# Supplementary Information: Modeling lost person behavior with spatial and temporal decision points

**Amanda Hashimoto[1], Bryson Howell[2], Robert Koester[3,4], and Nicole Abaid[5,*]**

[1]Virginia Polytechnic Institute and State University, Engineering Mechanics Program, Blacksburg, VA 24061, USA
[2]Virginia Polytechnic Institute and State University, Department of Electrical and Computer Engineering, Blacksburg, VA 24061, USA
[3]dbS Productions LLC, Charlottesville, VA 22911, USA
[4]University of Portsmouth, School of the Environment, Geography and Geosciences, Portsmouth, P01 2UP, UK
[5]Virginia Polytechnic Institute and State University, Department of Mathematics, Blacksburg, VA 24061, USA
[*]nabaid@vt.edu

## Map generation

The maps are generated using Python v3.6.8 along with several widely used open-source libraries, as well as elevation and layer data sourced from ArcGIS. The map creation process starts with inputting a GPS coordinate at the center of the target area, an extent in meters to define the size of the output matrix, and a set of ArcGIS feature layers. The resulting output consists of two binary matrices: one representing linear features and another for inaccessibility, where each cell indicates the presence of a linear feature or whether the area is inaccessible. These matrices are kept separate to allow the boundaries of inaccessible regions to function as linear features. In our case, GPS coordinates were derived from 65 specific incidents, with the extent set at 20 km, resulting in $3000 \times 3000$ cell matrices, and the ArcGIS feature layers are listed below.

The process begins by querying each ArcGIS feature layer from its respective URL and parsing it into individual features. The initial set of raw linear features is filtered to exclude any that would not appear in the final output, which is essential since some features may surround but not intersect the target location. The filtered feature set is then interpolated to match the specified extent, ensuring no gaps in the final matrices. If a layer does not provide inaccessibility data (such as walking trails or powerlines), the interpolated features are converted to matrix indices and added to the linear feature matrix. On the other hand, layers that do include inaccessibility information (like rivers or lakes) will also have any enclosed areas interpolated and added to the inaccessible matrix. These areas are determined by creating a 2-D polygon from the original features and checking whether each point falls within it. The final step merges the ArcGIS linear feature matrix with the elevation-based linear features, which are discussed below.

#### *ArcGIS layers for linear features and inaccessible areas*

- Local connecting roads: https://carto.nationalmap.gov/arcgis/rest/services/transportation/MapServer/31
- Local roads: https://carto.nationalmap.gov/arcgis/rest/services/transportation/MapServer/32
- Controlled-access highways: https://carto.nationalmap.gov/arcgis/rest/services/transportation/MapServer/29
- Secondary highways: https://carto.nationalmap.gov/arcgis/rest/services/transportation/MapServer/30
- Railroads: https://carto.nationalmap.gov/arcgis/rest/services/transportation/MapServer/38
- Powerlines: https://services2.arcgis.com/FiaPA4ga0iQKduv3/arcgis/rest/services/US_Electric_Power_Transmission_Lines/FeatureServer
- Walking Trails: https://partnerships.nationalmap.gov/arcgis/rest/services/USGSTrails/MapServer/0

- Rivers: https://hydro.nationalmap.gov/arcgis/rest/services/nhd/MapServer/6
- River interiors: https://hydro.nationalmap.gov/arcgis/rest/services/nhd/MapServer/9
- Lakes: https://hydro.nationalmap.gov/arcgis/rest/services/nhd/MapServer/12

