Modeling drug perturbations at multiple resolutions: Mechanism extraction from multi-omic, multiple timepoint data to single cell mechanistic simulations.

By

James C. Pino

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Chemical Physical Biology

August 7, 2020

Nashville, Tennessee

Approved:

Professor Carlos F. Lopez

Professor Jens Meiler

Professor Vito Quaranta

Professor John Wikswo

Professor Sandra Zinkel

To those who persevered when others could not.

TABLE OF CONTENTS

Page

DEDICATION ii

Chapter

1 Introduction 1
1.1 Approaches to model cellular behaviour 1
1.1.1 Introduction to global approaches 3
1.1.2 Introduction to mechanistic models 4
1.1.3 Introduction to apoptosis 5

2 Top down approaches to identify mechanism 7
2.1 Mechanism extraction through the development of novel computational tools . . . 8
2.1.1 Introduction 8
2.1.2 Results 10
2.1.2.1 Design and Implementation 10
2.1.2.2 Case study: Using MAGINE to explore the temporal response of bendamustine-treated leukemia cells 18
2.1.2.2.1 Bendamustine affects HIV infection-related genes . . . 30
2.1.3 Methods 32
2.1.3.1 MAGINE software implementation 32
2.1.4 Discussion 32
2.1.4.1 Limitations 34
2.1.5 Notes 34
2.1.6 Additional Methods 34
2.1.6.1 Data format 34
2.1.6.2 Enrichment term aggregation 36
2.1.6.3 Annotated gene set network construction 36
2.1.6.4 Software availability and requirements 39
2.1.6.5 Experimental methods 40

3 Advancements in mechanistic modelling 41
3.1 Background 41
3.1.1 Bottoms up approach 41
3.1.2 Stochastic and deterministic simulations 42
3.1.3 Extrinsic apoptosis reaction model (EARM) 43
3.2 Model simulation 44
3.2.1 Deterministic simulation 44
3.2.1.1 Enabling high throughput simulations using GPUs 44
3.2.1.2 Features and Implementation 45
3.2.1.3 Result 47
3.2.1.4 Conclusion 48

3.2.2 Efficient stochastic simulations . . . 49
3.2.2.1 Introduction . . . 49
3.2.2.2 Implementation . . . 50
3.2.2.3 Models . . . 54
3.2.2.3.1 Michaelis-Menten . . . 54
3.2.2.3.2 Schlögl . . . 54
3.2.2.3.3 Kinase Cascade . . . 55
3.2.2.3.4 EARM1.0 . . . 55
3.2.2.4 Results . . . 55
3.2.2.4.1 Validation of implementation . . . 55
3.2.2.4.2 Benchmarks . . . 57
3.2.2.4.3 Additional benchmarks . . . 60
3.2.2.4.4 OpenCL . . . 60
3.2.2.5 Discussion . . . 63
3.3 Fitting models to data . . . 65
3.3.1 Introduction to particle swarm optimization . . . 66
3.3.2 simplePSO: Fast modeling calibration for Rule based models. . . . 67
3.4 Quantifing the effects of initial protein concentrations on the model's prediction of cell fate. . . . 70
3.4.1 Results and Discussion . . . 75

4 Conclusion and Future directions . . . 81
4.1 Bridging the gap : Reactome2Rules . . . 82

5 Appendix . . . 86
5.1 MAGINE Jupyter notebooks . . . 95
5.1.1 MAGINE data exploration . . . 95
5.1.2 MAGINE network exploration . . . 95
5.1.3 MAGINE enrichment analysis exploration . . . 95
5.1.4 MAGINE AGN creation exploration . . . 95

BIBLIOGRAPHY . . . 96

LIST OF TABLES

Table		Page
2.1	Data module	12
2.2	Table of enrichment module functions	12
2.3	Network methods	17
2.4	Example data input used in MAGINE	35
3.1	Models used for PySB/cupSODA performance testing	47
3.2	Size of the different models used in this section.	55
3.3	Simulation run times.	64
5.1	Reference initial species populations for EARM.	87

