Journal of *Marine Science and Engineering*

Article

Enhancing Autonomous Underwater Vehicle Decision Making through Intelligent Task Planning and Behavior Tree Optimization

Dan Yu [†], **Hongjian Wang** [*,†], **Xu Cao**, **Zhao Wang**, **Jingfei Ren** and **Kai Zhang**

College of Intelligent System Science and Engineering, Harbin Engineering University, Harbin 150001, China; cocomomo@hrbeu.edu.cn (D.Y.); caoxu1020@163.com (X.C.); 814551366@hrbeu.edu.cn (Z.W.); renjingfei@hrbeu.edu.cn (J.R.); promotion5@foxmail.com (K.Z.)

* Correspondence: cctime99@163.com

† These authors contributed equally to this work.

Abstract: The expansion of underwater scenarios and missions highlights the crucial need for autonomous underwater vehicles (AUVs) to make informed decisions. Therefore, developing an efficient decision-making framework is vital to enhance productivity in executing complex tasks within tight time constraints. This paper delves into task planning and reconstruction within the AUV control decision system to enable intelligent completion of intricate underwater tasks. Behavior trees (BTs) offer a structured approach to organizing the switching structure of a hybrid dynamical system (HDS), originally introduced in the computer game programming community. In this research, an intelligent search algorithm, MCTS-QPSO (Monte Carlo tree search and quantum particle swarm optimization), is proposed to bolster the AUV's capacity in planning complex task decision control systems. This algorithm tackles the issue of the time-consuming manual design of control systems by effectively integrating BTs. By assessing a predefined set of subtasks and actions in tandem with the complex task scenario, a reward function is formulated for MCTS to pinpoint the optimal subtree set. The QPSO algorithm is then leveraged for subtree integration, treating it as an optimal path search problem from the root node to the leaf node. This process optimizes the search subtree, thereby enhancing the robustness and security of the control architecture. To expedite search speed and algorithm convergence, this paper recommends reducing the search space by pre-grouping conditions and states within the behavior tree. The efficacy and superiority of the proposed algorithm are validated through security and timeliness evaluations of the BT, along with comparisons with other algorithms for automatic AUV decision control behavior tree design. Ultimately, the effectiveness and superiority of the proposed algorithm are corroborated through simulations on a multi-AUV complex task platform, showcasing its practical applicability and efficiency in real-world underwater scenarios.

Keywords: autonomous underwater vehicles (AUVs); decision-making framework; behavior trees (BTs); MCTS; QPSO algorithm; task planning

check for updates

Citation: Yu, D.; Wang, H.; Cao, X.; Wang, Z.; Ren, J.; Zhang, K. Enhancing Autonomous Underwater Vehicle Decision Making through Intelligent Task Planning and Behavior Tree Optimization. *J. Mar. Sci. Eng.* **2024**, *12*, 791. https://doi.org/10.3390/jmse12050791

Academic Editor: Rafael Morales

Received: 11 April 2024
Revised: 30 April 2024
Accepted: 5 May 2024
Published: 8 May 2024

1. Introduction

As unmanned systems are increasingly deployed in complex domains, particularly in expanding underwater applications [1], autonomous underwater vehicles (AUVs) [2] have emerged as vital assets in deep-sea environments [3]. AUVs serve as mobile sentinels, extensively utilized for tasks such as observation, orientation, detection, monitoring, and other applications [4]. With the growing complexity of tasks in multi-AUV systems, the AUV control system demands higher levels of sophistication. Manual design of control systems is often impractical due to its inherent time-intensive nature and the need for specialized expertise, especially as robotic tasks become more intricate or applications require larger multi-robot teams [5].

As the number of autonomous underwater vehicles (AUVs) involved in a task increases, the design of different control architectures for various role tasks becomes necessary. Manual design for such scenarios proves challenging in terms of implementation and reusability. Automatic design, which has seen rapid development in recent years, offers an effective solution for addressing these complexities [6]. In the realm of automatic control system design, the process is transformed into an objective optimization problem, with the optimization objectives centered around the efficiency and safety of the control system [7]. Behavior trees (BTs) [8,9] were originally developed to address issues of unclear logic and poor interpretability in finite state machines. While initially applied to behavior control in computer games for non-human characters [10], BTs have found widespread use in robot control system design as well [9]. BTs excel over finite state machines (FSMs) [11], decision trees [12], and other algorithms in terms of readability, recursiveness, and modular design of control systems. These advantages are inherent in the construction of the behavior tree algorithm, where the design focuses on constructing different tasks, with state serving as a secondary consideration. In essence, a behavior tree is a directed rooted tree comprising leaf nodes that assess conditions and trigger actions, along with internal nodes that define the logical structure. The control flow within a behavior tree follows task switching, simplifying the manual design process, particularly when compared to FSM-based control system designs reliant on state transitions.

While behavior trees have offered a convenient solution for manual AUV control system design, their efficiency diminishes when task switching is frequent, the number of AUVs increases, or AUV autonomy rises. Given this context, expediting the design of AUV control systems without compromising functionality has become a pressing concern. Leveraging the design algorithm of automatic control systems based on optimization objectives, the behavior tree system has been identified as a key element in maximizing AUV action, sensor capability, communication capability, and task performance.

In this paper, we introduce a novel learning-based behavior tree construction algorithm tailored for complex multi-AUV tasks to generate behavior trees that optimize performance attributes. The algorithm's core principle is rooted in the fundamental characteristic of behavior trees as directed acyclic graphs, upon which a graph search algorithm is developed. This algorithm employs Monte Carlo tree search (MCTS) [13] to enhance tree performance on directed acyclic graphs efficiently and swiftly. By utilizing this search method, the algorithm can swiftly identify the most suitable subtree.

During the optimization process, the connectivity of directed acyclic graphs facilitates rapid dissemination of learning information. However, this feature can lead to challenges in generalization. Drawing inspiration from previous works [2,14], we incorporate quantum particle swarm optimization (QPSO) during subtree fusion to refine the final behavior tree. The algorithm alternates between MCTS and QPSO, accelerating convergence and expediting the design process of AUV control systems. Additionally, actions and states within the AUV behavior tree are pre-grouped for distinct tasks and states, reducing unnecessary search costs in MCTS searches.

In the current landscape of AUV intelligent decision-making systems, the following challenges stand out as areas requiring further attention and resolution:

- **Time-Intensive Manual Design**: Manual design of decision control systems proves inadequate in handling the timeliness demands posed by intricate tasks or an increased number of AUVs.
- **Balancing Safety and Efficiency**: Ensuring both the safety of AUV operations and the efficiency of task completion within complex scenarios remains a critical challenge in decision control for AUVs.
- **Enhancing Decision Structures**: The utilization of reactive decision control structures is pivotal for addressing the transformation from goal-driven behaviors to achieve more effective decision-making processes in AUV operations.

Building upon the aforementioned challenges, this paper introduces an automatic generation algorithm for AUV decision-making systems in complex task scenarios to address the limitations of manual control system design. Through a series of experiments, the efficacy of the proposed algorithm is validated. The key contributions of this study are as follows:

- **Establishment of Behavior Tree Search Model**: A behavior tree search model based on directed acyclic graphs is developed, serving as a foundational framework for the design of AUV control systems.
- **Introduction of Optimization Algorithm**: An optimization algorithm, leveraging MCTS-QPSO, is presented for the automatic optimization of the optimal behavior tree structure, enhancing the efficiency and effectiveness of decision-making processes.
- **Enhanced Optimization Efficienc**y: The optimization algorithm's efficiency is further bolstered through the pre-grouping of actions and states, reducing unnecessary search costs and streamlining the overall optimization process.

The paper's organizational structure is outlined as follows. In Section 2, a comprehensive review of the multi-agent decision framework is presented. Section 3 introduces the problems and hypotheses examined in this study. Section 4 delves into the intricate details of the automatic design of AUV control structures based on behavior trees using the QPSO-MCTS algorithm. Finally, Section 5 validates the algorithm through simulations, assessing its effectiveness across various dimensions on the multi-AUV countermeasure simulation platform. Furthermore, a comparative analysis of the strengths and weaknesses of related algorithms is conducted to provide a comprehensive evaluation.

2. Literature Review

The control and decision challenges associated with AUV complex tasks necessitate an analysis of tasks based on the intricate interconnections within the ocean's limited resources and complex environment [15], facilitating the acquisition of sequential task decisions [16]. Numerous scholars have approached this issue through the lens of task planning. Brito et al. [17] introduced a Markov chain for critical stage analysis, focusing on multi-task switching and sequential decision-making processes. Wei et al. [18] explored AUV complex task planning using heuristic algorithms, leading to enhanced task completion efficiency. Additionally, Bhatt et al. [19] proposed an embedded aided decision framework and illustrated its practical application in real-world tasks. These contributions underscore the diverse perspectives and methodologies employed in tackling the challenges of AUV decision making in complex task environments.

