# 3D large grid route planner for the autonomous underwater vehicles

Hua Cao, Nathan E. Brener and S. Sitharama Iyengar
*Robotics Research Laboratory, Department of Computer Science, Louisiana State University, Baton Rouge, Louisiana, USA*

Received 2 October 2008
Revised 20 December 2008
Accepted 3 January 2009

## Abstract

**Purpose** – The purpose of this paper is to develop a 3D route planner, called 3DPLAN, which employs the Fast-Pass A* algorithm to find optimum paths in the large grid.

**Design/methodology/approach** – The Fast-Pass A* algorithm, an improved best-first search A* algorithm, has a major advantage compared to other search methods because it is guaranteed to give the optimum path.

**Findings** – In spite of this significant advantage, no one has previously used A* in 3D searches. Most researchers think that the computational cost of using A* for 3D route planning would be prohibitive. This paper shows that it is quite feasible to use A* for 3D searches if one employs the new mobility and threat heuristics that have been developed.

**Practical implications** – This paper reviews the modification of the previous 3DPLAN in the ocean dynamical environment. The test mobility map is replaced with more realistic mobility map that consists of travel times of each grid point to each of its 26 neighbors using the actual current velocity data from the Navy Coastal Ocean Model – East Asian Seas version. Numerical comparison between the A* and genetic algorithms (GA) shows that the A* algorithm has significantly faster running time than GA.

**Originality/value** – These new heuristics substantially speed up the A* algorithm so that the run times are quite reasonable for the large grids that are typical of 3D searches.

**Keywords** Programming and algorithm theory, Underwater technology, Robotics, Underwater navigation

**Paper type** Research paper

## 1. Introduction

Route planning has been widely used for civilian and military purposes (Nishi *et al.*, 2005). 3D route planning is especially useful for the navigation of combat helicopters and autonomous underwater vehicles (AUVs). 3D route planning is a very challenging problem because the large grids that are typically required can cause a prohibitive computational burden if one does not use an efficient search algorithm. Several search algorithms have been proposed to perform 3D route planning, including case-based reasoning (Kruusmaa and Svensson, 1998; Vasudevan and Ganesan, 1996) and genetic algorithms (GA) (Smith and Sugihara, 1996; Sugihara and Yuh, 1997).

Case-based reasoning relies on specific instances of past experience to solve new problems. A new path is obtained by searching previous routes to locate the one that matches the current situation in the features, goals and constraints. The new path is

This paper was supported in part by DOE-ORNL grant No. 4000008407 and by NSF grant.

generated by modifying an old path in the previous path database using a set of repair rules. However, since the number of possible threat distributions is very large for most battle areas, it would not be feasible to store old routes for all or most of the possible threat arrangements. Thus, the case-based method is not suitable for handling threats in route planning. In addition, when it has to synthesize complete new routes (in an area where no old paths are available) or modify old routes by synthesizing new segments, it does not use a guaranteed best-first search algorithm such as $A^*$, but instead uses straight-line segments going around obstacles. Thus, the routes are neither locally nor globally optimal.

The GA method is a stochastic search technique based on the principles of biological evolution, natural selection, and genetic recombination. GA generates a population of solutions. And such solutions mate and bear offspring solutions in the next generation. The solutions in the population improve over many generations until the best solution is obtained. However, GA have been accepted slowly for research problems because crossing two feasible solutions does not, in many cases, result in a feasible solution as an offspring. The other disadvantage of GA is that although it can generate solutions to a route planning problem, it cannot guarantee that the solution is optimal, i.e. it can converge to a local, rather than a global, minimum (Zheng *et al.*, 2005; Zheng, 2008).

The $A^*$ algorithm (Hart *et al.*, 1968), is a guaranteed best-first search algorithm that has been used previously in 2D route planning by several researchers (Benton *et al.*, 1996; Brener *et al.*, 2004). A major advantage of the $A^*$ algorithm compared to the other methods is that $A^*$ is guaranteed to give the optimum path. In spite of this significant advantage, no one has previously used the $A^*$ algorithm for 3D route planning. Most people think that the computational cost of using $A^*$ for 3D route planning would be prohibitive (Szczerba *et al.*, 2000; McVey *et al.*, 1999). In this paper, we show that, it is quite feasible to use the Fast-Pass $A^*$ for 3D searches if one employs the new mobility and threat heuristics that we have developed. These new heuristics substantially speed up the $A^*$ algorithm so that the run times are quite reasonable for the typical large grids 3D searches.

## 2. 2D mobility and threat maps

### *2.1 Mobility and threat maps*

Brener and Iyengar, in collaboration with Benton, have previously developed a 2D $A^*$ route planner (Benton *et al.*, 1996; Brener *et al.*, 2004), called the Predictive Intelligence Military Tactical Analysis System (PIMTAS), for military terrain vehicles such as tanks.

Figure 1 shows an example of a 2D mobility map (top) and 2D threat map (bottom) that were used as input to PIMTAS. The upper map in the figure is an actual mobility map of an area near Lauterbach, Germany, and the lower map is a prototype threat map which was generated in order to test the program. Both maps have 237 by 224 grid points. The mobility map has four types of GO regions represented by the colors green, light green, yellow-orange, and orange, which denote unlimited, limited, slow, and very slow areas, respectively. The mobility penalty for grid points in the unlimited, limited, slow, and very slow regions is 1-4, respectively. Thus, the minimum mobility penalty at each grid point is 1. In general, the mobility map has three types of NO-GO regions represented by the colors red, blue, and white, which denote impassable

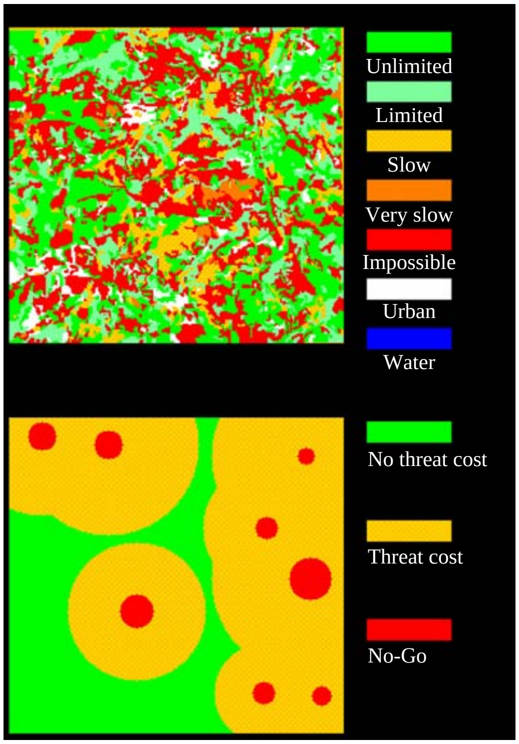


**Figure 1.**
2D mobility map (top) and 2D threat map (bottom)

obstacles, water, and urban areas, respectively. This last restriction follows military doctrine that urban areas are to be avoided. In the prototype threat map, each threat is modeled by a red inner circle where the vehicle is not allowed to go since it would almost certainly be destroyed if it came that close to the threat, and an orange outer circle where the vehicle is within range of the threat. The green regions are outside of the range of all of the threats. The threat penalty for each threat varies linearly from 1 to 0 as one goes radically outward from the boundary of the red circle to the boundary of the orange circle. If a grid point is within the orange circle of more than one threat, the threat penalty at that point is the sum of the threat penalties of all of the threats acting on that point. Grid points in the green regions have a threat penalty of 0.

## *2.2 Fast-pass A * algorithm*

PIMTAS employs the A$^*$ algorithm, in which the total cost, $f$, of the path that goes through a particular grid point (this grid point will be referred to as the current point) is given by:

$$f = g + h \tag{1}$$

where $g$ is the actual cost that was accumulated in going from the starting point to the current point and $h$ is an underestimate of the remaining cost required to go from the current point to the target. The heuristic $h$ is the key quantity that determines how efficiently the algorithm works. $h$ must not only be a guaranteed underestimate of the remaining cost, which ensures that no potential optimum paths will be discarded due to overestimating their total cost, but must also provide as close an estimate as possible of the remaining cost. The closer $h$ is to the actual remaining cost is, the faster the algorithm will find the optimum path. With a proper choice of $h$, the algorithm can be highly efficient.

