Multi-Goal Feasible Path Planning Using Ant Colony Optimization

Brendan Englot and Franz Hover

***Abstract*— A new algorithm for solving multi-goal planning problems in the presence of obstacles is introduced. We extend ant colony optimization (ACO) from its well-known application, the traveling salesman problem (TSP), to that of multi-goal feasible path planning for inspection and surveillance applications. Specifically, the ant colony framework is combined with a sampling-based point-to-point planning algorithm; this is compared with two successful sampling-based multi-goal planning algorithms in an obstacle-filled two-dimensional environment. Total mission time, a function of computational cost and the duration of the planned mission, is used as a basis for comparison. In our application of interest, autonomous undewater inspections, the ACO algorithm is found to be the best-equipped for planning in minimum mission time, offering an interior point in the tradeoff between computational complexity and optimality.**

I. INTRODUCTION

Multi-goal planning is a task which arises in many robotics applications, from autonomous inspection and surveillance to manufacturing and assembly. It combines the challenging requirements of planning feasible point-to-point trajectories in obstacle-filled — and possibly high-dimensional — state spaces with the complexity of combinatorial optimization. In planning inspections and surveillance missions for sensor coverage, several studies have solved multi-goal planning as a variant of the Chinese postman problem (CPP), in which a robot must sweep out a set of graph edges to complete an inspection [1], [2], [3]. Surveillance applications are also solved by mapping mission goals to graph nodes and modeling the multi-goal planning problem as some variant of the traveling salesman problem (TSP) [4], [5]. Multi-goal problems in industrial manufacturing applications have also been solved using the TSP, including spot welding [6], [7] and measuring fabricated parts to ensure tolerances have been met [8]. Additionally, TSP solutions have been developed for robotic systems with kinodynamic constraints [9], [10]. Although not all planning tasks require the mission to begin and end in the same configuration, this feature of the TSP is a useful requirement for periodic surveillance trajectories and manufacturing tasks. We assume in this paper that a robot's goals — be they specific configurations or end-effector positions — can be modeled as the nodes of a graph, and consequently that the multi-goal planning problem requires the solution of a TSP.

One characteristic feature of existing solution methods for multi-goal planning is the two-step process of constructing a graph in a robot's configuration space, followed by the construction of a tour which visits all of the goals on this graph. This is the approach we will take in this paper. In obstacle-filled environments with many goals, however, constructing a graph which describes feasible paths over all goal-to-goal pairings is a costly task. To decide which goal-to-goal pairings are collision-checked, and subsequently, to compute an efficient tour over the resulting feasible paths, we will use Ant Colony Optimization (ACO).

ACO is an evolutionary metaheuristic in which virtual ants walk along a graph and communicate with one another by depositing and removing pheromones from the edges of the graph, changing the desirability of the edges [11]. It has met with particular success in solving the TSP, and a variety of ACO procedures have been developed for this purpose, Ant Colony System (ACS) being the most aggressive and widely used of these variants [12]. ACO has also been extended to planning problems for autonomous systems, such as terrain coverage in obstacle-free workspaces [13], [14], point-to-point planning in obstacle-filled workspaces, [15], [16], [17], planning in dynamic environments, [18], and simultaneous planning and task allocation in distributed systems [19]. Despite the success of ACO in solving the TSP, planning a tour among multiple goals in obstacle-filled environments has not been addressed using ACO (although it is stated among the future goals of [15]).

In this study we use ACO to plan a feasible tour among a set of goals in an obstacle-filled environment, in which all goal locations and obstacles are known *a priori*. However, feasible goal-to-goal paths are not known *a priori*, and a key challenge is for the algorithm to decide which goal-to-goal feasible paths will be computed and considered for use in the tour. Our application of interest is the inspection of complex marine structures by the Bluefin-MIT Hovering Autonomous Underwater Vehicle (HAUV), [3], [20] a problem requiring collision-free travel to hundreds of goal locations with a five degree-of-freedom autonomous vehicle. Consequently, the ACO algorithm must be fast and scalable, and so a sampling-based planning method will be used to find feasible point-to-point paths between goals. Rather than exhaustively computing all-pairs feasible paths among the goals, point-to-point paths will be computed as needed. When an ant decides to walk along a new graph edge, a bi-directional rapidly-exploring random tree (RRT) [21] is used to find a path. Unlike the related work of [15], we use the ant colony metaheuristic to decide which goal to visit next, rather than to carve out the physical paths themselves.