Multi-AUV decision making stands as a pivotal research domain within the realm of unmanned systems. Whether operating cooperatively or in an adversarial capacity during missions, these unmanned vehicles coexist within a shared mission environment [20]. The decisions made by individual AUVs influence both the environment and the actions of other agents, underscoring the necessity for AUVs to adaptively make sequential decisions in response to environmental dynamics. In the context of decision making, scholars have emphasized the importance of not only continuously detecting and analyzing the environment but also incorporating predictions and estimations of other agents' behaviors. To address these multifaceted task requirements, various models such as partially observable Markov decision processes (POMDPs) [21], interactive POMDPs [22], and interactive dynamic influence diagrams [23] have been proposed. These predictive models enable decision making based on anticipated environmental and agent behaviors. Alternatively, a category of models grounded in agent behavior, including decision trees [24], finite state machines (FSMs) [25], and behavior trees [26], offer a different approach. These models do not necessitate extensive prior knowledge or detailed agent models but instead focus on elucidating an agent's behavior under specific conditions. Moreover, in many instances, decision problems can be reframed as optimal solution challenges, providing additional avenues for addressing complex decision-making scenarios.

Behavior trees have transcended their origins in the gaming industry and found applications in diverse fields beyond gaming [27,28]. Concurrently, significant advancements have been made in the design of robot control systems and behavior generation and control [29], with their application in underwater vehicles gaining traction this year [30]. This paper delves into the construction of an optimal behavior tree for multi-AUV complex task control. Manual design of behavior trees becomes cumbersome as the number of AUVs increases or their roles change due to the non-one-to-one nature of adversarial tasks. Behavior trees operate reactively, checking states and switching tasks accordingly. As the complexity of tasks and states grows, so does the size of the behavior tree, making manual design increasingly challenging [31]. To address this challenge, behavior trees can be constructed by amalgamating reactivity and target orientation with planning algorithms. Mainstream behavior tree design and learning algorithms leverage optimization techniques like evolutionary algorithms [32], ant colony optimization [33], genetic algorithms [34], etc., with variations in how they represent and construct tree structures.

To enhance AUV intelligence, behavior should encompass autonomous search and introspection beyond simple low-level actions. Learning algorithms have emerged as efficient tools for automatically constructing behavior trees [35–37]. Researchers have explored various approaches, such as combining manual design with neural networks [38], integrating reinforcement learning with behavior tree design [39], and optimizing manually designed behavior trees using Q-learning principles [40]. While most learning-based behavior tree construction methods focus on optimizing actions, the inherent logic of behavior trees is sometimes overlooked. Behavior trees, as directed acyclic graphs, can benefit from search algorithms [41,42] optimized to uphold the internal logic of the tree structure. The Monte Carlo tree search (MCTS) algorithm [43,44] has been extensively applied in decision-making scenarios like strategy selection. In prior research [45], MCTS was employed to address the control decision challenges in multi-UUV pursuit tasks, showcasing promising performance. Researchers have combined MCTS with simulated annealing (SA) [46] to construct behavior trees for underwater search tasks [2] and proposed fusion strategies using elastic potential fields and particle swarm optimization for AUV target interception missions [47].

Building upon these foundations, this study introduces a behavior tree control structure for AUV countermeasures based on the MCTS algorithm in conjunction with particle swarm optimization (QPSO). The efficacy of the proposed algorithm is validated through practical applications in AUV adversarial tasks and performance evaluations of behavior trees, demonstrating its superiority compared to several mainstream algorithms.

3. Problem Formulation

In an adversarial environment, multiple tasks for AUVs can be subdivided into smaller subtasks, as illustrated in Figure 1, utilizing a behavior library in task decision making. Task decomposition involves breaking down intricate tasks into more manageable subtasks. In the realm of AUVs, task decomposition is crucial for enabling the vehicle to navigate and function effectively in demanding environments. By breaking down complex tasks into smaller subtasks, AUVs can optimize resource allocation, enhance decision-making processes, and adjust to evolving conditions more efficiently.

From the research, we can see that conditions and actions in the behavior tree are closely related. During the construction of the behavior tree, in order to ensure the safety of AUV behavior and the effectiveness of the behavior tree, a number of condition judgment nodes are needed, and relevant conditions and behaviors are pre-grouped before the design of the control system.

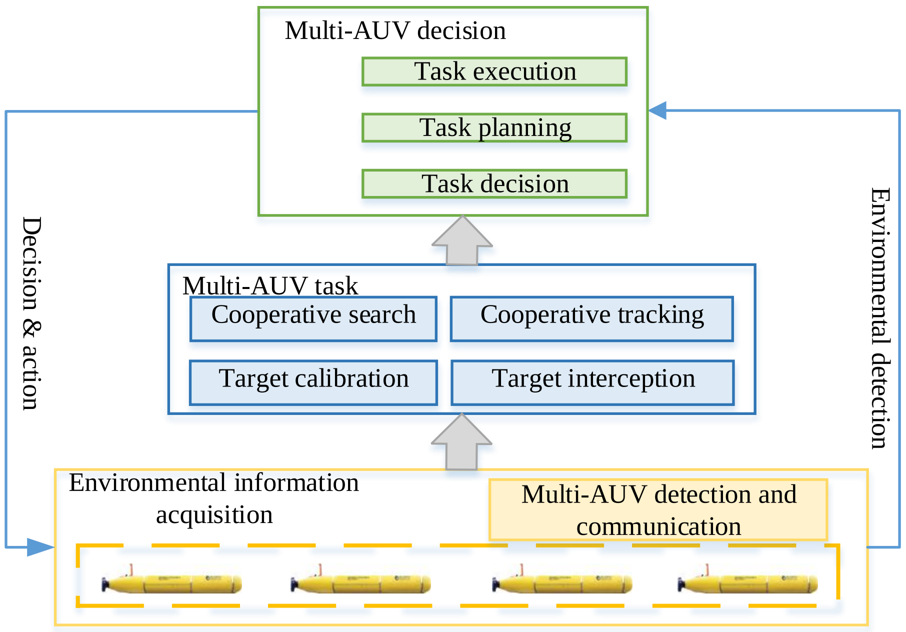


Figure 1. Multi-AUV complex task.

It can be seen from Figure 1 that the basic task of multi-AUV cooperation includes cooperative search, cooperative tracking, cooperative calibration, and cooperative interception, and the multi-AUV decision is divided into task decision, which refers to the decision made during the execution of the task about selecting an action plan or solving a problem. Task decision making usually involves weighing and evaluating different options to determine the best course of action to reach the goal of the task. Task decision making can involve selecting appropriate strategies, adjusting execution plans, handling unexpected situations, etc., to ensure that the task can be completed successfully. Task planning is a set of steps and plans of basic actions made before the execution of a task, which is used to determine how to reach a specific goal or complete a specific task. Thus, the decision-making process of multiple AUVs is divided into task decision making, task planning, and task execution, which is summarized in the order of upper to lower levels of decision making.

From the above analysis of complex tasks, the cooperative search, tracking, calibration, and tracking tasks of multiple AUVs can also be regarded as complex tasks. The difference from the previous single task decision is that the internal logic of the task needs to be analyzed in the complex task, and the decision also involves high-level decisions such as task switching, priority analysis, and state prediction. The complete decision control architecture for complex tasks is shown in Figure 2. As shown in Figure 2, unlike previous simple tasks that only need to analyze the heading, speed, etc., to generate control commands and send them to the actuator through the controller, complex tasks involve task planning, so the decision control is divided into two levels: high-level control and low-level control. High-level control is mainly responsible for situation estimation, task analysis, and policy selection, while low-level control mainly includes policy-based behavior and action output sent directly to the actuator through the controller.

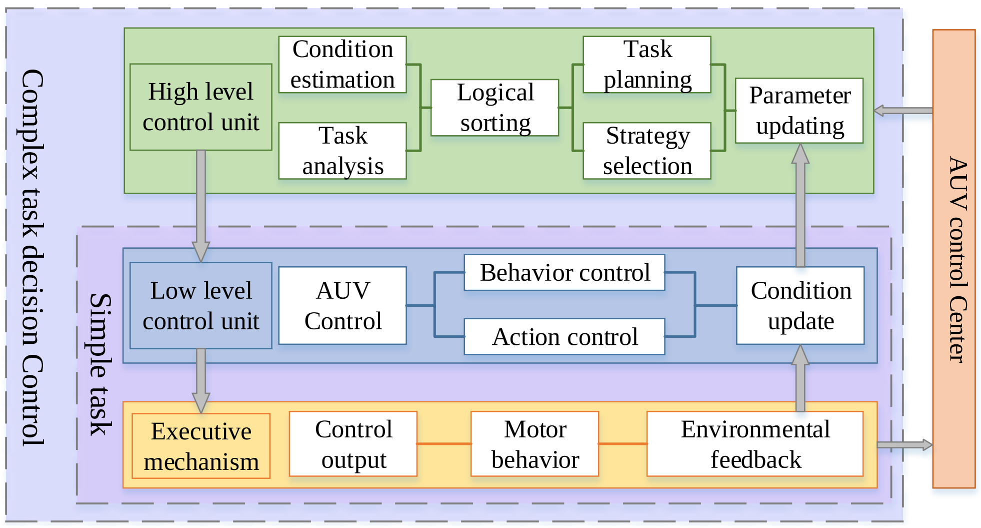


Figure 2. AUV complex task decision control architecture.