LIST OF FIGURES

Figure		Page
1.1	Summary of approaches of drug response	2
2.1	The MAGINE platform.	13
2.2	Example of MAGINE workflow.	14
2.3	Time series enrichment analysis pipeline.	15
2.4	Construction of data seeded network.	17
2.5	Overlap between experimental measurements.	19
2.6	Example workflow of bendamustine mechanism extraction.	22
2.7	Zoom in of Fig. 2.6.	23
2.8	Demonstration of enrichment compression.	24
2.9	Custom workflows for exploring known biological processes of bendamustine.	28
2.10	Similarity between Bendamustine and HIV infection.	31
2.11	Zoom in of Fig 2.10.	31
2.12	Annotated gene set network construction	38
3.1	Run time comparisons between PySB/cupSODA and SciPy/LSODA.	47
3.2	Workflow for the GPU-SSA implementation.	53
3.3	Validation of the CUDASimulator and OpenCLSimulator.	56
3.4	Comparing BioNetGen, StochKit and GPU-SSA timings run on the NVIDIA V100.	58
3.5	Fold change improvements of the CUDASimulator compare to CPU implementation	59

3.6 Timing analysis for multiple generations of GPUs. 60
3.7 OpenCL vs CUDA timings. 62
3.8 Timing analysis for CPU implementations. 62
3.9 Best fit parameters from simplePSO. 68
3.10 Example output using simplePSO on with stochastic simulations. 68
3.11 Performance of PSO vs simulated annealing 69
3.12 Summary view of EARM 2.0 sensitivity 75
3.13 Individual initial concentration sensitivity for EARM 2.0 76
3.14 Summary view of EARM 1.0 sensitivity 77
3.15 Comparing sensitivities of the BCL2 family of proteins 78
3.16 Individual initial concentration sensitivity for EARM 1.0 79

4.1 The EARM model expanded based on enriched terms identified by MAGINE. . 83
4.2 Reactome2Rules creation of MAGINE edge ‘BID activate BAX’ 85

5.1 Time course for active maturation-promoting factor (MPF) 88
5.2 Sensitivity analysis for the Tyson cell cycle model. 89
5.3 Time course for cAMP in the Ras/cAMP/PKA model. 90
5.4 Sensitivity analysis for the Ras/cAMP/PKA model. 91
5.5 Time course for active Smac (aSmac). 92
5.6 Sensitivity analysis for parameter set 1 of EARM 2.0. 93
5.7 Sensitivity analysis for parameter set 2 of EARM 2.0. 94

Chapter 1

Introduction

1.1 Approaches to model cellular behaviour

Understanding and predicting how a cell will respond to an external signal is a central theme in systems biology (Ideker et al., 2001; Hood et al., 2004; Kremling and Saez-Rodriguez, 2007). For example, several cancer therapies are currently being used to induce apoptosis, the main form of programmed cell death (Thompson, 1995; Fulda and Debatin, 2006; Ghobrial et al., 2005). In cancer, cells are unable to commit to apoptosis due an imbalance of anti-apoptotic signals or the lack of necessary machinery (Hanahan and Weinberg, 2011). Many therapeutics are designed to regulate apoptosis (Fischer and Schulze-Osthoff, 2005). Yet, current treatments often result in non-optimal response, resistance to the drug, or unwanted side effects (Stuckey and Shah, 2013). Thus, being able to predict how cells will respond prior to administration is clinical relevant. This is a difficult task due to the fact that the cell is complex (Csete and Doyle, 2002). There are an extremely large number of possibilities of molecular species: $\sim$ 20k genes, 40% to 60% of which have alternatively spliced variants, an average of 2.5 post-translation modifications per protein, resulting in an extremely large number of unique molecular states (Papin et al., 2005). It is these relationships and the resulting non-linear dynamics that make predicting drug response a difficult task (Huang, 2012).

Methods used to study the signaling dynamics can be broadly be classified into two categories: local simulation and global discovery Kitano (2002), summarized in Figure 1.1. The local approach uses mechanistic models of the most important molecules and the interactions involved in the cellular response based on extensive experimental analysis. These models are exploratory, allowing different mechanisms to be postulated, numerically tested via simulation, then validated by experimentation (Schleich and Lavrik, 2013; Kholodenko, 2006). If the model is not capable of explaining the observation, this means there is knowledge missing from the

model. Identifying what is missing is a difficult step. Additionally, after identifying the missing components or interactions, the model would need to be re-calibrated and characterized. The simulations required for calibrating and characterizing the model are already time consuming, which is further amplified with increasing the models size.

Conversely, the discovery-based global approach typically begins with the measurements of a complete physiological system. This can be performed before and after drug treatment, allowing the calculation of quantitative changes of the molecular species. This allows for a near unbiased discovery of components of responsible for the response. Unfortunately, for the purposes of mechanistic exploration, such approaches result in large, complex datasets and ultimately fail to provide insights about molecular interactions that could drive a process.

