

# Contemporary Models for Path Prediction of Dynamic Entities

*Sanjeeb Nanda*

SDS International Inc., Advanced Technologies Division  
3403 Technological Avenue, Suite 7  
Orlando, FL 32817  
407-282-4432  
[snanda@sdslink.com](mailto:snanda@sdslink.com)

*Dr. Joseph Weeks*

Air Force Research Laboratory  
Human Effectiveness Directorate  
Warfighter Readiness Research Division  
6030 South Kent Street  
Mesa, AZ 85212-6061  
480-988-6561 ext. 249  
[joseph.weeks@mesa.afmc.af.mil](mailto:joseph.weeks@mesa.afmc.af.mil)

Keywords:

Battlespace, DTED, Graph theory, Markov chain, optimization, prediction, pruning, and trafficability.

**ABSTRACT:** *As militaries across the world evolve, the roles of humans in various theatres of operation are being identified for elimination in favor of automation. Forward observation and coordination of supporting arms to neutralize threats from dynamic adversaries is one such area. However, contemporary systems for tracking and targeting lack the desired capabilities to serve as effective substitutes. Systems with greater intelligence to predict the future positions of multiple adversaries are needed. This paper presents two contemporary approaches as solutions to this challenge. The first models the operating environment of an entity as a directed graph with one or more decision variables governing the costs of its edges. The projected path of the entity is then derived by applying pruning heuristics to optimal paths containing the observed course of the entity as a subpath. The second approach creates a finite state machine with each state comprised of an ordered set of attributes describing the entity and its relationship to the environment at periodic intervals in time. The transitional probabilities between the observed states are computed and used in a Markov chain across multiple transitions to furnish the conditional probability of its state and position in the future.*

## 1. Introduction

The United States Air Force is pursuing a vision of networked sensors to shrink the sensor-to-shooter cycle and give commanders greater awareness in support of decision superiority and battlefield dominance. According to doctrine (Intelligence, Surveillance, and Reconnaissance Operations, Air Force Doctrine Document 2 - 9, 2005), "Surveillance is the systematic observation of [air and space], surface or subsurface areas, places, persons, or things, by visual, aural, electronic, photographic or other means." Although the goal is sustained and persistent surveillance, a host of environmental and operational factors would be expected to present obstacles to successful achievement. Consequently, support for persistent surveillance could be needed. Independent of the requirement for persistence,

there are ever-greater pressures for speed of detection and assessment. This has resulted in a greater doctrinal emphasis on anticipatory intelligence and predictive battlespace awareness. The confluence of these considerations implies the need for automated systems to support persistence and to overcome limitations in human processing speed and information distribution.

The algorithms we present assume that an entity is traveling with the intent to optimize a unique decision variable. Some examples include, the time required to travel to a destination designated by its mission that is unknown to the observer, or the cumulative time during which it is open to visual or electronic surveillance. We assume that a subset of this path is known to us that may include the origin of the entity's mission. To meaningfully describe approaches using these decision

variables, we first introduce the following definitions to enable us to model the problem as a graph followed by an overview of the candidate graph theoretic algorithms.

## 2. A Graph Theoretic Approach

We define a graph  $G$  as an ordered pair  $(V, E)$ , where  $V$  is a set of vertices, each corresponding to a DTED (Digital Terrain Elevation Data) post, and  $E$  is a set of directed edges between pairs of vertices, each assigned a real-valued cost. The posts are physically arranged in a square grid with each vertex having an edge to eight others to its north, northeast, east, southeast, south, southwest, west, and northwest. The distance  $d$  from a vertex to the neighbors to its north, east, south and west is uniformly either of 100 (Level 1), 30 (Level 2) or 5 (Level 3) meters. Similarly, the distance from that vertex to the diagonal neighbors to its northeast, southeast, southwest and northwest is uniformly  $\sqrt{2}d$ . Figure 1 illustrates the aforementioned layout. The costs assigned to the directed edges are a measure of the time it takes an entity to traverse the corresponding edge, and is a function of several criteria. They include the distance between the adjacent vertices, the difference in elevation between them, and the trafficability of the terrain in the path given by the directed edge in 3D space.

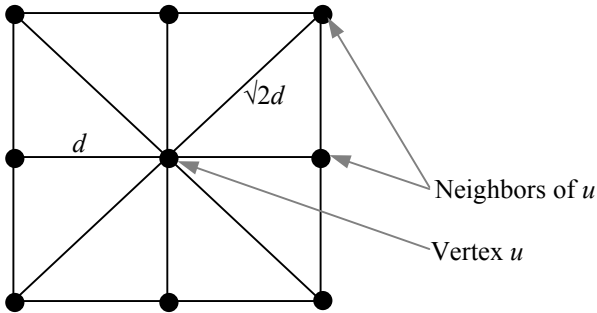


Figure 1. Layout of DTED vertices in a square grid.

For example, suppose vertices  $v$  and  $w$  are neighbors of  $u$  at identical distances to it. All other things being the same, if  $v$  is at a higher elevation than  $u$  and  $w$  is at a higher elevation than  $v$  then cost for traveling from  $u$  to  $w$  should exceed the cost of traveling from  $u$  to  $v$ . Using these definitions, we now introduce the candidate algorithms.

