

CE 903/CE913

TEAM 12

REDACTED

Final Report

Document

Version: (1)

Date: (21/03/2024)

Table of Contents

Introduction	3
1.1 Problem Description	3
1.2 Moran Process	3
1.3 Methodologies	4
1.3.1 Scrum Framework	5
1.4 Software and Tools	5
1.4.1 Tools for project management and planning	5
1.4.2 Development tools and libraries	6
1.5 Network Model Selection Barabasi-Albert vs. Planar Graphs vs. Delaunay Triangulation	7
1.6 The moran process in metastatic cancer spread	9
System Design	9
2.1 Brief recap of requirements	9
Design and Implementation: Sprints	11
Sprint 1	11
Sprint 2	11
Sprint 3	12
Sprint 4	13
Sprint 5	13
Sprint 6	14
Sprint 7	14
Key Code	15
Data Collection	18
Data Analysis	19
Conclusions	23
References	24

Introduction

1.1 Problem Description

The moran process is a birth-death process that models the spread of mutations in two-type populations (resident-mutants) whose structure is defined by a graph. One can think of this process as being performed on computer networks where the mutant is a virus, a social network where the mutant is a rumour or idea, or even tissue where the mutant is a cancerous cell. For this project our main objective is to create a program which simulates the moran process on different real life graphs, these can be either social network graphs, tissue graphs, etc. and recall different values such as win rate for residents or mutants, and with said values evaluate which of the different graphs tested is better at handling mutants.

1.2 Moran Process

Named after the biologist Patrick Moran, this constitutes a random mathematical model used to describe the dynamics of genetic drift within finite populations. It is particularly relevant in evolutionary biology and population genetics. The Moran Process provides an insight into the spread of mutations through connected graphs, where it reaches "fixation", meaning that all vertices are mutants, or "extinction", where none are [2]. The fixation probability determines how aggressively mutants propagate to their neighbours.

The essential attributes of the Moran process embrace [3][4][5]:

- *Population structure*

Represented by graphs where each node represents an individual and links define the neighbourhood of each individual.

- *Reproduction and selection*

In the Moran process, individuals reproduce asexually, producing offspring that inherit genetic material from their parents. One individual per period “dies” and is replaced by a newcomer, being like that the newcomer’s strategy is a random variable whose distribution is given by an updating function, depending only on the current state.

- *Random drift*

Random fluctuations of allele frequencies in a population over time due to sampling error in finite populations. Representing a specific balance between selection

and drift where advantageous mutations have a certain chance (without guarantee) of fixation.

- *Population dynamics*

Allows the fluctuation of the allele frequencies evolving over successive generations under the combined influence of the previous points; by simulating multiple iterations of the process, it can lead to the observation of patterns of genetic change and the prediction of long-term evolution.

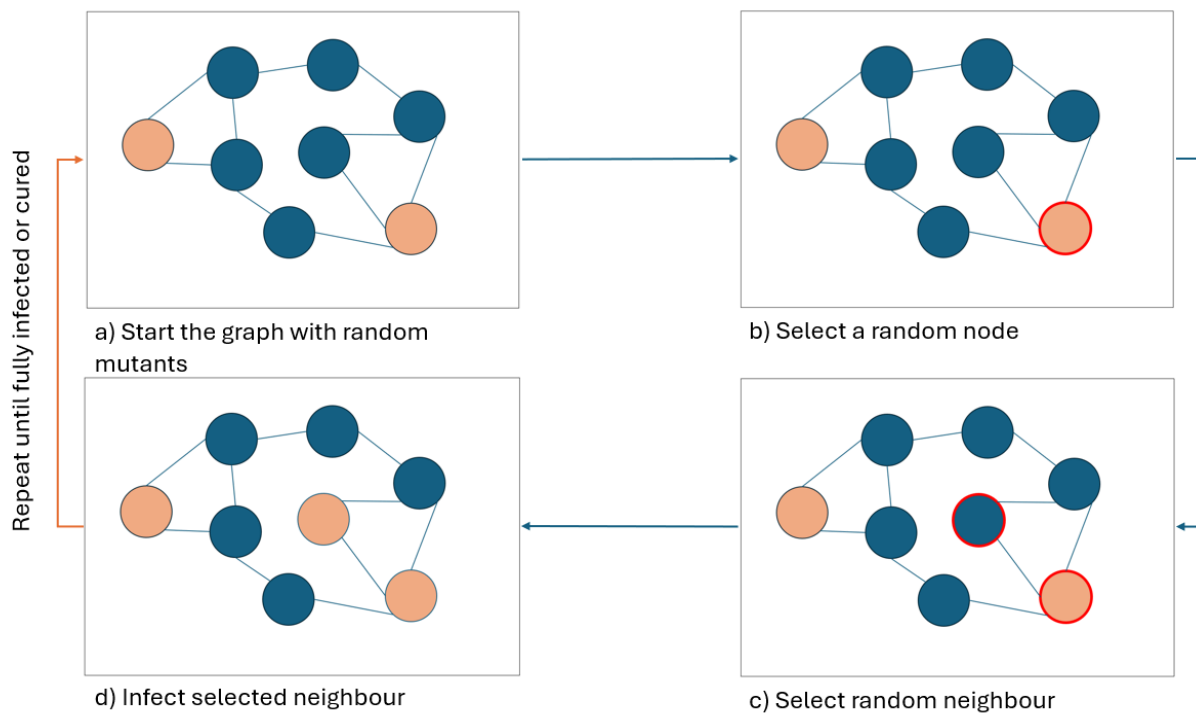


Figure 1. Moran process flow chart

1.3 Methodologies

For the planning of this project we opted for the use of the scrum framework over the use of the waterfall and traditional agile structures. The scrum framework brings a lot of flexibility in the development process and allows for changes that can come from feedback and new ideas, the absence of the rigidity that is present in the other frameworks can make the team work more fluidly and gives more freedom when in the development process.[1]

