Finding the Best Configuration for Biological Simulations at PhysiBoSS

School of Electrical and Computer Engineering,
Technical University of Crete

Marios Tsolekas

Chania, July 2021

Advisory Committee
Antonios Deligiannakis, Professor (Supervisor)
Michail G. Lagoudakis, Associate Professor
Vasilis Samoladas, Associate Professor

Abstract

Biological simulations for simulating large cell populations have become a necessary tool in modern science and the software that has been developed with that goal in mind is complex, having to emulate not only how neighboring cells interact with each other, but also how external stimuli and environmental factors affect the population as a whole and how it affects groups of it. It is of no surprise then, that these simulations have a considerable execution time, which makes it time consuming to exhaustively run those simulations for different configurations in search of an optimal result.

The system presented in this thesis ventures to remedy that problem by leveraging Bayesian optimization to stochastically find that optimal configuration, for PhysiBoSS's biological simulations, that minimizes the number of alive cancer cells. By treating simulations as black box functions and modeling them based on samples selected by an acquisition function, it is possible to rapidly converge to an optimal configuration that corresponds to the desired global optima using only a small number of simulations, for example in a search space of a few thousand data points this process would require less than fifty samples.

Εύρεση της Καλύτερης Σύνθεσης για Βιολογικές Προσομοιώσεις στο **PhysiBoSS**

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πολυτεχνείο Κρήτης

Μάριος Τσολέκας

Χανιά, Ιούλιος 2021

Περίληψη

Οι βιολογικές προσομοιώσεις με στόχο την προσομοίωση μεγάλων πληθυσμών κυττάρων έχουν καθιερωθεί πλέον ως ένα απαραίτητο εργαλείο της μοντέρνας επιστήμης και το λογισμικό που έχει αναπτυχθεί με αυτόν τον σκοπό κατά νου είναι περίπλοκο, καθώς χρειάζεται να προσομοιώσει όχι μόνο την αλληλεπίδραση των γειτονικών κυττάρων, αλλά και το πώς επηρεάζεται το καθένα από εξωτερικούς ερεθισμούς, καθώς και πως επηρεάζονται ομάδες εξ αυτών. Βάσει των παραπάνω δεν καθιστά έκπληξη το ότι αυτές οι προσομοιώσεις έχουν αρκετά μεγάλο χρόνο εκτέλεσης, πράγμα που καθιστά δύσκολη και χρονοβόρα την εκτέλεσή τους για κάθε δυνατό συνδυασμό παραμέτρων σε αναζήτηση ενός βέλτιστου αποτελέσματος.

Το σύστημα που παρουσιάζεται σε αυτή την διπλωματική εργασία έχει ως στόχο την επίλυση αυτού του προβλήματος, χρησιμοποιώντας την μέθοδο της Μπαεσιανής βελτιστοποίησης για να βρει στοχαστικά έναν βέλτιστο συνδυασμό παραμέτρων, για τις βιολογικές προσομοιώσεις του PhysiBoSS, ο οποίος θα ελαχιστοποιεί τον αριθμό των ζωντανών καρκινικών κυττάρων. Θεωρώντας τις προσομοιώσεις αυτές ως συναρτήσεις «μαύρα κουτιά» και μοντελοποιώντας τες βάσει δειγμάτων που επιλέγονται από μια συνάρτηση απόκτησης, καθίσταται δυνατό να συγκλίνει γοργά το σύστημα σε έναν βέλτιστο συνδυασμό παραμέτρων που αντιστοιχεί στο επιθυμητό ολικό μέγιστο ή ελάχιστο, εκτελώντας έναν πολύ μικρό αριθμό προσομοιώσεων, για παράδειγμα, σε έναν χώρο αναζήτησης μερικών χιλιάδων σημείων η διαδικασία αυτή θα χρειαστεί λιγότερα από πενήντα δείγματα.

Contents

List of Figures

List of Tables

Listings

1 Introduction

Research communities all over the world are relying more and more on computer simulations to provide data at a rate that would be nearly impossible by only performing real world experiments. A major downside of this trend is that the simulations that are required to be run often require major computational power and time. This poses a problem especially for smaller scale institutes that do not have the resources or for research subjects that are heavily time constrained.

The system presented in this thesis is an attempt to combat that issue, using a stochastic method and heurestics to drastically reduce the amount of simulations needed to be run in order to obtain the desired result. In its current state the scope of the system is fairly limited, focusing on finding the optimal combination of input parameters that minimize a certain value. That being said, the problem it tackles is in no way small, mainly due to the gigantic search space that is generated due to the dimensionality and multitude of the input parameters.

1.1 Motivation & Goal

A research subject that fits the above criteria is the development of cancer cells. Due to their rapid and unpredictable growth, there is a need for software tools to model the behavior of these cells in order to be able to predict their growth and treat it accordingly. One such tool is the PhysiBoSS framework that is used in this thesis which does exactly that. It can simulate the effect different stimuli have to these cells, allowing one to choose the appropriate method of treatment to minimize their population. As has already been mentioned though these simulations are costly in compute and long running, thus rendering the brute force method of computing every possible combination and then finding the optimal one, very inefficient. In this context, the stochastic method of Bayesian Optimization is applied to reduce the number of required simulations immensely, in order to find an optimal configuration. This is a method that has been used in a similar fashion by Alipoufard, etc. in a completely different field, namely finding optimal configurations for virtual machine acquisition. Their results were very promising.

The goal of the system presented in this thesis is to leverage Bayesian Optimization with the goal of finding the optimal combination of input parameters that produce a minimal population of cancer cells, using PhysiBoSS to simulate their growth. A key design feature is to be easily extensible for the use of different simulator software, providing the quality of life features present in this system to any project that requires them.

1.2 Outline

In Chapter 2, the method of Bayesian Optimization is presented in detail, together with information on the PhysiBoSS software framework. In Chapter 3, a detailed analysis on the system design and its usage is provided, with experimental results and commentary on them following in Chapter 4. Finally in Chapter 5, conclusions on the efficacy of the system are drawn and some goals are set for its expansion in the near future.

2 Background

2.1 Bayesian Optimization

Bayesian Optimization (BO) is a method used for global optimization of black box functions, that is finding minima and maxima without any assumptions of the function's form. It is a sequential design strategy suitable for big data applications and expensive to evaluate functions, very much like the biological simulations discussed in this thesis.

BO is an iterative process that relies largely on two steps being repeated for every sample of the objective function $f(\vec{x})$ that is taken. Since the objective function is unknown the first step is to model it as a stochastic process and compute the confidence interval according to the samples taken so far, which in essence is the area where $f(\vec{x})$ is more likely to pass through. The above is in essence the belief about the function that the process has so far. The second step is deciding which point to sample next. That is done using an acquisition function that is updated in tandem with the confidence interval. There are many different variations of acquisition functions, but generally they are easy to compute functions whose maxima/minima can be found using traditional methods like Newtons method. More details about these two steps will be given further below in the upcoming segments.

Prior function - Stochastic Model The prior function can be any stochastic process that is fit for a particular problem domain. For the purposes of this thesis a Gaussian Process is chosen, that is, the objective function's values are assumed to follow a multivariate Gaussian process. In this case the objective function $f(\vec{x})$ is described with a mean function μ and a covariance kernel k, that for any pair of sample points $\vec{x_1}, \vec{x_2}$ are [1]:

$$\mu_1(\vec{x_1}) = \mathbb{E}[f(\vec{x_1})]$$

$$\mu_2(\vec{x_2}) = \mathbb{E}[f(\vec{x_2})]$$

$$k(\vec{x_1}, \vec{x_2}) = \mathbb{E}[(f(\vec{x_1}) - \mu_1(\vec{x_1}))(f(\vec{x_2}) - \mu_2(\vec{x_2}))]$$

$$C_{5/2}(d) = \sigma^2(1 + \frac{\sqrt{5}d}{2} + \frac{5d^2}{3 \cdot 2^2}) \exp(-\frac{\sqrt{5}d}{2})$$

Where d is the distance units separating two points. The covariance function chosen in this thesis and shown above is a $Matern5/2$ [11] covariance function as it is the preferred function to model practical applications. In essence, the more similar two values of the objective function $f(\vec{x})$ are, the greater their covariance and vice-versa.

Acquisition function As stated briefly above, using the acquisition function, BO can determine which point to sample next. There are four main acquisition function strategies [4], each with their own merits and demerits, as well as their most suitable application domains. The following descriptions assume that BO tries to determine to global minimum.

- Probability of Improvement (PI): This is arguably the first acquisition function designed for BO. In essence, PI evaluates $f(\vec{x})$ at the point most likely to produce a value that will be less than the currently observed minimum. Its utility function gives a unit reward if that holds true and the acquisition function is the expected utility:

$$u_{PI}(\vec{x}) = \begin{cases} 0 & \text{if } f(\vec{x}) > f',\ where f' = min(f)\ observed\ so\ far \\ 1 & \text{if } f(\vec{x}) \leq f' \end{cases}$$

$$\alpha_{PI}(\vec{x}) = \Phi(f'; \mu(\vec{x}), k(\vec{x}, \vec{x}))$$

  where $f' = minf(\vec{x})$ the current minimum observed thus far. From this the point with the highest probability of improvement is selected. The main drawback of this method is that it doesn't account for the amount of improvement, which in practice is usually not desired and can lead to it being stuck on local minima.

