Patient-calibrated agent-based modelling of ductal carcinoma in situ (DCIS) II: Supplementary Material

Paul Macklin [1,2,3], Mary E. Edgerton [4], Alastair Thompson [4,5], Vittorio Cristini [3,6]

Abstract

We provide here the following supplementary materials for Macklin et al. (2011):

- Numerical techniques, including an acceleration for cell-cell interaction testing
- Full specification of an open, XML-based data format for multicell agent models, with benchmark datasets and open source processing code
- Additional parameter studies to: further support the modelling assumptions and choice of parameter values; assess the robustness of the model; and to further illuminate the model-predicted biological behaviour of DCIS.
- Full animations of the key simulation results.

Key words: agent-based modelling, individual based modelling, tumour simulation, nonhomogeneous Poisson processes, cell phenotype, patient-specific calibration, tumour growth, cell necrosis, cell mechanics, cell calcification
1991 MSC: 65C20, 92B05, 92C05

Email address: macklin@maths.dundee.ac.uk (Paul Macklin).
URL: http://www.maths.dundee.ac.uk/macklin (Paul Macklin).
[1] Corresponding author
[2] Division of Mathematics, University of Dundee, Dundee, Scotland, UK
[3] Formerly of: School of Biomedical Informatics, University of Texas Health Science Center, Houston, TX, USA
[4] M.D. Anderson Cancer Center, Houston, TX, USA
[5] Department of Surgery and Molecular Oncology, University of Dundee, Dundee, Scotland, UK
[6] Departments of Pathology and Chemical Engineering, University of New Mexico, Albuquerque, NM, USA

1 Numerical methods

We implement the model using object-oriented ANSI C++, where each agent is an instance of a `Cell` class. Each cell object is endowed with an instance of a `Cell_State` class, which contains the cell phenotypic parameters ($\overline{\alpha}_P$, α_A, τ_P, etc.), volumes (V_S, V_N, V), radii (R_N, R), maximum interaction distances ($R_{\text{cca}}, R_{\text{cba}}$, recorded as multiples of R), position $\mathbf{x}$, and velocity $\mathbf{v}$. We order the cells with a doubly-linked list structure: each agent is given the memory addresses of the previous and next cells. This allows us to easily delete apoptosed cells and insert new daughter cells following proliferation events. Wrapping the phenotypic properties in a `Cell_State` class makes it easy to pass heritable properties from parent to daughter cells in a generalised manner.

We discretise microenvironmental field variables (e.g, oxygen σ) on an independent Cartesian mesh with uniform spacing $\Delta x = \Delta y = 0.1L$, where L is the oxygen diffusion length scale. We represent the BM morphology with an auxilliary level set function (see Part I), and we use an auxilliary data structure to reduce the computational cost of cell-cell interaction testing and evaluation. (See Section 1.1.)

We now describe the program flow of this numerical implementation. In the discussion below, $N(t)$ denotes the total number of cells at time t.

(1) **Initialisation Routines:**

(a) **Parse simulation settings file:** Parse an XML file containing all information on the simulation domain, cell types and initial arrangement, phenotypic parameters, data output times, etc. Set global variables such as the current simulation time t, the current (dynamic) time step size (Δt, initially zero), etc.

(b) **Initialise cells:** Create new cell objects and place them within the computational domain as indicated in the prior step. For each cell, set its phenotypic parameters, and randomly select its state $\mathcal{S}$ with probabilities specified in the settings (e.g., to match immunohistochemistry). Lastly, set its progression within its state randomly (with uniform distribution), and update its volume, etc. accordingly.

(c) **Initialise BM morphology:** Create a level set function d on a mesh (with $\Delta x = \Delta y = 1\ \mu$m) to represent the basement membrane morphology as specified in the settings. Discretise the normal vector $\mathbf{n}$ on the same mesh, computing the gradient $\mathbf{n} = \nabla d$ either analytically or by the gradient discretisations in Macklin and Lowengrub (2006).

(d) **Initialise microenvironmental variables:** Introduce a regular Car-

tesian mesh and discretise the microenvironmental field variables on that mesh. For oxygen, initialise $\sigma \equiv \sigma_B$ and solve to steady state.

(2) **Main program loop:** While $t < t_{\max}$:

(a) **Update microenvironmental variables:** Each microenvironmental variable u must be updated from $u(\mathbf{x}, t - \Delta t)$ to $u(\mathbf{x}, t)$. Solve the various microenvironmental PDEs using standard finite difference schemes (Macklin and Lowengrub, 2005, 2008; Macklin et al., 2009). Compute volume-weighted, upscaled uptake and other reaction rates as necessary. Use independent time step sizes for each variable according to standard CFL stability criteria, until each variable has been updated to time t.

(b) **Update cell-cell interactions:** Update the data structure for cell-cell interaction testing and evaluation. See Section 1.1.

(c) **Update the cells:** For each cell:

(i) *Progress the current cell state:* Update the cell with the appropriate submodel for its previous state $\mathcal{S}(t - \Delta t)$ until reaching the current simulation time t. Any field variable values that are necessary for computing the cell phenotypic transition probabilities (e.g., oxygen) are interpolated at the cell's position $\mathbf{x}$.

(ii) *Choose the next cell state:* If the cell was not quiescent at the previous time step, then set $\mathcal{S}(t) = \mathcal{S}(t - \Delta t)$ unless it has been altered in (2c.i). If $\mathcal{S}(t - \Delta t) = \mathcal{Q}$, then choose $\mathcal{S}(t)$ by evaluating the (exponentially-distributed) random probabilities as described in Section 1.2.

(iii) *Set the cell velocity:* Set $\mathbf{v}$ according to Eq. 1 in Part II. Use the optimisation in Section 1.1 to truncate the summation to a smaller set of interacting cells.