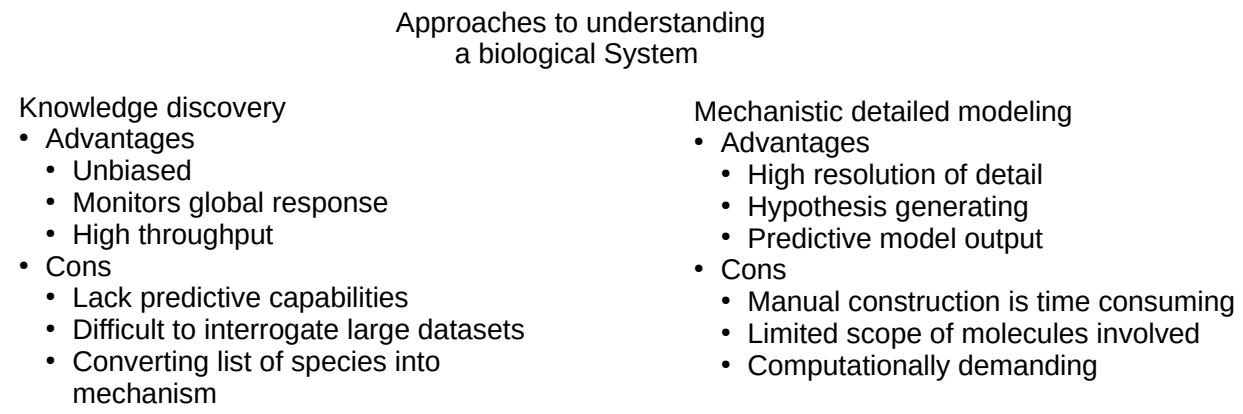


Figure 1.1: **Summary of approaches of drug response.** Pros and cons of the global, global discovery driven approach and the local, mechanistic modeling approach.

In this dissertation, I address the limitations mentioned above and introduce methods in modeling techniques in order to better understand how cells respond to drug. I demonstrate these advances in the mechanism(s) of apoptosis, an ideal system to study with computational modeling due to the extensive ground work provided by previous models. First, I introduce methods to use high throughput multi-omics data to identity components of models in section 2. The methods introduced allow large datasets to be explored and interrogated easily with the level of detail required for mechanistic models. I then address the bottlenecks facing mechanistic modeling by

introducing orders of magnitude faster simulation methods in section 3.2.2 and 3.2.1, introduce efficient model calibration implementations in section 3.3, and demonstrate their capabilities to describe published models of apoptosis in section 3.4. Finally, in future directions 4.1 I discuss how these two approaches complement one another and discuss how they can be used in the automation of model construction.

1.1.1 Introduction to global approaches

Identifying the molecular components (proteins, RNA, metabolites) and how their interactions give rise to a phenotypic response is a difficult task. To fill in these gaps, omics studies have become a vital tool (Aebersold and Mann, 2003, 2016). These studies typically measure and quantify a cellular state, that is everything detectable within the cell, before and after treatment. In this way, molecular species that are significantly altered before and after drug are identified for further study. However, there are often a large number of species that are altered due to drug, so identifying and selecting the key contributors is often hard. Multiple approaches to analyze -omics data have been proposed with the goal to extract knowledge from complex datasets. Of these approaches, enrichment- and network-based methods seem to provide the most insightful analyses that can be useful for hypotheses generation and further guide experiments. Enrichment analysis can provide relevant insights about the biological processes that are modulated when a system is perturbed (Subramanian et al., 2005; King et al., 2003). Typically, enrichment analysis compares the quantitative molecular profiles from two or more experimental conditions to yield a list of proteins that have been significantly modulated by the perturbation. The list of proteins is then analyzed and matched to ontology in databases, which effectively maps key protein hits with specific cellular processes. Unfortunately, for the purposes of mechanistic exploration, enrichment approaches fail to provide insights about molecular interactions that could drive a process, which can be essential to generate novel hypothesis for further experimental validation. In contrast, network analysis can generate maps between measured molecular species to generate a node-edge graph depiction of complex cellular processes. Data-driven approaches to network

analysis employ large datasets typically derived from multiple measurements to generate *ab initio* interaction networks (Omony, 2014). Topology-driven networks, on the other hand, employ prior knowledge from (curated) databases that comprise biochemical interactions derived from literature and provide mappings for many biochemical processes in the cell. Although these approaches can be very powerful to explore interactions among measured species, they become impenetrable after a certain number of interactions (i.e. the "hairball" problem) and thus obscures possible interpretation. Methods that enable the extraction of key species involved in the cellular response, as well as at the level of detail required for predictive modelling, are lacking and can greatly benefit the systems biology community.

