OVERVIEW

Purpose

This model aims to mimic human movement on a realistic topographical surface. It allows the user to explore three different ways in which an agent can choose an easy route to reach a certain goal, and explore multiple different scenarios. Most least-cost path tools available in GIS explore similar issues, but they work on the assumption that the whole world is perfectly known. They find the easiest route among all possibilities, and direct the agent to follow it. This model is different in that the agent does not have a perfect knowledge of the whole surface, but rather evaluates the best path locally, at each step, thus mimicking imperfect human behavior more accurately. Moreover, it allows exploring seven setup scenarios and three different optimization processes, with the simple change of parameter values.

Entities, state variables, and scales

Each run accommodates one agent – called 'hiker' henceforth – and their goal. Only the hiker can move and interact with their surroundings. The goal is there for visualization only. The geographical locations of both hiker and goal can be determined randomly or be based on given coordinates.

The model requires a DEM to create the landscape on which the simulation takes place. The model works best for DEM at 1km resolution, but can accommodate 100m to 2km resolution DEM. For all patches of any given slope, the model assumes that the "inclined surface" covers $1km^2$ (see Figure 1).

The global state variables are presented in Table 1, the patch state variables in Table 2, and the state variables of the hiker in Table 3.

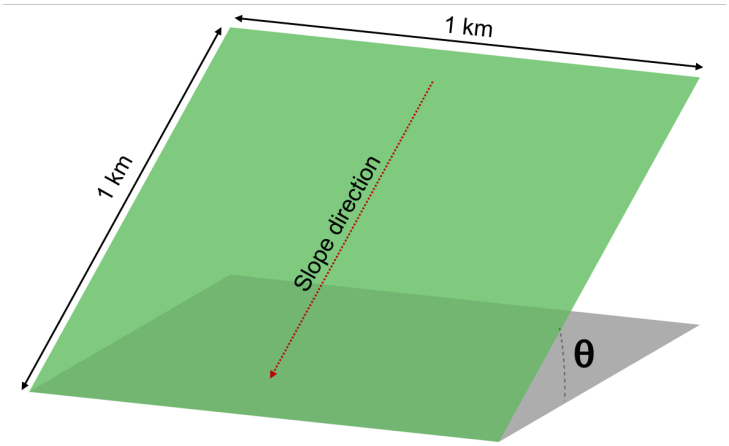


Figure 1. *For all patches with slope θ, both sides of the inclined surface (green) are assumed to measure 1km.*

Table 1. *Global state variables. *Change during the simulation*

Variable name	Description
map-resolution-km	User defined: The resolution of the DEM (km).
patch-size-km	User defined: The resolution at which the user wants to run the model (km).
snow-line	User defined: The elevation of snow cover on mountains, if applicable (m).
mode	User defined: Choose to define how the start and end points are determined.
create	User defined: Choose to create the agent or the goal.
optimization	User defined: Choose what type of path the agent will take.
switchbacks	User defined: Choose to allow bigger switchbacks during travel.
radius-km	User defined: The distance of the goal from the agent (Import mode only)
viewshed-threshold	User defined: Choose to allow the agent to walk over saddles.
outputs?	User defined: Choose to produce an output of the path plot or not.
lost-outputs?	User defined: Choose to produce outputs even if the agent dies while walking.
iter-start	User defined: For import and grid modes. Identifies the position of the start site in the shapefile.
iter-end	User defined: For import and grid modes. Identifies the position of the goal in the shapefile.
shp-var	User defined: For import mode: The name of the shapefile column where labels are stored (if applicable).
grid-size	User defined: For grid mode: The interval of sites required.
basemap	Takes on the imported elevation values – used with GIS extension.
goal	Identifies the ID of the goal.
hiker-n	Identifies the ID of the agent.
coord-start	Identifies the x and y of the agent.
coord-end	Identifies the x and y of the goal.
crow-fly	Calculates the Euclidean distance between start and end points (km).
dist-traveled*	Used to keep track of the distance traveled by the agent.
min-elev	Minimum elevation of the DEM (m).
max-elev	Maximum elevation of the DEM (m).
elev-hiker*	Records the elevation of the agent to help run the viewshed analysis.
list-slope*	Records the slope of walked cells (for sensitivity analyses).
switch	Gives a numerical value to the 'switchbacks' user switch.
sites	Import mode only. Records the coordinates of the imported sites shapefile.
sites2	Import mode only. Records the coordinates of the 2nd imported sites shapefile.
origin	Records the patch ID of the starting point (for outputs).
id-start	Import mode only. Identifies the label of the start site (shp-var).
id-end	Import mode only. Identifies the label of the goal (shp-var).
res-m	Calculates patch resolution in meters.
time-wd*	Takes on the time-walked value (for BehaviorSpace outputs).
hiker-status*	Records if the agent is dead or alive at the end of a simulation.
file-1	To create the outputs.
stamp1	A random float number that can be used to analyze outputs

