Min-Max Path Planning Algorithms for Heterogeneous, Autonomous Underwater Vehicles

Sandeep Banik, Sivakumar Rathinam and P.B. Sujit

Abstract—**Autonomous underwater vehicles (AUVs) are used in several maritime applications ranging from commercial to military use. In these applications, multiple AUVs are generally deployed from either a single base station or multiple base stations. AUVs constrained with time and resource rely on path planning algorithms to carry out the monitoring tasks efficiently. Given a set of targets and vehicles, this work addresses the problem of visiting each target at least once by a vehicle such that the minimum of the maximum distance traveled by any vehicle is minimized. Three insertion based heuristics are developed to solve the path planning problem. The performance of all the proposed heuristics are presented for 20 instances of the problem. The output solutions for one instance is also simulated in Neptus, an open source underwater mission planner.**

I. Introduction

Multiple autonomous underwater vehicles are commonly deployed in remote sensing applications like in situ ocean monitoring, water sampling, floor mapping and bathymetry [1]–[3]. In these applications, a vehicle is physically required to travel to a location to map using sonar or multi-resolution sensors or collect samples from the target region. The information collected must then be transferred by physically moving to a base station on the surface or to a data hub on the ocean floor. The latency involved in transferring the data from a target location to a base station is an important criterion in these applications and depends directly on the path travelled by the vehicles. This latency is also related to the freshness of information collected at each target. Given a set of vehicles and target locations/regions, this article considers the problem of finding a path for each vehicle such that each target is visited at least once and the maximum length of the path traveled by any vehicle is minimized. This problem is referred to as the min-max unmanned vehicle path planning problem (MUVPP).

The min-max path planning problem is a generalization of the standard Traveling Salesman Problem (TSP) and is NP-Hard [4]. Compared to the common min-sum objective, this problem is computationally challenging to solve for three reasons: i) finding tight lower bounds for the optimal cost is hard; ii) finding the optimal partitioning of targets among the vehicles is challenging and iii) finding an optimal path given the partition is non-trivial. There are several heuristics, approximation and exact algorithms available for this problem when all the vehicles are of the same type [4]. In this work, we consider the generalization of the min-max problem where the vehicles are heterogeneous. Heterogeneity could either arise if the vehicles are built different (they are structurally heterogeneous with different turning radius constraints) or carry different type of sensors (here, vehicles are functionally heterogeneous). There are several path planning algorithms that are developed for AUVs [5]–[8]. However, the focus of these path planners is mainly on designing optimal paths avoiding obstacles for a given source and destination location in the presence of ocean currents. The objective of this paper is different compared to the above planners.

In this article, we present fast heuristics with a-posteriori guarantees for solving the path planning problem. Specifically, we aim to understand the effectiveness of greedy heuristics for partitioning the targets among the heterogeneous vehicles. The greedy partitioning heuristics we consider are based on the well known insertion heuristics discussed in Rosengrantz et al. [9]. Given a partition of targets, we find an optimal path for each vehicle using the LKH heuristic. The insertion heuristics and the LKH are embedded in an iterative search procedure to find good feasible solutions. The performance of all the proposed algorithms are corroborated in terms of solution quality, bounds and computational time on several simulated instances.

II. Notations

There are n vehicles in the problem. Let D represent the set of depots and each depot can contain multiple vehicles. The set of common targets is denoted by T. T contains targets that can be visited by any vehicle. Let R_i denote the set of targets that must be visited by the i^{th} vehicle. The constraints that arise due to the presence of sets $R_i, i = 1, \cdots, n$ are also referred to as vehicle-target constraints. We assume that $R_i \cap R_j = \emptyset$ for all $i \neq j$. The set $V = D \cup T \cup_{i=1}^{n} R_i$ represents the set of all the vertices.

III. Overview of the approach

Insertion methods have shown some promise in solving single depot travelling salesman problem (TSP) as indicated by Rosenkrantz et al. [9]. Among the given insertion methods, we focus here on the farthest insertion, cheapest and nearest insertion to develop the insertion min-max (IMM) heuristics. The IMM heuristics have two stages in solving the MUVPP. The first stage is the initialization step which produces a feasible solution. The second stage is to perform an iterative search for load balancing which implicitly minimizes the maximum route for the given set of vehicles.

