Path Planning for Mine Countermeasures

Cheryl Resch[*a], Christine Piatko[a], Fernando J. Pineda[b], Jessica Pistole[a], I-Jeng Wang[a]

[a]Johns Hopkins University Applied Physics Laboratory, Laurel, Maryland 20723-6099

[b]Johns Hopkins Bloomberg School of Public Health, Baltimore, Maryland 20205-2179

ABSTRACT

We have developed path-planning techniques and tools to look for paths through minefields. Our techniques seek to balance the length and risk associated with different routes through a minefield. Our methods are intended to provide battlegroup commanders powerful new tools to evaluate alternative routes while searching for low risk paths. It is well known how to find the path of shortest distance, and well known how to find the path of least risk. However, to optimize both criteria at once is a challenging problem. We have developed and compared two methods for multi-criteria path planning. We have found that the better method uses a linear combination of the criteria with a user selected risk tolerance parameter. We also describe the novel use of dynamic graph algorithms to quickly find new paths as the risk is changed due to the neutralization (deletion) of mines and the discovery (insertion) of mines. We will also describe an algorithm used to convert paths to a set of straight-line paths for ship navigation. Our tool allows a commander to find a path of acceptable length and risk, to explore the effect of eliminating mines, and to obtain a set of waypoints for minefield navigation.

1. INTRODUCTION

Minefield data are probabilistic due to many factors such as position uncertainty, detection probabilities, classification confidence levels, and classification errors. Nonetheless, in the face of this uncertainty, a commander needs a meaningful way to measure risk in order to assess the best way to cross a potential minefield, or decide that the risk to cross is too great. Path planning is a well-studied problem in the setting where there is a single criterion to optimize. In this paper we describe two methods to find optimal paths based on multiple criteria. Instead of a single path quality of length, we want to include other optimization criteria that describe the desirability of paths such as level of risk. In addition, it is desirable for command and control to have alternative routes presented to help make a decision. Therefore, besides a single optimal path we would like to find several choices of paths to display as alternatives. The tool we describe in this paper allows the user to view several paths.

2. RISK MODEL

Here we describe the model for calculating the risk of detonating a mine at any location. The approximate locations of mines are obtained from sensors plus the detection and classification algorithms. The risk at a given location contributed by a single mine depends on the approximate location of the mine, and the model for the uncertainty that the approximate location is correct. Let **y'** be the unknown location of a mine and **y** be the measurement (output) from the sensor plus the detection and classification algorithms. Assume that **y** is normally distributed with the following density:

$$p(y \mid y') = \frac{1}{\sqrt{2\pi \det(\Sigma_s)}} \exp[-(y - y')^T \Sigma_s^{-1} (y - y')]$$

Σ_s is the covariance matrix of the difference between **y** and **y'**.

[*] cheryl.resch@jhuapl.edu

The risk also depends on the triggering model for the mine; this is the model for the area around the mine that will cause a mine to detonate. Let **x** be the position of the object attempting to detect or avoid mines. Let the random event E denote the event that the mine is triggered. Assume the following Gaussian-like triggering model for E:

$$P[E \mid x, y'] = c_m \exp[-(x - y')^T \Sigma_m^{-1} (x - y')]$$

c_m is the confidence that the mine is there (from sensor plus detection and classification algorithms). Σ_m is the covariance matrix of the triggering model, or the distance between **x** and **y'** required to detonate the mine. We are interested in characterizing the posterior probability that the mine is triggered given the ship position **x** and the sensor measurement **y**. This can be derived via application of Bayes rule:

$$P[E \mid x, y] = \frac{\int_{y'} P[E \mid x, y'] p(x \mid y, y') p(y \mid y') p(y') dy'}{\sum_E \int_{y'} P[E \mid x, y, y'] p(x \mid y, y') p(y \mid y') p(y') dy'}$$

Assume that mine location is uniformly distributed, so that p(**y'**) is a constant, and that the position of the ship is independent of the mine location. Then we obtain the following expression:

$$P[E \mid x, y] = \int_{y'} P[E \mid x, y'] p(y \mid y') dy'$$

Thus, the posterior probability of interest is the convolution of the two distributions. This results in the following expression for the posterior probability of triggering a mine given a mine location and a ship location:

$$P[E \mid x, y] = \frac{c_m \sqrt{2\pi \det(\Sigma_m)}}{\sqrt{2\pi \det(\Sigma_s + \Sigma_m)}} \exp[-(x - y)^T (\Sigma_s + \Sigma_m)^{-1} (x - y)]$$

