33rd European Modeling & Simulation Symposium
18th International Multidisciplinary Modeling & Simulation Multiconference

ISSN 2724-0029 ISBN 978-88-85741-57-7 © 2021 The Authors.
doi: 10.46354/i3m.2021.emss.013

PanaXea: A Framework for the Development and Parametrization of Agent-Based Models

Dario Panada[1,*] and Bijan Parsia[1]

[1]The University of Manchester, Oxford Road, Manchester, M13 9PL, United Kingdom

[*]Corresponding author. Email address: dario.panada@manchester.ac.uk

Abstract

This paper presents a framework for the development of agent-based models aimed at facilitating parameter space exploration by means of established parameter tuning strategies. Such simulations often require a high number of parameters to account for the complexity of the underlying processes. It is often the case that parameter values are not known, or that when they are known measurements are reported with a large margin of error. Despite this, publications in the field often rely on single values rather than considering larger search spaces for their parameters. It is therefore uncertain whether results obtained are an artifact of a very specific combination of parameter values or truly representative of the underlying phenomenon. Our solution is applicable to any sort of agent-based model and can easily be expanded to incorporate further parameter tuning algorithms. We then tested our framework by reproducing an existing model of angiogenesis and exploring changes in simulation results across parameter values. Our case-study results suggest the aforementioned model is highly sensitive to the choice of parameter values, with even small changes in these causing significant divergences in results.

Keywords: ABM Framework; Model Parametrization; Cancer Modelling

1. Introduction

Cancer research has been historically performed via *in-vivo* and *in-vitro* experiments. The former indicates experiments run on live animals or patients, whereas the latter refers to experiments performed in a laboratory environment such as cell cultures in a petri dish. While contributions from this research has been invaluable, in-vivo and in-vitro approaches do present several weaknesses. Some of these have been identified and discussed by *Kam et al.* (Kam et al. (2012)), we hereby summarise their main points.

With regards to in-vivo experiments, the authors highlight the difficulty in controlling the experimental environment. Especially in complex organisms, a number of factors beyond the control of the analysts impact the development of the tumour mass. They make it difficult to ascertain which phenomenon or factor is the driving force behind an observed behaviour of cancer cells. With regards to in-vitro experiments, while these allow a better control of the environment it becomes hard to replicate the complexity of in-vivo environments. As an example it is particularly difficult, if not impossible, to replicate the process of Angionesis in an in-vitro environment. Angiogenesis refers to the process whereby cancer cells stimulate the formation of new blood vessels, and is understood to be central to cancer growth.

A possible solution to address the limitation of in-vivo and in-vitro approaches is represented by *in-silico* models. These are artificial models simulating the development of biological systems, offering researchers the possibility of incorporating

processes which cannot be replicated in-vitro while at the same time allowing for a level of control of the experimental setup which is not achievable in-vivo.

In addition to research and hypothesis testing, in-silico models could also be used to develop personalized patient models. This would enable clinicians to predict whether a certain therapy would be succesful on a given patient, or whether it would present particular side-effects. As such, it would reduce the risk of patients undergoing ineffective or even damaging therapies reducing operational costs and increasing odds of success.

The strenghts of in-silico models are also discussed further in depth by *Saeidnia et al.* (Soodabeh Saeidnia (2013)), *Agur et al.* (Agur et al. (2016)) and *Ekins et al.* (Ekins et al. (2007)).

A challenge in the development of in-silico models is their parametrisation. Because of the moltitude of parameters needed, and because these need as far as possible to be informed by medical literature, it is often the case that multiple values are drawn from different studies and publications or estimated based on various other studies. This is the case, for example, in *McDougall et al.* (McDougall et al. (2006)) where the probability of vessels sprouting as a function of Tumour Angiogenic Factors (TAF) concentration is estimated as a sigmoid relation. This is also seen in *Anderson et al.* (Anderson and Chaplain (1998)), where due to the difficulty in obtaining exact measurements it was assumed that epithelial cells could respond to any non-zero TAF concentration. Furthermore, this is also the case in *Kather et al.* (Kather et al. (2017)) where assumptions have to be made about parameter values.

A limitation shared by many of the existing frameworks is a lack of support for parameter space exploration and no utility to easily deploy simulation instances to cloud services. (Eg: Amazon Web Services, Google Cloud, proprietary infrastructures.) As part of our investigation we developed a general-purpose framework for the development of hybrid agent-based models: PanaXea.

