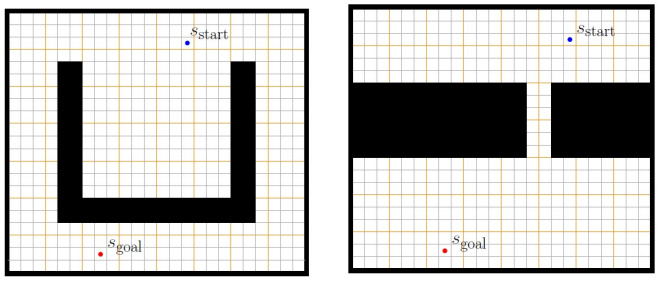
Efficient Path Planning of Unmanned Surface Vehicle (USV) Through Inland Water Bodies in Sunderban Delta

Alvin P Cheriyan[1]
Robert Bosch Center for Cyber-Physical Systems
Indian Institue of Science
Bangalore, India
alvinc@iisc.ac.in

***Abstract*— Unmanned Surface Vehicles (USVs) are remotely controlled by operators at sea or on land or programmed to complete predetermined missions with minimal human intervention. In order to save resources and complete the mission in the shortest time possible, it is essential to find an optimal path that avoids obstacles with a reasonable safety margin as fast as possible. In areas like the Sunderbans Delta, where the width of the water channel can change from small to large very quickly using, multiple resolutions and multiple heuristics can be used to obtain a quick solution. The approach proposed in this paper generates quick solutions whose optimality increase with time while leveraging multiple resolutions and multiple heuristics to escape local minima and reach the goal with the least number of node expansions. In pursuit of this goal, this paper proposes the use of Anytime Multi-Resolution Multi-Heuristic A* (AMRA*) to generate an optimal path quickly by sharing information across multiple resolutions and multiple heuristics while producing anytime solutions.**

I. Introduction

The majority of the earth is covered by Oceans, yet its importance is not matched by our knowledge. A lot of information can be gleaned from the oceans. Even forecasting the weather also requires information from ocean observations. It provides early warning of dangers like tsunamis, storm surges, and extremely large waves, which helps save lives and keeps marine operators productive. Currently, oceanographic monitoring is done by sending out ships with humans onboard to collect the data. Since the weather on the ocean is unpredictable, there could be sudden storms that could put the lives of the humans onboard in danger. Thus, Unmanned Surface Vehicles (USVs) are used to perform the monitoring. These robots can autonomously monitor the ocean and relay the data for analysis. For such autonomous USVs, to complete a mission given to them, path planning is extremely essential. If it can plan paths from starting to the destination with the least distance, then a lot of resources can be saved. The ocean is mostly empty and free, so path planning can be done at a very coarse (low) resolution. But, around islands and delta regions, it becomes difficult to plan paths because the width of the water channels can change drastically from very small to very large and vice versa. In such situations, a fine (high) resolution must be used to plan paths. If the finest (highest) resolution is used for the full path planning, then the time and computational complexity are very high. Thus a better solution is needed that can switch between resolutions and plan paths quickly.



(a) Cul de sac (b) Narrow Passage

Fig. 1: [3] It is very computationally expensive for the High (grey) resolution search to escape the local minimum in Fig.1a and the Low (orange) resolution search does not give a solution in Fig. 1b

In this work, the proposed approach to path planning for the USV using AMRA*[1], incorporates multiple resolutions and heuristics to ensure adaptability to varying conditions, with larger strides used in open areas and smaller ones in more restricted spaces. When faced with broad obstacles that could trap the search in local minima, the algorithm utilizes information sharing between alternate heuristics to escape these traps. To achieve quick solutions, anytime methods that relax the optimality bounds while improving the solution over time with more available resources are used.