The actual accumulated cost $g$ is given by:

$$g = \alpha_{\mathrm{m}} M + \alpha_{\mathrm{t}} T \tag{2}$$

where, $M$, accumulated mobility penalty; $T$, accumulated threat penalty; $\alpha_{\mathrm{m}}$, mobility weight; $\alpha_{\mathrm{t}}$, threat weight.

The weights $\alpha_{\mathrm{m}}$ and $\alpha_{\mathrm{t}}$ are entered by the military planner.

The accumulated mobility and threat penalties are given by:

$$M = \frac{\sum R_i (M_{i-1} + M_i)}{2} \tag{3}$$

$$T = \frac{\sum R_i (T_{i-1} + T_i)}{2} \tag{4}$$

where the sum is over the grid points traversed in going from the starting point to the current point, $M_i$ is the mobility penalty at grid point $i$, $T_i$ is the threat penalty at grid point $i$, and $R_i$ is the stepsize to go from grid point $i-1$ to grid point $i$.

The heuristic, which is an underestimate of the remaining cost required to go from the current point to the target, is given by:

$$h = \alpha_{\mathrm{m}} h_{\mathrm{m}} + \alpha_{\mathrm{t}} h_{\mathrm{t}} \tag{5}$$

where, $h_{\mathrm{m}}$, underestimate of remaining mobility penalty (mobility heuristic); $h_{\mathrm{t}}$, underestimate of remaining threat penalty (threat heuristic).

### *2.3 Mobility heuristic*

The mobility heuristic employed in the presented A$^*$ algorithm is more efficient than the straight line heuristic used in previous A$^*$ approaches, since our new heuristic is larger than the straight line heuristic and still is an underestimate of the remaining cost.

Figure 2 shows a 2D mobility map that consists of a square grid of points. The step size in the $x$ and $y$ directions is labeled $r_1$ and the diagonal step size will be labeled $r_2$, where $r_2 = \sqrt{2} \times r_1$. In this figure, each grid point has a mobility penalty of either 1, 2, 3, or 4 (i.e. 1 is the minimum mobility penalty at each grid point) and $P_1$, $P_2$, and $P_3$ are the start point, current point, and target point, respectively.

Let, $n_x$, number of steps in $x$-direction between current point and target; $n_y$, number of steps in $y$-direction between current point and target.

Then the mobility heuristic $h_{\mathrm{m}}$ is given by:

$$h_{\mathrm{m}} = \begin{array}{ll} n_y r_2 + (n_x - n_y) r_1 & \text{for } n_x > n_y \\ n_x r_2 + (n_y - n_x) r_1 & \text{for } n_y \geq n_x \end{array} \tag{6}$$

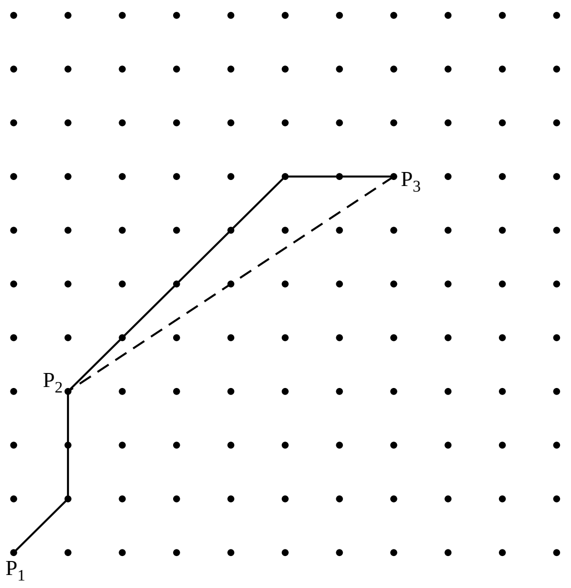


**Figure 2.** 2D mobility map consisting of a square grid of points

In the example shown in Figure 2, the mobility heuristic to go from the current point $P_2$ to the target $P_3$ is:

$$h_\mathrm{m} = 4 \times r_2 + 2 \times r_1 \tag{7}$$

This is larger than the straight line distance from $P_2$ to $P_3$, which is what other authors have used as the heuristic, and is still a guaranteed underestimate of the remaining mobility penalty to go from $P_2$ to $P_3$. Thus, our mobility heuristic will cause the $\mathrm{A}^*$ algorithm to run faster than it would with a straight line heuristic.

## 3. 3D mobility and threat maps

In our new 3D route planner (3DPLAN), the 2D mobility and threat maps described above have been extended to 3D in order to test the program. In 3DPLAN, the search region is represented by a digital map consisting of a Cartesian grid of points in which the step size in the *x*- and *y*-directions is the same but the step size in the *z*-direction is in general different. Both the 3D mobility map and the 3D threat map have $237 \times 224 \times 150$ grid points in the *x*, *y*, and z-directions for a total of almost 8 million points. To our knowledge, this is the largest number of grid points that has ever been used in an $\mathrm{A}^*$ search. In the 3D mobility map, each grid point in the GO regions has a mobility penalty of 1, 2, 3, or 4 depending on the mobility conditions. Grid points that are located in impassible areas are labeled as avoided points where the vehicle is not allowed to go. This test mobility map will be replaced with a realistic map of actual travel times when 3DPLAN is applied to underwater vehicles. In our prototype 3D threat map, each threat is modeled by an inner sphere where the vehicle is not allowed to go and an outer sphere where the vehicle is within range of the threat. For each threat, the threat penalty varies linearly from 1 to 0 as one goes radially outward from the surface of the inner sphere to the surface of the outer sphere. As in the 2D case, if a grid point is within the range of more than one threat, its threat penalty is the sum of the penalties of all of the threats acting on that point. Grid points that are outside of the range of all of the threats have a threat penalty of 0.

Given this underwater scenario, a starting point, and a target to be reached, the military planner enters a weight for each of the two path cost factors being considered:

(1) mobility; and

(2) threats.

3DPLAN will then quickly generate the lowest cost path from the starting point to the target, where the cost of the path is determined by multiplying the weight for each factor by the accumulated penalty for that factor. By entering a particular set of weights, the military planner can put any desired degree of emphasis on each of the cost factors. For example, a large weight for mobility and a small weight for threats would produce a fast path that may go close to enemy threats, while a large weight for threats and a small weight for mobility would produce a path that stays as far away from threats as possible and consequently may require a considerably longer travel time.

*3.1 3D mobility heuristic*

It is straightforward to extend this mobility heuristic to 3D. Figure 3 shows a cell in the 3D map in which the step size in the $x$- and $y$-directions is the same but the step size in the $z$-direction is in general different. The figure shows the five possible step sizes, labeled $r_1$,-$r_5$, that the vehicle can take to go from a grid point to one of its neighbors, where, $r_1$, step size in $x$- and $y$-directions; $r_2$, sqrt(2) Xr1; $r_3$, step size in $z$-direction; $r_4 = \sqrt{r_1^2 + r_3^2}$; $r_5 = \sqrt{r_2^2 + r_3^2}$.

Let, $n_x$, number of steps in $x$-direction between current point and target; $n_y$, number of steps in $y$-direction between current point and target; $n_{z,}$ number of steps in $z$-direction between current point and target.

