In Silico Evolution of Oscillating Mass-Action Chemical Reaction Networks

Lillian Tatka

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington
2024

Reading Committee:
Herbert Sauro, Chair
Patrick Boyle
Lucian Smith

Program Authorized to Offer Degree:
Bioengineering

University of Washington

Abstract

In Silico Evolution of Oscillating Mass-Action Chemical Reaction Networks

Lillian Tatka

Chair of the Supervisory Committee:

Herbert Sauro

Bioengineering

Computational models are increasingly used in high-impact decision making in science, engineering, and medicine. It is crucial that computational models meet a standard of credibility when using them in high-stakes decision making. For this reason, institutes including NASA, the FDA, and the EMA have developed standards to promote and assess the credibility of computational models and simulations. However, due to the breadth of models these institutes assess, these credibility standards are mostly qualitative and avoid making specific recommendations. On the other hand, modeling and simulation in systems biology is a narrower domain and several standards are already in place. These factors facilitate the development of a quantitative credibility standard. A systems biology credibility standard is essential given the rise in complexity and influence of models. This proposal describes the development of a testing suite to standardize and automate credibility testing and scoring of computational systems biology models. The test suite will include dynamic tests that examine the output of a simulated model, verification and validation tests, and a preliminary assessment of parameter sensitivity to gauge uncertainty in the model's predictions.

Contents

Chapter 1

Introduction

The *in silico* evolution of chemical reaction networks can be a useful tool to create functional networks with specific behaviors as well as to investigate possible pathways of natural evolution. Large sets of synthetic networks can be useful for exploring design patterns and network motifs, gathering statistics network characteristics, benchmarking software, and informing design of synthetic networks. Although several studies use *in silico* evolution to generate networks, the vast majority of them focus on genetic regulatory networks, and almost none of them provide software for others to reproduce their results or use evolutionary strategies in other work. A fast and user-friendly software package would allow researchers to experiment with *in silico* evolution without significant knowledge of programming.

The key objective of this work is to produce a software package that enables the *in silico* evolution of mass-action chemical reaction networks and to explore features and hyperparameters of the evolutionary algorithm that influence the evolution of oscillatory reaction network. The software should be accessible to researchers with minimal programming experience, produce results relatively quickly, and be problem agnostic, meaning that a variety of behaviors could be evolved.

This project began with the creation of a python tool to evolve oscillating chemical reaction networks. Described in chapter 3, this work was intended as a small initial step

towards understanding the function of biochemical oscillators and identifying common design patterns that enable oscillation. Oscillating systems of are of particular interest due to their biological relevance and interesting dynamics and serve as a good candidate problem due to the presence of an intermediate state (damped oscillations). A large population of oscillators was computationally evolved and their reaction characteristics compared to non-oscillating systems. The library of reaction networks generated in this research was used to construct a publicly available database to enable further oscillator/design pattern research as well as to serve as a standardized data set for testing novel software and algorithms.

However, the algorithm used to generate these oscillating networks had its flaws. It took a long time to execute, several minutes per run, and failed to generate oscillators most of the time. Only approximately 5% of evolution trials would result in an oscillator. It took several days of computing time to generate the small database. The most likely cause of this low success rate was the rapid convergence of solutions. The design of the algorithm allowed the most fit reaction networks to dominate the population, even if they were not oscillators or close to becoming oscillators. This dominance would reduce the space for innovation and prevent more fit networks from developing. If a promising candidate network failed to develop in the first few generations, it was unlikely that the evolution trial would be successful.

Additionally, the structure of the code and algorithm did not allow for easy modification and exploration. For example, a user might wish to implement a different selection technique in an attempt to avoid the problem of rapid convergence, but the software was not created to allow for such tinkering. Users could modify some hyperparameters, such as the number of generations and population size, but the fundamental evolution algorithm could not be modified.

To address these shortcomings and explore evolution strategies for mass-action reaction networks, a new software tool was developed, NetEvolve, described in detail in chapter 5. The software is written in the Julia programming language [CITTEEEEE], which uses

just-in-time (JIT) compiling, type stability, and multiple dispatch to gain significant speed advantages over interpreted languages such as Python. The evolution algorithm can be used immediately using default settings or customized using a JSON file. By default, the algorithm evolves oscillators, but users could provide any time series data and the algorithm would attempt to generate networks to match.

To combat premature convergence, NetEvolve separates candidate reaction networks into groups based on similarity. These groups are analogous to species in natural evolution and individuals are compared only against members of the same species. This prevents more fit candidate networks from dominating the population and shelters network innovations that may reduce fitness at first but develop into useful features over the course of several generations.

Inspired by the use of evolutionary algorithms to create artificial neural networks, NetEvolve also implements a form of crossover, a mutation operator that combines features from two "parent" networks to create a new "offspring" network. This feature was shown to improve evolution performance when evolving artificial neural networks and genetic regulatory networks, but has been thought to be disruptive for evolving mass-action networks. The application of crossover to mass-action networks is explored in detail in chapter 5.

- **Aim 1:** Develop a python package to perform dynamic tests on systems biology models in order to assess their credibility

- **Aim 2:** Create tools to automatically verify and validate models

- **Aim 3:** Build software to automate simple parameter sensitivity analyses

This research will result in two first author publications, the titles of which are to be determined. The first publication will describe the design and purpose of the software package and include some preliminary examples of implementation. The second publication will present the full software package and demonstrate its use on several existing models. I hope to finish this work by June 2024.

Chapter 2

Background

This chapter provides context for the major themes of this dissertation. Section 2.1 introduces the basic theoretical concepts behind computational modeling of chemical reaction networks. Section 2.2 seeks to explain the interest in oscillators by describing their history, dynamics, and biological relevance. Finally, section 2.3 provides a brief introduction to evolutionary algorithms, their use in systems biology, and current tools for evolutionary computation.

2.1 Introduction to Chemical Reaction Network Modeling

The projects described here focus on computational models of (bio)chemical reaction networks (CRNs). Although several modeling paradigms exist to describe CRNs, this work uses CRNs represented by systems of ordinary differential equations. A CRN is a set of chemical reactions R_i where i is the index over the range 1 to n, the total number of reactions in the network . Reactions involve the chemical species of the network, S_j where j is the index of

each species from 1 to the total number of species. A reaction is then represented as follows:

$$Ri : \sum_{j \in S} \alpha_{ij} S_j \rightarrow \sum_{j \in S} \beta_{ij} S_j$$

where α_{ij} and β_{ij} are non-negative integers, stoichiometry coefficients. For example, a simple 3-species, 2-reaction network can be represented by the following reactions:

$$S1 + S2 \rightarrow S3$$
$$S3 \rightarrow S1 + S2$$
(2.1)

However, these reactions alone are insufficient to completely describe the small network. Assuming mass-action kinetics, the rate at which a reaction proceeds is proportional to the concentration of the reactants. The constant of proportionality is referred to as the rate constant. To discern the dynamics of the system, the rate constants must be specified.

$$S1 + S2 \rightarrow S3; \qquad k_1 * S1 * S2$$
$$S3 \rightarrow S1 + S2; \qquad k_2 * S3$$
(2.2)
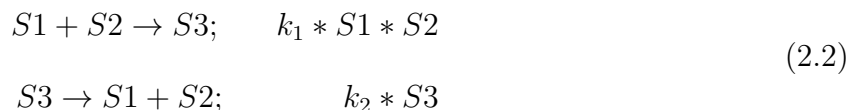
The rate of change of each species can then be put in the form of a system of differential equations. When the initial concentrations of the species are given, this system can be solved numerically to describe the change of species concentrations over time.

$$\frac{dS1}{dt} = k_2 S3 - k_1 S1 S2$$
$$\frac{dS2}{dt} = k_2 S3 - k_1 S1 S2$$
$$\frac{dS3}{dt} = k1 S1 S2 - k_2 S3$$
(2.3)

Chemical reaction network modeling is used across a variety of fields including pharmacology, biological sciences, physical chemistry, chemical engineering, and synthetic biology

CITATIONS. To this end, numerous tools have been developed to aid in model development, description, and simulation. This dissertation introduces an additional software tool for exploring small chemical reaction networks. Of particular interest are chemical oscillators.

2.2 Chemical Oscillators

Chemical oscillators provide a tangible and accessible means to study the dynamics of nonlinear systems. They offer insights into how simple chemical reactions can produce complex, time-varying behaviors, shedding light on fundamental principles of dynamical systems theory. Understanding how these patterns emerge from the collective behavior of individual molecules can inform our understanding of the mechanisms underlying biological oscillations, providing insights into the molecular processes driving rhythmic behaviors in living organisms. This dissertation explores evolutionary algorithms in systems biology with focus on the generation and study of synthetic oscillatory networks.

2.2.1 A Brief History

The first oscillating chemical system was described in 1828. Gustav Fechner described and electrochemical cell that produced an oscillating current. Later, in 1899, Fredriech Ostwald observed that the rate of chromium dissolution in acid periodically increased and decreased. Both these systems were heterogenous mixtures, leading to the belief that homogenous oscillating reactions were impossible, a belief that persisted through much of the 20th century. Their relatively recent discovery and study makes homogenous chemical oscillators a topic of particular interest.

In 1910, prior to leaving science to work for an insurance company, Alfred Lotka authored a monograph on theoretical biology [40]. He showed that a set of consecutive mass-action reactions can create damped oscillators which eventually settle at an equilibrium. A decade later he published a second paper on theoretical mass-action oscillatory systems [42, 41].

Shortly thereafter, physicist Vito Volterra's interest in mathematical biology was piqued by the observation that piscivorous fish had prospered in abandoned fisheries while populations of herbivrous fish remained constant. In 1926, he published a study to explain this phenomena proposing the same equations as Lotka, which he graciously acknowledged when Lotka penned a letter to Nature pointing out his publication of the same equations in 1925 [75, 3]. The Lotka-Volterra model is well-known today as a simple representation of predator-prey interactions, where both populations oscillate in time.

The Lotka-Volterra model consists of three irreversible steps describing the relationships between grass, rabbits, and lynxes. A represents grass, which is assumed to be constant. Rabbits, represented by X, consume grass to grow their population (equation 2.4). Lynx, in turn, consume rabbits to grow their population (equation 2.5) before eventually dying (equation 2.6).

$$A + X \rightarrow 2X \tag{2.4}$$

$$X + Y \rightarrow 2Y \tag{2.5}$$

$$Y \rightarrow P \tag{2.6}$$
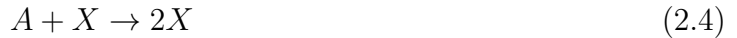
These "reactions" assume "mass-action" kinetics where the rate of each depends on the amount of grass (constant) and sizes of the population of rabbits and lynx respectively. These rates can be described by the following differential equations, where k_y describes the rate of lynx reproduction given a rabbit of population size x, k_d describes the mortality rate of lynx, and k_x describes the rate at which rabbits reproduce.

$$dx/dt = k_x a x - k_y x y \tag{2.7}$$

$$dy/dt = k_y x y - k_d y \tag{2.8}$$

The oscillations in this system result from the time delay between the growth in rabbit

population growth and lynx population growth as well as the delay from rabbit population decline to lynx decline. Rabbits reproduce because grass is in constant supply. As the rabbit population grows, the lynx population follows as prey becomes plentiful. Once the lynx population gets too high, it will begin consuming rabbits at a rate faster than they can reproduce due to the constant supply of grass. Once the rabbit population declines, the lynx population will follow as food grows scarce and they begin to starve. When the lynx population is depleted, there will be less predation pressure and the population of rabbits will begin to grow again. This phenomenom can be observed in nature, as was carefully documented by Charles Elton in 1942 [20]. The predator-prey model has also been demonstrated in the laboratory with paramecia that consume yeast [26].

2.2.2 Dynamics

A key feature of the Lotka-Volterra model, and of most chemical oscillating systems is the presence of autocatalysis. The rate of growth of a population increases as the size of the population increases. In chemical systems, Reactions are considered autocatalytic when the products increase the rate of reaction or when a chemical decelerates the rate of its own destruction [71] The prominence of autocatalysis is also observed in the evolution of *in silico* theoretical chemical reaction networks, as is described in chapter 3 [68].

Long lasting oscillations occur only if the proper feedback mechanism is present [61]. Often the feeback mechanism takes the form of autocatalysis. However, when unchecked, a system with autocatalysis can lead to explosion as the product concentration continues to build and increase the rate of product formation. An inhibition step is necessary to prevent this explosion of products. However, if autocatalysis and inhibition occur simultaneously, the steady state is stabilized and oscillation does not occur. The inhibition step must be delayed. Often this is accomplished by intermediate reactions.