We define the point risk at **x** contributed by a single mine using the following equation:

$$r(x, y) \equiv \log \frac{1}{1 - P[E \mid x, y]}$$

Next, we obtain the point risk contributed by multiple mines located at **y'$_1$, y'$_2$, y'$_3$,…., y'$_m$** with associated sensor measurements **y$_1$, y$_2$, y$_3$,…., y$_m$**. Assume that the mines are independent of one another and the sensor measurements are independent given the mine locations. The probability that any of the mines are triggered is the following:

$$P[E \mid x, y_1, y_2, ..., y_m] = 1 - \prod_{j=1}^{m} [1 - P[E \mid x, y_j]]$$

The point risk at **x**, r(**x**) is defined by the following expression:

$$r(x) \equiv \log \frac{1}{1 - P[E \mid x, y_1, y_2, ..., y_m]} = \log \frac{1}{\prod_{j=1}^{m} [1 - P[E \mid x, y_j]]} = \sum_{j=1}^{m} \log \frac{1}{1 - P[E \mid x, y_j]} = \sum_{j=1}^{m} r(x, y_j)$$ The point risk resulting from multiple mines is exactly the sum of the point risks contributed by each individual mine.

Next, we will expand our point risk model to calculate the risk along a path. Consider a discrete path *Pa* defined by the points on the path, that is:

$Pa = x_1 x_2 x_3 \ldots x_n$

The probability of triggering any mine while traveling over the path is:

$$P[E \mid Pa, Y] = 1 - \prod_{i=1}^{n} [1 - P[E \mid x_i, Y]]$$

where we use *Y* to denote all the sensor measurements of mine locations. We define the risk associated with the path *Pa* by:

$$r(Pa) \equiv \log \frac{1}{1 - P[E \mid Pa, Y]} = \log \frac{1}{\prod_{i=1}^{n} [1 - P[E \mid x_i, Y]]} = \sum_{i=1}^{n} \log \frac{1}{1 - P[E \mid x_i, Y]} = \sum_{i=1}^{n} r(x_i)$$

Thus, the risk of triggering a mine over an entire path is simply the sum of the point risk values along the path.

3. GRID GRAPH MODEL FOR PATH PLANNING

We discretize the minefield into a graph, with vertices evenly distributed over the minefield. Risk values *r(i,j)* are associated with the vertex at *i, j* according to the collection of mines and the risk model. The collection of mines is defined by a location **y** and confidence c_m for each mine. Location **y** is converted to the closest gridpoint location *i, j*. The triggering distance is converted into the number of gridpoints in each direction *Δi* and *Δj*. For every vertex, the risk contributed by every mine in the collection that is within *Δi* and *Δj* is calculated and summed. The risk due to the mines in the collection at location *i ,j* is thus calculated using the following equation:

$$r_{\text{mines}}(i, j) = \sum_{a=i-\Delta i}^{i+\Delta i} \sum_{b=j-\Delta j}^{j+\Delta j} \frac{c_{m,a,b} \sqrt{2\pi \det(\Sigma_m)}}{\sqrt{2\pi \det(\Sigma_s + \Sigma_m)}} \exp[(\Sigma_s + \Sigma_m)^{-1} * \sqrt{(i-a)^2 + (j-b)^2})]$$

$c_{m,a,b}$ is the confidence for the mine at location *a, b*. This value is zero if there is no mine there. At every vertex, there is also a possibility of an undetected mine. The confidence that there is an undetected mine at any vertex is calculated as $c_{\text{undetected}}$ = (number of undetected mines)/(number of vertices), where the number of undetected mines is found using the following expression.