4. Behavior Tree Learning Algorithm

BTs represent a concept and methodology for constructing control algorithms aimed at enhancing code reusability and readability, and the simplicity of control flow design [48]. Essentially, BTs are structured as directed acyclic graphs comprising encapsulated components. The nodes within a behavior tree can be categorized into six functional types: Fallback, Sequence, Parallel, Action, Condition, and Decorator, as illustrated in Table 1. The "Running" status denotes ongoing processes within the node that are not yet finalized. The "Success" and "Failure" states indicate the outcomes following the completion of node execution.

Table 1. The node types of behavior tree.

Node Type	Success	Failure	Running
Fallback	One child succeeds	All children fail	One child running
Sequence	All children succeed	One child fails	One child running
Parallel	>M children succeed	N−M children fail	Else
Action	Upon completion	Not complete	During completion
Condition	True	False	Never

Similar to other tree structures, BTs consist of root nodes, leaf nodes, and intermediate nodes. Typically, the leaf nodes in a behavior tree represent actions and conditions. The action executed based on the conditions yields a result that is passed back to the intermediate node, subsequently propagating up to the root node.

4.1. Traditional Behavior Tree and AUV Control

BTs are designed to consolidate various node types, including execution nodes, control flow nodes, and decoration nodes, to enhance the readability of the control system. Before delving into the algorithms discussed in this article, we will provide a brief explanation of the functions of these nodes to establish a strong foundation for the subsequent algorithms. An AUV adversarial control behavior subtree [49] containing fundamental nodes is presented to illustrate the nodes and logic inherent in the behavior tree algorithm, as depicted in Figure 3. In Figure 3, the square denotes the control flow node, the oval represents the condition node, and the rectangle signifies the action node. To clearly differentiate between the various nodes, distinct color markers are utilized.

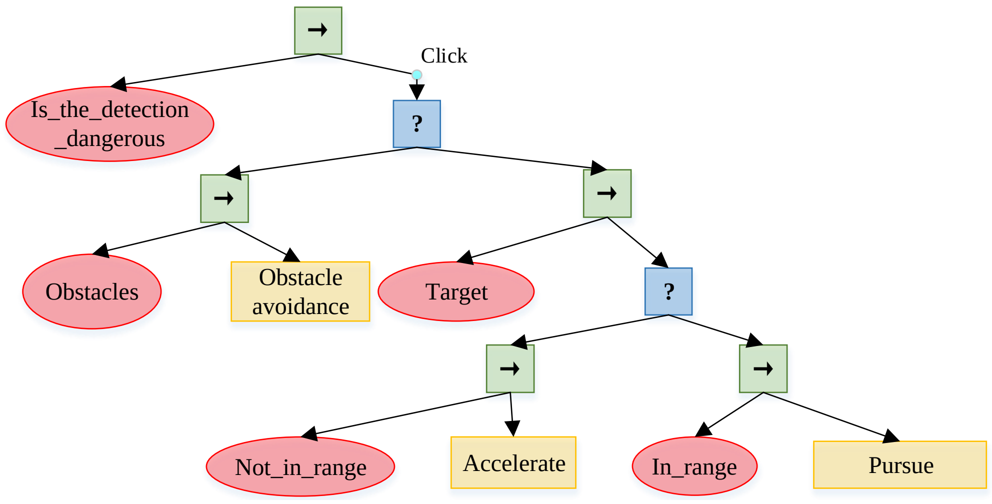


Figure 3. AUV adversarial control behavior subtree.

The Execution node comprises two fundamental types of nodes: conditions and actions. The basic functions of these nodes are self-evident, as illustrated in Figure 3, where the rectangular and oval leaf node boxes symbolize conditions and actions, respectively. These functions are integrated into the AUV countermeasure task. In the condition evaluation process depicted in Figure 3, the AUV utilizes sensors to determine if it is within the permitted targeting range, which corresponds to the condition node. Subsequently, upon a true evaluation, the targeting strategy is executed, representing an action node.

Control flow nodes, which are non-leaf nodes, serve to establish connections between nodes above and below. These nodes include Fallback, Sequence, and Parallel. Figure 3 showcases a subtree containing control flow nodes and execution nodes. We will sequentially introduce the roles of these nodes within the AUV adversarial behavior tree control structure.

Sequence "$\rightarrow$": Sequence nodes iterate sequentially until a failure is returned or all of their children succeed. Sequential nodes are commonly employed to guarantee the secure execution of tasks and are utilized in AUV adversarial tasks to continuously probe and update the state of the environment and the adversary. The Algorithm 1 description and pseudocode are provided below:

Algorithm 1 Sequence node of BTs

```
1. for child ∈ children do
2.     status = tick(child);
3.     if status = Running OR status = Failure then
4.         return status;
5.     end
6. end
7. return Success
```

Fallback "?": The Fallback node functions in the opposite manner to the Sequence node. Its logic is akin to the "or" operation in mathematics. When any child node within the Fallback node is executed and succeeds, the Fallback node will receive a success signal. In Figure 3, when the execution reaches the Fallback node, the child node assesses whether the targeting condition is met and executes one of the actions based on the condition, returning "success" to the Fallback node. The Algorithm 2 description and pseudo-code are provided below.

Algorithm 2 Fallback node of BTs

```
1. for child ∈ children do
2.     status = tick(child);
3.     if status = Running OR status = Success then
4.         return status;
5.     end
6. end
7. return Failure
```

Parallel "⇒": Although the Parallel node does not feature in the AUV subtree, it is a crucial category of control flow nodes in behavior trees. Parallel nodes execute all of their children concurrently. By default, Parallel nodes return "running" to the parent node that initiated them. They return a success if all children return a success, or a failure if any child returns a failure. The Algorithm 3 description and pseudo-code are outlined below.

Algorithm 3 Parallel node of BTs

```
1. for child ∈ children do
2.     status = tick(child);
3. end
4. if $\sum_i statuses(i) = Success \geq M$ textbfthen
5.     return success;
6. end
7. if $\sum_i statuses(i) = Failure > N - M$ textbfthen
8. end
9. return failure
```

Decorator nodes serve as a fusion of execution flow and control flow nodes. They define customized strategies for determining when the child node is evaluated, what the return value should be, or both.

4.2. Behavior Tree Formal Grammar

Before delving into the behavior tree construction algorithm outlined in this paper, let us briefly analyze the process of constructing behavior trees. The fundamental process of building the behavior tree for a multi-AUV adversarial control system is illustrated in Figure 4. Initially, the AUV condition and action library are established, with conditions and actions categorized based on the analysis of the adversarial task. Subsequently, the automatic design of the behavior trees is initiated.

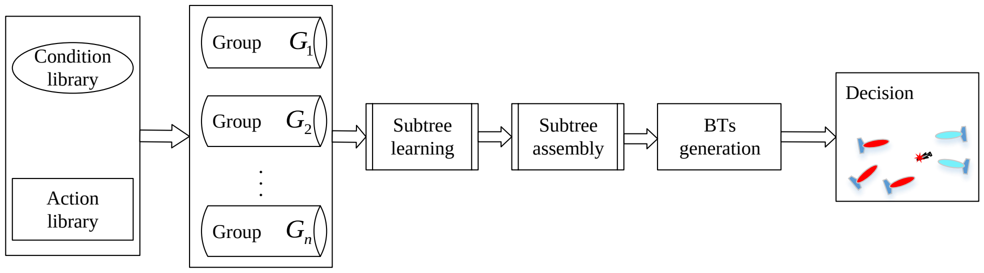


Figure 4. The basic process of behavior tree of autonomous underwater vehicle adversarial control system.

The grouping of action and state is an important part of the algorithm proposed in this paper, which can greatly reduce the search cost of the behavior tree [3]. Before grouping, the actions and conditions in the behavior tree are associated with the basic strategy and state of the AUV. The basic strategy set for defining an AUV is $A = \{a_1, a_2, a_3, a_4, a_5\}$, where the strategy in A corresponds one-to-one to the actions in the behavior tree. The state obtained by the sensor during the execution of the task corresponds to the conditions $C = [c_1, c_2, \cdots, c_n]$ of the behavior tree. We then group the states and actions according to the analysis based on the actual situation in the task: $g = [a_g, c_g]$ where $a_g \subseteq A$ is the strategy set, and $c_g \subseteq C$ is conditions.

Many different subtrees can be obtained through the search of actions and conditions. Designing the corresponding search algorithm to obtain the optimal subtree set $\mathrm{B} = \{b^1, b^2, \cdots, b^m\}$ is an important part of this section. The ultimate goal is to obtain the most efficient behavior tree through search and optimization, and use $\mathrm{B} = \bigcup b^*, b^* \subseteq \mathrm{B}$ which satisfies $\mathrm{B} = \max_{b \in B} f(b)$ as the control system for AUV complex tasks.

As shown in the Table 2, several sets of correspondence between condition and action are given. In other complex tasks, the correspondence between condition and action can also be preset according to the task requirements to reduce the optimization search space and accelerate the convergence of the algorithm.

Table 2. Examples of pre-grouped conditions and actions.