Sandeep Banik and P.B. Sujit are with IIIT Delhi, New Delhi – 110020, India.

Sivakumar Rathinam is with Department of Mechanical Engineering, Texas A&M University, College Station, Texas.

A. Initialization

We modify the the procedure in [4] to account for the vehicle target constraints. We first apply the insertion heuristic only for the common set of targets. Once we get subtours for visiting all the common targets, the vehicle-target sets are added suitably to produce a feasible solution. We provide more details in the following discussion.

Initially, consider that there are multiple depots each having a single vehicle. Let each depot with a vehicle be considered a sub-tour ST_k consisting only the depots itself. An insertion method (cheapest, nearest or farthest) [9] constructs a sequence of subtours for each vehicle,

$$ST_1, ST_2, ..., ST_{|D|}. \tag{1}$$

In the next step, for each vehicle k, we add the targets in the set R_k to ST_k using another insertion heuristic. This generates a feasible solution. If a depot contains more than one vehicle, to avoid any ambiguity in the application of the insertion heuristic, the location of each vehicle in the depot is perturbed by a small amount to generate a distinct location for each vehicle.

The tours obtained for each vehicle is then passed on to the Lin-Kernighan-Helsgaun (LKH) solver to generate a TSP pertaining to each vehicle. The tour with the longest length is referred to as the maximum tour.

B. Iterative search

In this step, we iteratively balance the load distributed among the vehicles. We choose the maximum tour and aim to remove a common node from this tour. Suppose, the i^{th} node in this tour is denoted by $node_i$. Let $node_i$ also be a common node. The node to be removed is calculated using, the cost,

$$\begin{aligned} d(node_i, node_{i-1}) + d(node_i, node_{i+1}) \\ -d(node_{i-1}, node_{i+1}) \end{aligned} \tag{2}$$

where $d(node_i, node_{i+1})$ is the euclidean distance between the node i and node $i+1$. Equation (2) provides the estimated reduction in this tour's cost if $node_i$ is removed. This estimate is computed for each of the common targets. Then, the common node that provides the maximum reduction in cost is removed first.

Next, the removed node is appended to any one of the remaining tours again using a similar estimated change in costs provided by 2. Inserting a potential node, say $node_i$ between nodes $node_{i-1}$ and $node_{i+1}$ is going to increase the cost by the amount given in (2). Therefore, the estimated increase of adding the removed node for adjacent pair of targets in all the tours is first computed. Then, the insertion corresponding to the least increase in the cost is computed, and the removed node to added correspondingly. Next, the new tours obtained for each vehicle is passed on to the LKH solver to generate a TSP solution for each vehicle. The tour with the new longest tour length is compared against the previous tour length and if the new maximum is lower than the previous one, the old tours are replaced by the new tours.

Let $\bar{L}_{th}$ denote the *average* tour length threshold and β represent the tour length threshold. The iterative search procedure is repeated until the difference between the maximum tour length and the minimum tour length divided by the minimum tour length does not exceed β, or the difference between the average tour lengths corresponding the last two iterations is greater than $\bar{L}_{th}$. The tour length threshold provides a degree of relaxation to meet the the vehicle target constraints. If no improvement is seen in the iterative search procedure for 5 iterations, the value of β is doubled and this continues until convergence.

IV. The IMM Heuristics

A. Initialization

In this section, we give more details of the algorithms along with the pseudo code. We first provide few definitions before we present the pseudo code.

- Given a travelling salesman graph (V, E), where V denotes the nodes and E the edges, a tour on a subset S of V will be called a subtour of (V, E).
- $d(i, j)$ is the distance function between any two nodes i and j.
- Given a subtour T and node p, we define a distance $d(T, p)$ between T and p as,

$$min(d(x, p) \text{ for } x \text{ in } T) \tag{3}$$

- $perturb(D_k, r)$ is the perturbation of the node D_k by a magnitude r.
- Let ST_{kj} denote the tour for the k^{th} vehicle computed during iteration j.
- $insertion(d(ST_{kj}, x)$ for x in $T - ST_{kj})$ defines the selection procedure of a node or target using either nearest, cheapest or farthest insertion.

