Research on Decentralized Task Allocation and Collaboration Based on Multiple AUVs

Minjie Xia

College of Ocean Engineering, Harbin Institute of Technology, Weihai, 264200, China

Minjie.xia.2021@uni.strath.ac.uk

Keywords: Decentralized Task Allocation and Collaboration; Multi-AUV systems； Maritime activities; Intelligent optimization

Abstract: In the 21st century, extensive human engagement in activities such as marine resource exploration, maritime transportation, and maritime security has led to a deeper understanding of the oceans. This increased interest has spotlighted Autonomous Underwater Vehicles (AUVs), which possess the capability to operate independently and covertly, making them a preferred choice for diverse underwater tasks. However, as underwater missions grow in complexity, the limitations of individual AUVs have spurred the exploration of multi-AUV systems. These systems offer higher efficiency, greater intelligence, and enhanced fault tolerance, rendering them indispensable in both deep-sea search and rescue missions and national maritime security efforts. Task allocation is a central challenge in the realm of multi-AUV systems. Balancing factors like search environments, ocean currents, obstacles, and target locations, various intelligent optimization methods have been employed. This paper references representative works to explore genetic algorithm-based decentralized task allocation and Hungarian algorithm-based decentralized task allocation, examining their potential to enhance cooperative underwater searches. Centralized task allocation faces robustness and scalability concerns, necessitating decentralized alternatives, particularly for mobile multi-agent systems. The presented research underscores the significance of decentralized methodologies in optimizing multi-AUV task assignments. The paper elucidates these two approaches, discusses their implications, and concludes by presenting an analysis of the results. Through this study, the paper contributes to advancing the field of multi-AUV systems and their effective coordination for efficient underwater search operations.

1. Introduction

Since the 21st century, human engagement in a range of activities including marine resource exploration, maritime transportation and maritime security has facilitated a deeper comprehension of the oceans. Particularly within these domains, Autonomous Underwater Vehicles (AUVs) have gained significant attention from scholars both domestically and internationally. AUVs operate independently of external energy sources and require no physical connection to mother ships, showcasing exceptional concealment capabilities and an expansive search range. Whether in civilian or naval contexts, AUVs exhibit broad potential applications, especially as the virtually exclusive option for deep-sea underwater search missions. Nonetheless, as the complexity of AUV search tasks continues to mount, the limitations of individual AUV functionalities have led to the prominence of multi-AUV systems as a pivotal research direction within the field of underwater robotics. In comparison to single AUV systems, multi-AUV systems offer diversified solutions, accompanied by elevated work efficiency, heightened levels of intelligence, and enhanced fault tolerance. The collaborative search facilitated by multi-AUV systems bears indispensable and irreplaceable significance in civilian deep-sea search and rescue endeavors, as well as in the establishment of national maritime security.

Currently, the challenge of task allocation remains a significant obstacle for multi-AUV systems. It necessitates a comprehensive consideration of various factors such as the underwater search environment, ocean current magnitude and direction, obstacles, restricted zones, and the central position of the target to judiciously allocate different areas awaiting exploration among individual AUVs.

Task allocation algorithms can be categorized into centralized and distributed approaches [1]. Due to recognized limitations in robustness and scalability, the need for developing decentralized task allocation algorithms has emerged, particularly in the context of multi-agent systems. In a centralized setup, a central planner possessing comprehensive information allocates tasks to all agents based on specific requirements. However, this approach disregards the potential computational capabilities of individual agents and exposes the system to security vulnerabilities. If communication between the central planner and agents encounters issues, tasks might remain unassigned, leading to incomplete assignments. In contrast, within a decentralized framework, each agent independently determines its task allocation using its own computational resources [2]. This approach eliminates the dependency on a single central planner and enhances the system's resilience to breakdowns in communication. Agents communicate among themselves to resolve conflicts and enhance their allocations. In the distributed mindset of task allocation, subsystems lack complete information about the entire system, and task allocation is a negotiated process involving multiple robots. Furthermore, these robots independently complete task allocation based on their perception of the environment. The application of market mechanism-based methods is widespread, with task allocation based on market mechanisms relying on negotiation among robots through certain protocols to ultimately achieve task distribution.

In paper [3], the authors present the Consensus Based Parallel Auction and Execution (CBPAE) algorithm. This algorithm is designed as a distributed, market-based approach for multi-robot task allocation (MRTA). Its primary focus is on dynamically allocating tasks, specifically addressing the task allocation challenges encountered within a Multi-Robot System (MRS) operating within a healthcare environment, such as a care home. Amanda et al. introduce refinements to the distributed Performance Impact (PI) algorithm in their work. They also showcase the outcomes of their experimental trials, highlighting how their contributions propel the advancement of the current state-of-the-art. Specifically, their enhancements prove significant in the realm of single-task, single-robot, time-extended, multi-agent task assignment, especially for missions with time-critical requirements [4]. In the context of the paper [5], a team of researchers introduces BnB FMS, a distributed algorithm tailored for task allocation. This algorithm ingeniously combines online domain pruning with fast-max-sum techniques. Moreover, it strategically integrates branch-and-bound search during the computation of messages that are to be transmitted by a function. This proactive approach facilitates the pruning of the state space, thereby curtailing the portions that need to be explored more extensively. Patel and colleagues present a pioneering methodology for decentralized multi-agent collaborative search within the domain of task allocation. Their approach leverages a decentralized genetic algorithm, introducing a novel way to address task allocation challenges [6]. In a similar vein, Choi and Kim introduced a two-stage genetic algorithm-centered approach for decentralized allocation. Here, each agent utilizes a genetic algorithm to define its task sequence and subsequently engages in communication with other agents to optimize costs through task exchange [7]. The Decentralized Hungarian (DH) algorithm, akin to the Consensus-Based Auction Algorithm (CBAA), was replaced by the Hungarian algorithm [8]. The advantages of Particle Swarm Optimization (PSO) algorithm lie in its simplicity, minimal parameter tuning, and rapid convergence. Geng et al. introduced an enhanced PSO algorithm for uncertain rescue task allocation with time constraints, with a particular focus on robot rescue task assignment [9].

