Time Constrained Randomized Path Planning Using Spatial Networks

Christopher Lum*

Department of Aeronautics and Astronautics

University of Washington

Seattle, WA 98195, USA

lum@u.washington.edu

Rolf Rysdyk**

Insitu, Inc.

118 East Columbia River Way

Bingen, WA 98605, USA

rolf.rysdyk@insitu.com

Abstract— **Real time planning of optimal paths remains an open problem in many applications of autonomous systems. This paper demonstrates a computationally efficient method for generating a set of feasible paths through parameterization into a series of nodes. The nodes and the arcs make up a directed graph. The state of the environment is embedded in an occupancy based map. A notion of optimality is introduced by combining the directed graph with this map. Network optimization techniques are used to find the best path through the directed graph.**

NOMENCLATURE

A	Original set of arcs (edges) in network
$\tilde{A}$	Arcs to add in network
B	Spatial domain of occupancy map
C_s	Node subset at step s
d	Num steps in prediction horizon
$D(j)$	Span interval of arc j
$d^-(j)$	Lower span interval of arc j
$d^+(j)$	Upper span interval of arc j
E	Incidence matrix of network
$f()$	Returns node coordinate ($f : i \in I \rightarrow \Re^2$)
$g()$	Returns occupancy map score ($g : x \rightarrow \Re$)
I	Set of nodes (vertices) in network
$j \sim (i, i')$	Arc starting at node i and ending at node i'
N	Num times spatial network algorithm is run
N^+, N^-	Starting/ending node sets in min path algorithm
M	Num ops for ($\wp_2$) w/ PF Algorithm
M'	Num ops for ($\wp_2$) w/ combinatorial approach
P	Path of arcs from N^+ to N^-
r_{max}	Distance agent can travel in a single step
u_0	Initial potential on nodes
x	Spatial coordinate
x_0	Agent's starting coordinate
x_d	Agent's desired ending coordinate
($\wp_1$)	Problem of finding feasible waypoints
($\wp_2$)	Problem of adding arcs to network
($\wp_3$)	Problem of finding optimal path in network

I. INTRODUCTION

Finding a feasible path for an autonomous agent subject to dynamic or kinematic constraints in a complex environment is often a difficult problem. Assuming that feasible paths

---

*Research Assistant. Department of Aeronautics and Astronautics.
**Advanced Development. Insitu.

exist, optimality of the path is a subjective matter and varies with environmental conditions and constraints. Efficient computation of optimal paths under such conditions remains an open problem.

The path planning problem is often addressed as a non-holonomic planning problem. Although this is accurate, Donald et al. have shown that finding an exact, time-optimal trajectories for a system with point-mass dynamics and bounded velocity and acceleration in an environment filled with polyhedral obstacles is NP-hard[1]. In order to find solutions in a reasonable amount of time, several approximations to this problem have been made. LaValle and Kuffner addressed the nonholonomic constraints and solved the kinodynamic planning problem using rapidly exploring random trees [2]. Groups such as Capozzi et al. have also looked at semi-randomized methods which provide quasi-optimal solutions to the path planning problem using evolutionary programming[3]. Later, Pongpunwattana et al. incorporated these ideas into overall mission planning and task management schemes which address an agent's state and timing constraints[4]. Previous work investigated classical convex optimization techniques to generate simple paths from a starting point to a goal location for agents with constrained velocity limits [5].

In general, many groups have extensively studied the path planning problem and there exists a significant amount of literature detailing the problem and proposed solutions. For additional material see [6], [7], [8], [9], and [10]. Difficulties with these strategies are: they require extensive computational power (evolutionary algorithms [4], [11], are limited to generating simple paths (convex optimization techniques [5]), and many other open problems. This paper presents a set of computationally efficient algorithms for generating quasi-optimal paths through a complex environment. These algorithms are most similar to work done by Sun and Reif where they introduce the concept of optimal path planning through a non-uniformly discretized two dimensional space (referred to as a weighted region) and efficiently compute a path which is an ϵ-accurate approximation of the true optimal path though this space[12]. Although the work presented here does not guarantee accuracy within a user defined limit, its advantages over previously cited works include the ability to satisfy agent performance and timing constraints (ie user specified number of edge traversals of the graph) and extensions to path planning in three dimensional space.

This is accomplished by building a set of feasible paths as a directed graph (network) and then combining this graph with information about the environment to introduce a notion of optimality. Subsequently, an optimal path is selected using network optimization techniques.

In this framework, the agent plans an optimal path between its current location and a desired location which it must reach within a certain time or number of discrete steps. The constraints on the agent's state at different times limit the feasible paths. The environment contains hard obstacles which the agent must avoid at all costs and undesirable or desirable regions where the agent is respectively penalized or rewarded if it travels through these regions. This is similar to occupancy grid mapping as discussed by Fox et. al [13] or weighted regions[12].

Section II describes the use of occupancy based maps to represent the state of the environment. Section III details the Spatial Network Algorithm that is used to generate waypoints which satisfy the agent's spatial and temporal constraints. The waypoints are formulated as a directed graph in Section IV, and subsequently, network optimization techniques are used to optimize a path through the network as illustrated in Section V. Finally, Section VI presents some conclusions and future directions of research with this work.

