ODD: Agent-based simulation of central-place foraging in hunter-gatherers

August 13, 2021

This description of our model follows the Overview, Design Concepts and Details (ODD) protocol, as presented in Garg et al.(2021).

The model was implemented using Python v.3.4. The source code of our model implementation can be found on Github. Because we used Python and popular libraries (such as numpy, pandas, and scipy), the core of this implementation is independent of platform and should run on most computers with Python.

1 Purpose

The purpose of this model is to simulate central-place foraging and mobility patterns based on hunter-gatherer foraging, and examine how such patterns could interact with environmental conditions to give rise to social networks that are efficient at information transmission.

Specifically, the goals of the model are to study how resource heterogeneity can affect mobility decisions (frequency and magnitude of logistical/foraging and residential movements) in central-place foragers, and how the emergent mobility strategies affect the probability and dynamics of inter-forager interactions.

The model was designed for human behavioral ecologists, movement ecologists, computational biologists, social scientists, and anthropologists who are interested in human social networks, movement and cultural evolution.

2 Entities and state variables

2.1 Space.

Space is represented as a square, two-dimensional landscape that ranges from 0 to 1, and comprises 50,000 uniformly distributed patches. Each patch was initially assigned resource content, $k_i \geq 1$, drawn from a normalized power-law probability distribution, $P(k) \approx Ck^{-\beta}$ where the exponent β determines the distribution of resource content and the total resource abundance (see Methods section and Fig. 1) and $C = 1/\sum_{k=1}^{\infty} k^{-\beta}$ is the normalization constant. When

$\beta \approx 1$, k has a broad range, patches vary widely in their resource content, and the environment is abundant with many rich patches. Conversely, $\beta \gg 1$ corresponds to smaller values of k, and scarcer resources that are homogeneously distributed across patches. Patches deplete by a unit every time-step a forager spends in it, and the patches do not regenerate

2.2 Model agents.

The agents in the model represent central-place foragers. Every forager had complete knowledge of resources, a randomly allocated home-base, and a foraging area with a given radius, r. Foragers could forage and change their home-base based on the following rules. Foraging moves to a new patch (p_j) from a depleted patch (p_i) minimized the cost/gain ratio (d_{ij}/k_j), where d_{ij} is the distance between the patches and k_j is the resource content of p_j. When foragers were on a patch with no food left, they made foraging moves to a patch (p_j) within r such that the cost/gain ratio (d_{hj}/k_j) was minimized, where d_{hj} is distance between home-base (p_h) and p_j, and k_j is the resource content of p_j. Before every move, foragers compared the cost/gain ratio of patches outside the radius to patches within the radius. When the resource quality within r diminished compared to the rest of the environment, instead of making a foraging move to profitable patches outside their radius, foragers made a residential move. Residential moves allowed foragers to select a new home-base (p_h) in a randomly chosen direction that minimized (d_{hh}/k_h) but was far enough from the current base to avoid overlap. Each time-step that a forager coincided with another forager on a patch, they formed a social network tie or added a unit of weight to an existing tie.

2.3 Agent attributes.

1. Patch (dynamic): Each agent is located in a patch, given by a vector (x,y) that represents the current location of the agent.

2. Name (static): Each agent is permanently assigned a number identity between 1 and population size. This number determines the order in which agents are updated at each time step.

3. Home Patch (dynamic): Each agent has a home position, given by a vector (x,y) that represents the center of the foraging radius.

4. Memories (dynamic): Each agent is given full knowledge of the food located in the environment at the initial time step. Every time that they are on a patch where the resources are depleted, they forget that patch.

5. Distance Moved (dismoved; dynamic): The distance traveled by the agent when moving from patch to patch within the foraging radius.

6. Home Moved (homedis, dynamic): The distance traveled by the agent when moving from one home patch to a new home patch with a foraging

region that does not overlap any part the region from the previous home patch.

7. Number of Foraging moves (logmoves, dynamic): The total number of foraging moves (i.e. moves within a radius) that a forager makes.

8. Number of Residential moves (resmoves, dynamic): The total number of residential moves (i.e. moves that change its home-patch) that a forager makes.

2.4 Patch Attributes.

1. Location(static): The x,y coordinates are initialized in the beginning of the simulation and selected from a uniform distribution.

2. Resource content (k, dynamic): The resource content of a patch decreases over time as foragers feed at it.

3. Name (static): Each patch is permanently assigned a number identity between 1 and 50,000.

2.5 Global state variables

1. Steps (dynamic): Time is modelled in discrete steps.

2. Runs (static): Multiple runs assist with calculating averages.

3. Population (static): The total number of foragers in the environment.

4. Patches (static): The total number of patches in the environment.

5. Resource content (k; dynamic): The cumulative number of food units remaining in the environment.

6. Beta (static): The amount of environmental richness and heterogeneity.

7. Foraging Radius (fRadius; static): The distance each forager is allowed to move around the home patch.

8. Effective Radius (er; static): The minimum distance a forager must travel to choose a new home patch in order to eliminate the possibility of the new foraging area coinciding with the previous foraging area. We have set this value to 2.

3 Process Overview and Scheduling

At model initiation, 50,000 Patches are created in the grid, and the resources are distributed in the patches according to the Beta parameter. Patches are initialized with no ocupants on the patch. The Beta determines the maximum k as well as the k for each patch, and the total resource content is the sum of k across all patches in the grid. The foragers are also created at this time with full knowledge of the resource distribution in each patch in the environment. Foragers are assigned an arbitrary identity between 1 and the total number of foragers, as well as an arbitrary home patch. All foragers are assigned the same foraging radius.