- Expected Improvement (EI): This method, in contrast to PI, sets to account for the amount of improvement. That is, it evaluates $f(\vec{x})$ at the point most likely to produce the smallest value. Its utility function gives a reward proportional to amount of improvement and its acquisition function is its expected utility again:

$$u_{EI}(\vec{x}) = max(0, f' - f(\vec{x})),\ where f' = min(f)\ observed\ so\ far$$

$$\alpha_{EI}(\vec{x}) = (f' - \mu(\vec{x}))\Phi(f'; \mu(\vec{x}), k(\vec{x}, \vec{x})) + k(\vec{x}, \vec{x})N(f'; \mu(\vec{x}), k(\vec{x}, \vec{x}))$$

  where $f' = minf(\vec{x})$ the current minimum observed thus far. In this case the point with the highest expected improvement is sampled next. EI has the added advantage of being able to automatically adjust the trade-off between exploitation and exploration. This is the method chosen in this thesis.

- Entropy Search (ES): This method tries to reduce the uncertainty of the location of the true minimum. Its utility function is the reduction of entropy with regards to the true minimum. Its expected utility cannot be easily determined and requires several approximations.

- Gaussian Process - Upper Confidence Bound (GP-UCB): With this acquisition function the point that has the smallest lower bound with regards to its uncertainty region is chosen. In this case a more apt name would be lower confidence bound, but upper confidence bound has become the standard term. Like EI, this method contains terms to control the trade-off between exploitation and exploration, but the evaluation of this acquisition function is not as simple as computing the expected utility.

Observation noise BO can accommodate observation noise when computing the confidence interval, assuming of course that the noise can be meaningfully modeled stochastically, as is the case with Gaussian noise for example. Given the above and together with the value of $f(\vec{x})$ observed through sampling, BO can infer the actual value of $f(\vec{x})$. Supposing the below example, where $f(\vec{x})$ is the actual value of the objective function and $e \sim N(\mu, \sigma^2)$ is the Gaussian noise of observation, the noisy objective function can be defined as:

$$\dot{f}(\vec{x}) = f(\vec{x}) + e$$

from which BO can infer $f(\vec{x})$. Of course that isn't useful for every application of BO, but is nevertheless a great boon when needed.

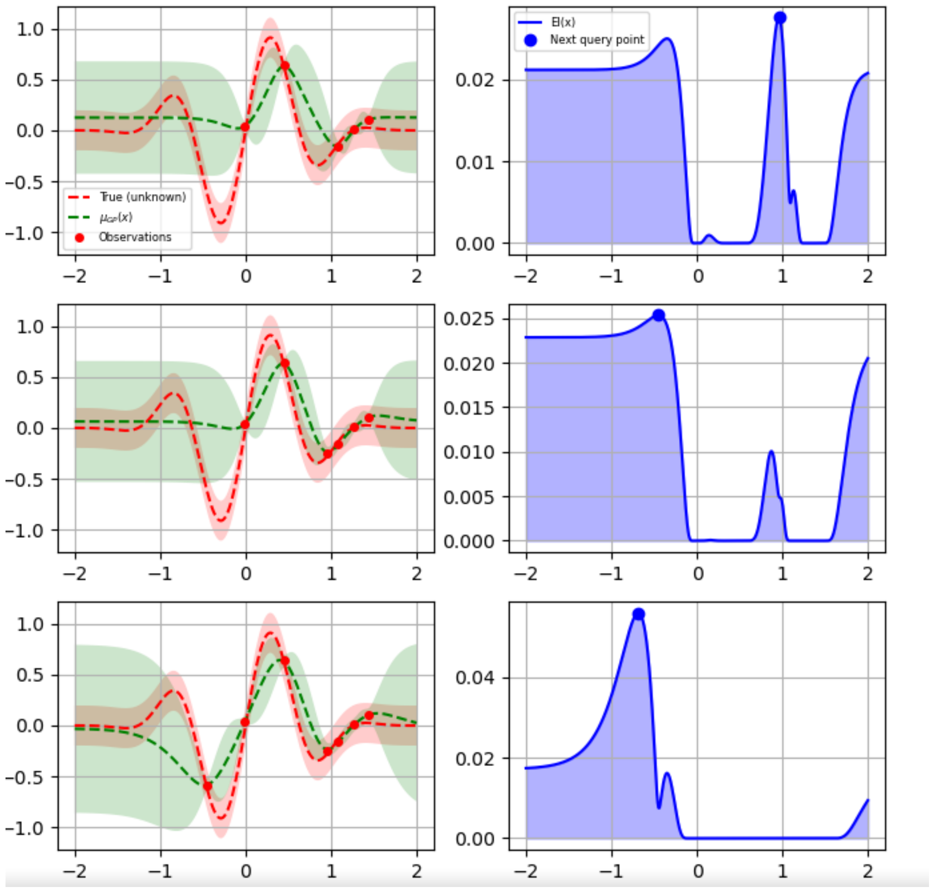


Figure 2.1: Bayesian optimization example [14]. On left is the model and on the right the acquisition function. It can be seen at each step of how the next sample is selected based on the maximum value of the acquisition function and how the beliefs of the system about the true function change with each new sample.

In summary for the process of BO, a stochastic model is created based on the samples that have been sampled thus far and the next point to sample is decided through the maxima of the acquisition function. This process repeats until the conditions have been met for it to end. These conditions are application specific and will be discussed in their own context later, but generally either a set number of samples must be taken, or the process has reached a result that is within a desired margin. Lastly it must be noted that the first model to be computed is based on randomly taken samples.

2.1.1 SciKit-Learn

SciKit-Learn [13] is an open source machine learning library for Python. Its features include, among others, functions and classes for regression, classification, clustering, model selection and dimensionality reduction. It is the basis of the SciKit-Optimize package described in the following subsection, which implements a lot of the functionality of BO.

2.1.2 SciKit-Optimize

SciKit-Optimize [14] is an open source Python library for sequential model-based optimization. It is built on SciKit-Learn. It is a small and simple library that provides facilities to globally optimize very expensive and noisy black-box functions. For this thesis it is used mainly to perform BO, for which it provides ample customization, from choosing which acquisition function to use, to setting stop conditions and even registering custom callbacks to be used during the process. It also has plotting functionality, albeit limited, which allows for easy visualization of the process of BO.

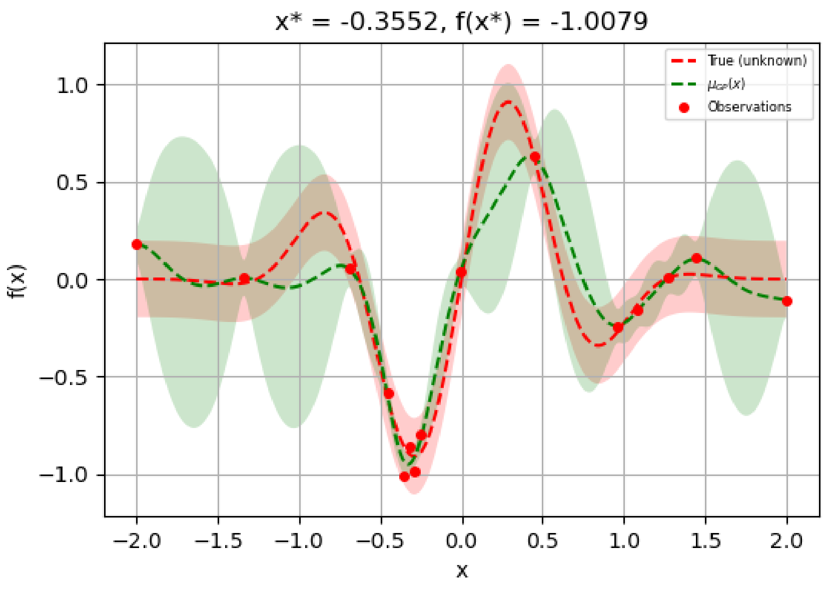


Figure 2.2: SciKit-Optimize Bayesian optimization example result [14]

2.2 Biological Simulations

Simulators have seen wide adoption in every field of research, from biology to astronomy, as they can accurately model the world around us without being constrained by the limitations of our physical world. As such results that could take months to acquire can be extracted with the use of simulators within a fraction of that time, as long as the required computational power is provided. In addition to that, the advent of big data provides efficient ways to parse the huge amount of data that can be generated from various simulations. this thesis focuses on biological simulations using PhysiBoSS.

2.2.1 PhysiCell

PhysiCell [5] is an open source software package that has the ability to simulate large systems of cells, even on desktop computers. To accomplish that it does not simulate the morphology, but only approximates it where needed, while also parallelizing the process using OpenMP where possible. This allows it to simulate the behavior of thousands of cells over several days in a matter of hours.