1.1.2 Introduction to mechanistic models

Mechanistic models represent the current state of knowledge of the system (Ingalls, 2013). Traditionally, models are constructed in a piece-wise manner, where components and their interactions are added as need until the observation can be recreated (Albeck et al., 2008; Le Novere, 2015). Simulations of the models provides validation of the models assumptions and allows the prediction of the system dynamics (Schleich and Lavrik, 2013; Kholodenko, 2006; Aldridge et al., 2006). Such approaches have been successfully applied in identifying the source of NF-κB oscillations (Kearns et al., 2006), predicting targets to control metabolism (Bakker et al., 1999), and identifying the source of the all-or-nothing commitment to apoptosis (Albeck et al., 2008). A models inability to recapitulate observation informs that there is missing information about the system. Identifying what is missing from the system is a difficult task. Additionally, calibrating models to experimental data can be computational expensive, requiring thousands to hundred of thousands of simulations (Eydgahi et al., 2013) This is because the parameters can vary over order of magnitudes of space, leading to near infinite number of parameter sets that fit the model equally well (Gutenkunst et al., 2007). Finally, once the model is fit, it can be characterized under different experimental settings (drug doses, knockdowns/outs) to provide experimental predictions. This could result in identifying molecular species that

negatively affect the desired outcome (key anti-apoptotic proteins), co-druggable targets (amplify death signal through alternative signaling pathways), or eliminate drug targets due to the models insensitivity to changes in their concentration. Again, such analysis involves thousands of time consuming simulations. Thus, there is a need to accelerate the number of simulations that can be performed in order to overcome the bottlenecks in training and characterizing models.

1.1.3 Introduction to apoptosis

Apoptosis is a main form of programmed cell death (Thompson, 1995). It is vital throughout development and its dysregulation can result in diseases such as neurodegenerative, auto immune diseases (Lockshin, 2016), and cancer (Hanahan and Weinberg, 2011). Apoptosis induction can be classified into either intrinsic or extrinsic (Reed, 2000). Extrinsic apoptosis is activated via external stimuli binding to surface death receptors. This binding results in the formation of the death-inducing signaling complex (DISC) and caspase-8 activation. Caspase-8 then activates BID, which ultimately activates BAX resulting in its translocation into the mitochondria outer membrane (MOM).Alternatively, intrinsic apoptosis is activated via intracellular signals that activate the pro-apoptotic BCL2 family of proteins. These proteins result in the activation of BAK in the MOM. From there, both intrinsic and extrinsic apoptosis share downstream components. Activate BAX or BAK oligomerize to form pores in the MOM, resulting in the mitochondria outer membrane permeabilization (MOMP). MOMP releases cytochrome-c and smac to from the mitochondria. Cytochrome-c binds to the apaf-1 and caspase-9 proteins, resulting in formation of the apoptosome. The apoptosome cleaves caspase-3/7, marking the commitment to cell death. It is these activate caspases that result in the destruction of the cell: protein degradation, DNA fragmentation, and collapse of the cell into apoptotic bodies (Lockshin, 2016).

Since many cancer therapeutics are designed to regulate apoptosis, deciphering the response of drugs could prove clinical useful (Fischer and Schulze-Osthoff, 2005; Ghobrial et al., 2005; Schleich and Lavrik, 2013). A prime example is the application of mechanistic modeling of the response of HL60 cells to TNF-Related Apoptosis Inducing Ligand (TRAIL) (Albeck et al.,

2008; Spencer et al., 2009). When the cells are exposed to TRAIL in the presence of cyclohexamide (a transcription disabling drug), the cells all displayed sharp, "snap action" dynamics, however the time at which they committed to death was variable. In the absence of cyclohexamide, some cells died and others did not. The models were able to recreate and explain the sharp, all-or-nothing dynamics marking the commitment to cell death, however were unable to explain or predict the heterogeneous responses outside the presence of cyclohexamide. This can be in part due to the exclusion of necessary molecular species or interactions involved in the response. These observations represent a gap in our understanding of apoptotic response and raise the following questions: *What molecular species are responsible for the cells response to the drug? Why do some cells die and others don't?* Answering such questions would allow us to design more efficient treatments that can increase desired outcome, such as cell death; prevent undesired side effects; and ultimately improve clinical outcomes. Thus, current models of apoptosis must be improved. Expanding the model to include additionally species requires one to first identify the species, properly incorporate them into the model, fit the new models parameters to the experimental data, reproduce observations, and make new predictions based on the model.

