Bottmedical Microbiome Project Report and Manual

Ilya Schneider: schnil1010@gmail.com

September 2023

1 Introduction

This document represents a summary of the microbiome simulation project, which contains a detailed description of the concepts implemented in the code, an exploration of parameter influence on the simulation outcome and examples of possible model implementations. The general structure and core concepts were inspired by one particular publication, which is further referred to as the reference paper [1]. All scripts and notebooks are stored in a private GitHub repository. Access to them can be granted upon demand.

This report is divided into two main sections. The first one centers on the proof of concept and aims to replicate observations in the reference paper. Additionally, it includes a data analysis part, which aims to provide an intuition for model parameters. The second part is an extension of the first one towards realistic applications of the model.

The entire model is based on the publically available MESA framework (version 1.2.1), which provides a great layout for projects related to agent-based modeling in Python.

As the scaffold for this project code structure of Adrian Altermatt and Sirinthip Tothom was taken over and extended under the supervision of Christian Kronseder and Tino Töpper.

2 Predator Prey Model

This segment begins with a detailed description of the code and more importantly, assumptions related to it. Further along, parameter selection for the batch simulation is analyzed and interpreted.

This proof of concept part of the project aims to replicate the behavior of a predator-prey system [1]. In this scenario, the predator is a bacteria species that can sense prey species in its proximity and as a result, become stressed. The stress reaction manifests as a secretion of antibiotic substance, that harms the prey and can lead to its death. Unintuitively, this setting does not always result in predator domination and can in fact lead to the opposite result. Key players in the fate of the competition are believed to be the ratio of prey to predator and the initial location of bacteria. The reference paper simulates this competition on a circular boundary, where edge regions are thought to be more beneficial for the prey's survival. In this project's setting, however, the boundary is defined by a square. This setting maintains the survival benefit on the edge and adds corners as an even safer location.

2.1 Code Sections

The following part goes in detail through each code chunk to provide an overview and better understanding of the implemented structures. To run the simulation in the browser a Server Setup file is provided and can be executed using the Run Server file. To do so from a code editor a general pipeline provided by the MESA tutorial can be used. Importantly, all the default values and parameters are to be seen as suggestions for the proper functioning of the model and not as the ground truth. The user is free and in fact, is encouraged to play around with the parameters and hidden variables to get better code intuition. Last but not least, while going through this report part it is recommended to keep an eye on the Python script to facilitate the comprehension.

2.1.1 Soil Agent

This agent contains information about the nutrients that are available for the bacteria and the antibiotics that are secreted by the predator. Each simulation coordinate contains a specific nutrient amount and can also receive antibiotic substances. During nutrient consumption, the nutrient levels decrease in the coordinates where bacteria are located. During the antibiotic action, prey also uptake the antibiotic from their coordinates. The current setup implies an infinite nutrient supply for the bacteria. Therefore, there is an automatic refill set up to avoid nutrient levels decreasing to zero.

2.1.2 Additional Functions and Constants

`get_average_pos`: the function was taken over from the previous developer and was kept in case it can be used in the future.

`avoid_identical_clones`: the function creates a normal distribution around the specified value but provides only positive values from it as an output. The standard deviation of this distribution is set as 10% of the specified mean. This function is a great tool to introduce populational variability into the simulation and thus make it more realistic.

`get_num_bacteria_per_type`: the function is used to calculate the number of each species at every timestep.

One of the main parameters in the simulation is the cell area. It is obtained by taking an approximated radius (`s_mutans_radius`) of the bacterial cell and, assuming it is a circular object, calculating the area of a circle with this radius (`average_bacteria_area`) [2].

2.1.3 General Concepts for Bacterial Behaviour

Both predator and prey actions and mechanisms are based on the same key concepts [1]. First, the common basics will be discussed in this section and then type-specific technicalities will be introduced in the following segments.

All bacteria apart from their normal `self.area`, possess also a `self.split_area` (if this area is reached bacteria can divide into two daughter cells) and the `self.min_area` (if the cell shrinks to the minimal area it dies).