Oscillation can also be achieved with negative feedback [72, 58]. In chemical reaction networks, this takes the form of end-product inhibition, where the final product in a series of
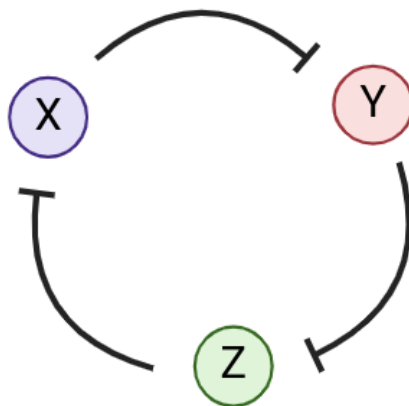
Figure 2.1: caption

reactions inhibits an enzyme from an earlier step in the sequence. A classic example of this is the repressilator, a three gene (or species) loop where each gene expresses a protein that represses the next gene in the loop (figure 2.1) [19]. The invention of the repressilator was a landmark achievement in the field of synthetic biology showcasing the feasibility of engineering complex gene circuits with predefined functions. It provided a foundational framework for the design and construction of synthetic genetic oscillators and inspired subsequent research in the development of artificial gene regulatory networks.

2.2.3 Biological Relevance

Oscillators are biological relevant beyond ecological interactions. A fundamental aspect of living systems, oscillators occur across several levels of biological organization.

The circadian clock exemplifies a well-studied molecular oscillator that regulates daily rhythms in behavior, metabolism, and physiology across organisms [2, 1]. At its core, the circadian clock consists of interconnected transcriptional-translational feedback loops involving clock genes and proteins. These feedback loops take the form of interconnected repressilators [81, 53]. These molecular oscillators synchronize cellular processes with the day-night cycle, optimizing organismal fitness and survival. Disruptions to circadian rhythms have profound implications for health, contributing to various disorders such as sleep disorders, metabolic

syndrome, and mood disorders [2].

Beyond the circadian clock, molecular oscillators operate within cells to regulate diverse processes such as gene expression, signal transduction, and metabolic pathways. Examples include calcium oscillations [62], cyclic AMP (cAMP) signaling [18], and oscillatory patterns of gene expression driven by transcriptional regulatory networks [7]. These intracellular oscillatory systems enable cells to respond dynamically to external stimuli, coordinate complex physiological responses, and maintain cellular homeostasis under changing conditions [8, 35]. Oscillations are also present in systems such as neuronal networks [6], cardiac muscle cells [78, 47], pancreatic beta cells [77], and glycolysis [28], among others, highlighting the importance of molecular oscillators in physiological function.

Synthetic oscillatory systems offer considerable potential for advancing biomedicine and biotechnology. These dynamic systems can serve as biosensors, detecting the presence and concentration of proteins, drugs, and other chemicals [37, 12]. Synthetic genetic oscillators can be employed to modulate the local delivery of effector molecules to tumors [12]. More broadly, synthetic oscillators could serve as essential components for engineering complex synthetic gene and protein networks. These networks could be used for disease diagnosis, pharmaceutical screening, and bio-manufacturing or used therapeutically to modulate endogenous oscillating systems [37].

2.2.4 Maybe a short thing about this research and oscillators?

2.3 Applications of Evolutionary and Genetic Algorithms in Systems Biology

Genetic and evolutionary algorithms are optimization techniques inspired by the principles of natural selection and genetics. These algorithms mimic the process of natural evolution to solve complex optimization problems across various domains. At the core of genetic

algorithms lies the concept of natural selection, a fundamental mechanism driving evolution in biological systems. Natural selection operates on populations of individuals, favoring those with traits that confer a reproductive advantage. Over successive generations, beneficial traits become more prevalent in the population, leading to the emergence of better-adapted individuals.

Evolutionary algorithms offer a computational framework for addressing the inherent complexity and nonlinearity of biological systems, providing a means to explore large solution spaces and optimize model parameters and structures.

The use of evolutionary algorithms in systems biology became popular in the late 1990s, often to better understand metabolic pathways. This approach has been used to demonstrate the optimality of the glycolysis [29, 67] and the Krebs citric acid cycle [46]. In these studies, genetic algorithms were used to optimize reaction rates but not the structure of the networks.

Francois and Hakim evolved both the structure and reaction rates of genetic regulatory networks in an effort to generate small genetic networks that functioned as bistable switches and oscillators [22]. The algorithm used only mutation and selection, avoiding crossover, to minimize the difference between the candidate networks' output and idealized time series data displaying the desired behavior. Francois and Hakim argued that one of their evolved oscillators was structurally similar to the networks responsible for the circadian rhythm. However, Chu extended this work and found no consistent patterns in the evolved oscillators [10]. She noted that it is computationally unfeasible to thoroughly examine all resulting networks, but argued that the similarity to the circadian rhythm was likely coincidental. Despite Chu's study occurring almost 20 years ago, the problem of thoroughly pattern mining evolved networks remains and issue. The results of these works encouraged the exploration of evolutionary techniques on similar problems.

Fujimoto et al. employed a similar strategy to study genetic networks leading to striped patterning and ultimately body segmentation in arthropods [23]. When analyzing the resulting networks, they found three distinct classes, feed-forward loops, feed-back loops, and

interconnections between the two. These three classes reproduced various segmentation strategies observed in arthropods.

Kobayashi et al. also explored the evolution of oscillatory genetic networks, however their algorithm fixed parameter values and only allowed connection rewiring to achieve oscillations with prescribed time periods [39]. Deckard and Sauro expanded this concept beyond genetic regulatory networks to evolve small networks with specific computational capabilities such as square root and cube route calculators [13]. As with the previously mentioned algorithms, their algorithm avoided crossover relying solely on mutation to develop candidate solutions. Attempts to use crossover did not result in fitter offspring, possibly due to combining heterologous networks and thus disrupting developing solutions. This work was continued by Paladugu et al seeking to generate mass-action network functional modules with specific behaviors, including oscillators [50]. This algorithm avoided crossover as the authors argued that it would be disruptive.

Marchisio and Stelling argued that brute force optimization algorithms lack efficiency and that implementing some element of rational design could speed up automatic design of genetic networks implementing boolean logic gates [44]. They separate structural and parameter optimization by first generating several possible circuits and then ranks feasibility based on the structure. Then the best solutions undergo parameter optimization. Separating structural design from parameter optimization certainly saves computation time, but it relies on the existence of a rational solution, and is better suited for genetic networks, where these attributes are more easily separated. For more complex tasks, a solution might depend on a precise combination of both structure and parameter.

Other approaches fix structure and make use of evolutionary algorithms to explore the parameter space while fixing network topology. Jin and Sendhoff used an evolution strategy to explore parameters of regulatory motifs with fixed structures [34]. Porubsky and Sauro use an evolutionary algorithm to search for parameters that yield oscillations in mass-action networks of fixed structure [55].

Most applications of evoltuionary algorithms in systems biology avoid the use of crossover as it tends to disrupt candidate solutions. For example, Drennan and Beer successfully evolved a repressilator (a specific type of oscillatory reaction netowrk [19]) using a genetic algorithm that included crossover [16]. Instead of evolving sets of differential equations, Drennan and Beer encoded their genetic regulatory networks using strings of 4 different letters, analogous to DNA encoding. Knabe et al. used a similar approach to evolve oscillators that interact with the environment [38]. This approach allows for the straightforward implementation of crossover, but limits the space of possible models. Due to this encoding, there was a high probability of breaking connections making it difficult to pass on structural innovations.

Stanley and Miikkulainen addressed this problem in 2002 with NEAT (NeuroEvolution of Augmenting Topologies), a powerful algorithm used for evolving artificial neural networks (ANNs) [66]. Unlike traditional methods of neural network training, which typically involve adjusting the weights and biases of fixed network architectures, NEAT evolves both the structure and weights of neural networks simultaneously. A two key features of NEAT are the use of a more meaningful structural crossover technique, and speciation, which protects innovations and allows them time to develop by having candidate solutions compete only against similar individuals instead of the entire population. Dinh et al. adapted this technique to genetic regulatory networks and found improved performance in evolving oscillators with the use of crossover.

2.4 Tools for Evolutionary and Genetic Algorithms

Although there are many tools that enable evolutionary algorithms and simulating chemical reaction networks, there are none that facilitate evolutionary algorithms using chemical reaction networks. This can likely be attributed to the challenge of encoding chemical reaction networks in a way that allows for easy modification and re-simulation, a requirement

of evolutionary algorithms. Most of the existing tools require advanced programming skill to use them.

2.4.1 Python Libraries

Most packages for implementing genetic and evolutionary algorithms are written in python [74]. Python is one of the most ubiquitous programming languages frequently topping lists of the most commonly used languages. It is often used in scientific applications due to it being open source, easy to use, and having vast library support. Likely for these reasons, most existing evolutionary algorithm tools are python libraries.

Although likely not an exhaustive list, the main python packages for evolutionary and genetic algorithms are described here.

DEAP

In 2012, Fortin et al. released the python package DEAP (Distributed Evolutionary Algorithms in Python) [21]. The packages offers a toolbox of evolutionary algorithm features including different types of crossover, evolutionary strategies, and mutation operators. Unlike other frameworks DEAP makes algorithm explicit and data structures transparent, as opposed to black-box frameworks. Although DEAP allows for extensive customization, individuals are represented as arrays of values. Users can customize these arrays, such as specifying integers or floats at specific locations, but individuals must be in this form. For most optimization problems, this representation is adequate, however it is difficult to fully represent a chemical reaction network in this form. Reaction networks have varying numbers of reaction, so a stoichiometry matrix would be inadequate unless it contained a row for every possible reaction. Additionally, reactions require rate constants, and networks require initial values for each species.

But beyond the encoding issue, the DEAP package, while highly versatile and proven in many research application, requires extensive programming knowledge. For example to build

a population, a user must: (1) register the data type for each gene, (2) register an individuals that combines these gene data types, (3) register a population that uses the individual, and finally (4) build the population. Running an entire genetic algorithm requires far more additional programming.

Pyevolve

Pyevolve is more simple framework for genetic algorithms compared to DEAP but has limited features [52]. Despite its relative simplicity, it still requires extensive programming knowledge to implement an evolutionary process and problems are still limited to those whose "genes" can be encoded as arrays of numbers.

Pyevolve also requires significantly more programming to accomplish the same task comapred to DEAP. DEAP requires 59 lines of code to solve the OneMax problem[1], compared to 378 for pyevolve [11].

The most recent release was 2014 and the library is no longer maintained.

EasyGA

EasyGA is a more recent (2021) python tool for genetic algorithms [79]. EasyGA makes it relatively easy to implement simple genetic algorithms, but still requires the user to have some advanced programming knowledge. The simplicity of easyGA makes it useful to only selected problems due to gene encoding restrictions.

The library has a limited number of genetic operators. Additionally, EasyGA randomly mutates both parents and offspring, a feature that goes against the basic premise of genetic algorithms.

At the time of writing, the EasyGA GitHub repository only had 47 stars, suggesting that it is infrequently used.

---

[1]The OneMax problem is a classic problem in the field of evolutionary computation. It is a simple bit sting problem where the objective is the maximize the number of ones in a binary string. The fitness of each individual is the total number of ones in its string. The simplicity of the OneMax problem makes it an ideal benchmark for testing optimization algorithms

LEAP

LEAP (Library for Evolutionary Algorithms in Python) is another recently developed library published in 2020 [11]. LEAP was designed to simplify the process of implementing evolutionary and genetic algorithms. However, LEAP has some core design weaknesses that impact usability.

Despite one of the objectives of LEAP being to simplify the process of implementing genetic algorithms, it still requires the user to write the evolution loop resulting in more code to solve simple problems. The user also has to manually increment the generation counter by calling a function specific to this purpose. Failing to call the function results in an infinite evolution loop, a problem that may not be immediately identifiable and solvable by less experienced users.

Additionally, LEAP has no means of tracing the results of each generation of evolution. This makes troubleshooting more difficult and renders LEAP an ineffective tool for those who may wish to study the evolution process and not just the final results.

LEAP is also a relatively unpopular choice for implementing evolutionary algorithms with only 77 stars at the time of writing.

PyGAD

In 2020, pyGAD was released with the aim to simplify the implementation of genetic algorithms for users with less programming experience while simultaneously proving control over "everything possible" [24]. PyGAD requires 3 core steps: (1) build the fitness function, (2) instantiate the relevant genetic algorithm class, (3) call the run function. The operation of pyGAD is relatively simple. Some programming knowledge is required for the user to create a fitness function and implement any further customization, but the bulk of the algorithm is run automatically.

As with the other software tools described here, the main drawback of pyGAD for applications in systems biology is the restrictions on encoding individual solutions. Solutions

must be encoded as a 1D vector of numbers. This is insufficient for evolving chemical reaction networks, where a solution is an entire reaction network and all parameters needed to simulate the network.

EvoX

Most recently, Huange et al. released EvoX, a python library for evolutionary computing [33]. Unlike other libraries described here, EvoX is optimized for distributed GPU acceleration. This improves scalability and performance while remaining in the Python ecosystem.

2.4.2   Libraries in Other Languages

Python is one of the more commonly used languages for scientific computing, however other languages offer various advantages. C++ and Java offer high performance compared to interpreted languages, but tend to have steeper learning curves and are less common in scientific computing. R excels at statistical computing but is less versatile than python. MATLAB is less common outside of academia likely owing to its high licensing costs. Julia is a relatively new language (2012) and has less package support and a smaller community than python. Likely for these reasons, fewer evoltuionary packages exist in other programming languages and are less likely to be used by scientists. Notable non-python packages are briefly reviewed here.