$$N_{\text{undetected}} = \frac{N_{\text{mines}}}{P_D} - N_{\text{mines}}$$

P_D is the probability of detection, and N_{mines} is the number of detected mines. A background risk is calculated for every vertex. It is the sum of the risk at every vertex in the triggering area surrounding that vertex. The following expression describes the calculation of the background risk:

$$r_{background} = \sum_{a=-\Delta i}^{\Delta i} \sum_{b=-\Delta j}^{\Delta j} \frac{c_{\text{undetected}} \sqrt{2\pi \det(\Sigma_m)}}{\sqrt{2\pi \det(\Sigma_s + \Sigma_m)}} \exp[(\Sigma_s + \Sigma_m)^{-1} * \sqrt{a^2 + b^2})]$$

Thus, $r(i,j) = r_{\text{background}} + r_{\text{mines}}(i,j)$.

Vertices are connected to each other by edges. The edges are directional, and are defined by the "from" vertex, the "to" vertex, and the weight, or cost of traversing the edge. Eight edges emerge from each vertex. The weight of each edge is calculated as follows:

$W_{\text{edge}} = (1-\alpha)\, r_{\text{to}} + \alpha\, l_{\text{edge}}$

r_{to} is the risk value at the "to" vertex, and l_{edge} is the distance from the "from" vertex to the "to" vertex. This distance is $\sqrt{2}$ for diagonal edges, and 1 otherwise. The parameter α is set by the user, and is a risk tolerance factor. Setting α between zero and one corresponds to optimizing over a linear combination over the two criteria of length and risk. This approach makes the multicriteria path planning problem tractable by reducing it to a single metric problem. Setting α to one corresponds to searching for the shortest distance path. Combining this with a maximum bound on the risk results in a path that minimizes the length of the path for which each vertex has a risk value below the bound. This is a second approach to making the multicriteria path planning problem tractable. We will explore both approaches below.

Given a starting location *s* and a desired goal *g*, we find the path that minimizes the following parameter:

$$O(P) = \sum_{edge \in P} W_{edge}$$

For fixed α, this is computed optimally in polynomial time using Dijkstra's algorithm[1,2]. Figure 1 shows a sample minefield with a path from start to goal calculated.

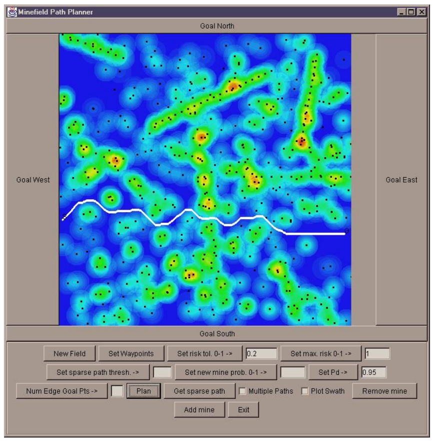


Figure 1 - A Minefield with the Optimum Path from s to g Determined

4. DYNAMIC GRAPH UPDATES

We have implemented a dynamic graph update algorithm to recalculate the shortest paths when only part of the minefield is changed. When a mine is added or removed, the weight of each edge in the affected area of the risk field is recalculated. The "to" vertex of each edge whose weight has changed is put into a set of vertices to be updated. With each of these vertices is saved the old cost of the shortest path from the source to that vertex. Figures A1, A2, and A3 in the appendix show the algorithm, adapted from Ramalingam and Reps[3] used to update the cost to all affected vertices.

This updating algorithm has a running time of O(A logA), where A is the sum of affected vertices and the edges into these vertices. The closer the updated mine is to the source, the larger A is, and the longer running time of the algorithm[4]. We performed a timing study to compare the time to find the shortest paths using Dijkstra's algorithm, the time to update the shortest paths when a mine is removed, and the time to update the shortest paths when a mine is added. We randomly generated 10 different minefields for each of 6 different grid resolutions (250 by 250, 300 by 300, and so on up to 500 by 500). Each minefield contained up to 10 randomly generated minelines, each with 5 to 15 mines. The simulated minefield also contained up to 100 randomly placed "noise" points (with lower mean posterior probabilities than the mines). For each minefield, we alternately removed and added mines; four mines were removed and four were added. We varied the distance to the source of the removed and added mines. This captures the effect of different numbers of affected vertices on the running times. Figure 2 compares using the full Dijkstra's algorithm versus the graph updating algorithm used to add a mine or remove a mine. The solid lines are the mean and the dashed lines above and below the solid lines indicate one standard deviation above and below the mean.