(d) **Set Δt:** Dynamically choose the simulation time step size via:

$$\Delta t = \frac{\epsilon}{\max\left\{|\mathbf{v}_i|\right\}_{i=1}^{N(t)}}. \tag{1}$$

Here, ϵ is the desired accuracy in the cell position; we use $\epsilon = 1\ \mu$m. Note that Δt is independent of the interaction and microenvironmental mesh sizes, as the agents themselves are lattice-free.

(e) **Update cell positions:** For each cell, update the position using:

$$\mathbf{x}(t) = \mathbf{x}(t - \Delta t) + \mathbf{v}\Delta t \tag{2}$$

While we use the forward Euler difference for simplicity, improved methods (e.g., Runge-Kutta (Gottlieb and Shu, 1997; Gottlieb et al., 2001)) are straightforward to implement.

(f) **Update the simulation time:** Increment t by Δt.

Each step in the main program loop requires at most iterating through the list of the cell agents. If interaction testing can be made similarly efficient, then the overall computational effort is linear in the number of cells. To attain this, we use an auxiliary cell-cell interaction testing data structure that can be constructed linearly in the number of cells, and allows a truncation of the summation in each cell's velocity in Eq. 1 in Part II, thus rendering the overall algorithm linear in the number of cells. See Section 1.1.

1.1 Accelerated cell-cell interaction testing

Let $\{k\}_{k=1}^{N(t)} = \{1, 2, 3, \cdots, N(t)\}$ be a list of all simulated cells in the computational domain $\mathcal{D}$ at time t. We construct a data structure that lists all possible cell-cell interactions at any point in the computational domain $\mathcal{D}$. We first introduce a uniform Cartesian mesh $\mathbb{M} = \{\mathbf{x}_{i,j}\} = \{(x_i, y_j)\}$ (the interaction mesh) with spacing $\Delta x = \Delta y = 1\,\mu$m. At each $\mathbf{x}_{i,j} \in \mathbb{M}$, let $\{k_m^{i,j}\}_{m=1}^{N_{i,j}(t)}$ be the list of (potentially) interacting cells at $\mathbf{x}_{i,j}$ at time t.

Step 1: Compute the maximum cell-cell interaction distance by

$$R_{\text{cca,max}} = \max\left\{R_{\text{cca}}^k\right\}_{k=1}^{N(t)}. \tag{3}$$

Step 2: For each $\mathbf{x}_{i,j} \in \mathbb{M}$, set $k_0^{i,j} = 0$ and $N_{i,j}(t) = 0$. Because no cell has index 0, this denotes the case of 0 possible interactions at $\mathbf{x}_{i,j}$.

Step 3: For each cell k and for each $\mathbf{x}_{i,j}$ satisfying:

$$|\mathbf{x}_k - \mathbf{x}_{i,j}| \leq R_{\text{cca,max}} + R_{\text{cca}}^k, \tag{4}$$

set:

$$k_{N_{i,j}(t)+1}^{i,j} = k \qquad \text{(append the cell to the list at } \mathbf{x}_{i,j}) \tag{5}$$

$$N_{i,j}(t) = N_{i,j}(t) + 1. \qquad \text{(increment the total at } \mathbf{x}_{i,j}) \tag{6}$$

At each $\mathbf{x}_{i,j}$, the result is a list of all cells that can interact with a cell centred at $\mathbf{x}_{i,j}$. In C++, we implement this scheme as a singly-linked list of cell memory addresses at each $\mathbf{x}_{i,j} \in \mathbb{M}$; a NULL pointer indicates either an empty list ($N_{i,j}(t) = 0$) or the end of the list (list member $N_{i,j}(t)$ points to NULL).

For fixed ℓ and $\mathbf{x} \in \mathcal{D}$, we use this list to evaluate expressions of the form

$$\text{for all cells } k \in \{k\}_{k=1}^{N(t)} \setminus \{\ell\} \text{ compute } f\left(\text{cell}_k, \text{cell}_\ell\right)(\mathbf{x}), \tag{7}$$

such as

$$\sum_{\substack{k=1 \\ k \neq \ell}}^{N(t)} f\left(\mathbf{x}_k, \mathbf{x}_\ell\right). \tag{8}$$

Let $\mathbf{x}_{i,j}$ denote the closest interaction mesh point to $\mathbf{x}_\ell$ (the position of cell ℓ). Then we evaluate Eq. 7 by truncating it to the members of the list at $\mathbf{x}_{i,j}$:

$$\text{for all cells } k \in \left\{k_m^{i,j}\right\}_{m=1}^{N_{i,j}(t)} \setminus \{\ell\} \text{ compute } f\left(\text{cell}_k, \text{cell}_\ell\right)(\mathbf{x}). \tag{9}$$

In the example above, we truncate the summation to

$$\sum_{\substack{m=1 \\ k_m^{i,j} \neq \ell}}^{N_{i,j}(t)} f\left(\mathbf{x}_{k_m^{i,j}}, \mathbf{x}_\ell\right). \tag{10}$$

Setting the interaction mesh spacing to 1 μm sufficiently resolves cells (generally 10 to 20 μm in diameter), which reduces the impact of the nearest-neighbour approximation above; in practice, a larger spacing may suffice.

Because our interaction potentials have compact support, there is a fixed upper bound M_1 for the number of operations required to update the interaction lists for each cell; the operation is linear in the number of cells. Similarly, each interaction mesh point $\mathbf{x}_{i,j}$ has a fixed maximum number of list elements M_2, and so evaluating Eq. 9 for all cells $1 \leq \ell \leq N(t)$ is linear in the number of cells. Contrast this with Eq. 7, which for each cell ℓ scales with $N(t)$; iterating this non-truncated form over all cells thus requires $N(t)^2$ computational effort.