### Algorithm 1

This algorithm assumes that an entity is traveling with the intent to minimize the time to travel to a specific destination designated by its mission. For this purpose, let  $P = \{u_1, u_2, u_3, \dots, u_N\}$  be the observed path taken by an

entity, where  $u_1$  and  $u_N$  are the source vertex closest to the first position and the sink vertex closest to the last position respectively at which that entity is observed. Furthermore, let the maximum known speed of that entity, or alternatively its observed speed during its progress from  $u_1$  to  $u_N$ , is  $v$ . Suppose its position is to be projected after time interval  $\Delta T$  following its departure from position  $u_N$ . Then, let  $V$  be the set comprised of all vertices  $u$  in  $P$  in addition to every vertex  $w$  that is reachable from  $u$  along a path with at most  $\Delta T/v$  vertices. Thus,  $V$  is the set of all vertices the entity can reach in the time interval  $\Delta T$  given its speed and its last known location. In reality, if  $u_1$  and  $u_N$  are substantially apart, many of the vertices in the neighborhood of  $u_1$  may be infeasible locations that should be pruned.

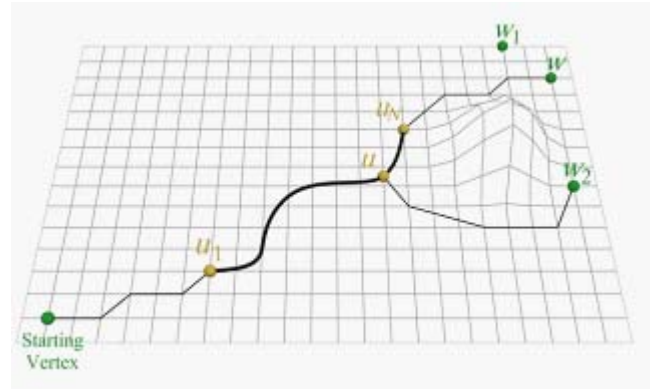


Figure 2. Hypothetical observed path of an entity and the candidate destinations.

Algorithm 1 is based on the assumption that if an entity is proceeding with the intent of minimizing its time to travel to a designated destination (that is unknown to the observer), then the observed path must be a subpath of that optimal path. Furthermore, the vertex actually reached by that entity after time interval  $\Delta T$  following its departure from  $u_N$  must be an optimal path from  $u_1$ . Let  $V_C$  be the vertices in  $V$  that are reachable from  $u_N$  by the entity in time  $\Delta T$  traveling at its last observed speed. Then, we can identify the likely destination by computing the optimal route from the source  $u_1$  to the set of vertices in  $V_C$  and choosing those vertices in it to which the optimal path includes  $P$  as a subpath. Figure 2 illustrates the aforementioned idea with a hypothetical path that an entity is observed to have traversed starting at  $u_1$  and ending at  $u_N$ . Now for example, the entity may be at any of the locations  $w$ ,  $w_1$  or  $w_2$  after time  $\Delta T$ , where  $w$  is the true destination at which the entity would arrive following the given time interval. (Note that a hill, represented by a bump on the grid separates vertices  $w$  and  $w_2$ .) However, if say the optimal path from  $u_1$  to  $w_2$  bypasses all vertices after some vertex  $u$  in  $P$  then we are assured that  $w_2$  cannot be a feasible destination. On the other hand, if the optimal path from  $u_1$  to  $w_1$  includes all the vertices in  $P$

then we must choose  $w$  as a potential destination. Thus, this procedure can yield multiple vertices as prospective destinations, many of which can be eliminated in the manner in which  $w_2$  was discarded.

We can calculate the shortest path from  $u_S$  to each vertex in  $V_C$  using any greedy algorithm that always adds the least distant unvisited vertex at each iteration from  $u_1$ . For this purpose we may subsume algorithms such as  $A^*$ . Once all the paths have been determined, we may find the path that includes  $P$  as a subpath. The pseudo-code in Figure 3 describes the complete algorithm encapsulated into a procedure that returns a set  $D$  of feasible destination vertices. In the pseudo-code, the term  $distance(u)$  gives the currently known shortest distance from vertex  $u$  to  $u_1$  and  $path(u)$  gives the vertex neighboring  $u$  on the shortest path to  $u_1$ . Thus, the shortest path from a vertex  $u$  to  $u_1$  is obtained by iteratively calling the step  $u \leftarrow path(u)$ .