The developed model tries to realistically replicate the metabolic transfer of nutrients into energy and then energy into the area of the bacteria (a proxy for mass growth). Before getting into the functional details, it is important to stress that the core assumption of the simulation is that **1 energy unit is equal to 1 nutrient unit and to 1 surface unit**.

At the initiation, each bacterium is allocated randomly on the simulation grid. Bacteria have an upper threshold for the nutrient consumption. There are two sources for its setting: positional and biological; the smaller of the two will always be chosen. At each location, there is a specified amount of nutrients. However, bacteria can only access a fraction of it, defined by `self.avaliability`. Bacteria's nutrient uptake is also delimited by a product of its area and the `self.nutrient_uptake_ratio`, which is the second possible upper limit. As mentioned before, the smaller of the two becomes the actual nutrient consumption. From these consumed nutrients energy is produced using `self.enegry_yield`. Additionally, bacteria also need energy to keep themselves alive. Maintenance energy is defined as a product of area and `self.maintenance`. Finally, `self.energy_netto` is the difference between the produced and maintenance energies. If it is non-negative the `self.area` is increased by `self.energy_netto * 0.5`, this is the process that leads bacteria towards `self.split_area`. If it is negative the `self.area` is decreased by 10% and bacteria shrinks towards `self.min_area`. All the mentioned scaling factors and constants intend to represent the loss of energy throughout the conversion of food into bacterial mass and thus make the simulation more realistic.

For each simulation cell (coordinate) there is a bacteria limit that it can contain. The recommended value is 2, however, it can be changed to a higher value. This constraint breaks the symmetry in the colonies formed by bacteria. When a bacterium is on the verge of division it can divide into two daughter cells either in the coordinate where it is already located (if the `self.max_num_bacteria_in_cell` is not reached yet) or in one of the neighboring coordinates (if they are not full either). The second option is referred to as microcolony growth. Microcolony is defined as a group of bacteria cells (at least two) of the same species. One of the free coordinates on the verge of the microcolony is found, the mother cell is relocated there and its area is divided by two (first daughter cell) and the second daughter cell is created and placed into the location freed up by the mother cell (its area is also set to the half of the area of the mother cell). It may be the case that the entire simulation grid is full and no free space is found. In this case, bacteria keeps increasing its area past the `self.split_area` and the `self.viability_index` is increased.

Each bacteria has a `self.viability_index`, which is initiated at 0. Every time a bacteria does not manage to divide when its area is bigger than the `self.split_area` or it is affected by antibiotics (only case for the prey species) `self.viability_index` is increased by 1. Once it reaches `self.max_viability_time` the bacteria dies. Finally, there is a possibility of a random bacterial death at every time step defined by the `self.dying_chance`.

Thus we have these basic actions that are executed for each bacteria consequently in each time step: **eat, reproduce, die**.

2.1.4 Type 1A, Predator Agent

The predator species has a unique attribute of antibiotic secretion. In order to do so, at every time step it scans to check what kind of neighbors it has. If one of the neighbors is from `self.stressed_by` list then the predator bacteria becomes stressed and secretes an antibiotic in all the neighboring and its own simulation cell. The antibiotic gets added to the Soil agent at the mentioned coordinates.

2.1.5 Type 2A, Prey Agent

In the presence of antibiotics in the same cell as the prey bacteria, there are two possible scenarios. In the first, the `self.immediate_killing` is set to True. This implies that the prey bacteria can die with a certain probability, which is equal to `self.aggressiveness`. In case when `self.immediate_killing` is set to False, upon encounter with the antibiotic the `self.energy_netto` is reduced by the `self.aggressiveness` factor.

