Autonomous Surface Vehicle Energy-Efficient and Reward-Based Path Planning using Particle Swarm Optimization and Visibility Graphs

Evan Krell*, Scott A. King

Control of Robots and Autonomous Agents Laboratory (CORAL)
Texas A&M University — Corpus Christi
6300 Ocean Dr, Corpus Christi, Texas

Innovation in Computing Research Labs (ICORE)
Texas A&M University — Corpus Christi
6300 Ocean Dr, Corpus Christi, Texas

Luis Rodolfo Garcia Carrillo

Klipsch School of Electrical and Computer Engineering
New Mexico State University
1780 E University Ave, Las Cruces, New Mexico

Abstract

Autonomous Surface Vehicles require path planning that considers complex shoreline obstacles and dynamic forces such as water currents. Here, path planning is used to achieve an energy-efficient route based on water current forecasts. Graph-based algorithms such as Dijkstra's Shortest-Path First generate an optimal solution, but scale exponentially with search space size. Metaheuristic algorithms such as Particle Swarm Optimization (PSO) sacrifice guaranteed optimality to substantially reduce computation. We compare PSO to Dijkstra and A* for energy-efficient planning in a high resolution coastal environment with dynamic water current forecasts. Observing high solution variance, we use Visibility Graphs (VGs) to generate a set of initial candidate solutions for PSO. We demonstrate that starting with feasible paths that include that with shortest-distance allows PSO to reliably converge to near-optimal paths much faster than Dijkstra. We also consider a path's data collection reward. The goal is to allow the vehicle to take advantage of nearby observation opportunities. Again, VGs are shown to aid convergence to fitter solutions.

Keywords: Path planning, ASV navigation, Autonomous, Particle swarm optimization, Visibility graph

1. Introduction

A primary Autonomous Surface Vehicle (ASV) task is to navigate in the marine environment toward a given location. Substantial research effort has been devoted to the NP-hard path planning problem. The

*Corresponding author
Email addresses: evan.krell@tamucc.edu (Evan Krell), scott.king@tamucc.edu (Scott A. King), luisillo@nmsu.edu (Luis Rodolfo Garcia Carrillo)

conventional criteria is distance minimization, but environmental forces such as wind and water currents increase the complexity. It may also be reasonable to sacrifice some efficiency to take advantage of observation opportunities. The vehicle might be traversing over nothing but sand on a shorter path, while interesting habitat is nearby. Taking advantage of opportunistic sampling reward increases the solution space complexity. The goal is to develop a planner to optimize path efficiency and reward (Figure 1).

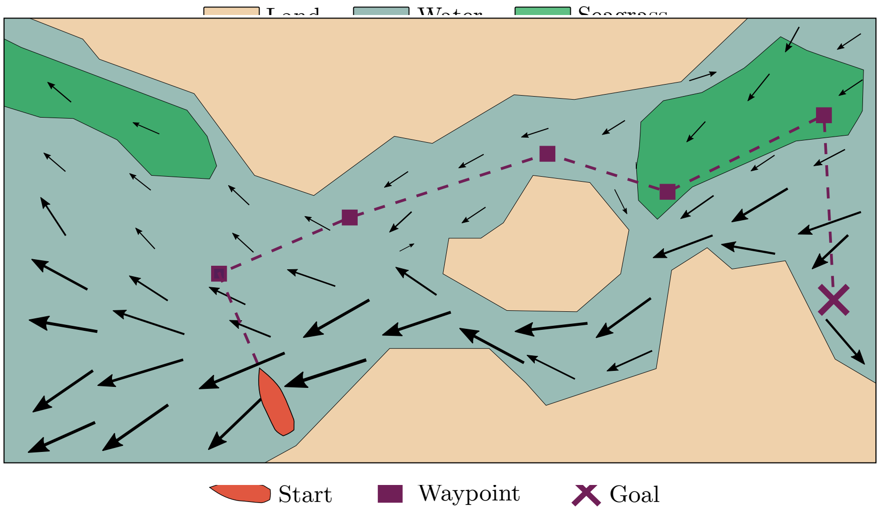


Figure 1: Planning for energy efficiency and sampling reward yields a sequence of waypoints forming a set of straight-line segments between start and goal. The shortest path would have led the ASV below the island. Instead, the path avoids strong currents and deviates slightly to collect seagrass observations.

This research expands on our work presented at the American Control Conference 2020 (Krell et al., 2020b). The PSO fitness function has been improved and we use Visibility Graphs (VGs) to seed the PSO initial population for improved performance. New experiments compare our algorithm with graph-based algorithms Dijkstra and A*. A companion to this paper is a software repository called `conch` (Krell, 2021a) that contains the planning software, experiment results, and scripts for replication. Other work by the authors in this area include shortest distance path planning for indoor mobile robots using PSO (Krell et al., 2020a), and a game-theoretic ASV planner for the worst-case water currents based on forecast uncertainty (Krell et al., 2019).

Classic graph algorithms such as Dijkstra's Shortest Path First algorithm can find an optimal path (Dijkstra, 1959). However, the complexity of the algorithm is $O(V * \log V)$ where V is the number of vertices, assuming that the graph is densely connected as typical for planning tasks. Planning over large, high-resolution space demands faster methods.

The A* algorithm employs a heuristic to constrain the search space for improved efficiency (Nosrati et al., 2012). An admissible heuristic is one that never overestimates the cost to reach the goal; this guarantees that A* will yield an optimal solution (Nosrati et al., 2012). With additional criteria beyond distance, the cost function is not guaranteed to be admissible. In sufficiently complex currents, the optimal path may deviate significantly from the shortest. Studies have shown that A* with inconsistent heuristics may or may not offer improved performance over Dijkstra (Zhang et al., 2009).

An alternative is planning with metaheuristic algorithms. These include evolutionary-based approaches such as Genetic Algorithm (GA) (Forrest, 1993) and Differential Evolution (DE) (Storn and Price, 1997), as well as colony-based algorithms such as PSO (Eberhart and Kennedy, 1995) and Artificial Bee Colony (ABC) (Karaboga, 2010). Nature-inspired behaviors speed up the search, but sacrifice the guarantee of finding the global optimum. Complicated search spaces may have numerous local optima. The goal is to reliably converge to near-optimal solutions while meeting time constraints. Metaheuristic algorithms have been applied to multi-objective path planning applications (Jones et al., 2002) including energy efficient marine path planning. Research in this area is more commonly focused on Autonomous Underwater Vehicle (AUV) planning, but very often only considering navigation on a 2D plane in the water column much like boats dealing with surface currents.

Yang and Zhang (2009) presented Adapted Inertia-weight PSO for AUV planning. The inertia weight is a PSO hyperparameter that directs the balance of exploration and exploitation (Bansal et al., 2011). Adaptive strategies have become common, where the weight is modified over the course of the search. An experiment was done using an electronic chart with complex island shapes and spatially-varying water currents.

Liu et al. (2011) presented a PSO-based AUV planner. Two modifications were made to avoid local minima. Inertia weight is again adaptive, but also the cognitive and social coefficients. At first, the cognitive (local information) is more important, but the weights are shifted so that over time the social information dominates. Experiments were performed based on a chart with polygonal islands. Water currents were based on the region's monthly average currents.

A system for AUV online replanning was developed by Zeng et al. (2015). The planner uses spatio-temporal water currents and uncertain obstacle locations. Objects are sensed rather than known beforehand. The planner is intended to ensure safe navigation despite imperfect sensor readings. Rather than reacting to the dynamic ocean environment by planning from scratch, replanning takes advantage of previous planning to speed up the computation. The search algorithm is Quantum-behaved PSO (QPSO) where the movement of the particles is inspired by quantum mechanics rather than conventional Newtonian physics. Experiments were performed on a 100x100 grid with dynamic circular obstacles and irregular static island shapes.

Another PSO implementation with dynamic hyperparameters was presented by Xu et al. (2018). Their improvements included varying the particle acceleration over time to again shift emphasis from global to local search as well as using a slowly-varying function to expand the search and maintain diversity in new candidates. Experiments occurred on a 2D plane with elliptical obstacles. No water currents are used, but the work is of interest because of the large 10,000x10,000 grid.

Lim et al. (2019) combined DE and PSO for both quantum-behaved and adaptive PSO. Not only energy-efficient, their system incorporates smooth, feasible control for a REMUS-100 AUV. Adaptive and quantum versions have several variants based on combinations of hard and soft constraints. Hard constraints guarantee solution properties, such as obstacle-free paths, but at higher computational cost. Soft constraints use a penalty function to (ideally) converge to meet the constraints. Both 2D and 3D environments with spatio-temporal currents and elliptical or spherical obstacles were used. Results suggest that the best choice is quantum with hard vehicle motion constraints and soft obstacle constraints.

Kuhlemann and Tierney (2020) modified GA for ASV planning. Wind, waves and piracy risk were incorporated for smooth paths minimizing fuel and risk while meeting arrival time constraints. Custom mutation and crossover operators were developed based on the structure of paths. Rather than vector fields, wind and waves are represented by their Beaufort number (Barua, 2005). Typical metaheuristic planners solve for an arbitrary number of waypoints, but here GA solutions are variable length. Experiments were on a world map with spatio-temporal wind and waves, perturbing the known weather to generate imperfect forecasts to test the algorithm under uncertainty. The initial GA population was generated from feasible solutions instead of the usual random population.