Table 2. *Patch state variables. *Change during the simulation*

Variable name	Description
elevation	Elevation (m) above sea level. Imported as ASCII from DEM.
effective-slope*	Effective slope (degree) of the cell. Calculated by the neighboring patch from which the hiker would arrive.
abs-eslope*	Absolute effective slope.
inter-elevation*	Records the elevation at a corner, interpolated from nearby patches using IDW
headed-to*	Angle towards the hiker. Calculated by the patch.
dist-to-goal*	Calculates the distance between itself and the agent's goal.
occupied-by*	Records the ID of the agent when they walk on the patch.
len*	Calculates the distance from the patch center to the agent.
time*	How fast the patch can be reached by the agent (seconds).
speed*	Calculated speed at which the hiker can reach it.
water	Distinguishes between land patches and ones covered in water or snow.
patch-counter*	Records how many ticks since being walked on.
obstructed?*	Determines if the patch is hidden from the agent's point of view.
site?	Import mode only. Identifies patches that have sites on them.

Table 3. *Hiker state variables. *Change during the simulation*

Variable name	Description
hiker-dist-to-goal*	Distance between the agent and their goal (as the crow flies).
patch-vision*	Set of patches visible by the agent facing a certain direction.
winner-patch*	The best patch to move to.
time-walked*	Calculates the time walked to the goal.

The length of a simulation depends on the distance between the agent and their goal, the ruggedness of the terrain, the chosen mode, the chosen optimization, as well as the *switchbacks* and *viewshed-threshold* parameter settings. The model stops when the agent reaches their goal, or if they get lost and die.

Process overview and scheduling

At each tick, the hiker evaluates the direction in which they need to travel to reach their goal. At all times, the hiker has a temporary local target that allows them to move slowly in the right direction with minimal effort. At the beginning of each tick, the hiker evaluates if their temporary target (called *winner-patch*) has been reached. The hiker can look for a new temporary target only when they reach its current one. If the goal is within sight (in the immediate neighbors), the hiker chooses the goal as their temporary target. If the goal is still far, they look at a certain number of patches in a vision cone, and identify which ones they will choose based on the chosen optimization.

At every step, the hiker evaluates nearby patches and identifies a set that could potentially be a temporary target. Each of those potential patches calculates the effective slope and speed

required to reach its center. The formula used can be found in the subprocedure explanations below.

Patches that are potential temporary targets change their status throughout the run. They become grey for visualization purpose; however, they regain their green hue at the end of the simulation. Each of them takes on a new slope and speed value as the hiker moves near them, as that value is calculated based on where the hiker is. Some of them acquire an interpolated elevation value if they are crossed at one of their corner – on the way to a knight patch. When the agent walks on a patch, that patch becomes *occupied-by* and cannot be chosen again as a potential path for the next 20 ticks. This is implemented so that the agent moves forward and does not get stuck circling around the goal. The path followed by the agent is represented as a red line.

DESIGN CONCEPT

Basic principles

This model tries to mimic human behavior in a topographical environment. It aims to go beyond the Dijkstra's GIS approach to least-cost path that requires perfect knowledge of the whole environment to identify the ONE best path between two points. It relies on the work of Naismith (1892, in Aitken 1977) and Langmuir (1984) on walking time expenditure in rugged environments.

Emergence

Natural switchbacks occur because of the hiker's aim to get closer to the main goal while choosing a relatively easy route. As some of the patches are chosen at random within a reduced set of possibilities, this model can produce very different paths in certain landscapes. Moreover, contrary to standard LCP tools, the hiker can sometimes fail to find a path between two points.

Adaptation

The hiker can create switchbacks and change direction completely when they find that the easy route is leading away from the goal. The hiker always tries to minimize the slope they walk on, within a certain range. Depending on the optimization chosen by the user, the hiker can also try to maximize speed, minimize distance, or explore their surroundings.

Objectives

The hiker's decision is influenced by the effective slope and the speed at which they can walk to a patch, and by that patch's distance to the main goal. These two values sometimes provide different results.

This model was created to allow researchers to explore possible paths between two points in a more realistic way than standard available LCP tools. The stochasticity written in the model allows this exploration, and allows compiling the different results of multiple runs to create least-cost path corridors and/or evaluate the patches that would most likely have been walked on.

Learning

Patches that are considered by the hiker as potential targets update their color as well as their *effective-slope*, *abs-eslope*, *elev-change, inter-elev*, *time*, and *speed* values as a function of the agent's position on the landscape.

Patches change their *occupied-by* and their *patch-counter* when they are walked on. The agent changes their temporary target when they reach it.

Prediction

The hiker looks at patches in a cone of vision that covers up to 2.5 patches. This implies that they can predict what the topography will be like further along the way and decide which path they will take based on that information. The depth of the cone is realistic on a flat terrain on maps at 1km resolution, and prevents the hiker from seeing too far ahead, as the path choice needs to remain local. While it may not be realistic with higher resolution maps (the hiker would see only 250m ahead on a 100m resolution map), changing the cone depth based on map resolution would increase complexity considerably, and therefore, it is not considered at this time.