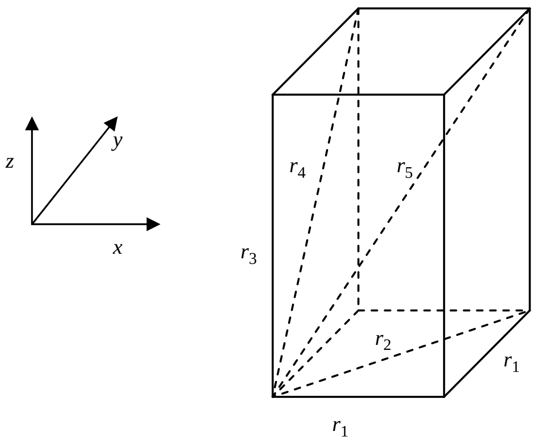


**Figure 3.**
A cell in the 3D mobility map

The 3D mobility heuristic $h_\mathrm{m}$ is then given by:

$$h_\mathrm{m} = \begin{cases} n_x{*}r_5 + (n_y - n_x){*}r_4 + (n_z - n_y){*}r_3 & \text{for } n_x \leq n_y \leq n_z \\ n_x{*}r_5 + (n_z - n_x){*}r_4 + (n_y - n_z){*}r_1 & \text{for } n_x \leq n_z \leq n_y \\ n_y{*}r_5 + (n_x - n_y){*}r_4 + (n_z - n_x){*}r_3 & \text{for } n_y \leq n_x \leq n_z \\ n_y{*}r_5 + (n_z - n_y){*}r_4 + (n_x - n_z){*}r_1 & \text{for } n_y \leq n_z \leq n_x \\ n_z{*}r_5 + (n_x - n_z){*}r_2 + (n_y - n_x){*}r_1 & \text{for } n_z \leq n_x \leq n_y \\ n_z{*}r_5 + (n_y - n_z){*}r_2 + (n_x - n_y){*}r_1 & \text{for } n_z \leq n_y \leq n_x \end{cases} \tag{8}$$

### *3.2 Threat heuristic*

A threat heuristic has been rarely used in the A$^*$ algorithm because the minimum threat penalty is 0 rather than 1. Thus, if one tried to use the same technique for the threat heuristic as was used for the mobility heuristic, the threat heuristic would be 0. In this paper, we present a nonzero threat heuristic that is different from the mobility heuristic and is. Thus, our new threat heuristic will speed up the A$^*$ algorithm compared to not having a threat heuristic.

Figure 4 shows the same square grid of points that was shown in Figure 2. The threat heuristic $h_\mathrm{t}$ will be an underestimate of the remaining threat penalty to go from the current point to the target point. In order to construct this threat heuristic, the concentric squares around the target are labeled 1, 2, 3... and the target point is square zero. In order to go from the current point to the target, the vehicle must visit each

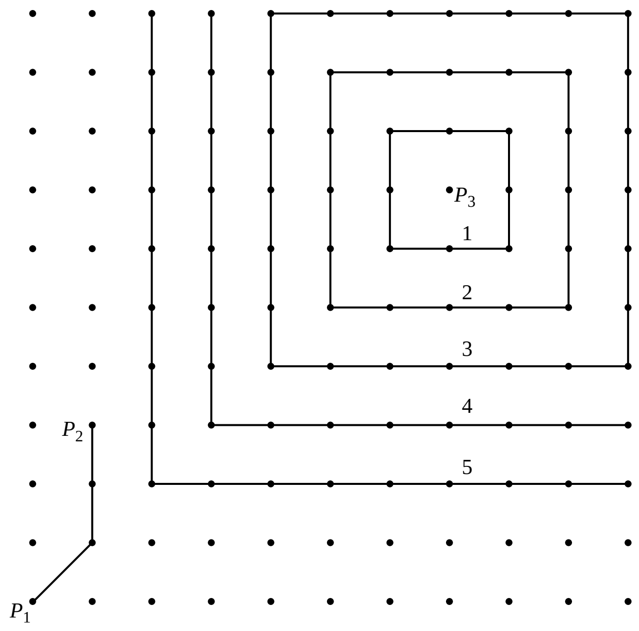


**Figure 4.**
Threat heuristic in 2D map

square at least once. In order to ensure that the threat heuristic is a guaranteed underestimate, we use the minimum threat penalty in the square as the threat penalty of the point that the vehicle visits. The threat heuristic is then given by:

$$h_t = \frac{r_1(T_c + T_{n,\min})}{2} + \frac{\sum r_1(T_{i,\min} + T_{i-1,\min})}{2} \tag{9}$$

where, $T_c$ is the threat penalty of the current point; $T_{i,\min}$ is the minimum threat penalty in square $i$; $n$ is the number of squares between the current point and the target, the sum is over all squares between the current point and the target, and we have multiplied $r_1$ by the smaller of the two step sizes that the vehicle can take, in order to ensure that the threat heuristic is an underestimate.

It is straightforward to extend this 2D threat heuristic to 3D. 3DPLAN uses concentric rectangular boxes around the target. The vehicle will then have to visit each box at least once to go from the current point to the target. For the large boxes, the minimum threat penalty in the box may often be 0, depending on the distribution of threats. However, for small boxes (the ones close to the target), the minimum threat penalty in the box is less likely to be 0, if there is a dense distribution of threats around the target. In these cases, the threat heuristic will significantly speed up the A$^*$ algorithm.

## 4. The implementation and comparison of heuristics

In the implementation, we used two different mobility maps, labeled $M_1$ and $M_2$, and two different threat maps, labeled $T_1$ and $T_2$. In the mobility map $M_1$, all of the grid points in the GO areas have a mobility penalty of 1 (i.e. the mobility is uniform except for the obstacles), while in mobility map $M_2$, the points in the GO areas have a mobility penalty of 1, 2, 3, or 4. The threat maps $T_1$ and $T_2$ contain 24 and 26 threats, respectively. All four of these maps have $237 \times 224 \times 150$ grid points for a total of almost eight million points, which is the largest number of grid points that has ever been used in an A$^*$ search. We used four different combinations of these maps: $M_1T_1$, $M_1T_2$, $M_2T_1$, and $M_2T_2$. For each of these combinations of a mobility map and a threat map, we calculated three different paths by choosing three different pairs of start/target points, which are given in Table I. Thus, altogether, we calculated 12 different optimum paths in a search space of approximately 8 million grid points. In all of the path calculations, the mobility weight $\alpha_m$ and threat weight $\alpha_t$ were both set equal to 1. For each of the 12 optimum paths, we did four different calculations using the following four combinations of the mobility and threat heuristics:

(1) our new mobility heuristic, our new threat heuristic;

(2) our new mobility heuristic, no threat heuristic;

(3) straight line mobility heuristic, our new threat heuristic; and

(4) straight line mobility heuristic, no threat heuristic.

**Table I.**
Start point and target of three paths

| Paths | Start point | Target |
|---|---|---|
| Path 1 | (6, 22, 0) | (236, 218, 149) |
| Path 2 | (6, 22, 0) | (136, 218, 89) |
| Path 3 | (100, 0, 50) | (236, 218, 149) |

Combinations 1-3 enable us to compare our new mobility and threat heuristics with the heuristic that other authors have used in A* searches (Combination 4).

These optimum path calculations were implemented on a Dell PC with a 3.06 GHz Pentium IV processor. One of these optimum paths, Path 1 for the map combination $T_1M_2$, is shown in Figure 5. The spheres in this figure are the inner spheres surrounding the threats, the rectangular solids are the obstacles in the mobility map, and the optimum path goes from the lower left to the upper right.