¡¡Note about the gene structure issue for all of these¿¿

Evolutionary Computation in Java

Evolutionary Computation in Java (ECJ) is a framework that supports several evolutionary computation techniques including genetic algorithms and differential evolution [43]. It offers a high level of flexibility and customization and is well-suited for large-scale optimization problems. ECJ has a steeper learning curve compared to other frameworks, especially for

those not familiar with Java programming. Java is less frequently used in scientific programming making ECJ beyond the reach of many computational biologists.

JMetal

Jmetal is a well-documented Java library offering a range of evolutionary algorithms and metahueristics for multi-objective programming [17]. It provides good performance and scalability for large-scale optimization problems. However, it requires knowledge of Java programming and a deeper understanding of the framework in order to customize it beyond the built-in algorithms.

Bio-Inspired Algorithms Building Blocks

Bio-Inspired Algorithms Building Blocks (openBEAGLE) is a library providing evolutionary computation tools and components [25]. Written in C++, it offers good performance for intensive tasks. However, it's documentation and community support are limited compared to more widely used frameworks and it may have a steep learning curve for those not familiar with C++.

Many-Objective Genetic Metric-based Algorithms

Many-Objective Genetic Metric-based Algorithms (MAGMA) is a library for many-objective optimization problems with a range of performance metrics for evaluating algorithms [76]. It is written in MATLAB making it more accessible to many scientific users, however this limits performance for large-scale optimization problems compared to compiled languages.

Genetic and Evolutionary Algorithm Toolbox for use with MATLAB

Genetic and Evolutionary Algorithm Toolbox for use with MATLAB (GEATbx) seamlessly integrates with MATLAB's core functionalities, making it easy to implement and test evolutionary algorithms within the MATLAB environment [54]. MATLAB's rich visualization

capabilities allow users to easily visualize optimization processes and analyze results, aiding in understanding and debugging algorithms. GEATbx can be easily integrated with other MATLAB toolboxes for tasks such as data preprocessing, feature selection, and model evaluation, facilitating end-to-end optimization workflows. While MATLAB provides a wide range of functions for implementing evolutionary algorithms, the level of customization may be limited compared to lower-level languages or more specialized frameworks, particularly for advanced users requiring highly customized algorithms or optimization processes.

Genetic Algorithms

Genetic Algorithms (GA) offers a simple and intuited interface for implementing genetic algorithms in R, making it accessible to users with varying levels of programming experience [59, 60]. It has a range of customization options, including different selection methods, crossover and mutation operators, and termination criteria, allowing users to tailor algorithms to their specific problem domains. GA supports parallel execution, enabling users to leverage multi-core processors for faster execution of genetic algorithms, particularly useful for computationally intensive tasks. The package is actively maintained and updated by the R community, ensuring compatibility with the latest versions of R and incorporating new features and improvements over time. While suitable for many applications, R may not offer the same level of performance as lower-level languages like C++ or Java, especially for large-scale optimization problems or computationally intensive tasks.

GALGO

Like the GA package, GALGO integrates seamlessly with the R programming language, allowing users to leverage the rich ecosystem of statistical and machine learning libraries available in R [69]. GALGO offers a variety of evolutionary algorithms beyond genetic algorithms, including differential evolution, particle swarm optimization, and simulated annealing, providing users with flexibility in choosing the most suitable algorithm for their optimization

19

problem. The package allows users to customize various aspects of the evolutionary algorithms, such as population size, crossover and mutation operators, and termination criteria, enabling tailored solutions for different optimization tasks. Like GA, GALGO may suffer from performance limitations compared to compiled languages.

Evolutionary.jl

Evolutionary.jl offers a user-friendly interface for implementing evolutionary algorithms in Julia, making it accessible to users with varying levels of programming expertise [80]. The package provides a range of evolutionary algorithms, including genetic algorithms, differential evolution, and particle swarm optimization, among others, allowing users to choose the most suitable algorithm for their optimization problems. Evolutionary.jl allows users to customize various aspects of the evolutionary algorithms, such as population size, crossover and mutation operators, and termination criteria, enabling tailored solutions for different optimization tasks. Julia is designed for high-performance numerical computing, and Evolutionary.jl benefits from Julia's performance advantages, providing efficient execution of evolutionary algorithms. However, this package is not suited for evolving chemical reaction networks due to the limitations of encoding solutions.

Chapter 3

A Public Database of Evolved Oscillatory Reaction Networks

3.1 Background

With the constant emergence of new software tools and techniques in systems biology, access to a large variety of chemical reaction network models with defined characteristics can be useful for testing and validation. For example, an existing data set enumerating small chemical reaction networks [14] has been used to test computational methods to analyze bistability [51], software to reverse engineer networks [48], and a novel framework to assess networks for multiple equilibria and other characteristics [15]. Generating these test data sets can be computationally expensive and time-consuming. The data set generated by Deckard et al. took several days to compute and contains approximately 47 million small models, but its purpose is to exhaustively enumerate small networks, not to assess and catalog their behavior [14]. Despite the usefulness of standard validation data sets, currently no public database exists containing numerous models displaying a variety of network behaviors.

In terms of network behavior, oscillatory chemical reaction networks are of particular interest due to their biological relevance, including embryogensis, DNA repair, and heart

function [49, 4, 27, 73]. Lotka et al. documented the first ordinary differential equation model of an oscillatory predator-prey network [40]. The study of oscillatory mechanisms in chemical reaction networks dramatically increased with the expansion of computing power and the improved ability to solve non-linear differential equations [30]. Novak and Tyson studied non-autocatalyic small oscillators to determine design patterns and basic requirements for oscillating systems [49]. These works focus on the mechanisms and characteristics of individual oscillators or to oscillators as a broad category. Paladugu et al. used in silico evolution to create several oscillators, bistable switches, homeostatic systems and frequency filters, but the library was not made publicly available for further study [50].

We present a public database, entitled "Cesium", containing approximately 1800 3-species, 450 6-species, and 750 10-species oscillating networks created using a customized evolution algorithm, an optimization process based on the iterative improvement of candidate models. There are also random non-oscillating networks to serve as controls. The database can be searched by the number of reactions, the presence of autocatalytic reactions, and the number of species. Each entry contains the above information as well as the Antimony string [63] of the model which can be simulated by any software capable of reading antimony or SBML (to which Antimony models can be converted). Future work will expand this database to include a wider variety of oscillatory models as well as models that display other behaviors such as bistability.

The population of evolved oscillatory reaction networks possessed different reaction type compositions compared to non-oscillating networks that were randomly generated. Oscillating networks had significantly more uni-bi reactions and autocatalytic reactions.

The purpose of this research is two-fold: basic research into characteristics of oscillatory systems, and the development of this database.

The availability of a standardized database for use in software and algorithm testing pushed me in the direction of model credibility. This is a step beyond reproducibility, the goal of which is the easy reproduction of a model and its results by third party users.
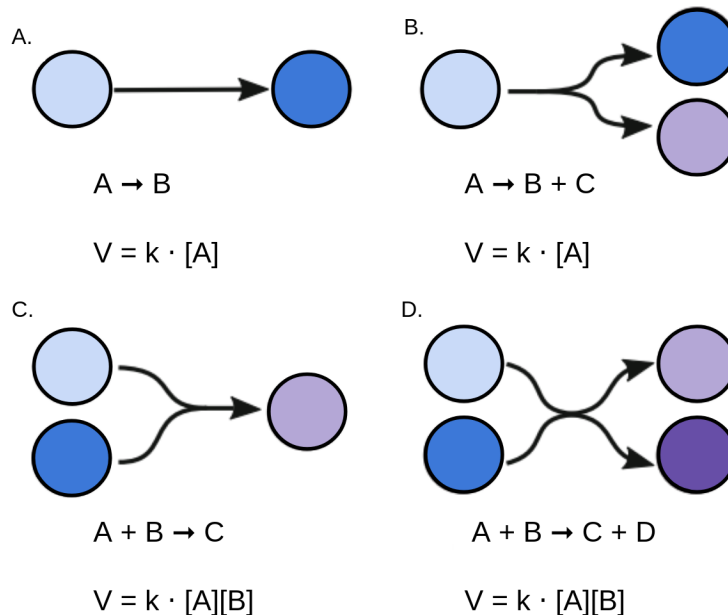
Figure 3.1: Models were composed of four reaction types and rate laws: (A) uni-uni, (B) uni-bi, (C) bi-uni, (D) bi-bi

Credibility refers to the trustworthiness of a model. This topic and related research will be covered in later chapters.

3.2 Methods

Oscillating models were generated using evolution scripts written in python, available at https://github.com/sys-bio/evolution. The process begins by randomly generating a population of models and gradually modifying them over time to produce oscillatory behavior. Four types of reactions are used: uni-uni, uni-bi, bi-uni, and bi-bi, all with mass-action kinetics (figure 3.1). At each step, every model is evaluated against an objective function scoring the model's ability to oscillate. Models with better fits are selected and further modified and models with lower fits are gradually eliminated.

3.2.1 Model Representation

Models and reactions were represented by custom data structures. An instance of the *reaction* object represents a single reaction and consists of one or two reactants, one or two products, a rate constant, and an integer representing the reaction type (uni-uni, uni-bi, bi-uni, or bi-bi). The *model* object represents a single reaction network and consists of a list of species, a list of initial concentrations, and a list of *reaction* objects. The software converts *model* objects into systems of ordinary differential equations which are then numerically solved to produce time series data of species concentrations. After the evolution process is complete, the models are converted to antimony [63], a human readable model definition language.

3.2.2 Objective Function

The objective function minimized error between the candidate model's time course data and a series of points corresponding to the peaks and troughs of an oscillator. This "idealized" oscillator time series consisted of nine concentration points, alternating between 5 and 30 concentration units over the course of 1.25 seconds (figure 3.2). The output time series of each species was compared to these points by summing the squared difference between the observed point and the "ideal" point and the smallest error value was considered the model's fitness. This is similar to MSE but the sum is not divided by the number of observations as this number is constant across all models. This has been demonstrated to be an effective objective function for the in silico evolution of chemical oscillators [50]. In cases where candidate models could not be simulated, an arbitrarily high fitness value was assigned resulting in their subsequent removal from the population in the selection step.

3.2.3 ODE Solver

Preliminary studies suggested that the most commonly used python ODE solver from scikit was inaccurate for solving these problems. Other off-the-shelf solvers lacked the ability to
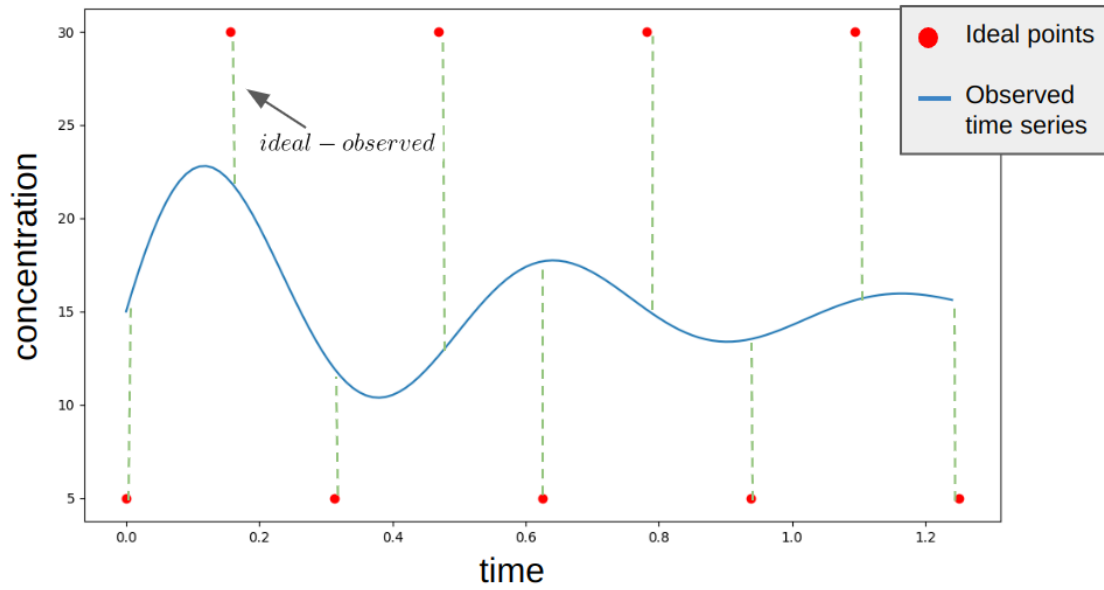
$$fitness = min(\sum(ideal - observed)^2)$$

Figure 3.2: For the time series data of this species, the error is calculated by summing the square difference between the observed time series data (blue) and a set of idealized oscillator points (red). This process is repeated for each species in the model and the fitness is defined as the minimum of these sums.

deal with the custom data structure encoding the models. The Sauro Lab's simulation software, RoadRunner [65], was also unsuitable for this purpose due to the computational cost of frequently modifying and recompiling models for simulation. Many algorithms, such as RK4 are not robust enough for this work. For this reason, a custom simulator using the CVODE solver from the Sundials suite was used [31].