1.3.1 Scrum Framework

The scrum framework outlines a set of values, principles, and practices that scrum teams follow to deliver a product or service. The scrum framework is based on continuous learning and adjustment to fluctuating factors. It acknowledges that the team doesn't know everything at the start of a project and will evolve through experience. Scrum works through a type of iteration known as Sprints, sprints are short periods of time in which the development team implements and delivers a product which has an achievable milestone set.[1]

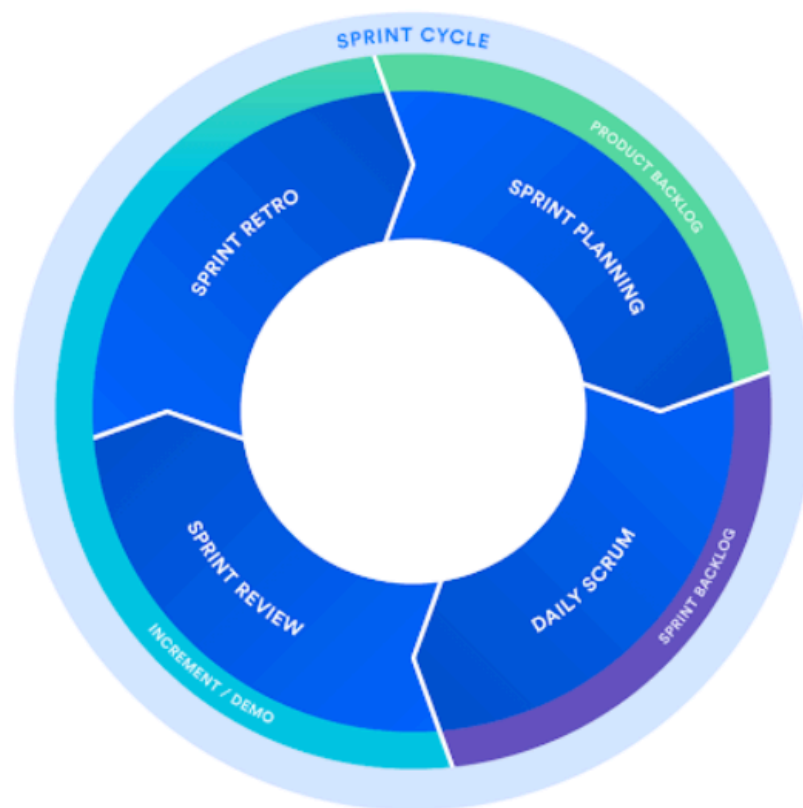


Figure 2. Example of a sprint structure.

1.4 Software and Tools

1.4.1 Tools for project management and planning

When it came to planning our project we had to find our planning platforms. We decided to use Trello and Jira as they are connected and you use both together. If we look at both of these Trello is a free project planning platform which allows you to create boards that you can follow. Next is Jira, we

used this to track the time scale of the project so we know when our deadlines are. We also used this to follow any bugs or errors that a team member found. When combined Trello and Jira can become a very powerful way to track our project allowing us to use the features both offer to greatly benefit us. [1]

1.4.2 Development tools and libraries

1. NetworkX

NetworkX is a Python package for creating, manipulating, and studying the structure and dynamics of complex networks.[6] It provides data structures and algorithms for working with graphs, including graph generation and layout algorithms. NetworkX is extensively used in this project for representing and manipulating the graph structure, making it an effective and appropriate choice for this simulation.

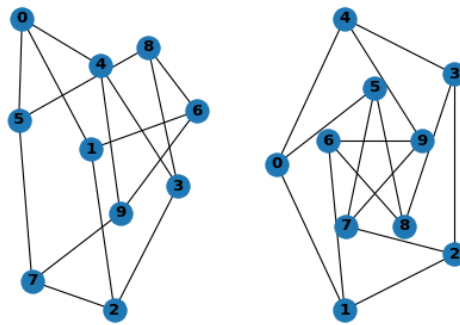


Figure 3. Example of a graph represented with Networkx.

2. NumPy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays.[7] In this project, we used NumPy for handling and manipulating the graph data, population status, and fitness values. NumPy's efficiency and versatility make it an appropriate choice for this application.


3. Matplotlib

Matplotlib is a 2D graphics package used for Python for application development, interactive scripting, and publication-quality image generation across user interfaces and operating systems.[8] Matplotlib in combination with NetworkX's drawing capabilities is being utilised in this project for visualising the graph and simulation results.

4. Tqdm

The tqdm library is a progress bar library for Python, which can be used to display progress bars for long-running operations.[9] It is used to provide visual feedback during the simulation process.

```
from tqdm.notebook import tqdm_notebook
for i in tqdm_notebook(range(2), desc = 'Loop 1'):
    for j in tqdm_notebook(range(20,25), desc = 'Loop 2'):
        time.sleep(0.5)
```



Loop 1: 100% 2/2 [00:05<00:00, 2.71s/it]

Loop 2: 100% 5/5 [00:02<00:00, 1.91it/s]

Loop 2: 100% 5/5 [00:02<00:00, 1.85it/s]

Figure 4. Example of a progress bar created with the tqdm library.

5. Copy

The copy module from the Python standard library is used for creating deep copies of objects. In this project, we used it to create a copy of the original MoranGraph instance, allowing for resetting the simulation to its initial state without modifying the original graph.

6. Random

The random module from the Python standard library provides functions for generating random numbers. We used it in the project for selecting random nodes and determining the offspring location based on fitness values.