II. Literature Survey

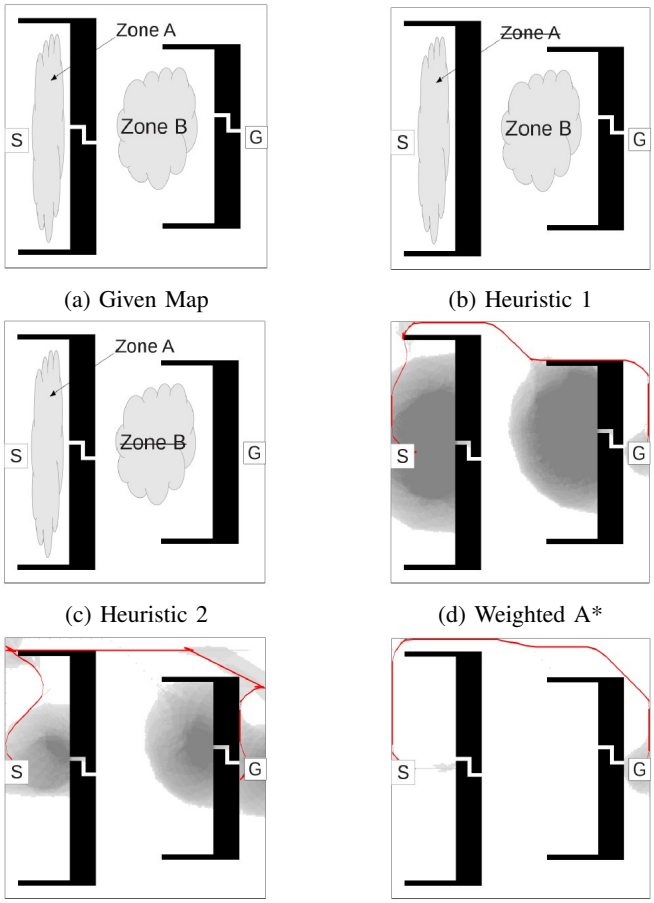
For path planning, heuristic-based search algorithms are used. Heuristic-based algorithms' effectiveness is closely related to the size of the state space, which is, in turn, determined by the spatial dimension, the degree of discretization used, and the size of the environment being explored. In order to avoid local minima, a larger number of expansions is necessary because higher resolutions cause the size of the state space to grow exponentially. Although coarser discretization of the search space is more successful in wide open spaces and when the heuristic estimate of the cost-to-goal has a weak correlation with the true cost-to-goal, which aids the search in avoiding local minima, finer resolutions are more successful in small spaces. Low-resolution search algorithms, however, might not be successful in locating a solution. To address this issue, it is best to use a combination of coarse and fine resolutions. Accordingly, multi-resolution

approaches that offer the advantages of multiple resolutions [3] by leveraging coarse resolutions for some parts of the search and fine resolutions for others are employed. A simple example of how the multiple resolutions are needed is shown in Fig 1.

The computational resources dedicated to exploring the search space are also influenced by the choice of heuristic function. Some heuristics may expand more states to navigate and escape local minima, while others may perform more efficiently in the same space. Multi-heuristic planners can achieve faster speeds than A* search algorithms by inflating (overestimating) heuristic values with an inflation factor ($w_1 > 1$) to impart the search with a depth-first flavor. However, when used in isolation, these additional heuristics provide little value as they cannot guarantee completeness (since they can be arbitrarily inadmissible) or fast planning times (due to the possibility of each heuristic having its own depression region). Here, multiple such heuristics are considered to explore different areas of the search space while using a consistent heuristic to ensure completeness. This approach may result in faster convergence if any of these heuristics (or their combination) can effectively guide the search around depression regions. This approach, known as Multi-Heuristic A* [2], enables information sharing between heuristics, thereby facilitating more effective guidance of the search process. This is shown in Fig 2.

The AMRA* algorithm employed to plan the path for USV, improves the quality of the solution found over time using the Anytime Repairing A* algorithm [4]. Providing sub-optimality bounds is useful as it helps assess the quality of the plan and decide whether to continue or preempt the search based on the current sub-optimality. A* search with inflated heuristics ($w_1 > 1$) is sub-optimal but fast while also providing a bound on the sub-optimality. An anytime algorithm can be constructed by running a succession of A* searches with decreasing inflation factors. ARA* also uses A* with inflated heuristics but reuses search efforts from previous executions to satisfy sub-optimality bounds and provides a bound on the time before it produces its first plan. Control over sub-optimality bounds can adjust the trade-off between computation and plan quality.

The results in the work show that AMRA* algorithm produces a quick solution based on the maximum permissible sub-optimality. With progressing time, solutions with improving optimality are generated. This approach is much better than A* at a single resolution in almost every metric. But AMRA* has an overhead in comparison to MRA* as the former involves sharing across multiple heuristics in each resolution, causing extra state expansions than those in MRA*. But when the above problem has been expanded to multiple dimensions in state space, AMRA* is the more suitable candidate, along with asymptotically optimal sampling-based methods.