We compare our algorithm with two successful algorithms previously applied to multi-goal feasible motion planning

This work was supported by the Office of Naval Research under Grant N00014-06-10043, monitored by Dr. T.F. Swean

B. Englot and F. Hover are with the Department of Mechanical Engineering, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge MA 02139 USA benglot@mit.edu hover@mit.edu

for autonomous systems in obstacle-filled environments. In Section II the first comparison algorithm is introduced, an exhaustive all-pairs search for feasible paths, followed by the use of the best-known heuristic upper bound on the TSP [8]. In Section III the second comparison algorithm is introduced, a fast algorithm designed to require as little collision checking as possible by iteratively computing and checking the minimum spanning tree (MST) [6]. Section IV presents ACO for feasible multi-goal planning, and Section V presents results comparing all three algorithms using a two-dimensional robot test case.

II. EXHAUSTIVE ALL-PAIRS ALGORITHM

First, we implement an exhaustive all-pairs algorithm with the goal of finding the shortest possible obstacle-laden TSP that can be computed in polynomial time. The algorithm is based on [8], in which a probabilistic roadmap (PRM) [22] is iteratively constructed until all goals from the planning mission are attached to a single connected component. All-pairs shortest paths are then computed among the goals, so each goal-to-goal cost reflects the cost of the shortest-known feasible path from goal i to goal j among obstacles. The goal-to-goal costs are stored in a weighted adjacency matrix, and any TSP algorithm may be used to find a tour among the goals. No specific TSP algorithm is adopted by [8], but the best polynomial-time upper bound on the TSP is achieved using Christofides' algorithm [23]. This algorithm provides a 3/2 upper bound on the cost of the optimal TSP solution by adding together a MST and a minimum-cost perfect matching that pairs up the odd-degree vertices of the MST. The resulting structure is an Eulerian graph, on which an Eulerian walk may be executed that visits every edge once, and consequently, every goal at least once. This procedure is outlined in Algorithm 1a.

An important requirement of Christofides' 3/2 upper bound is that the edge weights of the adjacency matrix must satisfy the triangle inequality. This is not guaranteed to hold if we work with feasible paths constructed using random sampling. In the worst case, the all-pairs shortest paths computation over the PRM will find that the shortest path from goal i to goal j must pass through other intermediate goals, rather than going directly from i to j. Since the goal-to-goal costs in our adjacency matrix correspond to the shortest existing paths, the triangle inequality will, in the worst case, be satisifed at equality instead of being a strict inequality.

Another limitation of this method is the scalability of PRM construction. In obstacle-filled and high-dimensional configuration spaces with many goals, joining all goals into a single connected component may be very costly, and especially challenging if we are undertaking a kinodynamic planning task. To quantify the complexity of this algorithm and allow comparison with the competing algorithms introduced below, we can describe the computational cost of this method as $O(n^3)+C*O(n^2)$. Since the competing algorithms described below use the RRT as a goal-to-goal planner, we will capture the cost of the PRM as the equivalent (in the worst case) of calling an RRT across every goal-to-goal pairing in the graph (this modified procedure, used for comparison only, is outlined in Algorithm 1b). The $O(n^3)$ term describes the complexity of Christofides' TSP algorithm, which is dominated by the worst-case complexity of a minimum-cost perfect matching over n goals. It also describes the cost of the all-pairs shortest path computation in the worst case of a complete graph with an RRT connection between all pairs of goals. The $C*O(n^2)$ term describes the cost of building feasible paths between all pairs of goals, where C is the representative cost of calling a goal-to-goal sampling-based planner. C is kept outside of the parentheses to denote its potentially high and hard-to-predict cost in a higher-dimensional space with many obstacles. Although the all-pairs method may not be scale-friendly, it provides a high-quality tour in polynomial time as a basis for comparison with other algorithms.

Algorithm 1a $RobotTour = AllPairsAlg1(Goals, Obstacles)$

```
1: $AdjMat \leftarrow EuclideanDistances(Goals)$
2: $RoadMap \leftarrow PRM(Goals, Obstacles)$
3: $NewAdjMat \leftarrow AllPairsShortestPaths(RoadMap, Goals)$
4: $RobotTour \leftarrow ChristofidesAlgorithm(NewAdjMat)$
5: return $RobotTour$
```

Algorithm 1b $RobotTour = AllPairsAlg2(Goals, Obstacles)$

```
1: $AdjMat \leftarrow EuclideanDistances(Goals)$
2: $RoadMap \leftarrow \emptyset$
3: for $Edge_{ij} \in AdjMat$ do
4:     $FeasiblePath_{ij} \leftarrow RRT(Edge_{ij}, Obstacles)$
5:     $RoadMap \leftarrow FeasiblePath_{ij}$
6: end for
7: $NewAdjMat \leftarrow AllPairsShortestPaths(RoadMap, Goals)$
8: $RobotTour \leftarrow ChristofidesAlgorithm(NewAdjMat)$
9: return $RobotTour$
```