This study draws inspiration from two seminal references [6] and [8], aiming to incorporate their ideas of distribution and decentralization to address practical issues: in underwater target search scenarios involving collaborative efforts of multiple Autonomous Underwater Vehicles (AUVs), effective task allocation is required. Throughout this process, maintaining optimal allocation and minimizing time consumption are crucial factors. Reference [6] employs a continuous running genetic algorithm that facilitates the exchange of complete solutions (task sequence sets) among agents. This real-time adaptability allows for ongoing refinement of task allocation during execution, potentially resulting in global optimization for the team. In contrast, [8] originates from the centralized nature of the original Hungarian algorithm, initially employing a cost matrix encompassing all data for initial allocation. Subsequently, an iterative cost minimization process generates the optimal solution. Nevertheless, centralized task allocation methods really have limitations in terms of robustness and scalability. Therefore, the development of decentralized task allocation algorithms based on the Hungarian method becomes pivotal. This study initiates by introducing the two aforementioned methods individually, followed by a discussion of their respective characteristics. Finally, the study concludes by presenting a data analysis of the results achieved through these methods. Through this investigation, the paper contributes to advancing the comprehension of multi-agent systems and offers insights into the efficiency of decentralized task allocation approaches.

2. Methods

2.1 Decentralized Genetic Algorithm

The optimization approach proposed by R. Patel et al. in their work [6] primarily focuses on decentralization and parallelization. In their study, the researchers introduce a parallelized strategy employing a genetic algorithm (GA) to address the task assignment challenge within a multi-agent system. Their method involves agents concurrently executing tasks in real-time while refining the allocation process. Every agent retains and enhances its individual population of solutions, signifying sets of task sequences for upcoming assignments. A key aspect of their approach is the coordination and sharing of solutions among agents. Periodically, agents exchange their best solutions and integrate received solutions into their own populations. This collaborative exchange of high-quality solutions enables agents to reach consensus and take advantage of the inherent parallelism in genetic algorithms. Furthermore, by allocating a distinct thread for the GA, agents can persistently search for enhanced solutions while carrying out tasks. The decentralized genetic algorithm proposed in this research provides a dynamic and adaptive solution for the task allocation problem in a multi-agent environment.

2.1.1 Process of Genetic Algorithm

The genetic algorithm (GA) is inspired by the evolutionary process in nature and uses simulated chromosome crossover, mutation, and selection to solve optimization problems. In this study, the chromosome encoding and decoding method is adopted from the reference [10]. The overall flow of the genetic algorithm is as follows:

Firstly, I initialize the initial population of chromosomes. This corresponds to the section where the initial population of routes and breakpoints is generated. Secondly, I calculate the fitness of individuals in the initial population, this aligns with the code snippet where each individual's total distance is calculated and stored in the total_dist array. Thirdly, I select individuals from the current population for chromosome crossover and mutation to generate offspring population. The code achieves this by performing genetic operations like flipping, swapping, sliding, etc., to generate new routes and breakpoints. Subsequently, I evaluate the fitness of individuals within the offspring population. Following the generation of the new population via genetic operations, I compute and store the fitness of individuals in the offspring population in the total_dist array. Then, I replace the existing population with the offspring population. I check for a predefined number of iterations or if the fitness of the population has converged. If these stopping conditions are fulfilled, the iteration is halted, and the outcome is returned; otherwise, I return to step 3. This part of the process involves updating the population with the newly generated individuals and checking the termination criteria such as the number of iterations or convergence of the fitness. The loop in the code iterates through multiple generations until the termination conditions are met.

2.1.2 Parallel Distributed Genetic Algorithm

Each agent operates with two threads: the first thread manages the execution of a Genetic Algorithm (GA) and the second thread orchestrates the execution of the task sequence. This is illustrated in the mathematical model [6], where $\boldsymbol{x_i} = (x_{i1}, \dots, x_{in(i)})$ is a task sequence, $d(x_{ij}, x_{ik})$

is the length between tasks x_{ij} and x_{ik} ,$q_1, \dots, q_m$ are the initial locations, v is the speed of the agents, $c_d(\boldsymbol{x_i}) = d(q_i, x_{i1}) + \sum_{j=2}^{n(i)} d\big(x_{i,j-1}, x_{ij}\big)$ represents the distance of agent i to complete the task sequence.

