0.1 Introduction

The Cell Behaviour Modeling Platform (CBMP) is a simulation environment for agent-based models of cells. It is a research tool that can simulate a variety of systems in which cells move through continuous space in response to diffusing environmental stimuli and each other. It can be used to create kinds of cells and environmental stimuli, where the cells behave according to user specified parameters.

It has been developed in Julia, a relatively new language that is highly optimized for performance but nonetheless offers readable syntax and short development time. This has resulted in a platform that can simulate many individual cells within a continuous space environment at good speed, whilst still being very accessible and easily modified to suit new users needs.

The CBMP can run as a command line interface or as a graphical user interface. The command line interface is designed for fast expert use, whereas the GUI has been developed to be easier to use, and to display the simulation in real-time. Either interface can output the results of simulations as a binary file to allow quantitative analysis in Julia or Python. The CBMP has been developed with a publicly available codebase in Julia, which can readily undergo modification by end-users.

The CBMP uses an agent-based model to model cell-behaviour, and partial differential equations to model their environment. An agent-based approach is ideal for modelling the cells as they are heterogeneous entities with complex interactions. For the cytokines, a continuum approach is essential because they are heterogenous and are too numerous to be modelled individually.

In the absence of an environmental stimulus, the cells carry out a walk that includes elements of persistence and randomness. When environmental stimuli are present, cells decide where to move based on several factors: the evaluation of chemical stimuli, the cell's current direction of motion and a random component. The growth of cells depends on whether they have sufficient space around them. The model implements simple rules describing how cells interact with each other and their environment, which gives the opportunity to observe emergent behaviour on the simulation and to compare this to real world experimental data. This comparison can be used to check whether the mechanics underlying the simulation are approximately correct or to discover previously unknown characteristics about the system undergoing investigation.

0.2 Installation

The CBMP has been developed in Julia 0.4. in RedHat Enterprise Linux Workstation release 6.2, and has also been tested on Ubuntu 14.04 in Julia 0.3. As the Julia language is updated on a nightly basis, some of the syntax used within the platform can become deprecated in as little as 24 hours. Whilst this has yet to pose a significant problem, we anticipate that the next large Julia upgrade (0.5) may require the platform to be updated. The platform should run on any operating system that has Julia installed, however there may be slight differences between them.

To run the CBMP, first Julia must be installed. If this has already been done, please skip to the Required Dependencies section.

0.2.1 Detailed Instructions

To install Julia, follow the instructions on the website, http://julialang.org/downloads/. Similarly, install Python at www.python.org/downloads if you do not have it already. The Julia packages Winston, Tk and PyCall are most easily installed using the interactive Julia prompt. The Julia prompt is opened by entering into the command line:

```
> julia
```

To install the required packages, enter the following into Julia's interactive prompt:

```
> Pkg.add("Winston")
> Pkg.add("Tk")
> Pkg.add("PyCall")
```

To return to the command linter, enter `> exit()` You can install the python package *Pickle* by typing into the command line:

```
> pip install pickle
```

If all of the dependencies are already installed, then all you need to do is download the repository and run CBMP.jl.

Dependencies

The installation depends on:

- Julia 0.3 - 0.4 with:
  - Winston
  - Tk
  - PyCall
- Python (tested on 2.8 and 3.4 but it should work for other versions) with:

– Pickle

Once you have installed the dependencies, all you need to do is download the repository and run CBMP.jl. If you have git installed, you do this by entering into your command line:

```
>git clone https://github.com/RyanCarey/abm-platform.git
>julia CBMP.jl
```

The software comes with 2 different methods of inputting parameters and starting simulations: an easy to use GUI and a command line version. The GUI has been created with experiment design in mind; it gives the user fine-grained control of all parameters that describe the behaviour of cells and the way in which environmental stimuli diffuse through the environment, whilst providing immediate feedback.

The command line version, whilst being more complicated to operate, is much faster at performing simulations due to the lack of simulation display and increased parameter input speed. Parameters are input directly into the code and as such requires slightly more expertise to use. This option is intended to be used when experimental set-up is already decided and multiple simulations are required to be run to collect data.

0.2.2 The Graphical User Interface

The overall structure of the graphical user interface is that of a main menu that controls the most important parameters, and three submenus that are used to control the details pertaining to diffusion, cell types and borders. In all menus, help buttons are available to provide detailed instruction on the function of each parameter.

Main Menu

The main menu allows the user to alter environment-wide parameters such as starting number of cells and the size of the environment (Figure X).

Figure X: The program's main screen, which contains settings for some of the major features of the model, and has buttons which open detailed settings. The settings are as follows:

1. **Number of cells:** The total number of cells at the start of the simulation.
2. **Number of timesteps:** The number of iterations made by the simulation before the simulation is terminated. At each iteration, one randomly chosen cell is updated.

3. **Width of Environment:** This denotes the width of the simulation environment in some arbitrary units.

4. **Height of Environment:** This denotes the height of the simulation environment in some arbitrary units.

5. **Energy Loss Coefficient:** This specifies the factor by which cellular momentum is multiplied upon each contact event.

6. **Type of Diffusion:** Allows the user to choose between integrative (constant) and normal (dirac) diffusion. These options can be further tweaked in the diffusion settings.