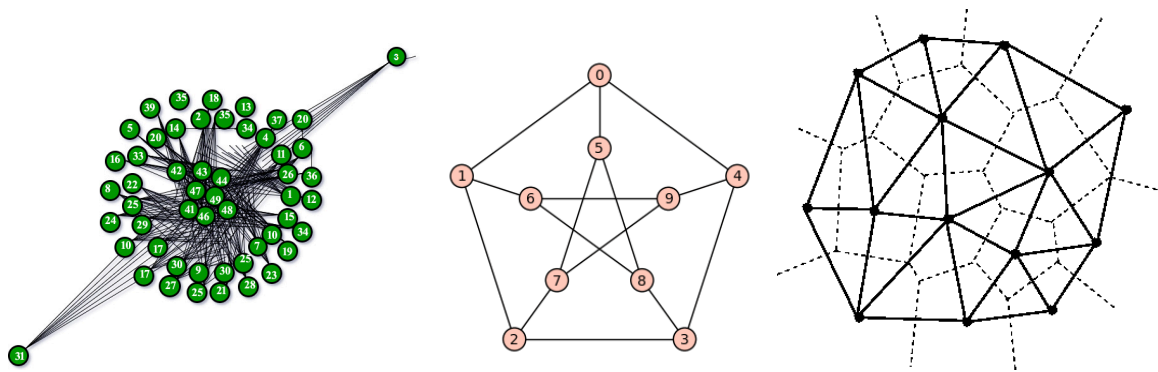
7. Pygame

Pygame is a python library used to create interactive python applications like games. It provides a number calls which make drawing graphics to the screen and taking user input during application runtime easy. This module was used extensively in the interactive visualisation of the Moran process.

1.5 Network Model Selection Barabasi-Albert vs. Planar Graphs vs. Delaunay Triangulation

The decision to employ the Barabasi-Albert model for our Moran process, initially conceived for real-life network graphs, signifies a deliberate departure prompted by the specific requirements of our project centred on cancer cells. We were opting for this model due to its capacity to simulate scale free-networks with a power law degree distribution, being this a pervasive feature in social networks, contrasting planar graphs and Delaunay triangulation's lack of this behaviour. Additionally, the model in question contemplates a dynamic incorporation of preferential attachment and growth, reflecting the evolving structure of social networks over time, while the rejected models remain static. [10]

Moreover, the Barabasi-Albert model offers several advantages for modelling network structures. It exhibits the small-world property, with short average path lengths between nodes, akin to real networks, unlike planar graphs and Delaunay triangulations [11], while also generating heterogeneous networks with varying node degrees, including hubs, contrasting with the more uniform distributions found in other models [12]. Finally, leveraging its scale-free and small-world characteristics, the Barabasi-Albert model facilitates spreading processes such as the Moran process due to rapid selection and replacement through hubs, features lacking in alternative models.



Figures 5,6,7. Examples of Barabasi-Albert (left), Planar Graph (middle) and Delaunay Triangulation (right)

Originally our objective was to test the Moran process in a variety of real-life scenarios such as the ones mentioned previously (Computer Networks and Social Networks), but, due to time constraints, we found ourselves having to decide only focusing on one type of network for this project. We decided to go with cell tissue representations in graphs because these are easier to find since we can generate them, on the other hand for our project to have validation with social networks and computer networks we would have to get real-life representations of them which we weren't able to do. According to “The mechanical anisotropy in a tissue promotes ordering in hexagonal cell packing” by Kaoru Sugimura [13] cellular representation comes in a hexagonal pattern which can be represented in a graph (Figure 8).

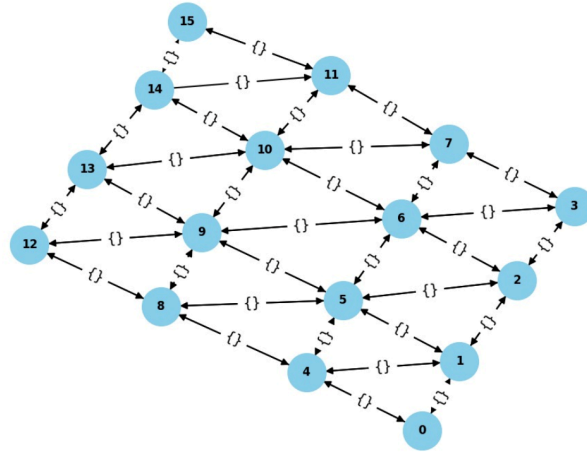


Figure 8. Graph representation of cellular tissue.

1.6 The moran process in metastatic cancer spread

In order to comprehend the approach of our project we need to understand the metastatic process, this referring to the spread of cancer from its original site to another site of the body [14]. The development of metastasis typically involves cancer cells breaking away from the primary tumour and entering the bloodstream or the lymphatic system where it can travel to distant sites and form new tumours when they settle and grow in different body parts.

In the Moran process, the probability of fixation (p) is a measure of the likelihood that a new mutant will dominate or infect the entire population. This concept can be applied to the spread of cancer cells, where (p) represents the probability of a successful metastatic event per cell or per unit of tumour volume.

Considering what was established by J.S. Michaelson [15] we can infer that the spread of cancer cells may occur with probabilities indicative of a nongenetic mechanism, by defining (p) as the probability of metastatic spread per cell (N) implies that not every cell in the tumour has the potential to spread but, for instance, if (p) is 1 in ten billion cells there will be at least one event of metastatic spread.

We can relate this to what J.G Scott [16] determines in his application of the Moran process, where each cell has a certain probability of being selected for reproduction or death, and the fixation of a cell depends on its relative fitness and transition probabilities between states. Then (p) defined in the context of cancer can be seen as analogous to the probability that a metastatic cancer cell will become fixed in the tumour cell population.

System Design

2.1 Brief recap of requirements

The Moran process simulation system is designed to model the spread of mutations in two-type populations based on graph structures. The main objective is to create a program that can simulate the Moran process on various real-life graphs, such as social networks, tissue structures, etc. The system is expected to gather and analyse data, including win rates for resident and mutant populations, to evaluate the effectiveness of different graph structures in handling mutants.