In mountainous terrain, the hiker does not consider patches that are obstructed from view by a higher patch, even if it is within the vision cone. However, changing the *viewshed-threshold* parameter allows increasing the threshold to see 'above small saddles', which allows the hiker to climb small mounds when needed.

Sensing

The hiker can access the *dist-to-goal* value of their surrounding patches, which allows identifying which patches lead towards their goal. They use the *abs-eslope*, and *speed* values of the potential patches to choose the one that will be their temporary target.

To calculates the speed and effective slope required by the hiker to reach potential patches, each potential patch asks the hiker's patch to calculate those values and then assign the values to itself. This allows calculating the cost of travel from the hiker's perspective. Sensing is errorless for both hiker and patches.

Interaction

The agent interacts indirectly with the patches as they evaluate their distance and walking cost. The patches that are considered potential temporary targets ask the patch on which the agent stands to communicate its new-slope value so that the walking cost includes both patches (as well as the patch(es) between the two, when applicable).

Stochasticity

The level of stochasticity varies based on the optimization selected. Exploration produces the most varied set of paths, whereas Distance usually produces only a few different ones. The stochasticity is set in the temporary target choice made by the hiker. See the *find-least-cost-path* subprocedure description below for detailed information.

Collectives

No collectives.

Observation

The interface provides several outputs that help follow the dynamics of the model. *Crow-fly* calculates the distance between the start and end points as the crow-flies. It can be compared to the *Distance walked (km)* value that is updated during a run. *Time passed* shows the number of hours spent walking at a pace determined by the effective-slope of the path. *Speed* divides the distance walked by the time passed to provide a km/h speed value. On flat terrain, this value should be 5km/h (Langmuir 1984). The plot presents the evolution of the values that are exported in a CSV file if *outputs?* is set to 'On'. *Path-x* and *path-y* record the coordinates of the path traveled by the agent. The *path slope* plot focuses on the effective slope of the walked-on patches. The *n red patches* box identifies the number of potential start-end points when using the grid mode. This can help use the correct iteration numbers in BehaviorSpace.

The different setup modes allow the user to create different scenarios, and evaluate the impact of changing *optimization*, *switchbacks*, and *viewshed-threshold* on the paths created.

On the interface, the user can also see the coordinates of the start and end points (*coord-start* and *coord-end*). This can be useful when using the 'manual' setup mode to create a defined set of coordinates that can then be used over multiple simulations. The coordinates of a point are updated as soon as it is placed on the landscape. The coordinates can then be entered manually in the *start-x*, *start-y*, *end-x*, and *end-y* boxes to use the 'repeat' mode.

DETAILS

Initialization

The world is set on a grid matrix with the origin located at the bottom left corner. The world does not wrap around. The patch size of the viewer changes based on the resolution of the maps provided and required.

The elevation of each patch is determined by the *DEM* map provided, which needs to be in ASCII format, be named 'DEM.asc', and be placed in a folder named *LCP_maps* located in the

same folder as the model itself. Before setup, the user needs to tell the model the resolution of the map provided (in km) in the *map-resolution-km* box. The user can then decide to change the resolution of the landscape in the model by using the *patch-size-km* slider. When *patch-size-km* differs from *map-resolution-km*, the elevation of patches is interpolated using the bicubic_2 method. The user needs to identify the elevation of the snow line, which will remove the snowy patches from walking possibilities.

The next step is to place one hiker and one goal on the landscape; however, this has to be defined before pressing *setup*, as their position depends on the *mode* chosen by the user. In *import-1* and *import-2* modes, the model imports the geographical coordinates of the site shapefile(s) provided – needs to be called 'Sites.shp' and 'Sites2.shp', and be placed in the same folder as the DEM – and it uses the iter-start and iter-end values to place the hiker on the site which order corresponds to the iter-start value and the goal on the site which order corresponds to the iter-end value. The order refers to the order in which each site is positioned in the shapefile (look at the attribute table). If two shapefiles are imported (import-2), the iter-start value refers to the order in the first shapefile (Sites), and iter-end to the order in the second shapefile (Sites2). This allows creating paths between a select set of sites to a different set of sites. In grid mode, the user needs to enter the interval at which it wants to create start and end points using the grid-size slider. The model will then place the hiker on the *iter-start*'nth point on the grid, and their goal on the *iter-end*'nth point on the grid. In *manual* mode, the user needs to click on the *Create context* button, choose one of the *create* options, and click on the desired patch to create each turtle type. There can only be one of each. In *random* mode, the hiker and the goal are placed randomly on a patch that is not covered by water or snow. In *repeat* mode, the user needs to enter the desired coordinates into the boxes *start-x*, *start-y*, *end-x*, and *end-y*. Finally, in *start-radius* mode, the user needs to enter the desired start coordinates into the *start-x* and *start-y* boxes, as well as a distance in km in the *radius-km* box. The model will then place the hiker at the start coordinates given, and the goal *radius-km* from the hiker.