2.1.6 Microbiome Model

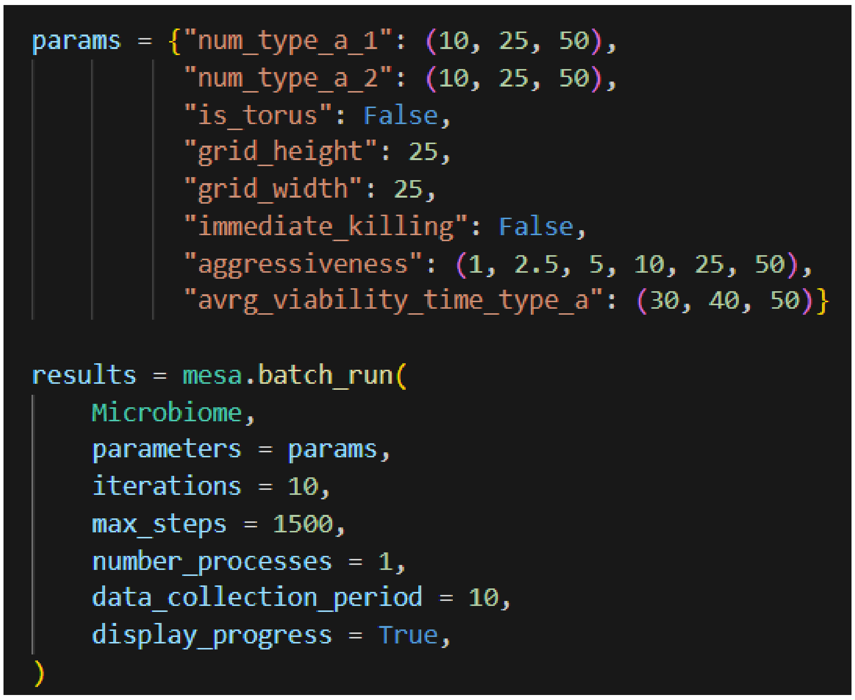
The model object is the engine of the simulation, which at the initiation creates the grid, each bacteria and places them randomly in the grid. It also quantifies the initial conditions (more on that in section **2.3**), collects the data about each species number at every timestep and finds free space for the microcolony growth: it assigns a free position to a species-specific list by checking which is the majority species in the neighborhood of the free coordinate.

2.2 Batch Simulations

In order to start the simulation following initial parameters need to be specified:

1. Number of Predators (Type A1)
2. Number of Preys (Type A2)
3. Grid Torus (Binary, indicates if the grid edges are virtually connected or not)
4. Grid Height
5. Grid Width
6. Immediate Killing (Binary)
7. Aggressiveness
8. Average viability time (Can be interpreted as bacterial life expectancy or stress resilience)

To see the influence of different initial settings different combinations of the parameters were tested (**Figure 1**). In total 1620 simulation runs, each with a unique parameters set, were conducted.



```
params = {"num_type_a_1": (10, 25, 50),
          "num_type_a_2": (10, 25, 50),
          "is_torus": False,
          "grid_height": 25,
          "grid_width": 25,
          "immediate_killing": False,
          "aggressiveness": (1, 2.5, 5, 10, 25, 50),
          "avrg_viability_time_type_a": (30, 40, 50)}

results = mesa.batch_run(
    Microbiome,
    parameters = params,
    iterations = 10,
    max_steps = 1500,
    number_processes = 1,
    data_collection_period = 10,
    display_progress = True,
)
```

Figure 1: All parameters and settings used for exploration in the batch run.

2.3 Data Analysis

All the results, observations and conclusions discussed in this section can be followed in the Data Analysis Jupyter Notebook. To distinguish the outcomes of the simulations in the batch run four final states were defined. Two pure domination states and two steady-state dominations (**Table 1**). Each of these two classes is also characterized by a species that is most abundant for example, "Prey Domination", or "Steady State, Predator Domination". The distinction between pure domination and steady-state domination is the slope of the species curves in the last 500 timesteps of the simulation. If the change in them was smaller than 7.5% the simulation was said to reach a steady state, otherwise it got a domination label assigned. Predator Domination also includes all the cases when there is no live prey left and only predator bacteria remain at the end of the simulation.

Table 1: Simulation outcomes of 1620 iterations.