PanaXea is developed in Python, one of the most commonly adopted languages in the scientific community, and leverages commonly used open-source libraries for data processing and analysis. This contrasts with other popular frameworks such as NetLogo (Wilensky (1999)) or GAMA (Taillandier et al. (2019)) which use proprietary languages and may therefore limit the developers in the toolkits and libraries which they may import. Popular frameworks exist which are written in non-proprietary languages, including Repast (North et al. (2013)) and MASON (Luke et al. (2005)). The main weakness of these, in our opinion is that they use Java rather than Python. This is not a matter of personal preference: Python is increasingly becoming the language of choice for the scientific community, with experts from domains other than Computer Science often being fluent developers and a growing ecosystem of tools for data analysis and visualization. (Eg: IPython Notebooks) It is also worth noting Python has a gentler learning curve than other languages, with syntax often resembling natural English and implementation not requiring the developers to master more in-depth concepts such as those related to Object-Oriented Programming.

In fact some Python framework for agent-based modelling do exist, among which we would like to acknowledge MESA Masad and Kazil (2015). Many of our design choices align to theirs, but key improvements made on the implementation include us supporting three-dimensional environments. This is fundamental, for example, to accurately model biological phenomena as highlighted in multiple publications. (Sinek et al. (2004); Lv et al. (2017); Pickl and Ries (2009)) We further support *Numerical Grids* which, overlaid to model environments, allow to store numerical properties associated to each position. (Eg: Nutrient concentration at a particular position.) Finally, we further implement each epoch as a three part process. Thus allowing, within the same epoc, all agents to complete a set of steps before a further set of agent actions is initiated. The value of this is further explained in section 2.1.1.

The Panaxea framework offers out-of-the-box tools to easily instantiate a model environment, define agents, populate environments with agents and syncronize the execution of all agents to a common schedule. Reproducibility is one of the key features of the framework. Panaxea enforces an explicit definition of starting conditions such that the same experiment can be reproduced multiple times on different hosts. It also offers features to monitor the evolution of the model and easily export observations for post-execution analysis.

Further, PanaXea offers tools to automatically perform and evaluate parameter space exploration (Eg: Grid Search, Random Search) reporting on which parameter values lead to best model performance. Some of these features are already present in other frameworks, such as *OpenMole*. (Romain Reuillon (2013)) Especially compared to this latter, our minimalist toolkit for parameter space exploration is considerably smaller in terms of size (355mb OpenMole vs 7kb our toolkit). It is worth noting that OpenMole offers many additional functionalities, but with regards to our research objectives we would not have been

leveraging many of them while paying the cost of increased project size. Further, OpenMole is written in Java, whereas our framework in Python. Having developed our parameter space exploration toolkit in Python we avoid requiring developers to setup two separate environments.

The framework further makes deployment to cloud infrastructures seamless. It is distributed with a Docker image setup to host the framework as well as tools to monitor simulations running on different cloud nodes and reconcile and aggregate results. While every effort was obviously made for the framework to be developed in a clean, efficient manner no assumption of users being proficient in Python is made. In the interest of the framework being approachable by as wide an audience as possible, knowledge of core concepts of programming and the basic syntax of Python should be sufficient to develop complex models.

The rest of the paper is structured as follows. We present an overview of our framework, its components and code extracts of its usage in section 2. Code extracts in particular refer to the case study we provide in section 3, where we implement a biological model of angiogenesis derived from existing literature and make use of our parameter space search toolkit to establish its sensitivity. Finally, we present our conclusions and final remarks in section 4.

2. Methods

2.1. PanaXea - Our Framework

The model was implemented using *PanaXea* [1], an open-source framework we developed. This is written in Python and aimed at facilitating and promoting the development of agent-based models. The framework has a modular approach, exposing tools which may be adapted to suit individual needs of developers. We enumerate the main components with a brief explanation of how they may be used. A class diagram detailing our implementation may be found in the appendix. (See figure 3.)

2.1.1. *Steppables*

Steppables refer to entities which are progressed as the simulation executes. That is, whose inner logic is executed and whose state is updated once per epoch. There are two main classes of steppables: Agents (see 2.1.2) and Helpers (see 2.1.4).

Each steppable allows to define behaviors that will be executed during the *prologue*, the *main* and the *epilogue* stages of an epoch. During an epoch, all steppables' *prologue* methods will be executed before *main* methods begin. And all *main* methods will be executed before *epilogue* methods begin. This allows, for example, for all agents to collect information about their environment without any other agent's action having changed environmental properties. That is, agents would collect information during *prologue*, and then execute actions in *main* or *epilogue*.