PhysiCell doesn't only simulate the interactions between cells, but also how certain environmental factors affect each cell leveraging BioVFM, allowing for very robust simulation results over the a population of cells.

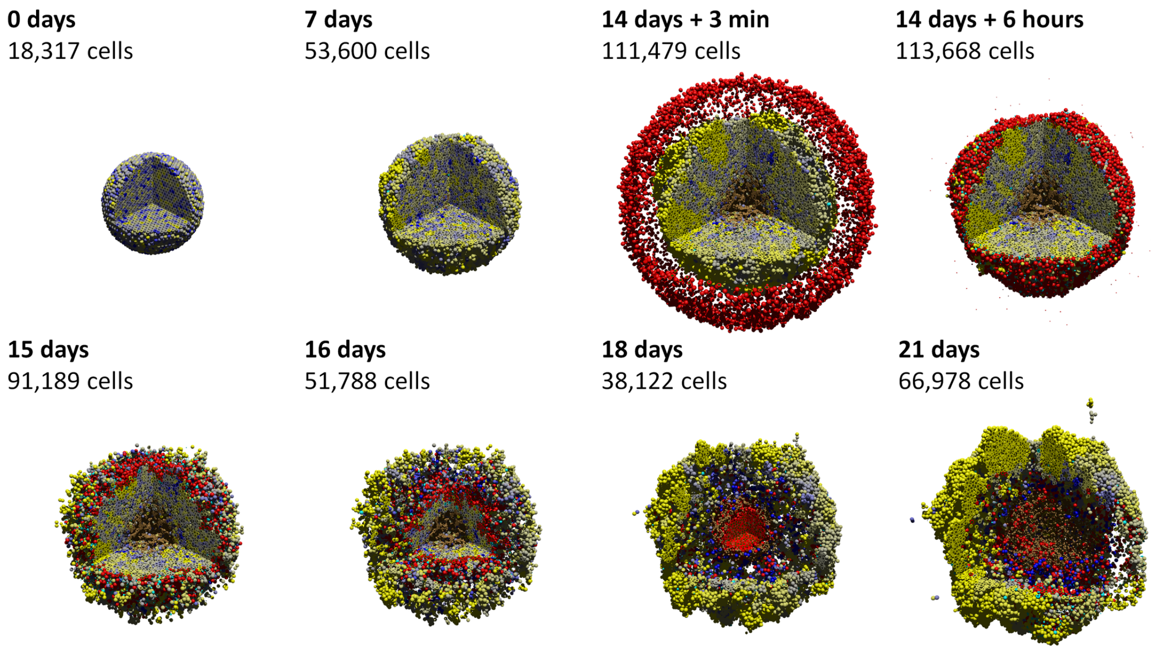


Figure 2.3: PhysiCell cancer immunology example [5]

2.2.2 MaBoSS

MaBoSS [15] is an open source software package that simulates both continuous and discrete Markov processes applied on a Boolean network. Starting from a given set of initial conditions and applying the Monte-Carlo kinetic algorithm to the aforementioned Boolean network, MaBoSS produces time trajectories and estimates the time evolution of probabilities.

In essence, variables which can represent genes, proteins, etc. take on the values 0 and 1 according to their activity. Each variable's value is stochastically calculated by the status of its regulating variables and those regulating variables are connected to the initial variable using Boolean operators.

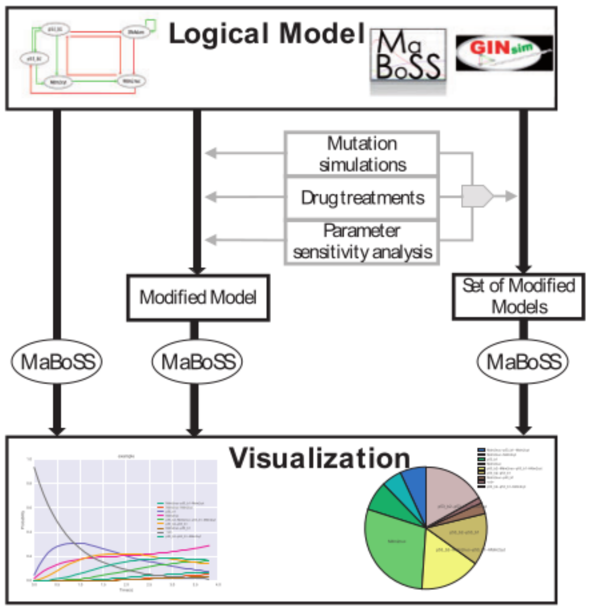


Figure 2.4: Pipeline of the use of MaBoSS 2.0 functionalities [15]

2.2.3 PhysiBoSS

PhysiBoSS [6] is an open source software package that performs multi-scale simulations of heterogeneous multi-cellular systems. It integrates the PhysiCell and MaBoSS software frameworks, which were described in the previous sections, to model each cell's behavior and its reactions to external stimuli, as well as the effects on the whole cell population.

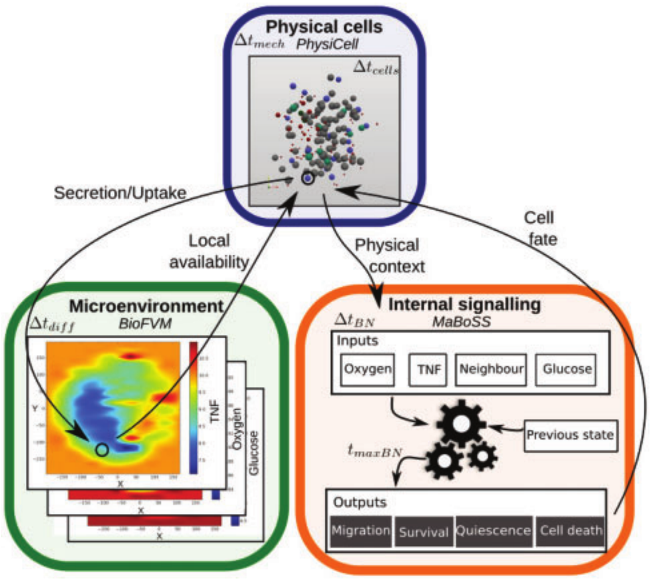


Figure 2.5: PhysiBoSS representation [6]

For the purposes of this thesis and by extension the system that is implemented, the usage of PhysiBoSS consists mainly of two executables, the actual simulator PhysiBoSS and the PhysiBoSS_CreateInitTxtFile which generates the initial conditions of the cells. Furthermore to successfully perform a simulation, a certain directory structure must be established below the top level project directory:

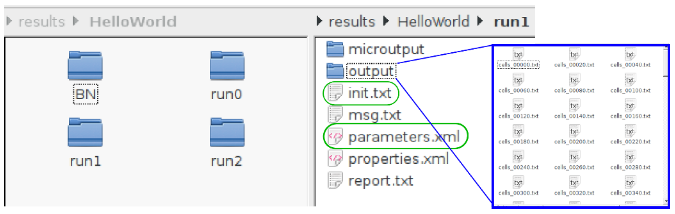


Figure 2.6: PhysiBoSS directory organization [9]

- The BN directory contains the Boolean network configuration file required by the MaBoSS component.
- Numbered run directories, which are specific to each simulation run. These contain:

  - The init.txt file which can be generated by the aforementioned executable and contains the initial states of the cells.
  - The parameter.xml which contains all the tunable properties of a simulation.
  - The files and directories generated after running a simulation, of which the output directory is a part of.

```xml
<?xml version="1.0" encoding="UTF-8" ?>

<simulation>
<time_step> 0.2 </time_step>
<mechanics_time_step> 0.1 </mechanics_time_step>
....
</simulation>

<cell_properties>
<mode_motility> 1 </mode_motility>
<polarity_coefficient> 0.5 </polarity_coefficient>
...
</cell_properties>

<network>
<network_update_step> 10 </network_update_step>
...
</network>

<initial_configuration>
<load_cells_from_file> init.txt </load_cells_from_file>
...
</initial_configuration>
```

Listing 2.1: Partial PhysiBoSS parameter file - parameter.xml [9]

The above is the typical structure required for running simulations manually, but as will be explained in the System Design section later the numbered run directories are generated by the system, thus the actual directory structure that must be setup is altered to the following:

- The BN directory contains the Boolean network configuration file required by the MaBoSS component.
- The init.txt file which can be generated by the aforementioned executable and contains the initial states of the cells.
- The parameters_template.xml which contains all the tunable properties of a simulation. The template is mostly an exact copy of the actual parameters.xml, with the only difference being that for the parameters that change between simulations, instead of an actual numerical value, a placeholder text must be in its place. Note that the file name parameters_template.xml can be changed to anything, as its path is specified through the systems configuration file.