Usually, metaheuristic algorithms were compared to other metaheuristics instead of an optimal solution. For example by comparing an improved PSO to a canonical PSO. Dijkstra is a reasonable benchmark since it is guaranteed to be optimal. However, the optimality is with respect to the input graph. In a fully-connected graph representation of a search space, adjacent nodes are typically connected to 4, 8, or 16 neighbors so that the solution path cannot include arbitrary heading angles. In practice, 4-way yields jagged, sub-optimal paths. Increasing the neighborhood increases fitness at the cost of a much larger search space. Still, the Dijkstra solution is a useful benchmark since the optimal metaheuristic result should be at least as fit given a sufficient solution dimension. Fixed-length metaheuristic solutions limit the number of possible waypoints, restricting the possible path complexity and fitness.

By default, most metaheuristic solvers are initialized with a random set of candidate solutions. Given a fitness function, the solutions are modified over each iteration. VGs are graphs that include only the edges that could make up the shortest-distance path (Shah and Gupta, 2019). Here, we use VGs to generate an initial population that includes the shortest-distance and many other feasible paths. Using VGs significantly speeds up finding the shortest path at the cost of a set-up time to build the graph; it can be reused for as long as the map remains static.

Experiments are performed with realistic scenarios. A raster of Boston Harbor with complex islands is used for static obstacles. Four water current forecasts from the Northeast Coastal Ocean Forecast System (NECOFS) (Beardsley, 2014) are used to get a spatio-temporal vector field for energy-efficient planning.

Research in marine metaheuristic path planning often have limited reproducability. Hyperparameters, map resolution, water currents, and tuning steps are often only partially reported. It is difficult to compare new research to the literature. Here, the software and results have been made available.

1.1. Contributions

- A VG is used to initialize the PSO candidate solutions to avoid local optimum. The VG is generated from the Boston Harbor raster, and A* is used to generate a number of paths that include that with shortest distance.
- Opportunistic sampling is explored using a synthetic reward raster. Experiments investigate how tuning the weight of reward interest impacts the planning behavior.
- Dijkstra is used for energy-efficient benchmarks. Results include Dijkstra with 4-way, 8-way, and 16-way neighborhoods using both graph-based and raster-based implementations. A* is also briefly evaluated to check performance given the non-admissible heuristic.

2. Materials and Methods

This research compares PSO to Dijkstra for energy efficient planning, with and without using VGs to initialize the PSO population. Additional experiments evaluate the use of PSO for opportunistic sampling. The graph search algorithms are discussed in section 2.1, followed by a description of VG in section 2.2. PSO is described in section 2.3. Finally, section 2.4 gives the fitness function for energy and reward.

2.1. Dijkstra's Shortest-Path First and A algorithms*

Dijkstra generates benchmark solutions, with the input graph limitations previously discussed. A* is also used since, for shortest-distance, it is guaranteed to match Dijkstra, but using a distance heuristic to reduce the search space. However, it remains to be seen if A* is effective when incorporating water currents since there is not an admissible heuristic. Here, the cost function is calculated based on energy expenditure, but the heuristic for node priority is the distance. It is assumed that the region has been projected into an appropriate 2D coordinate system. The distance measurement error depends on the accuracy of the projection and is assumed here to be sufficiently accurate for operational planning. In the following, Distance(p_1, p_2) represents the distance between points p_1 and p_2. For example, the Euclidean distance for a metric coordinate system and Haversine for latitude and longitudes.

Dijkstra and A* are implemented together, using a conditional statement to choose how the next node is selected. A* is enabled by using the distance heuristic. Otherwise, the behavior is Dijkstra's. Two versions are implemented: one solving directly on the occupancy grid and the other on a graph representation. Using the grid, cell neighborhoods are calculated at run time by checking if the adjacent nodes are obstacles. For the graph-based solver, the input raster has already been converted to a graph where each edge is to a valid neighbor. By checking at run time, only the cells actually encountered by the algorithm are evaluated. Using the graph-solver requires precomputing the entire graph, but this graph can be reused. Another advantage of the graph-based solver is that it is not limited to a uniformly connected graph. For example, Dijkstra and A* may be applied to a VG for a potential savings in solution cost and computation.

Algorithm 1 **GraphSolver** uses an energy-based cost function, extending a conventional shortest-distance implementation by Patel (2014). The pseudocode highlights computational details needed for the water currents. The main inputs are a graph $\mathcal{G}$ and start and goal locations, s and g, in (row, col) coordinates. Here, the rows and columns correspond to cells in the raster even though this algorithm is applied to a graph. The graph is implemented as a python dictionary where a (row, col) tuple is a key to query neighbor cells. The reason for preserving these locations is that the water current rasters are in the same dimensions as the environment raster and are used to access the water velocities on all cells between two nodes.

Work is calculated with fitness function **CalcWork**, discussed in Section 2.4. It requires additional inputs including the water current rasters (collectively referred to as $\mathcal{C}$), the boat's target speed, and geospatial metadata t. Also, a time offset from the start time of the water currents forecast is used. **GridSolver** only differs by runtime checking for neighbor nodes instead of querying precomputed values.

2.2. Visibility Graph

A VG's nodes and edges are only those potential segments of the shortest path (Niu et al., 2018). It is generated from a set of polygons where each vertex is added as a node in the VG. The edges are added by checking the visibility (obstacle-free, straight line), between all node pairs. To compute a path from a start to goal location, the start and goal are also added to the VG along with edges for all visible vertices from those points.

The `pyvisgraph` library is used for making VGs (Reksten-Monsen, 2018). Lee's Visibility Graph Algorithm is used which has a complexity of $O(n^2 \log_2 n)$ (Coleman, 2012). It is more complex than that of Ghosh and Mount ($O(e + n \log_2 n)$ (Kitzinger and Moret, 2003), but supports adding nodes after initial creation. Here, the user provides a raster occupancy grid, and obstacles are converted into the polygons. The VG is used both as a graph for Dijkstra and A* and for generating the PSO initial population.

To generate the initial PSO population, A* extracts paths from the VG. A* only finds the shortest path, so the VG is modified between each A* run. The middle waypoint of each path is removed from the VG. This prevents finding this same path again as well as others with that waypoint to increase path diversity. Since this scheme generates several shortest paths using A*, it is only useful if finding an energy efficient path with Dijkstra or A* is significantly more time consuming than finding a shortest path on a VG.

2.3. Particle Swarm Optimization

PSO is a metaheuristic algorithm inspired by biological colony behavior (Eberhart and Kennedy, 1995). A group of particles share information to find an optimal solution. Each particle has position and velocity in the

Algorithm 1: GraphSolver

```
Input: graph $\mathcal{G}$, start coords $s$, goal coords $g$, water currents vector field $\mathcal{C}$, elapsed time $e$,
       geographic transform $t$, number of raster rows $r$, target vehicle speed $v$, algorithm choice $a$
Output: $cameFrom$, $costSoFar$, $timeSoFar$
1  $\mathcal{F} \leftarrow$ PriorityQueue()
2  $\mathcal{F}$.put($s$, 0) // start graph search from $s$
3  $cameFrom$, $costSoFar$, $timeSoFar = \emptyset$
4  $cameFrom[s] =$ NULL // $s$ is the start of the path
5  $costSoFar[s] = 0$ // no cost to reach $s$
6  $timeSoFar[s] = e$ // elapsed time for correctly accessing temporal water currents
7  while $\mathcal{F} \neq \emptyset$ do
8      $c \leftarrow \mathcal{F}$.get() // get current location to search
9      if $c = g$ then
10         BREAK // reached goal
11     $\mathcal{E} \leftarrow \mathcal{G}[c]$ // get all nodes connected to current
12     for $n \in \mathcal{E}$ do
13         $d \leftarrow$ Distance($c, n$) // distance between current and next node
14         $\delta w$, $\delta e \leftarrow$ CalcWork($c_g, n_g, \mathcal{C}, v, t, timeSoFar[c]$) // cost, time between current and
               next
15         $cost \leftarrow costSoFar[c] + \delta w$ // cost to reach next from start
16         if $n_g \notin costSoFar$ & $cost < costSoFar[n]$ then
17             $costSoFar[n] = cost$ // update cost to reach next with minimal found so far
18             if $a =$ "dijkstra" then
19                 $p \leftarrow cost$ // use cost as priority
20                 $\mathcal{F}$.put($n$, $p$)
21             if $a =$ "A*" then
22                 $p \leftarrow cost +$ Distance($s$, $n$) // add distance heuristic to priority
23             $\mathcal{F}$.put($n$, $p$) // add to priority queue
24             $cameFrom[n] = c$ // link current and next node to generate path
25             $timeSoFar[n] = e + \delta e$ // time to reach next node
26     return $cameFrom$, $costSoFar$, $timeSoFar$
```

search space. Each iteration, these are updated based on local and neighborhood information discovered so far. Balancing exploration and exploitation is critical to success, and is influenced by the cognitive coefficient c_1 and social coefficient c_2 that weight the attraction to the local and neighborhood best, respectively. The speed at which it can change its trajectory is controlled by a constriction coefficient w where a higher value causes slower change of direction. The constriction coefficient may be constant or adapted over the PSO iterations. The adaptive version is intended to gradually shift the particle's attention from exploration to exploitation which promotes convergence. Similar to the constriction coefficient is the inertia weight, playing the same role in the velocity update equations but requiring a maximum velocity w_{max} to be specified. Various neighborhood topologies exist, including global and adaptive random strategies. Innocente and Sienz (2010) provide parameter value suggestions.