#### *Elevation-based linear features*

The elevation data of the map region is utilized to identify terrain-based linear features, such as mountain crests and drainage lines, as well as inaccessible areas like steep slopes. To detect these elevation-based linear features, we compute the local maxima and minima in the elevation matrix using MATLAB's `islocalmax` and `islocalmin` functions, respectively[1,2]. Given the density of features on the map, a minimum prominence threshold is applied to highlight significant peaks. Prominence measures how much a peak stands out based on its height and surrounding terrain. By setting a minimum prominence of 6 meters, we extract the local extrema in both the $x$ and $y$ directions, creating a binary matrix. This matrix is then merged with the linear feature matrix generated from the ArcGIS layers. To identify elevation-based inaccessible areas, we calculate the slope by determining the gradient of the elevation matrix. This is achieved using MATLAB's `imgradient` function with the Central Difference method[3]. The gradient at each pixel is a weighted difference between its neighboring pixels. For example, in the $y$-direction, the gradient is calculated as $dI/dy = (I(y+1) - I(y-1))/2$, where $I$ represents the elevation matrix. The slope is then determined by dividing the gradient magnitude by the cell size (6.67 meters). Areas with slopes exceeding 45 degrees are classified as inaccessible, and these locations are added to the inaccessible matrix.

#### *Spatial decision points*

To identify spatial decision points on the map, we use several image processing techniques to analyze the linear features and topography. The process starts by skeletonizing the binary matrix of linear features using the `bwmorph` function in MATLAB, which reduces the linear features to their essential structure[4]. This allows us to focus on the key terrain lines, such as ridges and valleys, without unnecessary details. Next, we identify **branch points** within the skeletonized matrix. These points, where multiple linear features intersect, represent decision points where individuals navigating the terrain may encounter multiple possible paths. The MATLAB function `bwmorph` with the `'branchpoints'` option is used to detect these junctions.

In addition to branch points, we also identify **endpoints** as the terminal locations of linear features where paths or terrain features come to an end. These are also considered decision points, as individuals would need to decide whether to continue or change direction. The `'endpoints'` option in `bwmorph` is used to find these locations.

We then incorporate **saddle points**, which occur where the terrain has a mix of local maxima and minima, such as in mountain passes or ridge lines. These are often natural decision points where the terrain dips between two peaks. Saddle points are identified by combining the local maxima and minima matrices, with the intersections marking their locations.

When applying `bwmorph` to the local elevation linear features (derived from the maxima and minima in the elevation matrix) to find the endpoints and branch points, the results were problematic. This might have been due to excessive resolution, as it produced an overwhelming number of endpoints. Features that should have been simple lines with two endpoints were instead fragmented into multiple segments. To address this issue with the local elevation matrix, we implement a workaround by splitting the matrix into smaller sections and processing each one individually.

First, we divide the binary matrix of local elevation features into smaller $9 \times 9$ matrices using MATLAB's `mat2cell` function[5]. Next, for each $9 \times 9$ matrix, we identify the spatial coordinates of the linear features and visualize them using a scatter plot. We save this visualization as an image file, which is then converted into a grayscale format and resized to match the dimensions of the smaller matrices. This conversion enables us to standardize the data while maintaining important spatial details. To refine the feature extraction, we adjust the grayscale image by setting pixel values of 255 to 0 and all others to 1. To ensure accuracy, we also remove boundary artifacts by setting the axes pixels (edges) to 0, effectively making the boundaries white.

With this refined matrix, we skeletonize the features using `bwskel` with a minimum branch length of 5, ensuring that only significant linear features are retained[6]. From this skeleton, we use `bwmorph` to identify both endpoints and branch points, or locations where the linear features either terminate or split.

To further clean up the data, we remove endpoints that were too close to the boundaries, as these could be artifacts of the division into smaller matrices. The remaining endpoints and branch points are then saved in a cell array for each $9 \times 9$ matrix.

Finally, after processing all the smaller matrices, we recombine them using `cell2mat`, resulting in a final unified matrix of endpoints and branch points[7]. This approach allows us to accurately detect decision points in the terrain while avoiding the excessive fragmentation that occurred in the original method.

To finalize the identification of decision points, we first examine the intersection of linear features by combining all the resulting matrices, including the local elevation features and ArcGIS layers such as rivers, roads, powerlines, lakes, trails, and highways. These layers are added together to create a composite matrix of all potential intersecting features. Any location where two or more features overlap is considered significant, so we set a threshold to mark these intersections as decision

points. Specifically, if the sum of overlapping features at any point is two or more, we retain that point as a decision point, and all others are set to zero.