Chapter 2

Top down approaches to identify mechanism

Modelling intrinsic apoptosis is more complicated than extrinsic apoptosis. This is because there are more ways to engage intrinsic apoptosis (such as cell cycle, DNA damage response, or anti-proliferation signals) than engaging extrinsic apoptosis (stimulating the death receptor). This simplicity explains the development of detailed extrinsic apoptosis models rather than intrinsic models. However, many therapeutics (such as chemotherapy) attempt to activate intrinsic apoptosis, creating a need for intrinsic apoptosis models. Once again, since the therapies can activate apoptosis via multiple mechanisms, these models will need to be drug specific. Constructing a model requires identifying the molecular species involved in the response to a drug. Traditional bottoms-up models often include only the most studied molecular components that are known to be involved with pathways involved with the drug. This narrows the scope of possible molecules and makes it easier to construct models. However, the focus of what is known, limiting the exclusion of molecules that may be vital. An alternative approach would be to measure the global response, that is everything following perturbation to the cell. This would allow a near unbiased view of the response of the cell. However, since measuring all the components can lead to large number of changes, it makes it difficult to decipher what are the most important components versus what could possibly be noise. To make things more complicated, responses are dynamic in time, requiring the monitoring of the cell across multiple time points (Norris et al., 2017). This makes managing, interrogating, and deciphering knowledge from these types of datasets difficult. This chapter introduces methods for exploring and extracting out the key components and trends of global measurements. The methods presented here should enable the identification of molecular components that are involved with any provided data set, making the identification of species needed for drug specific mathematical modeling.

2.1 Mechanism extraction through the development of novel computational tools

2.1.1 Introduction

The cellular response to chemical perturbation manifests across multiple molecular and macro-molecular subsystems, which can include changes in RNA transcript abundance, protein abundance and activity, metabolic activity, and–over longer time frames–changes in DNA (e.g. mutations). In cases of acute toxicity (seconds to days), we hypothesize that the first three molecular subsystems are the principal actors, i.e. the transcriptome, proteome, and metabolome. It is now possible to obtain deep measurements of these systems in a matter of hours using a plethora of techniques; primarily, mass spectrometry and RNA-sequencing (Aebersold and Mann, 2003; Toby et al., 2016; Wang et al., 2009). Falling cost and technological improvements make it feasible to obtain data from multiple platforms and time points, yielding thousands to millions of data points from one experiment (Narayan et al., 2016; Sacco et al., 2016; Aebersold and Mann, 2016). Recent work by our labs and others have shown how these kinds of datasets can be used to gain a systems-level understanding of cellular responses to perturbations (Norris et al., 2017; Palmer et al., 2019). Multiple methods to analyze these datasets have emerged over the years with the goal to translate -omics data into biological knowledge and use this knowledge to make predictions and guide experiments. Enrichment-based methods and literature-based network methods stand out as the most commonly used to analyze -omics data (Papin et al., 2004; Huang et al., 2008). However, the amount of data collected to compare multiple cellular responses to perturbations or to follow the dynamic response of cellular processes can easily surpass millions of datapoints. Although existing tools can handle some aspects of data analysis (Wang et al., 2017; Tang et al., 2016; Huang et al., 2007; Chen et al., 2013; Komurov et al., 2012), there is a dearth of tools necessary to integrate emerging large -omics datasets that will integrate data from multiple platforms and enable exploration of complex datasets to extract mechanisms of cellular response to perturbations, necessary to guide all aspects of cell biology experimentation.