3.2.4 Selection

In each new generation, the top 10% of models from the previous generation were copied without modification. The remainder of the new population was chosen by tournament selection [45] from the previous generation. This is where two models are randomly chosen from the previous generation (including from the top 10% of models already carried over) and the model with the better fitness is mutated by adding or deleting a reaction, or by modifying a rate constant. The modified model is then appended to the new generation.

3.2.5 Mutation

After tournament selection, the fitter model was either mutated by modifying a rate constant or adding/deleting a reaction with equal probability. In the case of rate constant modification, a random rate constant was adjusted by a random percentage between -15% and +15% of the rate constant's current value. This mechanism ensured that rate constants could not become negative, and there was no upper limit to their value.

In the case of reaction modification, a reaction was either added or deleted with 50%-50% probability. If deleted was selected, a random reaction was removed from the model. If addition, a new reaction was added with the equal probability of each reaction type: uni-uni, uni-bi, bi-uni, bi-bi (figure 3.1).
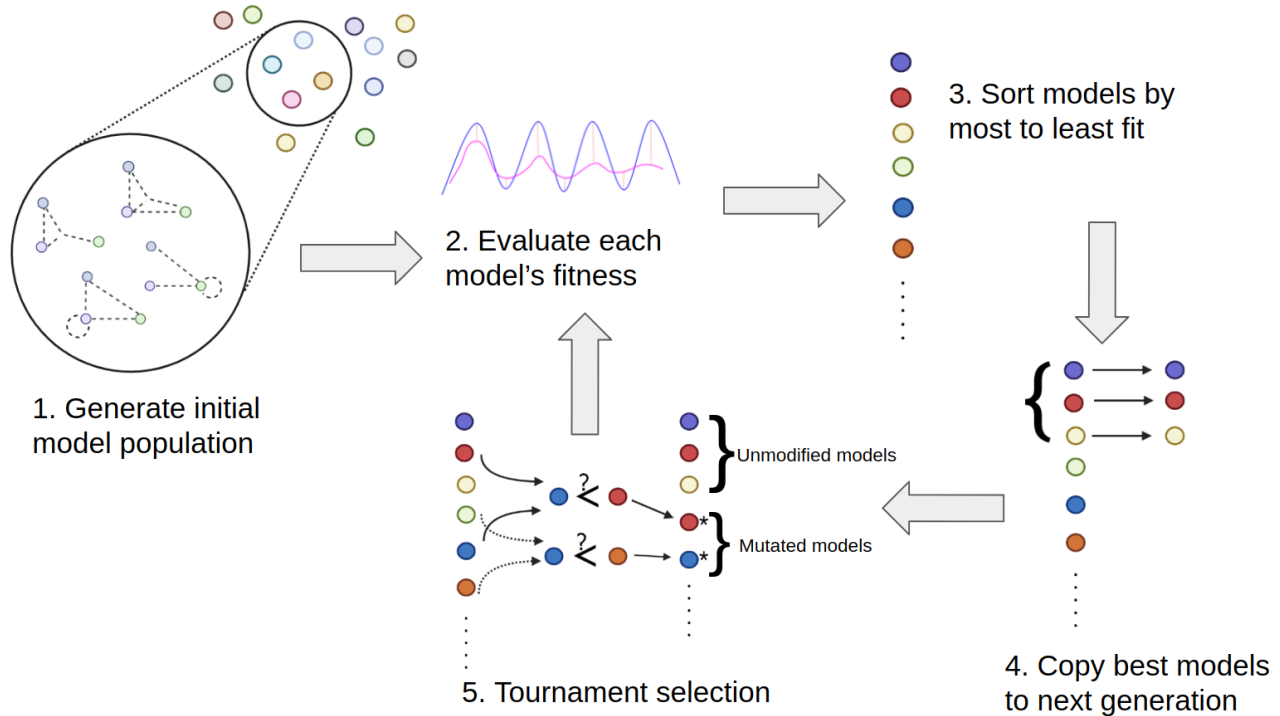
26

Figure 3.3: The evolution algorithm. (1) An initial population of random models is created. (2) The fitness of each model is evaluated. (3) The entire population is sorted by most fit to least fit. (4) The top 10% of the models are transferred to the subsequent generation unmodified. (5) Tournament selection is used to populate the remainder of the subsequent generation. Models are randomly selected, the more fit model is chosen to be modified and carried over to the subsequent generation. Steps 2-5 are repeated.

3.2.6 Random Networks for Evolution

A population of 40 random networks were generated using the `teUtils` python package [57]. Each network was initialized with three species, and nine reactions with the probability of each reaction type being 0.1, 0.4, 0.4, 0.1 for uni-uni, uni-bi, bi-uni, and bi-bi reactions. These settings were chosen to maximize the number of evolution trials that successfully product oscillators. All reactions had mass-action kinetics with a random rate constant between 0 and 50.

3.2.7 Custom Evolution Algorithm

The evolutionary algorithm mimics biological evolution in that populations of individuals, in this case candidate networks, are altered and forced to compete with each other. Over the course of generations, fitter individuals (models that oscillate or are likely to oscillate) out compete unfit individuals (models that do not oscillate or can not be simulated) and survive into the next generation. Although genetic and evolutionary algorithms are well characterized, their application to systems biology models remains a challenge as a key trait in these algorithms is crossover, the ability of two possible solutions to exchange information, creating a new, ideally more suitable, candidate solution [36]. Typically, genetic algorithms operate on objective functions for which solutions consist of vectors where values can easily be exchanged between two vectors. It is uncertain how crossover of mass-action kinetic models could be achieved as solutions (candidate networks) consist both of topology (how reactions are connected) as well as rate constants (vectors of values). Additionally, the number of reactions and rate constants vary and must be equal but can vary from model to model. For this reason, a custom optimization process was developed based on genetic algorithms but avoiding crossover.

This process begins with the generation of 40 random networks with pre-specified probabilities for each of the four reaction types. The fitness of each model is evaluated by comparing the model's time series data with an objective function representing the desired outcome, oscillation. Models that with time series data close to this desired outcome, those that oscillate or those with damped oscillations, are more fit than those that fail to oscillate or cannot be simulated.

These models are ordered from most to least fit and the top 10% of the models are carried over unmodified to the subsequent generation (figure 3.3). The remaining models are randomly paired and compared and the fitter of the two models is slightly modified by either adding or deleting a reaction or changing a rate constant. If the modification improves the fitness of the model, the modified model is added to the next generation. If

the modification makes the model less fit, 75% of the time the unmodified model is added to the next generation and 25% of the time the less fit model will be added. This allows for the chance that small changes initially make a model worse, but subsequent changes drastically improve the model. Once the new generation is fully populated, the models are again ordered from most to least fit and the process begins again. This is repeated for 400 generations or until a threshold fitness level is reached.

3.2.8 Random Control Networks

Random models were generated as described in the previous section. To control for changes made during evolution, the random models underwent the same mutation processes described previously. Instead of populating subsequent generations based on model fitness, 10% of the previous population was randomly selected to be carried over unmodified to the new generation. The remainder of the new generation was populated with models randomly chosen and mutated from the previous generation. The purpose of this procedure was to account for model variability introduced in the evolutionary process. Of 1000 random control networks generated, 1 was a spontaneous oscillator.

3.3 Results

Evolved models are processed to remove any undesirable oscillators from the population, namely oscillators that dampen over time or oscillators where one or more species concentrations rise indefinitely towards infinity. Next, duplicated reactions were fused and reactions that are superfluous to oscillation are deleted to minimize network size resulting in a population of reaction networks that oscillate indefinitely in species concentration, examples of which are shown in figure 4.
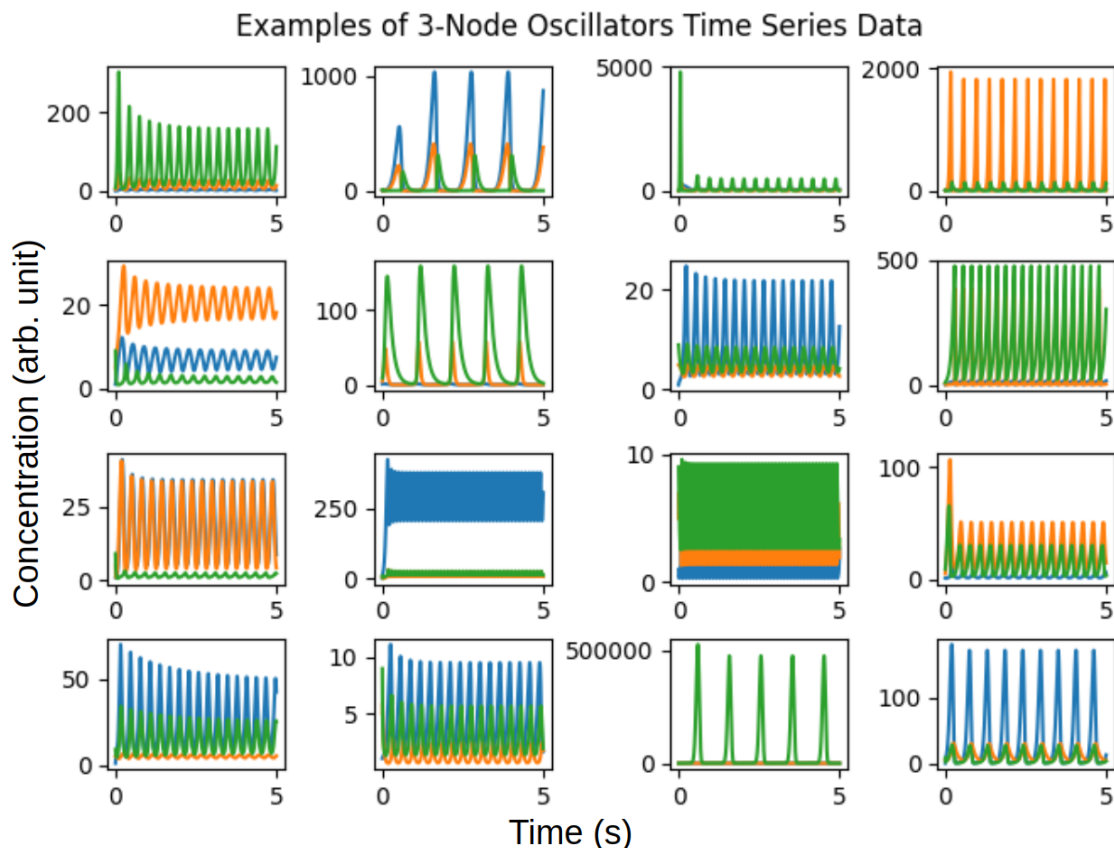
Figure 3.4: Time series data of sixteen 3-node oscillators generated using differential evolution.

3.3.1 Database Construction

A non-relational key-value database was deployed via Mongo Atlas using the PyMongo API to store all models and their information. The user enters the desired reaction network attributes into the web GUI (figure 3.5). Models are downloaded as a .zip file of text files, or if a single model a single text file, containing the antimony string of each model. If the "Download in simulatable Tellurium form" box is checked, then each model file will contain the necessary python package imports and formatting to be immediately simulated and plotted upon running the file.

The current website queries the database by model type (current options are oscillator or random), the number of nodes (species), the number of reactions, and the presence or absence of autocatalysis or degradation reactions. Both oscillator and non-oscillating control

networks are stored in the database. Each entry also includes a dictionary of tallies of each reaction type in the network, a list of redundant reactions that were fused during post-processing, a list of reactions that were deleted as their presence did not influence oscillation.

In addition to the Antimony string comprising the model, other characteristics such as its initial reaction probabilities, fused and deleted reactions, and reaction counts are also tracked in the database. The database can be queried to select models with any number of desired traits. Both oscillator and non-oscillating control networks are stored in the database.

The Cesium database is publicly available on the web at `https://cesiumdb.herokuapp.com/` and is intended to serve as source of reaction networks with specified characteristics for use in research and validation. It currently contains 2000 randomly generated models and approximately 1800 3-species oscillating networks. This database can be expanded in the future to include a wider variety of oscillatory networks as well as networks with different behaviors, such as bistability.

3.3.2 Oscillating Network Examples

Four oscillating networks from the database are shown in figure 3.6. Arrows symbolize reaction and lead from the reactant to the product. A double headed arrow indicates two products are formed and a double tail indicates two reactants.

