**INTRODUCTION**

A computational model is a mathematical model in computational science which studies the behavior of a complex system by computer simulation. The system under study is often a complex nonlinear system for which simple, intuitive analytical solutions are not readily available. Rather than deriving a mathematical analytical solution to the problem, experimentation with the model is done by adjusting the parameters of the system in the computer, and studying the differences in the outcome of the experiments. Operation theories of the model can be derived/deduced from these computational experiments [5].

Examples of common computational models are weather forecasting models, earth simulator models, flight simulator models, molecular protein folding models, and neural network models [5].

Stem cells are undifferentiated biological cells that can differentiate into specialized cells and can divide through mitosis to produce more stem cells. They are found in multicellular organisms [6]. Mitosis is the cell cycle process which divides into two new cells containing roughly equal shares of cellular components [8].

Cancer stem cells (CSCs) are cancer cells found within tumors or that possess characteristics associated with normal stem cells, specifically the ability to give rise to all cell types found in a particular cancer sample. CSCs are therefore tumor-forming, perhaps in contrast to other non- tumor-forming cancer cells.

Further, new information suggesting a dependence of tumour composition and growth on the microenvironment has yet to be studied theoretically [1]. The theoretical model is a computational model which is a hybrid, discrete/continuous computational cellular automaton model of a generalised stem-cell driven tissue with a simple microenvironment.

A cellular automaton consists of a regular grid of cells, each in one of a finite number of states, such as on and off. The grid can be in any finite number of dimensions. For each cell, a set of cells called its neighborhood is defined relative to the specified cell[3].

Using the phenotypic traits inherent to the tumour initiating cells, and the effect of the microenvironment on tissue growth can be explored[1]. A phenotype is the composite of an organism's observable characteristics or traits, such as its morphology, development, biochemical or physiological properties, phenology, behavior, and products of behavior. A phenotype results from the expression of an organism's genes as well as the influence of environmental factors and the interactions between the two. The regions in phenotype parameter space can be identified where tumour initiating cells (TICs) are able to cause a disruption in homeostasis, leading to tissue overgrowth and tumour maintenance.

Since the discovery of TICs in solid tumours, studies focussing on their role in cancer initiation and progression have abounded. The biological interrogation of these cells continues to yield volumes of information on their pro-tumourigenic behaviour, but actionable generalised conclusions have been scarce. Further, new information suggesting a dependence of tumour composition and growth on the microenvironment has yet to be studied theoretically[1].

Cancer stem cells (CSCs) are special type of cells which have been shown to associate with different aggressive cancer phenotypes including drug resistance. CSCs are transformed cancer cells possessing the properties similar to stem cells. In this project, we want to develop a computational model based on the available experimental data to gain further insights about the origin of CSCs and their role in promoting cancer. The computational model will be developed by combining discrete and continuous modeling approaches.

**PROBLEM STATEMENT**

Cancer Stem cells abbreviated as CSCs are cancerous cells that exhibit properties similar to normal stem cells. This means that CSCs are multipotent and are able to differentiate into cancer cells and can undergo self-renewal. CSCs essentially are tumorigenic, meaning they are capable of creating tumors, a quality other cancerous cells do not possess. Another quality of CSCs is immortality; whereas other cells have a limited number of times they can divide CSCs are able to divide indefinitely [13]. One of the main problems of CSCs in cancer treatment is that they are generally unaffected by chemotherapy used to kill most differentiated cancer cells (which make up most of the tumor). CSCs generally make up about 1-3% of a tumor [14]. Thus, following chemotherapy, CSCs left behind would be able to replenish a tumor and cause a relapse of the cancer [15]. In addition, tumor modeling and understanding relapse due to CSCs are currently ill understood because most organisms with relapse cancers *in vitro* die before they can be further studied. If unheeded, CSCs can, in theory, cause continual relapses of a tumor, and are capable of metastasis – the migration cancerous cells including CSCs to other organs or tissues in the body to create new tumors (carcinogenesis).

By applying discrete models, automata theory, and cellular automaton programming to create more accurate models of population growth and a better understanding of population dynamics[12].

**MOTIVATION**

Therefore, development of specific therapies targeted at CSCs holds hope for improvement of survival and quality of life of cancer patients, especially for patients with metastatic disease[7].