Tables II-V give the CPU time in seconds for the optimum path calculations. These tables show that when our new mobility and threat heuristics are used, all of the path calculations require less than one-and-a-half-minutes of CPU time. The tables also show

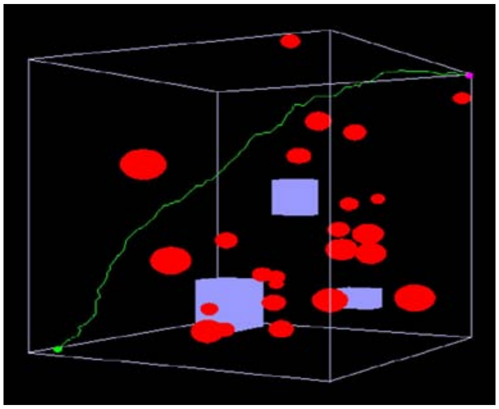


**Figure 5.** Path 1 for the map combination $T_1M_2$

| | Path 1 | Path 2 | Path 3 |
|---|---|---|---|
| Our mobility heuristic, and our threat heuristic | 27.718 | 9.469 | 11.718 |
| Our mobility heuristic, and no threat heuristic | 37.219 | 13.124 | 15.140 |
| Straight line mobility heuristic, and our threat heuristic | 51.343 | 22.64 | 26.281 |
| Straight line mobility heuristic, and no threat heuristic | 58.589 | 26.156 | 30.469 |

**Table II.** Map $T_1M_1$

| | Path 1 | Path 2 | Path 3 |
|---|---|---|---|
| Our mobility heuristic, and our threat heuristic | 62.438 | 11.843 | 20.327 |
| Our mobility heuristic, and no threat heuristic | 68.125 | 14.141 | 24.140 |
| Straight line mobility heuristic, and our threat heuristic | 80.984 | 22.313 | 35.359 |
| Straight line mobility heuristic, and no threat heuristic | 85.656 | 25.343 | 40.344 |

**Table III.** Map $T_1M_2$

| | Path 1 | Path 2 | Path 3 |
|---|---|---|---|
| Our mobility heuristic, and our threat heuristic | 8.391 | 30.343 | 15.828 |
| Our mobility heuristic, and no threat heuristic | 11.062 | 35.219 | 18.672 |
| Straight line mobility heuristic, and our threat heuristic | 28.828 | 41.039 | 34.094 |
| Straight line mobility heuristic, and no threat heuristic | 33.390 | 47.031 | 38.250 |

**Table IV.** Map $T_2M_1$

that for the 12 test paths considered, our new heuristics reduce the CPU time by up to 67 percent compared to a straight line mobility heuristic and no threat heuristic. In addition, the tables show that our new mobility heuristic alone reduces the CPU time by up to 60 percent and our new threat heuristic alone reduces the CPU time by up to 20 percent, compared to straight line mobility heuristic and no threat heuristic, respectively. These results demonstrate that our new mobility and threat heuristics significantly speed up the A* algorithm.

## 5. Application to autonomous underwater vehicles

In the AUV application, we replace the test mobility map described above with a map of realistic travel times which depend on ocean currents and the still water speed of the AUV. In the test calculations described in the previous section, the path cost depends on mobility penalties (to which the travel time is roughly proportional), but in the AUV application in this section, the path cost is equal to the actual travel time rather than just being approximately proportional to it.

In order to construct the map of travel times, we first created an ocean current map using the current velocity data from the Navy Coastal Ocean Model – East Asian Seas version, which was provided by the Naval Research Laboratory at the Stennis Space Center in Mississippi. In this 3D ocean current map, there are $229 \times 128 \times 35$ grid points in the $x$, $y$ and $z$-directions for a total of more than 1 million grid points. The $x$ coordinate goes from longitude 115° E to 135° E in steps of 11.4 min of arc, the $y$ coordinate goes from latitude 20° N to 30° N in steps of 11.4 min, and the $z$ coordinate goes from 0 (the ocean surface) to a maximum depth of 4,655 m.

At each grid point, the map gives the two components of the current velocity: the $U$ velocity, which is in the $x$ (East/West) direction, and the $V$ velocity, which is in the $y$ (North/South) direction, where East and North are positive and West and South are negative, and the velocities are in m/s. There is no current velocity in the $z$-direction.

The 3D ocean current map described above is used to construct the 3D travel time map, which gives the times in seconds required for the AUV to go from each grid point to its 26 neighbors, taking into account the ocean currents and the still water speed of the AUV. Thus, this map has 26 real numbers at each grid point. Figure 6 shows a grid point, labeled $Y$, and its 26 neighbors labeled $A$-$X$, above, and below.

The grid point $Y$ is the central grid point and the points $A$-$X$ are referred to as follows:

- A – Southwest above, B – Southwest same level, C – Southwest below;
- D – South above, E – South same level, F – South below;
- G – Southeast above, H – Southeast same level, I – Southeast below;
- J – West above, K – West same level, L – West below;
- M – East above, N – East same level, O – East below;

**Table V.**
Map $T_2M_2$

| | Path 1 | Path 2 | Path 3 |
|---|---|---|---|
| Our mobility heuristic, and our threat heuristic | 56.813 | 38.109 | 38.844 |
| Our mobility heuristic, and no threat heuristic | 66.406 | 42.328 | 41.594 |
| Straight line mobility heuristic, and our threat heuristic | 75.687 | 46.813 | 55.656 |
| Straight line mobility heuristic, and no threat heuristic | 79.250 | 52.187 | 58.968 |

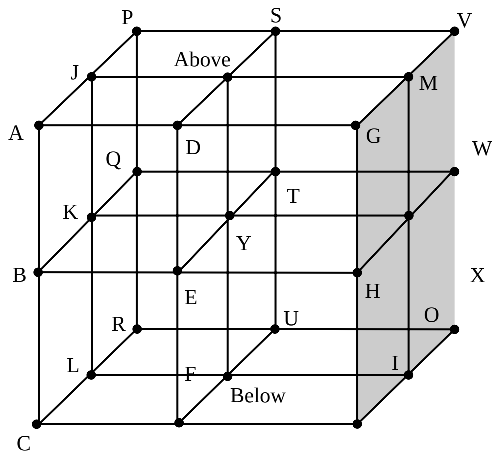


**Figure 6.**
A grid point with 26 neighbors

- P – Northwest above, Q – Northwest same level, R – Northwest below;
- S – North above, T – North same level, U – North below; and
- V – Northeast above, W – Northeast same level, X – Northeast below.

In this figure, the scale in the *z*-direction is different from the scale in the *x*- and *y*-directions so that the grid can conveniently be displayed. In order to calculate the travel times to the N, S, E, W, NE, NW, SE, and SW neighbors in the planes above and below, we use only the component of the distance in the *x*-*y* plane, because the AUV can move up or down by simply changing its ballast and thus vertical motion that occurs at the same time as horizontal motion does not require any additional travel time. This assumption is valid as long as the vertical spacing between the grid points is substantially smaller than the horizontal spacing, which is the case here. Hence, the difference in travel time to a neighbor in the plane above and the corresponding neighbor in the same plane (e.g. NE above and NE same level) are due only to the difference between the *U* and *V* velocities at the two neighbors and not to their distances from the central point.

In Figure 7, $P_1$ is the central point, $P_2$ is the NE same level neighbor, $d_1$ is the grid point spacing for the latitude of $P_1$, and $d_2$ is the grid point spacing for the latitude of $P_2$.