The fitness function calculates the relative energy needed across a collision-free sequence of waypoints at constant speed in the presence of water currents. It is possible to write a complex fitness function that modifies solutions to be collision-free before calculating the score. The more common alternative is a soft constraint — an arbitrarily high penalty cost. Hard constraints ensures feasibility, but may require conditional statements and loops that make it difficult to use vectorized operations. Lim et al. (2019) found soft obstacle avoidance constraints faster and more effective.

Typically the initial population is from a random or uniform distribution. Alternatively, pre-processing can produce realistic or at least feasible solutions. Kuhlemann and Tierney (2020) generated collision-free paths for their initial GA population. PSO is implemented with `pagmo`, a C++ library with implementations of biologically-inspired search algorithms. The library is focused on efficient implmentations and providing a consistent API to apply numerous search algorithms with a single fitness function. The path planning software developed here is written in python, taking advantage of the bindings to `PaGMO` provided by the `PyGMO` python bindings. `PaGMO`'s PSO includes 6 variants on the velocity update and 4 variants on the neighborhood topology. Here, the defaults of both are used, based on initial trials that did not demonstrate significant difference for this problem. The velocity update rule is controlled by the constriction coefficient. The neighborhood topology is adaptive random: when the global optimum fails to improve, each particles randomly selects k others as neighbors irrespective of their distance (Zambrano-Bigiarini et al., 2013).

2.4. Fitness function

Planning takes place in an environment represented with an $M \times N$ binary occupancy grid G_e. Water currents are vector fields represented by $M \times N \times T$ rasters where T corresponds to discrete time intervals. These are the magnitude and direction of the water velocity called G_m and G_d respectively. The reward assigned to each cell is stored in an $M \times N$ raster G_r.

A candidate solution path is represented as a sequence of n waypoints where each element is a tuple with an x and y coordinate into the environment grids as

$$path = [(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)] \tag{1}$$

The fitness function is designed to minimize collisions, distance, and energy expenditure (work). However, some efficiency is sacrificed to maximize path reward. Weights are used to influence the importance of each criteria. High weight on reward should cause the vehicle to be less concerned with path efficiency, while a smaller reward weight emphasizes efficiency. The fitness function's result, F_{path}, is the weighted sum of the four optimization attributes.

Path distance P_D is the sum of distances between each pair of adjacent waypoints, (x_i, y_i) and (x_{i+1}, y_{i+1}), as

$$\begin{gathered} P_D = \sum_{i=1}^{n-1} \text{Distance}(x_i, y_i, x_{i+1}, y_{i+1}) \\ \forall (x_i, y_i) \in (x_1, y_1), (x_2, y_2), ..., (x_{n-1}, y_{n-1}) \end{gathered} \tag{2}$$

The attribute P_O is the number of occupied cells along a straight line between adjacent path waypoints. The Bresenham algorithm (Bresenham, 1965) is used to select all cells along a line in G_e. Free and occupied cells in G_e have values of 0 and 1, respectively. Thus, a summation of G_e values in the calles returned by Bresenham is the obstacle count.

$$\begin{gathered} P_O = \sum_{i=1}^{n-1} \text{o}(x_i, y_i, x_{i+1}, y_{i+1}) \\ \forall (x_i, y_i) \in (x_1, y_1), (x_2, y_2), ..., (x_{n-1}, y_{n-1}) \end{gathered} \tag{3}$$

where

$$\begin{gathered} \text{o}(x_i, y_i, x_j, y_j) = \sum G_e(x_l, y_l) \\ \forall x_l, y_l \in \text{Bresenham}(x_i, y_i, x_j, y_j) \end{gathered} \tag{4}$$

The path reward P_R is the sum of reward values in G_r between each pair of adjacent waypoints. The function $\text{r}(x_i, y_i, x_j, y_j)$ sums the reward of a single segment, again using Bresenham to check all intersected cell in G_r.

$$\begin{gathered} P_R = \sum_{i=1}^{n-1} \text{r}(x_i, y_i, x_{i+1}, y_{i+1}) \\ \forall (x_i, y_i) \in (x_1, y_1), (x_2, y_2), ..., (x_{n-1}, y_{n-1}) \end{gathered} \tag{5}$$

where

$$\begin{gathered} \text{r}(x_i, y_i, x_j, y_j) = \sum G_r[(x_l, y_l)] \\ \forall x_l, y_l \in \text{Bresenham}(x_i, y_i, x_j, y_j) \end{gathered} \tag{6}$$

The work exerted by the vehicle P_W is the sum of work required along the path segments as it deals with water currents to maintain a constant speed s_{boat}. Calculating the work is based on the robot's applied force vector as it adjusts in response to the influence of water currents. This component is dependent on the elapsed time of all previous planning. The elapsed duration d and time resolution l of each band is used to select the bands in G_m and G_d. The water velocity is calculated by interpolating between the two nearest bands. For example, if the forecast begins at 00:00, l is 1 hour and d is 25 minutes, then the weighted interpolation combines 75% of the nearer, first band and 25% of the further, second.

The cell size d^c is used to estimate the distance that the vehicle's resultant force is applied. Bresenham is used by the function $\mathrm{w}(x_i, y_i, x_j, y_j)$ to sum the work required along a single path segment. Within, the function $\mathrm{w^c}(x_l, y_l, x_j, y_j)$ calculates the work required to traverse a single cell.

$$\begin{gathered} P_W = \sum_{i=1}^{n-1} \mathrm{w}(x_i, y_i, x_{i+1}, y_{i+1}) \\ \forall (x_i, y_i) \in (x_1, y_1), (x_2, y_2), ..., (x_{n-1}, y_{n-1}) \end{gathered} \tag{7}$$

where

$$\begin{gathered} \mathrm{w}(x_i, y_i, x_j, y_j) = \sum \mathrm{w^c}(x_l, y_l, x_j, y_j) \\ \forall x_l, y_l \in \text{Bresenham}(x_i, y_i, x_j, y_j) \end{gathered} \tag{8}$$

The work $\mathrm{w^c}$ at an individual cell requires the nearest discrete water currents forecast. The elapsed duration d is used to select the appropriate water currents raster band at index i. The elapsed duration d is divided by the duration of each discrete forecast interval l. Rounding yields the nearest band, and max avoids the zero band, since the indices start at one.

$$i = \max\Big(1, \text{round}\Big(d/l\Big)\Big) \tag{9}$$

Next, the boat's heading angle θ_{boat} is calculated based on the boat's current position (x_l, y_l) and next target waypoint (x_j, y_j) as

$$\theta_{boat} = \text{atan2}(y_j - y_l, x_j - x_l) \tag{10}$$

The boat's target velocity V^{boat} can now be determined using the boat's heading θ_{boat} and target speed s_{boat} as.

$$(V_x^{boat}, V_y^{boat}) = (s_{boat} \cos\theta_{boat}, s_{boat} \sin\theta_{boat}) \tag{11}$$

The water current velocity is determined by looking up the forecasts at the selected band b in environment grids G_m and G_d at (x_l, y_l) as

$$\begin{aligned} V_x^{water} &= G_m(i, x_l, y_l) \quad \cos G_d(i, x_l, y_l) \quad (1 - ((d \bmod l)/l)) \\ &+ G_m(i, x_l, y_l) \quad \cos G_d(i, x_l, y_l) \quad ((d \bmod l)/l) \\ V_y^{water} &= G_m(i, x_l, y_l) \quad \sin G_d(i, x_l, y_l) \quad (1 - ((d \bmod l)/l)) \\ &+ G_m(i, x_l, y_l) \quad \sin G_d(i, x_l, y_l) \quad ((d \bmod l)/l) \end{aligned} \tag{12}$$

The boat's required applied velocity $V^{applied}$ to maintain the target V^{boat} is calculated using V^{water} as

$$V^{applied} = V^{boat} - V^{water} \tag{13}$$

Finally, $\mathrm{w^c}$ is calculated based on applying the force over the distance d^c to traverse a cell as

$$\mathrm{w^c} = |V^{applied}|d^c \tag{14}$$

The fitness for the entire path is the weighted sum of these values, using weights W_D, W_O, W_W, and W_R as

$$F_{path} = W_O P_O + W_D P_D + W_W P_W - W_R P_R \tag{15}$$

The weight W_O is an arbitrary huge number.

3. Results and Discussion

Experiments are done with only energy optimization and with both energy and reward. Section 3.1 describes the experimental environment and data structure representations. Section 3.2 gives the results of VG generation. Results of energy-only with graph-based search and PSO is discussed in Section 3.3, and with energy and reward in Section 3.4.

3.1. Environment

Experiments used Boston Harbor (Figure 2a) because of its complex coastline, numerous islands, and available water current forecasts. The raster is a binary occupancy grid of land and water, produced from NGAI's World Vector Shoreline (1:250,000) data ([dataset] NOAA, 2017) in QGIS (QGIS Development Team, 2018).