III. LAZY MST ALGORITHM

To achieve greater speed, a lazy algorithm can be devised which solves many fewer point-to-point path queries than an all-pairs approach. This algorithm is based on the lazy-GMGP algorithm of [6], which approximates the optimal TSP solution using the MST heuristic. Similar to the Christofides heuristic, the MST heuristic yields a tour within a factor of two of optimality by doubling the edges of the MST and traversing every edge, visiting every goal node in the process. This upper bound is also subject to the satisfaction of the triangle inequality, which, unlike the all-pairs algorithm, may be violated severely in this case.

A feasible MST cannot be computed unless all-pairs feasible paths have been checked, so a lazy MST is computed instead which has not been checked for collisions and is based solely on goal-to-goal Euclidean norms. After computing the lazy MST, every edge on the tree is checked for collisions, and is replaced by a feasible path if a collision occurs. In this study we use a bi-directional RRT as the point-to-point planner for finding these paths. This procedure requires the computation of $n-1$ feasible paths to check the MST, a significant reduction from the collision-checking overhead of the all-pairs approach.

Because of the signficant savings, lazy-GMGP repeats the MST-finding procedure, using the feasible paths of the previous iteration to find a new lazy MST and check unexplored goal-to-goal paths until an iteration's true feasible MST cost falls within a designated constant factor of the lazy MST cost. For the purposes of comparison with our ACO algorithm, the iterative MST-finding and checking procedure will terminate when the cost of the feasible MST fails to improve over three consecutive iterations. The implementation of this algorithm as an iterative procedure makes it likely that severe violations of the triangle inequality (when the path from i to j is much higher than an available path from i to k to j) will be removed from the tree and replaced by shorter paths.

The original lazy-GMGP algorithm is designed for planning a tour among goal groups, in which only a single goal from each group needs to be visited. In our application, every goal is mandatory, and a regular MST can be used in lieu of the Steiner tree solution procedure of [6]. The computation time required by this method can be described as $N*(O(n^2)+C*O(n))$, where N is the number of iterations in which the lazy MST is computed, the $O(n^2)$ term captures the complexity of computing the MST over a complete graph (since all goal-to-goal pairings are considered as candidate edges), and the $O(n)$ term captures the worst-case number of RRT calls required per iteration. Once again, C represents the cost of calling the RRT, which can be signficant in problems of high dimension and in obstacle-filled environments. The lazy MST algorithm is summarized in Algorithm 2.

Algorithm 2 $RobotTour = LazyMSTAlg(Goals, Obstacles)$

```
1: $AdjMat \leftarrow EuclideanDistances(Goals)$
2: $UnclearedEdges \leftarrow GetEdgePairs(Goals)$
3: $ClearedEdges \leftarrow \emptyset$
4: while $FailCount < 3$ do
5:   $NewMSTCost \leftarrow 0$
6:   $LazyMST \leftarrow ComputeMST(AdjMat)$
7:   for $Edge_{ij} \in LazyMST$ do
8:     if $Edge_{ij} \in UnclearedEdges$ then
9:       $FeasiblePath_{ij} \leftarrow RRT(Edge_{ij}, Obstacles)$
10:      $ClearedEdges \leftarrow ClearedEdges \cup Edge_{ij}$
11:      $UnclearedEdges \leftarrow UnclearedEdges \setminus Edge_{ij}$
12:      $AdjMat(i,j) \leftarrow PathCost(FeasiblePath_{ij})$
13:    end if
14:    $NewMSTCost \leftarrow NewMSTCost + AdjMat(i,j)$
15:  end for
16:  if $NewMSTCost < BestMSTCost$ then
17:    $BestMSTCost \leftarrow NewMSTCost$
18:    $BestMST \leftarrow LazyMST$
19:    $FailCount \leftarrow 0$
20:  else
21:    $FailCount \leftarrow FailCount + 1$
22:  end if
23: end while
24: $RobotTour \leftarrow ConvertToTour(BestMST)$
25: return $RobotTour$
```

IV. ACO ALGORITHM

Our ACO algorithm combines point-to-point planning using a bi-directional RRT with the solution procedure for the well-known Ant Colony System (ACS) TSP-solver. Like the lazy MST approach, this algorithm adopts the lazy approach of delaying collision-checking until a goal-to-goal pairing is built into a candidate tour solution. Once this occurs, a feasible path is found and the path's cost replaces the goal-to-goal Euclidean norm. In the description to follow, all parameter values for the ACO algorithm are set to the values recommended in [11] for use with ACS.