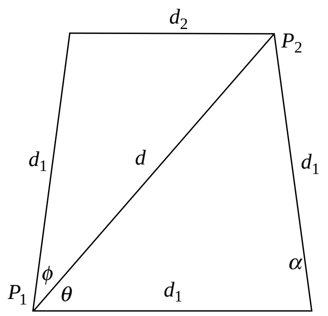


**Figure 7.**
Calculation of the travel time to the NE neighbor from $P_1$ to $P_2$

The grid point spacing for a given latitude gives the distance to neighboring points in the East, West, and North directions, while the distance to the South neighbor is given by the grid point spacing for the South neighbor's latitude, which is slightly larger. In other words, the spacing between the grid points slowly decreases as the latitude increases, which reflects the fact that the distance between latitude and longitude circles on the earth's surface gets smaller as the latitude gets larger. The angles $\alpha$, $\theta$, $\phi$, and the distance $d$ between the central point $P_1$ and the neighbor $P_2$ are given by:

$$\alpha = \cos^{-1}\left(\frac{d_1 - d_2}{2d_1}\right) \tag{10}$$

$$\theta = \cos^{-1}\left(\sqrt{\frac{d_1 + d_2}{4d_1}}\right) \tag{11}$$

$$\phi = \alpha - \theta \tag{12}$$

$$d = \sqrt{(d_1)^2 + d_1 d_2} \tag{13}$$

The travel time $t$ to go from $P_1$ to $P_2$ is then calculated as follows:

$$U_\mathrm{a} = \frac{U_1 + U_2}{2} \tag{14}$$

$$V_\mathrm{a} = \frac{V_1 + V_2}{2} \tag{15}$$

$$C_{12} = U_\mathrm{a}\cos(\theta) + V_\mathrm{a}\cos(\phi) \tag{16}$$

$$C_\mathrm{p} = U_\mathrm{a}\sin(\theta) - V_\mathrm{a}\sin(\phi) \tag{17}$$

$$S_{12} = \sqrt{S^2 - (C_\mathrm{p})^2} \tag{18}$$

$$V_\mathrm{N} = S_{12} + C_{12} \tag{19}$$

$$t = \frac{d}{V_\mathrm{N}} \tag{20}$$

where, $U_1$, $U$ velocity at $P_1$; $V_1$, $V$ velocity at $P_1$; $U_2$, $U$ velocity at $P_2$; $V_2$, $V$ velocity at $P_2$; $U_\mathrm{a}$, average of $U$ velocities at $P_1$ and $P_2$; $V_\mathrm{a}$, average of $V$ velocities at $P_1$ and $P_2$; $C_{12}$, component of current velocity along the line joining $P_1$ and $P_2$; $C_\mathrm{p}$, component of current velocity perpendicular to the line joining $P_1$ and $P_2$; $S$, AUV's still water horizontal velocity; $S_{12}$, component of AUV's still water horizontal velocity along the line joining $P_1$ and $P_2$; $V_\mathrm{N}$, net velocity of AUV (the net velocity is along the line joining $P_1$ and $P_2$).

In Figure 8, $P_1$ is the central point, $P_2$ is the E same level neighbor, and $d_1$, the grid point spacing for the latitude of $P_1$, is the distance between $P_1$ and $P_2$. The travel time $t$ to go from $P_1$ to $P_2$ is calculated as follows:

$$U_a = \frac{U_1 + U_2}{2} \quad (21)$$

$$V_a = \frac{V_1 + V_2}{2} \quad (22)$$

$$S_{12} = \sqrt{S^2 - (V_a)^2} \quad (23)$$

$$V_N = S_{12} + U_a \quad (24)$$

$$t = \frac{d_1}{V_N} \quad (25)$$

*5.1 Above and below neighbors*
The travel time to go from the central point to the neighbor directly above or below is given by:

$$t = \frac{d}{S_v} \quad (26)$$

where $d$ is the distance to the neighbor and $S_v$ is the velocity of the AUV in the vertical direction (due to changing the ballast).

*5.2 A * equations*
When the travel time map, rather than the mobility map, is used, equations (2), (3), and (5) become:

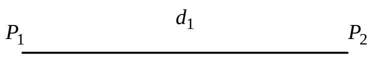
$$g = \alpha_{tt} TT + \alpha_t T \quad (27)$$

$$TT = \sum TT_i \quad (28)$$

$$h = \alpha_{tt} h_{tt} + \alpha_t h_t \quad (29)$$

where, $TT$, accumulated travel time; $\alpha_{tt}$, travel time weight; $TT_i$, travel time to go from grid point $i-1$ to grid point $i$; $h_{tt}$, travel time heuristic (underestimate of remaining travel time needed to reach the target).

The travel time heuristic $h_{tt}$ is given by an expression similar to the one in equation (8):



**Note:** Calculation of the travel time to east neighbor from $P_1$ to $P_2$

**Figure 8.**

$$h_{\mathrm{tt}} = \begin{cases} n_x{*}t_5 + (n_y - n_x){*}t_4 + (n_z - n_y){*}t_3 & \text{for } n_x \leq n_y \leq n_z \\ n_x{*}t_5 + (n_z - n_x){*}t_4 + (n_y - n_z){*}t_1 & \text{for } n_x \leq n_z \leq n_y \\ n_y{*}t_5 + (n_x - n_y){*}t_4 + (n_z - n_x){*}t_3 & \text{for } n_y \leq n_x \leq n_z \\ n_y{*}t_5 + (n_z - n_y){*}t_4 + (n_x - n_z){*}t_1 & \text{for } n_y \leq n_z \leq n_x \\ n_z{*}t_5 + (n_x - n_z){*}t_2 + (n_y - n_x){*}t_1 & \text{for } n_z \leq n_x \leq n_y \\ n_z{*}t_5 + (n_y - n_z){*}t_2 + (n_x - n_y){*}t_1 & \text{for } n_z \leq n_y \leq n_x \end{cases} \quad (30)$$

where, $t_1$, smallest travel time to go from a central point to an E, W, N, or S neighbor in the same plane; $t_2$, smallest travel time to go from a central point to a NE, NW, SE, or SW neighbor in the same plane; $t_3$, smallest travel time to go from a central point to a neighbor directly above or below; $t_4$, smallest travel time to go from a central point to an E, W, N, or S neighbor in the plane above or below; $t_5$, smallest travel time to go from a central point to a NE, NW, SE, or SW neighbor in the plane above or below.

### *5.3 Threats*

Threats are handled the same way as previously described, where each threat is modeled by an inner sphere called the NO-GO sphere where the AUV is not allowed to go, and an outer sphere called the penalty sphere where the AUV is within range of the threat and hence incurs a threat penalty. Thus, if the line between a central point and a neighbor passes through the no-go sphere of a threat, the AUV is not allowed to travel from the central point to that neighbor. If the threat is an underwater mine, it has a no-go sphere but no penalty sphere.

### *5.4 Sample path calculations*

We used Microsoft Visual C++6.0 to develop 3DPLAN, the travel time map, and the AUV version of 3DPLAN, which will be called AUVPLAN. We then used AUVPLAN and the travel time map to perform optimum path calculations in a region of the East China Sea that contains Taiwan and neighboring islands. Figures 9-15 show the results of these optimum path calculations, which were done on a Dell PC with a 1.7 GHZ Pentium IV processor. All of the path calculations required less than 2 min of CPU time. Note that the optimum paths move up and down in order to find the most favorable currents. As discussed above, the use of the A$^*$ algorithm guarantees that the calculated paths have the shortest possible travel times. This cannot be guaranteed when other route planning algorithms, such as the GA, are used.

### *5.5 Avoidance of underwater mines*

If there are underwater mines in the region between the start point and the target, AUVPLAN will calculate the fastest path that avoids the NO-GO spheres around the mines. Also, if previously undetected mines are discovered near an optimum path that has already been determined, AUVPLAN will quickly calculate a new path that safely bypasses the mines. As an example, Figure 16 shows an initial optimum path that was calculated in the Persian Gulf, previously unknown mines that lie close to this path, and the new path that maintains a safe distance from these mines. This new path is optimal subject to the constraint of avoiding the mines.