Condition	Action
Target is obstacles	Mobile_evasion
Non-maximum velocity	Accelerate
Heading is not satisfied	Adjust heading
In pursue range	Pursue

4.3. MCTS for Subtree Learning

We encode the behavior tree and devise a search algorithm based on Monte Carlo tree search (MCTS). The MCTS algorithm serves two key roles in the process of behavior tree generation: exploration and evaluation. In the context of the multi-AUV adversarial control behavior tree, optimizing the learning of subtrees corresponds to task optimization. For instance, the tasks of the AUV in this study encompass target search, threat estimation, maneuvering avoidance, and target pursuing, with the tree structure designed to include four subtrees. By quantifying the benefits of these four tasks in real-world scenarios, we can formulate a reward function that incentivizes MCTS to optimize the search process.

The MCTS algorithm consists of four key steps: selection, expansion, simulation, and backup, as depicted in Figure 5. In the context of the intricate AUV task, the MCTS search process involves selecting one of the four subtrees based on the current optimal return. The use of an isosceles trapezoid to represent subtrees in behavior trees (BTs) is common practice, as shown in Figure 5. This study incorporates four types of subtrees: the target search subtree (TS), threat estimation subtree (TE), maneuver evasion subtree (ME), and pursuing target subtree (PT). The relationship between these four processes and the generation of behavior trees is illustrated in Figure 5. Upon selecting the optimal exploration node, MCTS conducts the search and evaluation of the behavior tree on the subtree.

An important part of the MCTS algorithm is the evaluation of nodes, which is the basis of optimization. First, as shown in Equation (1), select the maneuver strategy with the greatest benefit according to the upper confidence bound for tree (UCT) algorithm, where $\overline{V_i}$ is the average return of the selection subtree s_i in the current state, n_i is the total number of experimental fields of the selection subtree s_i, and C is the super parameter used for adjustment, which can stimulate the search tree to search for more nodes with low selected times, so as to avoid the algorithm falling into local optimization. This step, that selecting is not only to select the current status but also to speculate and select the most potential node through the victory rate, lays the groundwork for subsequent selection.

$$UCT = \overline{V_i} + C\sqrt{\frac{\ln N}{n_i}} \tag{1}$$

For Equation (1), we can concretize the search and evaluation of the subtree by quantifying the adversarial situation. We can determine the calculation method of the draw profit value according to the antagonism rule, as shown in Equation (2).

$$\overline{V_i} = \sum_{i=1}^{N_R} \tilde{v}_i^{RR} L_i^R - \sum_{j=1}^{N_B} \tilde{v}_j^{RB} L_j^B \tag{2}$$

where N_R and N_B are the number of AUVs in the detection range of the red and blue opponents at this moment, respectively, N_R (N_B) is the cooperation (antagonism) value of the ith (jth) boat, and $\tilde{v}_i^{RR}$ ($\tilde{v}_j^{RB}$) is the survival probability (which changes in the event of an pursuing action). The other side calculates the same way.

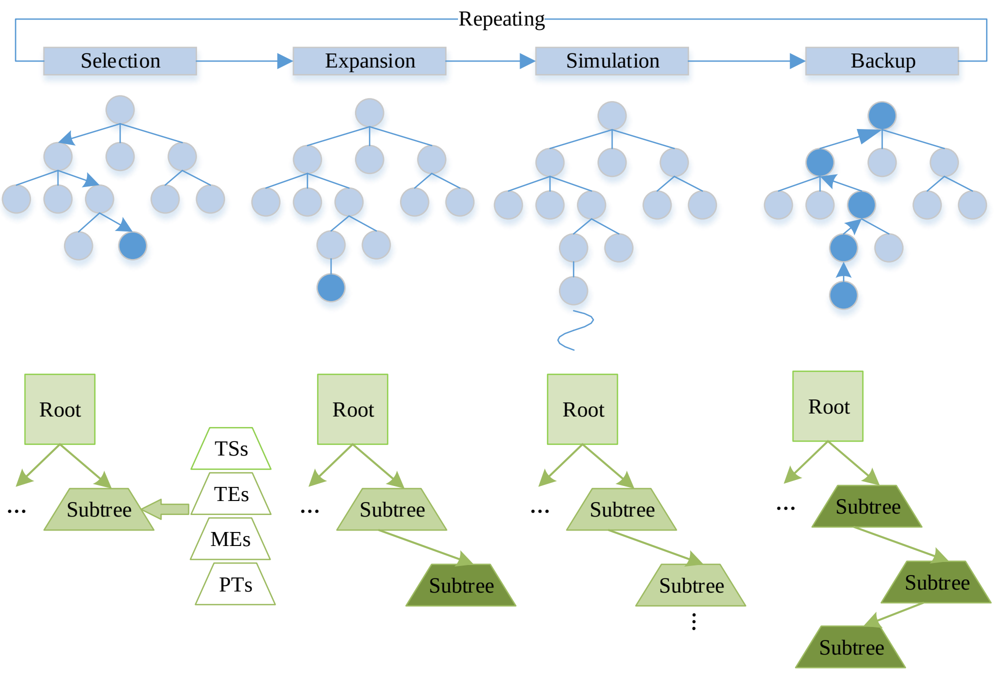


Figure 5. MCTS algorithm and BTs subtree search.

The MCTS algorithm proposed in this paper optimizes subtrees through the following steps:

Step 1: Initialize the number of AUVs on both sides and the four-seed tree structure.

Step 2: Construct an MCTS tree and initialize the root node of the input behavior tree.

Step 3: Commence iteration, reaching the maximum number of iterations, with the iteration process following the steps of selection, expansion, simulation, and backtracking.

Step 4: Update the subtree and optimize all the AUVs involved in the task based on the aforementioned steps.

By following the above algorithm, the optimal subtree can be obtained. The pseudocode of Algorithm 4 is as follows:

Algorithm 4 MCTS for subtree search frame

```
Input: AUVs number $N_R$, $N_B$,
subtree number $n_i$,
iterations $IterNum$,
initialized three types of subtrees: TSs, TEs, MEs, PTs
reward value $V_i$ calculation equation
1: for i ← 1 to $N_R$ or $N_B$ do
2:     Create a new tree with root node and initialize root:
3:     root. N←0, root. Q←0
4:     for j ← 1 to IterNum do
5:         Evaluate the draw profit value;
6:             while(True)
7:                 if $p \in N$ is leaf
8:             break;
9:                 end if
10:            find the best subtree of p and its index ind;
11:            $p$←best subtree of p;
12:            end while
13:        if $p$ is not leaf node
14:            Expand the node $p$;
15:        end if
16:        Simulation: Simulate returns according to the reward
17:        Back up
18:    end for
19: end for
Return optimized subtree
end
```

4.4. Optimization Algorithm for Subtree Fusion

We describe the space and state of the behavior tree in a more normative way [8]. Define the behavior tree $\mathrm{T}_i = \{f_i, r_i, \Delta t\}$ studied in this paper. f_i is the equation of state of the behavior tree, r_i is the return state after the operation of the action, and Δt is the clock signal of the node traversal in the behavior tree, in short, the time step. In the running state set r_i: $R^n \to \{\boldsymbol{S,F,R}\}$, all states belong to R^n, and different return states correspond to different fields in R^n. i in the formula represents the subtree index. Usually, a complex control system will evolve several different subtrees in the face of different tasks, and finally generate a complete tree. In the above state description, all subtrees can be deduced and evolved in the state space by defining $\mathrm{T}_i = \{f_i, r_i, \Delta t\}$. The integration of a tree of control nodes to execution nodes, as described in the behavior tree formal syntax mentioned above, also depends on these control nodes.

Definition 1. *(Sequence combination of subtrees): subtrees are fused using Sequence control nodes, and the state space of BT is described as:*

$$T_r = Sequence(T_m, T_n) \tag{3}$$

The same as the running rule of the execution node, if the subtree runs in order and the return success is returned to the Sequence node, the return failure is returned otherwise.

This combination is often applied to the combination of sequential tasks in AUV adversarial tasks. For example, if a BT needs to manually design a search–calibration–strike tree with the ultimate goal of strike, the Sequence node can be used to integrate the subtree when searching, and the target subtree and strike subtree are fused.

Definition 2. *(Fallback combination of subtrees): subtrees are fused using Fallback control nodes, and the state space of BT is described as:*

$$T_r = Fallback(T_m, T_n) \tag{4}$$

The process of integrating subtrees with Fallback nodes is also the same as that of connecting execution nodes. When a "success" status is returned in the subtree, the Fallback node receives "success".

There are many inherent properties in the structural design of a behavior tree, such as security, timeliness, and so on. After determining the functional structure of the tree, security, timeliness, and so on also need to be considered. In order to improve the security of AUV control, we propose a heuristic optimization algorithm to improve the robustness of BTs' structure with a fixed functional structure. Let us start by describing the timeliness and security of trees. These properties are also used to evaluate the performance of the behavior tree.