#### **EstimatePosition( $\Delta T$ )**

```

{
   $P \leftarrow \{u_1, u_2, u_3, \dots, u_N\}$  is the observed path;
   $V \leftarrow$  all vertices reachable from vertices in  $P$  along
    paths with less than or equal to  $\Delta T/v$  vertices;
   $S \leftarrow \{u_S\}$ ;

  // For each vertex  $u$  adjacent to  $u_1$  assign to
  //  $distance(u)$  the cost of the edge  $(u_1, u)$ .
  Step A:
  for each vertex  $u$  in  $V - S$  do
  {
     $path(u) \leftarrow null$ ;
    if (cost of  $(u_1, u)$  is finite)
    {
       $distance(u) \leftarrow$  cost of  $(u_1, u)$ ;
      Append  $u_1$  to the  $path(u)$ ;
    }
  }

  // Choose the vertex  $w$  with the least  $distance(w)$ 
  // and update  $distance(u)$  for each  $u$  adjacent to  $w$ .
  Step B:
  repeat  $(N - 1)$  times
  {
    Choose  $w$  in  $V - S$  with minimum  $distance(w)$ ;
    Add  $w$  to  $S$ ;
    for each vertex  $u$  in  $V - S$  do
    if ( $distance(w) + \text{cost of } (w, u) < distance(u)$ )
    {
      Append  $w$  to the  $path(u)$ ;
       $distance(u) \leftarrow distance(w) + \text{cost of } (w, u)$ ;
    }
  }

  // If the path leading up to some  $u$  in  $V_C$  contains  $P$ 

```

// as a subpath then  $u$  is a candidate destination.

#### **Step C:**

```

for each vertex  $u$  in  $V_C$  do
{
  while ( $path(u) \neq u_N$ ) do {  $u \leftarrow path(u)$ ; }
   $k \leftarrow N$ ;
  do
  {
     $u \leftarrow path(u)$ ;
     $k \leftarrow k - 1$ ;
  } while ( $u = u_k$ ) and  $(k \geq 1)$ ;

  if ( $k = 0$ ) {  $D \leftarrow u$ ; }
}

return  $D$ ;
}

```

**Figure 3. Procedure to derive a feasible set of destination vertices.**

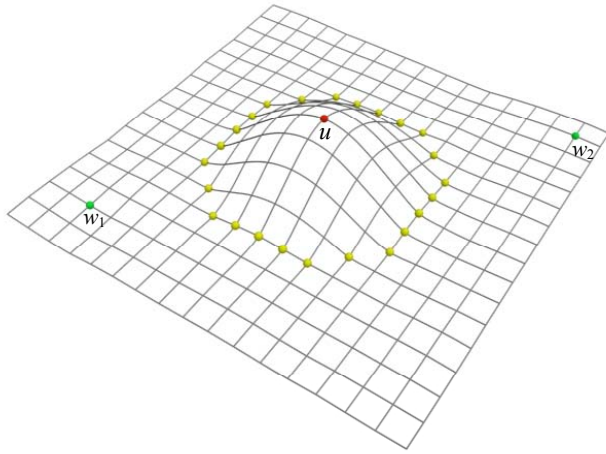
A caveat to the application of Algorithm 1 is that, the path  $P$  taken by an entity may not be exactly optimal. In fact that is seldom the case in practice. Hence, Step C may be replaced by a Euclidean test for distance to ensure that if a subpath of a candidate path is in close proximity to  $P$  as opposed to being equal to  $P$ , then the end of that path is considered a feasible destination.

From our previous discussion we observe that, Algorithm 1 deems all the feasible destinations to be the position of the entity in the future with equal probability. In practice, a number of criteria may be available for use in pruning the set of feasible destinations. One of the pruning techniques that may be employed for this purpose is as follows.

#### **Pruning Technique 1**

Let  $D$  be a set of feasible vertices derived by Algorithm 1, and  $R_d(u)$  the set of vertices that are reachable from  $u$  with path lengths less than or equal to  $d$ , for  $d \geq 1$ . Now, if for some vertex  $w$  in  $D$ , the cost of the path from  $w$  to each vertex in  $R_d(u)$  is less than the path from  $u$  to the same vertex in  $R_d(u)$ , then  $u$  cannot be a feasible intermediate destination because each vertex in  $R_d(u)$  can be reached with lesser cost by proceeding through  $w$ , and therefore any final destination as well. In practice, given  $u$ , discovering  $R_d(u)$  with this property may be computationally very expensive. Therefore, a prospective heuristic that may be alternatively applied is to substitute  $R_d(u)$  with the vertices of an isoline bounding  $u$  that has an elevation close to  $w$ .