Multiple approaches to analyze -omics data have been proposed with the goal to extract knowledge from complex datasets. Of these approaches, enrichment- and network-based methods

seem to provide the most insightful analyses that can be useful for hypotheses generation and further guide experiments. Enrichment analysis can provide relevant insights about the biological processes that are modulated when a system is perturbed (Subramanian et al., 2005; King et al., 2003). Typically, enrichment analysis compares the quantitative molecular profiles from two or more experimental conditions to yield a list of proteins that have been significantly modulated by the perturbation. The list of proteins is then analyzed and matched to ontology in databases, which effectively maps key protein hits with specific cellular processes. Enrichment analysis can be useful when exploring large -omics datasets to identify relevant biological processes by matching proteins that are consistently up- or down-regulated after a perturbation. Unfortunately, for the purposes of mechanistic exploration, enrichment approaches fail to provide insights about molecular interactions that could drive a process, which can be essential to generate novel hypothesis for further experimental validation. In contrast, network analysis can generate maps between measured molecular species to generate a node-edge graph depiction of complex cellular processes. Data-driven approaches to network analysis employ large datasets typically derived from multiple measurements to generate *ab initio* interaction networks (Omony, 2014). Topology-driven networks, on the other hand, employ prior knowledge from (curated) databases that comprise biochemical interactions derived from literature and provide mappings for many biochemical processes in the cell. Although these approaches can be very powerful to explore interactions among measured species, they become impenetrable after a certain number of interactions (i.e. the "hairball" problem) and thus obscures possible interpretation. Importantly, both methods fail when considering multiple experiments such as those comprising comparisons across multiple treatments or multiple time points.

Given the emergence of multi-omics experiments to characterize cellular response to perturbations, we have the Mechanism of Action Generator Incorporating Network Evidence (MAGINE). We believe MAGINE is a timely tool to address the emerging data challenges for the analysis of omics datasets. MAGINE departs from the approach taken by previously developed tools by providing an exploration framework, in the Python programming language, that

leverages existing tools traditionally used in isolation and provides a platform whereby these tools can be used concurrently to extract biological insights from -omics data. MAGINE provides a Pandas-based database platform for omics data integration, and provides a common interface for enrichment based databases through the EnrichR API as well as a link to multiple network-based analysis tools including KEGG, REACTOME, HMDB, SIGNOR, and BioGrid. We have chosen Python as a suitable language for this exploration platform due to its ease of use, large user base, and integration with almost 200,000 packages in the Python Package Index as of the writing of this manuscript. In this work, we demonstrate how MAGINE can be used to integrate RNA-Seq data, as well as multiple modalities of Mass Spectrometry proteomics to explore the response of HL-60 cells to Bendamustine. Although it is well established that bendamustine is a DNA-damage agent, our analysis reveals that the cellular response to the drug is far from a simple biochemical chain of events and instead comprises thousands of biochemical interactions and multiple cellular processes. We conclude this work by showcasing how MAGINE can be used for the exploration of Bendamustine side-effects related to HIV-treatments and validated by recent literature. Finally, given the Python-based nature of MAGINE, we introduce the use of Python notebooks as a method to transparently report the full analysis of our work, suitable for members of the scientific community to both, evaluate our analysis and expand on it for their own use. Therefore, MAGINE unifies multiple practices in the field onto a common sustainable platform that enables the extraction of cellular response mechanisms from large, complex -omics datasets.

2.1.2 Results

2.1.2.1 Design and Implementation

MAGINE is a software stool for interactive analysis and exploration of multi-time point, multi-omics cellular perturbation data. It can identify and elucidate which molecular species are involved in differential drug response and how this response manifests in molecular and macromolecular processes such as cell signaling, and in turn whose those processes interact. MAGINE summarizes and visualizes large datasets and analyses efficiently and interactively, allowing the user to ask

questions across biological scales and to iteratively ask and refine hypotheses of the data through flexible analysis workflows.

MAGINE is split into three main modules (Figure 2.1): data management and visualization, enrichment analysis, and network analysis. Each module can be used independently, yet is designed to allow easy transfer of information across modules. This is accomplished by designing each module to have accessible outputs that can be used as input for other modules (Figure 2.2).

Briefly, the data management module handles data analysis. Data are loaded in using a tabular, comma-separated values (CSV) file, with one measurement per row (see *Methods*). MAGINE stores these in an *ExperimentalData* class, which provides a simple, high-level interface including methods to access, filter, and search these data as required. A summary of these methods is shown in Table 2.1 and example usage can be found online or in Notebook 5.1.1).

The enrichment analysis module identifies over-represented sets of species that share common annotation compared to random background sets (Huang et al., 2007). MAGINE utilizes *EnrichR*, which includes 120 gene set "libraries" (Chen et al., 2013; Kuleshov et al., 2016). Users provide one or more lists of genes, which can be manually constructed or created by the *ExperimentalData* class (e.g. all up-regulated genes, species detected on a specific experimental platform), or filtered by time point). MAGINE automates analysis through EnrichR, as shown in Figure 2.3. The results are stored in an *EnrichmentResult* class which, like the *ExperimentalData* class, including methods to further query, filter, and visualize enrichment terms. Terms can be compared, ranked, grouped, and visualized with built in methods. Genes corresponding to each term can be extracted and used to subset ExperimentalData or create subgraphs.

