Bacteria Colony Simulation

Richard William Polónyi
John von Neumann Faculty of Informatics, Óbuda University, Hungary
richard.polonyi@stud.uni-obuda.hu

Sándor Szénási
John von Neumann Faculty of Informatics, Óbuda University, Hungary
Faculty of Economics and Informatics, J. Selye University, Slovakia
szenasi.sandor@nik.uni-obuda.hu, szenasis@ujs.sk

2023 IEEE 23rd International Symposium on Computational Intelligence and Informatics (CINTI) | 979-8-3503-4294-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/CINTI59972.2023.10381973

***Abstract*—This paper presents the results of a project simulating microbial communities using agent-based modeling. Bacteria are placed in a discrete space and make local decisions causing the colony to grow or shrink. To achieve this, the simulation tries to simulate as many agents and their decisions as possible. For this reason, bacteria operate according to a simple set of rules. Growth is governed by Blackman kinetics, which determines the biological functions of an agent based only on its glucose or sugar uptake. The accuracy of the simulation not only requires the biological processes of the agents to be carried out but tracking and representing the assigned bodies in a 3D environment. The presented model is implemented in C++ and the visuals are handled by Raylib. The simulation can simulate up to 15000 bacteria with 30 FPS and achieves several life stages of bacterial colonies swarming, stagnation, and death.**

***Keywords*—Bacteria growth, Blackman's kinetics, Bacteria colony, E. coli, Individual base simulation**

I. Introduction

Bacteria are simple, micrometer-sized organisms, the most common cellular life form on Earth. They can colonise every possible habitat. Therefore it seems appropriate to construct ecological models in terms of individual cells and their behaviour. This paper introduces spatially explicit individual-based modelling (IbM) to microbial ecology. The great potential of IbM lies in addressing the following question: is it possible to create amacroscopic world from data on microscopic entities?

Bacteria is the basis of the simulation. At the very beginning, we are at a crossroads, if we can achieve the desired result by considering the whole colony as a large uniform biomass, then we can describe the population behavior by differential equations, but this is inaccurate. However, if more precise calculations are needed, all bacteria must be treated individually. This can be solved by the agent-based modeling discussed later.

The bacteria will be the agents. There are two different logics for determining the movement and behavior of the individuals:

- the first approach is to consider the physical forces acting on the agents at run-time,
- or the second is to create a predefined policy that the agents decide on each cycle and act accordingly.

The interactions between agents can be described by potential functions ("Newton's method") or by behavioral rules ("Reynolds' method"). When it comes to choosing between Newton's and Reynolds' methods, the following should be considered:

- Newton calculates in terms of potential forces. More specifically, in potential functions, the forces between each pair of agents are calculated and after summing them, their resultant will determine the behavior of the agent.
- Reynolds, on the other hand, calculates on the basis of a pre-specified rule. Potentials are expensive to calculate. In many cases, it is sufficient to describe the behavior of an agent in terms of a rule.

Regardless of which method is chosen, the computations are limited only by imagination, e.g. agents can be endowed with memory and make decisions based on their experience in a given situation. The rule-based system includes cellular automata (cellular automata), but this is beyond the scope of my research, so this paper will not deal with it [1, 2].

The medium is the substance where the agents (bacteria) will be located. Here again, the question arises whether the implementation of the medium is necessary, or whether it is negligible in simpler models and can be considered a vacuum. One can also wedge in here one of the most famous agent-based studies, where Reynolds wanted to simulate birds. The model used there did not explicitly define the medium either, but only considered the forces acting on the agents (birds). The objects to be avoided are given a repulsive potential, thus avoiding collisions [3]. Otherwise, if it is necessary to specify the more important elements of the medium, we can specify them as the force acting on the agents or other chemical particles.