Here I introduce two cost function, which are

$$c(\boldsymbol{x}) = \sum_{i=1}^{m} c_d(\boldsymbol{x_i}) \tag{1}$$

$$c(\boldsymbol{x}) = \frac{\max\{c_d(\boldsymbol{x_1}), \dots, c_d(\boldsymbol{x_i})\}}{v}. \tag{2}$$

The structure is depicted in Figure 1 [6]. The initial thread employs an adapted version of a known GA [11], designed for solving the mTSP (multiple Traveling Salesmen Problem), to tackle the task allocation issue.

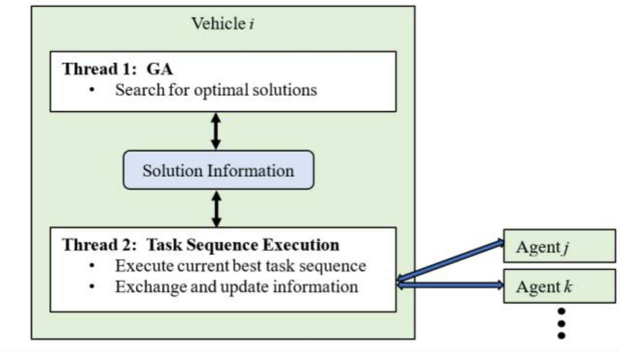
Vehicle i
Thread 1: GA
• Search for optimal solutions
Solution Information
Thread 2: Task Sequence Execution
• Execute current best task sequence
• Exchange and update information
Agent j
Agent k

Figure 1. Basic structure [6]

The pseudocode for the Task Sequence Execution thread is provided in Algorithm 1 [6].

Algorithm 1: Pseudocode for task sequence execution run in a separate thread on agent *i*.

```
1.  function TASK_SEQUENCE_EXECUTION()
2.  while mission not complete:
3.      if num_agents > num_locations:
4.          current_solution ← solution from nearest neighbors
5.      else:
6.          current_solution ← current best solution from GA
7.      end if
8.      task_sequence ← current_solution{$i$}
9.      execute(task_sequence)
10.     send information to all other agents
11.     receive information from whoever responds
12.     store received solutions for incorporation into GA
13. end while
14. end function
```

Algorithm 1 starts from a function defined as *" EXECUTION_TASK_SEQUENCE"*. The main loop of the function runs as long as tasks are not completed. Within this loop, the following steps are continuously executed. Firstly, it checks if the count of agents involved in the tasks is greater than the count of locations to be covered. This condition *(number_of_agents > number_of_locations)* implies that there are more agents involved in the tasks than the number of tasks to be completed. This might suggest a scenario where tasks need to be allocated among agents. If this condition is met, the current solution of agent *i* is assigned using a solution generated by a nearest-neighbor method. This approach aims to efficiently allocate tasks based on the proximity between agents and task locations. Secondly, if the condition in step 1 is not met *(number_of_agents <= number_of_locations)*, the current best solution of agent *i* is assigned, which was previously generated by a genetic algorithm (GA) used for optimizing the task sequence. Now, the algorithm holds the current solution of agent *i*. The sequence of tasks to be executed is determined based on the assigned current solution from the previous steps. Then, agents start executing the assigned task sequence.

After executing the task sequence, agents send messages to all other agents. This may include the status of tasks executed or any relevant updates. Next, agents wait to receive responses from other agents regarding the messages they sent. Upon receiving responses, agents process the received information. This information may include solutions or updates related to task sequences executed by other agents. The received solutions are stored, possibly for integration into the aforementioned genetic algorithm. These received solutions can contribute to improving the optimization of task sequences across all agents. After that, the loop continues executing the above steps as long as tasks are not completed. The *"EXECUTION_TASK_SEQUENCE "* function ends when tasks are completed.

This pseudocode outlines the algorithm for agents to execute task sequences within a larger task. The algorithm adjusts its approach based on the count of agents and tasks, employing either a nearest-neighbor strategy or a genetic algorithm for task allocation. Agents execute their task sequences, communicate with other agents, and incorporate received solutions into their optimization process, ensuring efficient task execution and collaboration among multiple agents.

2.2 Hungarian Algorithm and DHBA Algorithm

When the Hungarian algorithm is applied in an industrial environment, it demonstrates a rapid ability to solve task assignment problems, exhibiting excellent efficiency. The Hungarian algorithm employs specialized procedures to solve matching problems or more general allocation planning problems. It produces the minimum matching based on the provided cost matrix.

2.2.1 Hungarian Algorithm

This algorithm employs a cost matrix for allocation, where each element, c_{ij}, symbolizes the cost (or inverse of the reward) linked with assigning task j to agent i. To achieve an optimal assignment, Algorithm 2 [8] defines matrix operations through the subsequent steps.