1.2 Evaluating probabilities

Suppose we have (assumed independent) random variables $X_1, \ldots, X_n$ with cumulative probability distributions $F_i(t), 1 \leq i \leq n$. We test for the occurence of one of the events X_i in the interval $[t, t + \Delta t]$ by:

(1) Choose $r \in [0, 1]$ with uniform random distribution. Numerically, we use the `ran2` pseudorandom generator procedure from Press et al. (1992); the Mersenne twister pseudorandom generator is also commonly used.
(2) Set $p_i = F_i(t + \Delta t) - F_i(t)$ for $1 \leq i \leq n$. Define $p_0 = 0$. Set $a = b = 0$.
(3) For $1 \leq i \leq n$:

(a) Set $a = b$ and $b = a + p_i$. (i.e., $a = \sum_{j=0}^{i-1} p_j$ and $b = \sum_{j=0}^{i} p_j$.)
(b) If $a \leq r \leq b$, then say that event X_i has occurred in $[t, t + \Delta t]$, and end the procedure. Otherwise proceed.

(4) If we exit the loop, none of the X_i events has ocurred in $[t, t + \Delta t]$.

We note that in principle, this procedure can break down for large Δt, as $\sum_{j=1}^{n} p_j$ can exceed 1. In practice, we only evaluate probabilities on short time intervals, thus the p_i are small, and this is not an issue in simulations.

2 `MultiCellXML`: An open multicell simulation data format

We have developed a human-readable, XML-based data format for agent-based, multicell simulations (`MultiCellXML`), which includes the random seed state, global variables, information on (and filenames of) microenvironmental field variables, and a list of each cell object and its current state. This structure allows us to easily parse the data (using standardised XML parsers, such as Expat (Clark, 2007), xmlParser (Berghen, 2009), and TinyXML (Thomason et al., 2010)) for use in data visualisation and post-processing. The list of cells in the XML file is very similar to the object-oriented `Cell` data structure in the simulator, making the format well-suited to resuming simulations from saved states. Modifying simulation parameters during a simulation can be readily achieved with simple plaintext search/replace operations in the XML files. We note that the `MultiCellXML` format is under active development; readers should reference the project website[7] for the very latest standards, documentation, and software utilities.

We begin with XML header information (`<?xml>`) for XML 1.0 standards compliance, followed by a "root" `<data_set>` tag. In the `<data_source>` section, we include information on the originating simulation software (`<simulator>`), the user (`<user>`), and any publication information that may assist the recipient of a data file in (1) locating the original source of the data, and (2) proper academic citation (`<reference>`). See Fig. 1. Future `MultiCellXML` versions may include reference and citation information for the simulation software.

Following the `<data_source>` section, the `<globals>` section includes information such as the current simulation time and the random seed state–this is important for resuming saved simulation states without affecting the pseudo-random number generator. Where possible, we include information on physical units as XML tag attributes. We note that because this was initially a format developed for internal use, we have not been entirely consistent in our conventions–improvements are planned in future drafts of the file specifica-

[7] `http://multicellxml.sourceforge.net`

```
<?xml version="1.0" encoding="UTF-8" ?>
<data_set MultiCellXML_version="1.0">
   <data_source>
      <filename>data/output00000117.xml</filename>
      <created>29 July 2010</created>
      <simulator>
         <program_name>DCIS_2D</program_name>
         <program_version>1.38</program_version>
         <compiled>1 July 2010</compiled>
         <author>Paul Macklin</author>
         <contact>macklin@maths.dundee.ac.uk</contact>
         <URL>http://www.maths.dundee.ac.uk/macklin/</URL>
      </simulator>
      <user>
         <name>Paul Macklin</name>
         <contact>macklin@maths.dundee.ac.uk</contact>
      </user>
      <reference>
         <citation>Macklin et al. J. Theor. Biol. (2011) (in review)</citation>
         <URL>http://multicellxml.sourceforge.net</URL>
         <note>User notes may go here.</note>
      </reference>
   </data_source>
   <globals>
      <time units="minutes">7020</time>
      <next_output_time units="minutes">7020</next_output_time>
      <frame_number>117</frame_number>
      <random_seed_state>769969952</random_seed_state>
      <Domain_width_in_microns>1000</Domain_width_in_microns>
      <Domain_height_in_microns>340</Domain_height_in_microns>
   </globals>
   ...
```

Fig. 1. **Start of a `MultiCellXML` file:** The first tag is for XML 1.0 standards compliance. The `<data_source>` section indicates the source of the data, including the originating program, information on the user, and requested reference for citation (if any). The `<globals>` section gives information on program globals, including (in particular) the current simulation time and the random seed state.

tion. For dimensionless quantities, the scale should ideally be stated (e.g., as an additional XML attribute):

```
<local_oxygen units="dimensionless" scale="far-field">0.84</local_oxygen>
```

In future drafts, we may include a new `<scales>` section to facilitate this.

The file format continues with a list structure of all the cells (`<cell_list>`), with essentially all internal cell variables (i.e., member data of the `Cell` class) listed clearly. We give each `<cell>` both a numeric type (`<cell_type_code>`) to assist comparing and classifying cells in software, and a human-readable type (`<cell_type_text>`) to assist data recipients with interpretting the data. See Fig. 2. Note that we have included "type" attributes to indicate boolean

variables, rather than units. In future file version drafts, we may include both "type" and "units" attributes to all `<cell>` data fields. However, we can generally assume that the presence of units indicates a non-boolean variable, and the precence of a boolean type obviates "units."

Due to historical reasons stemming from code development, each `<cell>` is split into `<cell_properties>` and `<cell_state>` sections; future versions of the data standard will likely merge these into a single `<cell_state>` section, because many cell properties tend to change over time due to the cells' exposure to differing microenvironments.

After all data files have been listed, we include a `<global_variables>` section with a list of all saved field variables and file formation information. See Fig. 3. Note that we have included the full path of each data file; often all the files (including the XML file) are saved in the same directory, so postprocessing may need to strip part of the path by comparison to the `<filename>` filed in the `<data_source>` section. Due to the large size of 2-D and 3-D double-precision data arrays, we opted for a binary data format. For increased compatibility, we choose the MATLAB .MAT (Level 4) file format, which is relatively simple to implement directly from the published file format standard (Mathworks, 2010), and is simple to read and write with common open source software (e.g., Octave) as well as MATLAB. In the source code to follow, we include C++ code to read and write these MATLAB data.