2.1.2. *Agents*

This consists of templates to instantiate agents. Agents are the main actors of a simulation, and may represent any self-contained entity such as a person, a biological cell, etc. Agents have a position in an environment and one or more properties. Further, agents also encapsulate the logic each member of such class would execute at each epoch and which describes its behavior. In our simulation, we implemented two classes of endothelial cell agents: Tip Cells and Trunk Cells. Following, an extract of how Tip Cells are implemented in PanaXea:

```
class TipCell(Agent):

def __init__(self):
    super(TipCell, self).__init__()

def step_main(self, model):
    # Get current position
    current_position = \
        self.environment_positions[
        "agent_environment"]

    # p0 represents the probability of
    # the cell remaining stationary
    p0 = self.__calculate_p0(
        model,
        current_position
    )
    # p1-p4 represent probabilities
    # of the cell moving to any
    # of the 4 adjacent positions
    p1, p2, p3, p4 = self.__calculate_p1to4(
        model,
        current_position
    )

    next_position = self.__get_next_position(
        p0,
        p1,
        p2,
        p3,
        p4,
        current_position
    )

    # If the cell has moved at
    # least 180 positions along the x-axis
    # we assume it has
```

1 https://github.com/DarioPanada/panaxea

```
            # traversed the section of tissue
            if next_position[0] > 180:
                model.exit = True

            # Add a trunk cell to the old position
            t = TrunkCell()
            t.add_agent_to_grid(
                "agent_environment",
                current_position,
                model)

            # Move itself to the new position
            self.move_agent(
                "agent_environment",
                next_position,
                model)
```

The example above illustrates the behavior of TipCell agents. Namely, that at each epoch they make a decision as to whether they will move to an adjacent position or remain stationary. Where they decide to move, a TrunkCell is instantiated at their previous position. Developers are able to leverage common model functionalities provided by the framework (Eg: *add_agent_to_grid*) but also to program their own ad-hoc behavior specific to their simulation.

2.1.3. *Environments*

This consists of classes to instantiate 2D and 3D Cartesian grids. Grids may either be numerical, storing a value at each position, or object grids, storing one or more agents. As an example, in our implementation numerical grids were used to store TAF and Fibronectin concentrations whereas object grids were used to store endothelial cells. Environments also provide functionality to move agents between positions and to find adjacent neighbours with the most or fewest agents in them. A code extract is provided below:

```
    xsize = ysize = 200

    NumericalGrid2D("taf_environment", xsize,
    ysize, model)
    NumericalGrid2D("fib_environment", xsize,
    ysize, model)

    ObjectGrid2D("agent_environment", xsize,
    ysize, model)
```

This instantiates two 2D Numerical Grids and one 2D ObjectGrid. It also declares the environment size, names each environment so it can be referenced n the future and binds it to the model object so it becomes part of the simulation.

2.1.4. *Helpers*

Helper classes interface with the simulation and are executed once per epoch as agents. However, they are not "actors" in the simulation but rather perform auxiliary function such as recording population size, changing environment properties, etc. In our example, we made use of helpers to update TAF and Fibronectin concentrations at each epoch. A Helper may be defined as:

```
    class ConcentrationsHelper(Helper):
        """
        At the start of each epoch,
        updates TAF and Fibronectin
        concentrations
        """

        def __init__(self):
            super(ConcentrationsHelper, self)
            .__init__()

        def step_prologue(self, model):
            self.__update_taf(model)
            self.__update_fibronectin(model)

```

And added to the schedule so they become part of the simulation as:

```
model.schedule.helpers.append(ConcentrationsHelper())
```

In this specific case, the helper is responsible for updating soluble concentrations at each environment position before the start of each epoch. The Helper is not a member of the simulation per se, but its work is nonetheless essential.

2.1.5. *The Model*

The model is the parent object of the simulation and holds all other elements such as environments, agents, the schedule, etc. Further, the model may also hold a set of simulation properties. In our simulation, model properties included all values for parameters used in the equations.

2.1.6. *The Schedule*

The schedule is a singleton class which holds all steppables: agents and helpers, who are part of the simulation. Each epoch consists in traversing the schedule and executing the logic of each steppable. All steppables must be registered with the schedule to ensure that they are executed as part of the simulation.