II. OCCUPANCY BASED MAPS

To effectively plan a path through an environment, the system must keep track of the state of the world in terms of possible obstacle locations. To accomplish this an occupancy based map is employed. In this scheme, the domain is discretized into rectangular cells. Each cell is assigned a score based on the type of obstacle or reward that is associated with that cell. This is similar to a two dimensional, discretized probability density function [14], [15]. The spatial domain of the occupancy based map, B, consists of a two dimensional box with coordinates x_1 and x_2 such that

$$B = \left\{ x \ \middle| \ \begin{matrix} x_1 \in [x_{min}, x_{max}] \\ x_2 \in [y_{min}, y_{max}] \end{matrix} \right\} \tag{1}$$

The two dimensional occupancy based map is represented as a function, g, defined over the set $B \times \Re$ which assigns a scalar in the range $[0, 1]$ to each element $x \in B$ at a certain time step $k \in \Re$.

$$g(x, k) : B \times \Re \rightarrow [0, 1] \tag{2}$$

For purposes of this paper, the occupancy map is not time varying [16].

Path planning decisions are based on this map. The map now represents the world state with locations of obstacles and reward areas in the environment. The idea of embedding obstacles in the map is similar to defining obstacles in a configuration space [17]. The agent's belief of the world state is embedded in the occupancy based map. In this application, the occupancy based map only provides information about two states of the agent's pose (ie the planar position). An
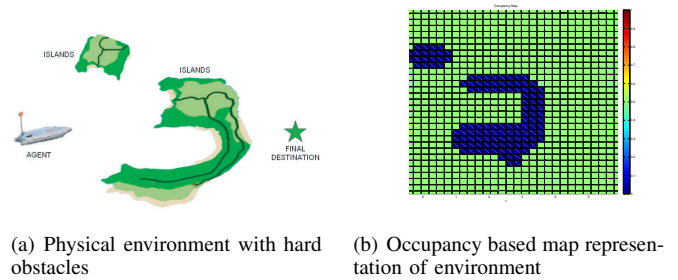

(a) Physical environment with hard obstacles

(b) Occupancy based map representation of environment

Fig. 1.   Abstraction of marine environment using occupancy based maps.

(a) Physical environment with soft and hard obstacles

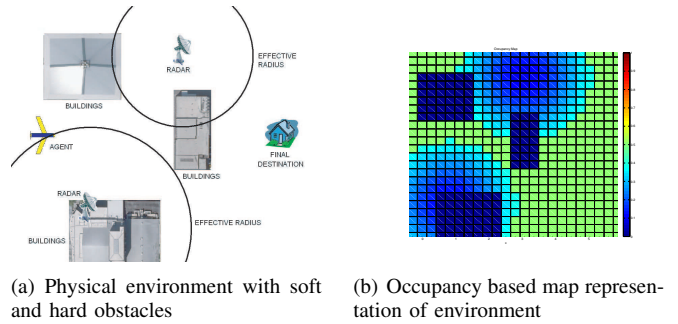(b) Occupancy based map representation of environment

Fig. 2.   Abstraction of urban environment using occupancy based maps.

example of an occupancy based map is shown below in Figures 1 and 2.

Figure 1(a) represents a marine environment that an agent may be forced to navigate through. In this scenario, there are two islands which are considered hard obstacles. This environment can be abstracted using an occupancy based map as shown in Figure 1(b). Here, the dark blue sections represent cells with zero scores (hard obstacles) and the green sections represent scores of 0.5 (neutral values).

An example of an urban environment with hard and soft obstacles is shown in Figure 2(a). The idea is the same except there are regions which are soft obstacles which should be avoided if possible but entering these regions does not violate a constraint. These sections are represented by the lighter blue shades with scores ranging from 0 to 0.5. If there were regions that were beneficial to the agent, these would be assigned scores greater than 0.5.

The occupancy based map and its associated features provides a versatile framework from which to build a path planning algorithm.

III. SPATIAL NETWORK ALGORITHM

The problem of autonomous path planning is conveniently conceptualized as finding a set of waypoints from a start point to an end point with intermediate waypoints located a reasonable distance away from the previous point. This problem is denoted ($\wp_1$).

A. Problem Statement

A feasible path for the agent consists of a sequence of waypoints ($x_i \in \Re^2$) where each consecutive waypoint is no more than a distance r_{max} away from the previous one. The problem becomes one of finding a sequence of points which

take the agent from its current point to the end point while obeying the constraints.

Given:	x_0	Agent's current location (starting point)
	x_d	Desired final location (ending point)
	r_{max}	Distance agent can travel in one step
	d	Number of steps

Assumptions:	$||x_0 - x_d|| \leq d \cdot r_{max}$	$(A.1)$
	$r_{max} > 0$	$(A.2)$
	$d \in \mathbb{Z}, d > 1$	$(A.3)$

Goal: Find a sequence $\{x_i\} : ||x_j - x_{j-1}|| \leq r_{max}$ for $i = 1, ..., d-1$ and $j = 1, ..., d$

The problem can be visualized as having a beaded necklace where each bead (represented by a waypoint) is connected to the next with a string of length r_{max} or less. One end of the necklace is fastened to x_0 and the other is fixed at x_d. Different paths correspond to different ways the necklace can be stretched while the two end points remain fixed. A stable algorithm to find the sequence $\{x_i\}$ is presented next.