To store the agent's positions in a 3D space, the approach is simple; it's all about dividing the space into areas of equal size. These areas can be thought of as cells. In the simulation, all objects are associated with the cell they overlap. If the cells are large enough, it can be said that an object can only collide with other objects that are in the same cell as itself or its neighbors. This makes it easy to narrow down the objects that should be tested in the narrow phase. Of course, there are exceptions, such as when an object is located exactly on the border of two cells, in which case it is stored in the cell where its center is located. Another approach is that if the size of a cell is the same as the size of the objects and the objects are not very different in size, then we only need to check around the adjacent cells to see if there are any elements in them [4].

Cells can contain either an array or, in fact, any data structure, be it a linked list or even a hash map, allowing multiple bacteria to be counted in a single cell. Whatever size is chosen, it is always worth paying special attention to the size of the cells, as this is one of the most important factors when we consider the speed of the data structure [4].

II. Related works

A. BacSim

BacSim is based on an individual-base modeling approach, i.e. it uses an agent-based approach to model bacteria, more specifically the bacterium Escherichia coli. The simulation focuses on growth from a single E coli cell to a

whole colony. The simulation is written in object-oriented Objective-C.

Based on the Gecko ecological simulation, it is built using the Swarm toolkit multi-agent modelling software. The model used here describes the bacterial substrate uptake, metabolism, division and degradation upon death. This allows the model to be easily adapted to model other species. The model works with 8 parameters. From these parameters: maximum substrate uptake, initial volume and the limiting weight of division at bacterial emergence are obtained from a normally distributed random generator. The remaining parameters are derived from the parent. The simulation simulates the space as a finite 2D grid. Diffusion is performed using it. Each lattice has a separate nutrient volume, so it tries to model the heterogeneous environment. Diffusion smooths out the substrate differences between cells at each run of the program cycle ensuring that the bacteria do not consume the nutrients in the cells too quickly. Our method uses a Quadtree data structure to partition the space to optimize the physical engine. Interestingly, for each agent, the largest square that can be placed in the body of the agent circle is counted separately, as this was used to simplify the substrate weight to volume calculations. The dry matter content of cells determines their growth. Cell mass determines the maximum substrate uptake. After uptake, the amount of energy required to sustain the uptake is subtracted from the substrate taken up, and only the remaining substrate is added to the cell mass. The uptake is based on the Michaelis-Menten equation [5].

Division is also determined by the dry matter content of the cells. Division results in two cells of nearly equal mass. Division in this model is instantaneous. A colony, also known as a biofilm, grows uniformly by shifting from one agent to another during division until one agent covers the other [5].

B. INDISIM

IDIMSIM, the INDividual DIScrete SiMulation program, was published by the MOSHIBO research group. This approach is also agent-based, but instead of looking at growth, it looks at the so-called lag phase. This research, unlike BacSim, does not use Monod kinetics but Blackman kinetics to describe bacterial growth [6].

Blackman kinetics was published by Blackman F.F. in 1905. It is similar to the Monod kinetics used by BacSim. Here again, growth at low substrate concentrations depends on the substrate, while at high concentrations, growth is limited by the nutrient values in the substrate. In microbiology, it is now accepted that Blackman kinetics is not accurate and is therefore little used, but INDISIM's agent-based model has been shown to give better results than Monod kinetics. A particularly interesting feature of the simulation is that it is a state-free simulation, as mentioned above, i.e. it does not capture the state of a step but calculates it for each step using the elapsed time and initial parameters. Accordingly, no physical bodies or collision handling are used here, as each aspect of the colony, i.e. the position of the agents, is described by a function. Nutrient uptake is described using Brownian motion. Brownian motion is nothing more than, per unit of time, a Gaussian distribution randomly determining their position. In this way, it is possible to model substrate diffusion in a whole-life manner. Since the denser the substrate concentration, the more the molecules collide, the less substrate they have, the greater distance they can travel without colliding so that molecules migrate from the cell with the denser concentration to the cell with the less dense concentration [6].