2.2. Parameter Space Exploration

PanaXea emphasizes parameter space exploration and sensitivity testing by enforcing a separation between a model and the set of parameter values which are used for a specific simulation. In the model, the user would define all aspects of the simulation such as the strategy whereby environment properties are set, agent behaviours, etc. These would include

the use of variables, whose value would however be assigned at runtime as they are loaded from an external configuration file. As an example, in our model the equations to generate probabilities for a tip cell to move to a position made use of a high number of variables. The equations were defined in the model and common to all simulations, but the actual value of the equations' variables would be different for each run.

PanaXea offers a script to generate experiments. Given a set of experiment parameters and value generation strategies, this produces a configuration file in the format of comma-separate values (csv) where each column corresponds to a parameter and each row to a simulation. Insofar, we support Grid Search and Random Search as parameter generation strategies. However, by design it is possible to add further algorithms with minimal changes to the existing codebase. The definition of an external configuration file also allows for experimental reproducibility.

For example, the following definition:

```
parameters = [
    {
        "name": "gamma",
        "search_strategy": "grid_search",
        "range": [0, 10, 2]
    },
    {
        "name": "alpha",
        "search_strategy": "random_search",
        "mean": 1,
        "stDev": 0.1
    },
    {
        "name": "epsilon",
        "search_strategy": "constant",
        "value": 5
    }

]
```

May result in the experiment definitions being generated reported in table 1.

experiment	gamma	alpha	epsilon
1	0	1.01	5
2	2	0.98	5
3	4	0.97	5
4	6	1.03	5
5	8	0.95	5
6	10	1.05	5

Table 1. Sample experiment definition generation with PanaXea. Parameter values for *gamma* are generated by grid search on range 0 to 10, parameter values for *alpha* are generated by random search on a distribution with μ=1 and σ = 0.1 and parameter values for *epsilon* are kept constant at 5.

2.3. Analysis of Results

PanaXea decouples simulation state logging during execution from model serialization. Tools are available to decide, at each epoch, which properties of the model the developers wish to record. This could be as simple as recording the entire state of the model (All environments, agents and their properties...) or more fine-tuned. For example, the developers may wish to only record a subset of agent properties from certain agent classes. As an example, a simulation of cellular biology may feature hundreds of gene expression rates across multiple types of cells. But, the developers may only be interested in expression rates of specific genes for certain types of cells. Recording everything at each epoch would lead to significant memory usage for no benefit.

Simulation results may then be serialized in any desirable format. By default PanaXea supports python "pickles" (object serializations). This is the most versatile and can be used to represent any sort of information. But with minimal effort information could also be mapped to any format. (Eg: Tabular: CSV, SQL, Semi-structured: XML, JSON...)

2.4. Extensibility and Integration

The framework is unprejudiced by design. Rather than enforcing modelling conventions it exposes to developers a set of flexible tools that may be deployed and arranged as best suits the problem at hand. Emphasis are on such tools imposing as little overhead as possible, thus freeing computational resources for the actual simulations. Wherever possible the use of established libraries is encouraged, such as Numpy for numerical calculations, Pandas for dataset management and Matplotlib for visualization.

The framework is designed to integrate into pipelines which include separate stages for experiment generation, execution and analysis. Because of this, open and accepted standards such as csv for experiment definition and python pickle files to store experiment outputs are encouraged. This latter point is particularly important, as it allows for results to be loaded into a developer's analysis software of choice, thus not constraining them to a preset showcase of tools distributed with our framework.

2.5. Cloud Deployment

Computational requirements of agent-based models often exceed the capabilities of normal home or office computers, and the amount of time required for these to complete may make this undesirable regardless.

PanaXea has been designed with cloud deploy-

ment in mind. To that end, the framework is distributed as a python package which may be installed via Pip (Python Package Index). A docker image is also made available which allows for simple execution of any PanaXea project within it.

Finally, a toolkit for Amazon Web Services (AWS) is provided. This provides scripts to write experiment files to AWS Queues as messages, where each experiment may be individually downloaded and executed by EC2 instances (cloud computing instances) part of a designated fleet. Upon completion, the result pickle and any other generated output are automatically uploaded to an S3 Bucket (cloud storage) from where they may be downloaded.

The rationale for selecting AWS as a technology is that our research is supported by AWS Cloud Credits for Research, but the scripts could be easily adapted to interface with other major providers such as Google Cloud or Microsoft Azure. Where a proprietary setup is preferred, we recommend an Apache Kafka instance to act as a queue service and a Kubernetes system to manage a cluster of Docker images where simulations would run. As a replacement to S3 Buckets, any storage server capable of accepting incoming SCP requests should fulfill the task.