B. Spatial Network Algorithm

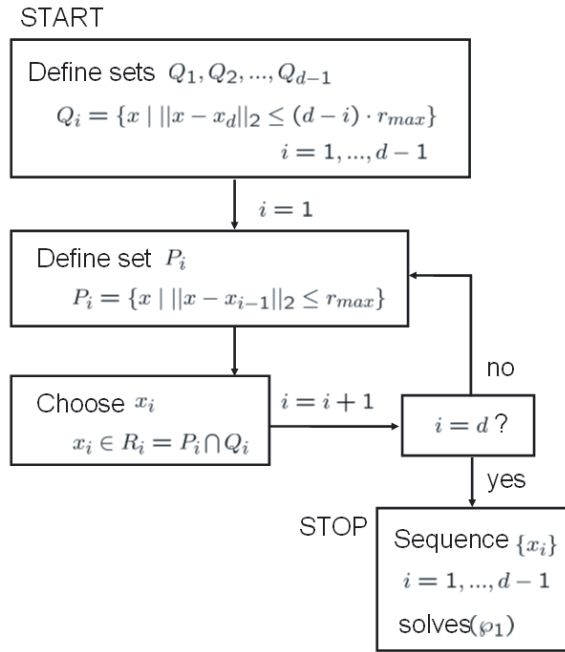A flow diagram of the Spatial Network Algorithm is shown in Figure 3.

Fig. 3. Spatial Network Algorithm for solving the $(\wp_1)$ problem.

As can be seen from the flowchart, the algorithm works by defining three sets: P_i, Q_i, and R_i.

The set P_i makes up a circle (if $x \in \Re^2$) or a sphere (if $x \in \Re^3$) centered around the point x_{i-1} with radius of r_{max}.

The set Q_i is also a circle or sphere centered around x_d (the end point). The radius of these sets changes as i changes.

Namely, as i increases by 1, the radius decreases by r_{max} until at the step $i = d - 1$, the radius is r_{max}.

Finally, the set R_i is simply the intersection of the the sets P_i and Q_i.

The algorithm is a recursive procedure. Choosing the next point x_i requires knowledge of the previous point x_{i-1}. Some parts of the algorithm do not have this coupled behavior. Namely, the sets $Q_1, Q_2, ..., Q_{d-1}$ can be defined a-priori outside of the main algorithmic loop. Inside the main loop, the additional two sets P_i and R_i are computed. The next waypoint x_i is then simply chosen from the set R_i. The natural question becomes, is there a situation where $R_i = \emptyset$? The set R_i is all the points which satisfy the following two inequalities.

$$R_i = \left\{ x \mid \begin{array}{l} ||x - x_{i-1}|| \leq r_{max}, \\ ||x - x_d|| \leq (d-i)r_{max} \end{array} \right\} \quad (3)$$

Provided that the problem adheres to the assumptions stated previously, it can be shown that $R_i \neq \emptyset$ for $i = 1, ..., d-1$. The proof is started for the case of $i = 1$. For the set R_1 to be non-empty, there must be a point which satisfies both inequalities in Eq.3. Consider the point

$$\tilde{x}_i = x_{i-1} + \frac{x_d - x_{i-1}}{||x_d - x_{i-1}||} r_{max} \quad (4)$$

At a given step i, the point $\tilde{x}_i$ is generated by taking the vector directed from the previous point x_{i-1} towards x_d and traveling along this vector a distance r_{max} (this would be the point to go to if the agent is trying to reach x_d as quickly as possible). From construction, $||\tilde{x}_i - x_{i-1}|| = r_{max} \leq r_{max}$ for all i, so the candidate point $\tilde{x}_i \in P_i$.

For the case of $i = 1$, it is now shown that this point satisfies the second inequality (and is therefore in the set Q_1).

$$||\tilde{x}_1 - x_d|| = ||x_0 + \frac{x_d - x_0}{||x_d - x_0||} r_{max} - x_d||$$

$$= ||(1 - \frac{r_{max}}{||x_d - x_0||})(x_0 - x_d)||$$

From assumption A.1, A.2, and A.3 in the problem statement, $||x_0 - x_d|| \leq d \cdot r_{max}$, $r_{max} > 0$, and $d > 1$ respectively.

$$\leq ||(1 - \frac{r_{max}}{d \cdot r_{max}})(x_0 - x_d)||$$

$$= \frac{d-1}{d}||x_0 - x_d||$$

$$\leq \frac{d-1}{d}(d \cdot r_{max})$$

$$||\tilde{x}_1 - x_d|| \leq (d-1)r_{max}$$

Therefore the candidate point $\tilde{x}_1 \in Q_1$. Since this candidate point satisfies both inequalities, it is in the set R_1 and therefore R_1 is non-empty. This demonstrates that the algorithm cannot fail at step $i = 1$. As the algorithm progresses for $i = 2, ..., d-1$, the set R_i can be shown

to always be non-empty using a similar proof as outlined below.

Once again, $R_i \neq \emptyset$ is proven by showing that the candidate point $\tilde{x}_i \in R_i$. As stated previously, $\tilde{x}_i \in P_i \; \forall i$ by construction of $\tilde{x}_i$. Showing that $\tilde{x}_i \in Q_i$ for $i = 2, ..., d-1$ relies on the fact that the algorithm is recursive and therefore the previous point $x_{i-1} \in R_{i-1}$. More specifically,

$$
\begin{aligned}
\{x_{i-1} \in R_{i-1}\} &\Rightarrow \{x_{i-1} \in Q_{i-1}\} \\
&\Leftrightarrow \{||x_{i-1} - x_d|| \leq (d-i+1)r_{max}\}
\end{aligned}
\tag{5}
$$

The proof that $\tilde{x}_i \in Q_i$ proceeds in an identical fashion to the proof $\tilde{x}_1 \in Q_1$, except instead of overbounding the right hand side using assumption A.1 from the problem statement, Eq.5 is used instead.

$$
\begin{aligned}
||\tilde{x}_i - x_d|| &= ||x_{i-1} + \frac{x_d - x_{i-1}}{||x_d - x_{i-1}||}r_{max} - x_d|| \\
&\leq ||(1 - \frac{r_{max}}{(d-i+1)r_{max}})(x_{i-1} - x_d)|| \\
&\leq \frac{d-i}{d-i+1}(d-i+1)r_{max}
\end{aligned}
\tag{6}
$$

$$||\tilde{x}_i - x_d|| \leq (d-i)r_{max}$$

This shows that $\tilde{x}_i \in Q_i$. Together with the previous proof $\{\tilde{x}_i \in R_i\} \Rightarrow \{R_i \neq \emptyset\}$, the algorithm is shown to always generate feasible flight paths because the set R_i is never empty at any step of the process.