Lastly, note that a primary goal of our specification is to make the format as human-readable as possible, rendering the format (partially) "self-documenting". This will make it simpler to interpret archived data long after the originating software is out of use, thus eliminating the need for reverse engineering–hence our choice of human-readable, non-binary data. While this results in much larger files, we regard data compression as a separate software problem from the specification of content. Compression can readily be applied to the data files after creation with widespread software, such as gzip.

2.1 Benchmark datasets

To demonstrate our open data format and serve as benchmark datasets, we are releasing [8] the full datasets for simulation times 0, 15, 30, and 45 days from the "baseline" simulation; see Section 7 in Part II. Included files:

(1) `output00.zip`: contains all data from 0 days:
   (a) `output000000000.xml`: `MultiCellXML` data

[8] No license applies here, aside from standard scientific citation ethics. Please reference Part II (Macklin et al., 2011).

```
...
<cell_list>
   <cell>
      <cell_properties>
         <cell_type_code>0</cell_type_code>
         <cell_type_text>DCIS cell</cell_type_text>
         <radius units="microns">9.95299956207</radius>
         <nuclear_radius units="microns">5.295</nuclear_radius>
         <volume units="cubic microns">4130.00487398</volume>
         <mature_volume units="cubic microns">4130.00487398</mature_volume>
         <solid_volume units="cubic microns">413.000487398</solid_volume>
         <cell_adhesion_1_level units="dimensionless">1</cell_adhesion_1_level>
         <cell_adhesion_2_level units="dimensionless">0</cell_adhesion_2_level>
         <matrix_adhesion_level units="dimensionless">1</matrix_adhesion_level>
         <calcite_level units="dimensionless">0</calcite_level>
         <mean_cell_cycle_time units="minutes">1080</mean_cell_cycle_time>
         <mean_G1_time units="minutes">540</mean_G1_time>
         <mean_time_to_apoptosis units="minutes">47196.6</mean_time_to_apoptosis>
         <mean_time_to_mitosis units="minutes">115.27</mean_time_to_mitosis>
         <cell_adhesion_exponent units="dimensionless">1</cell_adhesion_exponent>
         <BM_adhesion_exponent units="dimensionless">1</BM_adhesion_exponent>
         <calcite_adhesion_exponent units="dimensionless">1</calcite_adhesion_exponent>
         <cell_repulsion_exponent units="dimensionless">1</cell_repulsion_exponent>
         <BM_repulsion_exponent units="dimensionless">1</BM_repulsion_exponent>
         <cell_adhesion_max_distance units="x radius">1.214</cell_adhesion_max_distance>
         <BM_adhesion_max_distance units="x radius">1.214</BM_adhesion_max_distance>
         <calcite_adhesion_max_distance units="x radius">1.214</calcite_adhesion_max_distance>
      </cell_properties>
      <cell_state>
         <is_cycling type="boolean">true</is_cycling>
         <is_quiescent type="boolean">false</is_quiescent>
         <is_apoptosing type="boolean">false</is_apoptosing>
         <is_anoxic type="boolean">false</is_anoxic>
         <is_necrosing type="boolean">false</is_necrosing>
         <is_debris type="boolean">false</is_debris>
         <apoptosis_time units="minutes">360.85</apoptosis_time>
         <necrosis_time units="minutes">0</necrosis_time>
         <cell_cycle_time units="minutes">0</cell_cycle_time>
         <Position units="microns">(86.5665990925,53.5000597051,0)</Position>
         <Velocity units="microns/minute">(-0.108426856979,0.213070920989,0)</Velocity>
      </cell_state>
   </cell>
   <cell>
   ...
   </cell>
...
</cell_list>
...
```

Fig. 2. **Main content of a `MultiCellXML` file:** Within the `<cell_list>` section, we save each individual cell agent's data within a set of `<cell></cell>` tags, including `<cell_properties>` and the `<cell_state>`. In future revisions, these fields may be merged due to the fact that cell properties change in time. *Note:* These fields have been minimised from the actual published datasets to simplify the presentation.

```
   ...
   <global_variables>
      <variable>
         <name>oxygen</name>
         <format version="Level 4">MATLAB</format>
         <filename>data/oxygen_00000117.mat</filename>
      </variable>
      <variable>
         <name>Duct_Wall_Level_Set</name>
         <format version="Level 4">MATLAB</format>
         <filename>data/level_set.mat</filename>
      </variable>
   </global_variables>
</data_set>
```

Fig. 3. **End of a `MultiCellXML` file:** After the `cell_list` section, the `global_variables` section gives a list of all associated external field data (here saved in MATLAB format).

(b) `oxygen_000000000.mat`: (dimensionless) oxygen data
(c) `levelset.mat`: basement membrane morophology

(2) `output15.zip`: contains all data from 15 days:
(a) `output000000360.xml`: `MultiCellXML` data
(b) `oxygen_000000360.mat`: (dimensionless) oxygen data
(c) `levelset.mat`: basement membrane morophology

(3) `output30.zip`: contains all data from 30 days:
(a) `output000000720.xml`: `MultiCellXML` data
(b) `oxygen_000000720.mat`: (dimensionless) oxygen data
(c) `levelset.mat`: basement membrane morophology

(4) `output45.zip`: contains all data from 0 days:
(a) `output0000001080.xml`: `MultiCellXML` data
(b) `oxygen_000001080.mat`: (dimensionless) oxygen data
(c) `levelset.mat`: basement membrane morophology

The most up-to-date version of these datasets will be maintained at the `MultiCellXML` project website.