7. **Diffusion Settings:** Opens the menu for altering diffusion settings. See Diffusion Settings below.

8. **Cell Type Settings:** Opens the menu for altering cell type settings. See Cell Types below.

9. **Border Settings:** Opens the menu for altering individual border behaviours. See Border Settings below.

10. **Display / Write:** Selecting these options will display the simulation if desired and allow an output of the results of the simulation to be saved to file as a binary 'pickle' file. Note that displaying the simulation slows the process considerably, and should be turned off for long simulations.

Diffusion

The diffusion menu (shown in Figure X) has the settings that control how ligands will spread through the cell's environment. The results of tuning the diffusion parameters can be visualised and controlled interactively using a slider.

Figure X. The program's diffusion menu, which contains settings for controlling the how ligands diffuse through the cell's environment. On the screen's right is a graph that shows how the concentration gradient will change over time, where time is interactively changed using the slider in the lower right. The diffusion menu settings are as follows:

1. **Ligand Receptors:** The number of points on the cell surface where ligand concentration is sampled. These are evenly spaced out along the cell surface.

2. **Initial Concentration:** This is the A coefficient in the diffusion equation. Increasing this number proportionally increases the ligand concentration. Note that this coefficient only modifies the visualisation plot and does not affect the simulation.

3. **Gradient Coefficient:** This is the D coefficient in the diffusion equation. It is usually called the diffusion coefficient. Decreasing this number will increase the slope of the diffusion. It is equivalent to reducing the time of diffusion. Note that this coefficient only modifies the visualisation plot and does not affect the simulation.

4. **Time Limit:** This is the tau coefficient within the diffusion equation. It defines the time during which sources will emit ligand. In normal diffusion, this coefficient is absent. Note that this coefficient only modifies the visualisation plot and does not affect the simulation.

5. **See Diffusion:** Updates the graph to the right with the parameters specified in 2-4. Note that the parameters specified above do not affect the simulation and are only used in graphing the potential.

6. **Type of source:** Select which type of source is required.

7. **Number of sources:** The total number of sources present in the simulation from the start.

8. **Locations and Coefficients:** Allows the user to select the locations of each possible source, and change its characteristics. For more information see below.

9. **Diffusion Over Time:** Allows visualisation of the diffusion over time as per the coefficients given in 2-4.

Source Location

The source menu (in Figure X) controls for where the ligand is emitted from, for how long, and with what gradient.

Figure X. The source menu, which allows the user to specify where ligand comes from, and how it is produced.

1. **Location:** the source location is specified here. In the case of a line source, this is a single x-ordinate. In the case of a point source, x- and y-ordinates would be shown..

2. **Parameters:** Allows each individual ligand source to be customised, following the rules described above.

Cell types

The cell type menu controls most of the complexity of the CBMP's models by allowing a wide range of parameters for up to four cell types (Figure X).

Figure X. The cell type window, which allows the user to specify parameters for each cell type. The parameters are:

1. **Ratio:** The ratio of the total number of cells given over to that type upon the start of the simulation. Ratios are normalised and as such do not have to sum to one.
2. **Growth Rate:** The rate at which a cells area will increase per timestep.
3. **Division Threshold:** The ratio of current area to initial area at which a cell division event will occur.
4. **Average Speed:** The average distance a cell can travel in one step. Uses the same arbitrary units as other distance variables.
5. **Average Radius:** The average radius of cells at the start of a simulation. This is also used in calculating cell division.
6. **Ligand Response:** This specifies a value which multiplies a cells proposed move direction in response to its detected ligand concentration. A value of 1 indicates that a cell will travel towards a source, whilst -1 will repel the cell. Note that specified randomness will dilute this effect.
7. **Stem Threshold:** The ligand concentration boundary that defines what is considered stem cell niche, and what is not.
8. **Death Rate:** The chance that at any time step, the cell in question will die.
9. **Persistence:** The percentage of a cells move that is derived from its last move.
10. **Randomness:** The percentage of a cells move that will be random.

11. **Inertia Threshold:** The speed at which a cells momentum will overcome other cells inertia, causing bouncing.

12. **Signal Noise Ratio:** The ratio of maximum ligand concentration to mean ligand concentration detected by individual cells receptors (default: 8). If the ratio is below this value, the signal is deemed to be lost to noise, and the cell will disregard the inputs and move in a persistent random manner.

13. **Colour:** Used to distinguish cell types in simulation display. Possible cell type colours are: Red, Blue, Magenta, Green and Yellow.

14. **Special Options:** These represent specific special functions cell types can have.

    (a) **Left Placed:** Selecting this option will initialise all cells of the particular time close to the left wall of the environment.

    (b) **Stem Cell:** Selecting this option will allow this type of cell to have stem cell behaviour. A stem cell will change the type of its progeny depending upon its local ligand concentration and the stem threshold specified in the main menu. Progeny will be either of the same type (stem cell replication), or the type below (1 -> 2 -> 3 -> 4). Note that type 4 cells cannot be stem cells. Deselecting this option means that all types will only produce the same type.