The algorithm is initialized with each of m ants starting at one of the n goal nodes. Every goal-to-goal pair is initialized with an amount of pheromone τ_0 set to the reciprocal of nC^{nn}, where C^{nn} is the cost of a representative tour constructed using the nearest neighbor heuristic. For consistency, we set C^{nn} to the cost of the lazy MST. Once the pheromones are initialized, the goal-to-goal costs are set to the values of the Euclidean norm between goals. The desirability of a goal-to-goal path, η_{ij}, is the reciprocal of the goal-to-goal cost.

During an iteration of the ACO algorithm, each ant, in turn, takes a step from its current node i to a neighboring node j. Each ant is forced to choose from a list of at most five nearest neighbors; as a tour is nearing completion fewer than five choices will be available. From among the goals in i's neighborhood N_i^k, ant k chooses a destination according to (1), where $q \in [0,1]$ is a uniformly-distributed random variable. If q is greater than the constant q_0, then the outcome J is chosen at random according to the distribution given by (2). If a feasible path has not yet been constructed from i to j, then a goal-to-goal path is found using the bidirectional RRT, and the cost (and hence the desirability η_{ij}) from i to j is reset to the length of this feasible path. Information about path lengths is public and not limited to ant k.

$$j = \begin{cases} argmax_{l \in N_i^k}\{[\tau_{il}]^\alpha [\eta_{il}]^\beta\} & \text{if } q \le q_0 \\ J & \text{otherwise} \end{cases} \tag{1}$$

$$p_{ij} = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta} \quad if \;\; j \in N_i^k \tag{2}$$

$$\tau_{ij}^{\text{new}} = (1-\xi)\tau_{ij}^{\text{old}} + \xi \tau_0 \tag{3}$$

$$\tau_{ij}^{\text{new}} = (1-\rho)\tau_{ij}^{\text{old}} + \rho \frac{1}{C^{bs}} \tag{4}$$

When ant k moves from goal i to j, it reduces the pheromone on the $i-j$ pairing according to (3). The update rule prevents the pheromone on any edge from dropping below the initial value τ_0. Once each of the m ants completes a tour, the best tour is selected and compared with the cost of the best-so-far tour, C^{bs}, replacing this value if it is lower than C^{bs}. Before the next iteration begins, pheromones are added to the the goal-to-goal pairings of the best-so-far tour according to (4). The parameters ρ and ξ, which appear in (3) and (4), respectively, are first-order filtering parameters

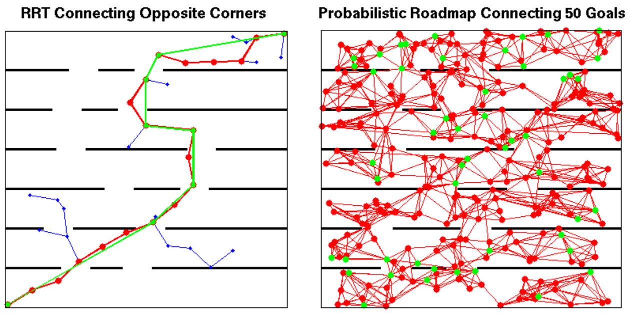
RRT Connecting Opposite Corners
Probabilistic Roadmap Connecting 50 Goals

Fig. 1. A representative example of the bi-directional RRT and the PRM used in the multi-goal planning algorithms of this study. At left, an RRT linking the two opposite corners of the workspace, with the original path marked in red, the post-optimized path marked in green, and unused tree branches plotted in blue. At right, a PRM used to link 25 randomly-selected goals, plotted in green, into a single connected component.

for tuning the pheromone update. This pheromone update procedure, the feature which makes ACS aggressive in comparison to other ACO solution methods, is biased toward the best-so-far tour. Although this will drive some ants to this tour, the removal of pheromones by ants also encourages exploration of edges unvisited by other ants. The algorithm is summarized in Algorithm 3.

This ACO algorithm provides us with a solution procedure based on the nearest neighbor heuristic, in which a tour is assembled by simply connecting goals iteratively to their nearest neighbors until a tour is completed. There are no upper bound guarantees on nearest neighbor tours, but ACO provides us with a framework to iteratively improve these solutions in cost. As is the case with the lazy MST, the algorithm will be repeated iteratively until the best-so-far tour cost, C^{bs}, fails to improve over three consecutive iterations. In total, we can describe the computational complexity of this method as $N*m*(O(n^2)+C*O(n))$, where m is the number of ants, $O(n^2)$ is the complexity of constructing a single nearest-neighbor tour, $O(n)$ is the worst-case number of RRT calls required to collision check a single nearest-neighbor tour, and N and C are the same as defined previously.