With just the above top level structure for each simulation a numbered run directory is created with the init.txt copied inside it and an appropriate parameters.xml generated from the template.

```
...
<initial_configuration>
  <load_cells_from_file> init.txt </load_cells_from_file>
  <membrane_shape> sphere </membrane_shape>
  <membrane_length> 470 </membrane_length>
  <time_passive_cells> 1500 </time_passive_cells>
  <oxygen_concentration> 40 </oxygen_concentration>

  <!-- These parameters have placeholder strings ,instead of actual values,
       which will be replaced with the appropriate values before a
       simulations is run -->
  <tnf_concentration> BAMIN_X2 </tnf_concentration>
  <duration_add_tnf> BAMIN_X1 </duration_add_tnf>

  <time_remove_tnf> 80000 </time_remove_tnf>
  <time_add_tnf> 25 </time_add_tnf>
  <mode_injection> 1 </mode_injection>
</initial_configuration>
 ...
```

Listing 2.2: Partial parameter template file - parameter_template.xml

2.3 Numerical Libraries

2.3.1 NumPy

NumPy [8] is an open source library for Python that adds support for n-dimensional arrays, along with efficient functions to operate on them and an abundance of numerical computation tools, from Fourier transform to linear algebra functions and more. It is the basis of many scientific packages, including SciKit-Optimize used in this thesis, and is used extensively to manipulate multi-dimensional arrays in the system described in this thesis.

2.4 Plotting Libraries

2.4.1 Matplotlib

Matplotlib [7] is an open source plotting library for Python with the ability to support multiple graphical back-ends suck Tk (Tkinter), Qt and GTK+. It is used in SciKit-Optimize for its plotting functionality, as well as in this thesis for visualizing different stages and parts of the optimization process.

2.5 Configuration Parsing

2.5.1 TOML

TOML (Tom's Obvious Minimal Language) [16] is a file format and specification for configuration files of any kind. One of its key features is that it is very easy and simple to read write, as well as map unambiguously to a hash map. TOML has libraries that help with its parsing in many programming languages.

```toml
# This is a TOML document

title = "TOML Example"

[owner]
name = "Tom Preston-Werner"
dob = 1979-05-27T07:32:00-08:00

[database]
enabled = true
ports = [ 8000, 8001, 8002 ]
data = [ ["delta", "phi"], [3.14] ]
temp_targets = { cpu = 79.5, case = 72.0 }

[servers]

[servers.alpha]
ip = "10.0.0.1"
role = "frontend"

[servers.beta]
ip = "10.0.0.2"
role = "backend"
```

Listing 2.3: TOML example - example.toml [16]

TOML is the language chosen for the configuration file of the system in this thesis and is parsed with the help of the open source Python package toml [10], which is a Python library for creating and parsing TOML.

2.6 Computing Grid

2.6.1 TUC Grid Computer

The grid computing [18] system of the Technical University of Crete began its operations in 2009 with the goal of providing a compute platform to members of the scientific community of the university with plans to integrate it with the national and European grids through HellasGrid-Eurogrid in the near future.

At the time of writing it is comprised of 2048 compute units, each with four processing cores, allowing for the timely execution of massive jobs in parallel, like those required by this thesis. Access is provided through SSH into a Unix shell and monitoring through an analytics reporting web platform. Job scheduling is performed via the TORQUE [17] resource manager, which provides the necessary command-line tools and inline script integration with PBS directives in order to schedule jobs across the compute cluster.

For the purposes of this thesis the grid computing platform is used to exhaustively run every simulation in the search space, in order to test the efficacy of the system presented, whose goal is to remove the need for so many data points.

2.7 Related Work

The central pillar of the system presented here is the use of Bayesian Optimizaton to efficiently find optimal configurations that produce a global optimum for long running simulations, that are in essence black box functions. Those requirements are in no way exclusive to this system, either in part or as a whole, which has lead to the use of this method in multiple scientific ventures, from fine tuning ML models to robotics and medical applications. Below are a few choice applications that leverage this method.

2.7.1 Bayesian optimization for sensor set selection [3]

This work focuses on selecting sensor sets, such that the resulting sensor network produces an optimal metric, for example in predictive accuracy. In addition it explores ways to find the optimal placement of sensors. In contrast to this thesis, where each sample is a simulation with an inherent numerical value as a result, this work required the introduction of custom metrics by the researchers in the real world, which made the problem domain all the more complex.

2.7.2 CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics [1]

CherryPick is an application that leverages BO in order to determine optimal cloud configurations, i.e. disk space, core count, ram size etc., that best fit a particular big data analytics job. It runs a test job that is indicative of the real load on each sample configuration and converges to an optimal one.

There are many parallels to be drawn between it and the system presented in this thesis, but the major difference is that CherryPick is "laser focused" on unearthing optimal cloud configurations with built-in integration for a multitude of cloud providers and a variety of domain specific functionality, while the system presented here is more application and domain agnostic allowing it to be adapted to any simulator with isolated code additions.

2.7.3 Application of Bayesian Optimization for Pharmaceutical Product Development [12]

This paper introduces BO to the development cycle of pharmaceutical products, using it to optimize manufacturing methods and cut down on unnecessary experiments. Like with the sensor set selection work, described above, the researchers defined their own metrics that suit this application domain in order to treat this process as a black box function. In essence, by using BO, they are trying to replicate the selection that an experienced human researcher would do, with all the added benefits of the process being computerized.

3 System Design & Implementation

The goal of this system is to efficiently find an optimal configuration of a simulations parameters in order to minimize the requested output parameter. Given that these simulations are long running and the dimensionality of the configuration parameters can require a huge number of simulations to be run in order to obtain the desired result the conventional way, the method of Bayesian Optimization has been chosen to reduce that number to double digits. The system has been broken down into modular parts that allow a user to very easily define the problem they are trying to solve and the outputs they desire, as well as be easily extensible in the future.

3.1 Overview

Overall the system can be broken down into four main sections. Those are the input parsing, the optimization process, the simulation dispatching and the plotting. Plotting is present throught the system and does as its name implies, thus wont be analyzed much here. In this stage, only the PhysiBoSS simulation framework is implemented, thus it is what will be used as an example throughout. The following paragraphs briefly describe the process as a whole. Detailed explanation on each part will be given in the corresponding sections.

Preparation The process begins by the user preparing two files. One is the configuration file for the system and the other is the configuration template for PhysiBoSS. On execution the configuration file is read and the stochastic process begins immediately. From here on any mention of samples, refers to a combination of input parameters for PhysiBoSS.

Iterative process A number of samples, dictated by the user, are taken randomly from the search space and the result of their simulations are saved. These initial samples are the basis of the first model and acquisition function. Once these are present, the iterative process begins. A new sample is selected based on the maximum value of the acquisition function and the dispatcher is requested to run its simulation. The returned value is saved and the model and acquisition function are updated. In the entirety of the process there are safeguards to avoid running simulations for samples that have been already run, mainly in order to ease development and testing efforts. The dispatcher uses the configuration template as a basis for each simulation.

Results The above process continues until the maximum number of samples is taken or the delta between a number of returned values is smaller than a given threshold. In the end the final results are printed along with various plots that may have been requested by the user.

