.
[29] B. Englot and F. S. Hover, "Three-dimensional coverage planning for an underwater inspection robot," *Int. J. Robot. Res.*, vol. 32, nos. 9–10, pp. 1048–1073, 2013.
[30] E. Galceran *et al.*, "Coverage path planning with real-time replanning and surface reconstruction for inspection of three-dimensional underwater structures using autonomous underwater vehicles," *J. Field Robot.*, vol. 32, no. 7, pp. 952–983, 2015.
[31] J. Han, J. Ok, and W. K. Chung, "An ethology-based hybrid control architecture for an autonomous underwater vehicle for performing multiple tasks," *IEEE J. Ocean. Eng.*, vol. 38, no. 3, pp. 514–521, Jul. 2013.
[32] J. Mcmahon and E. Plaku, "Mission and motion planning for autonomous underwater vehicles operating in spatially and temporally complex environments," *IEEE J. Ocean. Eng.*, vol. 41, no. 4, pp. 893–912, Oct. 2016.
[33] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, Jul. 1968.
[34] M. Dorigo and C. Blum, "Ant colony optimization theory: A survey," *Theor. Comput. Sci.*, vol. 344, nos. 2–3, pp. 243–278, 2005.
[35] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 26, no. 1, pp. 29–41, Feb. 1996.
[36] R. S. Parpinelli, H. S. Lopes, and A. A. Freitas, "Data mining with an ant colony optimization algorithm," *IEEE Trans. Evol. Comput.*, vol. 6, no. 4, pp. 321–332, Aug. 2002.
[37] D. Merkle, M. Middendorf, and H. Schmeck, "Ant colony optimization for resource-constrained project scheduling," *IEEE Trans. Evol. Comput.*, vol. 6, no. 4, pp. 333–346, Aug. 2002.
[38] X. Cao, D. Zhu, and S. X. Yang, "Multi-AUV target search based on bioinspired neurodynamics model in 3-D underwater environments," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 11, pp. 2364–2374, Nov. 2016.
[39] R. B. Wynn *et al.*, "Autonomous underwater vehicles (AUVs): Their past, present and future contributions to the advancement of marine geoscience," *Mar. Geol.*, vol. 352, pp. 451–468, Mar. 2014.
[40] X. Xiang, C. Yu, and Q. Zhang, "Robust fuzzy 3D path following for autonomous underwater vehicle subject to uncertainties," *Comput. Oper. Res.*, vol. 84, pp. 165–177, Aug. 2017.
[41] K. A. Belibassakis, B. Simon, J. Touboul, and V. Rey, "A coupled-mode model for water wave scattering by vertically sheared currents in variable bathymetry regions," *Wave Motion*, vol. 74, no. 2017, pp. 73–92, 2017.
[42] C.-B. Zhang, Y.-J. Gong, J.-J. Li, and Y. Lin, "Automatic path planning for autonomous underwater vehicles based on an adaptive differential evolution," in *Proc. GECCO*, 2014, pp. 89–96.
[43] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *J. Struct. Multidiscipl. Optim.*, vol. 26, no. 6, pp. 369–395, 2004.
[44] D. Jones and G. A. Hollinger, "Planning energy-efficient trajectories in strong disturbances," *IEEE Robot. Autom. Lett.*, vol. 2, no. 4, pp. 2080–2087, Oct. 2017.
[45] H. Duan, P. Li, Y. Shi, X. Zhang, and C. Sun, "Interactive learning environment for bio-inspired optimization algorithms for UAV path planning," *IEEE Trans. Edu.*, vol. 58, no. 4, pp. 276–281, Nov. 2015.
[46] D. Kruger, R. Stolkin, A. Blum, and J. Briganti, "Optimal AUV path planning for extended missions in complex, fast-flowing estuarine environments," in *Proc. IEEE ICRA*, 2007, pp. 4265–4270.
[47] Y. Ma, M. Hu, and X. Yan, "Multi-objective path planning for unmanned surface vehicle with currents effects," *ISA Trans.*, vol. 75, pp. 137–156, Apr. 2018.
[48] H. D. Nguyen, I. Yoshihara, K. Yamamori, and M. Yasunaga, "Implementation of an effective hybrid GA for large-scale traveling salesman problems," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 37, no. 1, pp. 92–99, Feb. 2007.

Xue Yu (S'15) received the bachelor's degree from Sun Yat-sen University, Guangzhou, China, in 2015, where she is currently pursuing the Ph.D. degree.

She is a Research Assistant with the School of Computer Science and Engineering, South China University of Technology, Guangzhou. Her current research interests include evolutionary computation algorithms and their applications on large scale computing, autonomous path planning, and intelligent transportation.

Wei-Neng Chen (S'07–M'12–SM'17) received the bachelor's and Ph.D. degrees from Sun Yat-sen University, Guangzhou, China, in 2006 and 2012, respectively.

He is currently a Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou. He has published over 70 papers in international journals and conferences. His current research interests include swarm intelligence algorithms and their applications on cloud computing, operations research, and software engineering.

Dr. Chen was a recipient of the IEEE Computational Intelligence Society Outstanding Dissertation Award in 2016 and the National Science Fund for Excellent Young Scholars in 2016.

Tianlong Gu received the M.Eng. degree from Xidian University, Xi'an, China, in 1987 and the Ph.D. degree from Zhejiang University, Hangzhou, China, in 1996.

From 1998 to 2002, he was a Research Fellow with the School of Electrical and Computer Engineering, Curtin University of Technology, Perth, WA, Australia, and a Post-Doctoral Fellow with the School of Engineering, Murdoch University, Perth. He is currently a Professor with the School of Computer Science and Engineering, Guilin University of Electronic Technology, Guilin, China. His current research interests include formal methods, data and knowledge engineering, software engineering, and information security protocol.

Huaqiang Yuan received the Ph.D. degree from Shanghai Jiao Tong University, Shanghai, China, in 1996.

He is currently a Professor with the School of Computer Science and Network Security, Dongguan University of Technology, Dongguan, China. His current research interests include computational intelligence and cyberspace security.

Huaxiang Zhang received the Ph.D. degree from Shanghai Jiaotong University, Shanghai, China, in 2004.

He is currently a Professor with the School of Information Science and Engineering, Shandong Normal University, Jinan, China, where he was an Associate Professor with the Department of Computer Science from 2004 to 2005. He has authored over 100 journal and conference papers and has been granted eight invention patents. His current research interests include machine learning, pattern recognition, evolutionary computation, and Web information processing.

Jun Zhang (M'02–SM'08–F'17) received the Ph.D. degree in electrical engineering from the City University of Hong Kong, Hong Kong, in 2002.

He is currently a Professor with the South China University of Technology, Guangzhou, China. His current research interests include computational intelligence, cloud computing, wireless sensor networks, operations research, and power electronic circuits. He was also appointed as the Changjiang Chair Professor in 2013. He has authored seven research books and book chapters, and over 50 IEEE TRANSACTIONS papers in the above areas.

Prof. Zhang was a recipient of the National Science Fund for Distinguished Young Scholars in 2011 and the First-Grade Award in Natural Science Research from the Ministry of Education, China, in 2009. He is currently an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, and the IEEE TRANSACTIONS ON CYBERNETICS. He is the Founding and Current Chair of the IEEE Guangzhou section and IEEE Beijing (Guangzhou) Section Computational Intelligence Society Chapters. He is the Founding and Current Chair of the ACM Guangzhou Chapter.