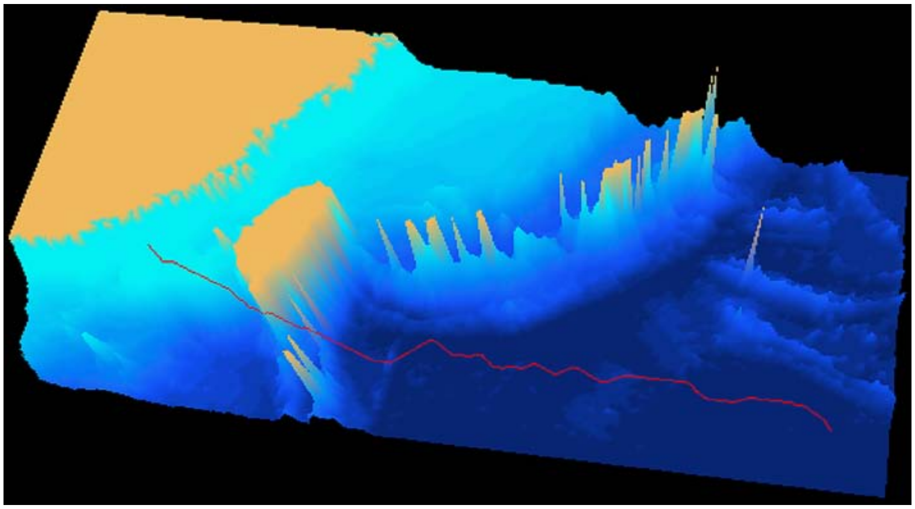


**Figure 9.**
Starting point (33, 40, 4), ending point (208, 8, 13)

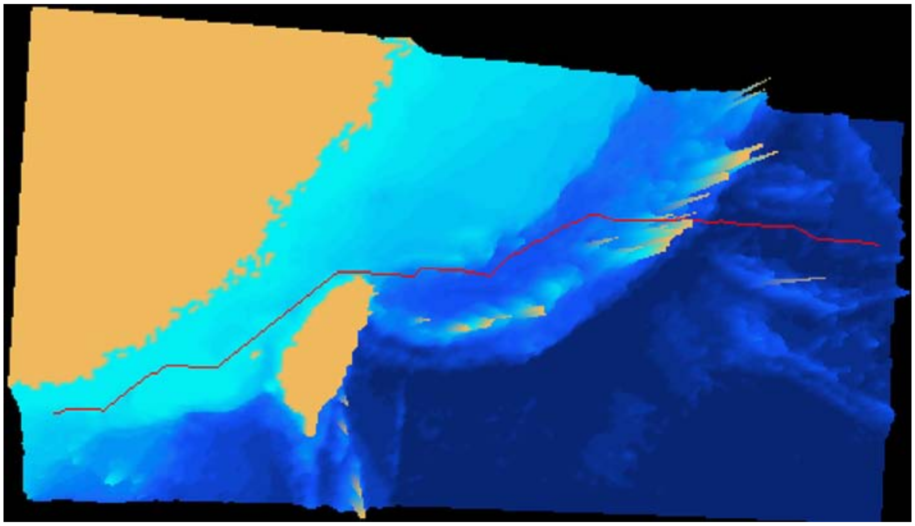


**Figure 10.**
Starting point (9, 28, 3), ending point (201, 83, 6)

## 6. Numerical comparison of A* algorithm and genetic algorithm

To evaluate the performance of A* algorithm with our heuristic, we did the numerical comparison of the route design results with those of GA. We created other smaller size maps with $22 \times 12 \times 16$ and $110 \times 60 \times 25$ dimensions. For Paths 1 and 2, both A* and GA find the optimal paths for all three maps. A* always finds the optimal paths for each of those five paths. GA does not. On average, GA got the optimal path for Path 1 by running the program for twice, for Path 2 by running the program for 20 times. For other Paths 3-5, GA failed to find any paths.

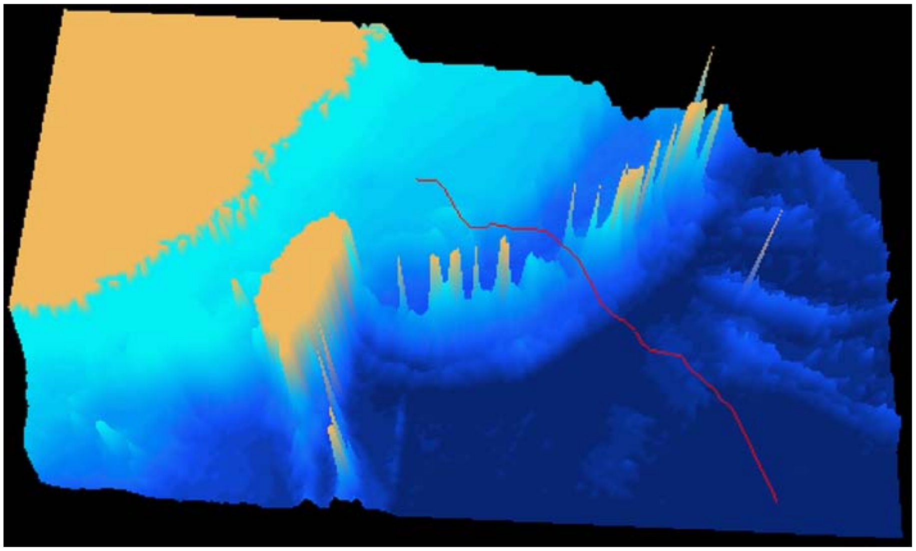


**Figure 11.**
Starting point (93, 86, 8), ending point (189, 2, 21)

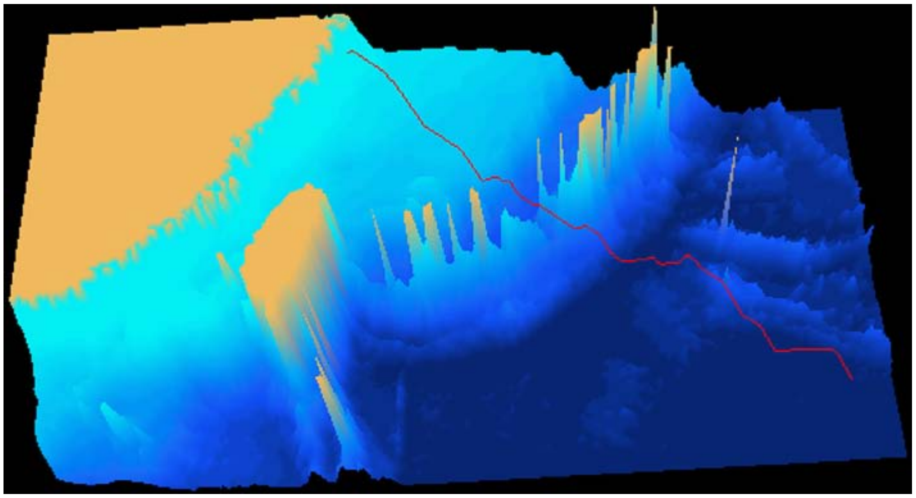


**Figure 12.**
Starting point (83, 116, 3), ending point (205, 2, 7)

*6.1 Small size map (map size is $22 \times 12 \times 16 = 4{,}224$ grid points)*
Tables VI-VIII show the small size map.

*6.2 Medium size map (map size is $110 \times 60 \times 25 = 165{,}000$ grid points)*
Tables IX-XII show the medium size map.

*6.3 Full size map (map size is $22912835 = 1{,}025{,}920$ grid points)*
Tables XIII-XVII show the full size map.