Definition 3. *(Timeliness of BTs): For any starting $x(0) \in R' \subset R$ of the tree, there is τ time limit and time point $\tau'(x(0))$, satisfied $\tau'(x(0)) \leq \tau$, and for any $t \in [0, \tau')$, satisfied $x(t) \in R'$, then the BTs are said to be a finite time success in the neighborhood R', and the time limit is used to describe the timeliness of BTs.*

Definition 4. *(Security of BTs:) If for any initial state $x(0) \in I$, and time t, the non-secure domain $O \subset R^n$ is initialized, the initial domain $I \subset R$ is initialized, if $x(t) \notin O$ is true, then BTs are safe.*

On the premise of defining timeliness and security, the subtree fusion is analyzed and calculated. The optimization objective function is given, and the optimization objective function of the automatic generation behavior tree is designed from three perspectives, which are task benefit M_r, behavior tree time benefit T_r, and behavior tree security benefit S_r. The optimization objective function can be obtained as follows:

$$f(tr_{sub}) = M_r + T_r + S_r \tag{5}$$

where M_r is related to the task type and T_r is related to the search time from the current node to the leaf node according to Definition 3. The specific form is as follows:

$$T_r = t_a + t_c + t_j \tag{6}$$

where t_a, t_c, t_j are the final action execution time, the running time of the control node in the behavior tree, and the state judgment time, respectively. t_j is the security benefit of the behavior tree, which can be obtained according to Definition 4. Security is the maximum requirement for the behavior tree, so t_j can be obtained in the form:

$$S_r = \begin{cases} 1 & \text{if Tree is safe} \\ -\infty & \text{otherwise} \end{cases} \tag{7}$$

Although the generation process of the subtree has been optimized by the MCTS algorithm, the optimized subtree combination may not meet the performance maximization. For this process, the QPSO [50] algorithm is proposed to fuse into the tree integration, generate particles from the root node, take the subtree in the connection process as the optimization path, and, finally, obtain the optimal behavior tree. Particles with quantum behavior can appear at any position in the whole feasible solution space, and the position of occurrence is determined by the particle updating probability; this position may have a better fitness value than the population optimal particle in the current population, so the algorithm can achieve the best solution in the feasible solution space. The algorithm principle is as follows:

$$mbest^k = \frac{1}{N_P} \sum_{t=1}^{N_P} pbest_t^k \tag{8}$$

$$P_t^k = r \cdot pbest_t^k + (1-r) gbest^k \tag{9}$$

$$X_t^{k+1} = P_t \pm \alpha |mbest^k - X_t^k| \ln(\frac{1}{u}) \tag{10}$$

where X_t^k is the position of the particle at the k^{th} iteration, $mbest^k$ is the middle position of the particle swarm at the k^{th} iteration, and r and u are random numbers between $[0,1]$. The optimal value of the t^{th} particle is $pbest_t$, and the optimal solution of the whole population is $gbest$. The basic steps of the algorithm are as follows:

(1) Initialize the position information of particles and determine the particle population size N_p and particle dimension $\sum m_i$.
(2) Calculate the *mbest* value of the middle position of the particle swarm.
(3) Calculate the fitness of each particle, and select the particle with the optimal fitness value as the optimal particle $pbest_i$.
(4) The fitness values of all are compared, and the particle with the best fitness value is selected as the global optimal particle *gbest*.
(5) For each dimension with particles, a random point is obtained between *gbest* and $pbest_i$.
(6) Obtain a new position of particle.
(7) Check whether the t^{th} particle meets the limit condition $X_t^{k+1} > 0$, otherwise, solve the control step and make it so that

$$X_t^{k+1} = P_t \pm \beta_t \cdot \alpha |mbest^k - X_t^k| \ln(\frac{1}{u}) \tag{11}$$

and make X_t^{k+1} return to the feasible mixed strategy space.
(8) Repeat steps 2–7 until the algorithm reaches the accuracy standard or the maximum number of iterations, and output the global optimal particle position and its fitness.

In summary, the subtree search learning and behavior tree integration optimization are all completed. The design of the automatic generation algorithm of the AUV control system behavior tree is completed through two ways of node value evaluation and behavior tree performance calculation. Next, we will demonstrate the effectiveness of the algorithm through simulation experiments and design control experiments to verify the superiority of the algorithm.

5. Simulation Experiment

To validate the efficacy of the proposed algorithm for designing the control system in AUV complex tasks, we conducted design validity verification experiments and superiority comparative analysis experiments against existing algorithms. The simulation platform was developed using an AUV emulation platform, operating on Ubuntu 18.04.5, ROS 1.14.11, and Gazebo 9.19.0. Figure 6 illustrates the control flow of the multi-AUV complex task control process verification platform in this study. The control instructions generated based on the task-designed behavior tree are analyzed by the controller and subsequently transmitted to the simulated AUV. Specifically, the logical structure of the behavior tree is stored by the behavior tree structure design software Groot in ROS, and the basic functions are completed by the function definition of the Condition node and Action node in the behavior tree. The control instructions are sent to the AUV controller through the internal communication of the computer. The AUV controller this study used is the incremental PID controller, with speed controller PID parameters for $K_{p1} = 15$, $K_{i1} = 1.5$, and $K_{d1} = 1.2$, and heading controller PID parameters as follows: K_{p2} =20, $K_{i2} = 0.2$, $K_{d2} = 0.8$, ultimately acting through the internal communication between the controller and Gazebo, which complete the connection.

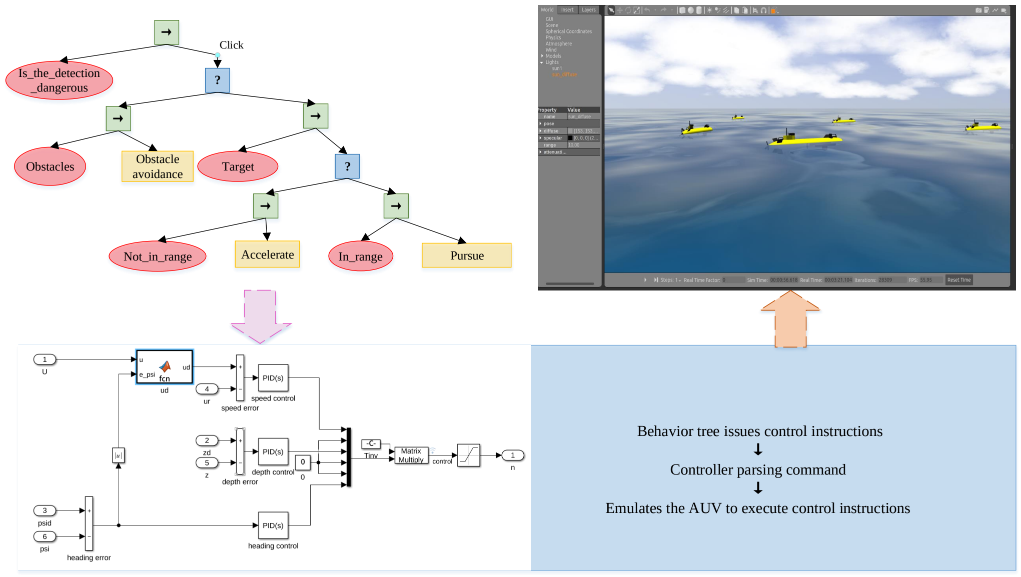
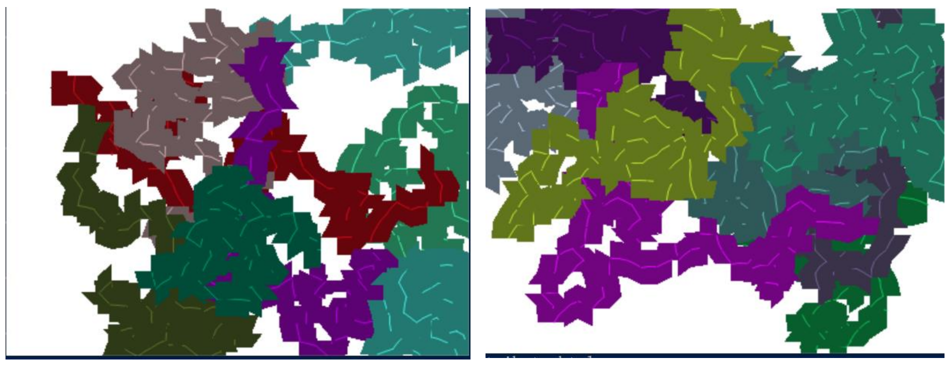


Figure 6. Multi-AUV complex task simulation platform.

5.1. Verification of Multi-AUV Cooperative Task Effect

There are many different decision control structures under different tasks in AUV cluster cooperation. In this summary, according to the automatic generation algorithm of the behavior tree designed above and the scenario of the AUV cooperative task, the optimization algorithm of the automatic generation of the behavior tree is analyzed. Based on the cooperative task decision in the above, the multi-AUV tasks can be divided into: cooperative search, cooperative tracking, and cooperative hunting for different task requirements. Taking the cooperative search task as an example, the behavior tree is designed to meet the decision control of the AUV based on the consideration of the cooperative relationship between AUVs.

It can be seen from the structure of the behavior tree that the optimization algorithm of the behavior tree further optimizes the repetition of state judgment and the redundancy of behavior actions, and improves the simplicity of the behavior tree. Quantitative analysis is described in detail in the next section. Finally, by randomly generating the initial context of the multi-AUV search task, the strategy control effect of the multi-AUV collaborative search behavior tree is tested. Compared with the word "Z" search algorithm, under the same search time, the search repetition area between AUVs is reduced by 7.82% on average, the search coverage reaches 95%, and the average increase is 12.79%. Taking the search coverage experiment of 6 kinds of underwater vehicles as an example, the search results in the same area are shown in Figure 7. The different colors in the figure indicate the range of search areas for different AUVs.