(a) Given Map (b) Heuristic 1

(c) Heuristic 2 (d) Weighted A*

(e) MHA* without sharing (f) MHA* with sharing

Fig. 2: [2] The map in Fig. 2a has 2 depression zones. Heuristic 1 in Fig. 2b avoids Zone A, while Heuristic 2 in Fig. 2c avoids Zone B. Without sharing information between Heuristic 1 and 2, the search is trapped in local depressions as shown in Fig. 2d and Fig. 2e. If the information is shared between the two heuristics, then the search is not trapped in local depressions as shown in Fig. 2f

III. AMRA* ALGORITHM

A. Problem Formulation

Consider a motion planning problem of the USV with the state space, starting state and goal set represented as $(\mathcal{X}, x_s, G)$, where x_s is in $\mathcal{X}_{free}$, where $\mathcal{X}_{free} \subset \mathcal{X}$ denotes the obstacle-free space and $G \subset \mathcal{X}$ is a set of goal states in $\mathcal{X}_{free}$. A solution to the motion planning problem, if one exists, is a collision-free path through water from x_s to G. Let $g : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$ be a cost function to compute the cost of propelling the USV from one state to another. The cost of a potential solution path $\mu = \{x_1, \ldots, x_n\}$ is denoted by defining the cost function as $g(\mu) = \sum_{i=1}^{n-1} g(x_i, x_{i+1})$. AMRA* aims to solve this least-cost path planning problem and find the optimal path $\mu^* = \arg\min_\mu g(\mu)$.

The least-cost motion planning problem of the USV is solved with a heuristic search algorithm over a graph $G = (V, E)$. The vertex set $V \subset X$ contains states of the USV in a

2D Grid space constrained to $\mathcal{X}_{free}$. Edges $e = (x_i, x_j) \in E$ connect two vertices $x_i, x_j \in V$ if the robot can execute an action that takes it from x_i to x_j. Thus each edge $e \in E$ is also an action a in the robot action space A.

AMRA* is initialized with the start state x_s, goal set G, action spaces $\{A_0, ..., A_N\}$, and heuristics $\{h_0, ..., h_M\}$. The state space $\mathcal{X}$ is discretized into N levels. The anchor action space is the union of all action spaces (i.e., $A_0 = \bigcup_{r=1}^{N} A_r$), and the anchor search is run at the finest resolution ($V_0 \equiv V_1$). The anchor heuristic h_0 is consistent or admissible, while the other heuristics may be inadmissible. There is at least one heuristic per resolution, thus $M \geq N$. A heuristic function (h) is said to be consistent if for every pair of states x_i and x_j, and the cost $g(x_i, x_j)$ of the cheapest action from x_i to x_j, the estimated cost of reaching the goal from x_i is no greater than the estimated cost of reaching the goal from x_j plus the cost of reaching x_j from x_i via the cheapest action, which is, $h_{x_i} \leq h_{x_j} + g(x_i, x_j)$. This property ensures that the heuristic never overestimates the cost of reaching the goal. In essence, an admissible or consistent heuristic underestimates the distance to the goal.

Let the cost-to-come for a state be $g : V \rightarrow \mathbb{R}_{\geq 0}$. The predecessor of state x on the best-known path from x_s to x, denoted by $bp(x)$, is referred to as its back pointer. The function $Resolutions(x)$ returns the set of resolutions that the state x lies in, or equivalently, $r \in Resolutions(x) \iff x \in V_r$. Each resolution r is associated with a list of states expanded in that resolution, $CLOSED_r$. $Res(i)$ returns the resolution associated with the heuristic h_i. Each heuristic h_i is associated with a priority queue $OPEN_i$. $Succs(x, A_r)$ generates all valid successors of x at resolution r using the appropriate action space A_r. For $r = 0$, this generates all valid successors of x for all resolutions in $Resolutions(x)$.

B. Action Space

AMRA* algorithm generates the vertex set V as a union of different levels of discretization or resolutions r in $\mathcal{X}$, or, $V = \bigcup_r V_r$. Each vertex set V_r has a corresponding edge set E_r that makes up the edges $E = \bigcup_r E_r$ used by AMRA* to construct the graph G. Each edge set E_r is denoted with an action space A_r that is available to the USV. The core assumption in this paper and motion planning with multiple resolutions is that, for every resolution r being used, the USV has access to actions A_r that take it between two states $u, v \in V_r$. This formulation allows for a state $x \in \mathcal{X}$ to exist at multiple resolutions r, and thus in multiple vertex and edge sets V_r, E_r. An example of the action space to be used is shown in Fig 3.