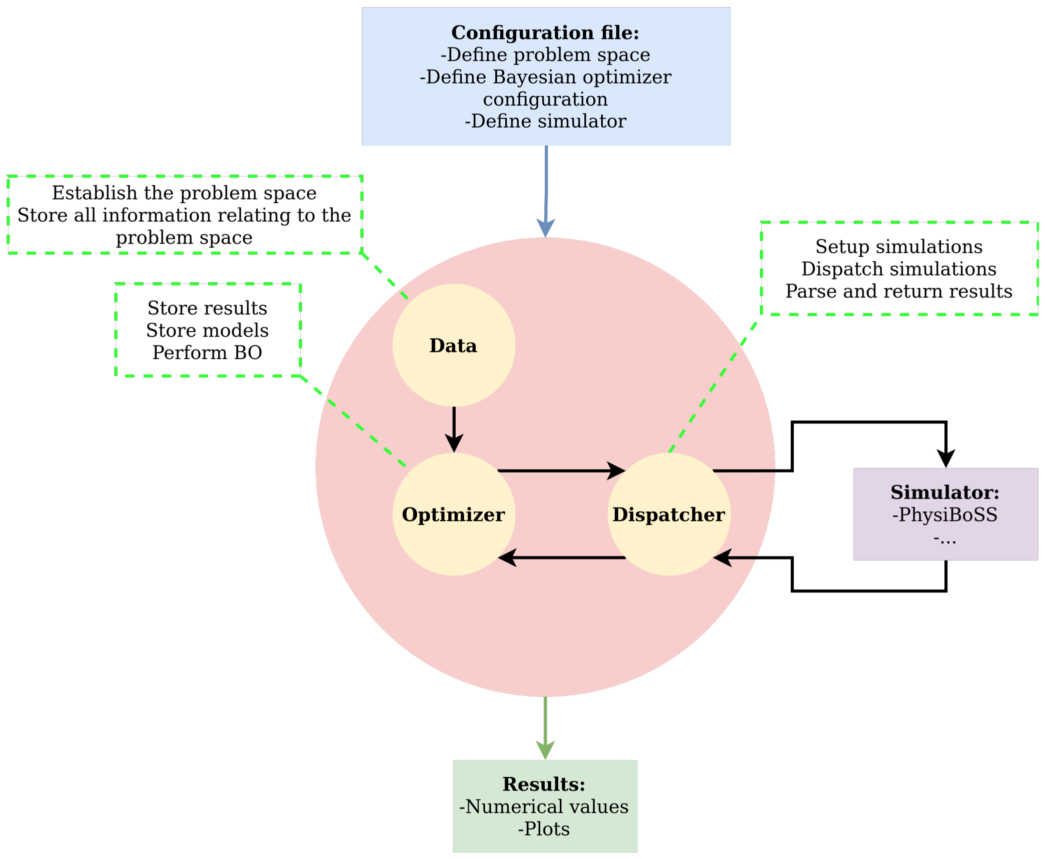


Figure 3.1: Graphical representation of the system

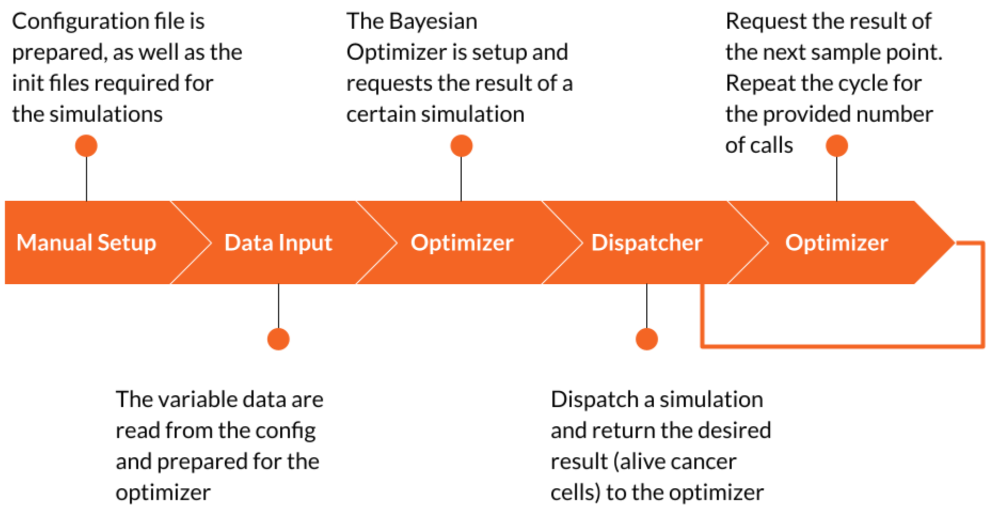

Configuration file is prepared, as well as the init files required for the simulations
The Bayesian Optimizer is setup and requests the result of a certain simulation
Request the result of the next sample point. Repeat the cycle for the provided number of calls
Manual Setup
Data Input
Optimizer
Dispatcher
Optimizer
The variable data are read from the config and prepared for the optimizer
Dispatch a simulation and return the desired result (alive cancer cells) to the optimizer

Figure 3.2: Graphical representation of the systems workflow

3.2 Input

The input of the system can be broken down two to parts, the configuration file and the command-line arguments.The configuration file focuses on parameters pertaining to the simulations, the data and the optimization process, while the command-line arguments contain some knobs to tune the behavior of the system as a whole.

The chosen format of the configuration file is TOML, as it provides enough structure to define anything this system needs, while also maintaining high readability and being easy for humans to generate. The configuration file is broken down to segments which control the corresponding parts of the system.

- The data segment, which contains information on each configuration parameter for simulation. This information includes the label, the range of the parameters values and the step with which to traverse the given range. All of the above are used to compute all the possible combinations that would be required in order to find the minimum, in essence bounding the problem space.

- The optimizer segment provides the optimizer parameters that control things like the noise of the objective function, the number of optimization calls to make and simulations to run before stopping, the number of initial random points and the value of the threshold that controls the early stopping of the optimization process.

- The dispatcher segment is a collection of paths for the simulator. These include the executable path of the simulator, where its configuration template can be found and where to output the results.

- Lastly the result segment contains the label of the output parameter on which the optimization process is performed for. For example the number of alive cancer cells is the focus of this thesis, but it could also be the number of apoptotic cells, or in the case of a different simulator something else entirely.

```
[optimizer]
noise = 0.1
n_calls = 30
n_random_starts = 3
delta_stop = 5

[dispatcher]
simulator = "physiboss"
exec_path = "Workspace/PhysiBoSS/bin/PhysiBoSS"
conf_template = "Workspace/examples/example4/parameters_template.xml"
output_path = "Workspace/examples/example4"

[data]
    [data.x1]
    label = "tnf_dur"
    limits = [2, 10]
    step = 2

    [data.x2]
    label = "tnf_conc"
    limits = [0.1, 0.8]
    step = 0.1

    [data.x3]
    label = "oxy_nec"
    limits = [0.5, 0.7]
    step = 0.1

    [data.x4]
    label = "time_add"
    limits = [5, 1400]
    step = 5

[result]
    label = "alive"
```

Listing 3.1: Configuration file example - config.toml

The command-line arguments have options such as whether to project the multiple dimensions into one for ease of plotting, for which iteration of the process, if any, to plot the model and acquisition function, or whether to perform Bayesian Optimization at all, in which case all of the search space is simulated, which is useful for testing and providing the systems other features without having to script everything from scratch. Furthermore, there are options to run one of the provided examples without the simulation process, which is again there for testing purposes. These examples have already been explored in full for specific output parameters. Lastly, there is an option to compare the results of different simulator versions in order to find discrepancies between them.

3.3 Optimization Process

Following the parsing of the input data, the process of finding a configuration that produces the minimum required value begins. The heavy lifting of this process, that is the Bayesian Optimization, is handled mostly by the open source python package SciKit-Optimize [14], which is based in turn on the SciKit-Learn [13] package. The search space is prepared according to the limits set by user and the optimizer configuration values are plugged in to it.

The two main parts of the Bayesian optimization process are the prior and acquisition functions. The methods chosen for this system are briefly outlined below. For further details and rationales refer to section 2.1.

Prior function A Gaussian process is chosen as the prior function, which leads to the objective function being described by a mean function and a covariance kernel for each pair of sample points $\vec{x_1}, \vec{x_2}$:

$$\mu_1(\vec{x_1}) = \mathbb{E}[f(\vec{x_1})]$$

$$\mu_2(\vec{x_2}) = \mathbb{E}[f(\vec{x_2})]$$

$$k(\vec{x_1}, \vec{x_2}) = \mathbb{E}[(f(\vec{x_1}) - \mu_1(\vec{x_1}))(f(\vec{x_2}) - \mu_2(\vec{x_2}))]$$

The kernel chosen is a Matern5/2 covariance kernel, whose simplified half integer form for $p = 2, v = p + 1/2 = 5/2$

$$C_{5/2}(d) = \sigma^2(1 + \frac{\sqrt{5}d}{2} + \frac{5d^2}{3 \cdot 2^2}) \exp(-\frac{\sqrt{5}d}{2})$$

Acquisition function The strategy chosen for the acquisition function is that of Expected Improvement (EI), where the sample point that is expected to give the maximum amount of improvement over the currently observed minimum is selected.

$$u_{EI}(\vec{x}) = max(0, f' - f(\vec{x})),\ where f' = min(f)\ observed\ so\ far$$

$$\alpha_{EI}(\vec{x}) = (f' - \mu(\vec{x}))\Phi(f'; \mu(\vec{x}), k(\vec{x}, \vec{x})) + k(\vec{x}, \vec{x})N(f'; \mu(\vec{x}), k(\vec{x}, \vec{x}))$$

In the entirety of the system, the search space is represented as discrete values, except when used as the search space for Bayesian Optimization. For that reason the search space values are given as continuous real values to optimizer and are discretized again at each step inside the objective function before any further processing happens.

The objective function's main role is to run a simulation with the given input parameters and save the returned result. The result of each simulation for a given combination of parameter values are saved inside a hash-map in order to prevent rerunning of the same combinations. This method uses the memory of the system and as such it cannot persist between executions. For that reason there exists another method of avoiding duplicate simulation runs, that persists through executions, in the dispatching of the simulation, which will be described in more detail in the following chapter.

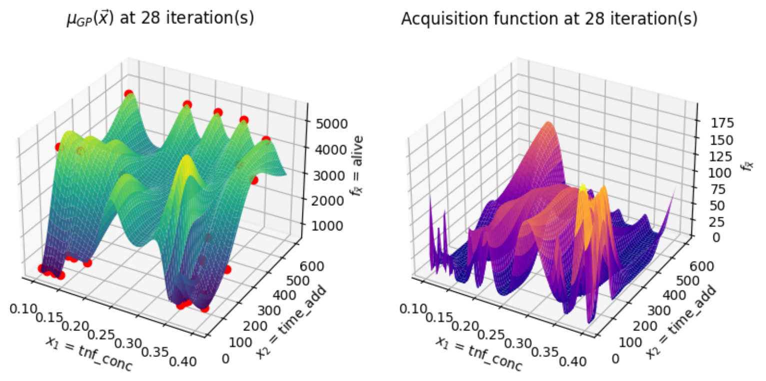


Figure 3.3: Model and acquisition function at the end of the process. In this simplistic case it can be seen that the system has converged to the true minimum, which is 536 alive cancer cells with a combination of parameters $tnf\_concentration = 0.4$ and $time\_add\_tnf = 125$, which has been traditionally computed beforehand. The red dots represent the samples taken, where it is shown that after some exploration, the process zeros-in on the parts where the minimum values lie.