	Name	Description
Function	load_data	Load MAGINE formatted csv
	create_summary_table	Create summary of counts per experimental method and sample_id
	subset	Filter data based on list of species
	require_n_sig	Filter data to include species that were measured at least N times
Plots	heatmap	Create heatmap or clustermap of species
	volcano_plot	Create volcano plot
	volcano_by_sample	Create a series of volcano plots for each sample
	plot_species	Plot scatter plots of selected species
	plot_histogram	Creates histogram of fold change values
Properties	sig	Filter data to include only significant flagged entries
	up	Filter data to include fold change >0
	down	Filter data to include fold change <0
	id_list	Compile a list of unique species from sample
	by_sample	Generate list of unique species for each sample

Table 2.1: Data module

	Name	Description
Functions	load_enrichment	Load MAGINE created enrichment analysis csv
	run	Run enrichment analysis for list of genes across selected gene sets
	run_samples	Run enrichment analysis for multiple lists of genes across selected gene sets
	run_enrichment_for_project	Run enrichment for entire ExperimentalData instance
	require_n_sig	Filter data to include species that were measured at least N times
	filter_multi	Filter by multiple criteria (p_value, combined_score, database, etc)
	find_similar_terms	Rank order terms by similar gene sets
	filter_based_on_words	Filter by key words
	all_genes_from_df	Generate list of all genes in current EnrichmentResult instance
	term_to_genes	Generate list of genes for given term
	remove_redundant	Remove terms that are less enriched but highly similar
Plots	heatmap	Find all paths between list
	dist_matrix	Plot distance matrix of all terms