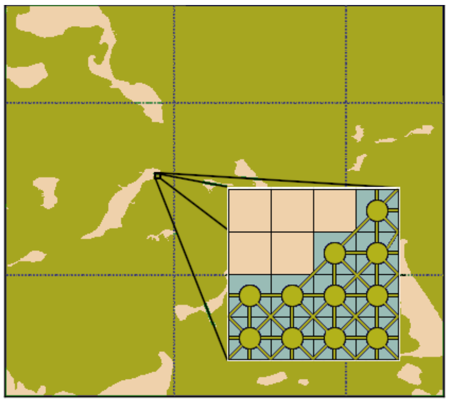
Water current forecasts are from The Northeast Coastal Ocean Forecast System (NECOFS) ([dataset] Northeast Regional Coastal Ocean Observation System, 2021), provided by the Northeast Regional Coastal Ocean Observation System (NERCOOS) Program, Massachusetts Fishery Institution, and the MIT sea grant. Nowcasts and forecasts are available for the region. The forecast horizon is 72 hours at 1-hour intervals. A collection of python scripts, `Whelk`(Krell, 2021b), was developed by the authors to access, store, and visualize the data. Given a start and stop time, and desired attributes, the unstructured grid is converted to a set of multi-band rasters. Each has a single attribute, such as the eastward water velocity components where bands are the values at a discrete time steps.

Four scenarios, $W = \{\mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_3, \mathcal{W}_4\}$, were selected since they exhibit various spatio-temporal behavior (see Figure 3). Each is represented as two rasters (eastward, northward components) with three bands each for three hours. $\mathcal{W}_0$ indicates that the water currents are ignored.

Experiments are performed for three tasks (start, goal pairs) $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_3, \mathcal{T}_3\}$ (Table 1). Experiments assume a constant speed of $0.5mps$. The purpose of $\mathcal{T}_1$ and $\mathcal{T}_2$ is to evaluate PSO in terms of solution cost

IMG/mass_bay.pdf

(a) Boston Harbor.



(b) 8-way uniform graph.

Figure 2: Boston Harbor as an occupancy grid raster of water (free) and land (obstacle). It is used to generate three uniformly connected graphs that differ in the number of neighboring nodes. More nodes allows give greater movement freedom for more efficient solutions, but at the cost of an increased search space. Experiments are performed with 4, 8, and 16-way graphs.

and variance. Multiple paths exist that navigate around the obstacles, and a better solution reliably chooses that with lower cost. Task $\mathcal{T}_3$ is to check the effectiveness of the soft obstacle constraint. PSO should plan a route around the peninsula, despite a local minima passing through the few obstacle cells in the narrow land mass.

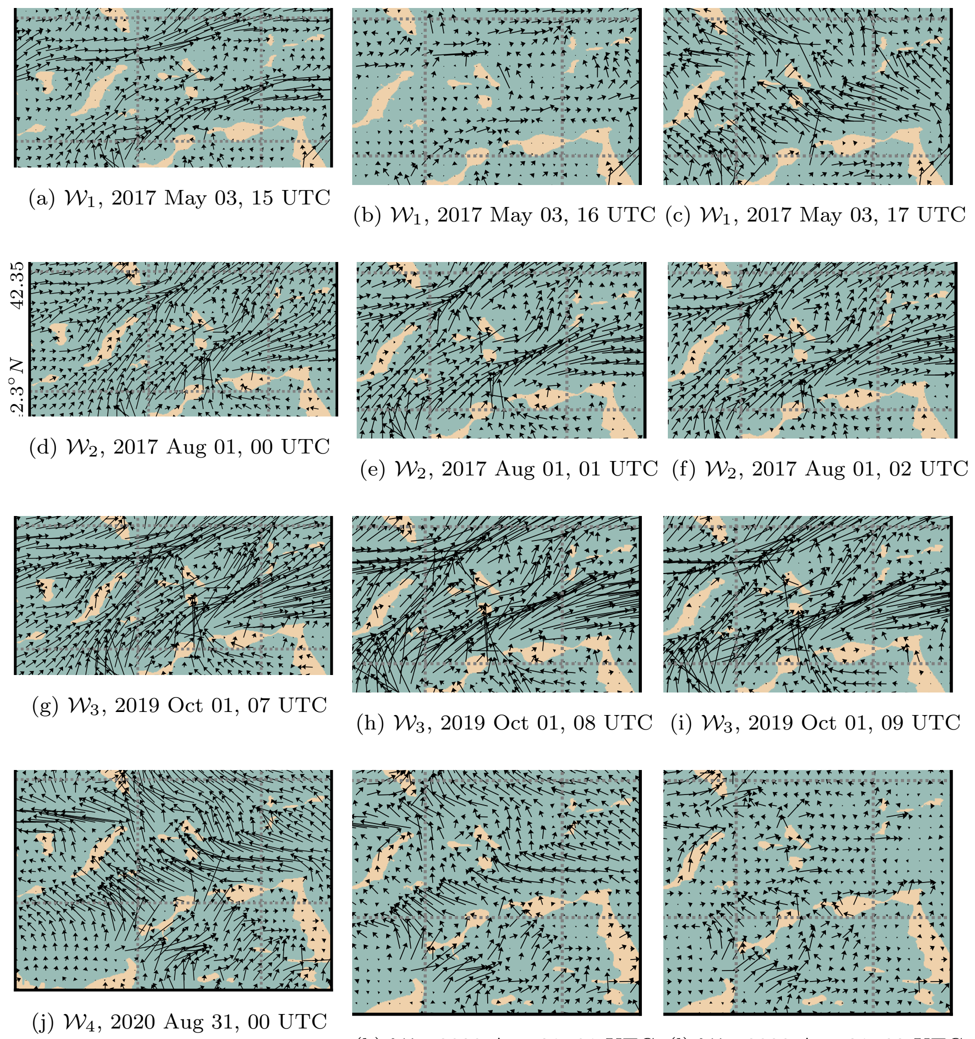
Table 1: Path planning tasks.

Task	**Start**	**Goal**
$\mathcal{T}_1$	42°19’24.4”N 70°59’39.4”W	42°20’09.6”N 70°53’14.5”W
$\mathcal{T}_2$	42°19’58.2”N 70°58’23.6”W	42°16’18.6”N 70°54’12.3”W
$\mathcal{T}_3$	42°21’44.0”N 70°57’22.2”W	42°21’10.2”N 70°58’46.3”W

3.2. Visibility graphs

A VG was generated using the `pyvisgraph` library (Reksten-Monsen, 2018). The first output was invalid with the edges intersecting the obstacles. Based on feedback from `pyvisgraph` author Christian Reksten-Monsen, this was caused by round-off errors when using complex polygons. Much of the complexity was from jagged edges of the occupancy grid, an artifact of the raster representation. The polygons were simplified until the VG was correct. It was modified for task-specific VGs by adding start and goal for each. The network size and generation time of the unmodified VG is compared to the three uniform graphs in Table 2. The runtimes are less meaningful, since the uniform graph generation is performed by a simple, unoptimized python script, but the topologies are comparable and demonstrate substantial search space reduction.

An initial population of 100 candidate, feasible solutions was generated for $\mathcal{T}_1$ in 0.66 secs (Figure 4a),



(a) $\mathcal{W}_1$, 2017 May 03, 15 UTC (b) $\mathcal{W}_1$, 2017 May 03, 16 UTC (c) $\mathcal{W}_1$, 2017 May 03, 17 UTC

(d) $\mathcal{W}_2$, 2017 Aug 01, 00 UTC (e) $\mathcal{W}_2$, 2017 Aug 01, 01 UTC (f) $\mathcal{W}_2$, 2017 Aug 01, 02 UTC

(g) $\mathcal{W}_3$, 2019 Oct 01, 07 UTC (h) $\mathcal{W}_3$, 2019 Oct 01, 08 UTC (i) $\mathcal{W}_3$, 2019 Oct 01, 09 UTC

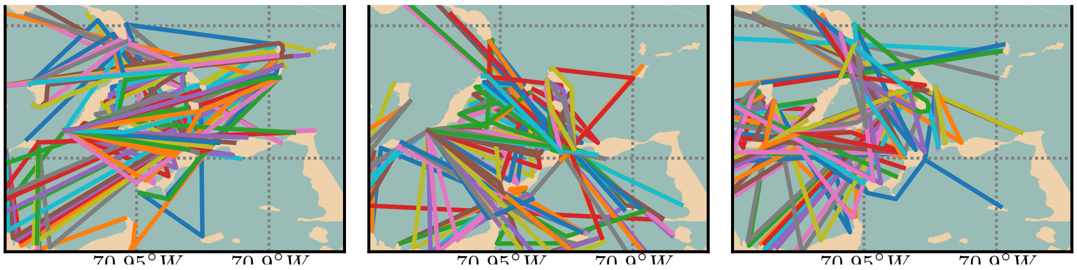
(j) $\mathcal{W}_4$, 2020 Aug 31, 00 UTC (k) $\mathcal{W}_4$, 2020 Aug 31, 01 UTC (l) $\mathcal{W}_4$, 2020 Aug 31, 02 UTC

Figure 3: The NECOFS (Beardsley, 2014) water current forecasts used in the experiments. Each row corresponds to a forecast, and each column to a time (hourly resolution). Each forecast is stored as a 3-channel raster, since 3 hours are sufficient to include the duration of all the paths produced by the experiments.

Table 2: Graph representations of Boston Harbor region.