Requirements	Description
Graph Types	The system should support different types of graphs, including social networks, tissue structures, etc., as specified in the SRS document.
Simulation Accuracy	The simulation should accurately model the Moran process, considering factors like birth-death dynamics and graph-based structures.
Data Analysis	The system should provide tools for analysing simulation results, including win rates for residents and mutants.
Flexibility	The system needs to be flexible enough to adapt to various graph structures, allowing users to input different parameters for simulation.
Usability	A user-friendly interface is required to facilitate easy interaction with the system, including inputting graph data, setting simulation parameters, and interpreting results.
Project Management Framework	The use of the Scrum framework for project management is specified to enhance flexibility, accommodate changes, and allow continuous learning throughout the development process.
Visualisation	A visualisation of the simulation should be developed to help aid viewers' understanding of the Moran process.

Table 1. Requirements overview

Having reviewed these requirements, the subsequent sections will delve into the design and implementation

Design and Implementation: Sprints

Since we decided to use an agile approach in the development cycles, we will explain our objectives and testing for each sprint (each week) in the following section:

Sprint 1

For our first sprint we separated the group into three teams: Graphics Development, Algorithm Development and Research. Each of the teams is responsible for setting goals for each sprint and working towards them. For the first sprint the following goals were defined.

- **Graphics**
Research was done to find tools that would be beneficial for creating interactive graph visualisations in Python, our implementation language of choice. The NetworkX library proved to be a formidable tool for generating and plotting node graphs, however, it lacked interactivity. The Pygame library was selected to implement that interactivity because of the tools it provides for making interactive content like games. Node graph data is generated using NetworkX and instead of plotting the data to an image, the Pygame graphics library runs through the graph data and draws circles (for nodes) and lines (for connections) to the screen buffer each frame. Pygame does not offer much boilerplate code like buttons and sliders. In order to solve this problem and avoid redundant code, prefabs for Buttons, Alignment Containers, Labels and Text Boxes were developed.
- **Development**
A simple implementation of the Moran process algorithm was started this sprint, the coding language that was decided by the development team was Python.
- **Research**
Different types of graph structures were researched on the first sprint. There was also a meeting with Dr David Richerby, who has done extensive research on the Moran process, to talk about the different types of graphs that could be used in this project. The main graphs that were found are Barabasi-Albert, Planar Graphs, Delaunay Triangulation.

There wasn't any testing in this sprint due to the fact that it is too early for any meaningful information to come out by testing anything.

Sprint 2

For the second sprint feedback from the supervisor was taken into account by each team.

- **Graphics**
Navigation of the interactive graph was implemented during this sprint. Calculations were made to create a suitable panning motion when the right mouse button is clicked and the mouse is dragged across the screen. Zooming in and out with the mouse scroll wheel also changes the scale of the graph drawn to the screen buffer. The process involves translation and scaling of the node positions as well as the positions of their connections. These features make navigating the visual graph much easier for users.
- **Development**
This sprint the development team corrected the algorithm that was coded last sprint as it had implemented the Moran process wrong as pointed out by the supervisor.
- **Research**
For this sprint further graph research was done. Real life social network graphs were researched as well, but, this sprint the research team wasn't able to find any graphs to be simulated.

Graphics testing and algorithm testing were done separately. However, both of the teams used the same testing method: Unit testing, for each change or function a specific test was done to evaluate if everything was implemented correctly and if not to change it as soon as possible.

Sprint 3

In our third sprint we were recommended to look for real life graphs to start testing the Moran process on, further improvements on the existing algorithm were talked with the supervisor.

- **Graphics**
A Node Prefab is created to hold and manage position and mutation data of the nodes on the graph. The code used to draw the nodes as circles is moved to this prefab to avoid redundancy. Creating this prefab makes it easier to modify the individual positions of the nodes. A bit of synchronisation between the NetworkX graph data and the data from the list of Node Prefabs has to be done to ensure that their positions are correct. To do this, the translation and scaling described in the last sprint is inverted to calculate the positions that would be stored back into the NetworkX graph data structure. With these done,

functionality to drag nodes around the screen and modify their mutation state is implemented successfully.

- Development

This week the algorithm didn't see much changes but bug fixing was still being done to ensure an efficient and correct process is being done. There was also a graph generator being developed for testing of the process on a small scale. These graphs were generated using the book (Evolutionary Dynamics: Exploring the Equations of Life) recommended to us by the supervisor. Theoretical Fixation Probability was created from this information.

- Research

Further research on real life graphs was being done this week mainly on finding graphs for testing. E-mails were sent to 31 different Universities in the UK asking for a map/graph of their computer networks, furthermore, some social network graphs of Twitch were found online for testing.

Graphics testing mainly focused on verifying if the algorithm is being shown on the visual representation or not and if the new node data function is working correctly this sprint. Algorithm testing focused on testing the working algorithm in different types of graphs with different amounts of nodes and edges.

Sprint 4

For the fourth sprint the supervisor told us to tidy up the code as fast as possible to give us enough time to do experiments using different graphs and this way we get all the data we need to get.

- Graphics

A node selection state is implemented to visually show which node is currently selected and to allow the program to focus interactions like the mutation interaction on that particular node. An interactive function to create nodes is implemented. An interactive function to add connections between the nodes on the graph is successfully implemented. The connections made are bidirectional and always have a weight of 1. Functionality to load graph data from text files in the form of adjacency matrices is implemented. Functionality to load some test graphs by entering their ID is also implemented.

- Development

Bug fixing and optimization of the code was the main objective of this sprint, they also started working with the graph representation of tissues that were found by the research team and how to generate them. Testing cases were written to make sure that the algorithm was working on the testing graphs. This ensured that there were no errors within the algorithm.

- Research

From this sprint the research team decided to start looking into cancer cells and how they are distributed. Responses from the different universities started to come in, all of them were negatives.