3. Case Study: Results & Discussion

We considered the discrete implementation of the model of angiogenesis proposed by *Anderson et al.* (Anderson and Chaplain (1998)). Such a model simulates the development of a blood vessel network in a tissue environment in response to chemotactic stimuli promoted by a growing tumour mass. Figure 1 illustrates a sample evolution of the model as reported by the authors. To briefly summarize, the environment is discretized as a two-dimensional Cartesian grid. Each position has a concentration of tumour angiogenic factor (TAF), which creates a chemotactic gradient, and of Fibronectin, which creates an haptotactic gradient. The model is advanced by discrete steps caled epochs. At each epoch the probability of each tip endothelial cell remaining stationary or moving in one of four directions (up, down, left or right) is calculated.

As an endothelial tip cell migrates to an adjacent position, an endothelial trunk cell replaces it at the original one, thus simulating the process of vessel elongation. The interplay between TAF and Fibronectin concentrations determines probability values, with endothelial cells privileging positions with higher concentrations of both.

The model is well-suited to our research aims because of its detailed reporting of all required

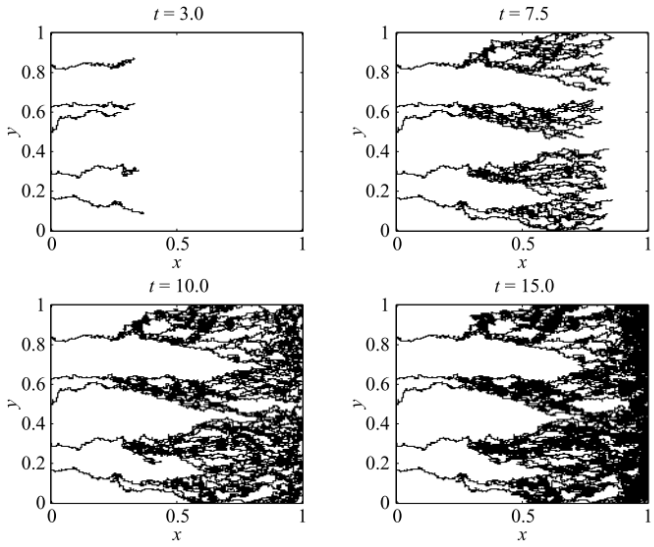

Figure 1. Development of existing blood vessels in time in response to chemotactic stimuli by means of elongation and sprouting, as reported by *Anderson et al.* (Anderson and Chaplain (1998)). Starting with 5 tip cells at $x = 0$, elongation is observed towards a tumour positioned at $x = 1$. As the vessels approach the tumour, sprouting is also observed.

parameters as well as their values, alongside all equations used and a thorough explanation of its mechanics. This allowed us to re-implement it with confidence and proceed to exploring the effect that changing parameter values had on the progression of the simulation.

The equations which govern endothelial movement are parametrized with 14 variables. Values for all of these are provided, but for many of them neither their biological significance nor the process by which the value was obtained is discussed. In several instances it was reported that trial and error was used to find values which allowed the simulation to behave as expected, although specific strategies and ranges tested were not disclosed.

A simplified version of the model was used which was seeded with a single endothelial tip cell at position $x = 50$ and did not account for sprouting. Overall, the environment grid had a size of 200 x 200 units. The rationale was that this would simplify considerably the interpretation of the results while still remaining generalizable to more complex implementations. Initial TAF and Fibronectin concentrations were calculated using a logistic function along the x-axis of the environment. Hence, TAF values were highest at $x = 200$ and lowest at $x = 0$, and vice-versa for fibronectin concentrations.

The fitness of each simulation was calculated as inversely proportional to the number of epochs it took for the tip endothelial cell to traverse the environment

and reach the tumour. Each simulation was repeated 10 times, with the fitness reported being the average of the runs. The model was allowed to run for a maximum of 500 epochs, after which the simulation would have been considered failed.

We studied the effect of changing values for parameters x_0, the chemotactic coefficient, and k, a coefficient present in all equations used to determine movement probabilities. We initially made use of grid search Bergstra et al. (2011). For k, which in the original publication was set to 0.75, we tested values from 0 to 10 in steps of 0.2. For x_0, which was originally set to 2,600, we tested values from 500 to 7,000 in steps of 500. The two parameters were explored independently. When x_0 parameters were explored the value of k remained constant at what originally used by the authors, and vice-versa. While this case-study demonstrates parameter space exploration by means of grid search, similar experiments using random search or other techniques would also have been supported by the framework. Results are shown in figure 2.