    (c) **Stick to Source:** If the detected mean ligand concentration is above the stem threshold, the cells speed will be divided by 10, effectively sticking it in place.

Borders

The last menu is the borders menu, which allows the user to decide which individual borders will respond to cells by reflecting, absorbing or removing them.

Figure X: The border menu**:** Each of the four borders can independently be set to perform reflecting or absorbing collisions, or to remove cells.

0.2.3 The Interactive Prompt Interface

Alternatively, the CBMP can be accessed from Julia's interactive prompt. The command line option is intended to be used as a way of performing

multiple identical simulations in order to collect data. As such the option to visualise simulations is disabled, and the option to save data to file is enabled by default. In order to run the CBMP in the command line, the user should simply type in the terminal:

>julia (to enter the REPL)

>include(" run_sim.jl")

>run(Number of Simulations, *Optional Preset Parameters*)

The command-line argument specifies the number of simulations. Any simulation specific parameter changes can be made in the file " run_sim.jl". This approach to parameter customisation was made in light of the amount of parameters that can be altered, as it would be unfeasible to include all parameters as function arguments. Any parameter that can be changed within the GUI is available for alteration within the command line version. If the user desires to replicate the simulations we discuss later, the command line version can be run with an additional function argument. These are " stem", " migrate_random" and " migrate_non_random". These presets already have the parameters that our simulations ran saved within them. The user can copy this code and substitute parameters to store their own simulations.

Comments are provided in *run_ sum.jl* so that users can see what different parameters do. These parameters are laid out similarly to the graphical menus (For instance, the cell type characteristics are shown in run_sim.jl), and are written as Julia variables.

Figure X: run.jl: choosing cell type parameters

0.2.4 Accessing Binary Outputs

Instead of viewing simulations on the graphical display, they can be stored to be used later. To reduce storage space, these are stored as binary files using Python's " Pickle" module (Pickle is in turn called using Julia's PyCall). Since it would be highly redundant to store the locations of all cells at every timestep, the locations of every cell are by default stored every 250 timesteps. Each pickle file documents the time that the simulation took place and the all parameters used.

This pickle file can easily be unpacked into an array in Julia or Python. In each language, this is done using an " unpickle" function that is provided in the repository in the " unpickle.jl" and " unpickle.py" files respectively. In Python, further utilities are provided in " unpickle.py" for conveniently

processing a series of simulations. "get all" can take the list of filenames printed out by the command line utility and store the cell positions in each simulation in an array. "stack" can be further used to combine the simulations together into one time series. All of these are provided for conveniently analysing the output from the CBMP.

0.3 The Model

The Modelling Framework

The framework of the model can be seen briefly outlined in Figure 1. To begin with, cells are allocated to non-overlapping locations using sequential random placement. If not all cells can be placed in the size of environment specified, the model will stop trying to place cells and report this. However, the simulation will then begin with the amount of cells that had already been placed. Once completed, the simulation will then iterate over five steps for the amount of user specified time steps, shown in Figure 1. These events take place in two-dimensional continuous space.

Figure 1: Global overview of the model. The five main stages of the simulation are cell death, environment dynamic, movement, cell growth and mitosis.

Initialisation

The agents are placed in a rectangular environment, in which space is continuous and time advances in discrete timesteps. The cells are positioned randomly, either on one edge of the environment (as with haematopoietic stem cells in a niche model), or across the whole field, depending upon their specified type location. If a proposed location overlaps with another cell, then a new location is randomly generated and proposed (random sequential placement). If many attempts are made to fit a cell in the given field, but the cells are simply too numerous and too large, then the simulation stops placing new cells and carries out the simulation with the amount of cells it had so far placed correctly.

The cells are modelled as agents with rigid, circular edges of varying radius. Each cell has a type, which determines its behaviours. Cells of the same type grow, move and divide at similar rates with a slight individual randomness. However, they differ in their physical characteristics - their radius, location and direction of travel.

Iteration

Once the cells are placed, one cell is selected for action each turn. This cell proceeds through the four major cellular functions: death, movement, growth and division. The cellular functions performed by the platform try to encapsulate most of the basic behaviours of cells. There is scope within the platform for the way these behaviours are implemented to be changed or removed, or new behaviours added.

Death

Cell death is modelled as random, and uniform for a given type of cell. This assumes that cell death is independent of the cell's environment, and in particular, independent of prior deaths that have occurred in that cell's vicinity. In reality, apoptosis is mediated by environmental factors, and this could be included with the development of a more sophisticated environmental model.

Movement

When a cell moves, it is stochastically assigned a vector that determines its change in position from timestep n to timestep n+1. If the cell's proposed position at time n+1 overlaps with another cell, or with the wall, then the overlap-resolving function is called and it tentatively suggests a new movement vector for that cell and any other cells that it collides with. The alternative movement vectors are then checked recursively for collisions.

The magnitude of the initial movement vector is drawn from a Chi-squared distribution with two degrees of freedom. The angle is drawn from a combination of i) a distribution that is proportional to the amount of a ligand present at the cell membrane and ii) a uniform random distribution iii) the current direction of motion of the cell.