Lastly, we combine all decision points by merging the branch points, endpoints, saddle points, intersection points, and the previously identified local elevation endpoints and branch points. This results in a unified matrix of decision points, where any non-zero value represents a decision point. We then simplify the matrix by converting all non-zero values to one, creating a clean binary matrix of decision points.

The map generation scripts are available at: https://github.com/ahashimoto13/LostPersonModelwithDecisionPoints.git.

## Model algorithm

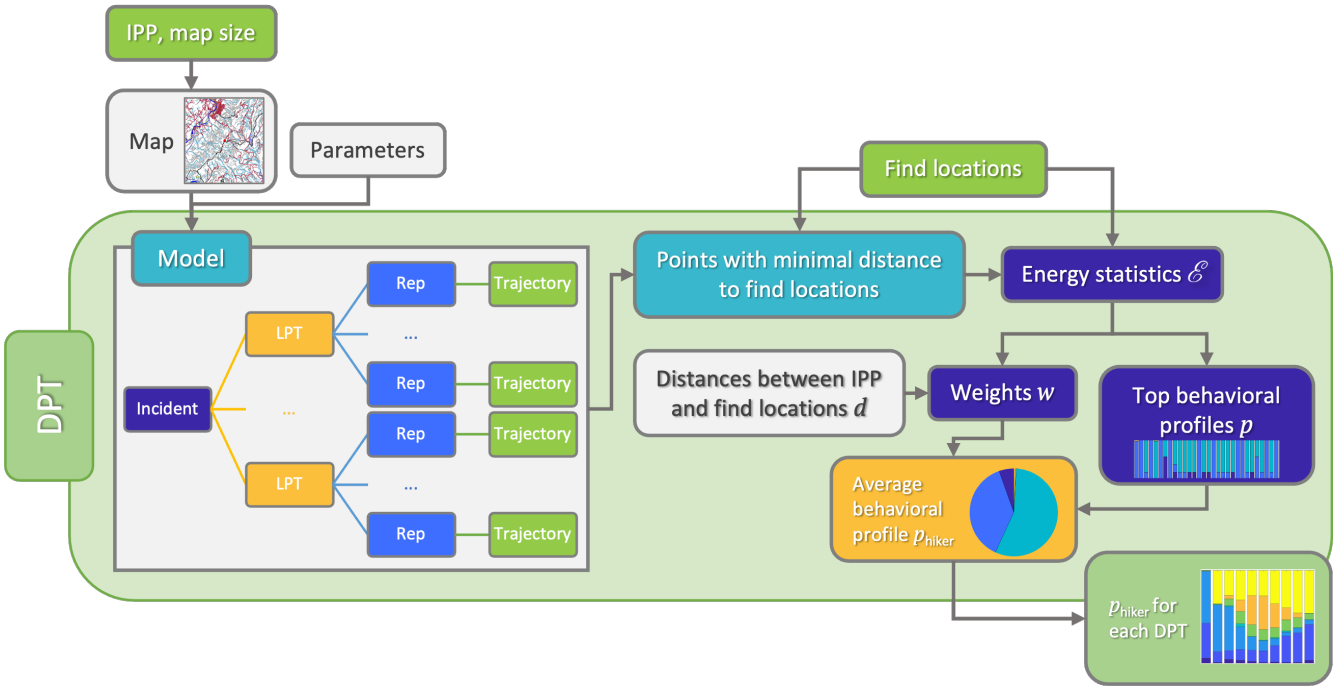


**Figure S1.** Visual schematic of the model algorithm.

## Model description using ODD protocol

### Overview

#### *Purpose*

The model represents how lost persons (LPs) navigate wilderness terrain, incorporating spatial and temporal decision points to capture when behavior changes occur. It is designed to test whether combining real WiSAR data with terrain features improves predictive accuracy, and to evaluate the importance of including contouring, a behavior that is novel to this model.

#### *State variables and scales*

- **Entities**: one lost person agent.
- **State variables**: grid position $(x, y)$, behavior state [RW, RT, DT, SP, VE, BT, CT], time elapsed, decision point status.
- **Environment**: $3000 \times 3000$ grid cells (6.67 m resolution) with elevation, linear features (roads, trails, rivers, crests, etc.), and inaccessible areas (water, slopes $> 45°$).
- **Temporal resolution**: discrete time steps (850 steps/hour) for up to 50 hours.