2.2 Sample post-processing

Because the cell data are saved in a standardised XML configuration, post-processing is a combination of XML parsing and visualisation (to interpret the data). In our implementation, we choose the relatively compact TinyXML library (Thomason et al., 2010) with customised interfaces to simplify the process; this allows us to distribute code as fully self-contained, without need for installation of external libraries. We use the open source EasyBMP library (Macklin, 2005–present) for image operations. Source code is provided at the `MultiCellXML` project website; this software has been tested in Windows (with

the mingw implementation of the g++ compiler), Linux, and OSX 10.6 in 32-bit and 64-bit environments.

In our post-processing code, we do the following:

(1) Parse the `<cell_list>` XML data:
   (a) Create a singly-linked list of a simplified `cell` objects (read from the `<cell_list>` section), consisting primarily of cell location, radius, degree of calcification, and phenotypic state.
   (b) Plot the cells in a temporary BMP image (in the program memory space–this is not actually saved as a file) for use in further geometric processing.
   (c) Plot a virtual "buffer" around all the cells to help fill in holes in the viable rim–this is essential for later cell density calculations, as well as for identifying the entire viable rim.
(2) Fill remaining holes in the viable rim to ensure its correct identification.
(3) Crop the virtual images at the leading and trailing edges to eliminate the "edge effects" and best match the patient data images. Remove the corresponding `cell` objects from the linked list.
(4) Count the total, proliferating, and apoptotic cells within the viable rim. Use these to calculate the proliferative index (PI) and apoptotic index (AI).
(5) Count the number of coloured pixels of the viable rim in the temporary image, and use this to calculate the area of the viable rim. (1 pixel is 1 μm^2.)
(6) Use the known length of the cropped domain to calculate the mean viable rim thickness.
(7) Use the viable rim area and total cell count in the cropped areas to calculate the cell density.
(8) Calculate the position of the farthest tumour cell (uncropped). Do the same for calcified cells.
(9) Use the known total numbers of (uncropped) viable tumour cells to find the 95% position (i.e., x_{V95} such that 95% of the tumour is in the region $\{(x, y) : x \leq x_{\mathrm{V95}}\}$). Do the same for calcified cells.
(10) Append these data to `PPdata.txt`.
(11) Once done looping over all specified files, write `legend.txt` to document the structure of `PPdata.txt`

To use this code, compile it according to your compiler instructtions. (`g++` users may use the supplied makefile. Windows 32-bit binaries are included in our distributions. Please note that the compiler optimisations are oriented towards 32-bit Core2 Intel processors and above.) To apply the code to the supplied data for time 30 days, type:

```
> ./PostProcessing output00000720.xml
```

To apply the code to all the supplied data, type:

```
> ./PostProcessing output*.xml
```

This software is licensed under the GPL 3.0. It is packaged with TinyXML (zlib/libpng license – see Thomason et al. (2010)) and EasyBMP (Modified BSD license – see Macklin (2005–present)). We request that users cite this paper and the project website in their "methods" section when publishing results that make substantial use of the code or derivative works.

2.3 Sample visualisation

Visualisation is performed similarly, but requires much less processing. We plot each cell as a circle with correct colour, overlay a solid nucleus, and draw a dark border. We draw the basement membrane based upon the zero contour of the level set function. Afterwards, we overlay a scale bar, label the time, and save the image.

We regard image creation and image compression as separate software problems. We use the BMP format because it is simple, universally understood, can be implemented without need for complex external libraries, and does not introduce visual artifacts to the data (in contrast to formats with lossy compression, such as JPEG). The lack of visual artifacts is also helpful for pixel-based image processing operations by other software. Users can readily compress the images using standard tools (e.g., ImageMagick and GIMP), or combine the BMP frames into an (uncompressed) AVI animation using tools such as EasyBMPtoAVI (Macklin, 2006–present).

Source code is provided at the `MultiCellXML` project website; this software has been tested in Windows (with the mingw implementation of the g++ compiler), Linux, and OSX 10.6 in 32-bit and 64-bit environments. To use this code, compile it according to your compiler instructtions. (`g++` users may use the supplied makefile. Windows 32-bit binaries are included in our distributions. Please note that the compiler optimisations are oriented towards 32-bit Core2 Intel processors and above.) To apply the code to the supplied data for time 30 days, type:

```
> ./visualize_DCIS_2D output00000720.xml
```

This software is licensed under the GPL 3.0. It is packaged with TinyXML (zlib/libpng license – see Thomason et al. (2010)) and EasyBMP (Modified BSD license – see Macklin (2005–present)). We request that users cite this paper and the project website in their "methods" section when publishing results that make substantial use of the code or derivative works.

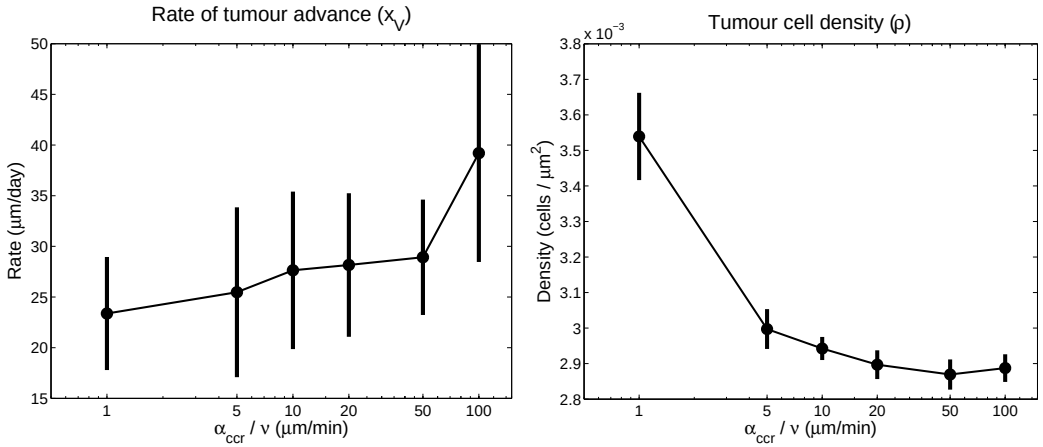

Fig. 4. **Robustness of the mechanics parameters:** We varied the cell-cell repulsive force $\alpha_{\mathrm{ccr}}/\nu$ while maintaining the relative balance of the forces; this is equivalent to varying the biomechanics time scale. For $\alpha_{\mathrm{ccr}}/\nu$ within an order of magnitude of our initial estimate, both the tumour front velocity (left plot) and viable rim cell thickness (right plot) varied little from our baseline simulation. Bars represent one standard deviation from the computed mean for each parameter value.