Algorithm 2 Hungarian Algorithm

```
1.  Form a $n$ by $n$ cost matrix
2.  Step 1: Subtract the smallest entry in each row from all the entries of its row
3.  Step 2: Subtract the smallest entry in each column from all the entries of its column
4.  Step 3: Draw lines through appropriate rows and columns so that all the zero entries of the cost matrix are
    covered and the minimum number of such lines is used
5.  Step 4:
6.  procedure Test for optimality:
7.      if the minimum number of covering lines is $n$:
8.          An optimal assignment of zeros is possible and the assignment is finished
9.      end if
10.     if the minimum number of covering lines is less than $n$:
11.         An optimal assignment is not yet possible. In that case, proceed to Step 5
12.     end if
13. end procedure
14. Step 5: Determine the smallest entry not covered by any line. Subtract this entry from each uncovered row,
    and then add it to each covered column. Return to Step 3
```

To begin, I construct an n x n matrix to signify task-to-agent assignment costs. Each cell (i, j) in the matrix denotes the cost of assigning task i to agent j. I initiate row reduction by finding the minimum value within each row and subtracting it from all entries in that row. This step normalizes the minimum value in each row to zero. Then, I proceed to column reduction. By detecting the smallest value in each column and deducting it from all entries within that column, I normalize the minimum value in each column to zero. After this, the process of line covering is initiated. This involves strategically placing lines across rows and columns to encompass all zero entries in the matrix. The objective is to cover the maximum number of zeros with the fewest lines, thereby identifying potential assignments for an optimal solution. For assessing optimality, I examine whether the minimum lines required for coverage match n (the number of tasks/agents). If they do, an optimal assignment is achieved, and the algorithm terminates. If the minimum lines used are fewer than n, I move on to Step 5. In the final step, involving matrix updating and repetition, I identify the matrix's smallest entry that isn't enclosed by any line. This entry's value is subtracted from every uncovered row and added to each covered column. Subsequently, I loop back to Step 3, repetitively executing the procedure of line coverage and assessing optimality.

In the mentioned paper, the auction algorithm and the Hungarian algorithm's performance is conducted. The analysis focuses on how the average time required for the execution of each group of drones increases with the growing number of agents. Table 1 in [8] clearly indicates an exponential growth trend in the computation time needed for the auction algorithm to accomplish the allocation. The Hungarian algorithm demonstrates a clear advantage in terms of scalability.

Table 1. Mean time of Auction and Hungarian Algorithm. [8]

Groups		**Average time (sec)**	
		A	**H**
#1	2	0.33	0.34
#2	4	0.34	0.34
#3	8	0.36	0.37
#4	12	0.47	0.38
#5	16	0.85	0.38
#6	24	2.57	0.41
#7	32	8.13	0.42
#8	50	71.01	0.43

2.2.2 Decentralized Hungarian-Based Algorithm (DHBA)

This paper introduces a Decentralized Hungarian-Based Algorithm (DHBA) for tackling the task allocation problem in scenarios where network connectivity is limited. In this context, each Autonomous Underwater Vehicle (AUV) only interacts with a subset of agents, forming local networks. Consequently, agents possess partial information and lack global awareness.

The algorithm's primary goal is to attain an optimal task allocation that minimizes overall costs. To achieve this, ensuring network connectivity is crucial, indicated by the positive value of the second smallest eigenvalue (λ_2) of the network's Laplacian matrix. This safeguards the convergence of each agent's cost matrix to a stable state during the algorithm's final phase. A loss of network connectivity ($\lambda_2 = 0$) leads to deviation from the optimal solution.

The core idea of the DHBA is as follows: Firstly, each agent possesses a cost matrix with information about tasks, such as distances to targets. Secondly, at each step, agents communicate directly with their neighbors and exchange updated data. Each agent merges its cost matrix data with its neighbors' data to obtain more accurate task allocation information. Subsequently, through the iterative process, each agent gradually obtains the same cost matrix, representing the optimal task allocation solution. Next, the algorithm optimizes task allocation iteratively by exchanging information within the local network, gradually converging to the optimal solution.

Algorithm 3 DHBA Algorithm

```
1.  procedure Initialization:
2.      For AUV $i$, form a cost matrix $C^i$, in which each element $C^i_{rj}$ represents the distance between AUV $r$ to target $j$, such that:
3.      if $r = i$:
4.          $C^i_{rj} = \sqrt{(X_u(r) - X_t(j))^2 + (Y_u(r) - Y_t(j))^2}$
5.      else:
6.          $C^i_{rj} = \infty$
7.      end if
8.  end procedure
9.  procedure Task allocation
10.     Phase 1: Apply Hungarian Algorithm (Algorithm 3) and get assignment for AUV $i$
11.     Phase 2: Update $C^i$
12.     procedure For AUV $i$:
13.         Connect with neighbor (AUV) $k$ and receive $C^k$
14.         Update $C^i$ such that:
15.         if $C^k_{ij} \neq \infty$:
16.             $C^i_{ij} = C^k_{ij}$
17.         end if
18.     end procedure
19.     Return to Phase 1
20. end procedure
```