#### *Process overview and scheduling*

At each step:

1. agent checks current state and environment,
2. if not at a decision point, continues current behavior,
3. if at a spatial or temporal decision point, selects a new behavior using an independent realization of the probability mass function for the lost person type (LPT),
4. behavior submodel determines next move,
5. trajectory is recorded.

### Design concepts

- **Emergence**: trajectories arise from local decisions and terrain constraints.
- **Adaptation**: behaviors can change at decision points.
- **Sensing**: agents detect local terrain and past route for backtracking.
- **Stochasticity**: behaviors are sampled probabilistically from LPT distributions.
- **Observation**: outputs include simulated trajectories, distances, and overlap with real WiSAR tracks.
- **Collectives**: single-agent focus, but validation compares across many real LP trajectories.

### Details

#### *Initialization*

Simulation begins at the Initial Planning Point (IPP) of an incident. Each run initializes an agent with an LPT-specific probability distribution and an environmental map containing terrain and features.

#### *Input*

- **Terrain data**: USGS GIS layers for elevation and land cover.
- **Incident data**: ISRID records with LPT, IPP, and find location.
- **Route data**: augmented dataset of real WiSAR trajectories for validation.

***Submodels***

- Random Walking (RW): random step to one of 9 neighboring cells, including the current state.
- Route Traveling (RT): follow linear feature along direction of motion.
- Direction Traveling (DT): move in fixed compass direction.
- Staying Put (SP): remain in place.
- View Enhancing (VE): move to neighboring cell with highest elevation.
- Backtracking (BT): retrace previous path.
- Contouring (CT): move to neighboring cell minimizing elevation change along direction of motion.
- Moves constrained by inaccessible cells.

## Formal definition of behavioral submodels

To ensure reproducibility, we provide an explicit definition of each behavioral update rule within the $3 \times 3$ body-coordinate grid centered at the agent's current position $x(t)$. At each time step, the sequence of operations is as follows: (i) determine whether a behavioral update is triggered by a spatial or temporal decision point; (ii) select the active behavioral strategy if an update is triggered, otherwise retain the previous strategy; (iii) compute the provisional update $\hat{x}(t+1)$ according to the selected behavior; (iv) apply the smoothing update from Eq. (1); and (v) enforce inaccessibility constraints. Importantly, the lost person type probability mass function is evaluated only when a spatial or temporal decision point is triggered. Between decision points, the previously selected behavioral strategy remains active.

Let

$$\mathscr{N}(x(t)) = \{x(t) + \delta : \delta \in \{-1,0,1\}^2\}$$

denote the set of neighboring cells, which includes the eight adjacent cells and the current cell. At each time step, a provisional update $\hat{x}(t+1) \in \mathscr{N}(x(t))$ is selected according to the active behavioral strategy. The smoothing update described in Eq. (1) of the main text is then applied to obtain $x(t+1)$.

For all behaviors, if the provisional update $\hat{x}(t+1)$ corresponds to an inaccessible cell, the agent remains at $x(t)$ for that time step.

**Random Walking (RW)** The provisional update is selected uniformly from $\mathscr{N}(x(t))$.

**Route Traveling (RT)** Let $\mathscr{L}(x(t)) \subset \mathscr{N}(x(t))$ denote the subset of neighboring cells that contain a linear feature and are aligned with the current direction of motion in body coordinates.

If $\mathscr{L}(x(t)) \neq \varnothing$, the provisional update is selected uniformly from $\mathscr{L}(x(t))$. If no such cells are present, the behavior defaults to Random Walking.

**Direction Traveling (DT)** The provisional update is selected as the forward cell in body coordinates, consistent with the agent's previous velocity direction. If this cell is inaccessible, the agent remains stationary for that time step.