Algorithm 3 $RobotTour = AntColonyAlg(Goals, Obstacles)$

```
1: $AdjMat \leftarrow EuclideanDistances(Goals)$
2: $UnclearedEdges \leftarrow GetEdgePairs(Goals)$
3: $ClearedEdges \leftarrow \emptyset$
4: while $FailCount < 3$ do
5:   $AntStates \leftarrow InitializeAntsRandomly(Goals)$
6:   $FoundImprovedTour \leftarrow false$
7:   for $g \in Goals$ do
8:     for $k \in AntStates$ do
9:       $Goal_i \leftarrow AntStates(k)$
10:      $Neighbors \leftarrow FindFeasibleNeighbors(Goal_i)$
11:      $Goal_j \leftarrow ChooseNextGoal(Goal_i, Neighbors)$
12:      if $Edge_{ij} \in UnclearedEdges$ then
13:        $FeasiblePath_{ij} \leftarrow RRT(Edge_{ij}, Obstacles)$
14:        $ClearedEdges \leftarrow ClearedEdges \cup Edge_{ij}$
15:        $UnclearedEdges \leftarrow UnclearedEdges \setminus Edge_{ij}$
16:        $AdjMat(i,j) \leftarrow PathCost(FeasiblePath_{ij})$
17:      end if
18:      $ReducePheromones(Edge_{ij})$
19:      $AntTours(k) \leftarrow AntTours(k) \cup TourEdge_{ij}$
20:      $TourCosts(k) \leftarrow TourCosts(k) + AdjMat(i,j)$
21:      $AntStates(k) \leftarrow Goal_j$
22:    end for
23:  end for
24:  $BestAnt \leftarrow argmin(TourCosts(k))$
25:  $BestTourThisRound \leftarrow AntTours(BestAnt)$
26:  $BestCostThisRound \leftarrow TourCosts(BestAnt)$
27:  if $BestCostThisRound < BestCostSoFar$ then
28:    $BestCostSoFar \leftarrow BestCostThisRound$
29:    $BestTourSoFar \leftarrow BestTourThisRound$
30:    $FailCount \leftarrow 0$
31:  else
32:    $FailCount \leftarrow FailCount + 1$
33:  end if
34:  $AddPheromones(BestTourSoFar)$
35:  $Reset(AntTours, TourCosts)$
36: end while
37: $RobotTour \leftarrow BestTourSoFar$
38: return $RobotTour$
```

V. ALGORITHM COMPARISON

A. Simulated Test Case

A simple test environment was constructed to evaluate the relative performance of the three multi-goal planning algorithms. A point robot inhabits the workspace on $[0,100]$ in each dimension of $\Re^2$. This is intended to represent the area, in units of square meters, of a typical HAUV inspection mission underwater. Six evenly-spaced walls partition the free space, and two openings of width 10m are randomly placed in each wall, to allow a feasible solution of the multi-goal planning problem. A designated number of goal states is then sampled at random, and the robot's mission is to find a feasible tour among these goals. Examples of an RRT and PRM grown in this environment are presented in Figure 1.

For several problem instances, ranging from 5 to 200 goals in size, each of the three algorithms was run 100 times. At the start of each run the workspace is populated with new goals and obstacles, and all three algorithms share this randomly generated workspace. As mentioned above, both the lazy MST and the ACO algorithm terminate when three consecutive iterations pass without any improvement in the tour cost. For all instances with ten or more goals, the ACO algorithm ran using ten ants on the graph, and for smaller instances one ant was placed on each goal. A representative solution of each algorithm for an instance of 25 goals is pictured in Figure 2.

B. Results of Simulation

After running several hundred simulations on various problem sizes, advantages and drawbacks of each algorithm are evident. As the upper left plot of Figure 3 indicates, the lazy MST algorithm nearly always produced a solution