The pseudocode for DHBA is provided in Algorithm 3. Initially, every unmanned aerial vehicle (AUV) detects its own position $(X_u(i), Y_u(i))$, along with the positions of neighboring AUVs $(X_u(k),\ Y_u(k))$, as well as the locations of all targets $(X_t(j),\ Y_t(j)$. Subsequently, AUV i constructs the neighbor matrix 'G' based on the collected data. Additionally, AUV i generates an individualized cost matrix $\boldsymbol{C}^i$. The entries within this matrix are predominantly set to infinity, except for the i^{th} row. This row encompasses the distances between AUV i and other neighboring AUVs within its local network. The DHBA (Decentralized Hungarian-Based Algorithm) algorithm operates through a cyclic process, alternating between two primary phases. In the initial phase, each AUV employs the centralized Hungarian algorithm outlined in Algorithm 2. This phase's purpose is to guarantee that every AUV is assigned a target for the entire mission duration, thereby achieving an optimal solution for task allocation. Following this, the algorithm advances to the second phase. Here, AUV i establishes connections with its neighboring AUVs, facilitating the exchange of cost matrices. Consequently, AUV i updates its own cost matrix in accordance with the received data. During this communication stage, AUV i compares its cost matrix with those of its neighbors to gain supplementary information, which aids in refining task allocation optimization. Progressing through Phase 2, the aim is for AUV i's cost matrix to synchronize with the cost matrices of all other neighboring AUVs. As each AUV successfully assimilates global network information into its cost matrix, the task allocation converges to a stable state. This convergence signifies the completion of the DHBA algorithm's execution.

3. Results

3.1 Section of Decentralized GA

3.1.1 Testing Procedure for decentralized GA

In study [6], the researchers conducted a comparative analysis between the decentralized Genetic Algorithm (GA) and alternative decentralized task allocation methods. This assessment involved a range of problem instances with varying sizes. To ensure the results' consistency despite fluctuations in simulation conditions and the stochastic nature of the genetic algorithm (GA) approach, each case was subjected to 20 distinct trials. Our implementations were carried out using the Python programming language, while the simulations were conducted through ROS. These discrete processes, effectively mirroring the functionality of a decentralized system. For effective inter-agent communication, each agent featured a communication interface scripted in C++, enabling seamless message dissemination among all system agents. The agents' velocity was set at a uniform 5 meters per second, and the invocation frequency was maintained at 0.01 seconds.

In reference to [6], a series of tests were carried out across five distinct instances. In these instances, the points were chosen at random in an area spanning 100 meters by 100 meters. The instances had different sizes $(n \times m)$ of $10 \times 5, 20 \times 4, 30 \times 3, 35 \times 5$, and 40×6. Instances of sizes 30×3 and 35×5 are illustrated in Figure 2 [6] below.

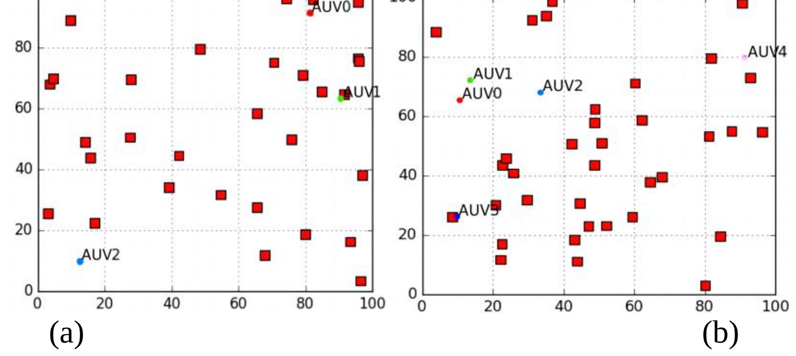

Figure 2. (a) 30×3 and (b) 35×5. The entities denoted as "AUV#" are represented by circular shapes, while the points of interest are visually depicted as red squares. The axes on the graph provide a metric for position, measured in meters. [6]

Multiple experiments were also conducted in the paper, comparing various decentralized task allocation methods as benchmarks. These methods included Greedy Nearest Neighbor (NN), Decentralized Hungarian Algorithm (DH) [12], Consensus-Based Auction Algorithm (CBAA) [13], Asynchronous Consensus-Based Bidding Algorithm (ACBBA) [14], Performance Impact Algorithm (PIA) [15], and Hybrid Information and Plan Consensus (HIPC) [16].

In these methods, a common underlying structure was employed. Figure 3 [6] below illustrates this structure, with only one task sequencing execution thread running. The execution thread perpetually operates in a continuous manner, alternating between two primary phases.

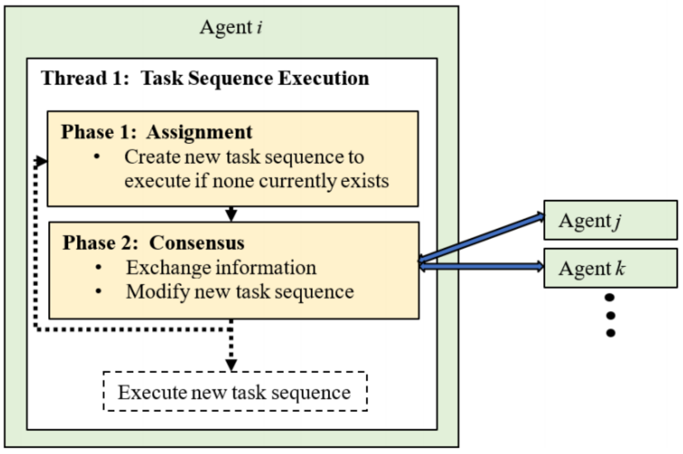
Agent *i*
Thread 1: Task Sequence Execution
Phase 1: Assignment
• Create new task sequence to execute if none currently exists
Phase 2: Consensus
• Exchange information
• Modify new task sequence
Execute new task sequence
Agent *j*
Agent *k*

Figure 3. Basic structure. [6]

3.1.2 Results Analysis for decentralized GA

The Paper simultaneously considered performance metrics of total time and total distance. A comprehensive evaluation of all methods and variables was undertaken to analyze their individual performance across different metrics. The objective was to determine which variables exhibited the most favorable outcomes for each metric.

