**Supplementary Information 1**

**ODD: Agent-based simulation of a honey bee colony foraging on different landscapes**

This description of our model follows the Overview, Design Concepts and Details (ODD) protocol, as presented in Grimm et al. (2020). Parts of this model description are adapted from a previous ODD (Mosqueiro 2018), available [here](https://github.com/thmosqueiro/ABBAS/blob/master/docs/ODD/README.md).

The model was implemented using Python v.2.7. The source code of our model implementation can be found on [Github](https://github.com/nlemanski/ABS_Honeybee_Foraging). Because we used Python and popular libraries (such as numpy, pylab and math), 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 examine how the spatial distribution of resources in the environment interacts with individual variation in behavior to shape collective foraging decisions. The model is intended to both reproduce observed patterns of collective behavior in honey bees and to generate new hypotheses for empirical studies of the interaction between individual variation in behavior and landscape structure in shaping collective decision making in honey bees.

Specifically, the goals of the model are to 1) reproduce the emergent pattern of a colony collectively choosing among food patches of different qualities, based on quality-based differences in recruitment, 2) develop predictions about how the spatial distribution of food resources influences a colony’s collective choice of food patch, 3) develop predictions about how the behavioral composition of the colony influences the colony’s collective choice of food patch, and 4) develop predictions about trade-offs between a colony’s ability to locate food patches and its ability to selectively exploit the highest quality patches.

This model was designed for ecologists, behavioral biologists, and computational biologists interested in the foraging behavior of honey bees and other social insects. We chose design principles to allow anyone with knowledge of Python to reproduce our results or for anyone with knowledge of agent-based modelling to re-implement our model in a language of their choice.

1. **Entities, state variables, and scales**

**Space**

Space is represented as a square, two-dimensional landscape, with the hive positioned at the origin. Agents may move anywhere within this landscape, however when an agent reaches the edge of the grid, it is transported immediately back to the hive.

The positions of food resources are stored as cells in a 500 by 500 square grid. Within this grid, food resources are aggregated into three square patches. Each patch has a different quality, defined as the number of food units a forager brings back to the hive when it collects a foraging load within that patch. These food patches can be arranged in two alternative distributions: clumped or dispersed. In the dispersed distribution, the three resource patches are evenly spaced in a circle around the hive. In the clumped distribution, all three resource patches are adjacent but not overlapping.

**Model agents**

The agents in the model represent honey bee foragers. The agents are divided into two basic types with different behavioral rules: scouts and recruits. Scouts spontaneously leave the hive and explore the environment. Once a scout discovers and exploits a food patch, the scout may report its location and quality to recruits. Recruits remain in the hive until another forager shares the location of a food patch and then may leave the hive to exploit that food patch. After exploiting a food patch, recruits may also report its location to additional recruits.

**Agent attributes**

Each agent in the model has the following attributes:

1. Forager type (static):

Each agent is permanently classified as either a scout or a recruit.

1. Forager number (static):

Each agent is assigned an arbitrary number, between 1 and 100. This number determines the order in which agents are updated at each time step.

1. Position (dynamic):

Each agent has a position, given by a vector (x,y). The position is measured with respect to the hive, located at position (0,0).

1. Drifting vector (dynamic):

Each agent has a drifting vector, which defines the agent’s preferred direction of movement after leaving the hive. An agent’s actual movement is determined by its drifting vector plus a random error term.

1. Patch memory (dynamic):

Each agent stores the location of the food patch they have most recently visited. The location of this stored food patch is given by a vector (x,y).

1. Foraging load (dynamic):

Each agent stores the food quality of the patch they have most recently visited, as an integer, from 1-3.

1. Behavioral mode (dynamic):

Agents classified as scouts move between four different behavioral modes, 1) searching for a new food patch, 2) returning to a previously visited patch, 3) returning to the hive, and 4) recruiting other foragers to a food patch.

Agents classified as recruits move between five different behavioral modes, 1) waiting to be recruited, 2) searching for an advertised patch, 3) returning to a previously visited patch, 4) returning to the hive, and 5) recruiting other foragers to a food patch.

1. Velocity (dynamic):

Each agent has a velocity, which defines how far the agent moves per time step. This velocity is set to 1.5 when a forager is searching for a new food patch and set to 1 when a forager is returning to a previously visited patch.