III. METOLOGY

A. Diffusion

The diffusion is based on Fick's law. We consider the substrate difference between two cells, denoted by delta Δcs. The distance between the two cells is denoted by d. The flux between the two cells is defined by *Js*. This gives Fick's law

$$Js = -Ds * \frac{\triangle Cs}{d}$$

B. Substrate

The space is located on a 3D plane. This plane is divided into cells. These cells are counted as a 3D vector. Each element contains only the concentration of the substrate. These cells will be used to simulate diffusion. The cells also play a role in substrate uptake. The bacteria will only take up all the substrate from the cell in which their center is located. The space is finite, so the colony can only grow up to a certain limit.

C. Plane

These cells are represented by objects, which can store pointers to specific objects, thus avoiding copying the entire object from one cell to another when deleting, adding, or just smoothly changing position.

D. Growth kinetics

The Blackman kinematics was chosen for its simplicity. It is also important to point out that these kinematics are interpreted separately for each agent. The agents are stored in a vector, as C++ can cache them quite nicely, so it is possible to quickly traverse the array and perform operations on them.

E. Bacteria movement

The bacteria do not move on their own. At each iteration of the simulation, there comes a point when, after growth and division, the bacteria locally calculate all the overlap and then shifting the opposite direction because if we push the agents away from each other by the total overlap, we just push them into each other, to avoid this, you break the shifts into smaller steps and this trick allows you to handle overlaps in a much more resource-efficient way.

F. Uptake

For each bacteria in a program cycle, its maximum uptake differs. With the mechanistic approach, an agent takes up substrate depending on its weight, but this means that in more abundant situations, it would take up too much nutrient. At each step, the value that the bacterium is allowed to take up is compared with the maximum value that it can take up, and then the smaller value is allowed to be taken, denoted by *U*. The uptake at each step is calculated using the following equation: $Umax = Z * m * a$, where Z is a random value obtained at the time the agent is created. The mass of the bacteria is m. The value of a depends on the shape of the bacterium; in this simulation, we consider the agents as spheres, hence a = 2/3 [6].

G. Metabolism

The metabolism of the bacterium converts the ingested substrate into biomass and end products, but it cannot convert all the ingested nutrients into mass in one go, only at a certain efficiency, which the constant *Y* will provide.

m = (m + (U - mI) Y), where *U* is the value taken up at a given time step, *mI = m * I*, where *m* is the weight and *I* is the unit of maintenance energy [6]. The maintenance reaction also generates a final product from biomass. It is also discharged, which is not considered from any point of view. The maintenance is also adjusted to the amount of dry matter.

H. Cell lysis

If it is not possible to cover maintenance from the substrate we take in, starvation will reduce bacterial cell numbers. The cells die when their biomass falls below a certain level. After death, they revert back to substrate. This allows some members of a starving colony to survive, grow and even divide under less than ideal conditions.

I. Cell division

A certain weight limit must be reached to initiate the division. Hence, there is one important parameter for the division, and that is the minimum division weight, denoted by (*mR*,*min*). The division occurs immediately; when the agent reaches the appropriate weight, it is immediately divided. Since the substrate uptake is not the same, and therefore the growth rate is not the same, the weights of the two agents produced at the time of division need not be considered separately since they will be different by default; only the weights of the initial bacteria in the simulation need to be determined.

J. Uniform Grid

The grid requires to pass in the constructor a lowerbound and an upperbound, which are Vector3D types that contain x, y and z coordinates. It also needs to specify the dimensions (Vector3D), which determines how many cells should be placed on the given axis in different dimensions of the space. Finally, maxObjects, which is an integer type, is needed to preallocate the maximum size needed for staring all agents. This class is responsible for storing all the agents and tracking their spatial movement. The agents themselves are stored in a separate vector. This vector is initialized at program startup, and its reserved (*site_t* amount) method allows to reserve enough space in advance to store the maximum amount of bacteria without having to reposition itself in memory. The other data structure is a 3D vector that stores GridCells. A GridCell contains only one vector, which is needed to store the objects it contains. It is important to emphasize that only pointers are stored here, so you only need to change what the pointer points to when updating agents.