The testing done by the Graphics team is the same as in the last sprint. The development team testing was done on a small scale with graphs that have a small number of nodes and edges to check if the algorithm works correctly on the distribution graphs of cells.

Sprint 5

For our fifth sprint the main feedback we got was to start testing around this time, so the development team focused on finishing the last parts of the code and started asking for access to the supercomputer to make these tests faster.

- Graphics

The task of combining the Moran process simulation code to the visualisation begins this sprint. Accommodations are made for this task by creating buttons, labels and textboxes which would be used to navigate between simulated frames as well as set parameters like relative fitness. Simulate and Play functionality is decoupled and buttons are made for each. The idea here is that after loading a graph and setting the graph's initial state by mutating some nodes, the graph must be simulated fully before the animation can be played. Work on animation begins.

- Development

Further parameter adjustments, bug fixing and testing in progressively bigger tissue grafts have been done in this sprint. We also tuned some parts of the algorithm based on the testing cases we have written to make sure everything passed.

- Research

This sprint the research team focused on finding the threshold where an organ is considered lost to cancer to minimise the run time in the algorithm. More emails declining our graph request to the universities have been received.

The same type of testing was done on this sprint as it was done on the last sprint as the functions that were being developed are the same as last sprint.

Sprint 6

For the sixth sprint the development team got access to the supercomputers so testing was able to be made in a faster manner. The research team started to look into

new parameters that could be used to shorten the durations of simulations even further down.

- Graphics

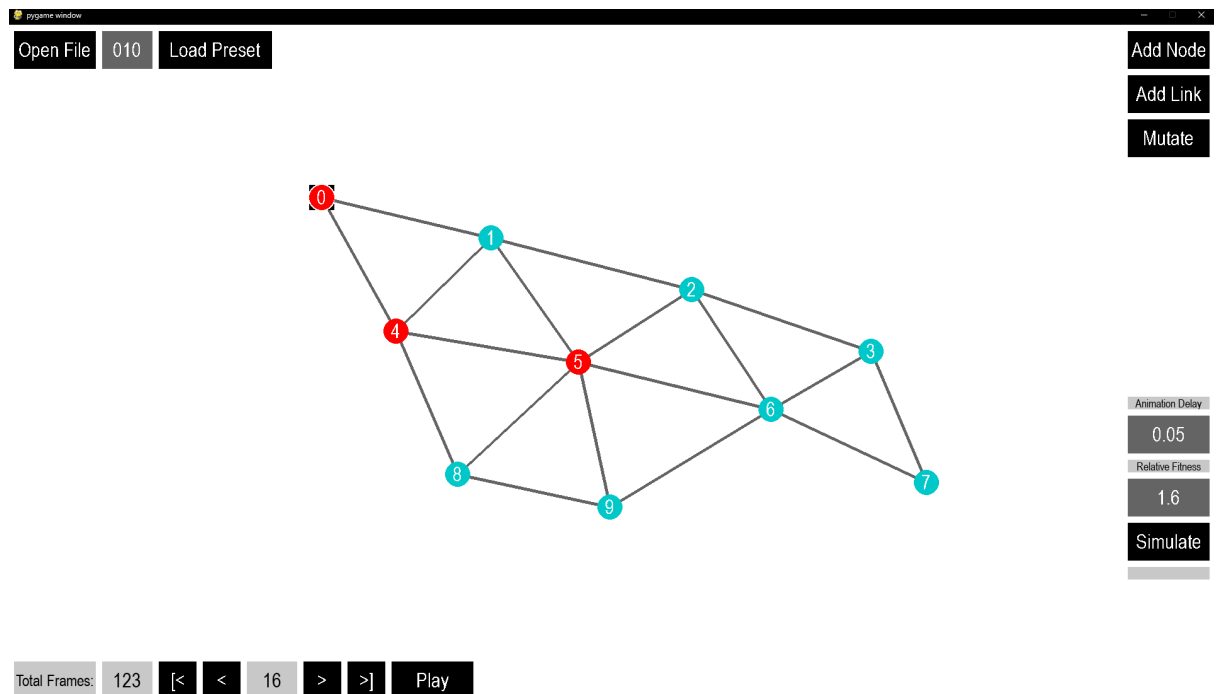


Figure 9. Interactive Visualisation Program

Animation successfully runs without need for multithreading or being locked in a 'for' loop. Tests on different graphs are run to ensure that the visualisation works as planned. Frame count and current frame labels are debugged as their extreme values seem to cause problems.

- Development

Progressively started to make test graphs bigger and created code to be able to run simulations in parallel to cut down the amount of time required for each test. We also started using the supercomputer within this sprint as the time it was taking to simulate was getting progressively longer.

- Research

Further research on cancer thresholds was done in this sprint and started writing the report document. More emails denying access to graphs of universities networks were received.

Started writing the report document this sprint. Graphics testing remains the same as last sprint. Development testing has seen a cut in time thanks to the code now being able to be run in parallel but tests remain the same just now with bigger graphs. The use of the university's supercomputer helps with doing multiple tests at the same time.

Sprint 7

For this last sprint feedback on the code was positive by the supervisor and with the time left before the deadline a focus on tidying up the last details on the code and testing was done.

- **Graphics**
Fixed some unhandled errors (program crashing because of trying to navigate through frames without simulating, initial graph state when simulating was incorrect, etc).
- **Development**
Started data collection with a modified version of the algorithm that uses the formula found in the data collection segment of this report. We also made sure that everything within the algorithm was running correctly and as intended so we didn't get any "Freak" results. Started work on the report.
- **Research**
Found articles presenting clinical data to calculate the probability of the spread of melanoma and breast cancer cells, correlating it with the size of the tumours and the other uses mathematical modelling to understand different aspects of the metastatic process.

For this last sprint tests were not a priority since the algorithm is working as intended so all teams focused on data analysis and data collection.