Graph	**Nodes**	**Edges**	**Generated (secs)**
4-way	751111	2991126	4.2
8-way	751111	5975766	7.3
16-way	751111	11911544	13.3
VG	382	7470	3.7

for $\mathcal{T}_2$ in 0.74 secs (Figure 4b), and for $\mathcal{T}_3$ in 0.55 secs. All PSO experiments were performed both with a random and VG initial population. These variants are called PSO_R and PSO_{VG}, respectively.



(a) Task $\mathcal{T}_1$ (b) Task $\mathcal{T}_2$ (c) Task $\mathcal{T}_3$

Figure 4: Initial populations for each planning task. These replace the randomly generated initial set of candidate solutions used by PSO. Thus, PSO improves on these solutions to generate the energy-efficient solution. The set of paths is generated using A* on VGs, and include the shortest-distance path.

3.3. Energy efficient planning

3.3.1. Dijkstra and A experiments*

A* and Dijkstra solved on a combination of 5 water forecasts (including $\mathcal{W}_0$, no currents), 4 graph representations, and 3 planning tasks. Further, the 3 uniform graphs were tested with both graph-based and grid (raster)-based search implementations. Given the distance heuristic, A* was not expected to perform as effectively for energy-based planning. As expected, A* outperformed Dijkstra in $\mathcal{W}_0$, or, when the forecasts are ignored for shortest distance planning. However, A* failed to reliably outperform Dijkstra when using forecasts. This assessment is based not only on the runtimes, but by plotting all nodes visited by both algorithms. The energy-based cost function caused the A* search space exploration to be almost identical to Dijkstra.

Dijkstra results are shown in Figure 5. Bars are the solution cost with the 4 graphs. The computation time is noted for graph-based and grid-based versions. Solution improvements can be substantial. In $\mathcal{T}_1$, for forecasts $\mathcal{W}_1$ and $\mathcal{W}_2$, the 16-way cost is less than half of 4-way. However, the computation speed reflects the increased complexity. VG outperforms 4-way, but does not match the performance of the larger neighborhood graphs. This is expected since all edges in the graph were selected to aid shortest-distance planning. Given the closeness in speed to 8-way Dijkstra, the unpredictable VG is not recommended. 16-way solutions require significantly more time to produce, but the cost is similar to 8-way. This shows the diminishing return with increased complexity. Dijkstra paths, column one of Figure 6, highlight the poor 4-way solutions. The nature of VG forces each path segment to touch island polygon vertices, limiting its ability to navigate around the currents in open water as shown in Figure 6l.

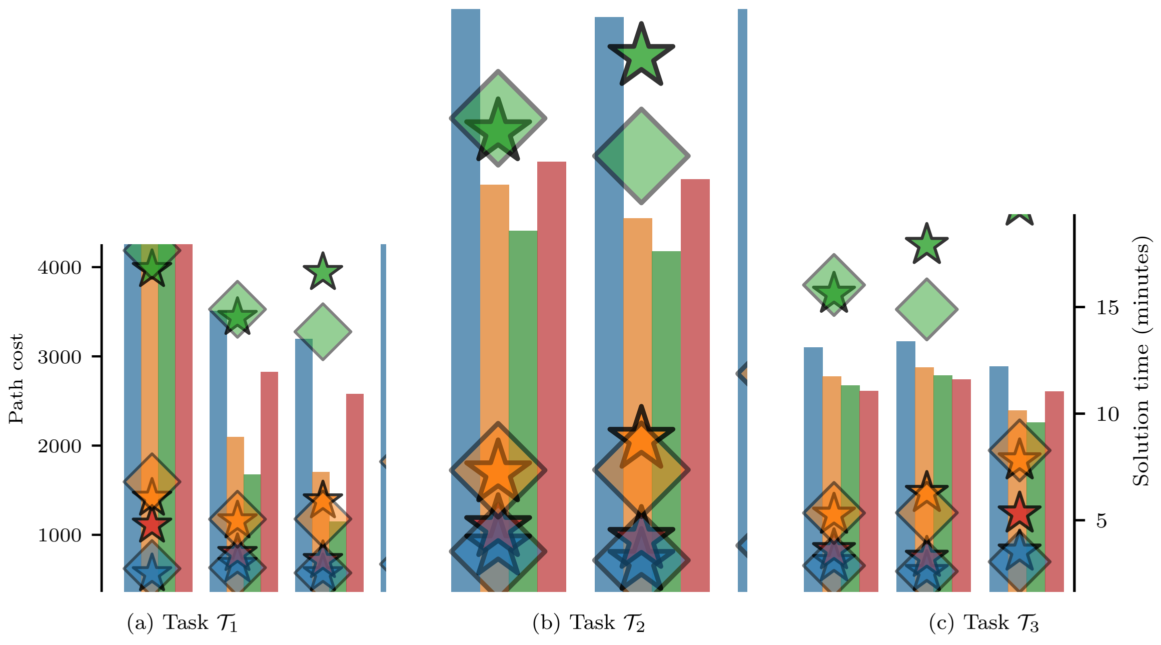
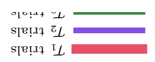


Figure 5: Dijkstra planning results using 4, 8, and 16-way neighborhood uniform graphs and VGs. Experiments include uniforms graphs represented directly as graphs and implicitly with an occupancy grid. Bars compare the solution cost between graphs. Stars and diamonds compare computation time needed for the graph and grid-based solutions, respectively. The choice of grid or graph impacts the time, but the neighborhood affects both time and solution cost.

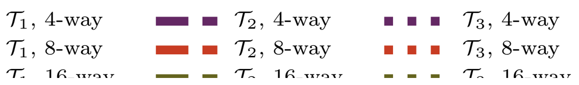
3.3.2. PSO parameter tuning

Candidate algorithms were two evolution-inspired (GA and DE) and two colony-inspired (PSO and DE). Initial tuning was performed to select the algorithm. This was done in previous work reported at ACC 2020 (Krell et al., 2020b). Each algorithm was run with combinations of a range of parameter values, 5 trials each. The GA parameters were crossover $(0.2, 0.4, \ldots, 1.0)$ and mutation $(0.2, 0.4, \ldots, 1.0)$ rates. DE parameters were the weight coefficient $(0.2, 0.4, \ldots, 1.0)$ and crossover probability $(0.2, 0.4, \ldots, 1.0)$. For ABC, only the trial limit $(10, 20, \ldots, 50)$ was tuned. Finally, tuning PSO involved the constriction w $(0.2, 0.4, \ldots 1.0)$, cognitive c_1 $(0.6, 1.6, \ldots, 4.0)$, and social c_2 $(0.6, 1.6, \ldots, 4.0)$ coefficients. PSO and ABC had similar behavior with slightly lower cost than PSO. GA and DE performed poorly, with extremely slow convergence and high cost variance. Specifically, the percent larger range of costs after 100 iterations compared PSO for ABC, DE, and GA were 25%, 2.5%, and 4.5%. Further, PSO's best was the lowest of the four.

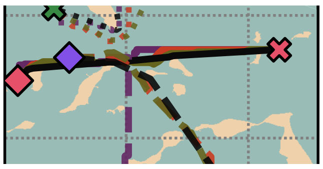
A second round of tuning has been performed with only PSO to account for any changes in hyperparams needed to better suit the improvements made to the cost function since ACC. All parameter combinations were evaluated using three trials with task $\mathcal{T}_1$ on forecasts $\mathcal{W}_0$ and $\mathcal{W}_1$. After finding the best results with the parameter ranges specified, additional finer-tuned local changes were made to hone in on the best. The



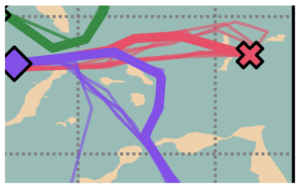
(a) Dijkstra legend.



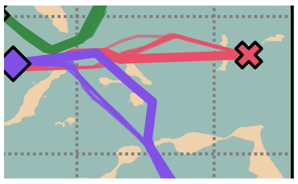
(b) PSO_R, PSO_{VG} legend.



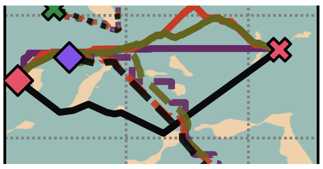
(c) $\mathcal{W}_1$, Dijkstra.



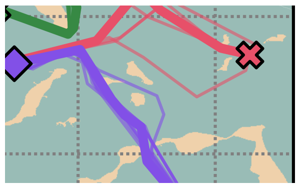
(d) $\mathcal{W}_1$, PSO_R.



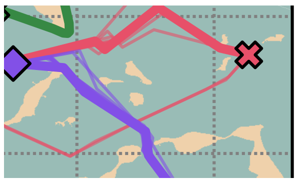
(e) $\mathcal{W}_1$, PSO_{VG}.



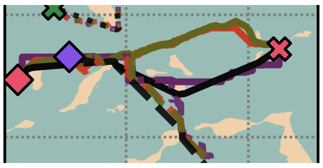
(f) $\mathcal{W}_2$, Dijkstra.



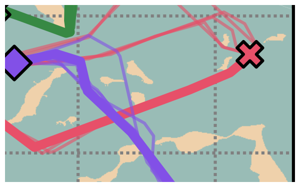
(g) $\mathcal{W}_2$, PSO_R.