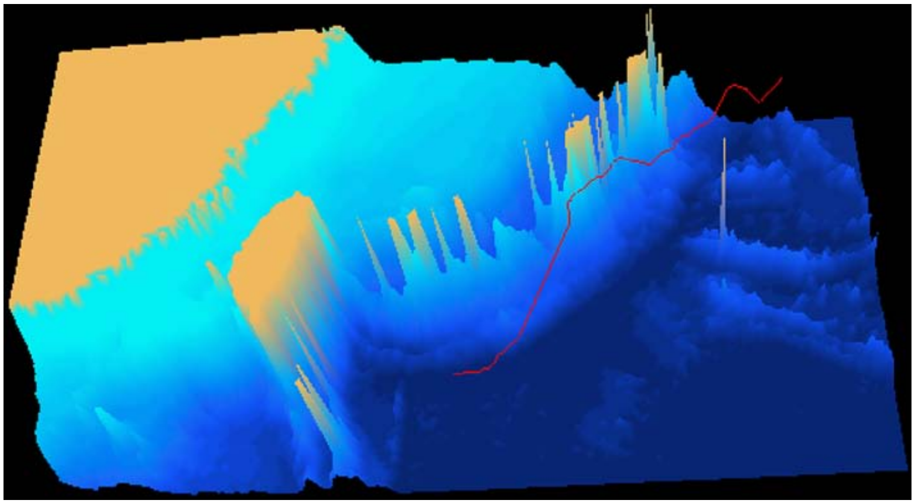


**Figure 13.** Starting point (109, 28, 23), ending point (205, 119, 17)

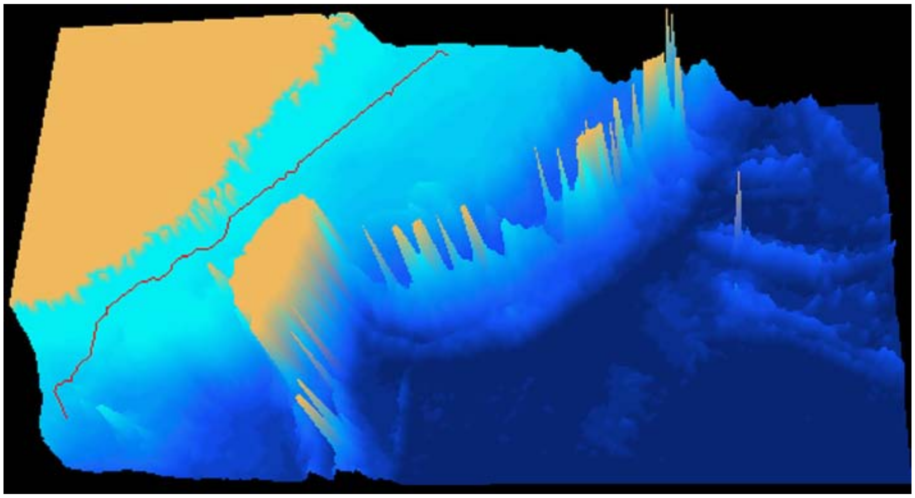


**Figure 14.** Starting point (9, 8, 13), ending point (108, 119, 7)

## 7. Conclusion

We have developed a 3D A$^*$ route planner, called 3DPLAN, which runs efficiently for the large grids that are typical of 3D maps. The A$^*$ algorithm has a major advantage compared to other search methods because it is guaranteed to give the optimum path. To our knowledge, this is the first time that A$^*$ has been used in 3D searches. Most researchers think that the computational cost of using A$^*$ for 3D route planning would be prohibitive. We have shown that on the contrary, it is quite feasible to use A$^*$ for 3D searches as a result of the new mobility and threat heuristics presented in this paper.

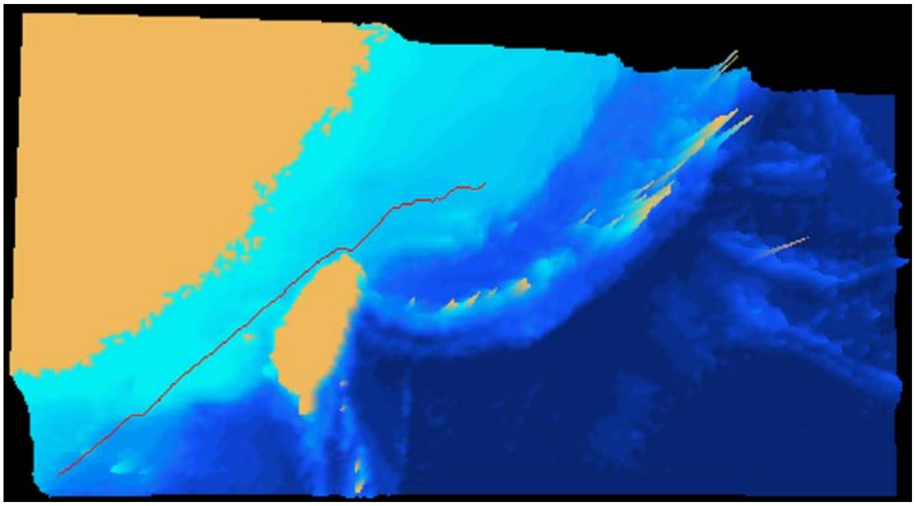


**Figure 15.**
Starting point (9, 8, 6), ending point (108, 88, 9)

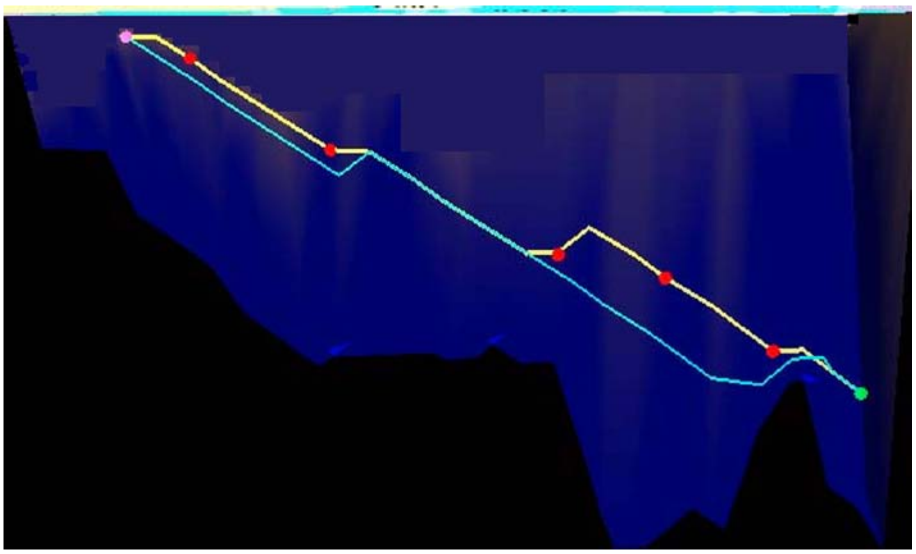


**Figure 16.**
Two paths in Persian Gulf – one path goes close to the mines, while the new path avoids the mines

**Table VI.**
Path 1: starting point (3, 2, 6), ending point (7, 5, 4)

| | Path cost (travel time) | CPU time | Path length |
|---|---|---|---|
| A$^*$ algorithm | 9.97 h | 0.015 s | 5 |
| Genetic algorithm | 9.97 h | 0.344 s | 5 |

These new heuristics substantially speed up the A* algorithm and make it a useful and efficient method for 3D route planning. We have also modified the previous 3DPLAN by replacing the test mobility map with the more realistic mobility map. The new mobility map we have created is based on the impacts of the oceans' dynamics on AUVs. The travel time, we have calculated from each underwater grid point to its neighbors consists of our new mobility map.

| | Path cost (travel time) | CPU time | Path length |
|---|---|---|---|
| A* algorithm | 20.42 h | 0.046 s | 12 |
| Genetic algorithm | 20.42 h | 19.546 s | 12 |

**Table VII.** Path 2: starting point (4, 6, 7), ending point (15, 9, 11)

| | Path cost (travel time) | CPU time | Path length |
|---|---|---|---|
| A* algorithm | 36.66 h | 0.062 s | 22 |
| Genetic algorithm | Failed | N/A | N/A |

**Table VIII.** Path 3: starting point (21, 1, 15), ending point (0, 11, 6)

| | Path cost (travel time) | CPU time | Path length |
|---|---|---|---|
| A* algorithm | 9.97 h | 0.047 s | 5 |
| Genetic algorithm | 9.97 h | 0.610 s | 5 |

**Table IX.** Path 1: starting point (3, 2, 6), ending point (7, 5, 4)

| | Path cost (travel time) | CPU time | Path length |
|---|---|---|---|
| A* algorithm | 20.42 h | 0.079 s | 12 |
| Genetic algorithm | 20.42 h | 45.600 s | 12 |

**Table X.** Path 2: starting point (4, 6, 7), ending point (15, 9, 11)

| | Path cost (travel time) | CPU time | Path length |
|---|---|---|---|
| A* algorithm | 36.66 h | 0.140 s | 22 |
| Genetic algorithm | Failed | N/A | N/A |

**Table XI.** Path 3: starting point (21, 1, 15), ending point (0, 11, 6)

| | Path cost (travel time) | CPU time | Path length |
|---|---|---|---|
| A* algorithm | 197.46 h | 1.719 s | 112 |
| Genetic Algorithm | Failed | N/A | N/A |

**Table XII.** Path 4: starting point (109, 59, 19), ending point (0, 1, 5)

3DPLAN can also handle dynamic (changing) threats. If new threats appear or known threats disappear or move while the vehicle is travelling along its path, the military planner can quickly update the threat map. 3DPLAN will then rapidly generate a new optimum path from the current position to the target.