At setup, the goal communicates its position to the *goal* and the *coord-end* global variables. The hiker updates the *agent-n*, *origin*, and *coord-start* global variables with its number ID and the ID of the patch where it stands, respectively. Those global variables can then be used by both turtles and patches throughout the code. The reason behind the creation of the *agent-n* global variable is simple: in the *manual* setup mode, square turtles are created and killed to show the movement of the mouse in the world. Depending on the time required to position the agent, their ID number might vary, and thus they could not be called directly in the code. *Coord-start* and *coord-end* are shown on their respective interface windows.

At setup, the model calculates the distance between start and end points (in km), which is shown in the *crow-fly* window.

Input data

Elevation and sites coordinates are imported into the model in ASCII and shapefile format, respectively. Both should be in the same projection.

Submodels

To draw:
Use during setup if mode is set to manual to select the start and end points of the model.

To stp-hiker:
Creates the hiker, represented as a red figure, and projects their coordinates into the *coord-start* window. The hiker's pen is down to follow their path.

When created, the hiker populates the *hiker-n*, *origin*, and *coord-start* variables, as mentioned above. They also select the patch where they stand as their first winner-patch.

If the goal has already been set, the model calculates the distance between the hiker and their goal, and reports it in the *crow-fly* window.

To stp-goal:
Creates the goal, represented as a blue house, and projects its coordinates into the *coord-end* window.

When created, the goal populates the *goal* and *coord-end* global variables, as mentioned above.

If the hiker has already been created, the model calculates the distance between the hiker and their goal, and reports it in the *crow-fly* window.

To go:
If using the grid mode and *iter-start* and *iter-end* are set to the same number, the hiker dies, and the model stops.

The observer asks if the hiker is still alive or if the model has reached the tick limit. If the hiker is dead or the limite has been reached:

- The patches update their colors
- An output is created if the *lost-outputs?* parameter is set to 'On'.
- The model stops.

The observer asks all patches that have been walked on recently (within the last 20 ticks) to update their counter, by subtracting 1 to their current value.

The hiker:

- Updates their distance from the goal.
- Updates the *elev-hiker* global variable with their current elevation – set at 1.75m above the elevation of their patch, which is the default value used in GRASS r.viewshed.
- Sets the patch where they stand as *occupied-by* itself. That patch registers the ID of the agent, and sets its counter to 20.
- If the distance to the goal is 0:
    - the grey patches regain their green colors,
    - an output is created if *outputs?* is 'On',
    - the hiker-status is updated to 'alive',

    - and the model stops.
- If they are on the previous *winner-patch* (temporary target), the hiker evaluates how far they are from the main goal.
- If the main goal is an immediate neighbor:
    - the goal is now the temporary target (*winner-patch*),
    - the hiker's patch calculates how fast the hiker can travel to it,
    - and moves to it.
- If the main goal is further, the hiker looks for an easy route using the find-least-cost-path procedure.
- Ticks are updated to tick + 1.

To find-least-cost-path:
A few temporary global variables are defined. For example, one temporary variable records the ID of the patch under the hiker. Another records the distance between the hiker and the goal.

The hiker:
- Faces the goal.
- Identifies unused patches in a cone of 200° and a depth of 2.5 patches.
    - This excludes the patch on which the agent stands, water- and snow-covered patches, as well as patches that were walked on in the past 20 ticks. It also excludes patches that are farther from the goal by the measure $a + (a*b)$, where a is the hiker's distance from the goal and b represents the switchback value defined by the user. If switchbacks is 'On', the value is 0.05, thus increasing the possible distance value by 5% - e.g., a hiker located 50 patches from the goal will consider patches that are up to 52.5 patches away from the goal. Closer to the goal, at 3 patches, for example, the hiker will only consider patches ≤ 3.15 from the goal. If it is 'Off', the value is 0.
    - Asks all potential patches to identify if they are obstructed from view by a middle patch(es) using the check-viewshed procedure.
    - Removes obstructed patches from the potential ones. If this leaves no more potential patches, the hiker extends their search to a 360° cone of vision. They repeat the above steps to the extended patchset.
- Creates a countdown of 5 ticks that gets updated when no potential patches are found.
    - When the counter gets to 5, the hiker dies.
- Asks the potential patches to change their color to grey and to calculate their distance from the hiker in meters.
    - The potential patches than ask the patch under the hiker to calculate the distance between the two, as well as the effective slope and speed required to reach its center.
    - As this is a simple potential patch, it sets its *boolean* value to 'false' (used in calculate-speed procedure).
    - The potential patches take the calculated *effective-slope*, *abs-eslope*, *speed*, and *time*. As each potential patch asks this one at a time, this allows the hiker's patch to calculate the cost to reach each patch individually, and reflects the cost that the hiker would have to go through – rather than ask potential patches to calculate the cost of moving to the hiker, which would be incorrect.