Table 2 [6] below (Time) depicts the average time taken to access all the designated points of interest for each instance when employing each method.

Table 2. The average time (measured in sec) required to cover all designated points was calculated more than 20 separate trials on every approach and instance. The bold represent the most optimal averages achieved for each instance. [6]

	Instance				
Approach	10 * 5	20 * 4	30 * 3	35 * 5	40 * 6
NN	19.6	45.5	41.6	33.4	26.2
CBAA-ms	19.6	45.5	41.6	33.4	26.2
CBAA-mm	14.7	28.8	41.4	27.6	26.0
DH-ms	**11.6**	29.6	42.1	25.8	23.5
DH-mm	**11.6**	29.6	42.7	25.9	23.3
ACBBA-ms	15.1	24.1	34.4	27.6	26.8
ACBBA-mm	14.6	25.5	34.4	24.6	25.8
PIA	19.2	31.3	45.1	34.6	31.8
HIPC	11.8	27.9	34.4	24.0	24.7
GA-ms	12.0	26.0	33.9	**21.8**	23.6
GA-mm	12.5	23.3	33.8	24.2	23.3
GA-multi	14.1	**22.5**	**33.3**	22.3	**22.5**

In the case of 10 * 5 instance, both the DH-ms and DH-mm methods exhibit the best average time of 11.6 seconds, showcasing their outstanding performance for this smaller-scale problem. Also, the GA-ms method performs well with an average time of 12.0 seconds. Shifting to 20 x 4 scenario, the GA-multi method stands out with the best average time at 22.5 seconds, demonstrating impressive results. Notably, the GA-mm method also performs admirably with an average time of 23.3 seconds. When it comes to 30 * 3 instance, the GA-mm method takes the lead with the best average time of

33.3 seconds, outperforming alternative methods. Both the HIPC and ACBBA methods also hold up well, each achieving an average time of 34.4 seconds. In the context of 35 * 5 instance, it's the GA-ms and GA-multi variants that shine, recording the best average times of 21.8 and 22.3 seconds, respectively, and significantly outpacing other approaches. Finally, for 40 * 6 instance, the GA-multi method once again demonstrates superiority with the best average time of 22.5 seconds, highlighting its remarkable efficiency.

In summary, in most instances, variants of the decentralized GA method exhibit impressive performance, particularly the GA-multi method, which excels on medium and large-scale instances. The DH methods perform well on small-scale problems but show relatively weaker performance as the problem scale increases. The CBAA and PIA methods have relatively poorer performance, and their suitability may depend on specific scenarios.

Table 3. The average total distance covered by all agents. The bold values highlight the most favorable averages achieved for each specific instance. [6]

	Instance				
Approach	10 * 5	20 * 4	30 * 3	35 * 5	40 * 6
NN	486.3	903.8	603.3	826.8	773.2
CBAA-ms	301.2	517.6	606.7	689	719.4
CBAA-mm	301.5	519.8	602.2	625.2	709.2
DH-ms	256.8	517.8	615.1	584.2	649.6
DH-mm	256.2	516.6	626.7	585.1	641.1
ACBBA-ms	327.8	404.3	441.1	624.8	690.8
ACBBA-mm	334.5	428.8	**439.7**	520.8	683
PIA	209.8	482.3	508.1	**481.5**	**563.9**
HIPC	**198.1**	**401.6**	445.0	443.0	590
GA-ms	222.7	448.9	456.6	460.5	600
GA-mm	268.1	429.6	468.8	553.9	636.7
GA-multi	245.6	404.6	455.8	494.9	604

Table 3 [6] presents the average cumulative distances traveled by all agents for each method across the five instances. Based on the data in the table (where bold numbers represent the shortest distances for each instance), unlike the case of minimizing total time, the decentralized Genetic Algorithm (GA) does not exhibit a significant advantage over other algorithms in terms of minimizing the total distance.

Firstly, GA does not consistently exhibit the best performance across all instances. Apart from specific cases where certain variants of GA might perform well, such as GA-multi showing comparable performance to the HIPC method in 20 x 4 instance, GA does not dominate in any instance. Except for the PIA and HIPC methods, the GA-ms method outperforms most others in 40 x 6 instance. Additionally, with increasing instance sizes, the decentralized GA variants consistently exhibit superior performance compared to other methods. This observation indicates that the method showcases commendable scalability.

Overall, while the decentralized Genetic Algorithm approach produces task assignments that excel over other methods in terms of the minimum time objective, the allocations it finds are not superior in terms of the minimum total objective. This might be attributed to the fact that other methods might find optimizing the minimum time objective more challenging, as it requires knowledge of task sequences of other agents, which isn't a barrier for the decentralized Genetic Algorithm method due to each agent considering a complete solution. On the other hand, other methods might be more adept at optimizing the minimum total objective, especially in cases where the instance size isn't large. Therefore, for smaller instances, the decentralized Genetic Algorithm method fails to find better solutions. Furthermore, the condition mandating that every agent must be allocated at least one task could potentially restrict the Genetic Algorithm's ability to discover solutions with more optimal overall travel distances. This is because, in specific scenarios, employing fewer agents might lead to a reduction in the collective travel distance.