Lastly there exists the option to compare different versions of a simulator. For each simulation that would be run, another one is run using the other version with the same parameters. The Root Mean Square Error of the results is computed and plotted for analysis. The main reason for this functionality is to discover errors in the previous version of PhysiBoSS, but it could be useful for other simulators as well.

3.4 Dispatching Simulations

The dispatching part of the system, as the name implies is used to dispatch simulations. This is the part that gets called from inside the objective function to run a simulation with the given parameters and return the result. This process is responsible for any setup the simulation requires, running it with the correct inputs and lastly parse the desired result and return its value.

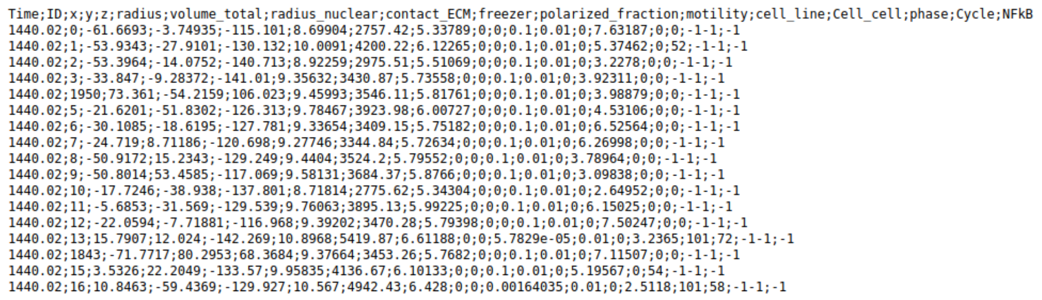
For the PhysiBoSS simulations the setup part consists of creating the directory hierarchy required for the simulations and generating each simulation's parameter.xml and init.txt from a given template file. The advantage of using a user provided template file is that every other parameter that is of no concern to the process of Bayesian Optimization can be tuned to anything the current user wants. The results of each simulation are compiled and for this case the number of alive cancer cells is extracted. The number of alive cells is extrapolated from the phase each cell is in for the last epoch of the simulation. A phase value of 0 or 1 signifies it is alive. Below is a table that maps each numerical phase value to their actual label.

Label	Phase ID	Phase
alive	0	Ki67 Positive Premitotic
	1	Ki67 Positive Postmitotic
apoptotic	100	Apoptotic
necrotic	101	Necrotic Swelling
	102	Necrotic Lysed

Table 1: Cell phase mapping to labels used for results

```json
{
  "0": "Ki67_positive_premitotic",
  "1": "Ki67_positive_postmitotic",
  "2": "Ki67_positive",
  "3": "Ki67_negative",
  "4": "G0G1_phase",
  "5": "G0_phase",
  "6": "G1_phase",
  "7": "G1a_phase",
  "8": "G1b_phase",
  "9": "G1c_phase",
  "10": "S_phase",
  "11": "G2M_phase",
  "12": "G2_phase",
  "13": "M_phase",
  "14": "live",
  "100": "apoptotic",
  "101": "necrotic_swelling",
  "102": "necrotic_lysed",
  "103": "necrotic",
  "104": "debris"
}
```

Listing 3.2: Cell phases hash-map - cell_phases.json. [2]



```
Time;ID;x;y;z;radius;volume_total;radius_nuclear;contact_ECM;freezer;polarized_fraction;motility;cell_line;Cell_cell;phase;Cycle;NFkB
1440.02;0;-61.6693;-3.74935;-115.101;8.69904;2757.42;5.33789;0;0;0.1;0.01;0;7.63187;0;0;-1-1;-1
1440.02;1;-53.9343;-27.9101;-130.132;10.0091;4200.22;6.12265;0;0;0.1;0.01;0;5.37462;0;52;-1-1;-1
1440.02;2;-53.3964;-14.0752;-140.713;8.92259;2975.51;5.51069;0;0;0.1;0.01;0;3.2278;0;0;-1-1;-1
1440.02;3;-33.847;-9.28372;-141.01;9.35632;3430.87;5.73558;0;0;0.1;0.01;0;3.92311;0;0;-1-1;-1
1440.02;1950;73.361;-54.2159;106.023;9.45993;3546.11;5.81761;0;0;0.1;0.01;0;3.98879;0;0;-1-1;-1
1440.02;5;-21.6201;-51.8302;-126.313;9.78467;3923.98;6.00727;0;0;0.1;0.01;0;4.53106;0;0;-1-1;-1
1440.02;6;-30.1085;-18.6195;-127.781;9.33654;3409.15;5.75182;0;0;0.1;0.01;0;6.52564;0;0;-1-1;-1
1440.02;7;-24.719;8.71186;-120.698;9.27746;3344.84;5.72634;0;0;0.1;0.01;0;6.26998;0;0;-1-1;-1
1440.02;8;-50.9172;15.2343;-129.249;9.4404;3524.2;5.79552;0;0;0.1;0.01;0;3.78964;0;0;-1-1;-1
1440.02;9;-50.8014;53.4585;-117.069;9.58131;3684.37;5.8766;0;0;0.1;0.01;0;3.09838;0;0;-1-1;-1
1440.02;10;-17.7246;-38.938;-137.801;8.71814;2775.62;5.34304;0;0;0.1;0.01;0;2.64952;0;0;-1-1;-1
1440.02;11;-5.6853;-31.569;-129.539;9.76063;3895.13;5.99225;0;0;0.1;0.01;0;6.15025;0;0;-1-1;-1
1440.02;12;-22.0594;-7.71881;-116.968;9.39202;3470.28;5.79398;0;0;0.1;0.01;0;7.50247;0;0;-1-1;-1
1440.02;13;15.7907;12.024;-142.269;10.8968;5419.87;6.61188;0;0;5.7829e-05;0.01;0;3.2365;101;72;-1-1;-1
1440.02;1843;-71.7717;80.2953;68.3684;9.37664;3453.26;5.7682;0;0;0.1;0.01;0;7.11507;0;0;-1-1;-1
1440.02;15;3.5326;22.2049;-133.57;9.95835;4136.67;6.10133;0;0;0.1;0.01;0;5.19567;0;54;-1-1;-1
1440.02;16;10.8463;-59.4369;-129.927;10.567;4942.43;6.428;0;0;0.00164035;0.01;0;2.5118;101;58;-1-1;-1
```

Figure 3.4: Partial PhysiBoSS simulation output. This is the CSV file that the dispatcher parses for its cell phases in order to determine whether a cell is alive, apoptotic or necrotic. That is it parses the phase column located 3_{rd} from the end and maps each phase code to the appropriate state. Note that for each simulation multiple of these outputs are generated, each corresponding to a certain point in time. In this system though only the final one is of concern, as that contains the cell information at the end of the simulation.

As was mentioned in the Optimization Process chapter, there is functionality included to avoid rerunning simulations for the same combinations, even between different executions. Each combination of parameters has a unique identifier in the directory name that contains the simulations results that can be computed both ways. Thus before running a simulation it is always checked if the corresponding directory and files exist. If they do the simulation is not run and the results are immediately parsed from the already existing files.

The functionality mentioned in the above paragraphs is specific to PhysiBoSS, but as can be inferred from the wording so far it is relatively easy to implement similar functionality for any other simulator, as the simulator specific parts are detached from anything else in the system. This is a core part of the system as it makes it possible with minimal effort to provide its features to the needs of other researchers.

3.5 Plotting

There exist 3 different plotting options depending on the options provided, available in both 2 dimensions and 3 dimensions. The most important one that has been already mentioned is the plotting of the model and acquisition functions at any given iteration of the process. The main reason for that is that the model plot allows some check of the validity of the results, combined with the acquisition function.

The rest of the plotting options are fairly standard, with the ability to plot the true function, which is usually unknown, if the data points are provided mainly for the purpose of testing. Another option is to live plot the sample points as the Bayesian Optimization process occurs. Lastly there is the option to plot the final result on top of the true function if possible, instead of simply printing it out to standard output.

4 Experimental Results

All of the experimental cases shown below serve the purpose of displaying the capabilities of the system and the method of Bayesian optimization by extension, while also shedding some light on possible deficiencies. Showcased are examples of varying dimensionalities and search spaces, in order to provide a more complete picture. The main objective in all cases is to find the optimal combination of parameter's values that minimize the amount of alive cancer cells. Thus the main parameters of interest here are those pertaining to TNF injections and oxygen.