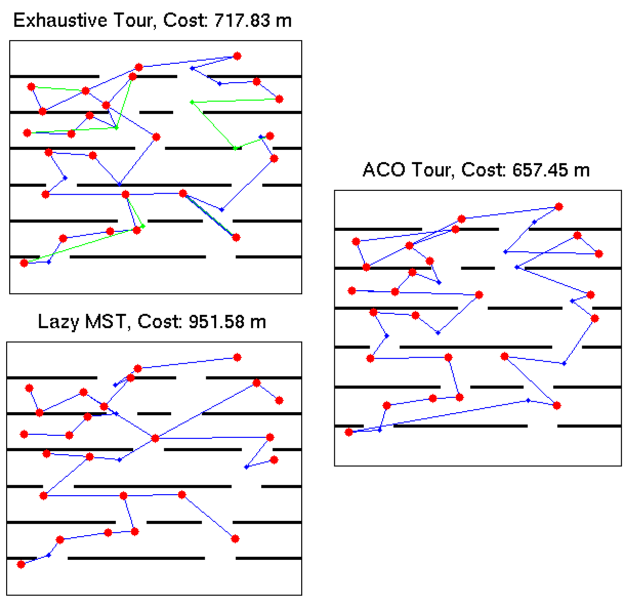

Fig. 2. A representative example of the multi-goal tours returned by the three algorithms examined in this study, for the same 25 randomly-selected goals. At top, the exhaustive tour formulated from all-pairs feasible paths. Components of the minimum-cost perfect matching are plotted in green, and components of the MST are plotted in blue. At bottom, the lazy MST tour after ten iterations, for which the cost is obtained by doubling that of the pictured MST. At right, the ACO tour after ten iterations with ten ants. The pictured workspaces are 100m by 100m in size.

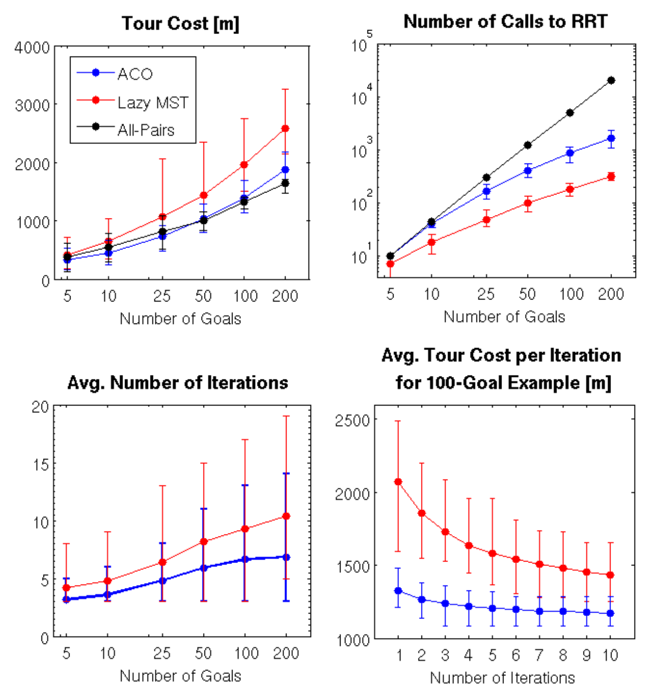

Fig. 3. Results of 100 instances of each algorithm on six different problem sizes, with randomly-generated goals. In all plots, the exhaustive all-pairs algorithm is plotted in black, the lazy MST algorithm is plotted in red, and the ACO algorithm is plotted in blue. The values plotted represent the mean over 100 trials, with the error bars representing minimum and maximum values over the 100 trials. In the upper right plot, the values for the exhaustive all-pairs algorithm are set to their theoretical worst-case values. In the bottom right plot, the dynamics of the lazy MST and ACO algorithms over ten iterations are plotted for the 100-goal problem only.

of inferior cost, while the exhaustive all-pairs algorithm and ACO produced solutions of nearly identical cost. ACO produced lower-cost tours on small problem instances, but began losing to the exhaustive all-pairs algorithm as the problem size approached several hundred goals. It is likely that the fixed number of ten ants, although recommended by [11], yields an increasingly local optimization as the number of goals grows exceedingly larger than ten.

In the upper right plot of Figure 3, the computational cost of the stochastic algorithm components, i.e., the number of calls to the RRT, is documented. As expressed earlier through our use of a collision-checking cost C, collision-checking becomes expensive in obstacle-rich environments and is likely to outweigh the computational cost of many deterministic algorithm components with comparable worst-case complexity. Our worst-case analysis presented earlier has correctly predicted the relative performance of the lazy MST algorithm and the ACO algorithm (ACO is expected to require more computation time due to the worst-case multiplicative factor m, the number of ants). The all-pairs algorithm was implemented using a PRM (hindering a meaningful direct comparison with lazy MST and ACO), and so the algorithm's theoretical worst case performance in terms of calls to an RRT has been plotted for comparison. It is useful in confirming that the computational cost of the other two algorithms is sub-quadratic.