With all the required definitions and functions, the initialization procedure is as follows:

1) Create an empty set ST.
2) Perturb the location of each vehicle by a magnitude of r. (Here $r = 0.1$)
3) For all the vehicles k, in the depots D append the depot location of each vehicle to ST_{k1}. This will be first subtour of all the vehicles.
4) Repeat until the set of common nodes goes to null.
   a) Iterate over all the vehicles and determine the node to be inserted as,

$$\begin{aligned} d_{in}^k \longleftarrow insertion(d(ST_{kj}, x) \\ \text{for } x \text{ in } T - ST_{kj}) \end{aligned} \tag{4}$$

   For each vehicle, the insertion will provide the node which satisfies the above equation.

   b) For each vehicle and the corresponding node (target) obtained from the previous step, add the node so as the minimize

$$d(i, node) + d(node, i+1) - d(i, i+1) \tag{5}$$

where $i \in ST_k$ and $node$ is the farthest node for the corresponding vehicle k.

c) A set of subtours for each vehicle k is obtained in the above step. The node to be removed from the common nodes/targets is determined by a common target whose removal will lead to the largest reduction in the lengths of the tours.

5) For all the functional constraint of each vehicle, append onto the existing subtour of each vehicle using the farthest insertion.
6) Pass the acquired approximate tours of each vehicle to the LKH solver.

B. Pseudo code initialization

Algorithm 1 IMM initialization

```
1: Input: $T$, set of common targets.
   $D$, set of depots.
   $R_k$, set of function constraints for each vehicle $k \in \{1, ...n\}$.
   $V = D \cup T \cup R_k$, is total number of vertices.
   $n$, is total number of vehicles.
   $ST \longleftarrow \emptyset$;
   $Nn \longleftarrow length(T)$
2: Output: $ST_k$for $k \in n$
3: for $k \in D$ do
4:     $Dp_k \longleftarrow prerturb(D_k, r)$
5:     for $k \in Dp$ do
6:         $ST_k \longleftarrow Dp_k$
7:     end for
8: end for
9: repeat
10:    $d_{in} \longleftarrow \emptyset$
11:    for $k \in Dp$ do
12:        $d_{in}^k \longleftarrow insertion(d(ST_k, x)$ for $x$ in $T - ST_k)$
13:    end for
14:    $k \longleftarrow \min(d_{in})$
15:    $ST_k = min((d(i,T_j)+d(T_j,i+1)-d(i,i+1))$for $i \in ST_k)$
16:    $T \longleftarrow T \cap i = \emptyset$ (remove node i from the set $T$)
17: until $T \longleftarrow \emptyset$
18: for $k \in R_k$ do
19:    $ST_k \longleftarrow insertion(R_k)$
20: end for
21: for $k \in D$ do
22:    $ST_k \longleftarrow LKH(ST_k)$
23: end for
```

C. Iterative search

Given the optimal/approximate tour of each vehicle, a threshold is computed for the difference between the extreme lengths of the tours, *i.e.*, let $\{L_1, L_2, ..., L_n\}$ be the lengths of the tour in the descending order, $\frac{L1-Ln}{Ln} \geq \beta$, where β is the threshold and a threshold for the average length of the tour $\bar{L}_{th}$. The goal of the iterative search is to minimize the maximum length of the tour and as the solution converges the load for each vehicle is balanced implicitly. The algorithm for iterative search procedure is given in Algorithm 2.