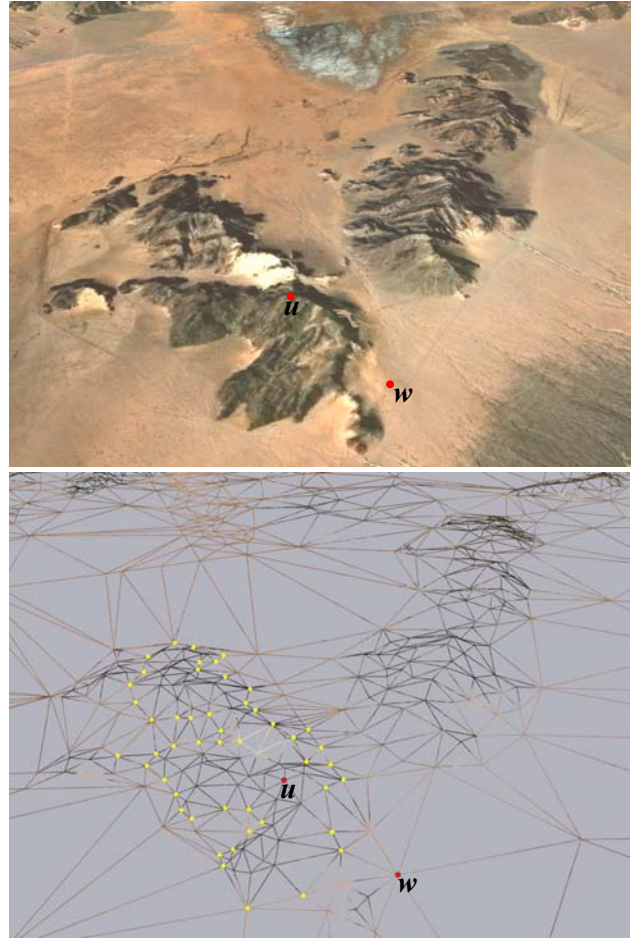
Figure 4 illustrates this idea, with an example where vertices  $u$ ,  $w_1$  and  $w_2$  and are all feasible destinations. Vertex  $u$  is atop a hill with steep gradient and the set  $R$  of yellow colored vertices at the base of that hill constitute an isoline with the same elevation as  $w_1$ . Then, if each vertex in  $R$  can be reached with lesser cost from  $w_1$  than from  $u$ , we can prune vertex  $u$  from the set of feasible destinations. While the aforementioned example of pruning was applied to a vertex atop a hill, that idea is easily extended to include vertices that lie within natural and man-made objects such as lakes, rivers, forests, dunes, valleys and buildings over which traversing is difficult, if not impossible.



**Figure 4. A destination vertex atop a hill that is a candidate for pruning.**

Figure 5 illustrates this concept more practically with a snapshot of two small hills in the 29Palms Marine Corps Air Ground Combat Center on the left and its underlying wire frame representing the DTED-based graph on the right. As discussed earlier, vertex  $u$  lies atop the hill, while  $w$  is on relatively flat terrain that is easily traversable, particularly by vehicles. Thus a vehicle can reach all the points on the terrain marked by the yellow colored vertices with greater ease than the point marked by  $u$ .

The pseudo-code in Figure 6 describes the salient steps in the generation of isolines from DTED posts (Nanda and Lickteig 2005). It modifies the *package wrapping* technique used for deriving convex hulls by iteratively choosing vertices in anticlockwise direction starting with the vertex with the least  $z$ -coordinate value. We assume that the coordinate systems is right handed with the  $x$ -axis pointing east, the  $z$ -axis pointing north, and the  $y$ -axis outwards from the center of the earth. In the pseudo-code, the meanings of the following terms are as follows. The term  $comp(u)$  denotes the component number of vertex  $u$ ,



**Figure 5. A hypothetical destination vertex atop a hill that is a candidate for pruning.**

$elevation(u)$  the elevation of vertex  $u$ ,  $minimum(u, w)$  the minimum of the vertex indexes  $u$  and  $w$ ,  $distance(u, w)$  the Euclidean distance between  $u$  and  $w$ ,  $C[k]$  the set of vertices having component number  $k$ ,  $L[k]$  the set of vertices comprising the isoline bounding the vertices in component  $k$ , and  $angle(u, v)$  the angle between the edge  $(u, v)$  and the line parallel to the  $x$ -axis passing through  $u$ .

**GetIsolineVertices()**

```
{
  // Each vertex is initially in its own component.
  Step A:
  for each vertex  $u$  in  $V$  do {  $comp(u) \leftarrow u$ ; }

  // Partition vertices into components, each at the
  // same elevation and within distance of  $\sqrt{2}d$  from
  // at least one other vertex in the same component.
  Step B:
  for each pair of vertices  $u$  and  $w$  in  $V$  do
  {
```

```

if (elevation(u) = elevation(w)) and
(distance(u, w) ≤ √2d)
{
  for each vertex v in V do
  {
    if (comp(v) = comp(u)) or (comp(v) =
comp(w))
    { comp(v) ← minimum(u, w); }
  }
}

// Scan all vertices and arrange them into
components.
Step C:
component_numbers ← 0;
for each vertex u in V do
{
  C[comp(u)] ← u;
  if (comp(u) > component_numbers)
    component_numbers ← comp(u);
}

// Wrap the vertices in each component starting with
// the vertex with the minimum z-coordinate value.
Step D:
for k ← 0 to component_numbers do
{
  u ← vertex with minimum z-coordinate in S;
  S ← C[k]; L[k] ← {u};
  do
  {
    minimum_angle ← ∞;
    for each vertex v in S do
    if ((angle(u, v) < minimum_angle) and
(distance(u, v) ≤ √2d))
    { minimum_angle ← angle(u, v); w ← v; }

    S ← S - {w}; L[k] ← L[k] + {w};
  } while w ≠ u;
}
return L;
}