**Staying Put (SP)** The provisional update is defined as $\hat{x}(t+1) = x(t)$.

**View Enhancing (VE)** Let $z(x)$ denote the elevation at cell $x$. The provisional update is selected as

$$\hat{x}(t+1) = \arg \max_{y \in \mathscr{N}(x(t))} z(y).$$

If multiple cells share the maximum elevation, one is selected uniformly at random.

**Backtracking (BT)** If the previous behavior was not backtracking, the provisional update is the most recently visited position prior to $x(t)$. If the previous behavior was also backtracking, the agent continues retracing its path to the most recent non-backtracking position in its stored trajectory history.

**Contouring (CT)** Under contouring, the agent evaluates only the three forward-facing cells in body coordinates within the $3 \times 3$ grid centered at its current position. Let $\mathscr{F}(x(t)) \subset \mathscr{N}(x(t))$ denote this set of forward-facing cells.

For each candidate cell $y \in \mathscr{F}(x(t))$, compute the elevation difference

$$\Delta z(y) = |z(y) - z(x(t))|.$$

Define the admissible set

$$\mathscr{C}(x(t)) = \{y \in \mathscr{F}(x(t)) : \Delta z(y) \leq \varepsilon\},$$

where $\varepsilon$ is a fixed elevation threshold.

If $\mathscr{C}(x(t)) \neq \varnothing$, the provisional update is selected uniformly from $\mathscr{C}(x(t))$. If no forward-facing cells satisfy the elevation threshold, the agent remains at its current position for that time step.

## Average behavioral profile for each temporal decision point time

**Table S1.** The average behavioral profile for a hiker as percentages of the 7 behavior strategies [RW, RT, DT, SP, VE, BT, CT] for each decision point time (DPT).

| | Behavior strategy* | | | | | | |
|---|---|---|---|---|---|---|---|
| **DPT** | RW | RT | DT | SP | VE | BT | CT |
| control | 5.4269 | 37.7275 | 55.9401 | 0.3330 | 0.5724 | 0 | 0 |
| 4 s | 0.8848 | 11.4830 | 50.7226 | 0.2125 | 0.8324 | 0 | 35.8647 |
| 1 min | 4.1935 | 9.3091 | 47.2459 | 1.3624 | 7.2728 | 3.7822 | 26.8341 |
| 5 min | 3.1636 | 11.6177 | 24.4450 | 3.4069 | 13.2135 | 12.3151 | 31.8386 |
| 15 min | 2.0338 | 11.8751 | 14.3543 | 0.7303 | 12.5764 | 31.3790 | 27.0500 |
| 30 min | 0.7112 | 12.5287 | 10.8412 | 0.7611 | 11.4206 | 36.3108 | 27.4265 |
| 1 hr | 1.3592 | 17.7073 | 7.3592 | 1.1900 | 10.7636 | 25.7551 | 35.8656 |
| 3 hr | 0.6199 | 28.6752 | 4.2979 | 1.2084 | 12.0551 | 12.8703 | 40.2731 |
| 12 hr | 0.7381 | 32.0658 | 4.8459 | 0.8147 | 10.7813 | 4.8669 | 45.8872 |
| 50 hr | 3.1635 | 38.6049 | 5.1419 | 0 | 7.0688 | 0 | 46.0208 |

**RW = Random Walking, RT = Route Traveling, DT = Direction Traveling, SP = Staying Put, VE = View Enhancing, BT = Backtracking, CT = Contouring*

## References


1. The MathWorks Inc. Find local maxima - MATLAB islocalmax (2024).
2. The MathWorks Inc. Find local minima - MATLAB islocalmin (2024).
3. The MathWorks Inc. Find gradient magnitude and direction of 2-D image - MATLAB imgradient (2021).
4. The MathWorks Inc. Morphological operations on binary images - MATLAB bwmorph (2024).
5. The MathWorks Inc. Convert array to cell array whose cells contain subarrays - MATLAB mat2cell (2024).
6. The MathWorks Inc. Reduce all objects to lines in 2-D binary image or 3-D binary volume - MATLAB bwskel (2024).
7. The MathWorks Inc. Convert cell array to ordinary array - MATLAB cell2mat (2024).