- Divide the potential patches into 3 categories based on their absolute effective slope (*abs-eslope*). This is inspired by the slope separations of Naismith's rule. It does not differentiate between negative and positive slopes, which is inaccurate. However, as there is no defined negative equivalent to the cost of a 5° positive slope, we deemed this classification sufficient. Moreover, reducing the effective slopes to their absolute value allows the hiker to try more slope variants than if thresholds differed by orientation.
- Chooses the temporary target among all potential patches. This choice differs based on the Optimization parameter setting.
    - Distance:
        - Chooses the patch that brings them closer to the goal within the following categories: looks first within the patches with *abs-eslope* < 5°. If not possible, looks within the patches with *abs-eslope* < 12°. If still not possible, chooses the patch with lowest distance to the goal, regardless of slope.
    - Exploration:
        - Chooses one of the patches from one of the same categories: looks first within the patches with *abs-eslope* < 5°. If not possible, looks within the patches with *abs-eslope* < 12°. If still not possible, chooses the patch with the lowest slope.
    - Speed:
        - Identifies the patch(es) that can be reached the fastest (max *speed*).
        - Chooses the patch that brings them closer to the goal within those fast patches.
- Asks the patch where they stand to re-calculate the effective slope and speed required to reach the identified winner-patch. As this is for the winner patch, it sets its *boolean* value to 'true' (used in calculate-speed procedure). The patch under the hiker keeps the calculated values for further computation.
- If the hiker did not identify a new *winner-patch*, they die.
- If the hiker has a new *winner-patch*, they:
    - Update the list-slope list with the effective-slope of all patches walked on.
    - Face winner-patch.
    - Call the move procedure.

To check-viewshed:
This procedure is called by the potential patches considered by the hiker, and it is used to identify if the patch is hidden from the hiker's view by another higher patch. Middle patches crossed on the way to a knight patch use 2 reporters that interpolate their elevation at the walked corner, using Inverse Distance Weighting (IDW) on the elevation of nearby patches. This procedure also uses 3 reporters to identify the threshold at which a middle patch would block the view of the hiker. These reporters are detailed after this subprocedure description.

Figure 2 demonstrates how the distance between potential patch and hiker affects the viewshed threshold for each patch in between.

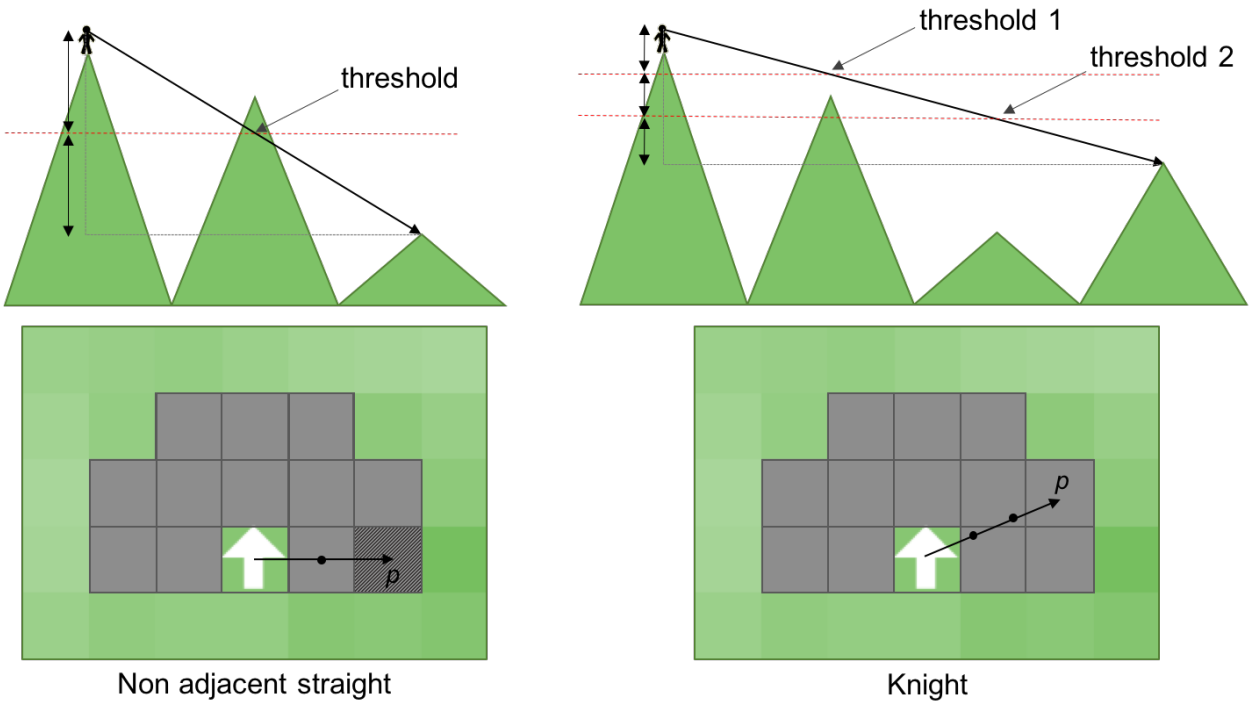
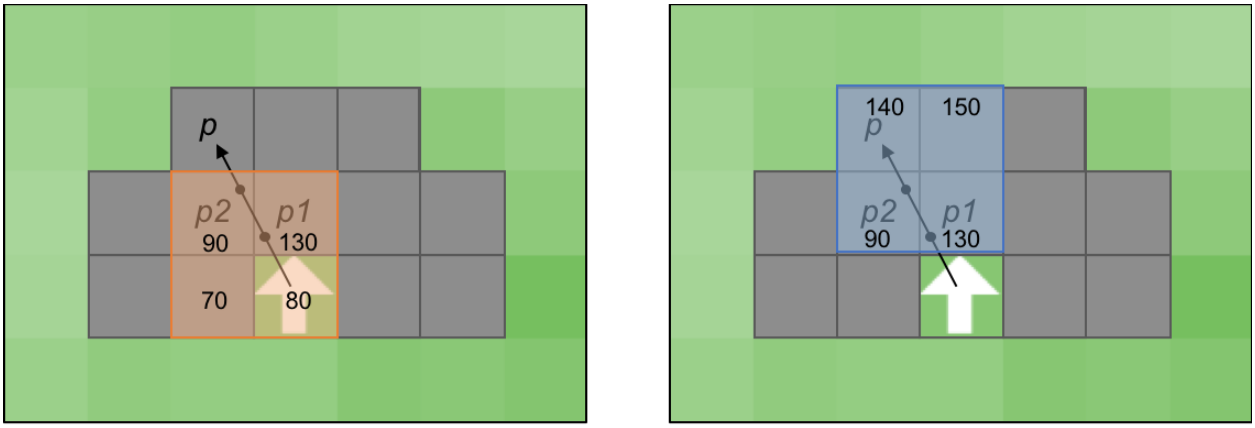