The bottom left plot of Figure 3 indicates that the lazy MST and the ACO algorithm were typically repeating between 3 to 10 times before terminating, with the number of repeats increasing in the number of goals. The plot at bottom right offers additional visualization of the stage-to-stage dynamics of these algorithms, including the wider variance of the lazy MST solution compared with that of the ACO solution.

To deliver a clearer picture of which algorithm offers the highest performance, the performance parameters of Figure 3 were quantitatively combined to rank the algorithms in the design space of overall vehicle mission time. It was assumed that calls to the RRT are the best representative parameter of computation time, and that each call to the RRT requires one second to return a solution. Hence, the x-axis of Figure 4 plots each algorithm instance according to its mean number of calls to the RRT, costing one second per call.

Secondly, the mean tour length of each problem instance was used to compute an estimated vehicle travel time in seconds. Since the HAUV, at top cruising speed, can achieve about 0.25 m/s, all tour costs were divided by this speed to obtain an approximate travel time, which is displayed on the y-axis of Figure 4. Added together, approximate computation time and approximate travel time sum to total mission time, which is a parameter associated with physically meaningful length and velocity scales from the HAUV inspection application. This allows us to compare all problem instances from all three algorithms in terms of approximate total mission

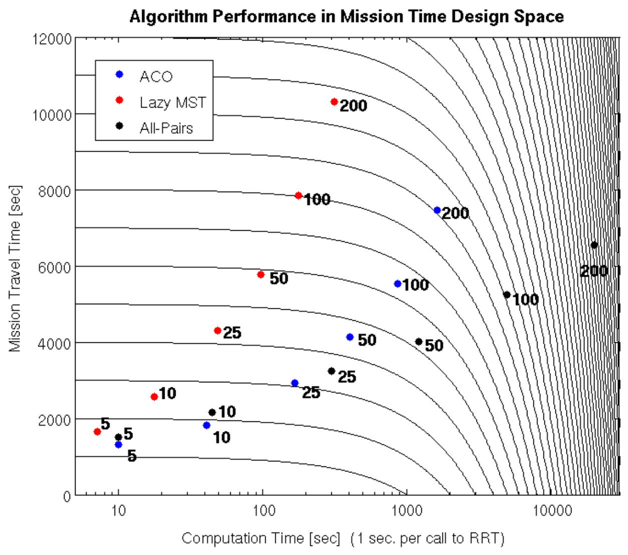

Fig. 4. A plot of algorithm performance in mission time design space, in which computation time is estimated using the mean number of RRT calls, assuming that each call requires one second of computation, and travel time is estimated using the mean tour cost and the maximum crusing speed of the Bluefin-MIT HAUV, which is 0.25 m/s.

time. In producing this plot, values for the exhaustive all-pairs algorithm were once again assumed to require RRT calls for all-pairs feasible paths.

For every one of the six problem size instances simulated in this study, the ACO multi-goal planning algorithm offered superior mission time. The parameters of mission time design space were not adjusted to favor the ACO algorithm, but were simply chosen as physically meaningful parameters from typical HAUV planning misssions. The ACO algorithm clearly fills a void between the expensive, but near-optimal exhaustive all-pairs algorithm, and the fast, but sub-optimal lazy MST algorithm.

VI. CONCLUSION

In evaluating the performance of a new multi-goal feasible planning algorithm that utilizes ant colony optimization, we have made quantitative comparisons with two existing sampling-based algorithms. The ACO algorithm, offering a compromise between solution quality and speed, was the superior choice given the physical parameters for HAUV mission planning.

It is evident that there exists a design space in which computational expense can be traded for a sacrifice in path quality, and the ACO algorithm, using only ”off-the-shelf“ settings obtained from [11], achieved a superior balance in this design space for the HAUV inspection application. It is likely that with greater effort, total mission time (as a function of computational cost plus time spent executing the path) can not only be reduced, but minimized, by tuning an algorithm such as ACO to suit a specific application. We hope that exploration of this design space may lead to such mission-time optimization in future applications.

REFERENCES

[1] K. Easton and J. Burdick, ”A Coverage Algorithm for Multi-robot Boundary Inspection,” *Proc. IEEE Int. Conf. on Robotics and Automation*, Barcelona, Spain, 2005, pp. 727-734.

[2] L. Xu and T. Stentz, ”A Fast Traversal Heuristic and Optimal Algorithm for Effective Environmental Coverage,” *Proc. Robotics: Science and Systems Conference*, Zaragoza, Spain, 2010.

[3] B. Englot and F. Hover, ”Inspection Planning for Sensor Coverage of 3D Marine Structures”, *Proc. IEEE Int. Conf. on Intelligents Robots and Systems*, Taipei, Taiwan, 2010.