1. Movement error (dynamic):

Each agent has a movement error, which determines the magnitude of the random deviation from the agent’s drifting vector. The movement error is set to 5 when a forager is searching for a new food patch and set to 2 when a forager is returning to a previously visited patch.

1. Return count (dynamic):

Each agent has a return count (an integer), which tracks the number of times the forager has visited its most recently visited food patch.

1. Elapsed dancing time (dynamic):

When a forager is in the behavioral mode, “recruiting other foragers,” its elapsed dancing time tracks the number of time steps it has spent recruiting so far.

**Global state variables**

The environment has the following global state variables:

1. Time (dynamic):

Time is modeled in discrete steps.

1. Food collected (dynamic):

The cumulative number of food units collected by all foragers in the colony.

1. Energy expenditure (dynamic):

The cumulative energy expended by all foragers in the colony.

1. Patch visits (dynamic):

The cumulative number of times each patch has been visited by a forager.

1. Patches found (dynamic):

The cumulative number of scouts that have found food.

1. Recruited bees (dynamic):

The number of recruits that are currently exploiting a food patch and unavailable to be recruited.

1. Proportion recruited (dynamic):

Proportion of all recruits that are currently exploiting a food patch and unavailable to be recruited.

1. Broadcasted positions (dynamic):

The set of all locations that are currently being recruited to, expressed as an array of vectors.

**Scale**

Each side of the two-dimensional model landscape is 36 m long, for a total area of 1.3 km2. Each unit of space is approximately 72 cm.

Each unit of time in the simulation represents approximately 1.2 seconds. The simulation was run for 21000 time steps, representing approximately 7 hours of simulated time.

On each foraging trip, a forager collects 1 foraging load, equivalent to 0.1 mL of nectar. Depending on the patch quality, 1 foraging load contains 1, 2, or 3 units of food; 1 unit of food is equivalent to approximately 34 mg sucrose.

1. **Process overview and scheduling**

At model initiation, the location, sizes, and qualities of the food patches are defined using the “resource landscape” submodel and 100 foragers are created using the “create forager” submodel. To determine the order in which agents will act, each forager is assigned an arbitrary *forager number* from 1 to 100.

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

1. The scout with the lowest *forager number* executes its “update scout” submodel:
   1. If the scout is in *behavioral* *mode* “searching for a new patch” or “returning to a previously visited patch,” it executes the “update position” submodel, in which its *position* is updated
   2. If the scout is in *behavioral* *mode* “returning to the hive”:
      1. If its *position* is equal to (0,0), it executes submodel “scout returned”
      2. If its *position* is not equal to (0,0), it moves 1 spatial unit closer to the hive
   3. If the scout is in *behavioral* *mode* “recruiting other foragers”:
      1. The scout adds 1 to its *elapsed dancing time*
      2. If the scout’s *elapsed dancing time* ≥ the total dancing time parameter, the *elapsed dancing time* is set to 0, the scout executes the “leave hive” submodel, and the scout’s *patch memory* vector is removed from the list of *broadcasted positions*
   4. If the scout’s *position* is outside the maximum boundaries of the landscape, the scout’s *position* is reset to (0,0) and its *behavioral mode* is set to “searching for a new patch.”
2. If the scout has found food, it:
   1. Executes its “found spot” submodel in which it:
      1. Updates its *patch memory* to its current *position*
      2. Sets its *behavioral mode* to “returning to the hive”
   2. Updates its *foraging load*
   3. Updates the state variable *patches found*
   4. Adds 1 to the state variable *patch visits*
   5. Removes 1 foraging load from its current position in the landscape
3. The scout updates the state variable *energy expenditure.*
4. Steps 1-3 are repeated for all remaining scouts, in ascending order of *forager number.*
5. The recruit with the lowest *forager number* executes its “update recruit” submodel:
   1. If the *behavioral mode* is “waiting to be recruited,” the recruit executes the “recruitment” submodel.
   2. If the *behavioral mode* is “searching for an advertised patch” or “returning to a previously visited patch,” it executes the “update position” submodel in which its *position* is updated.
   3. If the *behavioral* *mode* is “recruiting other foragers”:
      1. The recruit adds 1 to its *elapsed dancing time*
      2. If the recruit’s *elapsed dancing time* ≥ the total dancing time parameter, the recruit’s *elapsed dancing time* is set to 0, the recruit’s *patch memory* vector is removed from the list of *broadcasted positions*, and:
         1. If the recruit’s *return count* ≥ the persistence parameter, the recruit’s *behavioral mode* is set to “waiting to be recruited”
         2. If the recruit’s *return count* < the persistence parameter, the recruit’s *behavioral mode* is set to “returning to a previously visited patch”
   4. If the *behavioral* *mode* is “returning to the hive”:
      1. If its *position* is equal to (0,0), it executes submodel “recruit returned”
      2. If its *position* is not equal to (0,0), it moves 1 spatial unit closer to the hive
   5. If the recruit’s *position* is outside the maximum boundaries of the landscape, the recruit’s *position* is reset to (0,0).
6. If the recruit has found food, it:
   1. Changes its *behavioral mode* to “returning to the hive”
   2. Updates its *foraging load*
   3. Removes 1 foraging load from its current position in the landscape
   4. Adds 1 to the state variable *patch visits*
7. The recruit updates the state variable *energy expenditure.*
8. Steps 5-7 are repeated for all remaining recruits, in ascending order of *forager number.*
9. The state variable *recruited bees* is set equal to the number of recruits that are in *behavioral mode* “searching for an advertised food patch,” “returning to a previously visited patch,” or “returning to the hive.”
10. The global state variable *proportion recruited* is updated. *Proportion recruited* is equal to *recruited bees* divided by the total number of recruits.
11. **Design concepts**