Algorithm 2 IMM iterative search

```
1: Input: $ST_k$ for $k \in \{1, 2, ..., n\}$,
   $L$, tour lengths of $ST$
   $r$, tour length threshold.
   $\bar{L}_{th}$, average tour length threshold.
2: Output: $ST$ for $k \in \{1, 2, ..., n\}$
3: $L_s \longleftarrow reverse\ sort(L)$,
4: $\bar{L} \longleftarrow average(L)$,
5: $\bar{L}_P \longleftarrow (\bar{L})$
6: $count \longleftarrow 0$
7: while $\frac{L_s(0)-L_s(n)}{L_s(n)} \geq \beta \ || \ \frac{\bar{L}_P-\bar{L}}{\bar{L}_P} \geq \bar{L}_{th}$ do
8:     $\bar{L}_P \longleftarrow (\bar{L})$
9:     $L_{max} \longleftarrow k[L_s(0)]$
10:    $R_{max} \longleftarrow route_{save}(ST_{L_{max}}) \longleftarrow Eqn2$
11:    for $l$ in $R_{max}$ do
12:        $L_{in} \longleftarrow T(l)$
13:        $d_{in} \longleftarrow \emptyset$
14:        for $j$ in $k$ do
15:            $d_{in}^j \longleftarrow \min(route_{insert}(ST_j, L_{in}))Eqn2$
16:        end for
17:        $d_{min} \longleftarrow min(d_{in})$
18:        $ri \longleftarrow argmin(d_{in})$
19:        $C_{ST} \longleftarrow copy(ST)$
20:        $C_{ST_{ri}} \longleftarrow append(ST_{ri}, L_{in})$
21:        $C_{ST_{L_{max}}} \longleftarrow remove(ST_{L_{max}}, L_{in})$
22:        $C_{LKH} \longleftarrow LKH(C_{ST})$
23:        $P_{obj} \longleftarrow (L_s(0))$
24:        $N \longleftarrow C_{ST}$
25:        $C_{obj} \longleftarrow \max(N)$
26:        if $C_{obj} < Pobj$ then
27:            $ST \longleftarrow C_{LKH}$
28:            break
29:        end if
30:    end for
31:    $\bar{L}_{th}$, average tour length threshold.
32:    $L_s \longleftarrow reverse\ sort(L)$,
33:    $count \longleftarrow count + 1$
34:    if $mod(count, 5) = 0$ then $\beta \longleftarrow 2\beta$
35:    end if
36: end while
```

V. Simulation results

In this section, we compare the IMM algorithm against the MD algorithm developed by Wang et al. [10] for a set of test instances. The MD algorithm was modified in order to accommodate the functional constraint. In the first step, the clustering LP is solved as presented in the appendix. Next, we add the functional constraint for each vehicle and solve it using the LKH solver and proceed to the local search and perturbation. In the local search and perturbation the functional constraints are tracked in order to comply with vehicle target constraints. Twenty sets of data were generated with varying