Key Code

In this section, we will explain the main part of the code that does the Moran process as well as some important to note parts.


```

def addOffspring(self, parent: int, offspring: int):
    self.relative_fitness[offspring] = self.relative_fitness[parent]

def select_parent(self) -> int:
    selected_parent = random.choices(list(self.graph.nodes()), weights=self.relative_fitness, k=1)[0]
    return selected_parent

def getOffspringLocation(self, parent: int) -> int:
    neighbors = self.getAdjacentNodes(parent)
    if len(neighbors) > 0:
        return random.choice(neighbors)
    else:
        return parent

```

Figure 10. Moran process parts

The three functions that are defined in the above image are the essential parts of the Moran process. Firstly, `select_parent` makes a random selection of one node in the population and returns the selected node. Secondly, `getOffspringLocation` finds all neighbours of the selected node and selects one at random to either “infect” or “cure”. And finally `addOffspring` simply “infects” or “cures” the selected neighbour by making it become the same as the selected node.

```

def simulation(self):
    self.reset()
    # run until healthy or infected win within given parameters
    while self.low_thresh < self.getMutantPopulation() < self.high_thresh:
        parent = self.select_parent()
        offspring = self.getOffspringLocation(parent)
        self.addOffspring(parent, offspring)
    # check if healthy or infected won
    return 1 if self.getMutantPopulation() > self.low_thresh else 0

```

Figure 11. Moran process code.

By putting these three functions together and establishing a higher threshold where if the infected population exceeds a certain amount the organ is considered lost to cancer and another lower threshold where we consider the organ “cured”. We decided to add those thresholds to make the algorithm less computationally expensive, both thresholds are in opposite extreme ends (For example 0.5% and 70% of the population), these thresholds were put in place to make the algorithm run shorter because when the population reaches such extreme amounts of infected or cured the probability of the opposite population winning is close to null.

```

def getAdjacentNodes(self, node: int) -> list:
    if(not self.isTissue):
        # check adjacency matrix and return index where entry is > 0
        neighbours = [index for index, n in enumerate(self.W[node]) if n > 0]
        # use same algorithm used to create tissue graph to find neighbors
    else:
        neighbours = []
        rEdge = False
        lEdge = False
        totalPopulation = self.getTotalPopulation()
        size = np.ceil(np.sqrt(totalPopulation))
        # if not in left edge
        if(node%totalPopulation > 0):
            neighbours.append(int(node-1))
        else:
            lEdge = True
        # if not in right edge
        if(node%size < size-1):
            if(node+1 < totalPopulation):
                neighbours.append(int(node+1))
        # is right edge
        else:
            rEdge = True
        # if not in bottom edge
        if(node+size < totalPopulation):
            neighbours.append(int(node+size))
            # if matrix continues and not in right edge
            if(node+size+1 < totalPopulation and not lEdge):
                neighbours.append(int(node+size+1))
        # if not in top edge
        if(node-size >= 0):
            neighbours.append(int(node - size))
            # if not in right edge
            if(not rEdge):
                neighbours.append(int(node - size + 1))
    return neighbours

```

Figure 12. Code for getting all neighbours of the selected node

It's also important to note that for the getAdjacentNodes function there had to be some if statements where we evaluate the position of each node in the overall graph to make sure we are not in the outline of the graph, this is done to avoid code crashes since if you try to find a node outside of the existent graph it would result in crashes and bugs.

```

def getGraph(self) -> nx.Graph:

    elif self.graph_type == GraphType.TISSUE:
        graph = nx.DiGraph()
        # generate honeycomb graph nodes
        for i in range(self.total_population):
            graph.add_node(i)
        # get number of columns and rows needed to represent graph as a 2d square matrix
        size = np.ceil(np.sqrt(self.total_population))
        print('size', size)
        ...

        add edges between current node and the node in these relatives positions (if available)
        [(0,1), (0,-1), (1,0), (-1,0), (1,1), (-1,-1)]
        To move in the y axis we add or subtract the calculated size of columns/rows to represent the graph in a matrix
        this generates a slanted-square-like honeycomb
        ...

        # iterate through each node number from 0
        for i in range(self.total_population):
            rEdge = False
            lEdge = False
            # if not in left edge |
            if(i%size > 0):
                graph.add_edge(i,i-1)
            else: lEdge = True
            # if not in right edge
            if(i%size < size-1):
                if(i+1 < self.total_population):
                    graph.add_edge(i,i+1)
            # is right edge
            else:
                rEdge = True
            # if not in bottom edge
            if(i+size < self.total_population):
                graph.add_edge(i,i+size)
                # if matrix continues and not in right edge
                if(i+size+1 < self.total_population and not lEdge):
                    graph.add_edge(i,i+size+1)
            # if not in top edge
            if(i-size >= 0):
                graph.add_edge(i,i-size)
                # if not in right edge
                if(not rEdge):
                    graph.add_edge(i,i-size+1)
        return graph

```

Figure 13. Graph generation code

Lastly, we have the tissue graph generator code. For this generator, we create all the population of nodes first and then evaluate if the current node is on the outline of the graph or not, if it is on the outline we determine which side of it the node is on and then create all edges accordingly, if the node is not on the outline we create all the edges accordingly.

Data Collection

Using the formula found in this article [15] for the spread probability of melanoma as the fixation probability, where N is the number of cells in a cluster and a and b are constant values, several tests were made to approximate the relative fitness that would most closely resemble the expected behaviour.

$$p = aN^b$$

By running these tests, we hope to find the relation between the relative fitness and the size of the cluster and then compare it to the relation between fixation probability and cluster size.

To find the potential relative fitness for each cluster the following algorithm is used: we start with a relative fitness of 1 and then add or subtract a variable step (starting at .49), which decreases every time the difference between the target and the calculated fixation probability changes sign, with a similar process to a binary search. Since the moran process is a random process, finding an exact relative fitness to satisfy the theoretical fixation probability is not a feasible goal, for this reason, we consider a 10% tolerance when determining a success.