3.2 Section of DHBA

3.2.1 Test Procedure for DHBA

The study involved a comparative analysis between the DHBA (Decentralized Hungarian-Based Algorithm) proposed in the paper and the existing CBAA (Consensus-Based Auction Algorithm). Experiments were conducted using a cluster of Autonomous Underwater Vehicles (AUVs) assigned to monitor multiple targets. The positions of both AUVs and targets were chosen randomly within a designated application space measuring $m \times n$ units. The communication range was established at $R = m/2$. The AUVs were categorized into six groups, as detailed in Table 4 [8] provided below. Throughout these runs, critical parameters such as λ_2 (a specific variable), the final assignment outcomes, and iterations required to reach convergence were meticulously documented.

Table 4. Simulation setup. [8]

Group	Agents Count	Area Size (units)
1	5	5
2	10	10
3	20	20
4	30	30
5	40	40
6	50	50

3.2.2 Results Analysis for DHBA

During each simulation, λ_2 values, final assignments, and convergence steps for both algorithms were recorded. These results are visualized in Figure 4 [8].

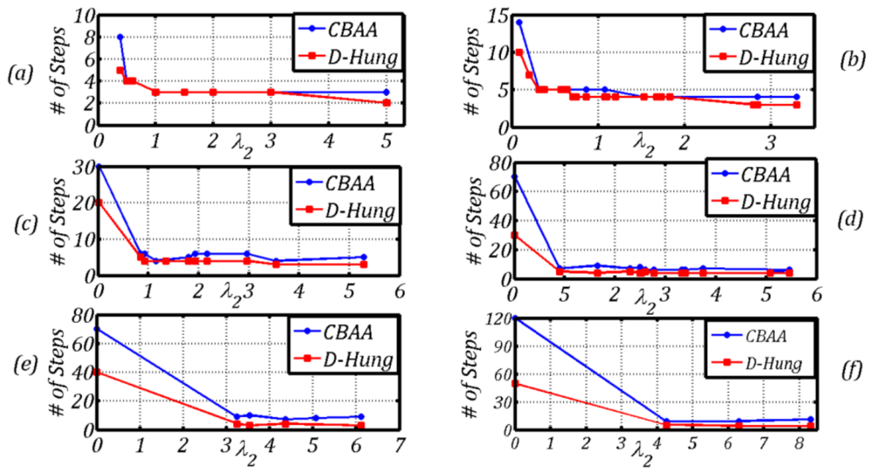

Figure 4. The required iteration count to achieve convergence towards the ultimate assignment, depicted as a function of λ2, varies across distinct groups: (a) 5 agents, (b) 10 ~, (c) 20 ~, (d) 30 ~, (e) 40 ~ and (f) 50 ~. [8]

Based on the presented figures, it is evident that when λ_2 is larger, the convergence speed of both algorithms will be faster. When λ_2 approaches 0, indicating a fragile network connection, there is a higher likelihood of conflicts, and CBAA exhibits a noticeably slower convergence speed compared to DHBA. Consequently, both algorithms necessitate additional iterations to attain the final assignment outcomes. When the number of agents is set to 5, CBAA and DHBA demonstrate relatively similar convergence speeds. However, as the number of agents increases, the difference between the two becomes more pronounced.

Across all five scenarios, DHBA consistently demonstrates superior performance compared to CBAA. This can be attributed to DHBA's dependence solely on the connectivity of the AUV system. In scenarios with loosely connected networks where the value of λ_2 is significantly smaller than 1, the steps necessary to achieve the final assignment are equivalent to the total number of AUVs within the network. Conversely, due to the requirement for more steps to resolve conflicts, CBAA mandates a higher number of iterations for convergence. Moreover, CBAA's reliance on auction algorithms generally leads to increased analysis time in comparison to the Hungarian algorithm employed within DHBA.

When contrasted with CBAA, the efficacy of the proposed DHBA in yielding optimal assignments becomes evident. The quantified cost difference, denoted as Δcost, is outlined as the divergence between the global costs produced by CBAA (CCBAA) and DHBA (CDHBA) — in essence, Δcost = CCBAA - CDHBA. This relationship is visually depicted in the subsequent Figure 5 [8].

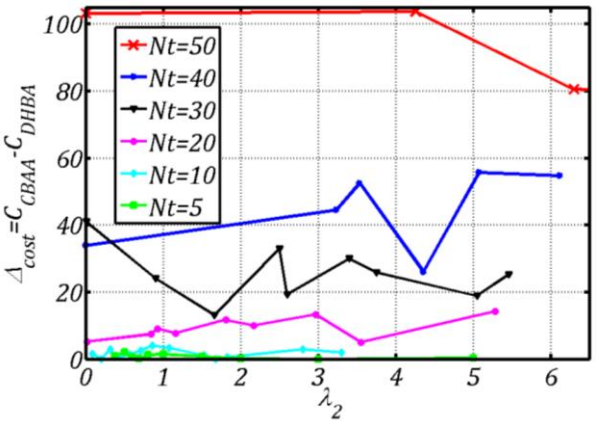

Figure 5. The cost discrepancy, represented as Δcost, between CBAA and DHBA across various agent groups in relation to distinct λ_2 values. [8]

The depicted figure vividly displays the positive nature of cost differences across all instances, and these differences amplify as λ_2 increases. Notably, the fluctuations in cost disparities across diverse instances remain relatively subdued with the upsurge in λ_2 values. This consistency underscores DHBA's persistent superiority over CBAA, preserving a well-balanced advantage.