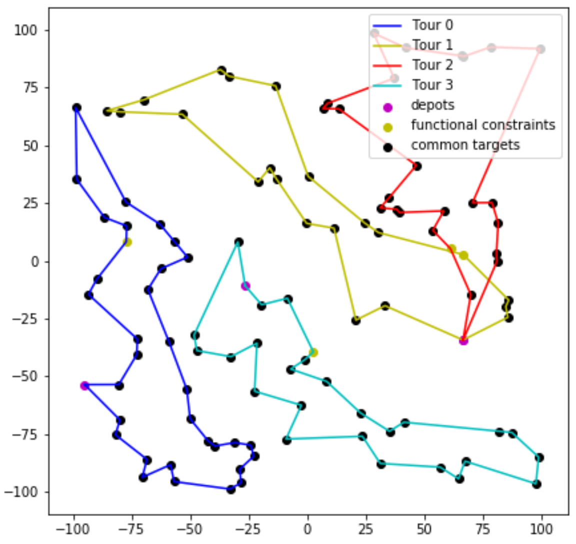

Fig. 1. IMM tour illustration

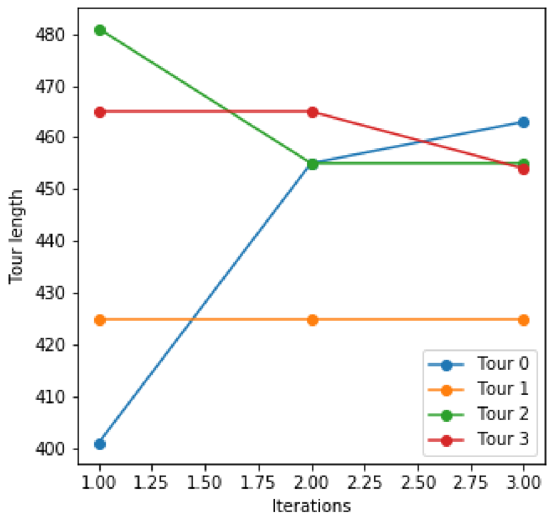

Fig. 2. IMM tour length convergence

number of depots, common targets and vehicle per depot. Each vehicle was considered to contain a single vehicle target constraint. The modified MD algorithm and IMM algorithm with the cheapest, nearest and farthest insertion methods were all written in python and pytsp was used to interface with the LKH solver. The detailed results are presented in the Table I. In Table I the first column indicates the instance number, the second column indicates the number of depots. The third column indicates the targets or common nodes, the fourth column indicates the number of vehicle in each depot. 1,1,2 indicates the first depot having one vehicle, the second depot having one vehicle and the third depot having two vehicles. The fifth column indicates the number of functional constraints. In the current work only 1 functional constraint was tested. The IMM algorithm is performed using the cheapest, nearest and farthest insertion methods. The maximum tour length and average tour length for each of the IMM algorithm and the MD algorithm are presented.

From the results we can infer that the IMM cheapest algorithm performs the best 8 out of 20 instances, the IMM nearest algorithm performs 1 out of the 20 instances and the MD performs 11 out of the 20 instances. The IMM algorithm perform better than the MD almost for half of the instances. The computational times required for most of the instances are lower than the time required for the MD algorithm. In addition to the 20 instances, 200 random instances were generated with random number of depots, vehicles per depots and number of depots. Out of the 200 random instances the IMM algorithm performed 73 times better than the MD algorithm, whereas the MD algorithm was better 127 times.

An illustration of the output of the IMM algorithm is

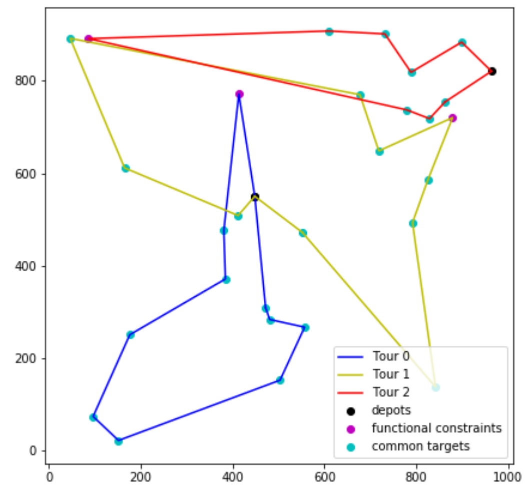

Fig. 3. IMM tour result

presented in the figure 1. The number of common targets used were 200, with 3 depots and number of vehicle in each depot as 1,2,1. The illustration presented is of the IMM nearest algorithm The convergence of the tour length are shown in the figure 2.

The output of the IMM algorithm provides the euclidean path for each of the vehicles present in each of the depots. However, a real world AUV will not follow euclidean paths along the given nodes. Therefore, Neptus [11] simulator is used along with Dune [12] from LSTS to simulate the path in a mission planner. Figures 4-6 show the trajectory of the