The experimental fixation probability is found by calculating the proportion of fixation in 10,000 simulations. Larger simulations allow the results to have varying precision, with 10,000 runs we achieve 4 decimal precision, allowing us to compare to some of the theoretical values retrieved.

Experimental relative fitnesses and fixation probabilities were found for the following sizes: 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100. Since the fixation probability decreases as the size of the cluster increases, analysing this behaviour in larger clusters would require a more thorough calculation of experimental fixation probabilities, which would require more time or higher computing power, which is why tests stopped with 100 nodes. For the same reason, these tests were executed with the values obtained by Michaelson for melanoma spread, since it has the highest fixation probability, reducing the precision needed to correctly estimate a relative fitness. These constraints result in the following formula:

$$p = 0.027N^{-0.7836}$$

Data Analysis

After doing numerous tests, with varying amounts of nodes we found the following to be true: the higher the population (amount of nodes) the smaller the fixation probability, while the relative fitness increases. This behaviour can be related to the findings of J. S. Michaelson (2005) [15], where he indicates that the bigger the tumour the smaller the probability of metastasis.

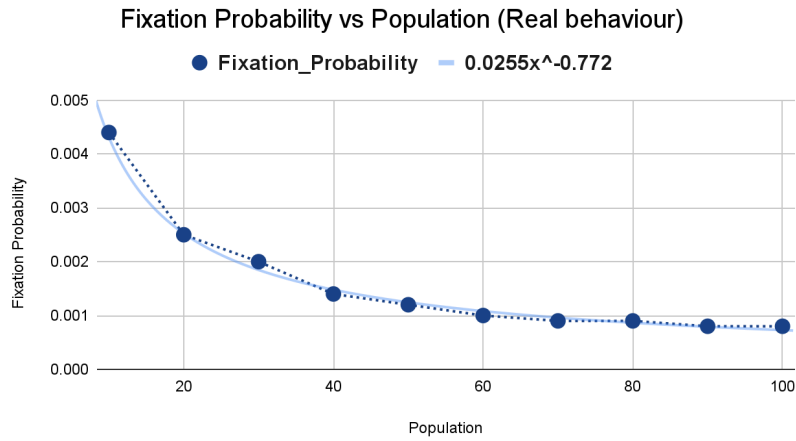


Figure 14. Fixation Probability vs Population graph (Given by our algorithm)

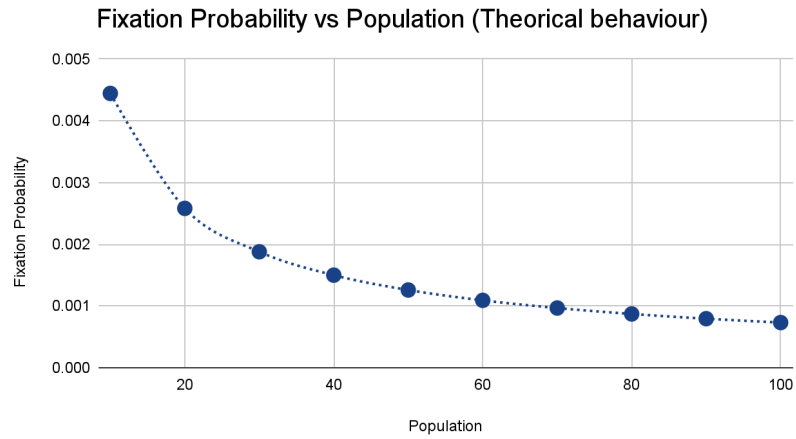


Figure 15. Fixation Probability vs Population graph

As we notice in Figures 14 and 15 it is worth mentioning that we compare the theoretical behaviour vs the real one and notice that the graphs look almost identical, so we can infer that the simulation is giving reliable results.

Starting from the idea that we are obtaining reliable results, the relative fitness that the algorithm is providing can be understood as the probability of a cell with metastatic potential reproducing as opposed to a cell with no metastatic potential.

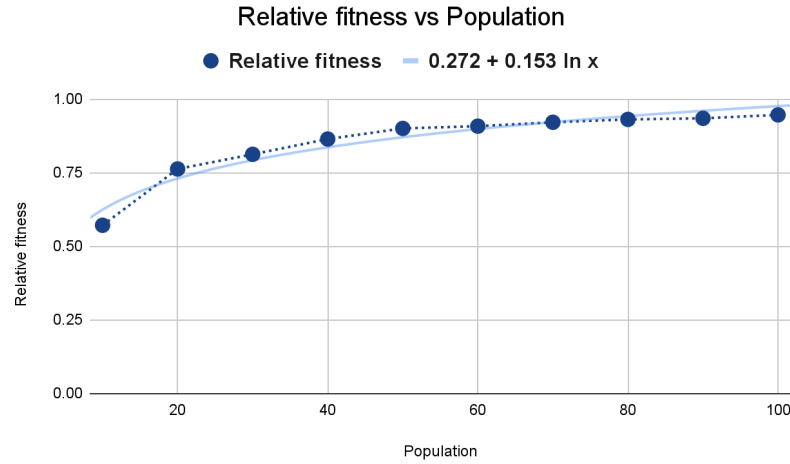


Figure 16. Relative fitness vs Population graph.

With these results, we can represent the behaviour found with the following two equations:

$$p = aN^b \quad (1)$$

$$f = c + d (\ln N) \quad (2)$$

where N represents the size of the tumour.

Due to the similarity of the original equation with the one obtained from the graphs, we discovered that the relative fitness behaviour can be modelled through a power series equation depending on the population size, just like the fixation probability.

Since both equations are dependent on N and each of them has its independent constants, we can find an equation that links the relative fitness to the fixation probability without the need for N . To begin we have to isolate N from one of the equations, we'll be doing this with (1).