Figure 2. *How viewshed is calculated for non-adjacent straight patches (left) and knight patches (right). The potential patch on the left is obstructed by the middle patch.*

The potential patch evaluates if the hiker is adjacent.

- If they are, nothing happens. The patch remains unobstructed.
- If they are not adjacent, but are separated from the patch by only 1 patch (straight line between the two) (see Figure 2, left):
    - It evaluates if the elevation of the patch between the two is higher than a certain threshold reported by viewshed.
    - If the middle elevation is higher, the potential patch becomes 'obstructed' and is ultimately removed from the pool of potential patches.
- If the hiker is separated from the patch by 2 patches (see Figure 2, right):
    - The middle patches use the triangle-knight1 and triangle-knight2 reporters to identify the heading towards the patch directly opposite itself (perpendicular to the path direction (unlabeled patches in Figure 3).
    - The middle patches interpolate their elevation at the walked corner, using Inverse Distance Weighting of the elevation of itself and the 3 closest patches (see Figure 3).
    - The potential patch evaluates if the interpolated elevation of at least one of the two patches between them is higher than a certain threshold reported by viewshed-knight1, for the middle patch closest to the hiker, and viewshed-knight2 for the middle patch furthest from the hiker.
    - If at least one middle elevation is higher, the potential patch becomes 'obstructed'.



$$p1_{sw} = \frac{\left(\frac{130}{0.47}\right)+\left(\frac{90}{0.74}\right)+\left(\frac{80}{0.74}\right)+\left(\frac{70}{0.94}\right)}{\left(\frac{1}{0.47}\right)+\left(\frac{1}{0.74}\right)+\left(\frac{1}{0.74}\right)+\left(\frac{1}{0.94}\right)} = 98.54$$

$$p2_{ne} = \frac{\left(\frac{90}{0.47}\right)+\left(\frac{130}{0.74}\right)+\left(\frac{140}{0.74}\right)+\left(\frac{150}{0.94}\right)}{\left(\frac{1}{0.47}\right)+\left(\frac{1}{0.74}\right)+\left(\frac{1}{0.74}\right)+\left(\frac{1}{0.94}\right)} = 121.46$$

Figure 3. *How Inverse Distance Weighting is used to calculate the elevation at the point walked on p1 and p2 middle patches. The numbers represent elevation at the center of each patch.*

To-report triangle-knight1 [x]
This identifies the direction (in degrees) in which the unlabeled orange patch is (Figure 3) in relation to p1.

The reported angle values are based on the rounded angle between the hiker and its potential patch.

- If the hiker is at angle 27 or 243 from the potential patch, it reports the value 315.
- If the hiker is at angle 63 or 207 from the potential patch, it reports the value 135.
- If the hiker is at angle 117 or 333 from the potential patch, it reports the value 45.
- If the hiker is at angle 153 or 297 from the potential patch (like in Figure 3), it reports the value 225.

To-report triangle-knight2 [x]
This is similar to the triangle-knight1 reporter, but it is used only by p2. This identifies the direction (in degrees) in which the unlabeled blue patch is (Figure 3) in relation to p2.

The reported angle values are based on the rounded angle between the hiker and their potential patch.