C. Anytime A* [4] (parameterised by w_1)

Standard A* search is obtained if w_1, the heuristic inflation factor is set to 1. This guarantees the best possible outcome. An imperfect solution can be found if $w_1 > 1$, but the imperfection is constrained by a factor of w_1. In other words, the length of the discovered solution does not exceed w_1 times the length of the ideal solution. Anytime Repairing A* (ARA*) search starts with a large w_1 and reduces it before each execution until w_1 is equal to 1. As a result, each search results in a solution that is within w_1 factors of optimality. However, it would be computationally expensive to restart the A* search each time the value of w_1 is reduced. In order to reduce computation, ARA* searches repurpose previous search results. The weighted A* module, which calculates a path for a given w_1, is taken into account first by the algorithm. The weighted A* module called Multi_RH_A*() in Algorithm 2 is then invoked repeatedly with a series of decreasing w_1s. The effect of changing the value of w_1 is shown in Fig 4

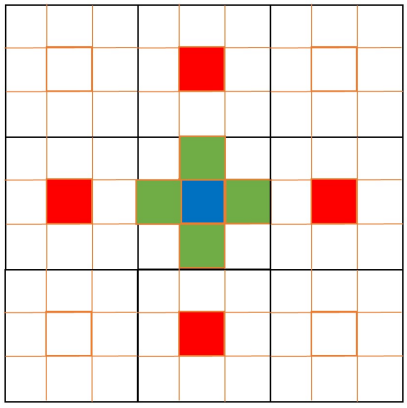


Fig. 3: Multi Resolution Action space for 4-connected grid navigation. The USV in the blue state can execute fine actions (1-step stride) to reach the green states and coarse actions (3-step stride) to reach the red states.

Local inconsistency is a new idea introduced by this algorithm. If a state's cost-to-come from the starting state decreases before the next time the state is expanded, that state is regarded as locally inconsistent. There is a local discrepancy between the state's cost-to-come values and those of its successors. When a state is expanded, the inconsistency is fixed by reassessing the cost-to-come of its successors. Due to a series of expansions, this leads to the inconsistency spreading to the child states. The child states eventually become independent of the parent state, their cost-to-come is not reduced, and they are not added to the $OPEN$ list. All locally inconsistent states are contained in the list $OPEN$. Every time a state's cost-to-come is reduced, it is added to the $OPEN$ state list, and every time a state is increased, it is removed from the list until its cost-to-come is reduced again. As a result, it is possible to think of the $OPEN$ list as a collection of states that must spread local inconsistency.

Each state can only be expanded once by an A* search using a consistent heuristic. Consistency may be broken, and A* search may expand states more than once if $w_1 > 1$ is set. The sub-optimality bound of w_1, however, still applies if each state is limited to being expanded only once. This restriction is implemented by checking any state whose cost-to-come is reduced and only inserting it into $OPEN$ if it hasn't already been expanded. In the CLOSED variable, a set of expanded states is kept. As a result of this restriction,

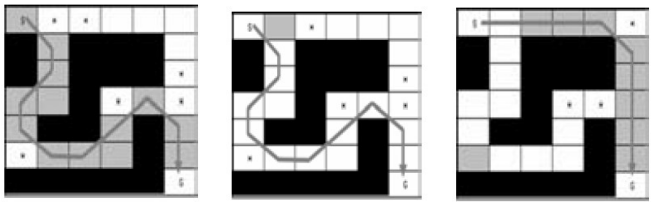
Algorithm 1 Weighted A* in Multi Heuristic and Resolution

```
procedure KEY(x, i)
    return g(x) + w_1 · h_i(x)
end procedure

procedure EXPAND(x, i)
    r ← Res(i)
    if i ≠ 0 then
        for all j > 0 do
            if j ≠ i∧ Res(j) = r then
                Remove x from OPEN_j
            end if
        end for
    end if
    for x' ∈ Succs(x, A_r) do
        if g(x') > g(x) + g(x, x') then
            g(x') ← g(x) + g(x, x'), bp(x') ← x
            if x' ∈ CLOSED_0 then
                Insert(x' ,INCONS)
            else
                Update(x_0 in OPEN_0 with KEY(x_0, 0))
                for all j ∈ {1, ..., M} do
                    l ← Res(j)
                    if l ∉ Resolutions(x') then
                        Continue
                    end if
                    if x' ∉ CLOSED_l then
                        if KEY(x', j) ≤ w_2 · KEY(x', 0) then
                            Update(x',OPEN_j, KEY(x', j))
                        end if
                    end if
                end for
            end if
        end if
    end for
end procedure

procedure MULTI_RH_A*
    while OPEN_i not empty ∀ i ∈ {0, ..., M} do
        i ← ChooseQueue                    ▷ Use Round-robin over inadmissible queues
        if OPEN_i.min() > w_2 · OPEN_0.min() then i ← 0
        end if
        x ← OPEN_i.pop()
        Expand(x, i)
        r ← Res(i)                         ▷ If i=0, r=0 which is the Union of all action spaces
        Insert(x, CLOSED_r)
        if x ∈ G then
            x_goal ← x
            return True
        end if
    end while
end procedure
```

each state can only be expanded once, but $OPEN$ might not contain every locally inconsistent state. The only locally inconsistent states that are present in $OPEN$ are those that have not yet been expanded. Nevertheless, it is crucial to