C. Spatial Network Algorithm Results

Since the algorithm always generates feasible paths, it can now be computationally implemented and evaluated. Flexibility in the algorithm comes from the freedom to choose $x_i \in R_i$. For example, by choosing $x_i = \tilde{x}_i$, a straight flight path from x_0 to x_d is achieved where the agent attempts to reach x_d as fast as possible. There exists significant literature regarding how to intelligently choose these points. An example would be to choose $x_i \in R_i$ such x_i maximizes the reward from the occupancy based map (ie $x_i \in \arg\max_{x_i \in R_i} g(x_i)$. An example of $x_i \in R_i$ chosen to generate longer flight paths is shown below in Figure 4.

IV. Network Representation

The Spatial Network Algorithm generates a sequence $\{x_i\}$ which represents the waypoints for a path from start to end point. By executing the Spatial Network Algorithm multiple times and indexing the corresponding waypoints, the resulting set of paths can be represented as a structured network.

A. Generating Primary Paths

Waypoints form the nodes in the network and paths between them are referred to as arcs or edges. Each node embeds information about its spatial location, i.e. the coordinates x_i. The spatial network algorithm generates a sequence $\{x_i\}$ for $i = 1, ..., d-1$. Running the algorithm multiple times, only the coordinates for nodes in the middle of the
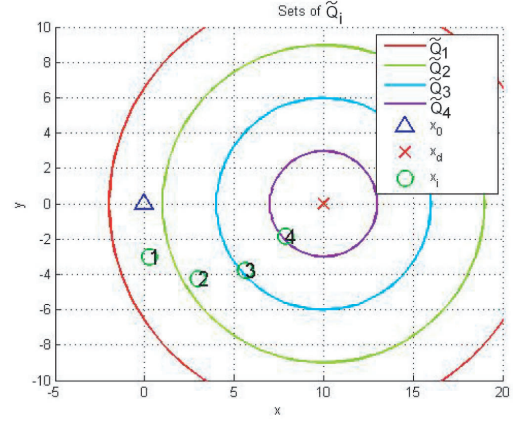

Fig. 4. A single path generated using the Spatial Network Algorithm by choosing $x_i \in R_i$ where x_i has the minimum possible y value. Situation shown for $d = 5$, $r_{max} = 3$.

path will change, the start and end points remain the same for all paths. The nodes associated with start and end points are numbered as nodes i_1 and i_2 respectively. After running the algorithm once, an additional $d - 1$ nodes are generated. These are labeled as nodes i_3 through i_{3+d-2}. The second time the algorithm is run, it adds another $d - 1$ nodes to the network. These nodes are labeled nodes i_{d+2} through nodes $i_{2\cdot d}$. This process of sequential number continues for as many times as the spatial network algorithm is run.

The arcs are defined in a similar manner. For the first path, the agent goes from node i_1 to node i_3. To represent this motion, an arc j_1 is added which originates at i_1 and terminates at i_3 (so $j_1 \sim (i_1, i_3)$). The agent then goes from i_3 to i_4, so $j_2 \sim (i_3, i_4)$ is added to the network. This continues until finally, the arc $j_d \sim (i_{d+1}, i_2)$ is added. When the Spatial Network Algorithm is run again, this numbering scheme of arcs continues in a similar fashion.

An example for $d = 3$ (number of time steps for agent to reach end point) and $N = 3$ (number of times the Spatial Network Algorithm is run) is shown in Figure 5.
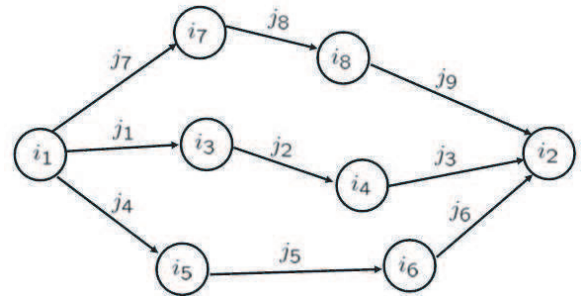

Fig. 5. An example network with $d = 3$ and $N = 3$.

Each time the algorithm is executed another path is generated from i_1 to i_2. For example, the first path is given by $P_1 : j_1, j_2, j_3$. Figure 5 is drawn specifically to draw attention to the fact that each node has coordinate data associated with

it and each node represents a waypoint in the path. Simple heuristics can be added to the algorithm to ensure that these paths do not share any nodes. These individual paths are called primary paths.