In network A, S1 is autocatalytic as is S0 (and catalyzed by S1). These linked positive feedback loops cause oscillation with products out flowing to S2, which essentially behaves as a boundary species, a species that is unaffected by the model and whose concentration remains fixed. Network B lacks autocatalytic reactions completely. Computing the jacobian

31

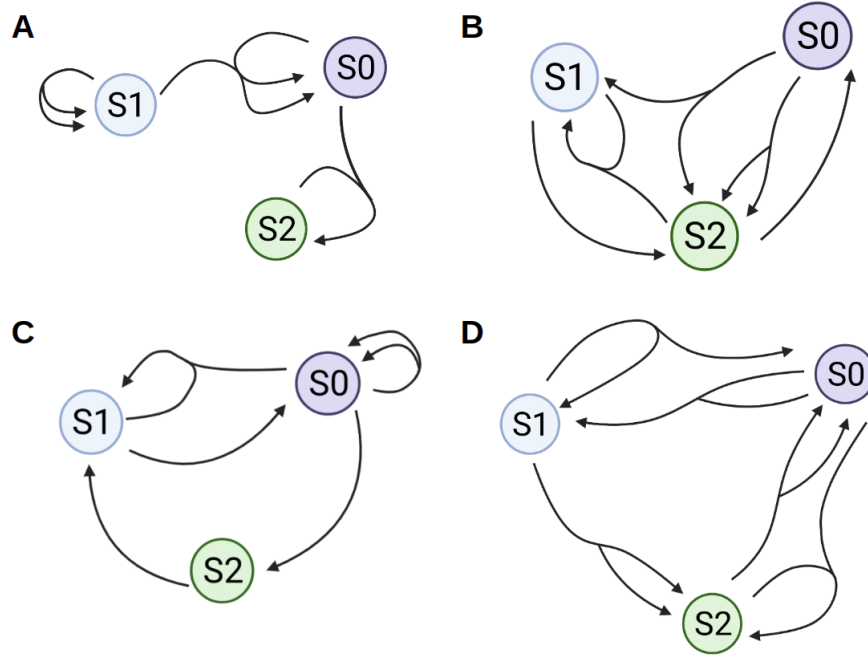Figure 3.5: Landing page for the Cesium database.

Figure 3.6: Examples of 3-species oscillating networks from the Cesium database.

matrix at the unstable focus results in the following matrix:

$$
\begin{array}{ccc}
S0 & S1 & S2
\end{array}
$$
$$
\begin{pmatrix}
-4.75 & 0 & 15.2 \\
0.75 & -0.3 & 0 \\
8.75 & -1.6 & -22.8
\end{pmatrix}
\begin{array}{c}
S0 \\
S1 \\
S2
\end{array}
$$

In the bottom row center, the negative number indicates that there is an inhibitory relationship between S1 and S2. An increase in S2 will decrease the production rate of S1. These interactions for a cycle with negative feedback, suggesting that this is a feedback oscillator (figure 3.7). Similar analyses reveal that networks C and D are also likely to be feedback oscillators.
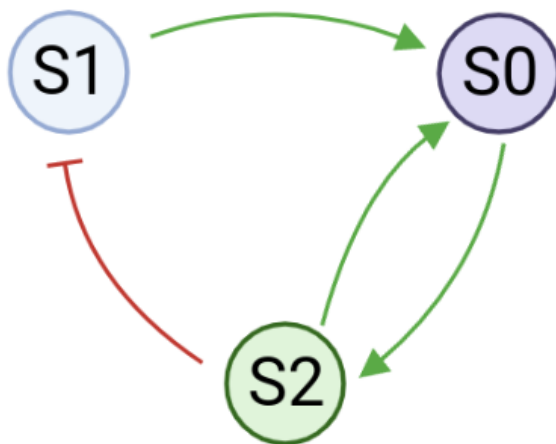
Figure 3.7: Species interactions of Network B. Green arrows indicate activation and blunt red arrows indicate inhibition. The interactions form a cycle with negative feedback suggesting that Network B is a feedback oscillator.

3.4 Discussion

Oscillating models were significantly enriched for autocatalytic reactions compared to control networks. Most oscillators are associated with unstable steady states. Destabilizing processes can be classified as 1) direct autocatalysis, 2) indirect autocatalysis (a positive feedback loop), and 3) end-product inhibition (a negative feedback loop) [72, 70]. Reactions are considered autocatalytic when the products increase the rate of reaction or when a chemical decelerates the rate of its own destruction [71]. Given the two types of oscillators, negative feedback and negative feedback coupled with positive feedback [58], it is unsurprising that autocatalytic reactions are enriched in the oscillator population as these reactions are a form of positive feedback. The portion of models containing at least one autocatalytic reaction in the oscillator and control populations were compared using a two-way chi square test. Of the 586 oscillating models, 83.1% (487 of 586) contained an autocatalytic reaction compared with 49.8% (498 of 1000) of the control models, a significant difference ($p < 0.0001$).

Given the enrichment of autocatalytic reactions, one might expect a similar enrichment of degradation reactions, reactions where one species is removed from the system (eg. $X + Y \rightarrow$

Y, where X is removed from the system), to prevent species concentrations from rising to infinity. Interestingly, the presence of degradation reactions were not enriched in oscillating models. Of the oscillating model population 79.4% (465 of 586) contained at least one degradation reaction compared to 76.3% (763 of 1000) of the control models, an insignificant difference suggesting that although autocatalysis is a common characteristic in oscillating networks, degradation reactions are not necessary to prevent species concentrations from rising to infinity. Although the presence of both an autocatalytic and a degradation reaction are not necessary for oscillation, it is rare that an oscillator contains neither. Only 0.5% (3 of 586) of the oscillators analyzed had neither an autocatalytic reaction nor a degradation reaction compared to 0.2% (2 of 1000) random models, an insignificant difference.

To determine if any reaction types were enriched in the oscillator population compared to the control population, the average model compositions were compared. The average model composition was assessed as the average number of reactions of the specific type were divided by the average total number of reactions for each group. For example, there were an average of 6.592 reactions per model in the oscillator population, of which an average of 0.99 were uni-uni reactions, for an average of 15.3% uni-uni reactions in the average oscillator model (Table 3.1). Compositions were compared with permutation tests, showing that model compositions and sizes were significantly different between the oscillator and control populations (the reduced size of oscillating networks can be accounted for by the fusion of duplicate reactions). Oscillators possessed significantly more uni-bi reactions compared to random control models. This is consistent with the enrichment of autocatalytic reactions which are often uni-bi. However, autocatalytic reactions can also be bi-bi reactions, which were not enriched in oscillating models. Although bi-bi reactions were less likely to be created during evolution due to the initial settings (10%), it is somewhat surprising that bi-bi reactions were decreased in oscillatory networks given that a bi-bi reaction is slightly more likely to be autocatalytic ($\frac{4}{27}$) compared to a uni-bi reaction ($\frac{1}{9}$).

Composition of Oscillator vs. Control Models

	Uni-Uni	Uni-Bi	Bi-Uni	Bi-Bi	Mean Number of Reactions
Oscillator	15.3%	40.7%	26.6%	17.4%	6.592
Control	16.6%	29.7%	30.1%	23.5%	9.425

Table 3.1: Average reaction compositions of oscillating networks compared to non-oscillating controls.

Next, oscillators containing autocatalytic reactions (498 models) were compared to oscillators lacking them (98 models). Populations were compared by permutation test. There was a significant increase in the portion of uni-bi reactions and a significant decrease in the portion of bi-uni reactions in non-autocatalytic models as compared to autocatalytic oscillators (Table 3.2). This result is interesting given that autocatalytic reactions are either uni-bi or bi-bi and both reaction types were reduced in oscillators with autocatalytic reactions compared to oscillators without. It is possible that oscillators without a single autocatalytic reaction are achieving autocatalysis through a combination of non-autocatalytic reactions.

Composition of Autocatalytic vs. Non-autocatalytic oscillators

	Uni-Uni	Uni-Bi	Bi-Uni	Bi-Bi	Mean Number of Reactions
Autocatalytic	15.1%	40.0%	27.6%	17.2%	6.727
Non-Autocatalytic	15.9%	44.8%	21.1%	18.1%	5.918

Table 3.2: Average reaction compositions of autocatalytic oscillating networks compared to non-autocatalytic oscillating networks.

Several manually inspected oscillators had reactions with extremely high rate constants (greater than 300, whereas most rate constants were between 5 and 75). The initial range

for rate constants is 0 to 50 and with each mutation, they can only increase a maximum of 15% of the current value. Although there is no upper limit for rate constants, it is nearly impossible to mutate a rate constant from the initial range to over 100 during the course of evolution. To bypass this limit and achieve high rate constants, many successful models have simply duplicated reactions during the evolutionary process. During post-processing, these duplicate reactions are fused and their rate constants summed. Fusing duplicate reactions to achieve high rate constants accounts for the observation that oscillators generally have fewer reactions than non-oscillators in this study. These high rate constant reactions can not be removed, nor can the rate constant be significantly lowered without impacting oscillation. Further investigation is needed to determine what essential function these high rate reactions seem to play in most of the oscillators included in this study.

These oscillatory models and the random control networks have been added to a database, accessible at `https://cesiumdb.herokuapp.com/`. Models can easily be selected by the number of species, the number of reactions, and the present of autocatalysis or degradation reactions. The selected models can be downloaded in a zip file containing a .txt for each model. Each .txt file contains an individual model's antimony string. If the "Download in simulatable Tellurium form" option is selected, the .txt file will also contain formatting and package imports allowing the model to be easily simulated when the file is run as a python script.

In the future, this database will be expanded to contain a variety of models with different behaviors besides oscillation. It is our hope that this database serves as a resource for the modeling community to study network behaviors and test novel software.

Funding

Acknowledgements

Chapter 4

Parking lot for abandoned text that might be useful later

Applying GAs to problems of parameter optimization is fairly straightforward as "chromosomes" can be sequences of values, easily mutated and crossed over with other strings of values. This approach becomes more complex when applied to network problems, where network topology is deeply intertwined with parameter values. In these problems, the challenges becomes innovating while preserving decent candidate solutions. At its core, NEAT employs a genetic algorithm to evolve populations of neural networks. Each individual in the population represents a neural network with its own unique topology (i.e., arrangement of nodes and connections). NEAT starts with a population of simple neural networks, often with minimal structure, and evolves them over generations to solve a given task.

One of the key innovations of NEAT is its ability to handle the complex task of evolving neural network topological structures. NEAT accomplishes this by employing three main mechanisms: speciation, historical marking, and structural innovation. Speciation encourages diversity within the population by comparing individuals against similar individuals as opposed to the entire population. This shelters innovations that may be weak at first, but will develop into robust solutions over several generations. A historical record of topological

innovations serves as a "chromosome" and enables meaningful crossover without computationally expensive matching procedures required in other algorithms. Lastly, NEAT allows for the introduction of new structural innovations in neural networks through mutation. New nodes and connections can be added to existing networks, providing the potential for incremental growth and improvement over successive generations.

4.1   Existing Software

Although there are numerous studies implemting genetic algorithms in systems biology applications, few software packages exist to aid in this pursuit. In 2008, Kratz and Krishna developed GeNESis, a C based software with GUI to enable evolution of gene regulation networks. Its purpose was to help researchers undersatnd the evoltuionary behavior of populations of genetic regulatory networks. Connections were represented by binary strings, enabling the use of crossover. The URL to this tool is currently inactive.

Chandran and Sauro devloped a C library with basic functions for evolution of biological networks including genetic networks, protein networks, and mass-action networks. Although highly versatile, the library requires some C programming knowledge in order to customize the evolutionary algorithm.

Beyond python, R and MATLAB are languages commonly found in science applications. Julia has fewer users and less package support, but this can likely be attributed to it's relative newness. Julia was first introduced 2012, a new language compared to MATLAB (1984), Python (1990), and R (1993). R is widely used in statistics applications. MATLAB is commonly used in academia, but less often in industry applications owing to its expensive license.

Chapter 5

In Silico Speciated Evolution of Oscillatory Mass-Action Reaction Networks

5.1   Introduction

Genetic algorithms (GAs), inspired by the principles of natural selection and genetics, have emerged as a powerful computational technique for solving optimization and search problems.Developed by John Holland in the 1970s, GAs mimic the process of biological evolution to solve optimization and search problems [32]. At their core, genetic algorithms operate on a population of potential solutions, represented as individuals or "chromosomes," each encoding a candidate solution to the problem at hand. Through iterative generations, genetic algorithms apply mechanisms of selection, crossover, and mutation to evolve and refine the population, gradually converging towards optimal or near-optimal solutions. Selection favors individuals with higher fitness, mirroring the process of natural selection, while crossover and mutation introduce variation and diversity into the population, allowing for exploration of the solution space. By leveraging the principles of evolution, genetic algorithms offer a

versatile and robust approach to solving complex problems across various domains, from engineering and optimization to biology and beyond.

This work describes the adaptation of the NEAT algorithm to mass-action chemical reaction networks and explores the utility of crossover, speciation, and other hyper-parameters in evolving oscillatory networks. A software package to systematize the study of hyperparameters is introduced. This package, written in the julia programming language [5], simplifies the customization of evolutionary algorithms. It has broad applications as it can be run from the command line and allows users to specify settings in a json file. It is capable of matching any time series data.

5.2   Evolving Reaction Networks

The purpose of this work it twofold: (1) to create a general purpose, easily customizable, module for evolving mass-action chemical reaction networks and (2) to explore the effects of crossover, speciation, and other hyperparameters on the success rate of the algorithm. Oscillatory chemical reaction networks were chosen as the target result of the evolutionary algorithm due to their biological relevance, poorly understood architectures (maybe??), and the presence of intermediary states (damped oscillations) which aids in gradual evolution. —-

Oscillatory behavior is useful in a variety of biological processes including p53, circadian rhythm, and cell cycle. But also, you might want to generate oscillators as a component in a syntehtic biology ciruit. But really, this isn't about oscillators, that's just the test case. You might want a certain behavior and don't know how to build a network to get it. Well This algorithm is for you!