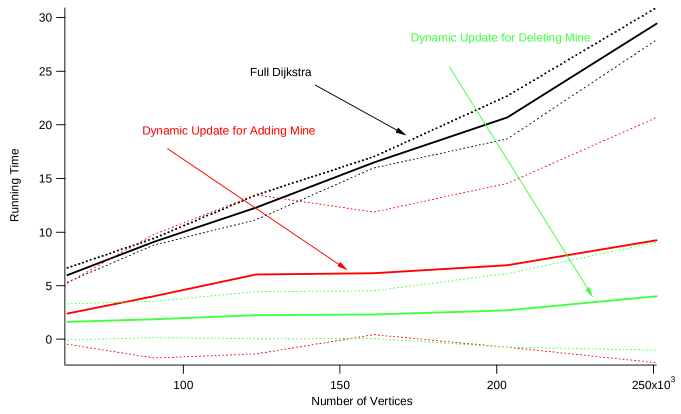
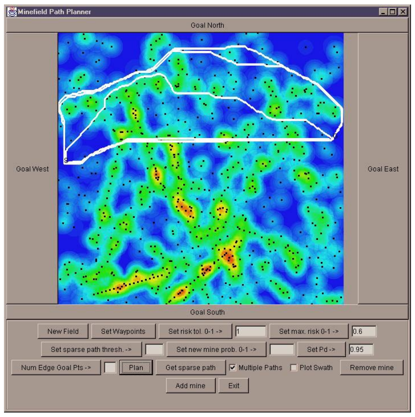


Figure 2 - Timing Comparison for Dijkstra's Algorithm for Entire Graph versus Updating for Adding or Deleting a Mine

Figure 2 shows that the dynamic updates are faster than performing Dijkstra's algorithm on the entire graph. Adding a mine takes more time than deleting a mine. This is because the vertices must be added to the heap, updated, and then added again, as shown in Figures A1-A3. The standard deviation of the time required for dynamic updates is higher than that for running Dijkstra's algorithm. This is because the running time for dynamic update depends on the number of affected vertices, which varies according to how close the added or deleted mine is to the source. The standard deviation of the time required to add a mine is higher than that for deleting a mine. If a mine is added near the source, the time to solve the dynamic update algorithm approaches the time required to perform Dijkstra's algorithm for the entire graph.

5. OPTIMIZING FOR LENGTH AND RISK

As discussed above, we are attempting to solve a multi-criteria optimization problem. We desire a short path, and a path that causes minimum risk. These two criteria work against one another. We have implemented two methods to attempt to balance path length and path risk. First, we weight the edges according to a linear combination of length and risk using the α parameter discussed above. When α is toward zero, length is sacrificed for low risk, and when the value is toward one, risk is sacrificed for a short path. Figure 3 shows the various optimal paths produced when α is varied from 0.01 to 1. As α tends toward a value of 1, the path becomes straighter.

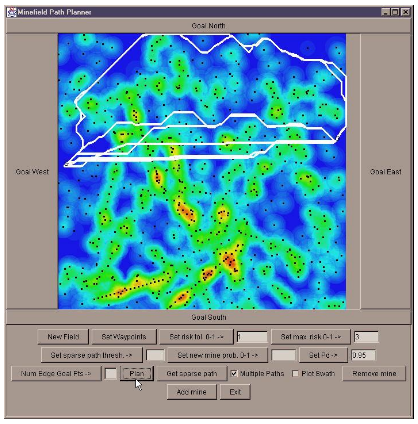


Set risk tol. 0-1 -> 1
Set max. risk 0-1 -> 0.6
Set Pd -> 0.95
✓ Multiple Paths

Figure 3 - Snapshot of Path Planner with Varying Risk Tolerance

Second, we implemented a method where the maximum allowable risk at a single point is set. The shortest path where all points have a risk below the maximum allowable single point risk is found. Figure 4 shows different paths found where the maximum allowable risk is varied. In this figure, as the maximum allowable risk is increased, the path becomes straighter. Curved paths indicate a lower maximum allowable risk. The maximum allowable risk for the minefield shown in Figure 4 could only be lowered to a value of 0.5. If the maximum allowable risk is lower, there are no paths between the waypoints.