```

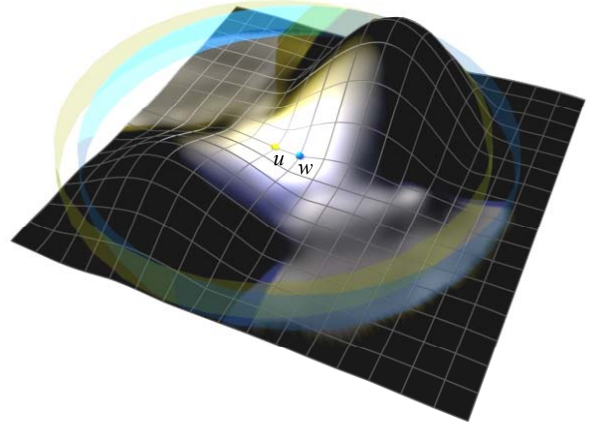
**Figure 6. Algorithm for deriving the vertices comprising isolines.**

As shown by the preceding discussion, the application of well-behaved pruning techniques is crucial to the derivation of smaller sets of feasible destinations. Some potential approaches include the investigation of the effects of logistical infrastructures and facilities such as roads, bridges, airports, and population concentrations on the paths of entities. Furthermore, the efficacy of these techniques can be measured in simulations using human

operators and the feedback used to choose the most desirable techniques for further refinement.

#### Algorithm 2

In Algorithm 1, the costs assigned to the directed edges were a measure of the time it took an entity to traverse along the corresponding edge. Alternatively, if the entity is traveling with the ostensible intent of minimizing the duration of its exposure to surveillance, we may assign costs to the directed edges that are a measure of the duration of time for which that entity is exposed. An entity is exposed electronically or visually if and only if there exists a line of sight (LOS) from an observing part to that entity. Therefore, a well-behaved function describing the cost from exposure to surveillance for an edge must be directly proportional to the size of the area from which that edge is visible. Since edges and surfaces of terrains are each comprised of an infinite number of points, the task of deriving such costs is intractable unless we consider only a finite number of points on each edge and terrain, to and from which LOS is determined. Hence, the treatment of terrains as graphs is very useful. We can therefore define a function for specifying the cost of each edge as follows.



**Figure 7. Line of sight used to determine the costs assigned to edges.**

Given an edge  $(u, w)$ , the cost assigned to it may be adequately chosen to be the ratio of the number of vertices in the graph that are simultaneously visible from  $u$  and  $w$  within a radius  $r$  up to which visual or electronic detection is achievable. Figure 7 displays this concept with two adjacent vertices  $u$  (colored yellow) and  $w$  (colored blue) at the center of a graph with each vertex having four neighbors. The translucent cylindrical boundary in yellow and blue have the correspondingly colored vertices in their respective centers and indicate the extents of visibility from  $u$  and  $w$  respectively. The

vertices that are visible to  $u$  and  $w$  are those that are shown lighted (with luminance inversely proportional to distance), while those that are not visible are under dark shadows. The ratio of vertices in this graph lying in the common area enclosed by the two translucent boundaries is then easily confirmed by visual inspection to be  $81/17^2 = 0.28$ .

The computation of LOS ray intersection tests are an integral part of a large number of graphics utilities. The computation load incurred in determining the costs of all the edges in a given graph in this manner is not an undue concern since they may be pre-computed and stored on-disk to be accessed as and when needed. Once a cost scheme has been determined, the method *EstimatePosition* may be applied to obtain a set of candidate destinations for an entity. As in Algorithm 1, these candidate destinations are then the end of the paths that include the path  $P = \{u_1, u_2, u_3, \dots, u_N\}$  observed to have been taken by an entity as a subpath. Furthermore, as in Algorithm 1, the set of candidate destinations derived are likely to be useful only after the application of well-justified pruning heuristics. Some of the candidate heuristics are as follows.

#### Pruning Technique 1

Let  $D$  be a set of feasible vertices derived by Algorithm 2. If an entity is deemed to be traveling with the apparent intent of reducing its exposure, then it is reasonable to assume that an intermediate destination and its surrounding area are not overly exposed when alternatives offering greater concealment exist. If they are, we may choose to prune such vertices from the set of feasible destinations. We may define this idea formally as follows. Let  $c(u)$  be the cost from exposure to surveillance of vertex  $u$  defined as the ratio of the number of vertices visible to  $u$  to the total number within the periphery bounding the extent of visibility from  $u$ . Let  $R_d(u)$  be the set of vertices that are reachable from  $u$  with path lengths less than or equal to  $d$ , for some relatively large  $d \geq 1$ . Then we may apply the test shown in Figure 8 to prune vertices. In this test, the cost of exposure to the zone surrounding each vertex in  $D$  is derived as the average cost of the vertices in that zone. If that cost varies from the mean value for the zones corresponding to all vertices in  $D$  by more than one standard deviation, then that vertex is eliminated. Assuming that the aforementioned costs have a normal distribution, we can eliminate 32% of the vertices in  $D$ , since 68% of the vertices fall within one standard deviation of the mean.