(a)Z-shaped conventional AUV search area (b) AUV region search by BTs

Figure 7. Comparative experimental results of region search.

5.2. Algorithm Effectiveness Analysis

In order to verify the effectiveness of the MCTS-QPSO algorithm in behavior tree design generation, we preset the AUV against the task, manually designed a behavior tree to meet the task requirements, and compared the performance of the manually designed BTs with that generated by the automatic designed behavior tree algorithm in this paper.

A manually designed behavior tree for the AUV complex task is shown in Figure 8. To envision a scenario where a threat is detected in the mission area, you must first determine the threat type and make a choice based on the threat type and your own posture. Based on this manually designed behavior tree, we analyze its performance and safety. As mentioned above, time efficiency is described in terms of finite time limits. We analyze the time efficiency of manually designed and automatically generated BTs in terms of the control subtree of the process from discovering the target to determining the target type to executing the action. The search signal "click" in the behavior tree is defined as time t_c, the judgment behavior in the tree: "Judge the pursue range" and "check the capacity" and "judge target type" are, respectively, T_1, T_2, and T_3, which condition judgment represented as ovals in the figure, respectively, and the actions in the tree are: "evasion" and "pursue", whose action time is t_1, t_2, respectively, which defined the above times as finite constants. As shown in Figure 8, we analyze the time limits of the labeled nodes to analyze the time efficiency of the behavior tree generated by the proposed algorithm and the manually established behavior tree. In this node, the threat type is determined as the AUV of the other party, and the control flow of the pursuing strategy is executed for analysis. As shown in Figure 8, the time limit for manually designing the behavior tree according to the above definition is:

$$\tau_m = T_1 + 2T_2 + T_3 + 10t_c + t_2 \tag{12}$$

The time limit of the tree generated based on the MCTS-QPSO algorithm is:

$$\tau_a = T_1 + 2T_2 + T_3 + 11t_c + t_2 \tag{13}$$

The difference t_c between the two can be obtained by comparison, but $t_c \ll T_i$ and $t_c \ll t_i$ can be obtained by analysis. Therefore, the behavior tree generated by the proposed algorithm in this paper is slightly inferior to the manually generated BTs in terms of time efficiency. However, in the actual task, the time difference can be ignored, and it can be considered that the performance of the automatically generated behavior tree in time efficiency meets the requirements.

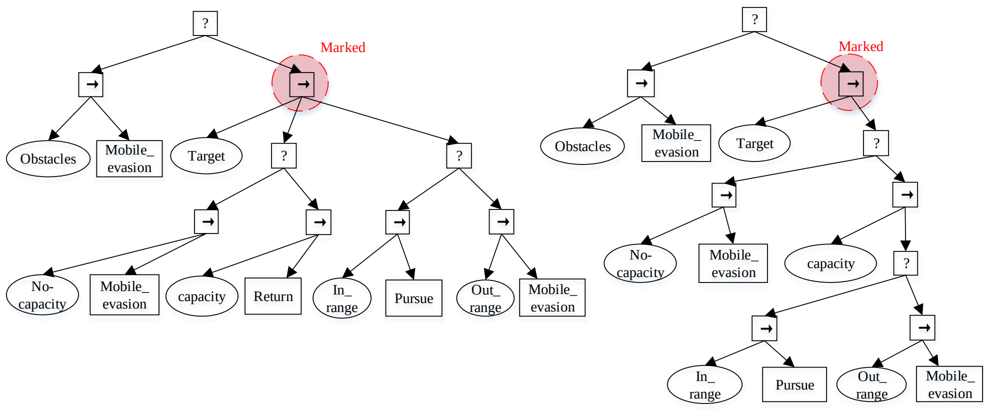


Figure 8. The comparison between automatically generated behavior tree (**left**) and manually designed behavior tree structure (**right**).

Next, analyze the security of the behavior tree. The definition of security can be interpreted as that all actions can be executed by determining the security domain and the obstacle domain before execution. As can be seen from the structure of the tree, controlling the flow of information in the book ensures the security of the control.

5.3. Algorithm Superiority Analysis

The essence of the automatic generation algorithm of the behavior tree is a multi-branch tree search. According to the analysis of the multi-branch tree, the time complexity of the search algorithm with limited depth is $O(b * d)$, and the space complexity is $O(d)$, where b is the node branch tree and d is the depth of the tree. Several controlled experiments were carefully designed to show the superiority of the algorithm. The salient feature of the proposed algorithm is the pre-grouping of conditions and actions in the behavior tree. In order to verify the impact of grouping behavior on the automatic design of behavior tree algorithms, a comparison was made in the scenario without pre-grouping and benchmarked against MCTS and QPSO.

As depicted in Figures 9 and 10, manual design impeccably aligns with established task requirements without necessitating iterative alterations, serving as the yardstick for evaluating optimal solutions (normalized income criteria). In Figures 9 and 10, the use of distinct colors for ✰ aids in clear visualization. Analyzing AUVs based on the behavior tree design for the overarching task goal reveals that only the algorithm proposed in this study can achieve equivalent task benefits to manual design by optimizing search iteratively. The "No-Group" control experiment yields a behavior tree structure with comparable benefits, yet the absence of pre-grouping leads to an abundance of particles in the solution space, escalating iteration count and hindering swift convergence. The proposed algorithm exhibits nearly 30% faster convergence compared to similar algorithms. The MCTS algorithm stands out for its rapid convergence by searching and amalgamating optimal subtrees. Despite not yielding the optimal benefits, the AUV antagonism task designed in this paper is adept at crafting control structures for simple tasks involving multiple agents. Surprisingly, the QPSO algorithm, relying solely on random search, also demonstrates commendable convergence speed, underscoring the pivotal role of pre-grouping in behavior tree automatic design.

Upon task decomposition, the automatic design of AUV behavior trees for patrol and trap roles exemplifies the algorithm's performance when reducing tree depth. Figures 10 and 11 underscore that presetting roles streamlines behavior tree generation, with the MCTS algorithm excelling in this aspect. It is evident that nearly all algorithms meet predetermined requirements when roles are preset. Delving into the analysis of complex task reconstruction, where the control behavior tree for AUV strategy with a singular goal is designed using four comparative algorithms, reveals intriguing insights. Upon completion of search patrol, target tracking, and target targeting, task reconstruction becomes imperative. Given the AUV's solitary targeting capability, a heuristic dictates task termination post direct capacity analysis. The performance of the four algorithms is depicted in Figure 12. In the MCTS-QPSO, MCTS, and QPSO algorithms, actions and conditions are pre-grouped to ensure that only essential condition analyses pertinent to actions are executed beforehand. Consequently, only one condition analysis precedes the actions highlighted in the blue circle. Notably, a capacity analysis conducted by the MCTS-QPSO algorithm before task reconstruction minimizes the count of action and condition analyses from the root node to the end node, underscoring commendable algorithm efficiency. Moreover, it is discernible that QPSO and MCTS exhibit some shortcomings in automatic behavior tree design. In the No-Group experiment set, the number of action nodes mirrors that of MCTS-QPSO, with redundant condition analyses preceding actions. This further validates the necessity of pre-grouping conditions and actions in behavior tree automatic design algorithms.

In light of the aforementioned analyses, the MCTS-QPSO behavior tree automatic search and generation algorithm proposed in this study excels in constructing complex control structures for AUV complex tasks. Not only does it approximate the income and structure of manual design calculations, but it also outperforms in terms of convergence efficiency.

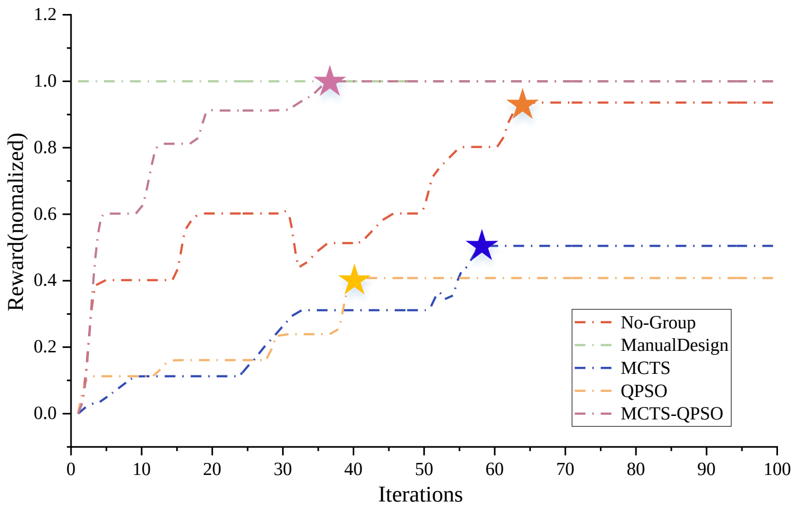


Figure 9. Comparison of complete task behavior tree design algorithms.