Next, we examine which method gives more desirable paths. We compare the maximum point risk along the path to the length of a path. We also compare the total risk along a path to the length of a path. The darker curves in Figure 5 show the comparisons for the experiment where the maximum allowable risk along the path is varied and α has a constant value of 1. The solid dark curve shows that the total risk along the path decreases as the path length increases. The dashed dark curve shows that the actual maximum risk along the path decreases exponentially as the length increases. The lighter curves in Figure 5 show the comparisons for the experiment where the maximum allowable risk along the path is held constant at the largest value in the minefield and the alpha parameter is varied from zero to one. The solid lighter curve shows that the risk total decreases exponentially as length increases. The dashed lighter curve shows that the maximum risk along the path is high for several path lengths (where the alpha parameter is high), then drops down significantly and decreases linearly as length increases. Comparing the two sets of curves, the experiment where α is varied produces significantly lower total path risk values than the experiment where maximum risk along the path is varied. At the cost of higher point risk along the path, the overall path risk is significantly lower. Since the goal is to traverse the entire path without detonating a mine, the overall path risk should be the figure of merit. Varying the alpha parameter produces paths that better reduce the total path risk. Although psychologically a commander may not want to traverse an area with a high risk, doing so will reduce the overall risk. Perhaps it would be best to automatically find inflection points such as that indicated by the circle in Figure 5. This appears to be an optimal combination of low maximum risk along the path, low total risk, and short length.



Set risk tol. 0-1 -> 1
Set max. risk 0-1 -> 3
Set Pd -> 0.95
✓ Multiple Paths

Figure 4 - Snapshot of Path Planner with Varying Maximum Point Risk Values

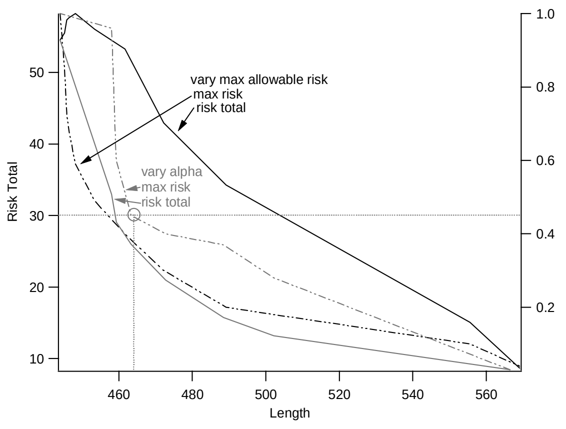
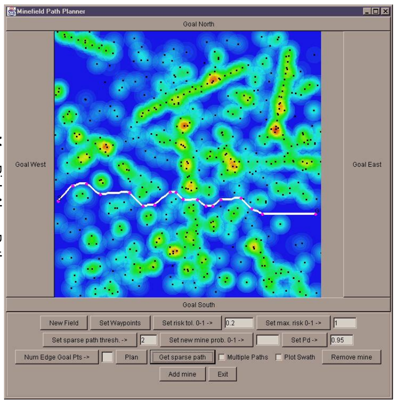


Figure 5 - Comparison of Risk Total and Maximum Risk Along Path When Varying Alpha or Maximum Allowable Risk



Set risk tol. 0-1 -> 0.2
Set max. risk 0-1 -> 1
Set sparse path thresh. -> 2
Set new mine prob. 0-1 ->
Set Pd -> 0.95
Num Edge Goal Pts ->

Figure 6 - Snapshot of Path Planner with Simplified Path

6. PATH SIMPLIFICATION

The graph shown in Figure 1 is an informative look at a path through a minefield. In the end, though, a ship or a marine or a tank will need specific directions on where to go. These directions will include specific locations to head for and direction to travel in. Thus, the path shown in Figure 1 must be reduced to a reasonable number of straight-line segments. We use the algorithm given by Douglas and Peuker[5] to simplify the path into a set of points between which straight-line segments adequately describe the path.