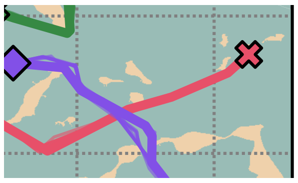
(h) $\mathcal{W}_2$, PSO_{VG}.



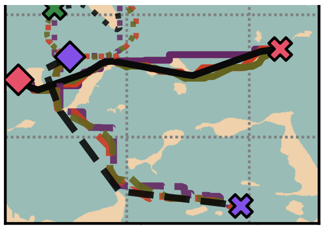
(i) $\mathcal{W}_3$, Dijkstra.



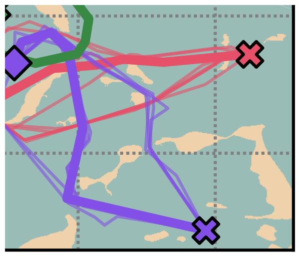
(j) $\mathcal{W}_3$, PSO_R.



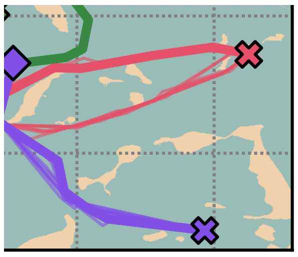
(k) $\mathcal{W}_3$, PSO_{VG}.



(l) $\mathcal{W}_4$, Dijkstra.



(m) $\mathcal{W}_4$, PSO_R.



(n) $\mathcal{W}_4$, PSO_{VG}.

Figure 6: Energy minimization results. With Dijkstra, a larger neighborhood yields a smoother path. PSO_R paths show much greater variation than PSO_{VG}.

best hyperparameters found were $w = 0.7, c_1 = 2.4, c_2 = 2.4$. The costs from these trials are within 1% of those when using the defaults hyperparameters of $w = 0.7298, c_1 = 2.05, c_2 = 2.05$, which suggests that the defaults would have been sufficient.

3.3.3. PSO experiments

PSO experiments were performed for all forecast, task combinations using PSO_R and PSO_{VG}. Each experiment was repeated for 10 trials to evaluate reliability. For each run, 5 waypoints were found such that the search space was 10 dimensional (each has an a and y component). This was based on previous experience with PSO planning and some trials that showed slower convergence with minimal solution improvements using 6 waypoints. Figure 6 shows all paths generated. Each row is for a water forecast with columns for Dijkstra, PSO_R, and PSO_{VG} results.

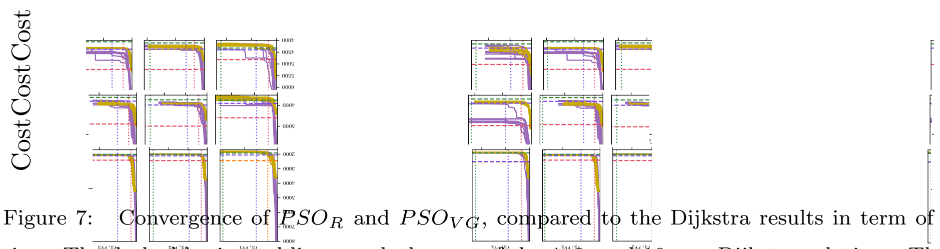


Figure 7: Convergence of PSO_R and PSO_{VG}, compared to the Dijkstra results in term of solution cost and computation time. The dashed horizontal lines mark the cost of the 4, 8, and 16-way Dijkstra solutions. The dotted vertical lines mark the time (in approx. number of PSO iterations) taken for each Dijkstra result. The goal is for PSO to approximate the 16-way Dijkstra cost while converging in fewer iterations.

For shortest distance ($\mathcal{W}_0$) planning, both PSO_R and PSO_{VG} results are almost identical to the Dijkstra VG result — the shortest path. A single T_2 outlier, using PSO_R, aligned with 8-way Dijkstra instead. The consideration of water currents greatly impacts the solution variance, as evidenced by the remaining results in Figure 6. The spatio-temporal cost landscape appears to introduce substantial local minima opportunities. The dynamic cost also makes it harder to visually assess the paths. Unlike the shortest distance, it is not obvious whether or not two spatially divergent paths may actually have a similar cost based on the currents.

In $\mathcal{W}_1$, the $\mathcal{T}_3$ results tend to agree with 16-way Dijkstra. Again, PSO_R has an outlier path that is closer to Dijkstra's worst (here, VG). $\mathcal{T}_2$ results are similar, but more dramatic. 4-way Dijkstra is very different from the other Dijkstra results, based on the decision to go the other way around a substantial island. PSO_R has a single path that does the same. In $\mathcal{W}_2$, it is PSO_{VG} that has a single outlier with significantly different behavior. For $\mathcal{W}_3$ and $\mathcal{W}_4$, PSO_R exhibits high path diversity.

From Figure 6, the overall trend is that the majority of PSO solutions are comparable to Dijkstra's. PSO_R results have very high path variance, with frequent outliers. Or, in the case of Figure 6m, not even a single dominant trend for $\mathcal{T}_1$ and $\mathcal{T}_2$ results. As expected, the use of VGs for the initial population has decreased the solution variance. The PSO_{VG} results include only a single major outlier (Figure 6h).

Figure 7 shows the evolution of costs over time with Dijkstra benchmarks. The solution costs for the four graphs are horizontal lines for comparing to the converged PSO cost. The vertical lines are seconds elapsed to find that solution. These results highlight the effectiveness of PSO_{VG}. The spatial variance is not reflected in the cost variance, suggesting that the complexity of the currents enable to vehicle to make spatially disparate paths with the same approximate cost. In general, PSO_{VG} results match or outperform 8-way. Depending on the task, it might not reach the 16-way solution. Since Dijkstra can make arbitrary turns and PSO is limited to 5 waypoints, this is expected. On the other hand, PSO_R is shown to be extremely unreliable. The random initial population strongly impacts the rate of convergence. PSO_{VG} begins with a lower cost, and quickly converges. Rather than a premature convergence, the solutions tend to be better than PSO_R. Instead, it is the slower PSO_R that clearly displays instances of convergence to local optimum. This is dramatically highlighted with the $(\mathcal{T}_2, \mathcal{W}_4)$ results. While not perfect, still limited by the sensitivity to local optima and the problem with exponential complexity with the increase in waypoints, the results suggest that PSO_{VG} has substantial advantage over the default random population.

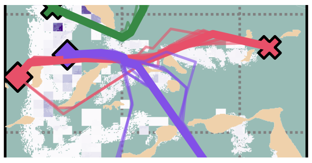
3.4. Balancing energy expenditure and reward

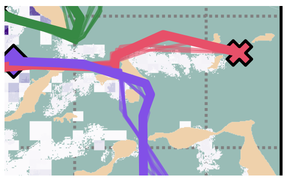
Reward-based planning is enabled using a reward raster G_r scaled by a reward weight W_R. Increasing W_R decreases the solution cost to encourage the path the intersect higher-reward cells. To explore planning in their combined, complex solution space, the following experiments incorporate both work (water currents) and reward. Based on a sweep across a variety of W_R values, two were selected for presenting the results here. The first, $W_R = 1500$, clearly demonstrates paths focused on reward collection. That is, the reward weight is high enough for visibly obvious path deviations for reward maximization. In practice, smaller weights are expected be more desirable for more subtle deviations. The second, $W_R = 2000$, results are included to compare how the PSO variants are impacted by the change in search space complexity. For brevity, the two reward weights will be referred to as $1x$ and $1.5x$, respectively. Experiments are performed with both PSO_R and PSO_{VG}.

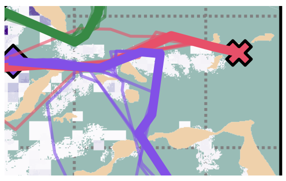
From Figure 8, the planning results can be analyzed by comparing PSO_R and PSO_{VG} as well as the effect of W_R values $1x$ and $1.5x$. Columns 1 and 2 show the paths for PSO_R and PSO_{VG}, respectively, given $W_R = 1x$. Columns 3 and 4 show the paths for PSO_R and PSO_{VG}, respectively, given $W_R = 1.5x$. The rows correspond to water current scenarios $\mathcal{W}_1 \ldots \mathcal{W}_4$. The $PSO_R, \mathcal{W}_R = 1x$ results demonstrate very high path variance and large deviations from the energy efficient paths in Figure 6. The paths appear to be significantly more reliable using PSO_{VG}. Not only is the variance constrained, but also the paths are relatively similar to the energy efficient paths. Consider the paths generated for scenario $\mathcal{W}_2$, Figures 8f and 8g. With PSO_R, the T_2 trials may yield complex paths with large segments going opposite to the goal. The PSO_{VG} trials manage to intersect reward cells while aligning with the goal of collecting opportunistic, along-route reward. Similar comparisons can be made for $\mathcal{W}_3$ and $\mathcal{W}_4$. The most dramatic is $\mathcal{W}_4$, comparing the $\mathcal{T}_2$ trials across Figures 8n and 8o. Here, PSO_{VG} trials exhibit some degree of variance, but the variance is constrained to a segment in the map with reward. Leaving that section, the paths match their energy efficient counterparts (Figure 6n).