Other people have done similar things, but this is different. It starts with the concept of genetic algorithms, way back in 1975. These are algorithms to come up with optimization solutions. The algorithms kind of look like biologic evolution. You have a bunch of

individuals, the fitter individuals share their genes and reproduce, less fit individuals die off, sometimes mutations occur and these can be either good or bad. Since then, the concept of genetic algorithms has been thoroughly explored, but here we present something slightly new. And that's how I'm going to get my PhD lol.

This all starts with the NEAt algorithm which is a way of evolving neural networks to do stuff. What's interesting about this is they claim to have meaningful crossover and they use speciation to protect innovation. Dinh et al somehwat adapted this appraoch to genetic circuits, but those are a much easier problem because they just have activation and inhibition. Here, I adapt this algorthim to mass-action networks. That's way harder because there is way more complexity. In the first case, you just have an edge that connects two nodes and a weight. In the second, you add that the edge can either be activating or inhibiting, but here we add that the edge could connect multiple nodes (in the case of everything beyond a uni-uni reaction) and the reaction weight depends on what nodes it connects. More on that later.

I need to go over a bunch of times that this was done for a similar thing. Just as a quick review

———————————————— Some review stuff

Evolutionary design of oscillatory genetic networks (Kobayashi 2010) Wanted to evolve genetic oscillatory networks. Get a large population of them and then mine them for statistics about oscillators and network motifs. They have fixed parameter values though and only change connections (?). Looks like they are also trying to evolve oscillators with specific frequencies. They have a cost function which looks at periodicity
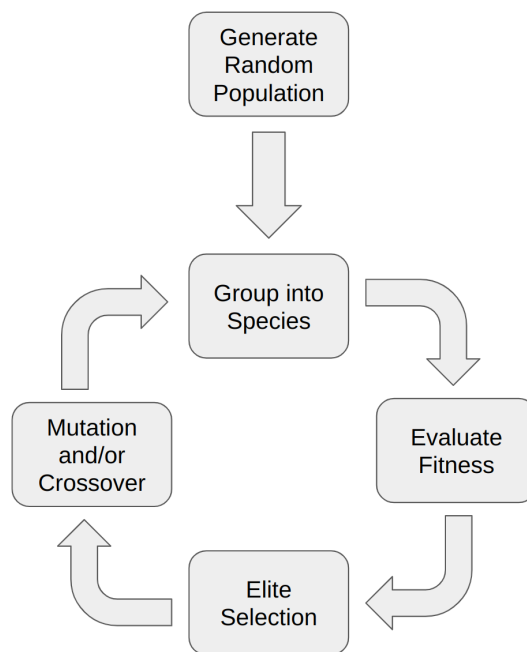
5.3 Methods

Figure 5.1: The evolutionary algorithm begins with a population of randomly generated reaction networks. These networks are then grouped into species and their fitness evaluated. The next generation is constructed by first selecting the top performing individuals from each species without modification. The rest of the subsequent generation is populated by networks that are modified by either crossover, mutation, or both. This new generation is then grouped into species based on structural similarity and the process repeats.

5.3.1 Encoding

Chemical reactions are represented as custom data structures that contain one or two reactants, one or two products, a rate constant, and a boolean value indicating if the reaction is active or inactive. Four types of mass-action reaction are possible: uni-uni (a single reactant to a single product), bi-uni (two reactants form a single product), uni-bi, and bi-bi. A reaction that is active participates in the network that it is part of. An inactive reaction is "turned off" but remains part of the network as a historical record. It can be activated again or crossed over under certain circumstances.

A reaction network is a custom data structure consisting of several reactions, initial concentrations for the floating species, and other information to track the individual network.

A network can include boundary species (which are assumed to be at a constant concentration throughout the simulation), but boundary species were not used in this study.

5.3.2 Mutation

There are two types of mutation: rate constant mutation and reaction addition or subtraction. The probability of each of these mutations occurring and the range in which rate constants are mutated can be set by the user. A rate constant mutation can occur in two possible forms. The rate can be increased or reduce by a random percentage, uniformly distributed in a custom range (by default $\pm 20\%$), or more rarely a completely new rate constant can be selected, uniformly distributed within a custom range (by default 0.1 to 50). Rate constants were not allowed to become negative, but they did not have an upper bound as previous studies observed that many synthetic oscillating systems rely on a single reaction with an abnormally large rate constant [68].

A reaction mutation is either the addition of a new random reaction or the deletion of an existing reaction, with 50% chance of each. When a new reaction is added, it is generated randomly with a probability of 0.1, 0.4, 0.4, 0.1 for uni-uni, uni-bi, bi-uni, and bi-bi reactions respectively. These probabilities are also configurable. If the randomly generated reaction is already present in the reaction network, the new reaction's rate constant is added to the existing reaction's rate constant. This feature can allow some rate constants to grow to several multiples of the next largest rate constant in the network. When a reaction is deleted, it is "switched off" and becomes inactive in the network, but is preserved as it can become reactivated under certain circumstances.

5.3.3 Crossover

During crossover, two parent networks from the same species group are chosen to mate. If a reaction is present in both parents, the reaction will be passed down to the offspring network with the rate constant randomly chosen from one parent. If a reaction is present in the more

fit parent, but not the less fit parent, the offspring network will inherit the reaction. If an inherited reaction is currently inactive meaning it was previously deleted during a mutation step), there is a 0.25 probability of the reaction becoming active again. If both parents are equally fit, one is randomly chosen to be the more fit parent.

5.3.4 Speciation

In many cases, modifying a network, especially through the addition or deletion of reactions, initially makes the system less fit. Without speciation, these innovations seldom last for more than a generation. However, often these topological innovations prove an essential step in evolution [66]. For this reason, a speciation strategy is employed to protect innovations and allow them time to optimize. Individuals with similar topologies are grouped into species. Members of a species compete only with each other and not the population at large. A compatibility distance metric, δ, is used to assign reaction networks to a species, defined as

$$\delta = \frac{M}{N} \tag{5.1}$$

where M is the number of reactions in the larger network that are not present in the smaller network and N is the total number of reactions in the larger network. Two reactions are considered identical if they have the same reactants and products. Rate constants are not considered. If δ is below the speciation threshold, δ_t, the two networks are members of the same species.

For each new generation, networks are assigned to species based on the previous generation's species groups. Each new network is compared to a randomly chosen individual from a species in the previous generation. If δ is less than the speciation threshold δ_t, then the network is assigned to that species. If δ is greater than δ_t, then the network is compared to a randomly selected individual from the next species, etc., until a match is found. If no species is assigned after comparing the network to all species in the previous generation, then a new

46

species is created.

The speciation threshold δ_t is adjusted with each generation to steer the total number of species towards the target number of species. The target number of species is 10 by default, but can be configured by the user. After each generation the total number of species is counted. If there are more species than the target value, δ_t is increased allowing networks that are more dissimilar to be grouped together. If there are fewer species than the target value, δ_t is decreased. Adapting δ_t as evolution proceeds maintains an optimal number of species. If there are too many species each with a small number of members, there are not enough opportunities to optimize solutions in a small parameter space. Similarly, if there are too few species with several members, the entire parameter space is not adequately explored.

$$\delta_t = \begin{cases} \delta + \epsilon, & \text{if number of species} > N_s \\ \delta - \epsilon, & \text{if number of species} < N_s \end{cases} \tag{5.2}$$

5.3.5   Objective Function

The objective function evaluated how well a candidate model oscillated at the imposed time period, T. Two arbitrary concentrations, $[C]_1$ and $[C]_2$, were chosen *a priori*. The fitness score of a particular network was then attributes to how well any of its chemical species approached $[C]_1$ at half periods ($T/2$, $3T/2$, $5T/2$,...) and $[C]_2$ at each period (T, $2T$, $3T$,...) over the course of 5.5 periods. The candidate model's single chemical species that best approached these points was selected and the fitness was evaluated as the maximum of the reciprocal of the absolute difference between the timeseries data and $[C]_1$ or $[C]_2$, the "ideal oscillator" time points, (equation 5.3). The reciprocal was taken so that higher values represented better fitness, which is necessary for determining the number of offspring for each species in subsequent steps. This objective function has been shown to be an effective means of evolving oscillators in similar and previous works [50, 22, 68]. In cases where candidate models could not be simulated, a fitness of 0.0 was assigned resulting in the candidate model's

subsequent removal from the population.

$$fitness = max(\frac{1}{\sum |ideal - candidate|}) \tag{5.3}$$

5.3.6 ODE Solver

During the fitness evaluation phase, reaction networks were converted to systems of ordinary differential equations. These equations were then numerically solved using the DifferentialEquations.jl julia package [56] and the CVODE solver from the Sundials suite for solving initial value problems [31]. The CVODE solver was shown to be effective for this application in previous work [68].

5.3.7 Reproduction

Each model species was allotted offspring in proportion with its fitness. More fit species were allowed more offspring. To calculate the number of offspring allotted to each species group the sum of fitness values was computed across the entire population. Then each species, s, was assigned an offspring number proportional to its contribution to the fitness of the entire population (equation 5.4). The calculated number of offspring for each species was rounded to the nearest integer. By default, a single species was only allowed to produce at most 10% of the subsequent generation in order to prevent a single species from taking over the entire population (but this value could be adjusted by the user). For these reasons, the total number of individuals fluctuated slightly over time.

$$offspring_s = \frac{\sum fitness_s}{\sum fitness_{pop}} * populationsize \tag{5.4}$$

The reproduction phase began with an elitism selection strategy. The top 10% of networks in each species were copied to the next generation directly. If a species had 10 or fewer individuals, the single best network was copied without modification to the next generation.

Then, the bottom 10% of networks in each species are deleted. For the remainder of the allotted offspring for each species, a random number, p, was generated. If the p was less than the probability of crossover, then two networks from the same species were selected and crossed over to produce a single offspring network. Otherwise a single network was chosen. If p was greater than or equal to 1 - the probability of mutation, the network was mutated. Outside of the elite networks, all networks were either crossed over or mutated. Some were both crossed over and mutated. The probabilities of crossover and mutation could be configured by the user. Probabilities for specific types of mutations are described in section 5.3.2.

Tournament selection was also explored and can be configured in settings. In this case, two networks from the same species were randomly chosen and the more fit network was then crossed over, mutated, or both. If crossed over, a second network was also chosen by tournament selection.

5.3.8 Output

The evolutonary aglorithm was repeated for a preset number of generations (800 by defualt) or until the best model reached a predefined fitness level. At this point, the best model from each species was converted to Antimony, a human readable model definition language [63] and written to a text file along with the model's fitness. All settings were also written out to a JSON file, which could be used without modification to repeat the evolution trial.

The best models from each evolution trial were automatically assessed with a Python script using the Tellurium modeling environment [9] and RoadRunner simulation software [64]. A model was considered an oscillator if its Eigen values at steady state contained a positive real number with a nonzero imaginary number and if all concentrations were positive at steady state. Models that met the Eigen value criteria but had negative steady state concentrations were flagged. Flagged models then underwent a simple automated repair process wherein each reaction was deleted one at a time in an effort to meet the oscillation

criteria. Models that met criteria after the simple repair process were counted as oscillators.

5.4 Results

The software was evaluated with multiple hyperparameter settings to assess its ability to generate oscillatory mass-action chemical reaction networks. Each set of hyperparameters were evaluated over 700 evolutionary trials and the success rate recorded. A trial was deemed successful if it generated a network with sustained oscillations, as evaluated by the criteria described previously.

5.4.1 Speciation Maintains Diversity and Increases Success Rate

In order to test the effects of speciation on population diversity and oscillator success rate, evolution was run without speciation, and with 10 and 30 as the target number of species. When speciation is disabled, the top 10% of the entire population is passed on to the next generation unmodified and the bottom 10% of the entire population is deleted. Then mutation is performed on the entire population to populate the remainder of the subsequent generation (crossover was not performed for these experiments). Each network must compete against every other network in the population. To account for fluctuations in population size when speciation is enabled, diversity was measured as the number of unique networks over the total number of networks in the population. A network was considered unique if differed by one or more reactions from all other networks. Reactions are considered different if they differ in product or reactant. Rate constants are not considered in these comparisons. Significance levels were calculated using two-tailed t-tests.