CSCs may generate tumors through the stem cell processes of self-renewal and differentiation into multiple cell types. Such cells are proposed to persist in tumors as a distinct population and cause relapse and metastasis by giving rise to new tumors. Therefore, development of specific therapies targeted at CSCs holds hope for improvement of survival and quality of life of cancer patients, especially for patients with metastatic disease[7].

**OBJECTIVES AND SCOPES**

As our parameters and model are non-specific, they could apply to any tissue TIC and do not assume specific genetic mutations.

Successful completion of the problem will contribute to our understanding of how CSCs contribute to cancer invasiveness.

**DESCRIPTION OF THE WORK**

Hence the cellular automaton , where an initial state (time t = 0) is selected by assigning a state for each cell. A new generation is created advancing t by 1, according to some fixed rule generally, a mathematical function that determines the new state of each cell in terms of the current state of the cell and the states of the cells in its neighborhood. Typically, the rule for updating the state of cells is the same for each cell and does not change over time, and is applied to the whole grid simultaneously[3].

Rule 3: Make Your Code Understandable to Others (and Yourself) When revising or adapting existing code, the absence of documentation and comments can result in errors and time drains. Such documentation not only makes your code more understandable to others but also to your future self (put simply, the code tells you ‘‘how’’, the comments tell you ‘‘why’’). The program code itself can be made more understandable by using meaningful variable names and formatting the code consistently. While commenting and documentation is often neglected when faced with deadlines, developing and maintaining a standardized way of commenting your code will be of great benefit. As well as low-level documentation in the code, you should maintain a record of the ‘‘big picture’’ functionality (i.e., interconnectivity of components and input/output formats).

This could take the form of a high-level

diagram or description of the system,

whether by hand on paper, in verbose

code comments, or using standardized

approaches such as UML (Unified Modelling

Language) (see Text S1). When you

are reviewing your code for documentation

you should actively seek ways to break

it up into modules. This not only aids

structure and readability but also avoids

the error-prone and tedious task of

debugging and updating two (or more)

copies of the same code. As a rule of

thumb, if you write the same code twice, it

should become a function, subroutine, or

method.

Develop a Prototype First Before writing any code, it is imperative to clarify what you are trying to implement: what functionality do you require, and what interfaces do you need? When implementing your latest developments, you should first begin by considering a prototype (i.e., a simplified version of the full system or algorithm) to gain insight and to guide the next steps. This is equally relevant whether building on existing code or starting from scratch. By prototyping new functionality and building code up incrementally, you can check that each element of your code operates as expected (and each incremental development can be tested; see Rule 8). Breaking your problem up into smaller elements like this will also help to provide structure to your code and will make it much easier when you subsequently need to extend it. From a practical point of view, it will usually be easier to prototype mathematical and statistical methods in a ‘‘higher-level’’ language, for example Matlab, R, or Python. Although these languages can be slower to run than optimized code in a ‘‘lower-level’’ language, their straightforward nature, built-in functionality, and available libraries mean that you will spend less time expressing your ideas in code and searching for bugs[2].

Rule 7: Version Control Everything Version control systems (VCSs) offer an easy way to store and back up not only the current version of your code that you are working on but also every previous version of the code (in what’s known as a repository). This not only saves you from having to keep multiple copies of the same file but also allows you to ‘‘roll back’’ to an older ‘‘working’’ version of the code if things go wrong. VCSs also allow you to share material between multiple machines, operating systems, and more importantly, users in a simple and robust manner. Two of the most popular VCSs are Subversion (http://subversion.apache.org) and Git (http://www.github.com), both of which offer many advanced features for managing your code. Cloud storage such as Dropbox (http://www.dropbox.com) and SkyDrive (http://www.skydrive.live.com) offer basic file sharing and backup facilities; however, they don’t offer the code management features of true VCSs, so the effort put in to learning a VCS is well worth it (see Text S1 for guides on getting started with VCSs). While the primary use of version control is to manage the development and distribution of code, many other collaborative endeavours can be stored in a versioncontrol repository. In particular, using version control tools while preparing publications can save time and effort, especially when dealing with input from multiple authors. For example, contributions to this manuscript were managed using a VCS.