The most important task of UnifromGrid and other space-division data structures is to quickly find all nearby neighbors of a given element. Naturally, the efficiency depends largely on the size of the search space and spatial cells. In this case, for n-body simulations, it is most optimal to set the cell to twice the radius of an agent. UnifromGrid is not only popular for its ease of implementation but also for its O(1) search in ideal cases. In many cases, it is necessary to use a hash function, but in this case, with a little spatial geometry, you can get around this and avoid hash collisions. The function can be used to assign any coordinate of the space to a given cell (Fig. 1)

```
Vector3D Grid::_getCellIndex(const Vector3D& position)
{
    float x = baciUtil::sat((position.x - _Lowerbound.x) / (_Upperbound.x -
_Lowerbound.x));
    float y = baciUtil::sat((position.y - _Lowerbound.y) / (_Upperbound.y -
_Lowerbound.y));
    float z = baciUtil::sat((position.z - _Lowerbound.z) / (_Upperbound.z -
_Lowerbound.z));

    float xIndex = std::floor(x * (_dimensions[0] - 1));
    float yIndex = std::floor(y * (_dimensions[1] - 1));
    float zIndex = std::floor(z * (_dimensions[2] - 1));

    Vector3D ret = { xIndex , yIndex, zIndex };
    return ret;
}
```

Fig. 1. Function to assign space coordinate to a given cell

The input parameter of the function is a position. From the position, subtract the lower boundary of the space and divide it by the distance of the space or the difference between the upper and lower boundaries of the space. The resulting value is given to the sat function. Saturation is a mathematical function that does nothing more than ensure that the given value is between 0 and 1. Finally, it is multiplied by the number of cells in that dimension. This ensures that the search is O(1). Unfortunately, of course, this applies to the cell, we still have to search for the given element in its corresponding GridCell.

K. Grid Update

As the colony grows, so do the spatial positions of its branches. Updating the position means removing the agent and reinserting it into the Grid, which is costly and should only be used when necessary. To ensure this, once an agent is placed, it saves the calculated cell coordinates. In the case of an update, when the agent is assigned, the corresponding cell should first be recalculated and only updated if it does not match the previous cell.

L. Neighbors search

This is the most important function of the UniformGrid. The function expects an agent and a search distance. Subtract half of the search distance from the agent position; this will be the lower bound, and add half of the search distance; this will be the upper bound of the search area. We assign the spatial cells to these two points using the method already described. This is convenient because in this way, we can walk all the cells between the two points in a nested loop and check the agents in them to see if they overlap with the agent specified in the function parameter; if so, then the pointer to that agent stored in the grid is placed in a vector and returned as the result of the function.

IV. EVALUATION

As already mentioned, the point of the simulation [7] was to simulate as many agents as possible in real time [8,9]. The implementation performed better than expected, but this is due to the optimization of C++ O2. The numbers are visualized in Fig. 2.

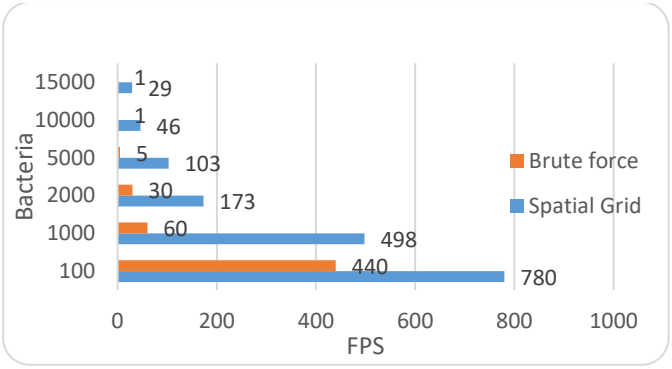

Fig. 2. The vertical axis represents the number of agents. The horizontal axis represents the number of frames.

The Grid is on average 15x faster than the naive approach. As a point:

- 100 agents: 780/440 = 1.77x
- 1000 agents: 498/60 = 8.3x
- 2000 agents: 173/30 = 5.77x
- 5000 agents: 103/5 = 20.6x
- 10000 agents: 46/1 = 46x
- 15000 agents: 29/1 = 29x

The life stages of the colony are visible. The initial swarming is followed by stagnation and then the death of the colony. These processes can be seen in Fig. 3.