3 Additional numerical parameter studies

We performed additional parameter students to those in Part II, which were cut from the main manuscript but nonetheless support and/or further investigate the model, and may be of interest to the reader.

3.1 Robustness of the mechanics parameters

In Section 3.6 in Part II, we estimated the cell-cell repulsion parameter $\alpha_{\mathrm{ccr}}/\nu$ to be on the order of 10 μm/min. To assess the sensitivity of the model to error in this estimate, we varied $\alpha_{\mathrm{ccr}}/\nu \in \{1, 2, 5, 10, 20, 100\}$ μm/min and simulated 30 days of growth with all other parameters as in the baseline case (Section 7 in Part II). In particular, we kept $\alpha_{\mathrm{cca}}/\alpha_{\mathrm{ccr}}$ constant for all the simulations to maintain the target cell density as in the calibration protocol, and we set $\alpha_{\mathrm{cbr}} = \alpha_{\mathrm{ccr}}$, and $\alpha_{\mathrm{cba}} = 10\alpha_{\mathrm{cca}}$. Changing α_{ccr} while maintaining these ratios of forces is equivalent to altering the biomechanics time scale.

For each combination of mechanics parameters, we calculated the smoothed tumour front velocity $x_{\mathrm{V}}'(t)$ at one-hour intervals from 10 to 25 days. (x_{V} exceeds 1 mm at 30 days for $\alpha_{\mathrm{ccr}}/\nu = 100$ μm/min.) For any t, we calculated the smoothed $x_{\mathrm{V}}'(t)$ based upon the least-squares linear fit to x_{V} on $t \pm 24$ hours. In Fig. 4: left, we plot $\langle x_{\mathrm{V}}' \rangle$ versus $\alpha_{\mathrm{ccr}}/\nu$; the bars denote one standard deviation above or below the mean. For $5 \leq \alpha_{\mathrm{ccr}}/\nu \leq 50$ μm/min, the tumour front velocity was comparable, indicating that our simulations are robust so long

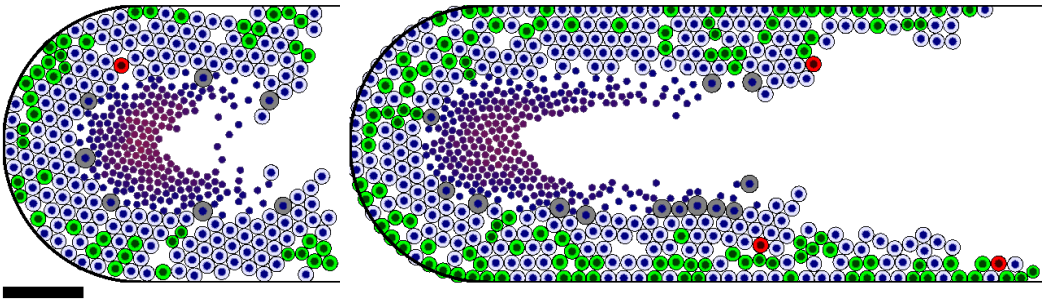

Fig. 5. **Impact of the ratio of cell-cell and cell-BM adhesive forces** $\alpha_{\text{cca}}/\alpha_{\text{cba}}$: Left: For $\alpha_{\text{cca}}/\alpha_{\text{cba}} = 1$ and greater, cells easily pull away from the duct wall except in regions of densely packed duct. Right: For $\alpha_{\text{cca}}/\alpha_{\text{cba}} = 0.01$, cells get trapped near the BM, leading to a "wetting" effect that accelerates the tumour's advance through the duct. High oxygenation near the duct wall acts as nonlinear feedback to accelerate the process. Cells are coloured as in Fig. 3. Bar: 100 μm. A colour version of this figure is available in the online edition.

as we can estimate the mechanics parameters within an order of magnitude. This is advantageous, as the individual cell mechanics parameters are among the most difficult to measure accurately.

To further evaluate the model's robustness, we calculated the mean cell density ρ throughout the viable rim at one-hour increments from 10 to 25 days for each of these simulations. In Fig. 4: right, we plot $\langle\rho\rangle$ versus α_{ccr}/ν; the bars denote one standard deviation above or below the mean. Similarly to $\langle x'_{\text{V}}\rangle$, the mean cell density was comparable for $5 \leq \alpha_{\text{ccr}}/\nu \leq 100$ μm/min. This again indicates that our simulations are robust so long as we can estimate the mechanics parameters within an order of magnitude. Note that if $\alpha_{\text{ccr}}/\nu \leq 1$ μm/min, then the cell density increases rapidly. This is consistent with our earlier results that the cell-cell repulsion parameter must be on the order of 10 to prevent overlapping cells.

3.2 On the balance of cell-cell and cell-BM adhesion

We studied the impact of the balance of cell-cell and cell-BM adhesive forces by varying $\frac{\alpha_{\text{cca}}}{\alpha_{\text{cba}}} \in \{0.01, 0.1, 1, 10\}$, while maintaining $\frac{\alpha_{\text{cca}}}{\alpha_{\text{ccr}}}$ constant. As this ratio is increased to 1 and above, the cells begin to pull off the BM except in regions of dense cell packing. See Fig. 5: left for a typical example (at 15 days) with $\frac{\alpha_{\text{cca}}}{\alpha_{\text{cba}}} = 1$. In 3D, the curvature of the duct may reduce this effect.