These outcomes affirm that the proposed DHBA consistently outperforms CBAA. This is attributed to the inherent characteristics of the original Hungarian algorithm, allowing DHBA to consistently furnish optimal solutions for linking multi-agent networks. CBAA's reliance on an auction mechanism introduces deviations from optimality, particularly as the agent count and conflicts escalate. Conversely, the DHBA, leveraging a centralized Hungarian algorithm, ensures absolute optimality in its results.

4. Conclusion

This paper leverages two innovative papers to conduct an in-depth analysis of the multi-AUV task allocation optimization problem and achieves successful resolutions. It demonstrates the viability of the decentralized algorithmic approach in this context. The first method employs a parallelized genetic algorithm. The results indicate that the decentralized genetic algorithm rapidly converges to high-quality solutions on the considered instances, outperforming existing methods for task completion time on larger instances. The second method develops a decentralized task allocation algorithm based on the Hungarian algorithm. It concludes that the algorithm consistently generates optimal solutions as long as network connectivity is maintained. Moreover, it outperforms CBAA in various aspects, including performance, analysis time, convergence speed, allocation optimality, and computational demands. To summarize, both approaches showcase distinct advantages, notably excelling in performance compared to conventional algorithms. Both algorithms robustly address the task allocation challenge posed in this underwater search scenario.

References

[1] J. Chen and P. Dames, "Distributed and collision-free coverage control of a team of mobile sensors using the convex uncertain voronoi diagram." *2020 American Control Conference (ACC)* (IEEE 2020).

[2] J. Chen and P. Dames, "Distributed multi-target tracking for heterogeneous mobile sensing

networks with limited field of views." *2021 IEEE international conference on robotics and automation* (IEEE 2021).

[3] G. P. Das, T. M. McGinnity, S. A. Coleman, and L. Behera, "A distributed task allocation algorithm for a multi-robot system in healthcare facilities," in *Journal of Intelligent & Robotic Systems 80* (2015), pp. 33–58.

[4] A. Whitbrook, Q. Meng, and P. W. Chung, "A novel distributed scheduling algorithm for time-critical multi-agent systems," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems* (IEEE, 2015), pp. 6451–6458.

[5] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents", *IEEE transactions on systems, man, and cybernetics, part b (cybernetics)* (1996), Vol. 26, No. 1, pp. 29–41.

[6] R. Patel, E. Rudnick-Cohen, S. Azarm, M. Otte, H. Xu, and J. W. Herrmann, "Decentralized task allocation in multi-agent systems using a decentralized genetic algorithm," in *International Conference on Robotics and Automation* (IEEE, 2020), pp. 3770–3776.

[7] H. Choi, Y. Kim, and H. Kim, "Genetic algorithm based decentralized task assignment for multiple unmanned aerial vehicles in dynamic environments," *International Journal Aeronautical and Space Sciences* (2011), Vol. 12, No. 2, pp. 163–174.

[8] S. Ismail and L. Sun, "Decentralized Hungarian-based approach for fast and scalable task allocation," in *International Conference on Unmanned Aircraft Systems* (IEEE, 2017), pp. 23–28.

[9] N. Geng, Z. Chen, Q. A. Nguyen, and D. Gong, "Particle swarm optimization algorithm for the optimization of rescue task allocation with uncertain time constraints," *Complex & Intelligent Systems 7* (2021), pp. 873–890.

[10] J. Kirk, "Fixed start/end point multiple traveling salesmen problem-genetic algorithm," *MATLAB® Central File Exchange* (2014).

[11] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics* (1955), Vol. 52, No. 1, pp. 7–21.

[12] M. Nanjanath and M. Gini, "Repeated auctions for robust task execution by a robot team," *Robotics and Autonomous Systems* (2010), Vol. 58, No. 7, pp. 900–909.

[13] L. Brunet, H. L. Choi, and J. How, "Consensus-based auction approaches for decentralized task assignment," in *AIAA guidance, navigation and control conference and exhibit* (2008), p. 6839.

[14] L. Johnson, S. Ponda, H. L. Choi, and J. How, "Asynchronous decentralized task allocation for dynamic environments," in *Infotech@ Aerospace 2011*(2011), p. 1441.

[15] W. Zhao, Q. Meng, and P.W. Chung, "A heuristic distributed task allocation method for multivehicle multitask problems and its application to search and rescue scenario," *IEEE Transactions on Cybernetics* (2016), Vol. 46, No. 4, pp. 902–915.

[16] L. B. Johnson, H. L. Choi, and J. P. How, "Hybrid information and plan consensus in distributed task allocation," in *AIAA Guidance, Navigation, and Control Conference* (2013), p. 4888.