#### **Prune()**

```
{
  total_sum ← 0;
  for each u in D do
```

```
  {
    sum(u) ←  $\sum_{w \in R_d(u)} c(w)$ ;
    total_sum ← total_sum + sum(u);
  }
   $\mu \leftarrow total\_sum / |D|$ ;
   $\sigma \leftarrow \sqrt{\frac{1}{|D|} \sum_{w \in D} (\mu - sum(w))^2}$ 
  if ( $|sum(u) - \mu| > \sigma$ ) then reject u as a destination.
}
```

**Figure 8. Test to prune vertices from a feasible set of destinations.**

It is possible that neither of the decision variables presented governs the path choices of an entity. In that case, *EstimatePosition* will fail to assign any feasible destinations to the set  $D$  and therefore, the aforesaid method will require alternative decision variables with practical merit for selecting prospective paths. Hence choosing a comprehensive set of decision variables beforehand is critical. Some alternatives include, the locations of surveillance assets on the observer's side that are known to the adversary and the locations of supporting arms that may be brought to bear on the adversary.

### **3. A Markov Model**

A Markov model for estimating the future positions of entities samples the state of an entity at discrete time intervals over a given span of time. Assuming that the number of states is finite, we derive the conditional probability of the entity being in a certain state in a given sample, subject to it being in another state in the previous sample. Note that the two states may not necessarily differ. The set of transitional probabilities for each pair of states for the current and previous samples is then used to determine the probability of the entity being in any of those finite states in the future. The potential of this approach is supported by promising research that substantiates the uniqueness of driving behavior of individuals and the use of Bayesian networks for their prediction (Kumagai *et. al.*, 2003, Pavlovic *et. al.* 1999). To appropriately describe this model and the proposed algorithm, we introduce the following definitions.

Let  $x_t$  be the state of a dynamic entity at an instance of time  $t$ . The sequence of states  $\{x_0, x_1, x_2, \dots, x_t, x_{t+1}, x_{t+2}, \dots\}$  of such an entity may be viewed as a stochastic process, where we can predict the occurrence of any state in the future with a probability only. Furthermore, it is intuitive to assume that the probability of occurrence of a

certain state in the future is dependent only on the current state as opposed to any past state. This assumption is the basis for a Markov chain model that may be stated formally as follows. Let  $S = \{s_0, s_1, s_2, \dots, s_M\}$  be the finite set of values that an entity's state can have. Then the probability that the state of the entity  $x_t$  at time  $t$  having the value  $s_i$  is given by the conditional probability dependent on the state of the entity  $x_{t-1}$  at time  $t-1$  having the value  $s_j$ . This may be denoted as  $\Pr(x_t = s_i | x_{t-1} = s_j)$  and is called the transitional probability between state  $s_j$  to  $s_i$ .

Let us assume that if an entity is in state  $s_j$  at time  $t$  the probability of it being in state  $s_i$  at time  $t+1$  is  $p_{ij}$  for all instances of  $t$ . Then  $Q$  represents the transition matrix of the Markov chain, where entry  $(i, j)$  i.e., in row  $i$  and column  $j$ , gives the probability  $p_{ij}$ .

$$Q = \begin{bmatrix} p_{11} & p_{12} & \dots & p_{1\ m-1} & p_{1\ m} \\ p_{21} & p_{22} & \dots & p_{2\ m-1} & p_{2\ m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ p_{m-1\ 1} & p_{m-1\ 2} & \dots & p_{m-1\ m-1} & p_{m-1\ m} \\ p_{m\ 1} & p_{m\ 2} & \dots & p_{m\ m-1} & p_{m\ m} \end{bmatrix}$$

In  $Q$ , each probability  $p_{ij} \geq 0$ . Also the sum of the probabilities in each row must equal 1 for the reason that from a given state with probability  $p_{ij}$  the state of the entity can transition to at most  $m$  states in  $S$ . If an entity has the transition matrix  $Q$ , its probability of being in state  $s_i$  at time  $t+2$  given that it is in state  $s_j$  at time  $t$  is given by entry  $(i, j)$  in the matrix  $Q^2$ . It is easy to see why, since a transition from state  $s_j$  at time  $t$  to state  $s_i$  at time  $t+2$  can transition through any of  $m$  states at time  $t+1$ . Hence the probability of that entity being in state  $s_i$  at time  $t+2$  must equal,

$$\sum_{k=1}^m p_{ik} p_{kj}. \text{ This, of course equals entry } (i, j) \text{ in the matrix } Q^2.$$

In general, therefore the probability of that entity being in state  $s_i$  at time  $t+\delta$  given that it is in state  $s_j$  at time  $t$  is given by entry  $(i, j)$  in the matrix  $Q^\delta$ . Then given the *state probability vector*  $w = (p_1, p_2, \dots, p_m)$  where  $p_j$  represents the probability of that entity being in state  $s_j$  at time  $t$ ,  $wQ^\delta$  gives the probability of that entity being in any of the states at time  $t+\delta$ . But since the state of the entity is known to be in some state  $s_j$  at time  $t$ ,  $w$  is a unit vector with 1 at index  $j$ , and 0 elsewhere.