On the other hand, for $\frac{\alpha_{\text{cca}}}{\alpha_{\text{cba}}} = 0.01$, it was very difficult for daughter cells to push away from the BM after proliferation, leading to a "wetting" effect along the duct wall similar to a fluid capillary force. See Fig. 5: right (plotted at 15 days). This increased the percentage of cells near the high-oxygen regions of the duct, which acted as a feedback that accelerates the tumour's advance

through the duct at unrealistic rates. See Video S2. Because the cell-BM adhesive force is modelled as normal to the BM (thereby neglecting any tangential component), the BM is effectively frictionless, further exacerbating this effect.

3.3 Amount of necrotic cell swelling primarily influences the gap

We varied the level of necrotic cell swelling $f_{\mathrm{NS}} \in \{0\%, 30\%, 100\%\}$ and found virtually no impact on the least-squares fit of the rate of tumour advance from 15 to 30 days. (24.78, 24.38, and 24.25 μm/day, respectively.) Instead, the primary impact was to increase the size of the physical gap, with increases scaling roughly as $(1 + f_{\mathrm{NS}})^{\frac{1}{3}}$ (result not shown).

3.4 Heterogeneous oxygen uptake causes perinecrotic bounary instability

In Section 3.5 in Part II, we estimated that proliferating and non-proliferating tumour cells uptake oxygen at approximately the same rate, with $\lambda_{\mathrm{p}} = \lambda_{\mathrm{np}}$. We close with an investigation of $\frac{\lambda_{\mathrm{p}}}{\lambda_{\mathrm{np}}} \in \{1, 10, 100\}$. In each simulation, we maintained $\langle\lambda\rangle = \mathrm{PI}\lambda_{\mathrm{p}} + (1 - \mathrm{PI})\lambda_{\mathrm{np}}$ constant. The results at time 30 days are plotted in Fig. 6. As $\frac{\lambda_{\mathrm{p}}}{\lambda_{\mathrm{np}}}$ is increased, the stability of the perinecrotic boundary is reduced, with greater mixing of necrotic cellular debris and viable tumour cells. This occurs because high oxygen uptake by isolated proliferating cells creates small pockets of hypoxia between these cells and the necrotic core. See Video S3. The result is a ragged perinecrotic boundary not typically observed in DCIS, further supporting the estimate that $\lambda_{\mathrm{p}} \approx \lambda_{\mathrm{np}}$.

However, it is interesting to observe that such an instability can result from microscopic variations in cell metabolism caused by cell-induced, fine-scale alterations in the tumour microenvironment; such instabilities are often attributed to variations in cellular adhesion. While the result here is likely non-physical for oxygen transport, similar behaviour could occur in glucose transport, where glucose uptake is much greater for hypoxic cells than non-hypoxic cells (Smallbone et al., 2007b,a; Gatenby et al., 2007).

4 Simulation Animations

To better illustrate the key results, we include the following animations below. In each animation, the cells are labelled as follows:

- Dark blue circles: cell nuclei

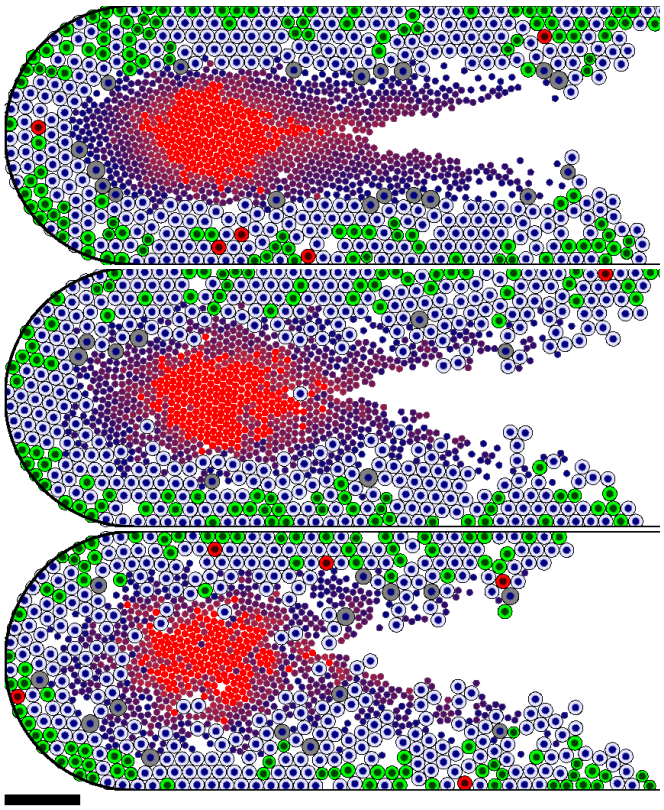

Fig. 6. **Impact of heterogeneous oxygen uptake rates:** As the ratio λ_p/λ_{np} of the oxygen uptake rates by the proliferating (λ_p) and nonproliferating cells (λ_{np}) is increased from 1 (**top**) to 10 (**middle**) and 100 (**bottom**), the stability of the perinecrotic boundary is reduced, with greater mixing of necrotic cellular debris and viable cells. These morphologies are not typical in DCIS, supporting the estimate that $\lambda_p \approx \lambda_{np}$. Cells are coloured as in Fig. 3. Bar: 100 μm. A colour version of this figure is available in the online edition.