TABLE I
INSTANCE RESULTS FOR MD AND IMM ALGORITHM

Instances	Data				IMM cheapest			IMM nearest			IMM farthest			MD		
	Number of depots	Number of targets	Number of vehicles per depot	Number of functional constraint	Max tour length	Average tour length	Time(s)	Max tour length	Average tour length	Time(s)	Max tour length	Average tour length	Time(s)	Max tour length	Average tour length	Time(s)
HMM1	3	20	1	1	377	366.33	0.354	377	366.33	0.307	540	507.0	0.208	370	360.66	2.678
HMM2	3	100	1	1	728	718.66	11.287	732	711.0	12.217	1071	1043.0	4.937	711	703.66	21.070
HMM3	3	200	1	1	739	719.66	151.147	803	796.33	34.776	1427	1401.0	33.52	847	845.33	249.349
HMM4	2	20	1	1	471	459.0	0.238	471	459.0	0.185	566	548.0	0.189	517	515.0	1.946
HMM5	2	75	1	1	886	878.5	4.81	886	878.5	2.277	1065	1063.0	2.276	794	793.0	21.46
HMM6	2	160	1	1	980	979.0	76.91	1050	1047.5	20.704	1484	1481.0	17.618	994	993.0	123.559
HMM7	5	50	1	1	563	521.8	3.629	551	501.4	1.058	655	617.0	1.015	459	448.6	5.1906
HMM8	5	112	1	1	619	517.8	15.933	632	535.4	7.531	896	870.2	7.074	467	457.2	26.075
HMM9	5	150	1	1	640	598.8	37.599	652	595.6	19.834	1092	1039.0	15.555	617	606.4	34.562
HMM10	7	50	1	1	516	430.42	3.12	510	442.42	1.39	649	608.71	1.153	427	385.14	8.196
HMM11	7	101	1	1	510	413.28	20.463	518	430.71	14.652	809	759.28	5.926	462	387.0	9.33
HMM12	7	155	1	1	494	409.28	83.16	543	499.14	18.53	941	890.71	18.10	512	456.0	25.027
HMM13	3	50	1,1,2	1	501	494.0	1.724	521	480.0	0.943	680	647.25	0.944	525	476.5	3.308
HMM14	3	50	1,2,2	1	482	404.0	6.418	482	391.4	5.570	677	656.2	1.008	377	359.4	6.1373
HMM15	3	70	1,3,2	1	474	436.33	5.68	526	445.0	7.109	722	682.83	2.279	410	404.0	6.257
HMM16	4	80	1,2,2,2	1	449	398.71	6.642	469	397.0	5.324	656	623.14	3.306	406	365.0	14.257
HMM17	4	110	2,2,3,1	1	465	402.37	15.95	487	433.5	8.397	848	805.87	10.55	515	443.62	8.35
HMM18	4	121	2,3,1,1	1	429	383.85	37.50	557	419.85	34.171	805	781.57	9.438	473	457.14	24.266
HMM19	5	100	1,2,2,1,1	1	604	441.12	39.89	606	436.37	33.88	779	752.87	9.57	387	363.37	17.660
HMM20	5	200	3,2,2,2,3	1	606	401.58	146.81	501	429.58	42.12	868	825.16	40.153	554	413.41	23.80
Best solutions					8			1			0			11		