The resulting network has a one-to-one mapping with its incidence matrix. The incidence matrix becomes a useful way to represent the network. Recall that the incidence matrix is defined as

$$[E]_{ij} = \begin{cases} +1 & \text{if } i \text{ is initial node of arc } j \\ -1 & \text{if } i \text{ is terminal node of arc } j \\ 0 & \text{in all other cases} \end{cases} \quad (7)$$

The benefit of the proposed numbering system is that the network composed of N primary paths may now be represented with the convenient form of Eq. 8.

$$E = \begin{pmatrix} J & J & \dots & J \\ K & L & \dots & L \\ L & K & \dots & \vdots \\ \vdots & \vdots & \ddots & L \\ L & \dots & L & K \end{pmatrix} \quad (8)$$

$$\text{where} \quad J = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & -1 \end{pmatrix}$$

$$K = \begin{pmatrix} -1 & 1 & 0 & \dots & 0 \\ 0 & -1 & 1 & 0 & \vdots \\ \vdots & 0 & -1 & 1 & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -1 & 1 \end{pmatrix}$$

$$L = \text{zeros}(d-1, d)$$

The incidence matrix is composed of three submatrices. The matrix J is a $2 \times d$ matrix composed of all zeros except for a 1 in the top left entry and a -1 in the bottom right entry. The matrix K is a $d-1 \times d$ matrix. The first row has entries -1 and +1 as the first two and then zeros elsewhere. This pattern of a -1 followed by a +1 moves down the diagonal of the matrix. The matrix L is simply a $d-1 \times d$ matrix of zeros. So the overall incidence matrix is a matrix of size $2 + N(d-1) \times Nd$ with non-zero entries along the top two rows and in a somewhat block diagonal shape.

B. Generating Secondary Paths

All of the forward paths in the network are feasible paths for the agent and transition it from its current location to the desired end location in d steps or less. However, the number of feasible paths is exactly equal to the number of times the Spatial Network Algorithm is run. In order to generate another feasible path, the Spatial Network Algorithm must be run again.

In Figure 5, notice that a path from i_1 to i_3 to i_8 to i_2 would also take the agent to the desired final coordinate in d steps. However, an arc from i_3 to i_8 does not exist.

To represent a feasible path, an arc must satisfy several requirements. First, the arc must join two nodes that are separated by a distance r_{max} or less. Second, traversal of the arc must take the agent closer to the desired end state. Third, the complete sequence of arcs must place the agent at the end state in d steps or less. The question becomes, where can arcs be added so that positive paths in the network are still feasible paths for the agent and places the agent at the end state in d steps or less? This problem is referred to as $(\wp_2)$. To solve $(\wp_2)$, the node subsets C_s are defined as follows.

$$C_s = \{i_{2+s+t(d-1)}, \text{for } t = 0, 1, \dots, N-1\} \quad (9) \\ \text{for } s = 1, 2, \dots, d-1$$

Using Figure 5 as an example, the sets defined by Eq. 9 yield $C_1 = \{i_3, i_5, i_7\}$, $C_2 = \{i_4, i_6, i_8\}$. The set C_1 represents the nodes which are 1 arc away from the initial point (therefore $d-1$ away from the final point). Similarly, the set C_2 represents the nodes that are 2 arcs away from the initial point (therefore $d-2$ arcs away from the final point). An algorithm for solving $(\wp_2)$ is shown in Figure 6. We refer to this algorithm as the Progressive Frontier Algorithm.
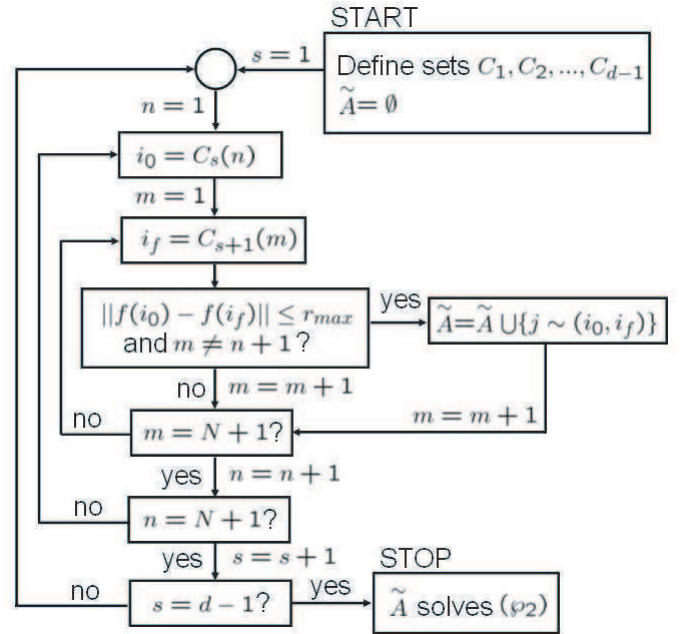

Fig. 6. Progressive Frontier Algorithm flowchart.

The Progressive Frontier Algorithm is a series a three loops. At a given s, the algorithm functions by choosing a node $i_0 \in C_s$ and then comparing it with every node in $i_f \in C_{s+1}$ (except for the next consecutive node because an arc already exists between them). If the distance is less than or equal to r_{max}, an arc from i_0 to i_f is added to the set $\tilde{A}$. Once this is completed for all nodes in C_s, s is incremented by one and the process is repeated.

Using Figure 5 as an example, the algorithm first chooses $i_0 = i_3$ and compares the distance between this node and

nodes i_6 and i_8 (but not with i_4 because $j \sim (i_3, i_4)$ already exists). Whenever the distance is less than or equal to r_{max} an arc is added to $\tilde{A}$. Then i_0 is changed to i_5 and it is compared with nodes i_4 and i_8. Finally, i_0 is changed to i_7 and it is compared with nodes i_4 and i_6.