When a movement is proposed for a cell, there are two ways that its movement can be interrupted: by collision with the wall, or by collision with another cell. If a cell is reflected off the wall, then it bounces of the wall as per the border parameterization, and this new movement vector is assessed for any overlaps. If there is overlap with another cell, then both cells bounce off each other, and their new movement vectors are assessed for overlap. These new movements are checked recursively until all cells are placed in non-overlapping locations and the movement phase concludes.

Growth and Division

Cells grow according to a user specified growth rate that depends on the cell's area. Because only one cell grows per timestep, the growth rate is adjusted to account for the number of cells in the simulation. Whether a cell divides depends on two factors: i) the ratio of its size to the size at which that type of cell is initialised. If this ratio exceeds the specified threshold, then the

cell will divide. ii) the adjacent space available. If there is insufficient space to place two daughter cells, then the cell will refrain from dividing. When cells divide, each of the daughter cells have half of the area of the parent cell.

To model stem cells, rules have been specified to determine which daughter cells they will give rise to. For stem cells, if the local ligand concentration is above the user-specified threshold, then division is likely to produce two stem cells. If the concentration is below the threshold, the division will result in one or both of the daughter cells becoming progenitor cells instead.

The CBMP allows generation of ligand as an instantaneous pulse, or continuously over finite time. There can be multiple sources, so long as they are all of the same variety. Modelling how cells are affected by these ligands depends on modelling its diffusion. These environmental models are best understood by beginning with the instantaneous case.

Instantaneous release of ligand

The equations governing the diffusion of ligand from an instantaneous source can be derived from the example of a single particle P that can move along an axis, as in Figure 2.

Figure 2: a particle moving stochastically through one-dimensional space

Consider a particle P, that, as in Figure X, has an equal probability of moving left or right. Let the length of each jump be *a*, and the time elapsed during each jump be *tau*, and assume that the particle takes another step immediately as soon as each previous step has concluded. Assume, without loss of generality, that at time t=0, the particle is located at x=0. After N jumps of which n are in a leftwards direction, the particle will be located at x_n = *(N-2n)*. After N jumps, there are ${}^{N}C_{n}$ ways for the cell to end up at x_n of 2^N possible moves in total, therefore the probability of being located at x_n is:

If each timestep *tau* is much less than 1 and the number of timesteps N is much greater than 1, then our model increasingly resembles continuous time. In the limit, using Sterling's approximation and some algebra, this gives a continuous model that is continuous in time:

The last expression still includes a spatial parameter, which was previously taken to be discrete:

However, we can assume that ? is much greater than a, and therefore the model will be correct in continuous space. The probability of being at time t between x and x+dx with ? >> dx >> a is:

If we let

then *dp*can be expressed as a function of x, t, and D:

These particles are assumed to be sufficiently numerous that their mass behaviour is guided by this probability distribution. Thus the concentration is proportional to this probability density function:

Sustained release of ligand

In general, it is more realistic to model biological signals as occurring over a finite time, rather than instantaneously - a Dirac pulse cannot occur in nature. Thus the CBMP allows a more complex model with constant output of ligand from a source. This model has one parameter, a duration over which the ligand is secreted, tau_0.

We have used the following equation to model the concentration:

In this model, tau_0 is the duration over which the source emits ligands. Contrary to the previous model where the input is only a Dirac pulse, the number of ligands in this model is increasing during the interval [1,tau_0]. Figure 3 highlights the differences of the two models.

Figure 3:*On the left, a plot of the concentration of ligands over time at a distance 1 from the source with the normal model. On the right, the same plot with the integrative model. The parameter equals 100 steps on the left.*

If the user wants to have a faster display, it is faster to use the normal model as it doesn't use any integration.

Figure 4: *The figures display the plot concentration against the distance from the source for the time t=1 (red), t=3 (orange), t=5 (yellow), t=7 (green), t=10 (blue) and t=15 (black). The left figures works with the normal model while the right one works with the integrative model.*

From one dimension to two

So that the environmental model has a closer resemblance to real biological systems, the CBMP can model sources as a point or as a line. To model point-sources, we have made a simplification using the separation of the variable technique. This consists of replacing the x2 by the square distance of the ligand receptor from the source.

Multiple sources

The user can choose the number of sources and the kind of sources he wants as the image of the display of the diffusion window illustrate. Each source has its specific parameter input. The parameters are chosen in the ligand window. To model the diffusion with more than one source, we have assumed that the diffusion is independent of the number of sources. Therefore, we add the contribution of each source at each point at each time. This simplification allows the algorithm to compute much faster, even if it is not physically accurate. The contribution of one source at the centre of another source should be equals to zero but in our model the concentration of ligand only depends on the distance from the source and is thus non-zero.

The diffusion timescale

In writing the CBMP simulator, we have had to take some care to keep the diffusion and cell movement on the same timescale. Since one step moves with each time increment, if no adjustments were made, then if there are more

cells, each cell would move slower compared to the ligand. An adjustment has been made, so that the amount of time that is understood to have passed is determined not just be the number of cell steps taken, but also on the number of cells. Thus the diffusion and cell movement occur on the same timescale.

The proposed move

A cell's movement is determined by three aspects: the ligand, the randomness and the persistence. These are explained in turn.