- If the hiker is at angle 27 or 243 from the potential patch, it reports the value 135.
- If the hiker is at angle 63 or 207 from the potential patch, it reports the value 315.
- If the hiker is at angle 117 or 333 from the potential patch, it reports the value 225.
- If the hiker is at angle 153 or 297 from the potential patch (like in Figure 3), it reports the value 153.

To-report viewshed:
This divides the elevation change between two non-adjacent patches separated by only one patch in two to evaluate at what elevation the middle patch would obstruct the view.

This also uses the viewshed-threshold value (if applicable) to increase the threshold of visibility.

It first calculates the elevation change between the top of the hiker's head and the potential patch.

Then, it calculates the threshold, which is:

$$t = h + \left(\frac{ep_h}{2}\right) + (vt * r_m)$$

where *h* refers to the hiker's elevation, ep_h to the elevation change between the potential patch and *h*, *vt* to the viewshed-threshold chosen by the user, and *rm* to the resolution of the map in meters.

To-report viewshed-knight1:
This is similar to the viewshed reporter detailed above, but the elevation change is multiplied by 1/3 instead, as it refers to the third of the path that is closest to the hiker.

To-report viewshed-knight1:
This is similar to the viewshed reporter detailed above, but the elevation change is multiplied by 2/3 instead, as it refers to the third of the path that is closest to the potential patch.

To assign-values [*p boolean*]:
This is used by the patch under the hiker to calculate the cost of moving to any other patch – here labelled '*p*'. It also takes a boolean value which identifies if *p* is the winner-patch or simply a potential one. This value is passed on to the procedure calculate-speed, which allows determining if the model needs to record all steps of a path.

This uses 5 temporary variables to make sure that the computation for a set of patches does not carry over to another set. The variables are the following:

- *dp*: the distance between the hiker and patch p
- *ep*: the elevation change between the hiker and patch *p* (elevation of *p* – elevation of hiker)
- *p1*: the patch between the hiker and patch *p* that is closest to the hiker, if applicable.
- *p2*: the patch between the hiker and patch *p* that is closest to patch *p*, if applicable.
- *dir*: the heading towards patch *p* (in degrees)

Figure 4 summarizes the different ways in which speed and effective-slope values are calculated based on patch *p*'s position in relation to the hiker.

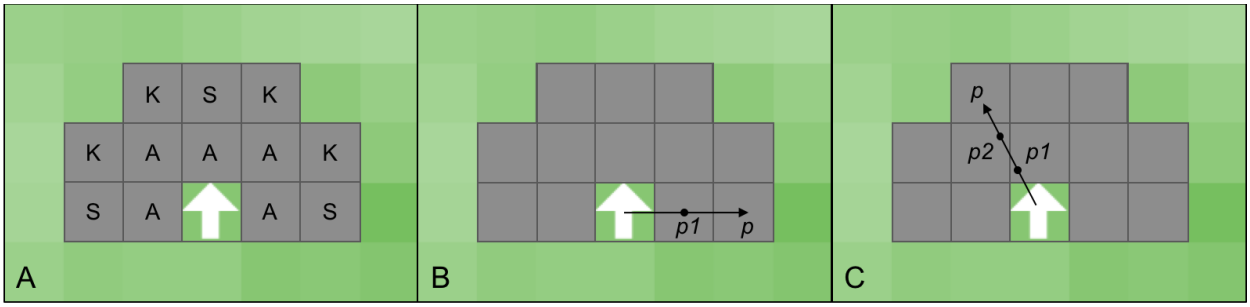


Figure 4. *A. the label of the different potential patches in relation to the hiker (A: adjacent, S: non-adjacent but in a straight line, K: knight). B. How values are calculated to move to a non-adjacent straight patch. C. How values are calculated to move to knight patches.*

The patch under the hiker evaluates if patch *p* is adjacent.

- If it is:
  - It uses the procedure calculate-speed with variables *p*, *dp*, and *ep* to calculate the speed of walking to *p*.
- If *p* is not adjacent, but is separated from the hiker by only 1 patch (straight line between the two) (see S in Figure 4.A):
  - It identifies *p1*, which is the patch separating itself from patch *p*.
  - It asks *p1* to calculate its distance and elevation change from patch *p* and to use calculate-speed with those values to obtain the cost of walking from *p1* to *p*.
  - It calculates its distance and elevation change from *p1*, and uses calculate-speed with those values to obtain the cost of walking to *p1*.
  - It uses the worst-slope reporter to identify the steepest slope between *itself-p1*, and *p1-p*, and takes that value as its effective-slope.
  - It sums up the *time* calculated for the two legs of the path and takes that sum as its *time* value.
- If *p* is a knight patch (separated from the hiker by 2 patches, at an oblique angle) (see K in Figure 4.A):
  - It identifies *p1*, which is the closest patch separating itself from patch *p*.
  - It identifies *p2*, which is the furthest patch separating itself from patch *p*.
  - It sets the distances between each leg at 0.74, which corresponds to the total distance divided by 3.
  - *p1* and *p2* calculate the interpolated elevation at the point they are walked on, using the reporters triangle-knight1 and triangle-knight2 to identify the patches to include.
  - *p1* uses calculate-speed with those values to obtain the cost of walking from *p1* to *p2*.
  - It asks the same for *p2* to move to patch *p*.
  - It does the same for itself to move to *p1*.
  - It uses the worst-slope reporter to identify the steepest slope between *itself-p1*, *p1-p2*, and *p2-p*, and takes that value as its effective-slope.
  - It sums up the *time* calculated for the three legs of the path and takes that sum as its *time* value.