At present, the threats in our threat map are the static small size threats like under water mines. In the future, we plan to introduce the dynamic larger size threats like enemy's submarines and surface destroyers. In the future, we also plan to test 3DPLAN with real battlefield and oceanographic data and adapt it to combat helicopters and various types of manned and unmanned underwater vehicles.

**Table XIII.**
Path 1: starting point (3, 2, 6), ending point (7, 5, 4)

| | Path cost (travel time) | CPU time | Path length |
|---|---|---|---|
| A* algorithm | 9.97 h | 0.218 s | 5 |
| Genetic algorithm | 9.97 h | 0.579 s | 5 |

**Table XIV.**
Path 2: starting point (4, 6, 7), ending point (15, 9, 11)

| | Path cost (travel time) | CPU time | Path length |
|---|---|---|---|
| A* algorithm | 20.42 h | 0.219 s | 12 |
| Genetic algorithm | 20.42 h | 12.656 s | 12 |

**Table XV.**
Path 3: starting point (21, 1, 15), ending point (0, 11, 6)

| | Path cost (travel time) | CPU time | Path length |
|---|---|---|---|
| A* algorithm | 36.66 h | 0.297 s | 22 |
| Genetic algorithm | Failed | N/A | N/A |

**Table XVI.**
Path 4: starting point (109, 59, 19), ending point (0, 1, 5)

| | Path cost (travel time) | CPU time | Path length |
|---|---|---|---|
| A* algorithm | 197.46 h | 5.016 s | 112 |
| Genetic algorithm | Failed | N/A | N/A |

**Table XVII.**
Path 5: starting point (228, 127, 30), ending point (0, 1, 5)

| | Path cost (travel time) | CPU time | Path length |
|---|---|---|---|
| A* algorithm | 383.55 h | 13.359 s | 229 |
| Genetic algorithm | Failed | N/A | N/A |

## References

Benton, J.R., Iyengar, S.S., Deng, W., Brener, N.E. and Subrahmanian, V.S. (1996), "Tactical route planning: new algorithms for decomposing the map", *International Journal on Artificial Intelligence Tools*, Vol. 5 Nos 1/2, pp. 199-218.

Brener, N., Iyengar, S. and Benton, J. (2004), Predictive Intelligence Military Tactical Analysis System (PIMTAS), Louisiana State University and US Army Topographic Engineering Center, Alexandria, VA.

Hart, P., Nilsson, N. and Raphael, B. (1968), "A formal basis for the heuristic determination of minimum cost paths", *IEEE Transactions of Systems Science and Cybernetics*, Vol. 4, pp. 100-7.

Kruusmaa, M. and Svensson, B. (1998), "Combined map-based and case-based path planning for mobile robot navigation", *Proceedings of International Symposium on Intelligent Robotic Systems, January*, pp. 10-12.

McVey, C., Clements, D., Massey, B. and Parkes, A. (1999), *Worldwide Aeronautical Route Planner*, American Association for Artificial Intelligence, Menlo Park, CA.

Nishi, T., Ando, M. and Konishi, M. (2005), "Distributed route planning for multiple mobile robots using an augmented Lagrangian decomposition and coordination technique", *IEEE Transactions on Robotics*, Vol. 21 No. 6, pp. 1191-200.

Smith, J. and Sugihara, K. (1996), "GA toolkit on the web", *Proceedings of the 1st Online Workshop on Soft Computing, August*, pp. 93-8.

Sugihara, K. and Yuh, J. (1997), "GA-based motion planning for underwater robotic vehicles", *Proceedings of the 10th International Symposium on Unmanned Untethered Submersible Technology*, Autonomous Undersea Systems Institute, Durham, NH, pp. 406-15.

Szczerba, R., Galkowski, P., Glicktein, I. and Ternullo, N. (2000), "Robust algorithm for real-time route planning", *IEEE Transactions on Aerospace and Electronic Systems*, Part 1, Vol. 36 No. 3, pp. 869-78.

Vasudevan, C. and Ganesan, K. (1996), "Cased-based path planning for autonomous underwater vehicles", *Autonomous Robots*, Vol. 3 Nos 2/3, pp. 79-89.

Zheng, C. (2008), "Evolutionary route planner for unmanned air vehicles", *Proceedings of the Annual 10th Conference on Genetic and Evolutionary Computation, Atlanta, GA*, pp. 1477-84.

Zheng, C., Li, L., Xu, F., Sun, F. and Ding, M. (2005), "Evolutionary path planner for UAVs in realistic environments", *IEEE Transactions on Robotics*, Vol. 21 No. 4, pp. 609-20.

## About the authors

Hua Cao received the BE degree of Management Information Systems from University of Finance and Economics, China in 2000, the MS degree of Systems Science from Louisiana State University, USA in 2003, and the PhD degree of Computer Science from Louisiana State University, USA in 2008. She is a Senior Research Scientist in the Computer Science Department and Ophthalmology Department at Louisiana State University, USA. Her current research interests include route planning, biomedical imaging, feature detection, data registration, image fusion, and artificial intelligence. Hua Cao is the corresponding author and can be contacted at: hcao@csc.lsu.edu

Nathan E. Brener received the BA degree of Physics from Brandeis University in 1965 and the PhD degree of Physics from Louisiana State University in 1971. He is a Faculty Member of Computer Science Department, Louisiana State University. He has approximately 35 years experience in the development of fast algorithms for high-performance computing. Since 1983, he has conducted research on applications of artificial and predictive intelligence to military analysis and planning systems. He has developed a software package, called the Combat Helicopter Expert Navigation Assistant. He later developed, in collaboration with Dr Iyengar and John Benton of the US Army Topographic Engineering Center, the PIMTAS, which is a military route planning system applicable to terrain vehicles. He has more than 50 research publications in refereed journals and has made numerous presentations at scientific meetings. His research interests include predictive intelligence, artificial intelligence, route planning algorithms, and parallel processing. His personal homepage is: www.csc.lsu.edu/faculty/people _brener.html

S. Sitharama Iyengar received the MS degree from the Indian Institute of Science, Bangalore, in 1970 and the PhD degree from Mississippi State University in 1974. He is the Chairman and Roy Paul Daniels Chaired Professor of Computer Science at Louisiana State University, Baton Rouge, and is also the Satish Dhawan Chaired Professor at the Indian Institute of Science. His publications include 13 books (Prentice-Hall, CRC Press, IEEE Computer Society Press, Wiley, etc.) and more than 280 research papers. He is the founder and Editor-in-chief of the *International Journal of Distributed Sensor Networks*. He has been involved with research in high-performance algorithms, data structures, sensor fusion, data mining, and intelligent systems. He was awarded the Distinguished Alumnus Award by the Indian Institute of Science in March 2003. He has served as an Associate Editor for the IEEE and as a Guest Editor for the IEEE Transactions on Knowledge and Data Engineering, the IEEE Transactions on Systems, Man, and Cybernetics, and the IEEE Transactions on Software Engineering. He is a fellow of the IEEE, the ACM, and the AAAS. His personal homepage is: http://csc.lsu.edu/~iyengar