- Green cells: proliferating cells ($\mathcal{S} = \mathcal{P}$; cells in non-$G_0$ phase)
- Pale blue cells: quiescent cells ($\mathcal{S} = \mathcal{Q}$; cells in G_0 phase)
- Red cells: apoptotic cells ($\mathcal{S} = \mathcal{A}$)
- Gray cells: necrotic cells prior to lysis ($\mathcal{S} = \mathcal{N}$)
- Red circles: necrotic cellular debris after lysis ($\mathcal{S} = \mathcal{N}$); shade of red indicates the degree of calcification
- Bright red circles: clinically-detectable calcified cellular debris ($\mathcal{S} = \mathcal{C}$)

The movies are in AVI format and compressed with the Xvid codec. The open source VLC media player can play these movies on multiple platforms, including Windows, OSX, and Linux. Alternate formats are indicated below.

(1) **Video S1:** the "baseline" simulation from Section 7 in Part II, plotted in 1.5 mm of duct from 0 to 45 days.
Alternate format: `http://www.youtube.com/watch?v=b_GVnZWVhgk`.

(2) **Video S2:** "wetting" behaviour when cell-BM adhesion is strong relative to cell-cell adhesion ($\alpha_{\text{cba}} = 100\alpha_{\text{cca}}$; see Section 3.2 in Part II), plotted from 0 to 30 days in 1 mm of duct.
Alternate format: `http://www.youtube.com/watch?v=9q9LGzX9fok`.

(3) **Video S3:** unstable perinecrotic boundary (between the viable rim and the necrotic core) resulting from heterogeneous cellular oxygen uptake rates ($\lambda_{\text{p}} = 100\lambda_{\text{np}}$; see Section 3.4 in Part II), plotted from 0 to 30 days in 1 mm of duct.
Alternate format: `http://www.youtube.com/watch?v=Brgw8qI8k-k`.

References

F. V. Berghen. xmlParser project website, 2009. URL `http://www.applied-mathematics.net/tools/xmlParser.html`.

J. Clark. Expat XML parser project website, 2007. URL `http://expat.sourceforge.net/`.

V. Cristini and J. Lowengrub. *Multiscale modeling of cancer*. Cambridge University Press, Cambridge, UK, 2010. ISBN 978-0521884426.

R. A. Gatenby, K. Smallbone, P. K. Maini, F. Rose, J. Averill, R. B. Nagle, L. Worrall, and R. J. Gillies. Cellular adaptations to hypoxia and acidosis during somatic evolution of breast cancer. *Br. J. Cancer*, 97(5):646–53, 2007. doi: 10.1038/sj.bjc.6603922.

S. Gottlieb and C.-W. Shu. Total variation diminishing runge-kutta schemes. *Math. Comp.*, 67(221):73–85, 1997. doi: 10.1090/S0025-5718-98-00913-2.

S. Gottlieb, C.-W. Shu, and E. Tadmor. Strong stability-preserving high-order time discretization methods. *SIAM Review*, 43(1):89–112, 2001. doi: 10.1137/S003614450036757X.

P. Macklin. EasyBMP Cross-Platform C++ BMP library project, 2005–present. URL `http://easybmp.sourceforge.net`.

P. Macklin. EasyBMPtoAVI movie creator project, 2006–present. URL `http://easybmptoavi.sourceforge.net`.

P. Macklin. Biological background. Cristini and Lowengrub (2010), chapter 2, pages 8–24. ISBN 978-0521884426.

P. Macklin and J. S. Lowengrub. Evolving interfaces via gradients of geometry-dependent interior poisson problems: application to tumor growth. *J. Comput. Phys.*, 203(1):191–220, 2005. doi: 10.1016/j.jcp.2004.08.010.

P. Macklin and J. S. Lowengrub. An improved geometry-aware curvature discretization for level set methods: application to tumor growth. *J. Comput. Phys.*, 215(2):392–401, 2006. doi: 10.1016/j.jcp.2005.11.016.

P. Macklin and J. S. Lowengrub. A new ghost cell/level set method for moving

boundary problems: Application to tumor growth. *J. Sci. Comp.*, 35(2–3): 266–99, 2008. doi: 10.1007/s10915-008-9190-z.

P. Macklin, S. McDougall, A. R. A. Anderson, M. A. J. Chaplain, V. Cristini, and J. Lowengrub. Multiscale modeling and nonlinear simulation of vascular tumour growth. *J. Math. Biol.*, 58(4–5):765–98, 2009. doi: 10.1007/s00285-008-0216-9.

P. Macklin, M. E. Edgerton, and V. Cristini. Agent-based cell modeling: application to breast cancer. Cristini and Lowengrub (2010), chapter 10, pages 216–244. ISBN 978-0521884426.

P. Macklin, M. E. Edgerton, J. Lowengrub, and V. Cristini. Discrete cell modeling. Cristini and Lowengrub (2010), chapter 6, pages 92–126. ISBN 978-0521884426.

P. Macklin, M. E. Edgerton, and V. Cristini. Patient-calibrated agent-based modelling of ductal carcinoma in situ (DCIS) II: From microscopic measurements to macroscopic predictions of clinical progression. *J. Theor. Biol.*, 2011. (in review).

Mathworks. MATLAB 7 MAT-file format, 2010. URL `http://www.mathworks.com/help/pdf_doc/matlab/matfile_format.pdf`.

T. Pham, editor. *Computational Biology: Issues and Applications in Oncology*. Springer, New York, NY USA, 2009. ISBN 978-1-4419-0810-0.

W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 2nd edition, 1992. ISBN 0-521-43108-5.

K. Smallbone, R. A. Gatenby, R. J. Gillies, P. K. Maini, and D. J. Gavaghan. Metabolic changes during carcinogenesis: Potential impact on invasiveness. *J. Theor. Biol.*, 244(2):703–713, 2007a. doi: 10.1016/j.jtbi.2006.09.010.

K. Smallbone, D. J. Gavaghan, P. K. Maini, and J. M. Brady. Quiescence as a mechanism for cyclical hypoxia and acidosis. *J. Math. Biol.*, 55(5–6): 767–79, 2007b. doi: 10.1007/s00285-007-0105-7.

L. Thomason et al. TinyXML project website, 2010. URL `http://www.sourceforge.net/projects/tinyxml`.