The patch under the hiker then calculates its *abs-eslope* variable based on its *effective-slope*.

It also calculates the *speed* required to walk to *p* by dividing the distance to it by the time calculated.

To calculate-speed [*p dp ep boolean*]:
Any patch can call this procedure when calculating the cost of moving to an adjacent patch. It takes the variables *p*, *dp*, *ep*, and *boolean*, which represent the patch to move to, the distance to that patch, the elevation change of that patch, and a boolean value identifying if that patch is the winner-patch or a simple potential-patch.

At the beginning of the procedure, the following temporary global variables are defined:
- *dS*: the distance to *p* in meters.
- dH_u: Δ uphill elevation change. Set to 0.
- dH_{md}: Δ moderate downhill elevation change. Set to 0.
- dH_{sd}: Δ steep downhill elevation change. Set to 0.

Then, the asking patch calculates the effective slope (ε) of walking to *p*, using the formula:

$$\varepsilon = \sin^{-1}\frac{O}{H}$$

where *O* refers to the opposite side (elevation change, *ep*), and *H* to the hypothenuse (distance walked, *dS*). In NetLogo, this is implemented using:

$$\varepsilon = \sin^{-1}\frac{x}{\sqrt{1 + x^2}}$$

where *x* refers to the elevation change (*ep*) divided by the distance in meters (*dS*). This calculation is actually done using the To-report-arctan reporter.

If the boolean value is true, the hiker updates the list-slope list with the effective-slope of the patch doing the calculation. This allows recording the effective slope of each steps in the walking process, not just the patches walked to.

The effective-slope (ε) value is then used to populate the Δ variables mentioned above with the elevation change (*ep*). For example, if ε = -7 (moderate downhill), dH_{md} takes on the value of *ep*, and all other values remain at 0. The values of the temporary variables are then aggregated to determine walking time in seconds (*T*) using Langmuir's (1984) correction on Naismith's (1892, in Aitken 1977) rule for anisotropic travel:

$$T = 0.72 \times dS + 6 \times dH_u + 1.9998 \times dH_{md} - 1.9998 \times dH_{sd}$$

To-report worst-slope [*a b*]:
This reports the steepest effective slope of a set of two (slopes of patches *a* and *b*). Therefore, *a* and *b* need to be patches. The code takes their respective effective-slope values and calculates which is the steepest. As the steepest value are the lowest when dealing with negatives, and the highest, when dealing with positives, this uses a lot of *if-then* statements.

If both values are negative, the ID of the patch with the lowest effective-slope is reported.
If both values are positive, the ID of the patch with the highest effective-slope is reported.
If one is positive and the other is negative, the negative value is converted into its absolute, and the ID of the patch with the highest absolute effective-slope is reported.

To move:
Simple procedure to move the hiker to the *winner-patch*.

Before they move, however, a few temporary global variables are created:
- *dist-winner-patch*: records the distance between the hiker and where they will move.
- *time-ph*: records the time required to move to *winner-patch*.

If the dist-winner-patch is > 2, the hiker:
- Advances by 0.74
- Updates the plots, which records the effective slope of the patch where the hiker stands
- Advances by 0.74 and updates the plots again.
- Moves to the winner-patch and update the plots.

If the dist-winner-patch is > 1, the hiker:
- Advances by 1 and updates the plots.
- Moves to the winner-patch and update the plots.

If the dist-winner-patch is < 1, the hiker:
- Moves to the winner-patch and updates the plots.

When the hiker reaches the winner-patch, they:
- Update their *dist-traveled* values by adding the distance just traveled to their previous value.
- Update their *time-walked* value by adding *time-ph* to their *time-walked* previous value.
- Update the global *time-wd* variable with the new *time-walked* value.

To update-colors:
This is a patch procedure that is called at the start and end of a run to give the patches their DEM colors.

For land patches, it uses a green color scale with the maximum elevation and -400 as end points.
For water patches, it uses a blue color scale with the minimum elevation and 1000 as end points.
For snow-covered patches, the color is white.

This part of the model changes the *elevation* and *effective-slope* of water and snow-covered patches to 0.

To export:
This exports the values of the plot (x and y coordinates) to a CSV file that can then be found in the same folder as the model.

REFERENCES CITED

Aitken, R. 1977. *Wilderness areas in Scotland*. Aberdeen, Scotland.

Langmuir, E. 1984. *Mountaincraft and leadership*. Cordee, Leicester: The Scottish Sports Council/MLTB.