The Progressive Frontier Algorithm performs the logical test of $||f(i_0) - f(i_f)|| \leq r_{max}$ exactly $M = (d-2)N^2 + (2-d)N$ times. If a combinatorial approach is taken and the distance between a given node is compared with every other intermediate node in the network, the logical test must be performed exactly $M' = d(d-2)N^2 + N^2 + 3(d-1)N + 2$ times. Assuming that $N >> d$, the ratio of the two quantities is approximately $M'/M \approx d$. The advantage of this approach instead of a combinatorial approach is immediately evident by the significant computational savings.
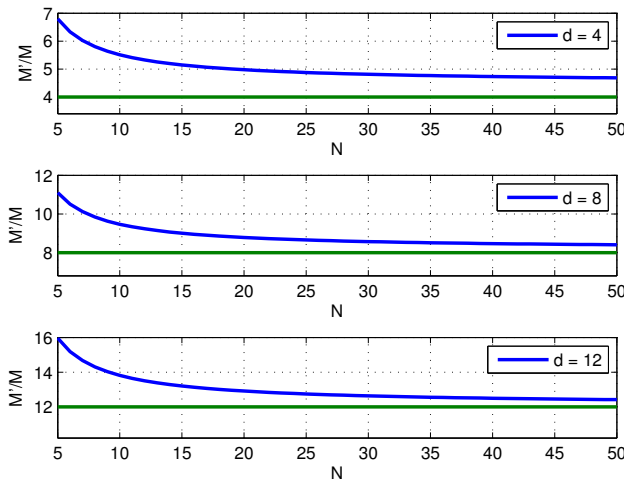

Fig. 7.   Ratio of M'/M approaching d for $d = 4$, 8, and 12.

When the Progressive Frontier Algorithm terminates, the network is updated by adding the new arcs, $\tilde{A}$, to the current arc set, A, to obtain a new set of arcs, A'.

$$A' = A \bigcup \tilde{A} \tag{10}$$

An example of the Progressive Frontier Algorithm run on a network with $d = 5$, $N = 3$ is shown in Figure 8. The original, primary paths are shown as black arcs. The algorithm is run with $r_{max} = 1.5$ and the new arcs which make up $\tilde{A}$ are shown in green.

V. NETWORK OPTIMIZATION

The remaining goal is to find an optimal path among all the generated feasible paths from node i_1 to node i_2. This problem is referred to as $(\wp_3)$.

A. Combining with Occupancy Based Map

To solve problem $(\wp_3)$, the notion of span intervals are introduced. The span interval is a a nonempty real interval $D(j)$ assigned to each arc $j \in A$. [18]
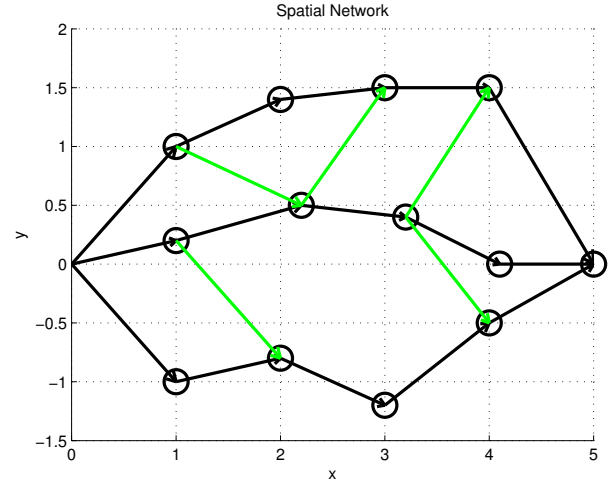

Fig. 8.   Arcs added for $d = 5$, $N = 3$, and $r_{max} = 1.5$.

$$D(j) = [d^-(j), d^+(j)] \tag{11}$$

The value $d^-(j)$ is the lower span interval and in the context of $(\wp_3)$, this represents the cost of traversing the arc j in the reverse direction. Similarly, $d^+(j)$ is the upper span interval and represents the cost of traversing the arc j in the forward direction. Information about the environment is embedded into the network by combining the location of the nodes with the scores of the occupancy based map.

$$\begin{aligned} d^-(j) &= -\infty \\ d^+(j) &= \frac{1}{g(f(i'))} \end{aligned} \quad \forall j \sim (i, i') \in A \tag{12}$$

The lower span intervals are set to $-\infty$ for all arcs. The upper span intervals are set using information from the occupancy based map. For the arc $j \sim (i, i')$, the terminal node's coordinates are returned by the function f. The function g then maps these coordinates to the corresponding score in the occupancy based map. Since the range of g is $[0, 1]$, the upper span interval ranges from 1 (when the terminal node is located in a region with score equal to 1) to $+\infty$ (when the terminal node is located in a region with score equal to 0).

One final detail must be addressed in order to formulate the $(\wp_3)$ as a network optimization problem. This concerns the initial potential u_0 which assigns a scalar value to each node $i \in I$. The differential induced by this potential must be feasible with respect to the span intervals. In other words, the differential $v_0 = -u_0 E$ must be in the span interval $D(j)$ for all $j \in A$. Note that since $0 \in D(j) \ \forall j \in A$, the initial potential can always be taken as $u_0 = 0 \ \forall i \in I$. [18]

B. Min Path Optimization

With the span intervals and initial potential set, $(\wp_3)$ is now in the form of a network optimization problem. A path P is a signed, ordered set of arcs that can be further decomposed into sets P^+ and P^- (arcs which are traversed

in the positive and negative direction, respectively). The cost function below evaluates the cost of traversing a certain path P.

$$d^+(P) = \sum_{j \in P^+} d^+(j) - \sum_{j \in P^-} d^-(j) \qquad (13)$$

The well known Min Path Problem consists of minimizing $d^+(P)$ over all paths $P : N^+ \to N^-$. In the context of this path planning algorithm, $N^+ = i_1$ and $N^- = i_2$. The Min Path Problem is solved efficiently using the well known Min Path/Max Tension algorithm [18] or the Bellman-Ford algorithm [19]. With some problem reformulation, Dijkstra's Method may also be used to further increase computational efficiency [20].