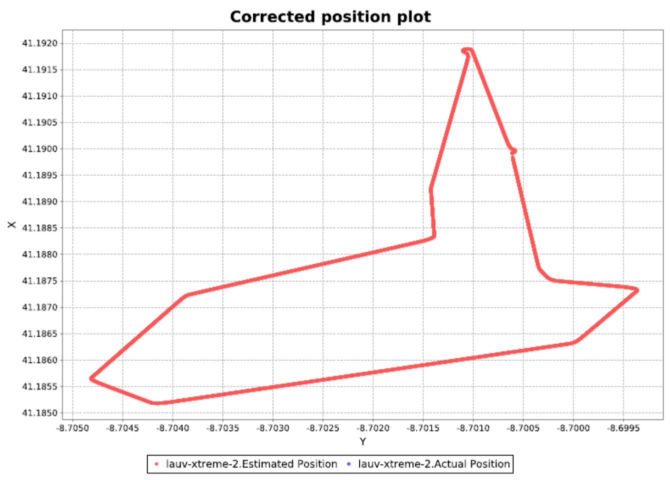

Fig. 4. Neptus tour 0

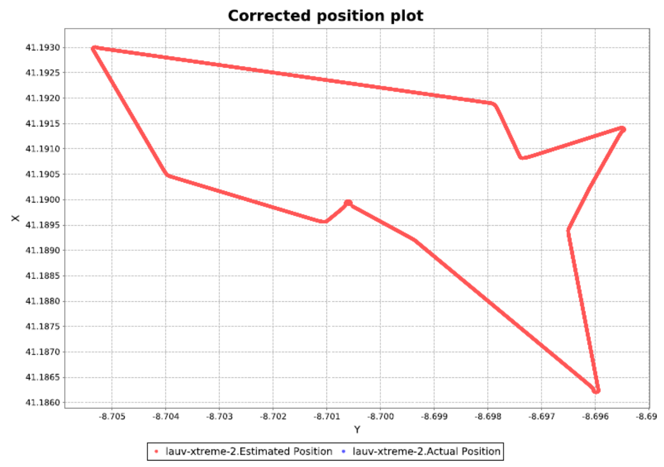

Fig. 5. Neptus tour 1

vehicles obtained after simulating in Neptus.

The instance created is of 25 common targets, 2 depots, and one vehicle in a depot and two vehicles in another depot. The IMM algorithm was evaluated with nearest, cheapest and farthest algorithm. The best tour was obtained by using the cheapest algorithm. The output of the tour is shown in the figure 3. The vehicle used in Neptus was lauv-xtreme-2. The mission

VI. CONCLUSION

We developed three heuristic using insertion methods namely; IMM cheapest, IMM farthest, IMM nearest for solving the min max MUVPP problem with functional constraint. The algorithm is compared against the modified MD algorithm. The insertion heuristics and the MD algorithm are compared against 20 instances by varying number of depots, number of customers and vehicles per depot. From the results it is observed that the insertion heuristic performed better 9

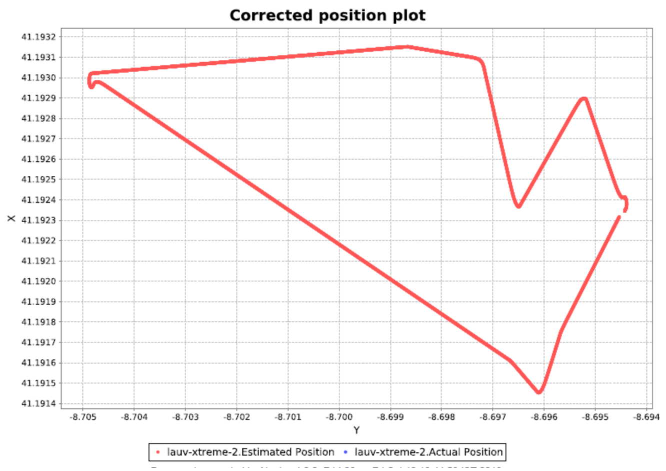

Fig. 6. Neptus tour 2

out of 20 times and the MD performed 11 out of 20 times. For majority of the instances the insertion heuristics takes lower computational time than the MD algorithm.

The insertion algorithm has shown some promise in solving the min max MUVPP problem and can further be improved using some perturbation techniques similar to the ones used in MD algorithm. The future work includes using an algorithm similar to simulated annealing for perturbation of the tours and comparing the IMM against the MD and its variations. Furthermore, the tours developed in this paper do not consider kinematic constraints, therefore, the problem of one is a set TSP [13] needs to be incorporated into IMM to solve for real world vehicle paths taking kinematic constraints into account. Also, the other metrics like fuel/energy consumed on a path can be considered which will indirectly use the current information while determining paths. Also the proposed work can be integrated to the mission planning architecture as a future work [14].