The algorithm draws a straight line between the start and the end of the path. Next, the distance between every vertex on the line is calculated. The vertex with the largest distance to the line is marked and compared to “epsilon”, a tolerance value for how close the simplified line must be to the original line. If the distance is larger than epsilon, the algorithm is called recursively on the line segment between the start and the vertex with the largest distance, and on the line segment between the vertex and the end. If the distance is smaller than epsilon, start and end are added to the list of points used to describe the line. Figure 6 shows a snapshot of the path planning software showing a sparse path.

7. CONCLUSIONS

We have developed and implemented efficient methods for mine path planning problems with multiple constraints. This work has demonstrated the effectiveness of such algorithms for mine path planning. In future work, we will use the Ramalingam and Reps algorithm to create a “risk slider” that uses batches of updates to allow a user to vary the risk threshold interactively. We also seek to develop methods to address the very interesting problem of deciding which mines would be most effective to remove. We would also like to extend this work to include navigability of terrain when finding the optimal path.

REFERENCES

1. Cormen, T. H., Leiserson, C. E., and Rivest, R. L., *Introduction to Algorithms*, McGraw-Hill, New York, 1997.

2. Dijkstra, E. W., "A note on two problems in connexion with graphs," *Numerishe Mathematik*, **1**, 269-271, 1955.

3. Ramalingam, G., and Reps, T., "An Incremental Algorithm for a Generalization of the Shortest-Path Problem," *Journal of Algorithms*, **21**(2), 267-305, 1996.

4. Piatko, C. D., Diehl, C. P., McNamee, J. P., Resch, C. L., and Wang, I., "Stochastic Search and Graph Techniques for MCM Path Planning," *Proceedings of the SPIE*, **4742**, 2002.

5. Douglas, D. H., and Peucker, T. K, "Algorithms for the Reduction of the Number of Points Required to Represent a Line or its Caricature," *The Canadian Cartographer*, **10**(2), 112-122, 1973.

APPENDIX

```
For each edge in area where mine added or deleted
    Change edge weight
    Calculate "new cost" for path to "to" vertex with that edge value changed.
    Update "to" vertex and place on min-heap
        According to "new cost" if edge weight decreased ("over-consistent")
        According to "old cost" if edge weight increased ("under-consistent")
  Process Heap
```

Figure A1 - Dynamic Graph Update Algorithm

```
While heap is not empty
  Pop
  If vertex is "over-consistent" ("new cost"<"old cost")
      From all the edges with this vertex as its "to" vertex, find the edge that produces "new cost" (the
      edge that the shortest path comes through)
      If found
          Set "cost" associated with vertex equal to "new cost"
          Set "previous vertex" associated with vertex to edge's "from" vertex
          Mark the vertex as processed
      If not found
          Find the edge that produces the lowest cost
          If the "from" vertex for this edge has been processed
              Set "cost" to the lowest cost
              Set "previous vertex" to the edge's "from" vertex
              Mark the vertex as processed
          If the "from" vertex for this edge hasn't been processed
              Set "new cost" equal to the lowest cost
              Put vertex back on heap according to "new cost" as "over-consistent"
      Update successor
  If vertex is "under-consistent" ("new cost">"old cost")
      From all the edges with this vertex as its "to" vertex, find the edge that produces the lowest cost to
      the vertex
      Set "new cost" to equal to the lowest cost
      Put the vertex back on the heap according to "new cost" as "over-consistent"
      Update successor
```

Figure A2 - Procedure "process heap" in Dynamic Graph Update Algorithm

```
For all edges with "from" vertex equal to vertex just processed
    Get "to" vertex for edge
    Get "old cost" (current value associated with vertex)
    Get "new cost" = cost to "from" vertex (just processed) plus edge weight
    If ("new cost"<"old cost")
        Put "to" vertex on heap according to "new cost" as "over-consisent"
    If ("new cost">"old cost")
        Find lowest cost to vertex
        If ("old cost"<lowest cost)
            Put "to" vertex on heap according to "old cost" as "under-consistent"
        If ("old cost"=lowest cost)&(shortest path to "to" went through "from" vertex)
            Put "to" vertex on heap according to "old cost" as "under-consistent"
```

Figure A3 - Procedure "update successor" in Dynamic Path Update Algorithm