Notably, what can be observed is that is that the model is not sensitive to x_0, and only sensitive to k when the value for this is below a certain threshold. x_0 is the chemotactic coefficient, its assigned value in the model we are reproducing is 2,600, it represents the impact the TAF gradient has on vessel elongation. The higher this value, the more endothelial cells should privilege regions of higher TAF concentration to those of higher VEGF concentration. In practice, this means that higher x_0 values should induce the vessels to elongate away from their origin and towards the tumour, whereas lower values should promote the opposite. We would therefore expect a positive correlation with fitness, but this is not observed. What this suggests is that, in this model, the implementation of the chemotactic coefficient does not reflect its biological role.

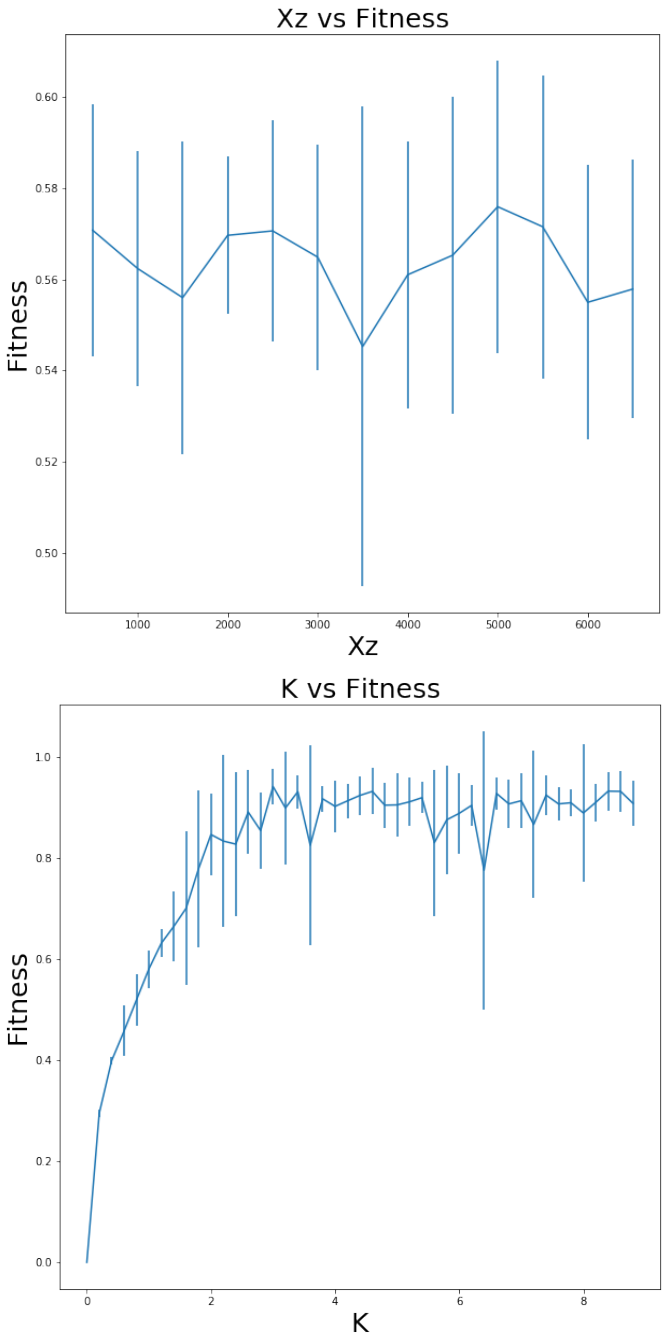

Figure 2. Grid search on parameters x_0 and k. x_0 was sampled from 500 to 7,000 in steps of 500, k was sampled from 0 to 10 in steps of 0.2. Simulations were repeated 10 times per parameter value, with average values reported. Error bars show standard deviation. At each occurrence, simulations which did not complete within 500 epochs were excluded. With the exception of $k = 0$, more than 2 in 10 simulations were never excluded.

The role of the parameter k is not fully explained. However, it is present as a coefficient in all terms of every equation used to calculate movement direction probabilities. In the original model it is given a value of 0.75. As can be seen, this leads to sub-optimal model per-

formance, with the highest fitness occurring for $k > 1$. Accounting for stochasticity, increasing k beyond 1 does not lead to significant alterations in performance.

4. Conclusion

We have presented a framework to facilitate the development and parametrization of agent-based models. Our framework, PanaXea, offers a minimalist yet complete toolkit to rapidly develop agent-based models in Python. It allows for easy integration with libraries commonly used for data analysis as well as deployment to the AWS cloud.