VII. APPENDIX

A. Clustering Problem Formulation

The min-max problem is formulated on an un-directed graph $G = (V, E)$, where E is a set of edges joining any two vertices in the set V. For any edge $e \in E$, let c_e^k represent the cost of traveling e for vehicle k. For any vehicle $k \in \{1, 2, ..., n\}$ and a target $i \in T$, a binary variable x_i^k is 1 if the target i is visited by the vehicle k and 0 otherwise. A relaxed linear programming clustering for the min max MUVPP problem from [4] is formulated as,

$$min\ max \sum_{k=1}^{n} \sum_{e \in E} c_e^k x_e^k \tag{6}$$

$$subject\ to: \tag{7}$$

$$\sum_{e \in E} x_e^k = \frac{T}{n}, \quad \forall k \in \{1, 2, ..., n\} \tag{8}$$

$$\sum_{k=1}^{n} x_i^k = 1, \quad \forall i \in T \tag{9}$$

$$x_i^k = \{0, 1\}, \quad \forall k \in \{1, 2, ..., n\}, \forall i \in T \tag{10}$$

The above formulation follows the work of Carlsson et al. [4] where the constraint in the *equation 2* represents the load balancing of each vehicle with only common targets. For the given number of vehicles n and number of targets T, it is assumed that there exists two integers p and q such that $T = pn + q$. If $\frac{T}{n}$ is an integer, then q is zero; if not q is the residue. The constraint in the *equation 3*, ensures that each target $i \in T$ is visited by some vehicle.

REFERENCES

[1] S. W. S. N. J. Seiler, T. Anderson and S. Hill, "Autonomous underwater vehicle (auv) for mapping marine biodiversity in coastal and shelf waters: Implications for marine management," *IEEE OCEANS*, p. 16, 2010.

[2] T. P. L. B. e. a. R. B. Wynn, V. A. I. Huvenne, "Autonomous underwater vehicles (auvs): their past, present and future contributions to the advancement of marine geoscience," *Marine Geology*, p. 451468, 2014.

[3] J. B. D. K. S. Xuri Yu, T.Dickey, "The application of autonomous underwater vehicles for interdisciplinary measurements in massachusetts and cape cod bays," *Continental Shelf Research*, pp. 2225–2245, 2002.

[4] J. G. Carlsson, D. Ge, A. Subramaniam, A. Wu, and Y. Ye, "Solving min-max multi-depot vehicle routing problem ," 2007.

[5] D. Kruger, R. Stolkin, A. Blum, and J. Briganti, "Optimal auv path planning for extended missions in complex, fast-flowing estuarine environments," in *Robotics and Automation, 2007 IEEE International Conference on*, pp. 4265–4270, IEEE, 2007.

[6] J. Witt and M. Dunbabin, "Go with the flow: Optimal auv path planning in coastal environments," in *Australian Conference on Robotics and Automation*, vol. 2008, 2008.

[7] C. Shen, Y. Shi, and B. Buckham, "Integrated path planning and tracking control of an auv: A unified receding horizon optimization approach," *IEEE/ASME Transactions on Mechatronics*, vol. 22, no. 3, pp. 1163–1173, 2017.

[8] J. Cao, J. Cao, Z. Zeng, B. Yao, and L. Lian, "Toward optimal rendezvous of multiple underwater gliders: 3d path planning with combined sawtooth and spiral motion," *Journal of Intelligent & Robotic Systems*, vol. 85, no. 1, pp. 189–206, 2017.

[9] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, *An analysis of several heuristics for thetraveling salesman problem*, pp. 45–69. Dordrecht: Springer Netherlands, 2009.

[10] X. Wang, B. Golden, and E. Wasil, "The min-max multi-depot vehicle routing problem: heuristics and computational results," *Journal of the Operational Research Society*, vol. 66, no. 9, pp. 1430–1441, 2015.

[11] "Neptus, command and control software." https://lsts.fe.up.pt/toolchain/neptus. Accessed: 2018-10-20.

[12] "Dune, unifined navigation environment." https://www.lsts.pt/toolchain/dune. Accessed: 2018-10-20.

[13] S. G. Manyam, S. Rathinam, D. Casbeer, and E. Garcia, "Tightly bounding the shortest dubins paths through a sequence of points," *Journal of Intelligent & Robotic Systems*, vol. 88, pp. 495–511, Dec 2017.

[14] P. S. Dias, R. M. Gomes, and J. Pinto, "Mission planning and specification in the neptus framework," in *IEEE International Conference on Robotics and Automation*, pp. 3220–3225, IEEE, 2006.