Starting with (1)

$$p = aN^b$$

$$\ln(p) = \ln(a) + b * \ln(N)$$

$$\ln(p) - \ln(a) = b * \ln(N)$$

$$\frac{\ln(p)-\ln(a)}{b} = \ln(N)$$

$$N = e^{\frac{\ln(p)-\ln(a)}{b}}$$

$$N = e^{\frac{\ln(\frac{p}{a})}{b}} \quad (3)$$

Now that we have N isolated from the rest of the equation we can use this to substitute N in the other equation to remove the need of the population on it.

Starting with (2)

$$f = c + d (\ln N)$$

And from (3)

$$N = e^{\frac{\ln(\frac{p}{a})}{b}}$$

$$f = c + d * \ln(e^{\frac{\ln(\frac{p}{a})}{b}})$$

$$f = c + d(\frac{\ln \frac{p}{a}}{b})$$

$$f = c + \frac{d}{b} * \ln \frac{p}{a} \quad (4)$$

Now we have an equation that allows us to estimate the relative fitness given the fixation probability for this specific type of cancer (melanoma). Given the formula obtained by Michaelson [15] we can get the fixation probability for any size of melanoma. Using this probability we can then use the formula obtained with testing (4) to get an estimate of the relative fitness of metastatic cells. Since this is an estimation using a logarithmic regression, the best results will be obtained with fixation probabilities near the ones used to obtain this formula.

While current formulas only work for melanoma in a limited range of sizes, the process followed to obtain these results can be replicated using other types of cancer and sizes to obtain a formula to model their behaviour.

Conclusions

The final product offers an effective implementation of the Moran process using tools like the NetworkX library for graph generation and manipulation and the Pygame library for the visual representation of said graphs. Even though our initial objective was to test social networks and computer networks in addition to cancer cells, results show a good implementation of the Moran Process on cancer cells. Further testing could be done on different types of networks, with their corresponding research and data analysis.

The data obtained through testing allowed the formulation of a model for estimating the relative fitness of metastatic cells given the fixation probability of cancer metastatic spread for melanoma in small tumour clusters. This is an important breakthrough, as the process used can be easily replicated for different types of cancer and different tumour sizes which would help predict and model the behaviour of individual cells inside a tumour to better understand how they spread. Moreover, the same procedure can be used to estimate fixation probabilities given relative fitness in case future studies find them experimentally.

References

- [1] Software Requirements Specification, taken from the software requirements document from the last assignment.
- [2] Ann Goldberg, L., Lapinskas, J., & Richerby, D. (2019). Phase transitions of the moran process and algorithmic consequences. *Random Structures & Algorithms*, 56(3), 597–647. <https://doi.org/10.1002/rsa.20890>
- [3] de Souza, E. P., Ferreira, E. M., & Neves, A. G. (2018). Fixation probabilities for the Moran process in evolutionary games with two strategies: Graph Shapes and large population asymptotics. *Journal of Mathematical Biology*, 78(4), 1033–1065. <https://doi.org/10.1007/s00285-018-1300-4>
- [4] Whigham, P.A., Dick, G. (2006). Evolutionary Dynamics on Graphs: The Moran Process. In: Wang, TD., et al. Simulated Evolution and Learning. SEAL 2006. Lecture Notes in Computer Science, vol 4247. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11903697_1
- [5] Hauert, C. (2023, October). *Moran process*. EvoLudo. https://wiki.evoludo.org/index.php?title=Moran_process
- [6] Official NetworkX development team, Software for Complex Networks. <https://networkx.org/documentation/stable/index.html>
- [7] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. *Nature* 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2.
- [8] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," in *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90-95, May-June 2007, doi: 10.1109/MCSE.2007.55.
- [9] Rahul Shah, "Overview of TQDM: Python Progress Bar Library". <https://www.analyticsvidhya.com/blog/2021/05/how-to-use-progress-bars-in-python/>
- [10] Piva, G. G., Ribeiro, F. L., & Mata, A. S. (2021). Networks with growth and preferential attachment: Modelling and applications. *Journal of Complex Networks*, 9(1). <https://doi.org/10.1093/comnet/cnab008>
- [11] Ribeiro, D. (2023, July 3). *Exploring the barabási–albert network model: Unveiling the secrets of Complex Networks*. Medium. <https://medium.com/data-science-as-a-better-idea/exploring-the-barab%C3%A1l>

[si-albert-network-model-unveiling-the-secrets-of-complex-networks-314fd38a78cd](https://doi.org/10.1007/s41109-019-0152-1)

[12] Bertotti, M.L., Modanese, G. The configuration model for Barabasi-Albert networks. Appl Netw Sci 4, 32 (2019). <https://doi.org/10.1007/s41109-019-0152-1>

[13] Kaoru Sugimura, Shuji Ishihara; The mechanical anisotropy in a tissue promotes ordering in hexagonal cell packing. Development 1 October 2013; 140 (19): 4091–4101. doi: <https://doi.org/10.1242/dev.094060>

[14] ¿Qué es la metástasis?. Cancer.Net. (2023, May 30). <https://www.cancer.net/es/desplazarse-por-atenci%C3%B3n-del-c%C3%A1ncer/conceptos-b%C3%A1sicos-sobre-el-c%C3%A1ncer/%C2%BFqu%C3%A9-es-la-met%C3%A1stasis>

[15] Michaelson, J.S. et al. (2005) Spread of human cancer cells occurs with probabilities indicative of a nongenetic mechanism, Nature News. Available at: <https://www.nature.com/articles/6602848>

[16] Scott, J. G., Gerlee, P., Basanta, D., Fletcher, A. G., Maini, P. K., & Anderson, A. R. A. (2013). Mathematical modelling of the metastatic process. Experimental Metastasis: Modeling and Analysis, 189–208. https://doi.org/10.1007/978-94-007-7835-1_9