(a) $w_1 = 2.5$ (b) $w_1 = 1.5$ (c) $w_1 = 1.0$

Fig. 4: ARA* search [4] iterations with decreasing w_1. Locally inconsistent states at the end of each iteration are shown with an asterisk

.

record all locally inconsistent states because they will serve as the initial stages of inconsistency propagation in upcoming search iterations. All locally inconsistent states that are not in $OPEN$ are kept in the set $INCONS$. Prior to each new search iteration, inconsistency propagation can be started from the set of all locally inconsistent states represented by the union of $INCONS$ and $OPEN$.

Algorithm 2 Anytime Algorithm: Main loop

procedure MAIN($x_s, \mathcal{G}, \{\mathcal{A}_r\}_{r=0}^{N}, \{h_i\}_{i=0}^{M}, w_1^{init}, w_2^{init}$)
  $w_1 \leftarrow w_1^{init}, w_2 \leftarrow w_2^{init}$ ▷ Initialise Inflation factor
  $g(x_s) \leftarrow 0,\ bp(x_s) \leftarrow$ NULL
  Clear all lists $OPEN_i$, i $\in \{0, ..., M\}$ ▷ Lists based on heuristic
  Insert($x_s, INCONS$)
  while $\{w1 \geq 1 \wedge w2 \geq 1\ \}$ **do**
    for all $x \in INCONS$ **do**
      Update($x, OPEN_0$,KEY($x, 0$))
    end for
    Clear INCONS list
    for all $x \in OPEN_0$ **do**
      for all $j \in \{1, ..., M\}$ **do**
        if Res(j) $\in$ Resolutions(x) **then**
          Update($x, OPEN_j$, KEY(x, j))
        end if
      end for
    end for
    Clear all lists $CLOSED_r$, r $\in \{0, ..., N\}$
    if Multi_RH_A*() is TRUE **then**
      Print current solution using $bp(x_{goal})$ till x_s
    end if
    Decrease w_1, w_2
  end while
end procedure