Assuming that assumption $(A.1)$ is satisfied in $(\wp_1)$, a forward path $P : N^+ \to N^-$ must exist. By setting $d^-(j) = -\infty \ \forall j$, the resulting cost function has the behavior that $\{P^- \neq \emptyset\} \Rightarrow \{d^+(P) = \infty\}$. Therefore, this formulation ensures that only forward paths can solve the min path problem. In some situations, it may be optimal to traverse an arc in a backwards direction. However, if this is done it is not possible to guarantee that the path will take the agent to the final positing in d steps or less. Furthermore, the careful selection of $d^+(j)$ also yields the behavior that if an arc terminates in a location where the score of the occupancy map is zero, then $d^+(P) = \infty$ as well. This allows the notion of a hard constraint (or obstacle) to be enforced. In other words, the Min Path/Max Tension algorithm will only find paths that do not have arcs which terminate in a location where the occupancy map score is zero. Applying the Min Path/Max Tension algorithm to the network constructed for the marine situation depicted in Figure 1 yields an optimal path P^* shown in Figure 10.
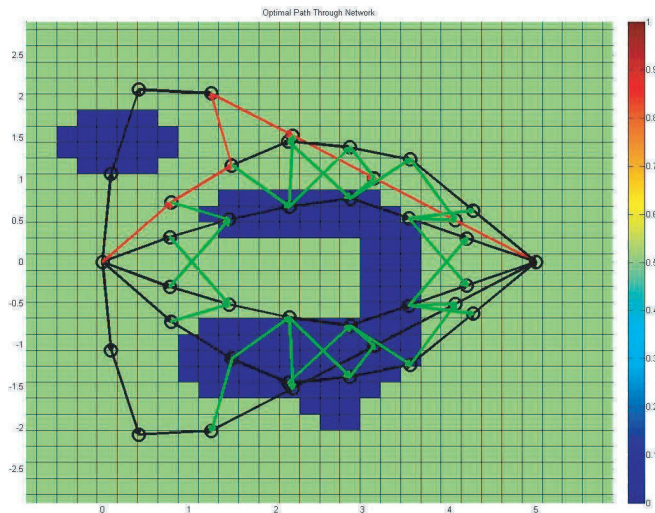

Fig. 9. Optimal path through environment with $d = 7$, $r_{max} = 1.06$, $N = 6$. Optimal path P^* shown in red.

Figure 9 is generated using the methods described in the previous sections. First, the Spatial Network Algorithm is executed six times to generate the six primary paths

which are shown in black. Then, the Progressive Frontier Algorithm is run to solve $(\wp_2)$ which adds the green arcs. The span intervals of each arc (both black and green arcs) are set by combining the network with the occupancy based map using Eq.12. In this example, the islands which the agent must navigate around represent hard obstacles which must be avoided. Therefore, the occupancy based map cells corresponding to the islands are given a score of zero (the dark blue regions). Once the span intervals are set, the initial potential is set to zero for all nodes and the Min Path/Max Tension algorithm is run to generate the optimal path, which is shown in red. The agent successfully plans a path which circumnavigates the islands on its way to the final destination. Notice that this problem is one of simply finding a feasible path since all the scores of each cell are either 0 or 0.5. Any of the feasible paths yield the exact same score under the cost function of Eq. 13. This is why the optimal path appears to go in the backwards direction from step 2 to step 3. Despite this, the agent is still placed at the desired end goal in d steps or less. This issue can be addressed by incoporating the concept of Euclidean distance into the cost function (Eq.13) by changing the way the span intervals are defined in Eq. 12. Namely, adding a term of $+\alpha \|f(i) - f(i')\|$ to the definition of $d^+(j)$ where α is a parameter to trade off between occupancy map score and distance traveled.

Note that the middle island has a box canyon type shape. Many methods which plan using a receding horizon type algorithm may fail with this geometry. The agent enters into the canyon and does not realized until it is too late that it has effectively entered a dead-end. This algorithm avoids this pitfall by planning waypoints over the entire path.

Path planning in the urban environment depicted in Figure 2 is done using the same methods. The optimal path through this environment is shown in Figure 10.
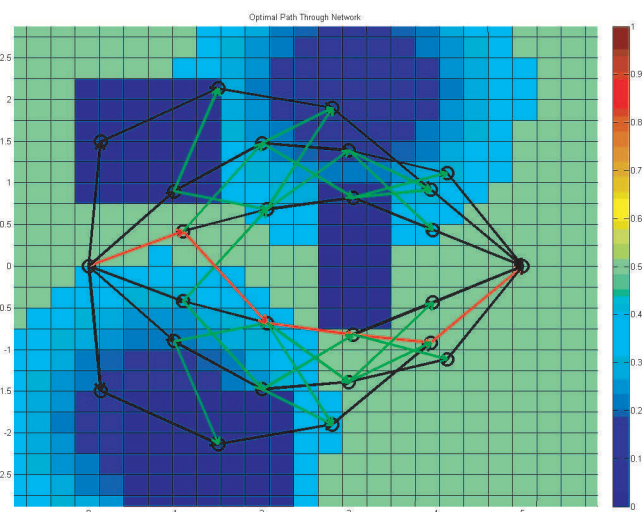

Fig. 10. Optimal path through environment with $d = 5$, $r_{max} = 1.5$, $N = 6$. Optimal path P^* shown in red.