In trials where no speciation was allowed, the average portion of unique networks was 0.228 ± 0.026 across 800 generations. This is significantly ($p < 0.0001$) less than the portion of unique networks in evolution trials with a target number of species of 10, 0.298 ± 0.033 (figure 5.2).
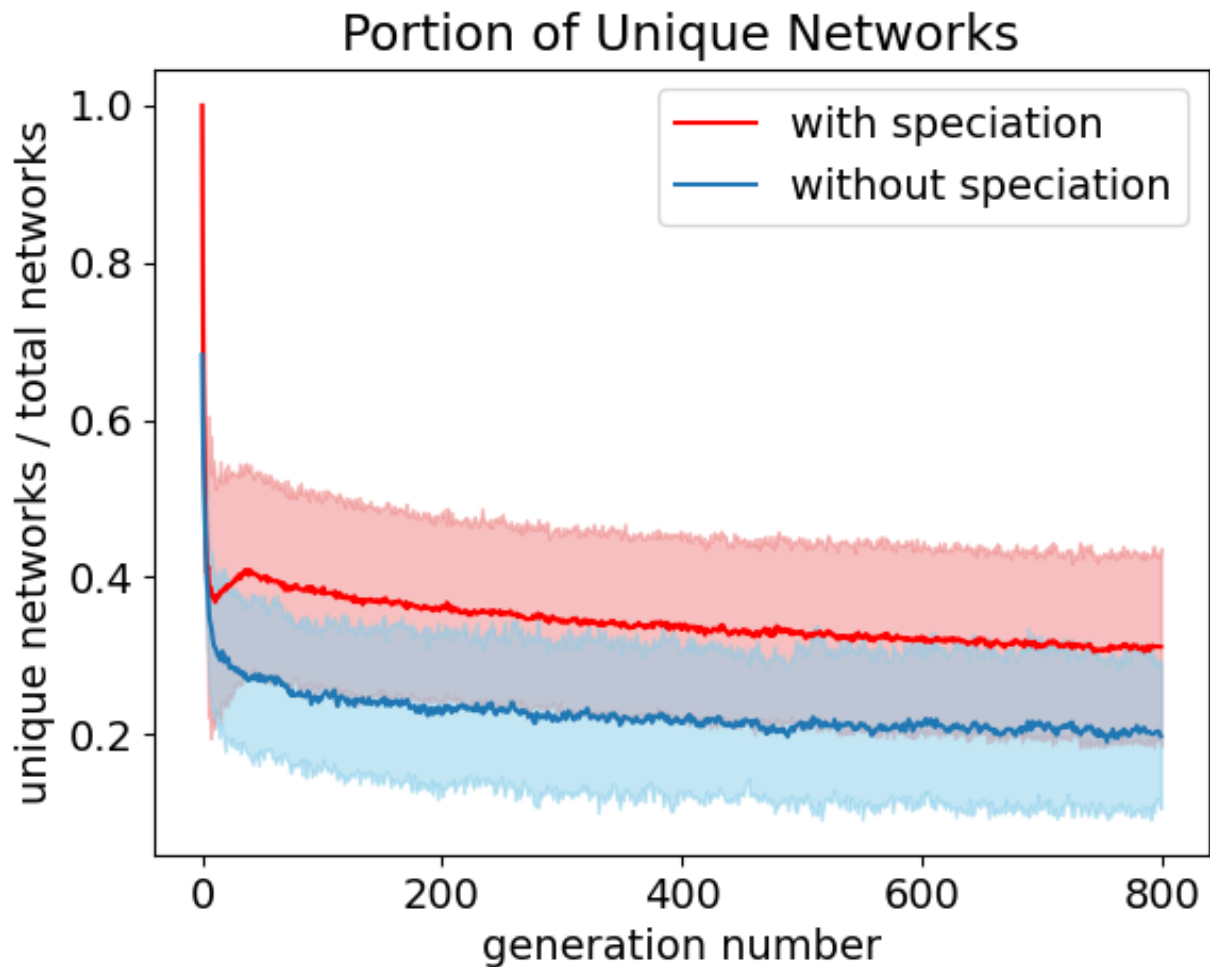
Figure 5.2: The average portion of unique networks over 800 generations with speciation (left), 0.0298 ± 0.033, and without (right), 0.228 ± 0.026. Dark blue line represents the average portion for each generation and light blue shading is the 95% confidence interval.

One purpose of speciation was to prevent the most fit network from overtaking the population over time. When a single network dominates the population, there are not enough unique networks to adequately explore the solution space and the evolution algorithm converges on a solution prematurely. To account for fluctuating population sizes when speciation is allowed, the number of copies of the best network divided by the total number of networks was measured at each generation with speciation (with a target species number of 10) and without speciation. When speciated was enabled, the portion of the population occupied by copies of the most fit network remained constant with an average portion of best networks

of 0.071 ± 0.008. Without speciation, the portion of best networks rose gradually over time with an average portion of XXX ± XXX(figure 5.3). This shows that speciation effectively prevents a single network from dominating the population (p <XXX).
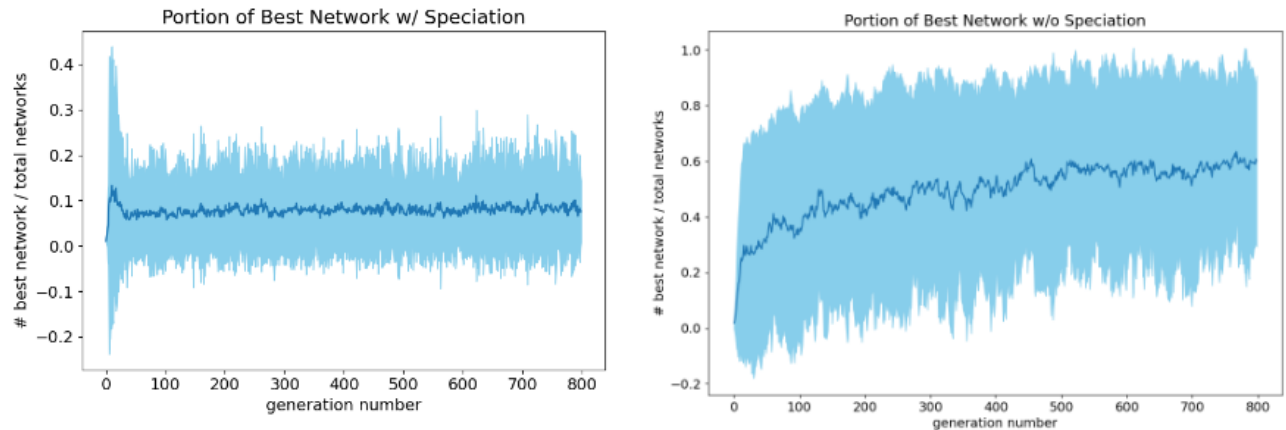

Figure 5.3: The average portion of best networks over 800 generations with speciation (left), 0.079 ± 0.011, and without (right), 0.495 ± 0.094. Dark blue line represents the average portion for each generation and light blue shading is the 95% confidence interval.

Next, the effects of speciation on evolution success rate was evaluated comparing no speciation to speciation with a target species number of 10 and 30. Again, crossover was omitted from these studies. For each speciation level, a batch of 700 trials of evolution were run and the number of oscillators resulting from each batch. Significance levels were calculated using a binomial distribution.

With a target species of 10 (the default condition), evolution trials resulted in sustained oscillators at a rate of 0.19 (134/700). Without speciation, the success was significantly reduced (p <0.0001), with oscillators occurring at a rate of 0.06 (44/700). TO more thoroughly evaluate the benefits of speciation, a batch of 700 evolution trials was run with a target species number of 30. This resulted in a success rate of 0.17 (117/700). This rate is not significantly different (p = 0.1127) than the rate with a target species number of 10. Speciation increases the success rate of evolution, but grouping networks into more species with fewer members does not improve success (figure 5.4).

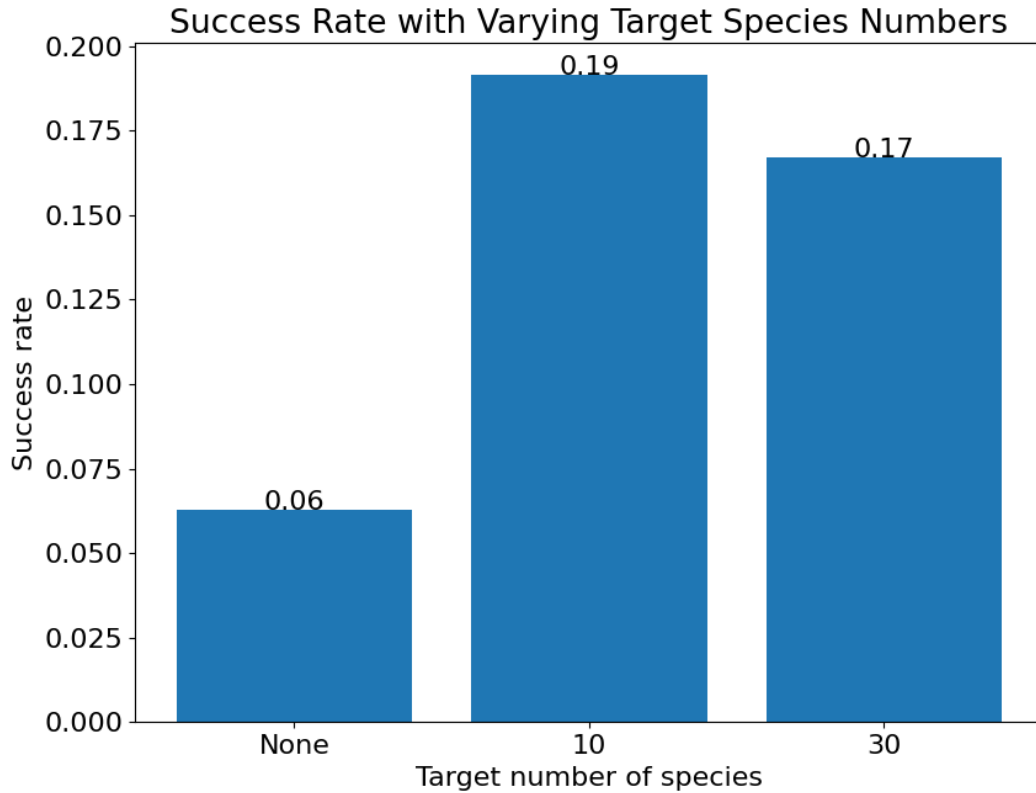**Might be interesting to try with 50 species and see if I can get it to be worse just to make a point**

Figure 5.4: Evolution success rate of 700 trials with no speciation, with a target species of 10, and with a target species of 30. Speciation significantly improves success rate, but more species does not improve success.

5.4.2 Crossover Does Not Improve Success Rate

Batches of 700 evolution trials with and without crossover were compared. The batch without crossover was the same batch described in 5.4.1 with a target species number of 10 and a success rate of 0.19. This batch had 0.0 probability of crossover and 0.75 probability of mutation. This meant that there was a 25% chance that a network was passed on to the next generation without modification. The batch with crossover used the crossover procedure described in the Methods section. The probability of crossover was 0.75 and the probability of mutation was 0.75. All networks were either crossed over or mutated

$(p_{crossover} + p_{mutation} - p_{crossover} \cap p_{mutation} = 1)$. This meant that networks had a 50% chance of being crossed over and then mutated $(p_{crossover} \cap p_{mutation} = p_{crossover} + p_{mutation} - 1)$. With these settings, evolution trials with crossover enabled resulted in a sustained oscillator at a rate of 0.16 (112/700). This was significantly lower than the success rate without crossover (p = 0.0344).

Just realized that in the non crossover batch, there were some networks that wouldn't get changed at all under the right circumstances. Should redo this

5.4.3   Effect of Selection Method

5.5   Discussion

Configurability is cool. You could also use this to match timeseries data or evolve other network behaviors As of today's results, it seems that people were right and there's not really a good computationally efficient way to crossover networks without wrecking them.

5.6   Conclusion

Yay I did it.

Test	Description	Possible Implementation
Negative Concentration	Ensure that there are no concentration values less than zero	Examine time series data for negative values
Infinite Concentration	Ensure that there are no concentration values that approach infinity	• Examine time series data for species concentration that never decrease or fail to reach steady state • Simulate the model for increasing time periods and check for software exceptions
Specific Changes in Variables	Check that a change in a specific variable produces the expected change in a related variable	• Check that derivatives have the expected negative or positive correlations • Use "events" to instantaneously change variables and check for the expected change in the related variable
Bounds Checking	Ensure that a specific species concentration does not deviate beyond a given range	Examine time series data for values outside the given range
Validation	Check that a model adequately reproduces validation data (data that was not used in the construction of the model)	Compare simulation data to validation data, assess relative error
Verification	Ensure that the model can be simulated using different platforms, operating systems, versions, etc. without changing the results	Use BioSimulations to simulate the model on various platforms and compare results
Parameter Sensitivity	Identify highly sensitive parameters and their effects on result uncertainty	• Use simple OAT perturbations to test sensitivity • Implement a statistical parameter sensitivity measurement algorithm or result uncertainty quantification method (longer term)

Table 5.1: Summary of proposed tests and possible implementations.

Bibliography

[1] Overview of circadian rhythms. *Alcohol Research and Health*, 25, 2001.

[2] Circadian rhythm, lifestyle and health: A narrative review. *Iran Journal of Public Health*, 47, 2018.

[3] Stefano Allesina. chapter Generalized Lotka-Volterra. 2020.

[4] Alexander Aulehla and Olivier Pourquié. Oscillating signaling pathways during embryonic development. *Current opinion in cell biology*, 20(6):632–637, dec 2008.

[5] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[6] Ana Maria Cebolla and Guy Cheron. Understanding neural oscillations in the human brain: From movement to consciousness and vice versa. *Frontiers in Psychology*, 10, August 2019.

[7] Luca Cerone and Zoltán Neufeld. Differential gene expression regulated by oscillatory transcription factors. *PLoS ONE*, 7(1):e30283, January 2012.