D. Multi-Heuristic Search (parameterises heuristics by w_2)

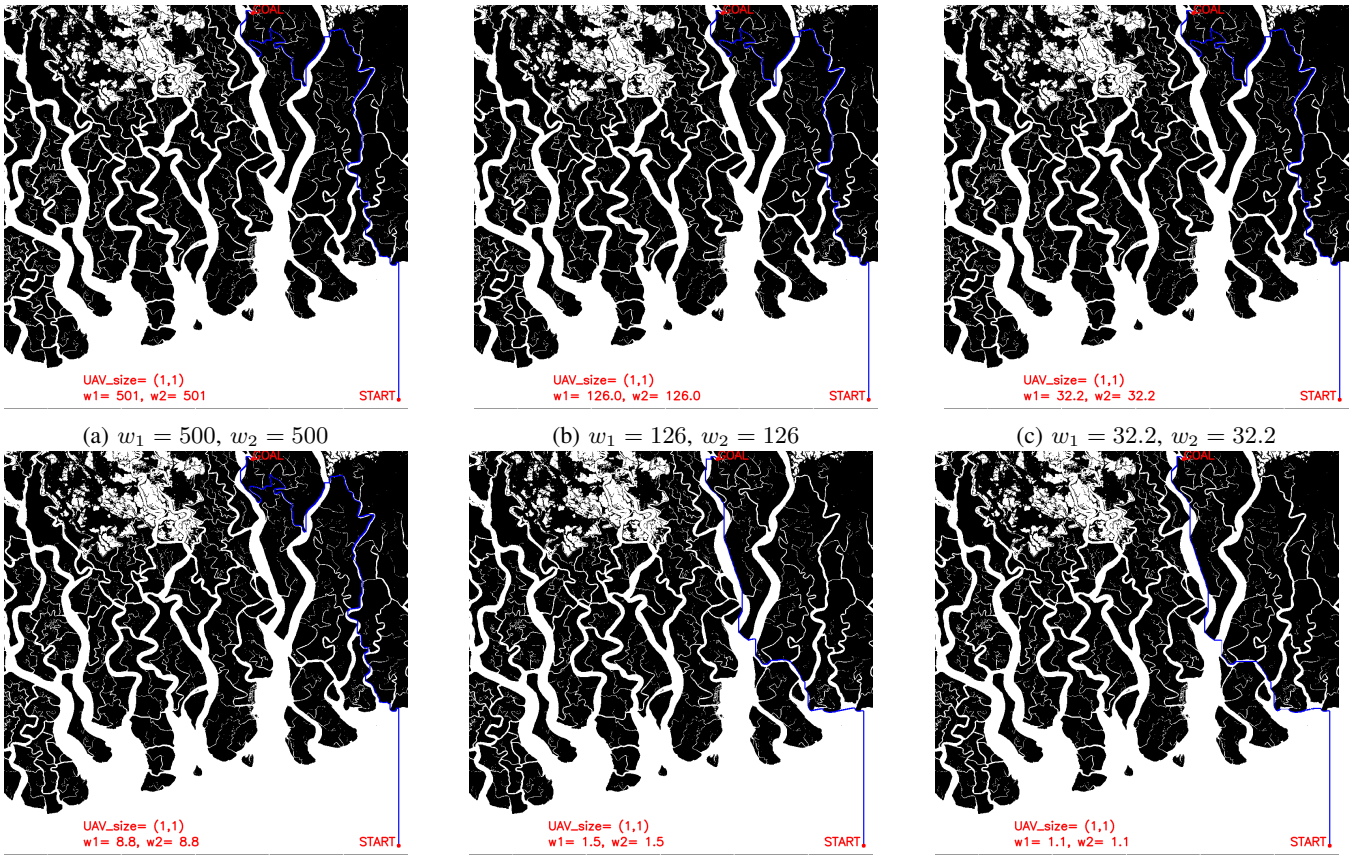
For a consistent heuristic, as previously discussed, the heuristic from a state x_s is never greater than the actual cost of getting to the goal from x_s. Inflating the heuristic frequently leads to significantly fewer state expansions and consequently faster searches (using $w_1 \times h(s)$ for $w_1 > 1$). A solution is no longer guaranteed to be the best one because inflating the heuristic may also violate the admissibility property [5]. The path to a particular state is shared by all searches in multi-heuristic search. As long as the state is present in a resolution using the heuristic, this means that if any of the searches find a better route to a state, the information is updated in all priority queues. Therefore, for each state, Multi Heuristic search uses a single cost-to-come value and back pointer. Additionally, path sharing enables Multi-Heuristic Search to expand each state no more than twice (1 inadmissible expansion + 1 anchor expansion). After initialization, it runs inadmissible searches in a round-robin manner as long as the check $OPEN_i$.min() $> w_2 \cdot OPEN_0$.min() $\forall i \in \{1, \ldots, m\}$ is satisfied. Here, w_2 is the factor by which inadmissible heuristics are favored over the anchor search. If the check is violated for a specific search, it is suspended, and a state is expanded from $OPEN_0$. When a state x is expanded, its child states, x_c, are simultaneously updated in all priority queues, provided that x_c has not been expanded yet. It is only inserted into $OPEN_0$ if x_c has been expanded in any of the inadmissible searches but not in the anchor search. A state x_c that has been expanded in the anchor search is never re-expanded and is consequently never added to any of the priority queues.

E. Multi-Resolution Search

AMRA* is a search algorithm that employs various heuristics according to their resolution mapping, which designates the scale at which they are suitable. With at least one heuristic for each resolution used, a set of search heuristics is offered to get things going. As a result of the adaptability offered by the MHA* framework, additional heuristics can also be added for each resolution. The initialization of AMRA* with a consistent anchor heuristic is a crucial prerequisite, though. Resolution r = 0 is the name given to this heuristic. All states at the coarsest resolution must also correspond to a state at the finest resolution ($V_r \subset V_1\ \forall r > 1$) in order for the algorithm to be correct. For the search to be comprehensive, this requirement is essential.

AMRA* controls the anytime nature of its search as shown in Algorithm 2, and the sub-optimality of its solutions using parameters w_1 and w_2. At the end of each iteration, AMRA* returns a solution that is at most $w_1 \cdot w_2$ sub-optimal with respect to the graph $G_0 = (V_0, E_0)$. AMRA* decreases w_1 and w_2 to potentially improve the solution quality in the next iteration. To do so, AMRA* maintains the list $INCONS$. They are added back into the appropriate $OPEN_i$ for consideration by the search as shown in Algorithm 1.