Outcome	Count	Percentage of All Iterations
Predator Domination	1373	84.75%
Steady-state, Predator Domination	95	5.86%
Steady-state, Prey Domination	141	8.70%
Prey Domination	11	0.68%

To distinguish the initial conditions of each iteration the following quantifications were calculated:

1. Predator vs Prey Ratio
2. Initial Distance to the Edge for each Species (Average for each species)
3. Initial Aggressiveness (Median of all predators)
4. Prey Competition Index (Predator vs Prey Ratio, but conducted for the neighbors of the preys within a one-fifth of the grid size radius. Median of these ratios for all preys)

2.3.1 Statistical Analysis

Predator Domination is the largest final state due to its overpowering in the 1:1 ratio (514/540; the rest is Steady-state, Predator Domination), which is the main chunk of the simulation (33% of all runs). Ratios of 1:1 and smaller allow prey to survive. An initial ratio of 1:5 has mostly final states with live prey (Predator Domination is rare). Starting from 1:2.5 the balance starts quickly shifting toward the Predator Domination. Prey Domination only occurs at the two lowest ratios.

Both Steady-states appear to be the result of higher initial viability time. A decrease in the input viability pushes the simulation away from the steady state. However more towards Predator Domination and more rarely towards Prey Domination. Generally, the median initial competition index increases the survival chances of the prey and thus avoids Predator Domination. Therefore, the competition index might be the factor that tilts the balance towards one of the dominations if other parameters are at a comparable level. The distance to the edge could be another tilting factor however, its numbers are less consistent.

Aggressiveness levels are less influential than expected since all four final states occur within each of the six aggressiveness levels. The fluctuations of final state distributions are small: the biggest change is a 9% increase in Prey Domination compared between 1% and 50% aggressiveness.

2.3.2 PCA

PCA analysis shows that the prey and predator-favored states could be separated along the second PC. However, there are no clear final state cluster structures (**Figure 2**).

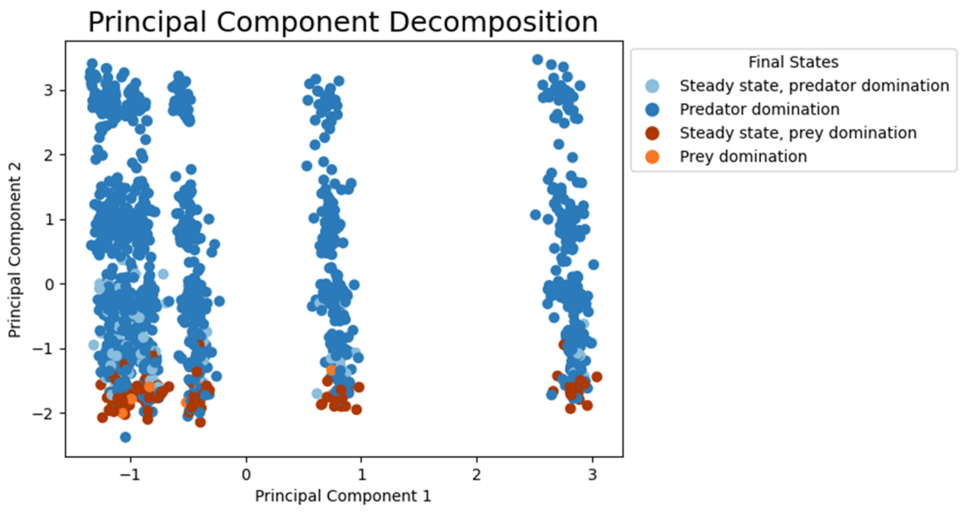


Figure 2: PCA on the 1620 iterations of the batch run.

The first PC is dominated by the input aggressiveness and the deviation from it (**Figure 3**). Such influence of the aggressiveness parameter explains the fact that the data is not linearly separable along the PC1.