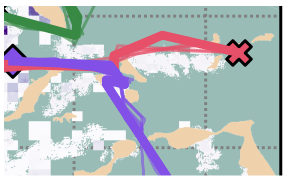
As shown in Figure 8's columns 2, increasing the value of W_R substantially increased path variance for

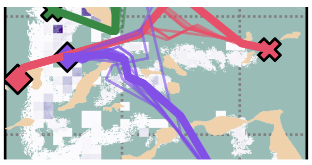
(a) Legend.

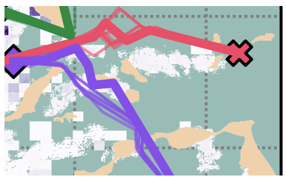

(b) $\mathcal{W}_1$, PSO_R, $W_R = 1x$.

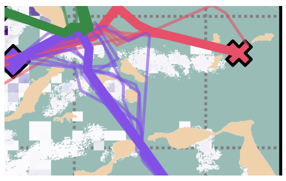

(c) $\mathcal{W}_1$, PSO_{VG}, $W_R = 1x$.

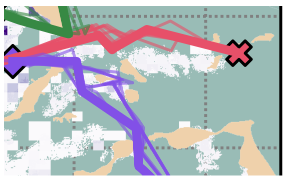

(d) $\mathcal{W}_1$, PSO_R, $W_R = 1.5x$.

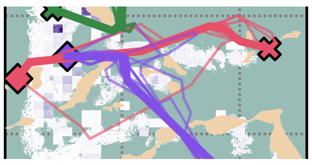

(e) $\mathcal{W}_1$, PSO_{VG}, $W_R = 1.5x$.

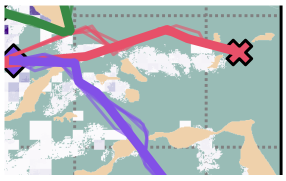

(f) $\mathcal{W}_2$, PSO_R, $W_R = 1x$.

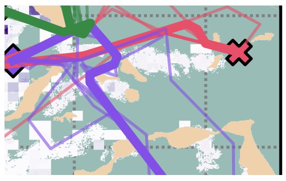

(g) $\mathcal{W}_2$, PSO_{VG}, $W_R = 1x$.

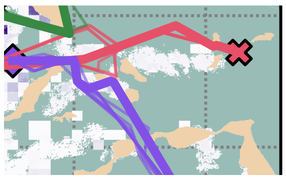

(h) $\mathcal{W}_2$, PSO_R, $W_R = 1.5x$.

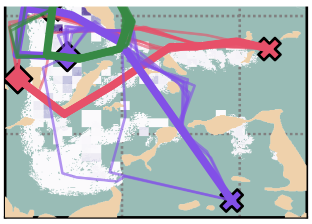

(i) $\mathcal{W}_2$, PSO_{VG}, $W_R = 1.5x$.

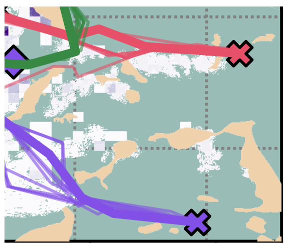

(j) $\mathcal{W}_3$, PSO_R, $W_R = 1x$.

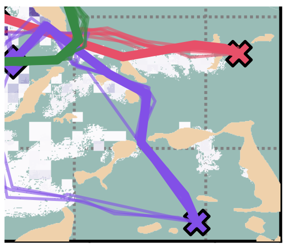

(k) $\mathcal{W}_3$, PSO_{VG}, $W_R = 1x$.


(l) $\mathcal{W}_3$, PSO_R, $W_R = 1.5x$.


(m) $\mathcal{W}_3$, PSO_{VG}, $W_R = 1.5x$.


(n) $\mathcal{W}_4$, PSO_R, $W_R = 1x$.


(o) $\mathcal{W}_4$, PSO_{VG}, $W_R = 1x$.


(p) $\mathcal{W}_4$, PSO_R, $W_R = 1.5x$.

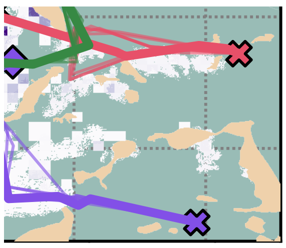

(q) $\mathcal{W}_4$, PSO_{VG}, $W_R = 1.5x$.

Figure 8: Planning results given water forecasts and reward.

tasks $\mathcal{T}_1$ and $\mathcal{T}_2$. In fact, there is one instance where the path is infeasible — intersecting a very narrow piece of land (Figure 8l). While this does highlight that the use of soft constraints for obstacle avoidance requires care in complex, multi-objective problems, it also shows that the constrained initial population of PSO_{VG} seems to avoid this problem. Comparing columns 2 and 4, PSO_{VG} appears to be much more robust to the increasingly complex solution space. The paths are similar overall, with the $W_R = 1.5x$ paths showing greater coverage of reward cells. This is exactly the desired tunable behavior.

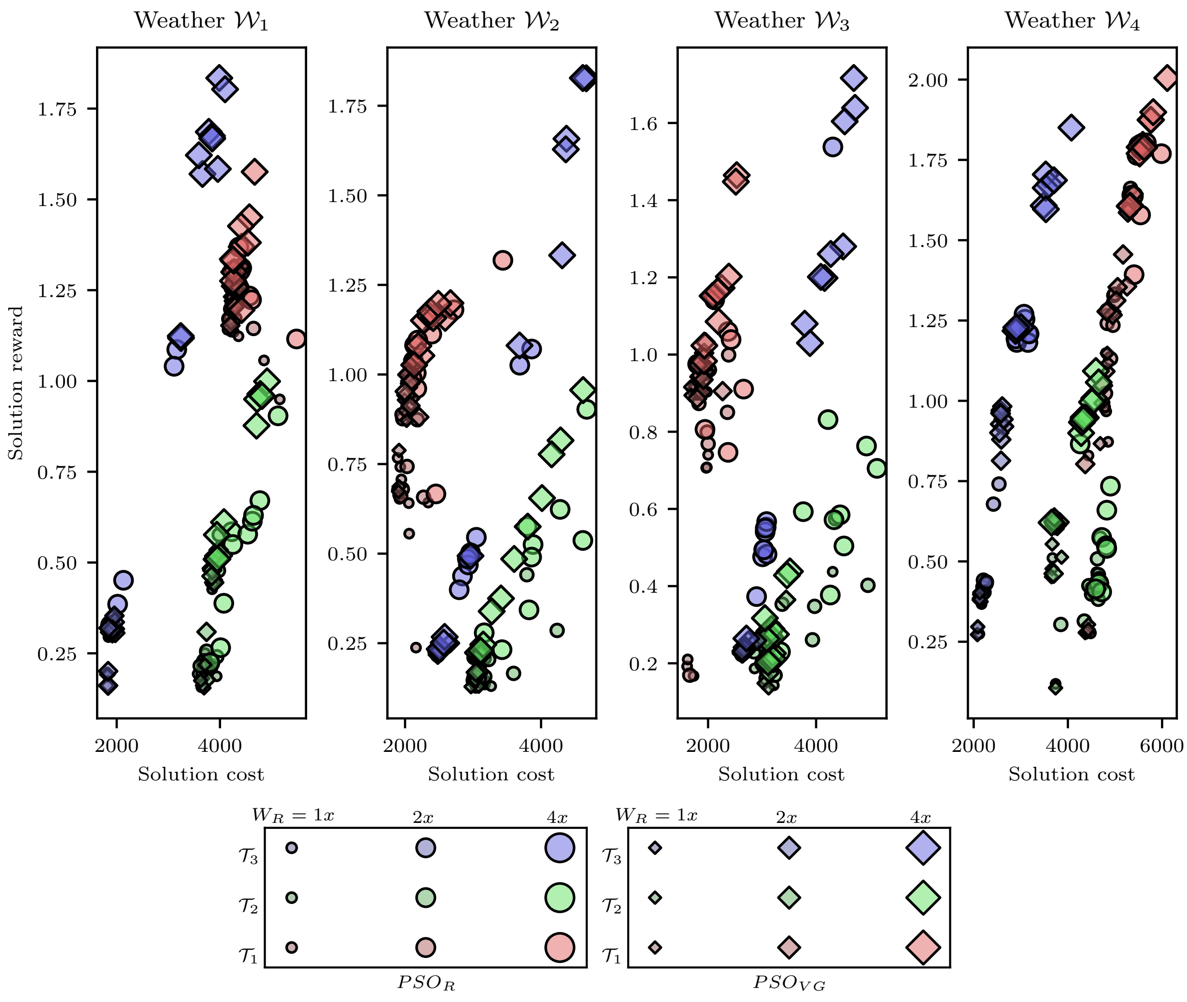


Figure 9: The influence of reward weight W_R on the path cost and reward. Experiments are performed with reward weights $W_R = 1x$, $1.5x$, and $2x$.

The influence of modifying the reward weight on the solution in shown in Figure 9. Each point represents a single trial. This includes the trials when $W_R = 1\times$ and $2\times$, the paths shown in Figure 8, as well as $W_R = 1.5\times$. Increasing W_R tends to increase both reward and cost, as expected. In some scenarios, there appears to be a roughly linear relationship between reward and cost. However some scenarios, most obviously ($\mathcal{T}_1$, $\mathcal{W}_2$), the increase in reward slows rapidly.