In order to decide where to move, cells detect the amount of ligand at receptor locations on their cell membrane. In the model, the number of receptors is parameterised by the user, and these receptors are spaced equally around the circumference of the cell.

The cell is then most likely to go in the direction that - according to its receptors - has maximal concentration. Since it would not be biologically realistic for a cell to reliably detect infinitesimal differences in concentration around its surface, the user is allowed to decide the minimum ratio (and also a minimum absolute amount) of ligand that cells are able to detect. In the case of a stem cell, this threshold is also used to indicate whether the stem cell is in the niche, and this will determines its mitotic behaviour.

If the cell cannot detect the surrounding ligand, it's motion is determined by a combination of its last movement vector and random chance. this combination is weighted by user-specified persistence and randomness parameters.

Summary of cell movement

If the cell can detect concentration that is of a sufficient level, and of a sufficient ratio, then

Otherwise the cell is moving randomly. We have also added a randomness parameter that is specific to each type of cell, so the user can decide that for instance stem cells are purely deterministic while other cells have a purely random movement.

0.3.1 Ballistics: how cells bounce in the environment

Once a cell has a proposed movement, the CBMP detects whether the cell with collide with the walls, or any of the other cells, and resolves these collisions recursively.

Figure X: The algorithm for moving cells

The algorithm for detecting collisions begins by storing all of the cells' initial location, in case the simulation needs to be restored to that original state. as shown in figure X, the next step is to check whether the cell is going to touch any other cells, or to touch the border. If the cell has more than one collision in its path, then the cell is made to collide with whichever object it would hit first, and then the consequences of that collision are played out.

The first consequence of a collision is that the moving cell is placed adjacent to whatever it has collided with. If the cell has momentum to bounce off in a new direction, it does so, and it undergoes a new movement. In the case of a cell-wall collision, that is all. Alternatively, in the case of a collision between two cells, then both cells undergo further movement if they have sufficient momentum, and this occurs by restarting the algorithm with both cells.

Once all of the cells have settled on new locations and do not have sufficient speed to move, one final check is made for any overlaps. If there are overlaps, this can suggest that there was an error, such as in floating point arithmetic. In the case of an overlap (this occurs less than once per 10,000 iterations), the cells are restored to their original positions

It is instructive to describe in some more detail how it is detected that a cell is colliding with another cell, or that it is colliding with the wall.

Detecting cell-cell collisions

To understand how cell-cell collisions are identified, we can consider the most challenging case, where two cells lie on the path of a moving cell to its destination. As shown in Figure X, the moving cell, illustrated in black, is denoted m. Its initial position is m_0 and its proposed position is m_1. The two cells that lie in its path are denoted k_1 and k_2.

Let the cells be denoted as per Figure X.

Figure. X *Cell m would intersect with two other cells - k_1 and k_2 - on the way to its destination. $m_0$is its initial position, m_1 is the proposed position and $m_{0'}$ is its final position.*

The task of finding where cell m will first touch another cell is divided into four questions:

1. Which cells lie near enough to the line (of infinite length) that passes through m_0 and m_1?
2. Which cells overlap the finite region that m moves through?
3. Where might m touch those cells?
4. Of those positions, which is the nearest to m's starting location?

The first question is answered by computing the orthogonal projection of each stationary cell k onto m_0m_1, denoted h. If the distance kh is less than the sum of the radii of both cells

($h_1k_1 < r_m + r_k$), then the stationary cell lies sufficiently close to the path of cell m to collide with it. The length of h_1k_1 is given by:

where $y = mx + p$ is the equation of the line linking m_0 to m_1, with coefficients:

The second question is answered by ascertaining that the angles $?m_0m_1k$ and $?m_1m_0k$ are both acute, or that cell k is overlapping with m_1. Both of these angles must be acute if $\cos(?m_0m_1k)$ and $\cos(?m_1m_0k)$ are both positive, and these cosines can be easily computed using the cosine rule.

So far, cells that meet these criteria lie on the the path of cell m. However, one needs to decide which cell, of all those that remain, will be hit by cell m first. This is done by calculating where each of these cells would intersect with the moving circumference of cell m, and selecting the collision that is closest to m's starting point. These possible positions $m_{0'}$, are computed by solving the system:

Of these solutions for m_0, the one that minimizes $mm_{0'}$ is the true location at which m must collide with another cell.

As noted previously, once the moving cell is moved to $m_{0'}$, if cells m and k have sufficient speed, they will both continue to move. It is assumed that when a cell bounces, 0.9 percent of its momentum is conserved, and the other 0.1 contributes to energy loss of the collision.

The motion of cells after a collision is modelled like the collision of pool balls. Cell k, which was previously stationary, will set out in a direction parallel to m_0k with a movement vector v_k. Cell m will have a new movement vector v_1 that is perpendicular to v_k, and whose scalar product with v_0 is positive.

Figure X. *Illustration of the choice of angles when cell m bounces with cell k*

These movements do not add to v_0 and instead the momentum of each cell is scaled down by a factor of $g = 0.9$ so that the following equality holds:

The weight of each cell assumed to be proportional to its squared radius as the simulation takes place in a two-dimensional environment, although this code can easily be changed if it is desirable to make the mass proportional to the cubed radius instead.