Test Everything Any non-trivial computer program will have bugs when first written, often subtle ones that are hard to detect, which may lead to incorrect results. Indeed, in extreme cases this has caused high-profile retractions of papers [10]. Simple tests that the software behaviour matches expectations are essential for ensuring robust results, minimising the presence of bugs, and gaining confidence in your code (for you and others). As a result of the time pressures inherent in academia, often software testing is performed manually in an ad hoc manner, to determine whether results ‘‘look roughly right’’ [11]. However, a systematic approach to testing pays dividends. You should learn how to test effectively to avoid the illusion of reliability. For example, compare low-level routines against analytical or prototype solutions (see Rule 2) or experimental data and consider ‘‘corner cases’’ and both branches of ‘‘if’’ statements. Get the computer to run tests for you automatically and alert you to problems, using a suitable testing framework (see Text S1). Ideally this should be tied to a version control system (see Rule 7) so that tests are run automatically whenever new code is committed to the repository. A useful rule is to turn bugs you fix into new tests to avoid them recurring. Testing gives you the confidence to modify your code without worrying that you are breaking it. Testing can also provide a means for reproducing results of published papers. By setting up a test comparing against published values, you can easily find out when fixing a newly identified bug changes published results.

The model will integrate the cell evolution cycle related to CSC and effects of microenvironmental parameters on this cycle. Computational predictions will be experimentally tested by PhD students in the lab.

**CONCULSION**

How to develop computational models for getting insights into biological problems

**REFERENCES**

[1] Jacob G. Scott, Anita B. Hjelmeland, Prakash Chinnaiyan, Alexander R. A. Anderson, David Basanta : Microenvironmental Variables Must Influence Intrinsic Phenotypic Parameters of Cancer Stem Cells to Affect Tumourigenicity, PLOS Computational Biology, e1003433, Volume 10, Issue 1, 7 January 2014

[2] James M. Osborne, Miguel O. Bernabeu, Maria Bruna, Ben Calderhead, Jonathan Cooper, Neil Dalchau, Sara-Jane Dunn, Alexander G. Fletcher, Robin Freeman, Derek Groen, Bernhard Knapp, Greg J. McInerny, Gary R. Mirams, Joe Pitt-Francis, Biswa Sengupta, David W. Wright, Christian A. Yates, David J. Gavaghan, Stephen Emmott, Charlotte Deane : Ten Simple Rules for Effective Computational Research, PLOS Computational Biology, e1003506, Volume 10, Issue 1, March 2014

[12] Andrew Q. Ninh : Two Discrete Stochastic Cellular Automata Models of Cancer Stem Cell Proliferation, International Journal of Bioscience, Biochemistry and Bioinformatics, Vol. 3, No. 5, September 2013

[13] H. Sugihara : “Cell kinetic and genetic lineage analyses of cancer development,” Thesis paper. Dept. of Pathology (Division of Molecular and Diagnostic Pathology), Shiga University of Medical Science, Shiga Prefecture, Japan, n.d.

[14] University of Michigan Stem Cell Research (2011):Cancer Stem Cell Research Introduction Defining Stem Cells: http://www.cancer.med.umich.edu/research/stemcells\_two.shtml

[15] G. Vogel. : (2012) Cancer Stem Cells can Fuel Tumor Growth. ScienceNOW. http://news.sciencemag.org/sciencenow/2012/08/cancer-stem-cells-can-fuel-tumor.html.

[3] Wolfram, Stephen : Statistical Mechanics of Cellular Automata, Reviews of Modern Physics 55 (3): 601–644. Bibcode:1983RvMP...55..601W. doi:10.1103/RevModPhys.55.601.

[4] Phenotype : http://en.wikipedia.org/wiki/Phenotype

[5] Computational model : http://en.wikipedia.org/wiki/Computational\_model

[6] Stem cell : http://en.wikipedia.org/wiki/Stem\_cell

[7] Cancer stem cell : http://en.wikipedia.org/wiki/Cancer\_stem\_cell

[8] Mitosis : http://en.wikipedia.org/wiki/Mitosis

[9] Cellular automaton : http://en.wikipedia.org/wiki/Cellular\_automaton