[8] Raymond Cheong and Andre Levchenko. Oscillatory signaling processes: the how, the why and the where. *Current Opinion in Genetics and Development*, 20(6):665–669, December 2010.

[9] Kiri Choi, J. Kyle Medley, Matthias König, Kaylene Stocking, Lucian Smith, Stanley Gu, and Herbert M. Sauro. Tellurium: An extensible python-based modeling environment for systems and synthetic biology. *Biosystems*, 171:74–79, September 2018.

[10] Dominique Chu. Evolving genetic regulatory networks for systems biology. In *2007 IEEE Congress on Evolutionary Computation*, 2007.

[11] Mark A. Coletti, Eric O. Scott, and Jeffrey K. Basset. In *Proceedings of the 2020 Genetic and Evolutionary Computationa Conference Companion*, 2020.

[12] Andres Cubillos-Ruiz, Tingxi Guo, Anna Sokolovska, Paul F. Miller, James J. Collins, Timothy K. Lu, and Jose M. Lora. Engineering living therapeutics with synthetic biology. *Nature Reviews Drug Discovery*, 20(12):941–960, October 2021.

[13] Anastasia Deckard and Herbert M Sauro. Preliminary Studies on the In Silico Evolution of Biochemical Networks. *ChemBioChem*, 5(10):1423–1431, October 2004.

[14] Anastasia C Deckard, Frank T Bergmann, and Herbert M Sauro. Enumeration and online library of mass-action reaction networks. *arXiv preprint arXiv:0901.3067*, 2009.

[15] Pete Donnell, Murad Banaji, Anca Marginean, and Casian Pantea. CoNtRol: an open source framework for the analysis of chemical reaction networks. *Bioinformatics*, 30(11):1633–1634, 01 2014.

[16] Barry Drennan and Randall D. Beer. Evolution of repressilators using a biologically-motivated model of gene expression. In *Artificial Life X: Proceedings of the Tenth International Conf. on the Simulation and Synthesis of Living Systems*, 2006.

[17] Juan J. Durillo and Antonio J. Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771, October 2011.

[18] Oleg Dyachok, Yegor Isakov, Jenny Sågetorp, and Anders Tengholm. Oscillations of cyclic amp in hormone-stimulated insulin-secreting β-cells. *Nature*, 439(7074):349–352, January 2006.

[19] Michael B. Elowitz and Stanislas Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, January 2000.

[20] Charles Elton and Mary Nicholson. The ten-year cycle in numbers of the lynx in canada. *The Journal of Animal Ecology*, 11(2):215, November 1942.

[21] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[22] Paul Francois and Vincent Hakim. Design of genetic networks with specified functions by evolution in silico. 2004.

[23] Koichi Fujimoto, Shuji Ishihara, and Kunihiko Kaneko. Network Evolution of Body Plans. *PLoS ONE*, 3(7):e2772, July 2008.

[24] Ahmed Fawzy Gad. Pygad: An intuitive genetic algorithm python library. *Multimedia Tools and Applications*, pages 1–14, 2023.

[25] Christian Gagné and Marc Parizeau. Genericity in evolutionary computation software tools: Principles and case study. *International Journal on Artificial Intelligence Tools*, 15(2):173–194, 2006.

[26] G.F. Gause. Experimental demonstration of volterra's periodic oscillations in the numbers of animals. *The Journal of Experimental Biology*, 1935.

[27] Naama Geva-Zatorsky, Nitzan Rosenfeld, Shalev Itzkovitz, Ron Milo, Alex Sigal, Erez Dekel, Talia Yarnitzky, Yuvalal Liron, Paz Polak, Galit Lahav, and Uri Alon. Oscilla-

tions and variability in the p53 system. *Molecular systems biology*, 2(1):2006–0033, jan 2006.

[28] A. Ghosh and B. Chance. Oscillations of glycolytic intermediates in yeast cells. *Biochemical and Biophysical Research Communications*, 16(2):174–181, June 1964.

[29] R. Heinrich, E. Melendez-Hevia, F. Montero, J.C. Nuno, A. Stephani, and T.G. Waddell. The structural design of glycolysis: an evolutionary approach. *Biochemical Society Transactions*, 27:294–298, 1999.

[30] Joseph Higgins. The theory of oscillating reactions - kinetics symposium. *Industrial & Engineering Chemistry*, 59(5):18–62, May 1967.

[31] Alan C Hindmarsh, Peter N Brown, Keith E Grant, Steven L Lee, Radu Serban, Dan E Shumaker, and Carol S Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.

[32] John Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications in biology, control, and artificial intelligence.* U Michigan Press, 1975.

[33] Beichen Huang, Ran Cheng, Zhuozhao Li, Yaochu Jin, and Kay Chen Tan. Evox: A distributed gpu-accelerated framework for scalable evolutionary computation. *IEEE Transactions on Evolutionary Computation*, page 1–1, 2024.

[34] Yaochu Jin and Bernhard Sendhoff. Evolving in silico bistable and oscillatory dynamics for gene regulatory network motifs. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 386–391, Hong Kong, China, June 2008. IEEE.

[35] Ingunn W. Jolma, Xiao Yu Ni, Ludger Rensing, and Peter Ruoff. Harmonic oscillations in homeostatic controllers: Dynamics of the p53 regulatory system. *Biophysical Journal*, 98(5):743–752, March 2010.

[36] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80(5):8091–8126, October 2020.

[37] Zoltán Kis, Hugo Sant'Ana Pereira, Takayuki Homma, Ryan M. Pedrigi, and Rob Krams. Mammalian synthetic biology: emerging medical applications. *Journal of The Royal Society Interface*, 12(106):20141000, May 2015.

[38] J. Knabe, C. Nehaniv, M. Schilstra, and T. Quick. Evolving biological clocks using genetic regulatory networks. In *Proceedings of the Artificial Life X Conference (Alife 10)*, 2006.

[39] Y. Kobayashi, T. Shibata, Y. Kuramoto, and A. S. Mikhailov. Evolutionary design of oscillatory genetic networks. *Eur. Phys. J. B*, 76(1):167–178, July 2010.

[40] Alfred J. Lotka. Contribution to the theory of periodic reactions. *The Journal of Physical Chemistry*, 14(3):271–274, March 1910.

[41] Alfred J. Lotka. Analytical note on certain rhythmic relations in organic systems. In *Proceedings of the National Academy of Sciences*, volume 6, 1920.

[42] Alfred J. Lotka. Undamped oscillations derived from the law of mass action. *Journal of the American Chemical Society*, 42(8):1595–1599, August 1920.

[43] Sean Luke. ECJ evolutionary computation library, 1998.

[44] Mario A. Marchisio and Jörg Stelling. Automatic Design of Digital Synthetic Gene Circuits. *PLoS Comput Biol*, 7(2):e1001083, February 2011.

[45] Brad L. Miller and David E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Syst.*, 9, 1995.

[46] J.E. Mittenthal, B. Clarke, T. Waddell, and G. Fawcett. A new method for assembling metabolic networks with application to the krebs citric acid cycle. *Journal of Theoretical Biology*, 2001.

[47] Nicola Montano, Chiara Cogliati, Valdo Jose' Dias da Silva, Tomaso Gnecchi-Ruscone, and Alberto Malliani. Sympathetic rhythms and cardiovascular oscillations. *Autonomic Neuroscience*, 90(1–2):29–34, July 2001.

[48] Marco S. Nobile, Daniela Besozzi, Paolo Cazzaniga, Dario Pescini, and Giancarlo Mauri. Reverse engineering of kinetic reaction networks by means of cartesian genetic programming and particle swarm optimization. pages 1594–1601, 2013.

[49] Béla Novák and John J. Tyson. Design principles of biochemical oscillators. *Nature Reviews Molecular Cell Biology*, 9(12):981–991, oct 2008.

[50] SR Paladugu, V Chickarmane, A Deckard, JP Frumkin, M McCormack, and HM Sauro. In silico evolution of functional modules in biochemical networks. *IEE Proceedings-Systems Biology*, 2006.

[51] Casian Pantea and Gheorghe Craciun. Computational methods for analyzing bistability in biochemical reaction networks. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 549–552, 2010.

[52] Christian S. Perone. Pyevolve: a python open-source framework for genetic algorithms. *SIGEVOlution*, 4, 2009.

[53] J. Patrick Pett, Anja Korenčič, Felix Wesener, Achim Kramer, and Hanspeter Herzel. Feedback loops of the mammalian circadian clock constitute repressilator. *PLOS Computational Biology*, 12(12):e1005266, December 2016.

[54] Hartmut Pohlheim. Geatbx-genetic and evolutionary algorithm toolbox for matlab. 1994.

[55] Veronica Porubsky and Herbert Sauro. Application of parameter optimization to search for oscillatory mass-action networks using python. *Processes*, 7(3):163, mar 2019.

[56] Christopher Rackauckas and Qing Nie. Differentialequations.jl – a performant and feature-rich ecosystem for solving differential equations in julia. *The Journal of Open Research Software*, 5(1), 2017. Exported from https://app.dimensions.ai on 2019/05/05.

[57] Herbert Sauro. Network dynamics. *Methods in Molecular Biology*, 2009.

[58] Herbert Sauro. teUtils [software]. *retrieved from https://github.com/sys-bio/teUtils*, 2020.

[59] Luca Scrucca. GA: A package for genetic algorithms in R. *Journal of Statistical Software*, 53(4):1–37, 2013.

[60] Luca Scrucca. On some extensions to GA package: hybrid optimisation, parallelisation and islands evolution. *The R Journal*, 9(1):187–206, 2017.

[61] Monika Sharma and Praveen Kumar. Chemical oscillations: 1. basic principles and examples. *Resonance*, 11(2):43–50, February 2006.

[62] Erik Smedler and Per Uhlén. Frequency decoding of calcium oscillations. *Biochimica et Biophysica Acta (BBA) - General Subjects*, 1840(3):964–969, March 2014.

[63] Lucian Smith, Frank Bergmann, Deepak Chandran, and Herbert Sauro. Antimony: A modular model definition language. *Bioinformatics (Oxford, England)*, 25:2452–4, 08 2009.

[64] Andy Somogyi, J Kyle Medley, Kiri Choi, and Herbert Suaro. Roadrunner [software]. 2020.

[65] Endre T Somogyi, Jean-Marie Bouteiller, James A Glazier, Matthias König, J Kyle Medley, Maciej H Swat, and Herbert M Sauro. libroadrunner: a high performance sbml simulation and analysis library. *Bioinformatics*, 31(20):3315–3321, jun 2015.

[66] Kenneth O. Stanley and Risto Miikkulainen. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*, 10(2):99–127, June 2002.

[67] A. Stephani, J.C. Nuno, and R. Heinrich. Optimal stoichiometric designs of atp-producing systems as determined by an evolutionary algorithm. *Jounral of Theoretical Biology*, 199, 1999.

[68] Lillian T. Tatka, Wesley Luk, Timothy C. Elston, Joseph L. Hellerstein, and Herbert M. Sauro. Cesium: A public database of evolved oscillatory reaction networks. *Biosystems*, 224:104836, February 2023.

[69] Victor Trevino and Francesco Falciani. Galgo: an r package for multivariate variable selection using genetic algorithms. *Bioinformatics*, 22(9):1154–1156, March 2006.

[70] J.J Tyson. Biochemical oscillators. In *Computational Cell Biology*. Springer New York, 2007.

[71] John J. Tyson. Biochemical oscillations. In *Interdisciplinary Applied Mathematics*, pages 230–260. Springer New York.

[72] John J. Tyson. Classification of instabilities in chemical reaction systems. *The Journal of Chemical Physics*, 62(3):1010–1015, February 1975.

[73] Balth. van der Pol and J. van der Mark. LXXII.the heartbeat considered as a relaxation oscillation, and an electrical model of the heart. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 6(38):763–775, nov 1928.

[74] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[75] Vito Volterra. Fluctuations in the abundance of a species considered mathematically. *Nature*, 118, 1926.

[76] Xujian Wang, Fenggan Zhang, and Minli Yao. A many-objective evolutionary algorithm with metric-based reference vector adjustment. *Complex and Intelligent Systems*, 10(1):207–231, July 2023.

[77] Margaret Watts, Bernard Fendler, Matthew J. Merrins, Leslie S. Satin, Richard Bertram, and Arthur Sherman. Calcium and metabolic oscillations in pancreatic islets: Who's driving the bus? *SIAM Journal on Applied Dynamical Systems*, 13(2):683–703, January 2014.

[78] James N. Weiss and Jun-Hai Yang. Oscillations at odds in the heart. *Journal of General Physiology*, 135(4):303–305, March 2010.

[79] Daniel Wilczak and Jack Nguyen. *EasyGA - Genetic Aglorithms made Easy*, 2021.

[80] Art Wilde, Daniel Thiem, , Leo, Abhinav Gupta, Andrew Haselgrove, Daniel Molina, Honeypot95, Matago, Rogerluo, and Valentin Churavy. Evolutionary.jl, 2021.

[81] Takafumi Yamashino. From a repressilator-based circadian clock mechanism to an external coincidence model responsible for photoperiod and temperature control of plant architecture inarabodopsis thaliana. *Bioscience, Biotechnology, and Biochemistry*, 77(1):10–16, January 2013.