Cell-border collisions

The other important kind of collision is between a cell and the border of the environment. To put the cell at a border, one needs to know which border the cell will touch first. To do this, the first step is to calculate the coefficients of the line D: y=mx + p that passes through the cell's starting location and its proposed location. It is not sufficient to simply extrapolate D to the nearest wall (Figure X). Instead, one must equate the line D to the lines x = x_E - r_m, x = x_W + r_m, y = y_N - r_m, y = y_S + r_m, where x_W, x_E, y_N and y_S are the respective x and y ordinates of each of the four borders.

Figure X. *The challenge of deciding which border m strikes. m_0 is the initial position, m_1 is the final position and m_2 is the proposed position. Note that m strikes the South wall although the line D that it is travelling on strikes the West wall.*

This gives the four places where the cell would intersect with the extension of each border. To know which border the cell is going to hit first, you only look at the two borders that the cell is moving towards, and disregard those that are behind. The border that the cell strikes first is the one that minimizes the distance m_1m_0, as shown in Figure X.

The cell can respond to the border in three different ways:

- If the border is reflecting, then the cell reflects as in optics, with a new movement vector that has an outgoing angle the same as the incoming angle. The new movement vector has length equal to 0.95 * m_1m_2, to account for inelasticity.
- If the border is absorbing, then the cell stays at the location m_1, and its speed reduces to zero.
- If the border is removing, then the cell disappears.

0.4 Examples

The platform has been designed model diverse biological scenarios with slight changes in parameters resulting in large changes in emergent behaviour. The commonality between these models is that a chemoattractant is hypothesised that biases cells' behaviour according to the concentration gradient established by a ligand source.

Three examples that demonstrate some of the features of the model are:

- The migration of macrophages to a wound.
- The behaviour of cells around a stem cell niche
- Cell culture

These examples are far from exhaustive. For one more example, similar applications could arise in modelling the mesenchymal cells present in embryogenesis, where it has been shown that cells are highly influenced by attractive and repulsive chemokines in their local environment.

Macrophage migration

The first model was designed to investigate how macrophages migrate to a laceration. This model was designed to copy an experimental setup in which lacerations are performed on a fruit fly, and the migration of macrophages is microscopically examined on the resultant wound. To demonstrate how the CBMP could be useful for modelling these data, three competing hypotheses were tested. In the first hypothesis, the cell migrate towards the wound deterministically. In the second hypothesis, the cells are not biased towards the wound at all, but they slow down when they get near to it. In the third case, the cells are biased towards but their movement towards the wound is stochastic, and they again have no particular inclination to slow down when they get there.

In all three simulations, the laceration is modelled as a straight-line at the location x=15 and secretes ligands (cytokines) at a constant rate for finite duration. The immune cells start in randomly assigned locations. The growth rate and death rate are set to zero, and the cells are 0.5 units in size. The borders were set to remove stray cells. The rest of parameters are available as the preset " migrate_random" and " migrate_non_random" models. For each model, ten simulations were performed, and their results were analysed as a batch.

In the first model, in which the the cells to be attracted to the wound deterministically, they moved to the wound quickly (in 300 timesteps) and

stayed there for the duration of the simulation. On gross visual inspection of this cell motion (Figure X) and the boxplot that describes their distance (Figure X), the cells move to the laceration more surely than would be seen in nature.

Figure X. *Timelapse of deterministic migration to a laceration at X = 15. Timesteps are every 500, starting at T = 1 (Top left). Cells quickly migrate towards the laceration and form a line over the source. T = 3000 has been left off as there was no difference from T = 2500*

Figure X. The horizontal location of cells over time when they migrate deterministically to the wound. Cells move to x=15, the location of the wound, rapidly and remain there for the duration of the simulation.

In the second model, the macrophages were initialised with high randomness and no bias, but a tendency to slow down when they neared the wound. This simulation provided some evidence macrophages do not arrive at wounds solely by being slowed by cytokines or exposed collagen at those wounds, as they did not differentially associate at the wound as would be expected *in vivo* (Figure X). After 3000 timesteps, the distribution of cells within the simulation was almost identical, with the average position not noticeably closer to the laceration than at the beginning of the simulation.

The third simulation provided the only realistic results (Figure X). Initialising cells to have a movement pattern made up of equal parts persistence, randomness and bias resulted in a slow attraction of cells towards the laceration. This is much more representative of the gross pattern of behaviour that would be expected in this scenario. Obviously, these models are very different and merely aimed to coarsely reproduce lifelike dynamics. There may be more insights from comparing more subtly different models to the data from automatic microscopy of these experiments.

Stem Cell Replication

It is of interest to assess how effectively the CBMP can model the dynamics of a stem cell niche because the niche hypothesis has been used to describe the dynamics of a wide range of environments, ranging from gonadal stem cells in the drosphila ovary to endothelial stem cells in mammals. One example that presents a potential expertimental collaboration for the authors is the case of haematopoetic stem cells in mouse models. Experimental evidence suggests that the Notch1 ligand is involved in the expansion of stem cells in a region that is referred to as the stem cell niche. In the CBMP model, we defined the stem cell niche as the region where stem cells are differentially likely to remain as stem cells when they replicate. The stem cell niche is defined in the simulation as any area in the environment where the local ligand concentration exceeds the concentration threshold. In practice, this results in a zone around any designated sources where stem cells grow in number. Once outside of this area, any new daughter cells are more likely be progenitors.