The main difference between this example and the marine

one is that this environment contains soft obstacles (the radar stations). Therefore in addition to finding feasible paths, an optimal one can be selected. As can be seen, the agent's first waypoint avoids the areas of low score but the second waypoint must enter the area slightly so that the third waypoint can avoid the region with zero score. The fourth waypoint also avoids the obstacle and the final, fifth waypoint brings the agent to the desired final position.

VI. CONCLUSION AND FURTHER RESEARCH

This paper presents a computationally efficient approach to obtain feasible paths through complex environments. The approach formulates path planning as a network optimization problem. A network of feasible paths is generated first. Next, the network is combined with occupancy based maps to embed information about the environment. The well known Min Path Algorithm generates optimal paths through the network. Although these algorithm are not able to claim a bounded error from a true optimal path, the network formulation ensures that a quasi-optimal path avoids hard obstacles and reaches the goal in the required number of discrete steps.

In the current implementation, an arc is given an infinite upper span interval only if the terminating node is located in a region of zero score. Therefore, an arc that passes through but does not terminate in a region of zero score would receive an upper span interval of less than infinity. In some situation, this may be unacceptable because the agent will intersect with a hard obstacle in the environment. The upper span interval can be assigned to be infinite if any part of the arc passes through a region of zero score. This significantly increases computational complexity but guarantees that all paths are feasible in the sense that there are no intersections with hard obstacles. Methods are currently being investigated which can make this problem more tractable.

Although the examples presented were two dimensional, the algorithms and proofs presented apply to three dimensions. This may have applications for path planning for underwater vehicles or unmanned aerial systems which can maneuver in three dimensions.

VII. ACKNOWLEDGEMENTS

REFERENCES

[1] Donald, B., Xavier, P., Canny, J., and Reif, J., "Kinodynamic Motion Planning," *Journal of the Association for Computing Machinery*, November 1993, pp. 1048–1066.

[2] LaValle, S. M. and Kuffner Jr, J. J., "Randomized Kinodynamic Planning," *International Journal of Robotics Research*, May 2001, pp. 378–400.

[3] Capozzi, B. J. and Vagners, J., "Navigating Annoying Environments Through Evolution," *Proceedings of the 40th IEEE Conference on Decision and Control*, University of Washington, Orlando, FL, 2001.

[4] Pongpunwattana, A. and Rysdyk, R. T., "Real-Time Planning for Multiple Autonomous Vehicles in Dynamic Uncertain Environments," *AIAA Journal of Aerospace Computing, Information, and Communication*, December 2004, pp. 580–604.

[5] Lum, C. W., Rysdyk, R. T., and Pongpunwattana, A., "Occupancy Based Map Searching Using Heterogeneous Teams of Autonomous Vehicles," *Proceedings of the 2006 Guidance, Navigation, and Control Conference*, Autonomous Flight Systems Laboratory, Keystone, CO, August 2006.

[6] Mitchell, J. S. B. and Papadimitriou, C. H., "The Weighted Region Problem: Finding Shortest Paths Through a Weighted Planar Subdivision," *Journal of the ACM*, January 1991, pp. 18–73.

[7] Alexander, R. and Rowe, N., "Path Planning by Optimal-Path-Map Contruction for Homogeneous-Cost Two-Dimensional Regions," *Proceedings of the IEEE International Conference on Robotics and Automation*, IEEE, 1990.

[8] Rubio, J. C., Vagners, J., and Rysdyk, R. T., "Adaptive Path Planning for Autonomous UAV Oceanic Search Missions," *Proceedings of the 1st AIAA Intelligent Systems Technical Conference*, 2004.

[9] Pongpunwattana, A., Rysdyk, R. T., Vagners, J., and Rathbun, D., "Market-Based Co-Evolution Planning for Multiple Autonomous Vehicles," *Proceedings of the AIAA Unmanned Unlimited Conference*, Autonomous Flight Systems Laboratory, 2003.

[10] LaValle, S. M., *Planning Algorithms*, Cambridge University Press, 2006.

[11] Rathbun, D. and Capozzi, B., "Evolutionary Approaches to Path Planning Through Uncertain Environments," *Proceedings of the AIAA Unmanned Unlimited Conference*, AIAA, May 2002.

[12] Sun, Z. and Reif, J. H., "On Finding Approximate Optimal Paths in Weighted Regions," *Journal of Algorithms*, January 2006, pp. 1–32.

[13] Thrun, S., Burgard, W., and Fox, D., *Probabilistic Robotics*, MIT Press, 2005.

[14] Bourgault, F., Furukawa, T., and Durrant-Whyte, H., "Coordinated Decentralized Search for a Lost Target in a Bayesian World," *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, Australian Centre for Field Robotics, Las Vegas, NV, October 2003.

[15] Elfes, A., "Using Occupancy Grids for Mobile Robot Perception and Navigation," *IEEE Computer*, 1989, pp. 46–57.

[16] Lum, C. W., Rysdyk, R. T., and Pongpunwattana, A., "Autonomous Airborne Geomagnetic Surveying and Target Identification," *Proceedings of the 2005 Infotech@Aerospace Conference*, Autonomous Flight Systems Laboratory, Arlington, VA, September 2005.

[17] Latombe, J.-C., *Robot Motion Planning*, Kluwer Academic Publishers, 1991.

[18] Rockafellar, R., *Network Flows and Monotropic Optimization*, Athena Scientific, Belmont, Mass, 1st ed., 1998.

[19] Bellman, R., "On a Routing Problem," *Quarterly of Applied Mathematics*, April 1958, pp. 87–90.

[20] Bertsekas, D. P., *Network Optimization: Continous and Discrete Models*, Athena Scientific, Belmont, Mass, 1st ed., 1998.