PSO_{VG} tends to achieve higher reward than PSO_R. In some scenarios, PSO_R and PSO_{VG} solutions are relatively close to each other, but with the PSO_{VG} results towards the upper-left, indicating more reward at less cost. This can be observed for all scenarios for tasks $\mathcal{T}_1$ and 3/4 for $\mathcal{T}_2$. The PSO_{VG} results also tend to be better clustered than PSO_R, suggesting greater reliability. An exception is seen in ($\mathcal{T}_2$, $\mathcal{W}_2$), where there is a large variation in both PSO_{VG} and PSO_R solutions, the former achieving strictly better reward and cost.

With $\mathcal{T}_3$, PSO_{VG} tends to achieve substantially more reward, at a larger cost. This is apparent under each forecast at $W_R = 2\times$. The PSO_{VG} solutions may be $> 2\times$ the reward of PSO_R. In this case, we argue that PSO_{VG} outperforms PSO_R because it responds more strongly to the tunable W_R. The user could decide that the PSO_{VG} solutions under $W_R = 2\times$ incur too much cost, and can choose to lower W_R. On the other hand, PSO_R solutions under $W_R = 2\times$ are often clustered closely with those under $W_R = 1.5\times$, less separable with the tuning parameter.

These results suggest that PSO_{VG} outperforms PSO_R in the opportunistic reward task, in terms of the solution fitness, solution reliability, and response to tuning with W_R. However, both PSO_R and PSO_{VG} show scenarios with high variation. For a given (task, forecast, weight), individual trials may vary in cost by $2\times$. Additional work is required to constrain the costs, without sacrificing the computational speedups gained with a metaheuristic algorithm under soft constraints.

4. Conclusions

In this work, ASV planning was performed using PSO with and without using VGs to initialize the population. A fitness function was developed for, depending on the weights, both energy efficiency and reward. The results were compared to the optimal Dijkstra solutions. Dijkstra results showed substantial improvement going from 4-way to 8-way, and subtle improvement with 16-way. VG solutions were slightly faster than 8-way, but without reliably matching its fitness. PSO experiments were performed with two variants. PSO_R used a random set of waypoint sequences. PSO_{VG} had feasible solutions, including the shortest path, extracted from a VG using A*. Experiments were performed for energy efficient planning. PSO_R was unable to reliably produce competitive paths with significant variance, slow convergence, and poor solutions. However, PSO_{VG} exhibited tight solution bounds with fast convergence and fitness comparable to at least 8-way Dijkstra, but sometimes meeting or surpassing 16-way depending on the scenario.

Reward optimization was added, increasing search complexity. A qualitative assessment showed PSO_{VG} results meeting planning objectives. The paths increased sampling while staying close to efficient paths. Tuning the reward weight W_R adjusted the balance between efficiency and sampling.

Our next step is to make it easier to control how much efficiency is allowed to be sacrificed when optimizing opportunistic reward. That is, how much should the solutions deviate from the low cost solution to be considered opportunistic. We will add a parameter that is the maximum percentage that the reward maximization can take a solution from the lowest cost found so far (after an initial exploration period). At each step, the PSO population will be (at least partially) pruned such that only those within a threshold defined by the most energy efficient individual is allowed. This is expected to lower the cost variance when considering the path reward.

Acknowledgements

The computations in this study were performed on resources provided by the Island Science and Engineering Research Cloud (ISERC) funded by the National Science Foundation [grant number 1828380].

References

Bansal, J.C., Singh, P.K., Saraswat, M., Verma, A., Jadon, S.S., Abraham, A., 2011. Inertia weight strategies in particle swarm optimization, in: 2011 Third World Congress on Nature and Biologically Inspired Computing, pp. 633–640. doi:`10.1109/NaBIC.2011.6089659`.

Barua, D.K., 2005. Beaufort Wind Scale. Springer Netherlands, Dordrecht. pp. 186–186. doi:`10.1007/1-4020-3880-1_45`.

Beardsley, R.C. & Chen, C., 2014. Northeast coastal ocean forecast system (necofs): A multi-scale global-regional-estuarine fvcom model, in: AGU Fall Meeting Abstracts.

Bresenham, J., 1965. Algorithm for computer control of a digital plotter. IBM Syst. J. 4, 25–30. doi:`10.1147/sj.41.0025`.

Coleman, D., 2012. Lee's o (n^2 log n) visibility graph algorithm implementation and analysis.

Dijkstra, E., 1959. A note on two problems in connexion with graphs. Numerische mathematik 1, 269–271.

Eberhart, R., Kennedy, J., 1995. Particle swarm optimization, in: Proceedings of the IEEE international conference on neural networks, Citeseer. pp. 1942–1948.

Forrest, S., 1993. Genetic algorithms: principles of natural selection applied to computation. Science 261, 872–878.

Innocente, M., Sienz, J., 2010. Coefficients' settings in particle swarm optimization: insight and guidelines. Mecánica Computacional 29, 9253–9269.

Jones, D.F., Mirrazavi, S.K., Tamiz, M., 2002. Multi-objective meta-heuristics: An overview of the current state-of-the-art. European journal of operational research 137, 1–9.

Karaboga, D., 2010. Artificial bee colony algorithm. scholarpedia 5, 6915.

Kitzinger, J., Moret, B., 2003. The visibility graph among polygonal obstacles: a comparison of algorithms. Ph.D. thesis. University of New Mexico.

Krell, E., 2021a. Conch software. `https://github.com/ekrell/conch`. Commit: 5582d5bca1cba328d3463965b1e9fa69a454708f.

Krell, E., 2021b. Whelk software. `https://github.com/ekrell/whelk`. Commit: aec3ed3455656755fa16ce9921ecb03685df5e81.

Krell, E., Garcia-Carrillo, L., King, S., Hespanha, J., 2020a. Game theoretic potential field for autonomous water surface vehicle navigation using weather forecasts, in: 2020 American Control Conf., IEEE. pp. 2112–2117.

Krell, E., King, S., Garcia-Carrillo, L., 2020b. Autonomous water surface vehicle metaheuristic mission planning using self-generated goals and environmental forecasts, in: 2020 American Controls Conf., IEEE.

Krell, E., Sheta, A., Balasubramanian, A., King, S., 2019. Collision-free autonomous robot navigation in unknown environments utilizing pso for path planning. Journal of Artificial Intelligence and Soft Computing Research 9, 267–282.

Kuhlemann, S., Tierney, K., 2020. A genetic algorithm for finding realistic sea routes considering the weather. Journal of Heuristics 26, 801–825.

Lim, H., Fan, S., Chin, C., Chai, S., Bose, N., Kim, E., 2019. Constrained path planning of autonomous underwater vehicle using selectively-hybridized particle swarm optimization algorithms. IFAC-PapersOnLine 52, 315–322.

Liu, C., Hao, Y., Gao, F., 2011. A path planning method for underwater vehicle based on ocean current information, in: 2011 Fourth International Joint Conf. on Computational Sciences and Optimization, IEEE. pp. 987–991.

Niu, H., Lu, Y., Savvaris, A., Tsourdos, A., 2018. An energy-efficient path planning algorithm for unmanned surface vehicles. Ocean Engineering 161, 308–321.

[dataset] NOAA, 2017. World vector shorelines. `http://www.ngdc.noaa.gov/mgg/shorelines/`.

[dataset] Northeast Regional Coastal Ocean Observation System, 2021. The northeast coastal ocean forecast system. http://fvcom.smast.umassd.edu/necofs/.

Nosrati, M., Karimi, R., Hasanvand, H.A., 2012. Investigation of the*(star) search algorithms: Characteristics, methods and approaches. World Applied Programming 2, 251–256.

Patel, A., 2014. Implementation of a*. www.redblobgames.com/pathfinding/a-star/implementation.html. Accessed on 2021-11-02.

QGIS Development Team, 2018. Qgis geographic information system (version 2.18). http://qgis.osgeo.org.

Reksten-Monsen, C., 2018. pyvisgraph. github.com/TaipanRex/pyvisgraph. Commit: 9472e0df7ca7bfa5f45932ca348222bdc1f7e688.

Shah, B., Gupta, S., 2019. Long-distance path planning for unmanned surface vehicles in complex marine environment. IEEE Journal of Oceanic Engineering .

Storn, R., Price, K., 1997. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization 11, 341–359.

Xu, J., Gu, H., Liang, H., 2018. Path planning for unmanned underwater vehicle based on improved particle swarm optimization method. International Journal of Online and Biomedical Engineering (iJOE) 14, 137–149.

Yang, G., Zhang, R., 2009. Path planning of auv in turbulent ocean environments used adapted inertia-weight pso, in: 2009 Fifth International Conf. on Natural Computation, IEEE. pp. 299–302.

Zambrano-Bigiarini, M., Clerc, M., Rojas, R., 2013. Standard particle swarm optimisation 2011 at cec-2013: A baseline for future pso improvements, in: 2013 IEEE Congress on Evolutionary Computation, IEEE. pp. 2337–2344.

Zeng, Z., Sammut, K., Lammas, A., He, F., Tang, Y., 2015. Efficient path re-planning for auvs operating in spatiotemporal currents. Journal of Intelligent & Robotic Systems 79, 135–153.

Zhang, Z., Sturtevant, N., Holte, R., Schaeffer, J., Felner, A., 2009. A* search with inconsistent heuristics., in: IJCAI, pp. 634–639.