We now identify some of the prospective candidates for the definition of a state. We may define a state as a tuple  $(x_1, x_2, \dots, x_L)$  where each  $x_i$  is an attribute with a value in

the set  $A_i$ . For example, we may choose  $A_1$  to be the set of discrete velocities assumed by an entity with a granularity of 1 m/sec;  $A_2$  to be the set of discrete acceleration (and deceleration) values assumed by an entity with a granularity of 1 m/sec<sup>2</sup>;  $A_3$  to be the set of discrete orientation values assumed by an entity with a granularity of 1 second;  $A_4$  to be the set of discrete values for the rate of change of orientation assumed by an entity with a granularity of 1 second/sec; and finally,  $A_5$  to be the set of discrete values defining the distance of an entity from a natural or manmade obstacle on its current heading. These are just a few choices for attributes that may be used to define the state of an entity.

The choice of the aforementioned attributes is based on the following reasoning. Human operators generally dictate the dynamics of entities in a contemporary battlefield directly or indirectly, and each operator has unique driving, navigating or piloting habits. The driving characteristics of people on the road provide interesting parallels that include, the potential cruising speed given the current width of a road, the potential rate of deceleration given the current cruising speed, the potential rate of cornering given the current radius of the turn. Similarly, for example, the potential value of an attribute in  $A_4$  given the current value of an attribute in  $A_5$  describes the expected rate of turn given the current proximity of an obstacle to the entity; the potential value of an attribute in  $A_4$  given the current value of an attribute in  $A_1$  describes the expected rate of turn given the current velocity of the entity. Alternatively we may look at the probabilities of the dependency of a complete state comprised of the values of attributes in  $A_1, A_2, A_3, A_4$  and  $A_5$  with the previous state.

### The Feasibility of States Dependent on Environment

At this point we look at the dependency of states containing attributes that are dependent on the terrain or environment such as  $A_5$ . For example, suppose that we are given the states  $s_1 = (a_1, a_2, \dots, a_5)$  and  $s_2 = (b_1, b_2, \dots, b_5)$  where  $a_5$  and  $b_5$  belong to  $A_5$  defining the distance of an entity from a natural or manmade obstacle on its current and future heading respectively. Then the probability of an entity being in the state  $s_2$  in the future given  $s_1$  as its current state implies the entity will be at  $b_5$  in the future given that it is currently at  $a_5$ . Such an implication about the position of an entity with respect to geography is misleading and requires that  $\Pr(x_t = s_2 | x_{t-1} = s_1) = 0$ . For this reason we introduce an *oblivious value* for attributes that make assumptions about the environment. This value is indicated by the variable “\*”. Now we can define the transitional probabilities from arbitrary states to ones with oblivious values for environmentally dependent attributes as not necessarily being zero. For example, if  $s_3 = (c_1, c_2, \dots, *)$ , then we can have  $\Pr(x_t = s_3 | x_{t-1} = s_1) \geq 0$ .

Now if  $w$  is a unit vector defining the current state of an entity. Then  $wQ$  gives the state probability vector for that entity at time  $t + 1$ . Now one or more of the states in  $wQ$  can have non-zero probabilities. The attributes of those states with oblivious values need to be given tangible values to enable the derivation of the state probability vector at time  $t + 2$ . This can be achieved by dead reckoning from the entity's state at time  $t$  using attributes such as its velocity and position.

### Computational Load

Matrix computations are expensive with  $N^2(2N - 1)$  multiplication and addition operations required to multiply two  $N \times N$  matrices. In our case, the matrices are very large. For instance if we construct a Markov model comprised of states defined by just four attributes, each with 10 values, we would have  $10^4$  possible states. Then,  $Q$  would be of size  $10^4 \times 10^4$ , requiring  $1.99999 \times 10^{12}$  operations to derive  $Q^2$ . Note that, contemporary system processors running at over 3 GHz deliver approximately  $10^{10}$  floating point operations per second and would require over 3 minutes to compute  $Q^2$ . In practice, we are likely to have far more attributes and larger sets of discrete values as the domain for each attribute. The problem is further exacerbated by the fact that  $Q^2$  only provides the probabilities of outcomes at the next instance of time. Obtaining a glimpse of the likelihood of outcomes further out magnifies the computational load significantly. For instance, if the state of a dynamic entity were sampled with a periodicity of 5 seconds to obtain the probabilities of outcomes after 5 minutes, we would need to compute  $Q^{60}$ . This task is likely to take well over 3 hours using a single processor. While this is not a prohibitive value, it influences design choices for the implementation of the model and how it plugs into the overall architecture of an overall solution. The pseudo-code in Figure 9 describes the algorithm.