To simulate the stem cell niche, we designated stem cells as type 1, with a low growth rate of 0.1, a randomness of 0.7 with a persistence of 0.2 and a concentration response of 1, initialised to the left edge of the field. Type 2 cells were designated the progenitors, with a high growth rate of 0.2, and randomness of 1.0 but no attraction to the stem cell niche. The remainder of parameters can be seen in the *niche_sim* preset in run_sim.jl. A constant ligand source can be placed as a line or multiple points along the line X = 0 to generate the niche; in our simulation, the five units of space on the left hand side as the niche, and the rest was designated as non-niche. The borders were initialised as the left hand border being absorbing and the other 3 as removing. This resulted in the population of stem cells stabilising in the niche whilst a population of progenitors was quickly established. This population began to drift out of the simulation, creating room for new progenitor cells at the stem cell niche border. After about 2500 iterations the rate of growth of progenitor cells in the field of the simulation decreased, and would eventually approach equilibrium (Figure X).

A sense of how this growth happens across space is given in Figure X. Initially, the stem cells begin at the leftmost wall of the simulation, and they quickly fill out the niche region. Conversely, the progenitor cells far outnumber stem cells in the non-niche region, although each type of cell spills over to a small degree to the other side of the niche boundary.

Figure X.*Stem Cell Niche Simulation. Parameters used: Stem Cells(Blue):*

Growth Rate: 0.1, Threshold: 0.000793, Persistence: 0.2, Randomness 0.7. Progenitor Cells (Green) Growth Rate: 0.25, Threshold 0.000793, Persistence: 0.2, Randomness: 1.0. The stem cell population reaches equilibrium around 2000 timesteps and the progenitor cell population does not yet reach stable equilibrium within 5000 timesteps.

Figure X. *Frequency of cells by distance to X = 0. The simulation was performed five times, of which all cell counts are a total of these runs. The region 0 <= x <= 5 is designated the stem cell niche. The border between the niche and non-niche area is indicated by the black vertical line. The stem cells, which are initialised to the leftmost wall, predominate within the stem cell niche for the duration of the simulation. The progenitor cells predominate in the non-niche area, and their proliferation slows, but does not completely stabilise in the second half of the simulation.*

Cell culture

Initialising with a cell speed of 0 allows for the simulation of a cell culture with immobile cells, such as the simulation of tumour growth or bacteria culturing. Whilst additions would need to be made to the platform to create simulations that could have scientific implications, simple simulations can be performed that can reveal some interesting dynamics. In a simulation of two identical types of cells competing against each other - with the only difference being death rate - it shows how many of one population have to die

before the other is dominant enough to outcompete them for space within the model.

Performance

One significant factor that will determine the usefulness of the CBMP, especially for the purpose of parameter inference, is its performance. On an AMD 1.9GHz single processor desktop computer, the CBMP is able to perform 700 iterations per second with a model of sustained ligand secretion for twenty cells with eight receptors. This is highly dependent on which model is used for diffusion, and how many receptors each cell has.

0.5 Further Development of the Platform

For a researcher who can program in Julia, modifications can open up the possibility of more diverse models. Alteration to the program's source code have been made possible by placing the software on Github, where any user can branch it or submit a pull request. The code has been commented and moreover, it is logically organised into modules that perform specific roles. If a user wants to model cells that decide their movement in a different fashion they can alter the move file; if a user wants to give the cells more complex characteristics, they can alter the Cell type in cell_type.jl, and so on. The table below can be used to ascertain which file or files are needed for any desired alterations (Table 1).

Function	**File(s)**
Cell Properties	cell_type.jl, gui_type.jl
Diffusion	diffusion.jl, gui_diffusion.jl
Growth, Death and Division	birth_and_death.jl
Movement	angle.jl, move.jl
Command Line	command_line.jl
Process Flow	entry.jl, simulator.jl

Table 1. Platform files and their relevant real world functions.

In " move.jl", the function " tentative_move" would plausibly be useful to edit, and is an instructive example. Its role is to suggest a movement for the cell, that is later modified by any cell-wall or cell-cell collisions. However this function may be edited, cells will collide as per usual, and will not overlap or exit the bounds of the simulation. So users can freely edit the cells' angle or speed on lines 11 and 16 respectively. Currently, the angle is generated by calling *angle_from_both* but instead, one can assign an angle using another expression like *rand()*2pi*, which samples the angle from a uniform random

distribution. The cell's speed is sampled from a Chi-squared distribution (the inverse of this cumulative distribution function is given at the start of line 16) but it can easily be changed to a constant speed by shortening this line of code to $moving\_cell.speed = categories[moving\_cell.cell\_type].avg\_speed$, which will deterministically assign the same speed to all cells of each type. Or to a speed that is proportional to the maximum concentration e.g. $max(concentrations)categories[moving\_cell.cell\_type].avg\_speed$. These are some simple suggestions but much more complex functions are feasible.