The core function that searches for a path between x_s and G is Algorithm 1. Here, x_{goal} belongs to G and satisfies the termination condition. If no such x_{goal} is found before all $OPEN_i$ are exhausted, AMRA* terminates with failure. A simple round-robin is used to schedule policy over all heuristics. Every time a state is expanded at a particular resolution, it is removed from all inadmissible (non-anchor) heuristics at that resolution. This is because the cost to come of an in-admissibly expanded state is independent of the heuristic it was expanded from.



(a) $w_1 = 500$, $w_2 = 500$ (b) $w_1 = 126$, $w_2 = 126$ (c) $w_1 = 32.2$, $w_2 = 32.2$

(d) $w_1 = 8.8$, $w_2 = 8.8$ (e) $w_1 = 1.5$, $w_2 = 1.5$ (f) $w_1 = 1.1$, $w_2 = 1.1$

Fig. 5: AMRA* search iterations with decreasing w_1 and w_2 with starting state as (1000,1000) and goal state as (20,630). The last path obtained at $w_1 = 1$ and $w_2 = 1$ is the optimal path

IV. Experimental Results

A high-resolution image of the Sunderbans delta was obtained from [6]. This image was later processed using OpenCV to convert it into a black-and-white image. Following this, it was binarized to get an image that contained zeros and ones. This image is shown in Fig 6. Here, zero represents free space, and 1 represents obstacles. After this, heuristics and resolutions were chosen. The heuristics were chosen such that there is at least one admissible heuristic. The resolutions were chosen such that a node present in the coarser (lower) resolution is also present in all the finer (higher) resolutions below it. This way, sharing is maximized. Additionally, the anchor resolution is chosen as the lowest resolution, and the admissible heuristic was assigned to it. Then the parameters w_1 and w_2 were chosen to control the rate of solution generation. With this setup, the AMRA* algorithm is run to obtain solutions with increasing optimality with time.

The AMRA* algorithm was implemented using Python. A round-robin scheduler was made to choose among the heuristics. The OPEN priority queues associated with each heuristic were implemented using Python dictionaries. The algorithm was run to find the path between different start and goal states. The results were compared with the MRA*, MHA*, and A* algorithms. These results are shown in table I. The values of w_1 and w_2 were started at 500 for the anytime nature. They were reduced by a factor of 0.5 after each anytime iteration. The heuristics used were Euclidean and Manhattan distance. A four-connected action space was chosen. The resolutions chosen were (1,1). (3,3), (9,9), and (81,81), with the (1,1) resolution being the anchor resolution. The anchor heuristic was the Euclidean distance. The scheduler used to select the OPEN queues was a simple round-robin scheduler. The paths generated in some of the iterations are shown in Fig: 5.

From the results shown in table I, it can be seen that using AMRA* algorithm is the most efficient way to find an optimal path. The number of node expansions that occur in AMRA* is lesser than those occurring in A*, while the path cost is the same. The MRA* algorithm has fewer node expansions than A*, but the path is sub-optimal. The MHA* algorithm provides an optimal path for all the cases but has a greater number of node expansions than A*.

The results in table I are for when the USV is considered a point object which may not always be true. So, the AMRA* algorithm was run again by assuming the USV to have a finite size of 150m, which corresponds to a 3x3 pixel size. The settings for w_1 and w_2, the heuristics, and the

Start and Goal States	AMRA*		MRA*		MHA*		A*	
	Number of expansions	Path Cost	Number of expansions	Path Cost	Number of expansions	Path Cost	Number of expansions	Path Cost
[430, 177] to [605, 130]	6014	260	6330	261	6350	270	8810	260
[479, 296] to [630, 71]	32963	632	33915	674	34914	632	33481	630
[721, 51] to [674, 362]	41326	570	41327	620	43911	570	42404	570
[314, 154] to [323, 417]	15135	512	15577	518	16110	512	15148	512
[314, 154] to [289, 651]	68755	1038	69987	1040	76456	1038	69977	1038
[1000, 1000] to [20, 630]	190100	1442	185034	2270	278524	1442	204085	1442