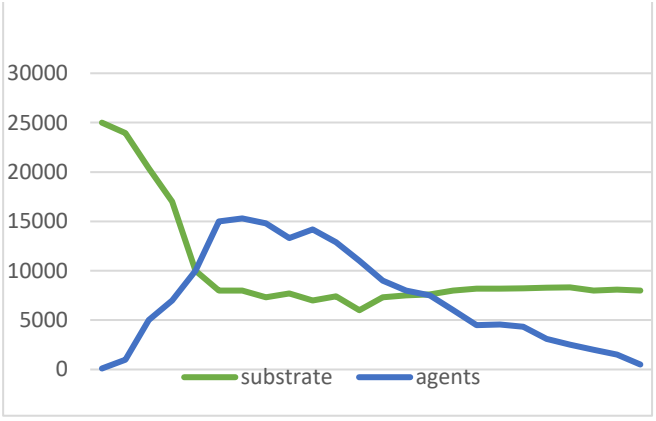

Fig. 3. Number of agents

Initially, as long as the maximum substrate uptake is abundantly available for all agents, we can see that they are in the swarming stage. At about 10,000 agents, we reach the point where not all agents can take up maximum glucose per round, but the decline is not yet apparent because there is still room for expansion, where there is still plenty of glucose in the cells and so the graph does not yet show internal death, because there is still much more division than cell death. And they run out of new cells are about 15,000 agents.

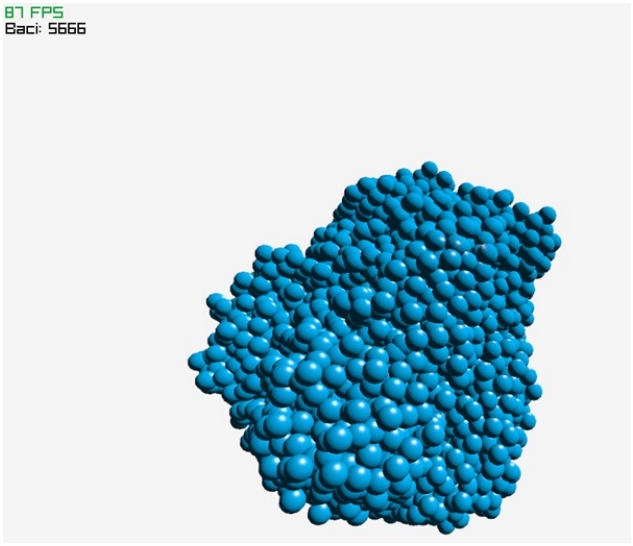

Fig. 4. The stage of swarming, here you can see that there is uniform growth in all directions, because the agents do not yet differ that much form the initial values like weight and maximum substrate uptake etc.

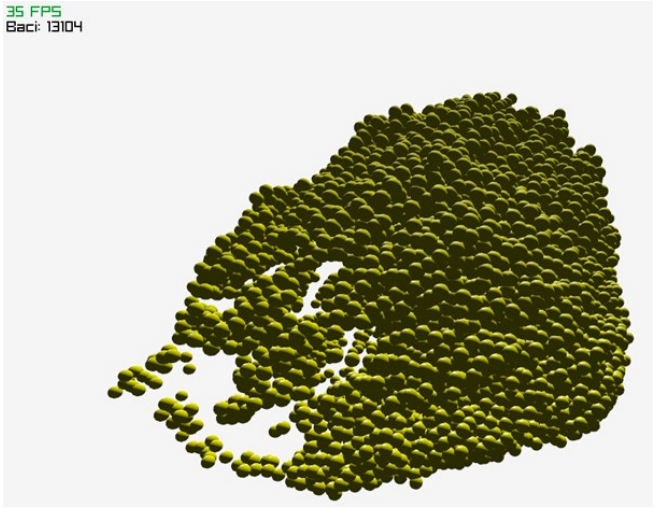

Fig. 5. The colony started on the left side and over time spread to the right side, where there was still substrate. Notice that the left side is now only fed by the cell lysate with the remaining bacilli. But the growth on the right side is still greater than the death on the left.