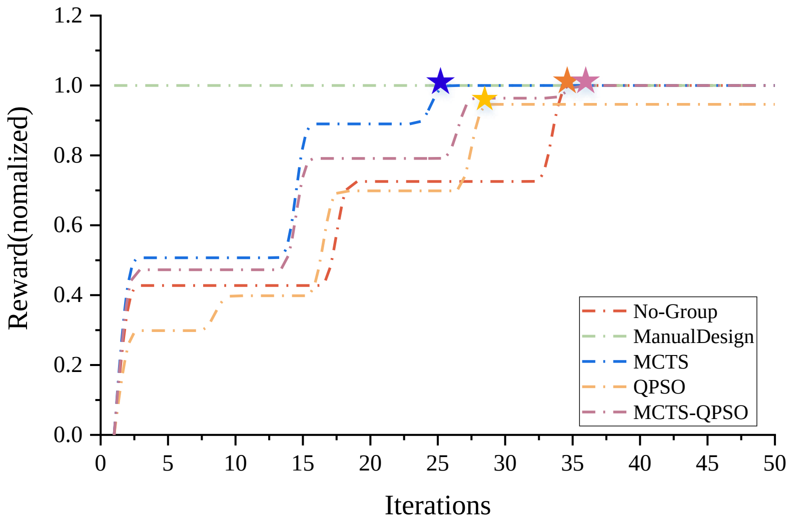


Figure 10. Comparison of patrol and tracking task behavior tree algorithms.

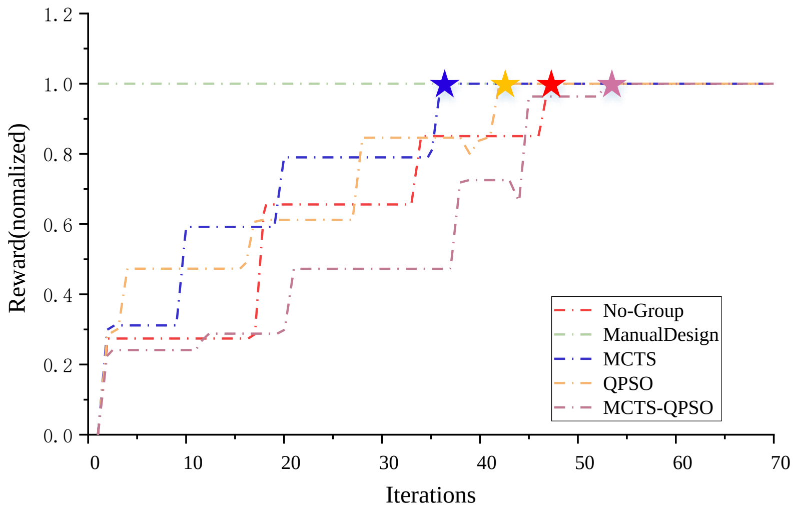


Figure 11. Comparison of patrol task behavior tree algorithm.

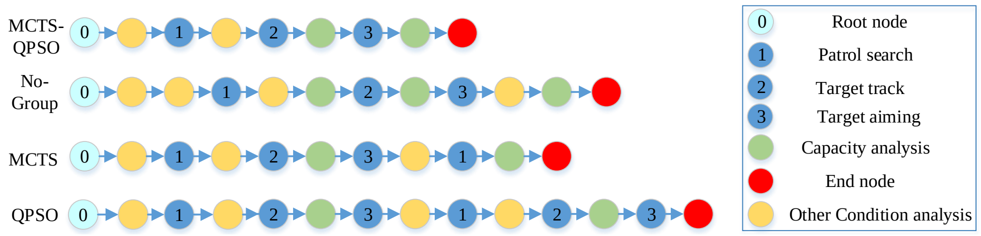


Figure 12. Simplified graph of behavior tree nodes.

6. Conclusions and Future Work

In this study, we have demonstrated the capability of the MCTS-QPSO algorithm to automatically design and generate the necessary behavior tree framework for complex multi-AUV tasks. The experiments conducted validate the three main contributions proposed in this paper, providing robust support for the automatic generation of behavior trees.

By pre-grouping conditions and actions based on prior knowledge, the algorithm effectively reduces the optimal solution space and enhances the convergence speed of behavior tree generation optimization. The algorithm's superiority is further substantiated through simulation experiments. Leveraging the inherent tree structure of behavior trees, we can readily explore the optimal structure of subtrees using MCTS to derive the most effective subtree configurations. By optimizing the nodes of the behavior tree through the design of task-specific reward functions, the algorithm can identify the optimal combination of subtrees, akin to solving an optimal subtree combination path from the root of the behavior tree. Through iterative optimization using the QPSO algorithm, a structurally optimal behavior tree aligning with the fundamental properties of behavior trees can be obtained.

In conclusion, the proposed algorithm facilitates the automatic generation of an AUV control system for AUV countermeasure tasks, streamlining the workload of experimental personnel involved in AUV strategy control design. Moving forward, our future research will focus on optimizing the online utilization of behavior trees and refining behavior tree structures in real-world scenarios through practical experimentation to enhance their applicability.

Author Contributions: D.Y., H.W. and X.C. designed the study, performed the research, analyzed data, and wrote the paper. Z.W., J.R. and K.Z. contributed to refining the ideas, carrying out additional analyses, and finalizing this paper. All authors have read and agreed to the published version of the manuscript.

Funding: This research work is supported by the National Science and Technology Innovation Special Zone Project (21-163-05-ZT-002-005-03), the National Key Laboratory of Underwater Robot Technology Fund (No. JCKYS2022SXJQR-09), and a special program to guide high-level scientific research (No. 3072022QBZ0403).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Wei, W.; Wang, J.; Fang, Z.; Chen, J.; Ren, Y.; Dong, Y. 3U: Joint design of uav-usv-uuv networks for cooperative target hunting. *IEEE Trans. Veh. Technol.* **2023**, *72*, 4085–4090. [CrossRef]
2. Lin, C.; Cheng, Y.; Wang, X.; Yuan, J.; Wang, G. Transformer-based dual-channel self-attention for uuv autonomous collision avoidance. *IEEE Trans. Intell. Veh.* **2023**, *8*, 2319–2331. [CrossRef]
3. Scheide, E.; Best, G.; Hollinger, G.A. Behavior tree learning for robotic task planning through monte carlo dag search over a formal grammar. In Proceedings of the 2021 IEEE International Conference on Robotics and Automation (ICRA), Xi'an, China, 30 May–5 June 2021; pp. 4837–4843. [CrossRef]