TABLE I: Comparative results obtained with different start and goal states

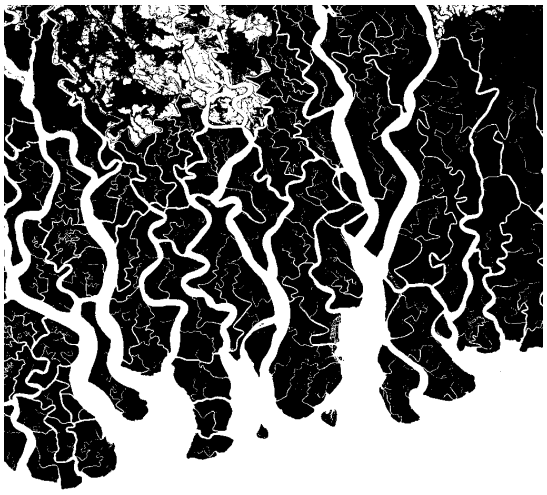


Fig. 6: Binary image of the Sunderbans Delta, which was used as the map

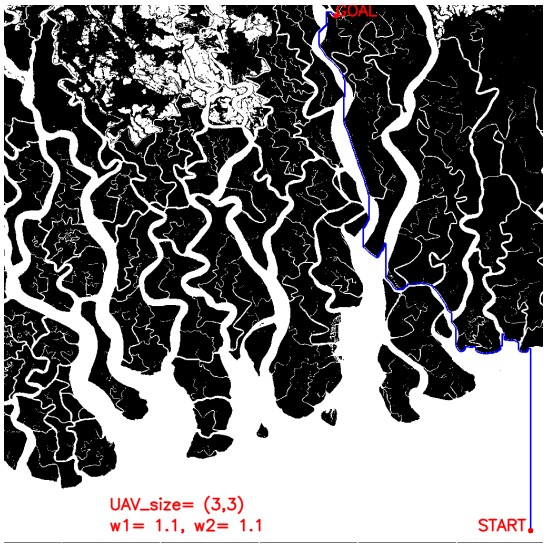


Fig. 7: Path obtained at $w_1 = 1.1$ and $w_2 = 1.1$ when considering the USV to be size 3x3 pixels

resolutions were the same as that of the case where the USV was considered as a point object. In this case, the AMRA* algorithm generated a solution after 12260 expansions with a cost of 1668, while the A* algorithm generated a solution after 15680 expansions with a cost of 1668. The path obtained is shown in Fig 7.

V. Discussion and Future Work

In this paper, an efficient way to plan paths for USVs through inland waterbodies in Sunderbans Delta has been demonstrated. The path planning was performed using the AMRA* algorithm, which is an Anytime Multi-Resolution Multi-Heuristic A* algorithm. It was shown that this algorithm is well-suited for path planning in inland waterbodies where the width of the water channel can increase or decrease drastically as it leverages multiple resolutions and multiple heuristics to quickly get a path to the goal state and with its anytime nature gives quick solutions. The solutions obtained were compared with previous algorithms of MRA*, MHA*, and A*. AMRA* has been shown to outperform them to generate an optimal path in the most efficient manner.

This paper discusses using AMRA* for planning a path for USVs in the Sunderbans Delta. But the AMRA* algorithm is not limited to just 2D path planning. This algorithm shows its greatest power when used in higher dimensional problems like 3D UAV path planning, manipulator path planning, etc. A very efficient implementation of this algorithm with the use of optimal data structures can be used to perform onboard path planning for high-dimensional problems.

References

[1] Saxena, D.M., Kusnur, T. and Likhachev, M., 2022, May. AMRA*: Anytime Multi-Resolution Multi-Heuristic A. In 2022 International Conference on Robotics and Automation (ICRA) (pp. 3371-3377). IEEE.

[2] Aine, S., Swaminathan, S., Narayanan, V., Hwang, V. and Likhachev, M., 2016. Multi-heuristic A*. The International Journal of Robotics Research, 35(1-3), pp.224-243.

[3] Du, W., Islam, F. and Likhachev, M., 2020. Multi-resolution A*. In Proceedings of the International Symposium on Combinatorial Search (Vol. 11, No. 1, pp. 29-37).

[4] Likhachev, M., Gordon, G.J. and Thrun, S., 2003. ARA*: Anytime A* with provable bounds on sub-optimality. Advances in neural information processing systems, 16.

[5] Pohl, I., 1970. Heuristic search viewed as path finding in a graph. Artificial intelligence, 1(3-4), pp.193-204.

[6] Zhang, Z., Ahmed, M.R., Zhang, Q., Li, Y. and Li, Y., 2023. Monitoring of 35-Year Mangrove Wetland Change Dynamics and Agents in the Sundarbans Using Temporal Consistency Checking. Remote Sensing, 15(3), p.625.