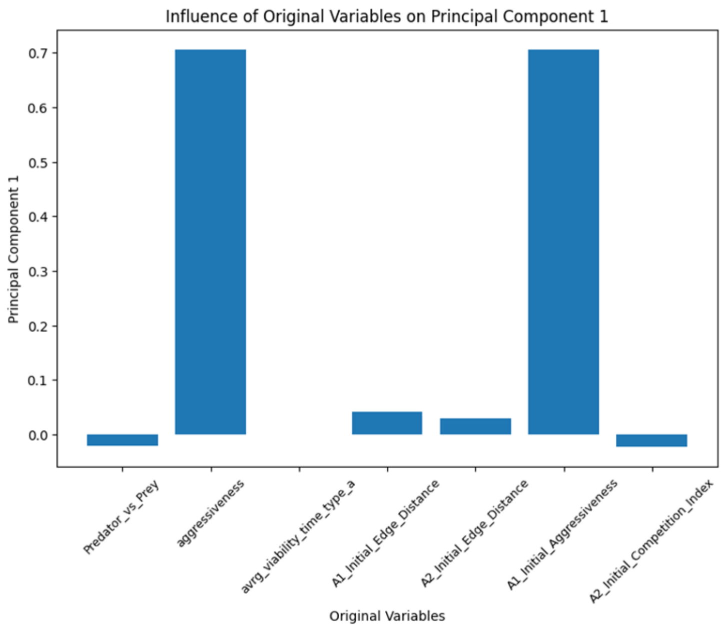


Figure 3: PC1 decomposition into original variables.

The second PC is dominated by the initial ratios and the competition index (**Figure 4**). These two parameters are not independent of each other, since initial ratios have an influence on the likelihood of prey bacteria being close to the predator ones at the simulation start. The importance of these two parameters for achieving a final state where there is no pure predator domination is highlighted by this dimension reduction technique.

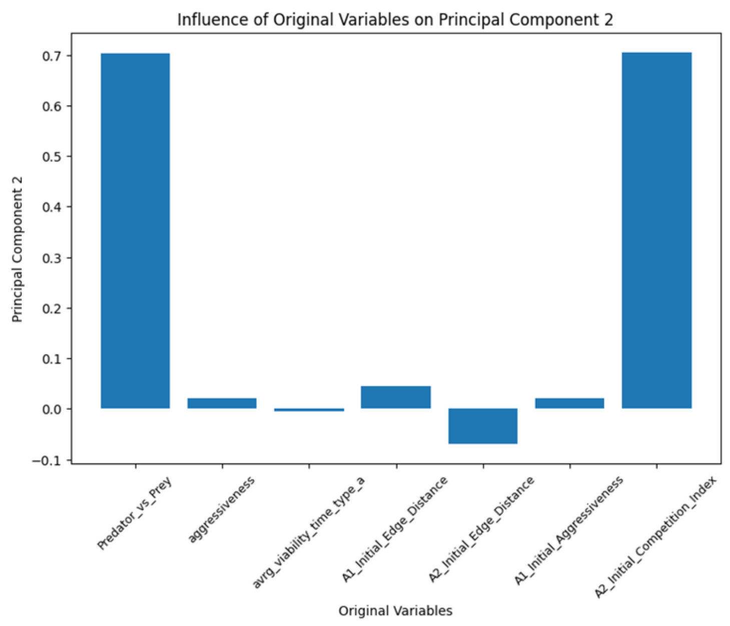


Figure 4: PC2 decomposition into original variables.

2.3.3 UMAP

UMAP did not provide any improvements in clustering and similarly to PCA highlighted the importance of the competition index. No clear grouping of the final states can be observed (**Figure 5**).