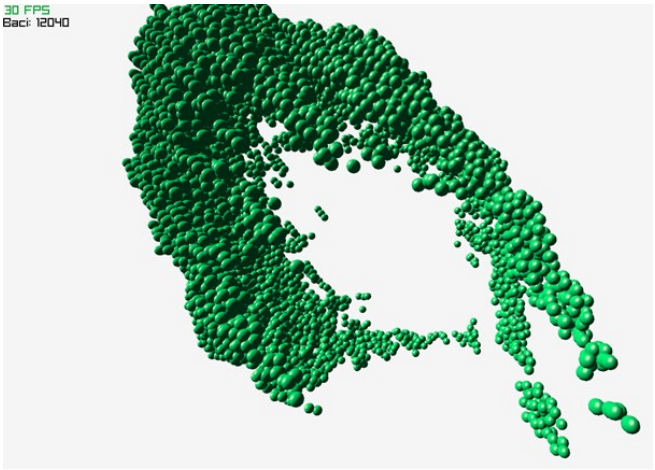

Fig. 6. Here we have the agents at the end of stagnation and the onset of decline.

V. CONCLUSIONS

The simulation can simulate 15.000 branches at 30 frames per second, and the three stages of the colony can be observed in the simulation. The simulation is a good example that you don't need a very complex data structure, often, a simple data structure can do a lot if you know what is behind it. Uniform Grid ensured that when looking for neighbors, with the naive approach we had to look at an average of 224 neighbors per 10,000 agents, while with Grid we only had to look at 7. We had to break the collisions into smaller steps because if the bacteria are pushed away from each other by the overlapping full vector space, they will push into each other.

It would be worthwhile to make the model parallel. For example, diffusion takes a lot of time and resources. GPU could help with this, and uniform grids already have been implemented with CUDA. The substrate could be supplemented with more nutrients like Nitrate and Sodium. Together with this, we would then be able to simulate more types of bacteria. The graphical library supports user interface design. Implementing a proper user interface could also be a nice upgrade.

ACKNOWLEDGMENTS

The authors thank the High Performance Computing Research Group of Óbuda University for its valuable support. The authors also thank NVIDIA Corporation for providing graphics hardware for the experiments.

REFERENCES

[1] J. Juhász, Mikrobiális közösségek koordinációjának vizsgálata ágensalapú modellekkel, PhD Thesis, 2018

[2] D. Bihary, A baktériumok quorum érzékelésének ágens alapú modellezése, Phd Thesis, 2014

[3] Hoveyda, Amir H., David A. Evans, and Gregory C. Fu. Substrate-directable chemical reactions. Chemical reviews, vol. 93, no. 4, 1993, pp. 1307-1370.

[4] C. Ericson, Real-time collision detection, Crc Press, 2004

[5] Jan-Ulrich K. Ginger B. Julian W. T., BacSim, a simulator for individual-based modelling of bacterial colony growth. Microbiology, vol. 144, no. 12, 1998, pp. 3275-3287

[6] C. Prats, Individual-based modelling of bacterial cultures in the study of the lag phase, PhD Thesis, 2008

[7] Spodniak, Miroslav, et al. "Methodology for the Water Injection System Design Based on Numerical Models." Acta Polytech. Hung 18 (2021): 47-62.

[8] K. Czakóová, O. Takáč, The application of modern technologies for image processing and creating real model in teaching computer science at secondary school, ICERI2020 Proceedings, 2020, pp. 6180-6187.

[9] Namestovski, Ž.; Kovari, A. Framework for Preparation of Engaging Online Educational Materials—A Cognitive Approach. Appl. Sci. 2022, 12, 1745.