**Basic principles**

This model explores the interaction between the behavioral composition of the colony and the spatial distribution of food resources in determining the colony’s ability to locate food patches and choose among patches with different qualities. The two types of foragers (scouts and recruits) perform different tasks. Scouts explore the environment randomly to locate new food patches. Once a patch is discovered, scouts may share the direction, distance, and quality of the food patch with recruits and then return to exploit the patch a fixed number of times. When a scout shares the location of a patch with a recruit, the recruit sets its drifting vector to that location and flies until it finds the advertised resource. After visiting a food patch, recruits may share the direction, distance, and quality of the patch with other recruits, then return to exploit the patch a fixed number of times. Our model captures the process of how honey bee foragers communicate about food resources using the waggle dance language. This model also incorporates flight behavior similar to that observed in field experiments.

**Adaptation**

Recruits and scouts that are returning to a particular resource spot have lower dispersion than that of scouts exploring the environment. This difference is because bees that are exploiting a resource patch are familiar with its location, and thus faster and more precise than those that are exploring the environment.

This change in flight precision increases the chances of a forager to find new resource spots when exploring the environment, while also increasing the chances of foragers returning to the same location with resources available. However, a remarkable precision in the flight patterns of recruited honey bees is empirically observed in many contexts, with errors of 5% over distances of 4km.

**Prediction**

No predictive models were employed in the decision-making processes involved in our model. For instance, recruits decide to leave the hive randomly; foragers decide to quit exploiting resources according to their persistence, which is fixed.

**Sensing**

When a forager reports to other recruits the location of a resource spot, recruits that decide to leave the hive and exploit that spot perceive the location of that spot. Once a forager is within 1 unit of distance (~70cm) from a resource spot, it is able to sense it and, therefore, pick up 1 foraging load of food.

**Interaction**

Foragers only interact with each other during the recruitment process. When one forager recruits another forager to a food patch, the recruit changes its drifting vector to the reported location of the food patch.

**Stochasticity**

There are four stochastic elements in this model: creation of resources patches in the environment, assignment of scout drifting vectors, forager flight dynamics, and which advertised resources patches recruits decide to exploit.

1. The environment is created with a stochastic process. In all model scenarios, we use an environment with three patches of resources, equally distant from the hive. Each cell within a resource patch contains a stochastic number of resource units, from 0-50.
2. The angle of the drifting vector of scouts was assigned from a uniform distribution between 0 and 2pi. Because the drifting vector of a forager defines its preferred direction of flight, scouts were equally likely to leave the hive in any direction. The magnitude of the vector was always the same to produce the same average velocity.
3. The flight dynamics of each forager follow a diffusion process. This diffusion process is usually referred to as Wiener process, which is a particular case of a Random Walk. On top of this diffusion process, bees had a preferred direction of movement determined by their drifting vectors.
4. Recruits stochastically decide which recruiting forager to follow, with the probability of following any given forager proportional to the quality of the patch they advertise.