At each time step, the processes are executed in the following order:

(1) Each forager checks whether or not there is food at their patch using the nextState submodel.If there is food, they do not move. If there is no food

  (I) The forager deletes the patch from memory using the removeMemory submodel.

  (II) The forager returns to the home patch using the nextState submodel.

  (III) The forager searches within their memory for the patch that will minimize the cost/benefit ratio using the check_radius_forfood submodel.

    (A) If the best patch is within the foraging radius

      (i) The forager moves to the best patch.

      (ii) The forager's patch is updated.

    (B) If the best patch is outside of the foraging radius

      (i) The forager relocates to a new home patch using the check_forhome_avoidoverlap submodel

      (ii) The forager's home patch is updated.

      (iii) The forager's patch is updated.

(2) Each patch checks for foragers on the patch using the nextState submodel

  (I) If there are no foragers, nothing changes.

  (II) If there are foragers on the patch, one piece of food on the patch is removed for each forager using the removeFood submodel.

4 Design Concepts

4.1 Basic Principles

This model simulates and compares point-to-point and central-place foraging (with different home-range radii) behavior across a range of environments. The model is based on previous work by Ramos-Fernández et al.(2006) which modeled the effect of environmental heterogeneity on the interaction networks that emerge from multiple agents foraging independently (representing spider monkeys).

4.2 Interaction

There is no interaction within the model that affects forager decisions directly. However, we consider foragers to interact with each other if they coincide on a resource patch.

4.3 Stochasticity

The stochastic parameters in this model include the creation of patches within the environment, the distribution of resources within the patches, and the assignment of foragers to home patches.

1. Patch Creation- the patches are created with a stochastic process. A uniform distribution creates their locations.

2. Resource Distribution- Power-law distribution with a specific exponent (β) assigns them resource content. The parameter controls the resource content of a patch and the variance in resource content between all patches.

3. Home Patch Assignment- The foragers are initialized with random coordinates that are considered to be their initial home-patches.

4.4 Initialization

Simulation parameters. The simulation is for 100 time-steps. We run every parameter combination 50 times. However, for sensitivity analyses, we ran the simulation for 1000 time-steps, 20 times.

Grid. To initialize the resource landscape, we execute the submodule 'createPatches' which defines the locations of the 50,000 resource patches and their resource content. The resource content is dependent on the parameter β in . The locations of the patches are derived from a uniform distribution (0-1).

Agents. Each agent is created by executing the submodule 'createForagers' which assigns every forager a unique ID from 1 to the population of foragers. When each forager is created, their location is chosen randomly, and they have a fully memory which contains the location and resource content of every patch.

4.5 Output Data

The data collected from the model are:

1. The position coordinates of every agent at every time-step.

2. A list of logistical/foraging and residential moves defined by their distance made by the agent over the course of the run.

3. The total number of logistical/foraging and residential moves made by the forager over the course of the run.

4. The total resource content left in the environment at every time-step.

4.6 Input Data

There is no input data required.

5 Submodels

5.1 class Forager

nextState

(1) A forager checks its current patch for its resource content. If the patch has resources left, it consumes a unit per time-step and does not change its location.

(2) If the patch has no resources left,

  (I) the forager removes the patch from its memory using the submodel 'removeMemory'.

  (II) it checks for food within its radius through the submodel 'check_radius_forfood'

    (A) if the submodel returns a patch within the radius

      (i) the current patch calls the submodel 'removeGuest'.

      (ii) it updates its position to that of the new patch found.

      (iii) it updates its patch to the current patch.

      (iv) the current patch calls the submodel 'putGuest'.

      (v) it adds a unit to its total logistical/foraging moves.

      (vi) it appends the distance moved to the list of foraging moves.

    (B) if the submodel returns None

      (i) it checks for a new home using the submodel 'check_forhome_avoidoverlap'.

      (ii) the current patch calls the submodel 'removeGuest'.

      (iii) the current home-patch calls the submodel 'removeGuest'.

      (iv) it updates its location to the new home found and designates it as both the home-patch and patch.

      (v) the new home-patch calls the submodel 'putGuest'.

      (vi) it removes the new patch from memory

      (vii) it adds a unit to its total residential moves.

      (viii) it appends the residential distance moved to the list of residential moves.

removeMemory This submodel subtracts a given patch from the forager's memory.

check_radius_forfood This submodel searches through the forager's memory to find a new suitable patch that maximizes its foraging efficiency (benefit/cost).

(1) It calculates cost/benefit ratio of every patch in the forager's memory where the cost is defined as the distance between the patch and the home-patch, and the benefit is defined by the resource content of the patch.

(2) Next, the model selects the patch which has the minimum cost/benefit ratio.

(3) Then, it checks whether the distance of the selected patch is less than the radius. If it is less than the radius, the submodel returns the selected patch. Else, it returns None.

check_forhome_avoidoverlap

This submodel searches through the forager's memory to find a new suitable home-patch that maximizes its foraging efficiency (benefit/cost) and avoids overlap with its previous foraging area.

(1) It calculates the distance from its current home-patch to every patch in the forager's memory.

(2) If the distance is greater than 2*radius :

  (I) It calculates cost/benefit ratio of the patch where the cost is defined as the distance between the patch and the home-patch, and the benefit is defined by the resource content of the patch.

(3) Next, the model selects the patch which has the minimum cost/benefit ratio.

(4) The submodel returns the selected patch.

5.2 class Patch

nextState This submodel checks whether there are any foragers on the patch. If there are foragers, it calls upon the submodel 'removeFood' to update its resource content.

putGuest This submodel adds a unit every time a forager is present on the patch.

removeGuest This submodel removes a unit every time a forager is present on the patch.

removeFood This submodel subtracts a unit resource for every forager that is present on the patch at every time-step.