4.1 Single Parameter Case

In this simplistic and limited case only a single parameter is explored in order to affirm the validity of the results. As the search space is very small it isn't enough to showcase all the benefits of using Bayesian optimization, but nevertheless it is enough to establish some trust in the system's workings.

```toml
[optimizer]
noise = 0
n_calls = 30
n_random_starts = 3
delta_stop = 5

[dispatcher]
simulator = "physiboss"
exec_path = "Workspace/PhysiBoSS/bin/PhysiBoSS"
conf_template = "Workspace/examples/example1/parameters_template.xml"
output_path = "Workspace/examples/example1"

[data]
    [data.x1]
    label = "tnf_conc"
    limits = [0.1, 0.8]
    step = 0.01

[result]
    label = "alive"
```

Listing 4.1: Use case 1 configuration - config.toml

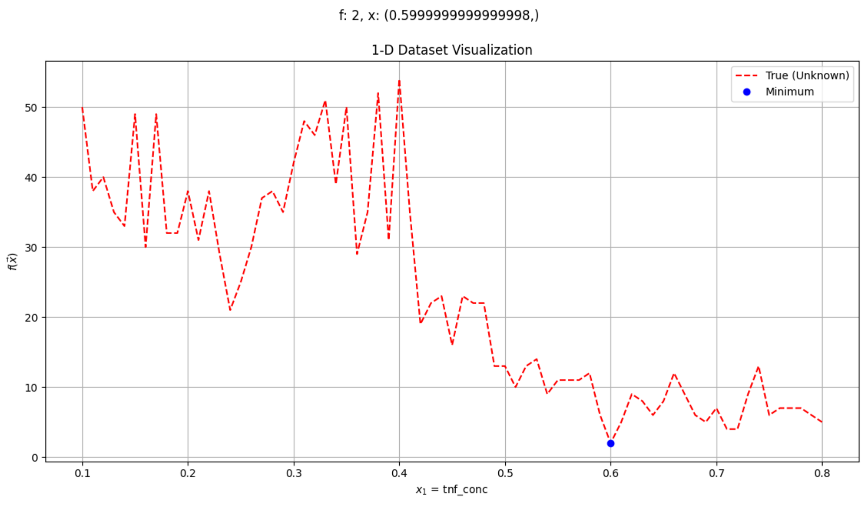


Figure 4.1: Case 1 true function. The true function is normally unknown and only demonstrated here, and in the other cases, in order to compare it with the model created by the process and the resulting minimum found.

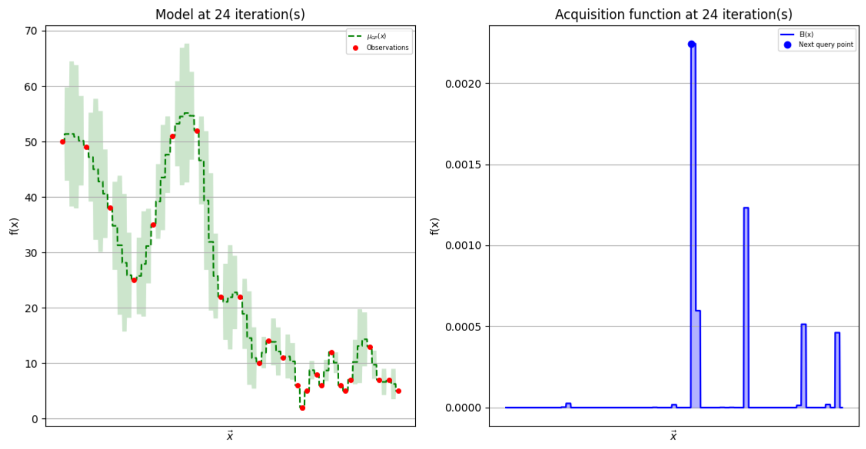


Figure 4.2: Case 1 model. In this and the rest of the examples all models and acquisition functions shown are taken from the final iteration.

Through this example it is shown that the system works as intended, at the very least in this simple case, modeling the unknown function fairly accurately and arriving at the minimum value without having to evaluate every data point. The acquisition function shows where the next data point to be sampled is, but in this particular case the operation was cut short, as the system determined early it was close enough to the optimal configuration.

4.2 Double Parameter Cases

The two cases examined below showcase a more realistic application than the single parameter case. Here the search space is large enough to require significant time and computational power in order to fully explore it, making the use of BO a major need for finding the global minimum in a timely fashion.

4.2.1 TNF Concentration & Time Add TNF

```
[optimizer]
noise = 0
n_calls = 30
n_random_starts = 3
delta_stop = 5

[dispatcher]
simulator = "physiboss"
exec_path = "Workspace/PhysiBoSS/bin/PhysiBoSS"
conf_template = "Workspace/examples/example2/parameters_template.xml"
output_path = "Workspace/examples/example2"

[data]
    [data.x1]
    label = "tnf_conc"
    limits = [0.1, 0.4]
    step = 0.1

    [data.x2]
    label = "time_add"
    limits = [25, 600]
    step = 25

[result]
    label = "alive"
```

Listing 4.2: Use case 2 configuration - config.toml

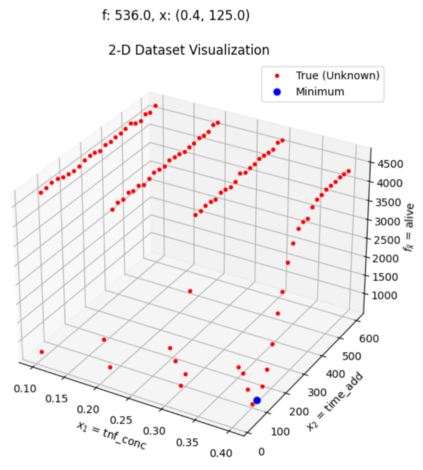


Figure 4.3: Case 2 true function.

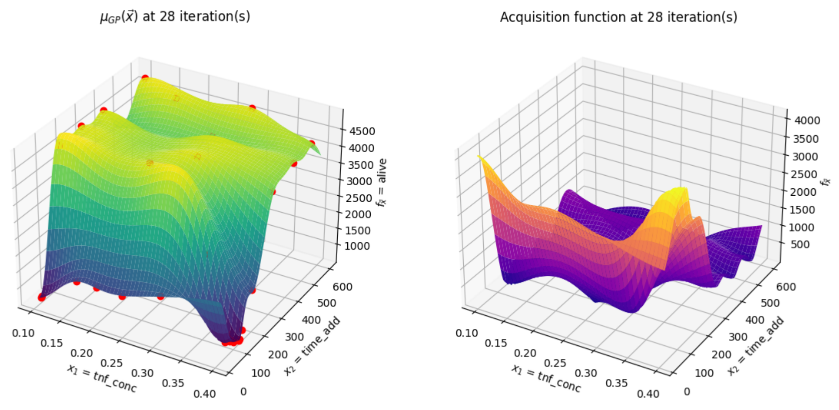


Figure 4.4: Case 2 model.

In this more involved case with two parameters to consider, while the size of the problem space is comparable to the first case, it shown that the system works again as intended. As could be expected and demonstrated by the true function, the number of alive cells is reduced by either increasing the concentration of the dosage or decreasing the interval between doses of the administered drug. The same behavior is demonstrated by the model created by process, culminating in the minimum amount of alive cells when the concentration is at its peak and interval between doses at its lowest.

4.2.2 Duration Add TNF & Time Add TNF

```
[optimizer]
noise = 0
n_calls = 30
n_random_starts = 3
delta_stop = 5

[dispatcher]
simulator = "physiboss"
exec_path = "Workspace/PhysiBoSS/bin/PhysiBoSS"
conf_template = "Workspace/examples/example3/parameters_template.xml"
output_path = "Workspace/examples/example3"

[data]
    [data.x1]
    label = "tnf_dur"
    limits = [2, 10]
    step = 2

    [data.x2]
    label = "time_add"
    limits = [5, 1400]
    step = 5

[result]
    label = "alive"
```

Listing 4.3: Use case 3 configuration - config.toml

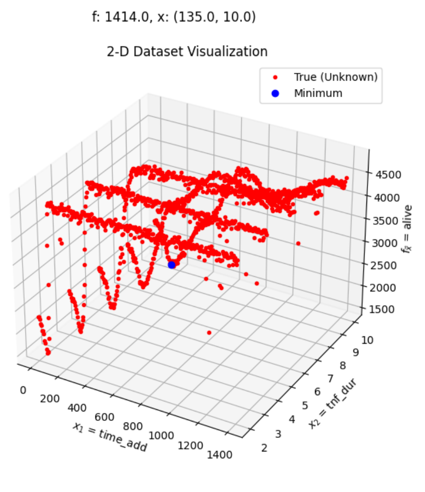


Figure 4.5: Case 3 true function.