**Collectives**

All foragers in the model belong to a single colony. Foraging performance is calculated as the total amount of resources collected by all foragers in the colony.

**Observation**

The data collected from the model are:

1. The total amount of food collected by all foragers at each time step
2. The total amount of food collected from each of the three patches at each time step
3. The total number of times each of the three patches is visited by a forager
4. The total energy expended by all foragers at each time step
5. The proportion of recruits that are actively recruited at each time step
6. **Initialization**

**Simulation parameters**

The simulation is run for 21000 time steps. We run each model scenario 150 times.

**Resource landscape**

To initialize the resource landscape in which the colony forages, we execute the submodel “resource landscape”, which defines the locations of three food patches on a two dimensional 500 x 500 square grid with the hive at the center. Each food patch is 40 x 40 units long (5.76 m), with the center located 200 units (14.4 m) from the hive. Within each food patch, the exact location of food is determined by a Poisson point process. Each point has a 0.6 probability to contain resources and each occupied point contains between 0 and 40 foraging loads, drawn from a uniform distribution.

We simulate two different model scenarios: one in which the three resource patches are clumped and one in which they are equally dispersed. In the dispersed scenario, the three patches are 120° from each other, relative to the hive. In the clumped scenario, the three patches are 24° from each other, relative to the hive.

**Colony**

At the start of the simulation, we create 100 foragers using the submodel “create forager”. Each forager is assigned a unique *forager number* from 1 to 100. When each forager is created, its *position* is set to the hive (0,0), its *patch memory* is an empty vector, and its *foraging load* is set to 0. For all foragers, *elapsed dancing time* and *return count* are set to 0.

When a forager is created, its *forager type* is set to either scout or recruit and does not change throughout the simulation. To explore how the colony’s behavioral composition affects collective foraging decisions, we ran the simulation with 10, 20, 30, 40, 50, 60, 70, 80, or 90 foragers set as scouts.

For scouts, the variable *behavioral mode* is set to “exploring,” while for recruits, the *behavioral mode* is set to “waiting to be recruited.” For scouts, initial *velocity* is set to 1.5 and *movement error* is set to 5. For recruits, initial *velocity* is set to 1 and *movement error* is set to 2. Each scout is assigned a random *drifting vector*, (x,y), by drawing an angle from a uniform distribution between 0 and 2pi. For recruits, the initial *drifting vector* is an empty vector.

**Global state variables**

The variables *food collected*, *energy expenditure*, *patch visits*, *patches found*, *recruited bees*, and *proportion recruited* are set to 0. The variable *broadcasted positions* is an empty array.

1. **Input data**

The model does not use any input data to represent time-varying processes. Model dynamics are driven entirely by the initial conditions and stochastic events.

1. **Submodels**

**Resource landscape**

This submodel generates the resource landscape in which the colony forages (see Table 1 for full list of parameter values).

1. The landscape consists of three 2-dimensional grids, each defining the location of one of the three food patches.
   1. Patch 1 contains high quality food
   2. Patch 2 contains medium quality food
   3. Patch 3 contains low quality food
2. Each patch has an x and y coordinate, which define the patch’s center.
3. The model can have two alternate types of resource landscape, where *d* is the distance from the hive to each patch (200 spatial units):
   1. For the evenly dispersed landscape:
      1. The coordinates (x,y) of patch 1 are
      2. The coordinates of patch 2 are (,
      3. The coordinates of patch 3 are (,
   2. For the clumped landscape:
      1. The coordinates (x, y) of patch 1 are (
      2. The coordinates of patch 2 are
      3. The coordinates of patch 3 are (,
4. The length and width of each patch (parameter *c*) is 40 spatial units.
5. Within each patch, the exact location of food is determined by a Poisson point process:
   1. Each cell within the patch has a 0.6 probability to contain food (parameter *p*).
   2. Cells with food are assigned between 1 and 50 foraging loads, drawn from a uniform distribution.