4. Yu, D.; Wang, H.; Li, B.; Wang, Z.; Ren, J.; Wang, X. Prometheebased multi-auv threat assessment method using combinational weights. *J. Mar. Sci. Eng.* **2023**, *11*, 1422. [CrossRef]
5. Ligot, A.; Kuckling, J.; Bozhinoski, D.; Birattari, M. Automatic modular design of robot swarms using behavior trees as a control architecture. *PeerJ Comput. Sci.* **2020**, *6*, e314. [CrossRef] [PubMed]
6. Birattari, M.; Ligot, A.; Bozhinoski, D.; Brambilla, M.; Francesca, G.; Garattoni, L.; Ramos, D.G.; Hasselmann, K.; Kegeleirs, M.; Kuckling, J.; et al. Automatic off-line design of robot swarms: A anifesto. *Front. Robot. AI* **2019**, *6*, 59. [CrossRef] [PubMed]
7. Francesca, G.; Birattari, M. Automatic design of robot swarms: Achievements and challenges. *Front. Robot. AI* **2016**, *3*, 29. [CrossRef]
8. Masek, M.; Lam, C.P.; Kelly, L.; Wong, M. Discovering optimal strategy in tactical combat scenarios through the evolution of behaviour trees. *Ann. Oper. Res.* **2023**, *320*, 901–936. [CrossRef]
9. Sprague, C.I.; Özkahraman, Ö.; Munafo, A.; Marlow, R.; Phillips, A.; Ögren, P. Improving the modularity of auv control systems using behaviour trees. In Proceedings of the 2018 IEEE/OES Autonomous Underwater Vehicle Workshop (AUV), Porto, Portugal, 6–9 November 2018; pp. 1–6. [CrossRef]
10. Colledanchise, M.; Gren, P. How behavior trees generalize the teleoreactive paradigm and and-or-trees. In Proceedings of the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, Republic of Korea, 9–14 October 2016; pp. 424–429. [CrossRef]
11. Malviya, V.; Reddy, A.K.; Kala, R. Autonomous social robot navigation using a behavioral finite state social machine. *Robotica* **2020**, *38*, 2266–2289. [CrossRef]
12. Yan, Y.; Ma, W.; Li, Y.; Wong, S.; He, P.; Zhu, S.; Yin, X. The navigation of mobile robot in the indoor dynamic unknown environment based on de- cision tree algorithm. *Comput. Intell. Neurosci.* **2022**, *2022*, 3492175. [CrossRef]
13. Browne, C.B.; Powley, E.; Whitehouse, D.; Lucas, S.M.; Cowling, P.I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; Colton, S. A survey of monte carlo tree search methods. *IEEE Trans. Comput. Intell. Games* **2012**, *4*, 1–43. [CrossRef]
14. Shen, G.; Lei, L.; Zhang, X.; Li, Z.; Cai, S.; Zhang, L. Multi-uav cooperative search based on reinforcement learning with a digital twin driven training framework. *IEEE Trans. Veh. Technol.* **2023**, *72*, 8354–8368. [CrossRef]
15. Pandey, P.; Pompili, D.; Yi, J. Dynamic collaboration between networked robots and clouds in resource-constrained environments. *IEEE Trans. Autom. Sci. Eng.* **2015**, *12*, 471–480. [CrossRef]
16. Perera, L.P.; Carvalho, J.P.; Guedes Soares, C. Intelligent ocean navigation and fuzzy-bayesian decision/action formulation. *IEEE J. Ocean. Eng.* **2012**, *37*, 204–219. [CrossRef]
17. Brito, M.P.; Griffiths, G. A markov chain state transition approach to establishing critical phases for auv reliability. *IEEE J. Ocean. Eng.* **2011**, *36*, 139–149. [CrossRef]
18. Abbasi, A.; MahmoudZadeh, S.; Yazdani, A. A cooperative dynamic task assignment framework for cotsbot auvs. *IEEE Trans. Autom. Sci. Eng.* **2022**, *19*, 1163–1179. [CrossRef]
19. Bhatt, E.C.; Howard, B.; Schmidt, H. An embedded tactical decision aid framework for environmentally adaptive autonomous underwater vehi- cle communication and navigation. *IEEE J. Ocean. Eng.* **2022**, *47*, 848–863. [CrossRef]
20. Pan, Y.; Ma, B.; Tang, J.; Zeng, Y. Behavioral model summarisation for other agents under uncertainty. *Inf. Sci.* **2022**, *582*, 495–508. [CrossRef]
21. Chang, Y.; Garcia, A.; Wang, Z.; Sun, L. Structural estimation of partially observable markov decision processes. *IEEE Trans. Autom. Control* **2023**, *68*, 5135–5141. [CrossRef]
22. Doshi, P.; Zeng, Y.; Chen, Q. Graphical models for interactive pomdps: Representations and solutions. *Auton. Agents Multi-Agent Syst.* **2009**, *18*, 376–416. [CrossRef]
23. Pan, Y.; Ma, B.; Zeng, Y.; Tang, J.; Zeng, B.; Ming, Z. An evolutionary framework for modelling unknown behaviours of other agents. *IEEE Trans. Emerg. Top. Comput. Intell.* **2023**, *7*, 1276–1289. [CrossRef]
24. Ostonov, A.; Moshkov, M. On complexity of deterministic and nondeterministic decision trees for conventional decision tables from closed classes. *Entropy* **2023**, *25*, 1411. [CrossRef] [PubMed]
25. Yan, Y.; Deng, H.; Yue, J.; Chen, Z. Model-erence adaptive control of finite state machines with respect to states: A matrix-based approach. *IEEE Trans. Circuits Syst. II Express Briefs* **2023**, *70*, 2171–2175. [CrossRef]
26. Gugliermo, S.; Schaffernicht, E.; Koniaris, C.; Pecora, F. Learning behavior trees from planning experts using decision tree and logic factorization. *IEEE Robot. Autom. Lett.* **2023**, *8*, 3534–3541. [CrossRef]
27. Nicolau, M.; Perez-Liebana, D.; Neill, M.O.; Brabazon, A. Evolutionary behavior tree approaches for navigating platform games. *IEEE Trans. Comput. Intell. AI Games* **2017**, *9*, 227–238. [CrossRef]
28. Dortmans, E.; Punter, T. Behavior trees for smart robots practical guidelines for robot software development. *J. Robot.* **2022**, *2022*, 3314084. [CrossRef]
29. Abiyev, R.H.; Akkaya, N.; Aytac, E.; Ibrahim, D. Behaviour tree based control for efficient navigation of holonomic robots. *Int. J. Robot. Autom.* **2014**, *29*, 44–57. [CrossRef]
30. Bhat, S.; Stenius, I. Controlling an underactuated auv as an inverted pendulum using nonlinear model predictive control and behavior trees. In Proceedings of the 2023 IEEE International Conference on Robotics and Automation(ICRA), London, UK, 29 May–2 June 2023; pp. 12261–12267. [CrossRef]
31. Iovino, M.; Scukins, E.; Styrud, J.; Ögren, P.; Smith, C. A survey of behavior trees in robotics and ai. *Robot. Auton. Syst.* **2022**, *154*, 104096. [CrossRef]

32. Scheper, K.Y.W.; Tijmons, S.; de Visser, C.C.; de Croon, G.C.H.E. Behavior Trees for Evolutionary Robotics. *Artif. Life* **2016**, *22*, 23–48. [CrossRef] [PubMed]
33. Kuckling, J.; Ligot, A.; Bozhinoski, D.; Birattari, M. Behavior trees as a control architecture in the automatic modular design of robot swarms. In *Swarm Intelligence*; Springer International Publishing: Cham, Swizerland, 2018; pp. 30–43.
34. Yao, J.; Wang, W.; Li, Z.; Lei, Y.; Li, Q. Tactics exploration framework based on genetic programming. *Int. J. Comput. Intell. Syst.* **2017**, *10*, 804–814. [CrossRef]
35. Colledanchise, M.; Parasuraman, R.; Ögren, P. Learning of behavior trees for autonomous agents. *IEEE Trans. Games* **2019**, *11*, 183–189. [CrossRef]
36. Venkata, S.S.O.; Parasuraman, R.; Pidaparti, R. Kt-bt: A framework for knowledge transfer through behavior trees in multirobot systems. *IEEE Trans. Robot.* **2023**, *39*, 4114–4130. [CrossRef]
37. French, K.; Wu, S.; Pan, T.; Zhou, Z.; Jenkins, O.C. Learning behavior trees from demonstration. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 7791–7797. [CrossRef]
38. Sprague, C.I.; Ögren, P. Adding Neural Network Controllers to Behavior Trees without Destroying Performance Guarantees. In Proceedings of the 2022 IEEE 61st Conference on Decision and Control (CDC), Cancun, Mexico, 6–9 December 2022; pp. 3989–3996. [CrossRef]
39. Hólzl, M.; Gabor, T. Reasoning and Learning for Awareness and Adaptation. In *Software Engineering for Collective Autonomic Systems*; Springer International Publishing: Cham, Swizerlalnd, 2015; pp. 249–290. [CrossRef]
40. Dey, R.; Child, C. Ql-bt: Enhancing behaviour tree design and implementation with q-learning. In Proceedings of the 2013 IEEE Conference on Computational Inteligence in Games (CIG), Niagara Falls, ON, Canada, 11–13 August 2013; pp. 1–8. [CrossRef]
41. Hoffman, M.; Song, E.; Brundage, M.; Kumara, S. Online Maintenance Prioritization Via Monte Carlo Tree Search and Case Based Reasoning. *J. Comput. Inf. Sci. Eng.* **2022**, *22*, 041005. [CrossRef]
42. Chiu, T.-Y.; Ny, J.L.; David, J.-P. Temporal logic explanations for dynamic decision systems using anchors and monte carlo tree search. *Artif. Intell.* **2023**, *318*, 103897. [CrossRef]
43. Seiler, K.M.; Palmer, A.W.; Hill, A.J. Flow-achieving online planning and dispatching for continuous transportation with autonomous vehicles. *IEEE Trans. Autom. Sci. Eng.* **2022**, *19*, 457–472. [CrossRef]
44. Swiechowski, M.; Godlewski, K.; Sawicki, B.; Mandziuk, J. Monte carlo tree search: A review of recent modifications and applications. *Arti Ficial Intell. Rev.* **2023**, 56, 2497–2562. [CrossRef]
45. Yu, D.; Wang, H.; Huang, W.; Huang, S. Application of extended game in multi-uuv pursuit-escape task. In Proceedings of the Ocean, Offshore and Arctic Engineering, Melbourne, VIC, Australia, 11–16 June 2023; Volume 5. [CrossRef]
46. Dorling, K.; Heinrichs, J.; Messier, G.G.; Magierowski, S. Vehicle routing problems for drone delivery. *IEEE Trans. Syst. Man Cybern. Syst.* **2017**, *47*, 70–85. [CrossRef]
47. Sun, B.; Ma, H.; Zhu, D. A fusion designed improved elastic potential field method in auv underwater target interception. *IEEE J. Ocean. Eng.* **2023**, *48*, 640–648. [CrossRef]
48. Ögren, P.; Sprague, C.I. Behavior trees in robot control systems. *Annu. Rev. Control. Robot. Auton. Syst.* **2022**, *5*, 81–107. [CrossRef]
49. Özkahraman, O.; Ögren, P. Combining control barrier functions and behavior trees for multi-agent underwater coverage missions. In Proceedings of the 2020 59th IEEE Conference on Decision and Control, Jeju, Republic of Korea, 14–18 December 2020; pp. 5275–5282. [CrossRef]
50. Fu, Y.; Ding, M.; Zhou, C. Phase angle-encoded and quantum-behaved particle swarm optimization applied to three-dimensional route planning for UAV. *IEEE Trans. Syst. Man Cybern.-Part A Syst. Hum.* **2011**, *42*, 511–526. [CrossRef]