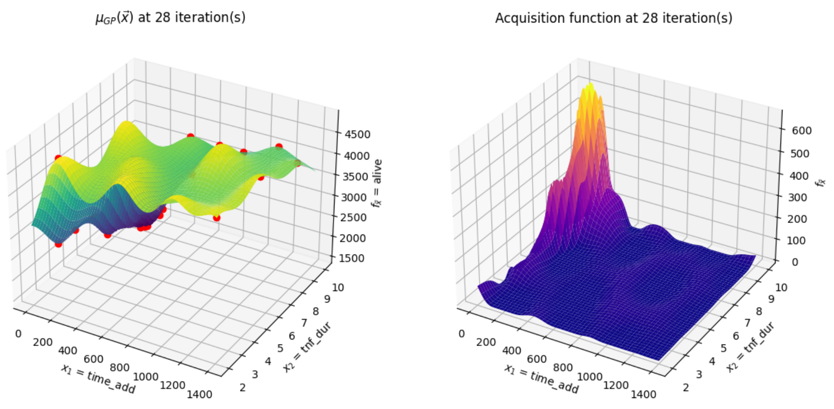


Figure 4.6: Case 3 model.

This case is the first true test of the usefulness of this system. While the previous two cases were more indicative of the validity of the system and its results, in truth with the use of BO on those problem spaces one could save at best a couple of days, this case presents a much larger problem space allowing for BO to reduce what would be weeks or even months of processing time into a day. The results are reassuring, seeing that the model accurately follows the true function and a near optimal configuration is found with the expected number of samples taken. Again, the optimal configuration is as expected, seeing that maximizing the duration of the drug and minimizing the interval between doses produces the least amount of alive cells.

4.3 Quad Parameter Case

The final case is in essence the true stress test of system since four parameters are changing, making the search space considerably larger and more complex than the previous cases.

```
[optimizer]
noise = 0
n_calls = 50
n_random_starts = 3
delta_stop = 5

[dispatcher]
simulator = "physiboss"
exec_path = "Workspace/PhysiBoSS/bin/PhysiBoSS"
```

```
conf_template = "Workspace/examples/example4/parameters_template.xml"
output_path = "Workspace/examples/example4"

[data]
    [data.x1]
    label = "tnf_dur"
    limits = [2, 10]
    step = 2

    [data.x2]
    label = "tnf_conc"
    limits = [0.3, 0.6]
    step = 0.1

    [data.x3]
    label = "oxy_nec"
    limits = [0, 0.01]
    step = 0.01

    [data.x4]
    label = "time_add"
    limits = [5, 600]
    step = 25

[result]
    label = "alive"
```

Listing 4.4: Use case 4 configuration - config.toml

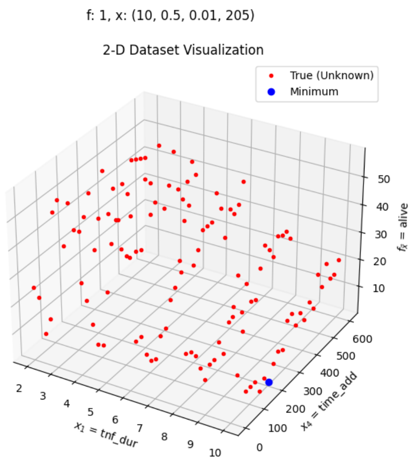
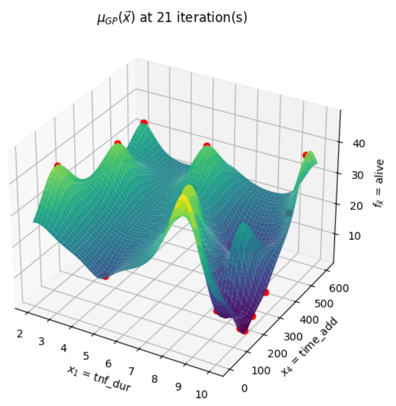


Figure 4.7: Case 4 true function. For this case the above two parameters of the total 4 are chosen to showcase the true function and model as they produce the most data points between them. In addition the other parameters either don't contribute as much to the final results or provide a very drastic between their values making their depiction non instructive.



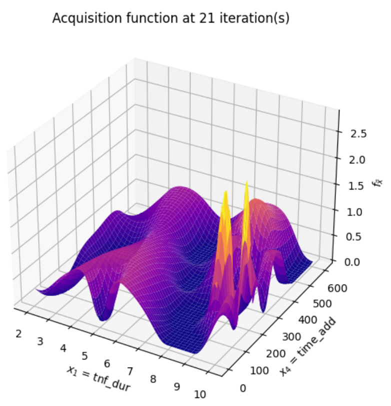


Figure 4.8: Case 4 model. The samples depicted don't contain one or two samples taken with the third parameter at 0 (oxygen necrotic) because the values they produced were much higher than the rest, making the depiction of the model non informative.

As is shown, the system has arrived in a near optimal configuration, even taking less samples than expected to do so. Thus it has been demonstrated that the system works correctly and efficiently even on the more complex and realistic cases. One thing to note is that the requirement for less iterations than expected is partly based on the luck of a particular execution and that by increasing the problem space even further more iterations would generally be required.

5 Conclusions & Future Work

In conclusion, the system presented in this thesis leverages Bayesian optimization in order to find near optimal configurations for long running biological simulations on PhysiBoSS that minimize the number of alive cancer cells. The results are promising, drastically cutting down the number of simulations needed to be run, from possibly thousands to double digits. It also provides major automations in setting up and dispatching those simulations, although some manual setup is required still.

More work could be done in improving the flexibility and extensibility of the system. Although it is possible to substitute PhysiBoSS with a different simulator with relative ease, this process could be further simplified by adding more abstraction layers on the dispatching of simulations.

Another point that could be improved is cutting down the number of simulations needed to be run even further. While part of that is already taken care of by saving simulation results both in volatile and non-volatile storage, more advanced methods and heuristics could be utilized, for example stopping a simulation early after determining that its results are not promising.

References

[1] Omid Alipourfard et al. "CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics". In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, Mar. 2017, pp. 469–482. ISBN: 978-1-931971-37-9. URL: `https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/alipourfard`.

[2] Effrosyni Anesti. "Forecasting Promising Biological Simulations at PhysiBoSS". Diploma Work. Chania, Greece: School of Electrical and Computer Engineering, Technical University of Crete, Feb. 2020. DOI: `10.26233/heallink.tuc.84787`. URL: `https://doi.org/10.26233/heallink.tuc.84787`.

[3] R. Garnett, M. A. Osborne, and S. J. Roberts. "Bayesian Optimization for Sensor Set Selection". In: *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*. IPSN '10. Stockholm, Sweden: Association for Computing Machinery, 2010, pp. 209–219. ISBN: 9781605589886. DOI: `10.1145/1791212.1791238`. URL: `https://doi.org/10.1145/1791212.1791238`.

[4] Roman Garnett. *CSE515T Lecture Notes, Washington University in St.Louis, Computer Science & Engineering Department*. 2015. URL: `https://www.cse.wustl.edu/~garnett/cse515t/spring_2015/files/lecture_notes/12.pdf`.

[5] Ahmadreza Ghaffarizadeh et al. "PhysiCell: An open source physics-based cell simulator for 3-D multicellular systems". In: *PLOS Computational Biology* 14.2 (Feb. 2018), pp. 1–31. DOI: `10.1371/journal.pcbi.1005991`. URL: `https://doi.org/10.1371/journal.pcbi.1005991`.

[6] Gaelle Letort et al. "PhysiBoSS: a multi-scale agent-based modelling framework integrating physical dimension and cell signalling". In: *Bioinformatics* 35.7 (Aug. 2018), pp. 1188–1196. ISSN: 1367-4803. DOI: `10.1093/bioinformatics/bty766`. eprint: `https://academic.oup.com/bioinformatics/article-pdf/35/7/1188/28277837/bty766.pdf`. URL: `https://doi.org/10.1093/bioinformatics/bty766`.

[7] *Matplotlib homepage*. URL: `https://matplotlib.org`.

[8] *Numpy homepage*. URL: https://numpy.org.

[9] *PhysiBoSS GitHub repository*. URL: https://github.com/gletort/PhysiBoSS.

[10] *Python TOML package*. URL: https://pypi.org/project/toml.

[11] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006, pp. 84–85. ISBN: 0-262-18253-X. URL: http://www.gaussianprocess.org/gpml/.

[12] Syusuke Sano et al. "Application of Bayesian Optimization for Pharmaceutical Product Development". In: *Journal of Pharmaceutical Innovation* (2019), pp. 1–11.

[13] *SciKit-Learn homepage*. URL: https://scikit-learn.org/stable.

[14] *SciKit-Optimize homepage*. URL: https://scikit-optimize.github.io/stable.

[15] Gautier Stoll et al. "MaBoSS 2.0: an environment for stochastic Boolean modeling". In: *Bioinformatics* 33.14 (Mar. 2017), pp. 2226–2228. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btx123. eprint: https://academic.oup.com/bioinformatics/article-pdf/33/14/2226/25156782/btx123.pdf. URL: https://doi.org/10.1093/bioinformatics/btx123.

[16] *TOML homepage*. URL: https://toml.io/en.

[17] *TORQUE Resource Manager*. URL: https://adaptivecomputing.com/cherry-services/torque-resource-manager/.

[18] *TUC Grid Computer*. URL: https://www.grid.tuc.gr/index.php?id=1011.