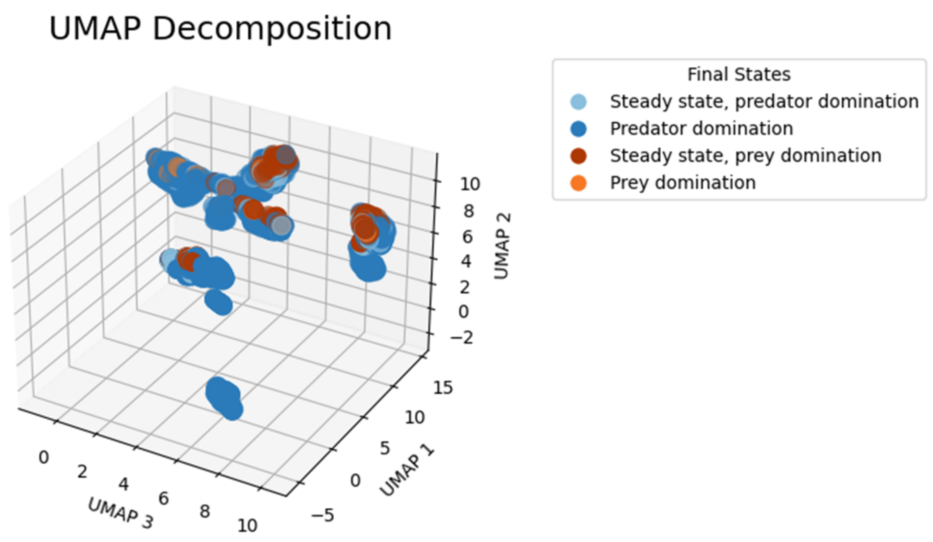


Figure 5: UMAP on the 1620 iterations of the batch run.

2.3.4 Conlusions about Parameter Influence on the Final State

Statistical analysis gave a feeling of which parameters can be tweaked to get to a desired final state. It was observed that even the highest aggressiveness level can be canceled out by combinations of other parameters (for example high viability time + low ratio + small competition index) and allow a steady-state outcome. PCA and UMAP did not provide any additional hidden trends or insights. This implies multiple suggestions for improvements. The quantification of initial conditions can be done more precisely. Testing of other parameters like grid dimensions and in general more runs can improve data separability. In general, increasing the number of iterations and replicates can only be beneficial.

3 Predator Prey Model Extended

This part of the report presents an extension to the proof of concept chapter with the subsequent explanation of the code additions.

3.1 Additions to the Proof of Concept Script

3.1.1 Type 2.XA, Prey Agent

Three mutants for the prey types were created. The only difference from the original prey agent is the parameters involved in the nutrient uptake, energy transfer, and cell division. The backbone and the implemented concepts remain unchanged. For the extended populational mutants, the initial conditions are not quantified. With this addition competition among species can be simulated.

3.1.2 Perturbations

Perturbation is a new feature that simulates antibacterial treatment of random size and symmetrical radius. It can be specified how many treatments should be done and till which time point should they all be completed. It is recommended to test this feature with the immediate killing parameter set to True, but is not compulsory.

4 Discussion and Conclusion

The final model has complex processes integrated into it by taking advantage of multiparameter interactions. This allows for observing realistic behavior of both single species (sigmoidal growth curves) and the interspecies competition (obtaining the outcomes described in the reference paper [1]). The trade-off is that an influence of a single variable change on the simulation becomes rare. The transition from one final state is rather attributed to an ensemble of variables.

As mentioned in section **2.3.4** better quantification of the initial conditions, bigger parameter scope and number of iterations should improve clustering according to the final outcomes. The end states of the simulations can also be seen as a subject for review and extension. So far all assumptions about them are based on the final species number and the slope of the growth curves. The midpoint dynamics of the competition are not taken into account at all.

Bacterial competition and antibiotic treatment can be simulated with the extended model. The code additioins allow a prey vs predator model, where one predator acts against multiple prey species. Finally, all of these implementations can be also combined together and with a few code changes can be adapted for desired needs (for example instead of random perturbation size, timing and location switch to a systematic and consistent antibiotic treatment).

The next logical step for this tool would be integrating lab-obtained data. It would be interesting to observe if the described assumptions will hold up and provide realistic fit of the growth curves or if the interactions of bacteria and their surroundings need further refinement.

References

(1) Wilmoth, J. L.; Doak, P. W.; Timm, A.; Halsted, M.; Anderson, J. D.; Ginovart, M.; Prats, C.; Portell, X.; Retterer, S. T.; Fuentes-Cabrera, M. A microfluidics and agent-based modeling framework for investigating spatial organization in bacterial colonies: the case of Pseudomonas aeruginosa and H1-Type VI secretion interactions. *Frontiers in microbiology* **2018**, *9*, 33.

(2) Tortora, G. J.; Case, C. L.; Bair III, W. B.; Weber, D.; Funke, B. R. Microbiology: an introduction. *(No Title)* **2004**.