Among the design principles of our implementation we have privileged cross-domain participation. Our framework aims at being accessible to experts and researchers who do not have a strong computer science background. In fact, even an elementary knowledge of Python would be sufficient to contribute to the development of a PanaXea model.

Parametrization is a common weakness in the field of modelling, with simulations often relying on a single set of hand-picked values. Such an approach means that models are not tested for stability against small changes to parameter values, leaving it unknown whether results are truly representative of the phenomena they simulate or an artifact of an exact parameter value combination. Our framework facilitates parameter space exploration and sensitivity testing by leveraging established parameter tuning algorithms and allowing to easily adapt them to any sort of agent-based model.

We have then reproduced a highly dimensional existing model which simulated the process of angiogenesis. Parameter space exploration on 2 of its parameters highlighted how one of them (x_0) did not have any correlation to fitness and another (k) did correlate over a specific range ($0 \leq k \leq 1$) but did not over higher values. We further noted how the value of k used in the original publication did not achieve maximum fitness.

Our results emphasize the importance of parameter space exploration and sensitivity testing. Biological phenomena are driven by the interplay between numerous factors, which invariably leads to highly dimensional models. To avoid unnecessary complexity, it is essential that all parameters have a clear impact and exercise a defined role over the simulation. Similarly, parameters where even small changes in their value causes significant diverges in performance should be investigated to ensure that such bifurcations correctly trace their biological counterparts.

For in-silico models to provide a reliable complement to laboratory research it is essential that any result is accompanied by a discussion concerning its parametrization, including the role of each parameter and the model's stability with regards to it. Towards this, our framework exposes tools to rapidly define and execute parameter space search strategies. It is also the case that parameter space exploration and sensitivity testing may be computationally expensive. To address that, we provide a strategy and tools to deploy simulation on clusters of cloud resources and easily distribute experiments and aggregate results.

Overall, our research has highlighted the risks of poor parametrization techniques. Namely, that these risk adding unnecessary complexity to a model while undermining the reliability of its results. By accompanying this discussion with a practical solution in the form of an open-source and freely available framework, we hope to contribute to a drive towards more rigorous paradigms relating to parameter space exploration and sensitivity testing of agent-based models.

References

Agur, Z., Halevi-tobias, K., Kogan, Y., and Shlagman, O. (2016). Employing dynamical computational models for personalizing cancer immunotherapy. *Expert Opinion on Biological Therapy*, 2598(August).

Anderson, a. R. and Chaplain, M. a. (1998). Continuous and discrete mathematical models of tumor-induced angiogenesis. *Bulletin of mathematical biology*, 60(5):857–899.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for Hyper-Parameter Optimization. *Advances in Neural Information Processing Systems (NIPS)*, pages 2546–2554.

Ekins, S., Mestres, J., and Testa, B. (2007). In silico pharmacology for drug discovery : applications to targets and beyond. (December 2006):21–37.

Kam, Y., Rejniak, K. A., and Anderson, A. R. (2012). Cellular modeling of cancer invasion: Integration of in silico and in vitro approaches. *Journal of Cellular Physiology*, 227(2):431–438.

Kather, J. N., Poleszczuk, J., Suarez-Carmona, M., Krisam, J., Charoentong, P., Valous, N. A., Weis, C. A., Tavernar, L., Leiss, F., Herpel, E., Klupp, F., Ulrich, A., Schneider, M., Marx, A., Jäger, D., and Halama, N. (2017). In silico modeling of immunotherapy and stroma-targeting therapies in human colorectal cancer. *Cancer Research*, 77(22):6442–6452.

Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., and Balan, G. (2005). Mason: A multiagent simulation environment. *Simulation*, 81(7):517–527.

Lv, D., Hu, Z., Lu, L., Lu, H., and Xu, X. (2017). Three-dimensional cell culture: A powerful tool in tumor

research and drug discovery.

Masad, D. and Kazil, J. (2015). Mesa: An agent-based modeling framework. pages 51–58.

McDougall, S. R., Anderson, A. R. A., and Chaplain, M. A. J. (2006). Mathematical modelling of dynamic adaptive tumour-induced angiogenesis: Clinical implications and therapeutic targeting strategies. *Journal of Theoretical Biology*, 241(3):564–589.

North, M. J., Collier, N. T., Ozik, J., Tatara, E. R., Macal, C. M., Bragen, M., and Sydelko, P. (2013). Complex adaptive systems modeling with repast simphony. *Complex Adaptive Systems Modeling*, 1(1).