**Create forager**

1. Each forager is assigned a unique *forager number*, between 1 and 100, to determine the order in which the agents act.
2. Each forager has an initial x and y *position* of (0,0), the location of the hive.
3. Each forager’s initial *foraging load*, *elapsed dancing time*, and *return count* are set to 0 and initial *patch memory* is set to an empty vector.
4. Each forager’s persistence (the number of times they return to a patch) is set to 20.
5. Each forager is assigned a *forager type* of scout or recruit:
   1. For scouts:
      1. The initial *behavioral mode* is set to “searching for a new food patch.”
      2. The initial *velocity* is 1.5 and the initial *movement error* is 5.
   2. For recruits:
      1. The initial *behavioral mode* is set to “waiting to be recruited.”
      2. The initial *velocity* is 1 and the initial *movement error* is 2.

**Update scout**

1. If a scout is in *behavioral mode* “searching for a new food patch” or “returning to a previously visited patch”:
   1. The scout executes submodel “update position.”
2. If the scout is in *behavioral mode* “returning to the hive”:
   1. If the scout’s *position* is within 3 spatial units of the hive, the scout executes submodel “scout returned.”
   2. If the scout’s *position* is not within 3 spatial units of the hive, the *position* is moved one spatial unit closer to the hive.
3. If the scout is in *behavioral mode* “recruiting other foragers”
   1. The scout adds 1 to the *elapsed dancing time.*
   2. If the *elapsed dancing time* is ≥ *tr* (the total dancing time):
      1. The *elapsed dancing time* is set to 0.
      2. If the *return count* ≥ persistence, the behavioral mode is set to “searching for a new food patch.”
      3. If the *return count* < persistence, the behavioral mode is set to “returning to a previously visited patch.”
      4. The scout executes the “leave hive” submodel.
      5. The scout’s *patch memory* is removed from the set of b*roadcasted positions.*
4. If the scout’s x or y *position* is > 250 spatial units from the hive:
   1. The *position* is set to (0,0).
   2. If the *behavioral mode* is not “searching for a new patch,” the *behavioral mode* is set to “returning to a previously visited patch.”

**Update recruit**

1. If the recruit’s *behavioral mode* is “waiting to be recruited”:
   1. The recruit executes the submodel “recruitment.”
2. If the recruit’s *behavioral mode* is “searching for an advertised patch” or “returning to a previously visited patch”:
   1. The recruit executes submodel “update position.”
3. If the recruit is in *behavioral mode* “returning to the hive”:
   1. If the recruit’s *position* is within 3 spatial units of the hive, the recruit executes submodel “recruit returned.”
   2. If the recruit ‘s *position* is not within 3 spatial units of the hive, the *position* is moved one spatial unit closer to the hive.
4. If the recruit is in *behavioral mode* “recruiting other foragers”:
   1. The recruit adds 1 to the *elapsed dancing time.*
   2. If the *elapsed dancing time* is ≥ *tr* (the total dancing time):
      1. The *elapsed dancing time* is set to 0.
      2. If the *return count* ≥ persistence, the behavioral mode is set to “waiting to be recruited.”
      3. If the *return count* < persistence, the behavioral mode is set to “returning to a previously visited patch.”
      4. The recruit executes the “leave hive” submodel.
      5. The recruit ‘s *patch memory* is removed from the set of b*roadcasted positions.*
5. If the recruit ‘s x or y *position* is > 250 spatial units from the hive:
   1. The *position* is set to (0,0).

**Update position**

1. If the *behavioral mode* is “searching for a new patch”:
   1. A random number, *n,* is drawn from a uniform distribution between -0.5 and 0.5.
   2. The forager is assigned a movement angle, θ, equal to their *movement error* \* *n* + the angle of their *drifting vector*.
   3. The forager’s x position increases by *velocity* \* cos(θ).
   4. The forager’s y position increase by *velocity* \* sin(θ).
   5. The global variable energy expenditure is increased by *dE* where:

and

1. If the *behavioral mode* is “searching for an advertised patch” or “returning to a previously visited patch” and the current *position* – the *drifting vector* > 0:
   1. A random number, *n,* is drawn from a uniform distribution between -0.5 and 0.5.
   2. The forager is assigned a movement angle, θ, equal to their *movement error* \* *n* + the angle of their *drifting vector*.
   3. The forager’s x position increases by *velocity* \* cos(θ).
   4. The forager’s y position increase by *velocity* \* sin(θ).
   5. The global variable energy expenditure is increased by *dE* where:

and

1. If the *behavioral mode* is “searching for an advertised patch” or “returning to a previously visited patch” and the current *position* – the *drifting vector* ≤ 0:
   1. The change in x *position*, dx = 3 \* a random number between -0.5 and 0.5
   2. The change in y position, dy = 3 \* a random number between -0.5 and 0.5
   3. The global variable energy expenditure is increased by:

**Scout returned**

1. If the scout’s *return count* ≥ persistence:
   1. The *behavioral mode* is set to “searching for a new patch.”
   2. The *return count* is set to 0.
2. If the scout’s *return count* = 0, the scout begins recruiting with a probability of 0.33 x *foraging load*.
   1. If the scout begins recruiting:
      1. The scout’s *behavioral mode* is set to “recruiting other foragers.”
      2. The scout’s *patch memory* is added to the set of *broadcasted positions*.
   2. If the scout does not begin recruiting:
      1. The *behavioral mode* is set to “returning to a previously visited patch.”
      2. The scout executes the submodel “leave hive.”
   3. The scout’s *return count* increases by 1.
3. If the scout’s *return count* < persistence but > 0:
   1. The *behavioral mode* is set to “returning to a previously visited patch.”
   2. The scout’s *return count* increases by 1.
   3. The scout executes the submodel “leave hive.”
4. The scout’s *position* is set to (0,0).
5. The global variable *food collected* is increased by the scout’s *foraging load*.
6. The scout’s *foraging load* is set to 0.

**Recruit returned**

1. If the recruit’s *return count* ≥ persistence:
   1. The *behavioral mode* is set to “waiting to be recruited.”
   2. The *return count* is set to 0.
2. If the recruit’s *return count* = 0, the recruit begins recruiting with a probability of 0.1 x *foraging load*.
   1. If the recruit begins recruiting:
      1. The recruit’s *behavioral mode* is set to “recruiting other foragers.”
      2. The recruit’s *patch memory* is added to the set of *broadcasted positions*.
   2. If the recruit does not begin recruiting:
      1. The *behavioral mode* is set to returning to a previously visited patch.
      2. The recruit executes the submodel “leave hive.”
   3. The recruit’s *return count* increases by 1.
3. If the recruit’s *return count* < persistence but > 0:
   1. The *behavioral mode* is set to “returning to a previously visited patch.”
   2. The recruit’s *return count* increases by 1.
   3. The recruit executes the submodel “leave hive.”
4. The recruit’s *position* is set to (0,0).
5. The global variable *food collected* is increased by the recruit’s *foraging load*.
6. The recruit’s *foraging load* is set to 0.

**Leave hive**

1. If the forager’s *behavioral mode* is set to “searching for a new patch”:
   1. The *patch memory* is set to an empty vector.
   2. The *return count* is set to 0.
   3. The *velocity* is set to 1.5.
   4. The *movement error* is set to 5.
   5. The forager is assigned a new *drifting vector*, (x,y), by drawing an angle from a uniform distribution between 0 and 2pi.
2. If the forager’s *behavioral mode* is set to “returning to a previously visited patch”:
   1. The *velocity* is set to 1.
   2. The *movement error* is set to 2.
   3. The *drifting vector* is set equal to the *patch memory* vector divided by the norm of the *patch memory* vector.

**Found spot**

1. When a forager’s *position* is within 1 spatial unit of a cell that contains food:
   1. The forager’s *behavioral mode* changes to “returning to the hive.”
   2. The forager’s *patch memory* is set equal to the forager’s current *position*.

**Recruitment**

1. If the set of *broadcasted positions* contains at least one location vector, the recruit is activated with probability equal to 0.1 / the number of recruits in *behavioral mode* “waiting to be recruited.”
2. If the recruit is activated:
   1. Its *behavioral mode* is set to “searching for an advertised food patch.”
   2. Its *patch memory* is set equal to a vector from the set of *broadcasted positions*. The probability of choosing each vector is proportional to the quality, *q*, associated with that location.
   3. Its *drifting vector* is set to equal to the recruited position divided by the norm of the recruited position.