[4] T. Danner and L. Kavraki, ”Randomized Planning for Short Inspection Paths,” *Proc. IEEE Int. Conf. on Robotics and Automation*, San Francisco, CA, 2000, pp. 971-976.

[5] P. Wang, R. Krishnamurti, and K. Gupta, ”View Planning Problem with Combined View and Traveling Cost,” *Proc. IEEE Int. Conf. on Robotics and Automation*, Rome, 2007, pp. 711-716.

[6] M. Saha, G. Sanchez-Ante, T. Roughgarden, and J.C. Latombe, ”Planning Tours of Robotic Arms Among Partitioned Goals,” *Int. J. Robotics Research*, vol. 25(3), 2006, pp. 207-223.

[7] C. Wurll, D. Henrich, and H. Worn, ”Multi-Goal Path Planning for Industrial Robots,” *Proc. Int. Conf. on Robotics and Applications*, Santa Barbara, CA, 1999.

[8] S. Spitz and A. Requicha, ”Multiple-Goals Path Planning for Coordinate Measuring Machines,” *Proc. IEEE Int. Conf. on Robotics and Automation*, San Francisco, CA, 2000, pp. 2322-2327.

[9] K. Savla, F. Bullo, and E. Frazzoli, ”On Traveling Salesperson Problems for a Double Integrator,” *Proc. IEEE Conf. on Decision and Control*, San Diego, CA, pp. 5305-5310.

[10] K. Savla, F. Bullo, and E. Frazzoli, ”Traveling Salesperson Problems for the Dubins Vehicle,” *IEEE. Trans. Automatic Control*, vol. 53(6), 2008, pp. 1378-1391.

[11] M. Dorigo and T. Stutzle, *Ant Colony Optimization*, MIT Press, Cambridge, MA, 2004.

[12] M. Dorigo and L. Gambardella, ”Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem,” *IEEE. Trans. Evolutionary Computation*, vol. 1(1), 1997, pp. 53-66.

[13] S. Koenig and Y. Liu, ”Terrain Coverage with Ant Robots: A Simulation Study,” *Proc. Fifth Int. Conf. on Autonomous Agents*, Montreal, Canada, 2001, pp. 600-607.

[14] A. Agarwal, M. Lim, M. Er, C. Chew, ”ACO for a New TSP in Region Coverage,” *Proc. IEEE Int. Conf. on Intelligent Robots and Systems*, Edmonton, Canada, 2005, pp. 1717-1722.

[15] M. Mohamad, N. Taylor, and M. Dunnigan, ”Articulated Robot Motion Planning Using Ant Colony Optimisation,” *Proc. IEEE Int. Conf. on Intelligent Systems*, London, 2006, pp. 690-695.

[16] S. Liu, L. Mao, and J. Yu, ”Path Planning Based on Ant Colony Algorithm and Distributed Local Navigation for Multi-Robot Systems,” *Proc. IEEE Int. Conf. on Mechatronics and Automation*, Luoyang, China, 2006, pp. 1733-1738.

[17] M. Garcia, O. Montiel, O. Castillo , R. Sepulveda, and P. Melin, ”Path Planning for Autonomous Mobile Robot Navigation with Ant Colony Optimization and Fuzzy Cost Function Evalutation,” *Applied Soft Computing*, vol. 9, 2009, pp. 1102-1110.

[18] T. Krenzke, *Ant Colony Optimization for Agile Motion Planning*, Masters Thesis, Massachusetts Institute of Technology, 2006.

[19] A. Kulatunga, D. Liu, G. Dissanayake, and S. Siyambalapitiya, ”Ant Colony Optimization Based Simultaneous Task Allocation and Path Planning of Autonomous Vehicles,” *Proc. IEEE Conf. on Cybernetics and Intelligent Systems*, Bangkok, 2006, pp. 1-6.

[20] F. Hover, et al., ”A Vehicle System for Autonomous Relative Survey of In-Water Ships,” *Marine Technology Society Journal*, vol. 41(2), 2007, pp. 44-55.

[21] S. Lavalle and J. Kuffner, ”Rapidly-Exploring Random Trees: Progress and Prospects,” *Proc. Workshop on the Algorithmic Foundations of Robotics*, 2000, pp. 293-308.

[22] L. Kavraki, P. Svestka, J.C. Latombe, and M. Overmars, ”Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces,” *IEEE Trans. on Robotics and Automation*, vol. 12(4), 1996, pp. 566-580.

[23] N. Christofides, ”Worst-case Analysis of a New Heuristic for the Traveling Salesman Problem,” Technical Report CS-93-13, Carnegie Mellon University, 1976.