Pickl, M. and Ries, C. H. (2009). Comparison of 3D and 2D tumor models reveals enhanced HER2 activation in 3D associated with an increased response to trastuzumab. *Oncogene*, 28(3):461–468.

Romain Reuillon, Mathieu Leclaire, S. R.-C. (2013). Openmole, a workflow engine specifically tailored for the distributed exploration of simulation models. *Future Generation Computer Systems*, 29(8):1981 – 1990.

Sinek, J., Frieboes, H., Zheng, X., and Cristini, V. (2004). Two-dimensional chemotherapy simulations demonstrate fundamental transport and tumor response limitations involving nanoparticles. *Biomedical Microdevices*, 6(4):297–309.

Soodabeh Saeidnia, Azadeh Manayi, M. A. (2013). The Pros and Cons of the In-silico Pharmaco- toxicology in Drug Discovery and Development. (August).

Taillandier, P., Gaudou, B., Grignard, A., Huynh, Q.-N., Marilleau, N., Caillou, P., Philippon, D., and Drogoul, A. (2019). Building, composing and experimenting complex spatial models with the GAMA platform. *GeoInformatica*, 23(2):299–322.

Wilensky, U. (1999). Netlogo. http://ccl.northwestern.edu/netlogo/, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

Appendix A - Class Diagram

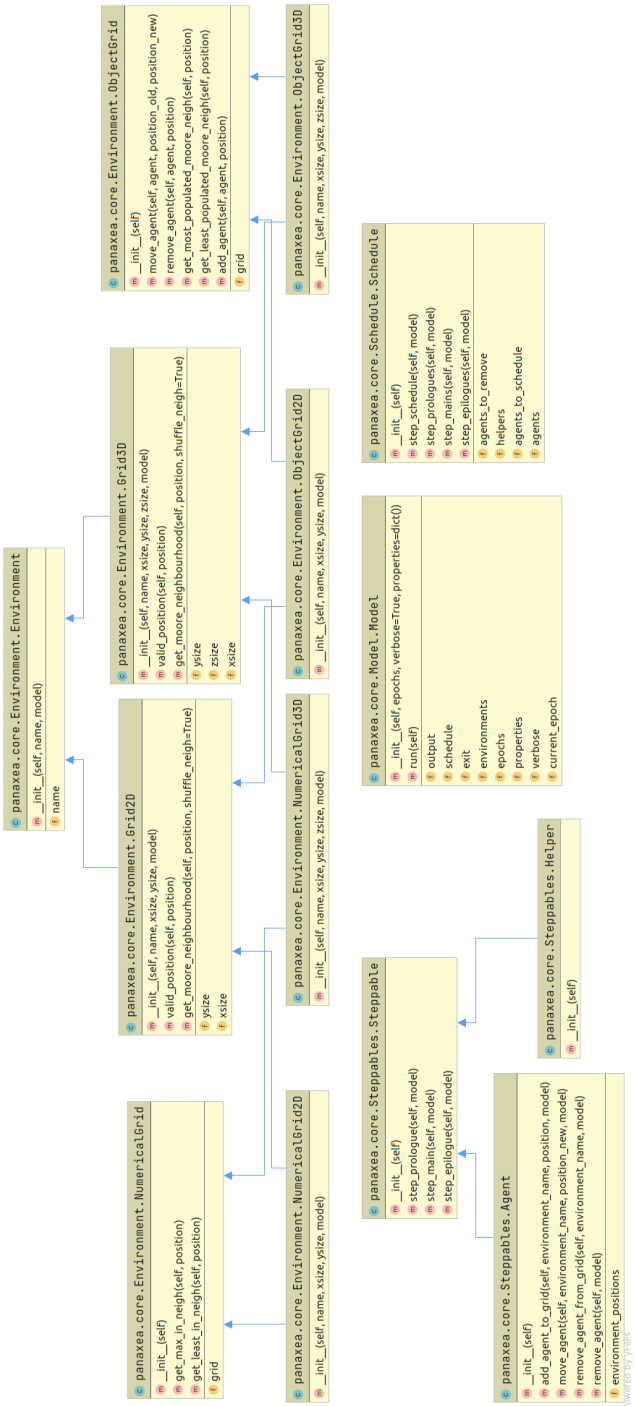

Figure 3. Class Diagram for PanaXea. Note environments may be either 2D or 3D, and in each variant may hold objects (Ie: Agents) or numerical values. Steppables include both agents and helpers.