#### **MarkovChainEstimation()**

```
{
  // Initialize the number of observed occurrences
  // of all possible state transitions to zero.
  Step A:
  for all possible  $s_i$  and  $s_j$  do
  {  $sum(s_i, s_j) \leftarrow 0$ ;  $sum(s_i) \leftarrow 0$ ; }
   $past\_state \leftarrow 0$ ;

  // Tabulate the states of the entity across time.
  Step B:
  for time  $t \leftarrow 1$  to  $N$  do
  {
    if ( $current\_state = s_i$ ) and ( $past\_state = s_j$ )
    {
       $sum(s_i, s_j) \leftarrow sum(s_i, s_j) + 1$ ;
    }
  }
}
```

```
}
}

// Compute the state transition probabilities.
Step C:
for all possible  $s_i$  and  $s_j$  do
{  $Pr(s_i, s_j) \leftarrow sum(s_i, s_j)/sum(s_j)$ ; }
Construct  $Q = [q_{ij}]$ , with  $q_{ij} = Pr(s_i, s_j)$ ;

// Compute the state probability vector after time
//  $\Delta T$ . Let sampling frequency be  $f$  Hz, and the
// initial state probability (unit vector) be  $w$ .
Step D:
for  $t \leftarrow 0$  to  $\Delta T/f$  do
{
   $w \leftarrow wQ$ ;
  Dead reckon all oblivious values in  $w$ ;
}
}
```

**Figure 9. The Markov chain computation across multiple transitions.**

As seen earlier, the computational load incurred in estimating the future positions of entities using a Markov model is substantial. Therefore to develop a usable solution using this approach, the computational efficiency of the algorithm needs to be enhanced by employing efficient methods for multiplying sparse matrices and suitable pruning introduced to eliminate infeasible state derivations.

## 4. Conclusions

An automated system providing a capability for accurate path prediction would directly support anticipation of entity movement and position. Geo-position accuracy is a crucial requirement for target acquisition, especially with the employment of precision-guided munitions, and especially in environments where dispersed and fleeting targets are expected. Furthermore, such a capability could help mitigate the fog of war and avoid fratricide by supporting observations of position and movement of friendly and neutral entities through air, land, sea, or space.

## 5. References

- [1] M. Akamatsu, Measuring Driving Behavior, detecting unusual behavior for driving assistance, SICE Annual Conference in Osaka (2002).

- [2] Y. Azar, A. Fiat, A. R. Karlin, F. McSherry, and J. Saia. "Spectral analysis of data," *ACM Symposium on Theory of Computing*, 2000, pp. 619–626.
- [3] P. Dagum, A. Galper, E. Horvitz and A. Seiver, Uncertain Reasoning and Forecasting, *International Journal of Forecasting*, 11, pp.73–87 (1995).
- [4] P. J. Emmerman and U. Y. Movva, Intelligent Agent Battlespace Augmentation, *12<sup>th</sup> International Symposium on Foundations of Intelligent Systems*, October 2000, pp. 12-20.
- [5] P. J. Emmerman, U. Y. Movva and T. Gregory, "Integration of battlefield visualization and agent technology," *Visualization of Temporal and Spatial Data for Civilian and Defense Applications*, SPIE, Vol. 4368, 2001, pp. 9-17.
- [6] U. Kjærulff, A computational scheme for reasoning in dynamic probabilistic networks, *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers, San Mateo, California, 1992, pp.121-129.
- [7] T. Kumagai, Y. Sakaguchi, M. Okuwa and M. Akamatsu, Prediction of Driving Behavior through Probabilistic Inference, *Proceedings of the Eighth International Conference on Engineering Applications of Neural Networks*, 2003.
- [8] S. Nanda, C. L. Lickteig, "Methods for Creating Intermediate Morphologies to Aid 2D to 3D Visualization," *Proceedings of the Spring 2005 Simulation Interoperability Workshop*, Vol. 1, (April 2005), pp. 93 – 102.
- [9] N. Oliver and A. Pentland, Graphical Models for Driver Behavior Recognition in a Smart Car, *IEEE Intl. Conference on Intelligent Vehicles* (2000).
- [10] V. Pavlovic, J. M. Rehg, T-J. Cham, and K. P. Murphy, A dynamic Bayesian network approach to figure tracking using learned dynamic models, *International Conference on Computer Vision*, 1999, pp. 94-101.
- [11] A. Pentland and A. Liu, Modeling and Prediction of Human Behavior, *Neural Computation*, 11, pp.229–242 (1999).
- [12] A. Robles-Kelly and E. R. Hancock, "Steady State Random Walks for Path Estimation", *Syntactical and Structural Pattern Recognition*, Springer-Verlag Lecture Notes on Computer Science, 2004, pp. 143-152.
- [13] Y. Sakaguchi, M. Okuwa, Ken'ichiro Takiguchi, and M. Akamatsu, Measuring and modeling of driver for detecting unusual behavior for driving assistance, *Proceedings of the 18<sup>th</sup> International Conference on Enhanced Safety Vehicles*, 2003.

## 6. Author Biographies

**Sanjeeb Nanda** is a senior research and development engineer with the Advanced Technologies Division of SDS International. His interests span the areas of simulation, biometrics, parallel computing, and fault-tolerance in storage. He is currently pursuing a doctorate in Computer Science at the University of Central Florida.

**Joseph Weeks** serves as a research and engineering psychologist with the Air Force Research Laboratory. His research interests include acquisition of expertise in decision-making and effectiveness of simulation training. He holds a Ph.D. in Educational Psychology from the University of Texas.