Figure X: The tentative_move function

While it has been attempted to make the platform as comprehensive as possible, a few powerful developments could not be included in the platform, some of which are discussed below.

First, extensions to the simple implementations of the growth and death cellular functions to bring their sophistication in line with movement and division would increase the scope of the simulations the platform is able to perform. At their most basic, both these functions should be linked to local ligand concentration providing growth and survival signals.

This would allow the platform to perform simulations of necrosis and potentially preliminary models of migrating mesenchymal cells in embryogenesis, where it has been shown the presence of survival signals are crucial to the migrational pattern of embryonic cells. Implementing a distinction between epithelial and mesenchymal cells would work well with the aforementioned development, and could all be involved in modelling tumour growth and the role of angiogenesis and epithelial to mesenchymal transition (EMT) in the progression of the disease state.

Second, a logical extension of the platform would be to enable it to distinguish and keep track of different ligand types throughout the environment. This would provide a chance for competing signals to influence cellular decisions, which could then be more exquisitely responsive to their environment. Another interesting feature would be to model how cells respond to secretion of ligand that starts at different times. For example, it would be interesting to see how reaction of cells that have been migrating towards a laceration that then encounter a sharp, strong pulse of attractive chemokine from elsewhere. Given that the simulation already allows users to specify when each source should cease to produce ligand, it would be a small extension to make the diffusion start at a time that is also specified by the user, rather than at the beginning of the simulation.

Third, the current method of all cells growing at a type defined rate,

whilst acceptable, is a huge abstraction from the reality of cellular growth. Although this could complicate the model substantially, implementing some simple version of the cell cycle in response to diverse environmental stimuli would add a lot of functionality to the platform. This would allow cells within the same type to be in proliferating and nonproliferating phenotypes based on their local environment, potentially allowing additional complexity to be added to the stem cell niche simulation.

Fourth, it would be beneficial to build a suite to use the simulations for parameter inference. The first step in inferring model parameters from experimental data would be to set the size of the cells, the field, the environment and the timestep appropriately. If these parameters were calibrated to an experimental setup, this would allow more careful comparison of results produced by the two. The next steps would be more complex and interesting - using Bayesian methods, the movement models for one or more cells could be calibrated so that the observed behaviour is more likely. This inference would have to be restricted to one or two parameters, such as the persistence and bias. Either by manual exploration or by formal Bayesian inference, the diffusion coefficients could also be set to more reasonable values. This could give more biological meaning to the relative amounts of ligand predicted in a stem cell niche or at an injury.

Fifth, time could be made to advance with a Poisson distribution instead of a discrete timescale (e.g. as in Gillespie's Exact Algorithm). Currently, the platform uses discrete time steps to iterate through the cells in the simulation whilst randomly choosing one cell at a time to update. Although this solution works, a more elegant solution would be to implement a Poisson distribution to describe the time at which each cell moves, with a ? drawn from the number of cells in the simulation at the time. This would allow the diffusion equation to update in a more natural manner and improve the overall realism of the simulation.

Sixth, it would be beneficial to expand the model to allow ligand to be secreted by moving cells. This has immediate biological relevance in the case of the haematopoietic stem cell niche, where experimental evidence suggests that osteoblastic cells are involved with secreting Notch ligand jagged 1, which is involved with maintaining the stem cell niche. [5]

If user-developers are able to contribute to these facets of the program or any others that are useful for their own applications, then they are encouraged to submit a pull request to *https://github.com/RyanCarey/abm-platform*, and these will be promptly actioned. We hope that you enjoy using the software and are available to be contacted regarding it using GitHub or via email, at ryan.carey14@imperial.ac.uk, lewis.kindeleit14@imperial.ac.uk, antoine.messager14@imperial.ac.uk.

0.6 An alternative agent-based model

We also attempted to implement a semi-spatial model based on the model described by Roeder to simulate the development of leukaemia and the effect of the tyrosine kinase inhibitor imatinib on normal and disease state cells. The concepts behind this model broadly match our own, with cells moving between compartments stochastically and transitioning from stem cells to progenitors depending upon their most recent surroundings. This results in slow expansion until the first progenitors appeared within the system (after roughly 24 hours). After this, the speed of the expansion increased exponentially. After 500 iterations (1 hour time steps), the speed of the simulation had slowed dramatically due to size.

However, despite the lack of results from this alternative model, it revealed a few properties that could be adapted and incorporated into our model. Amongst these, an attempt at modelling the cell cycle in a more accurate manner, with proliferating and nonproliferating same type cells (currently not possible in our platform), represents an exciting prospect.

With one of the intended benefits of the Julia programming language supposedly being the ease with which parallelisation can be applied to programs encouraged us to attempt to adapt the model for comparison to our own. However, with Julia being in its embryonic state, its parallelisation ability is not quite what we'd hoped, and as such we think revisiting this problem in 6 months to a year will hopefully allow the Julia developers to fully implement parallelisation functionality in their language, therefore reducing the time it takes for these simulations to run.

Bibliography

[1] Roeder, Ingo, et al. "Dynamic modeling of imatinib-treated chronic myeloid leukemia: functional insights and clinical implications." *Nature medicine* 12.10 (2006): 1181-1184.