Table 2.2: Table of enrichment module functions

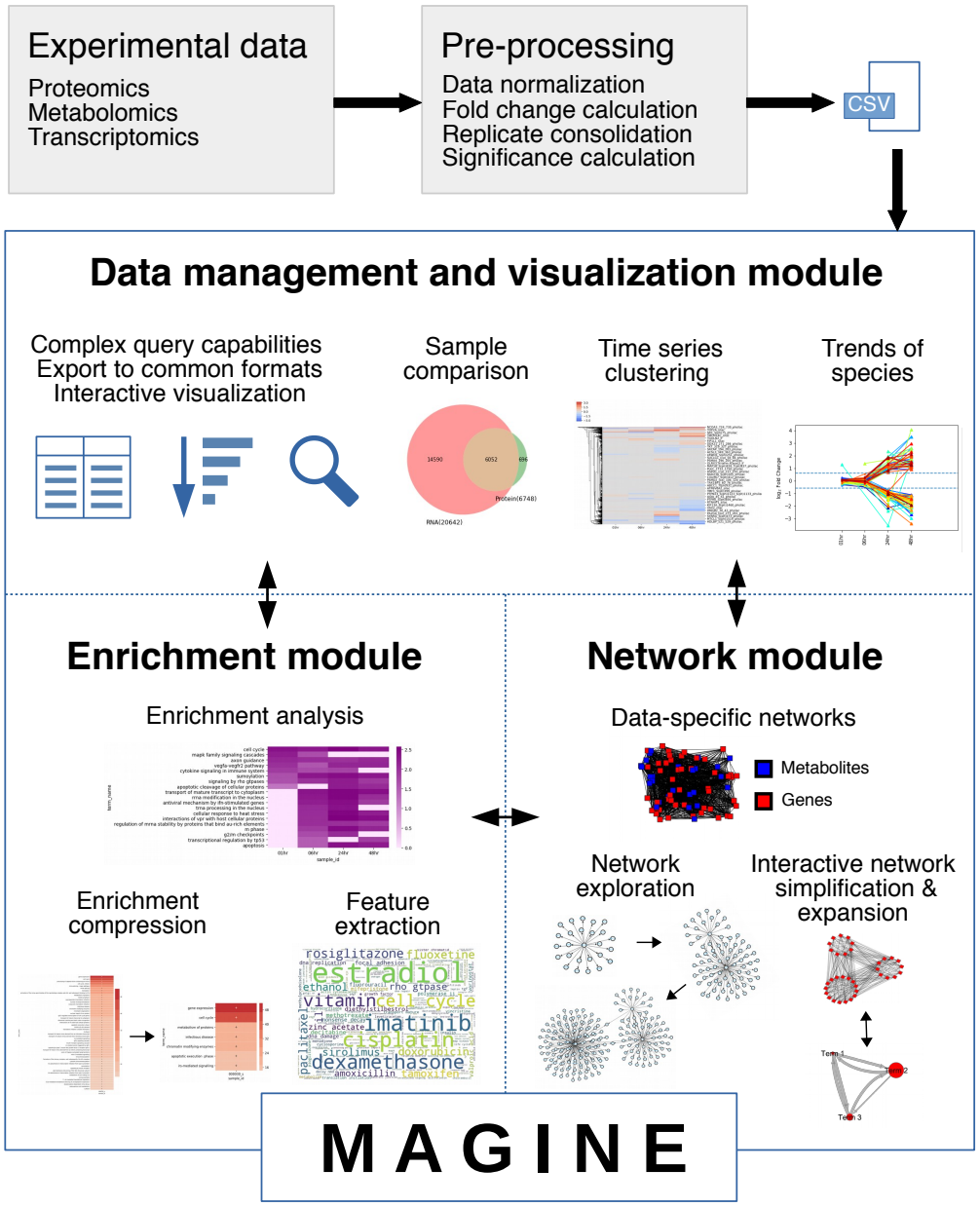


Figure 2.1: **The MAGINE platform.** MAGINE is designed for fold change quantified multi-sample, multi-omic data. It is built around three concepts: data management, enrichment analysis, and network exploration. The modular design allows flow of information between the data, enrichment, and network module, allowing an iterative cycle with varying levels of resolution. Example outputs of each module is provided in each panel.

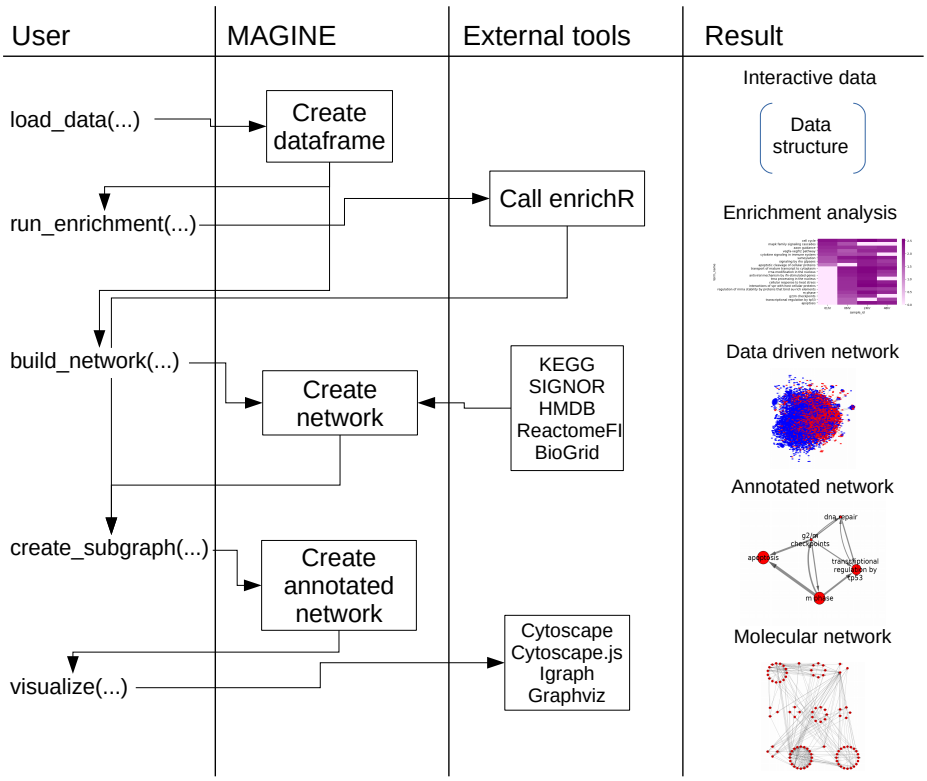


Figure 2.2: **Example of MAGINE workflow.**Calls made by the user (in left column) and how this are handled by MAGINE, external tools, and examples of resulting output (right column). This is a typical processing workflow where users desire to run enrichment analysis, construct a data specific network, and construct a compressed annotated gene set network.

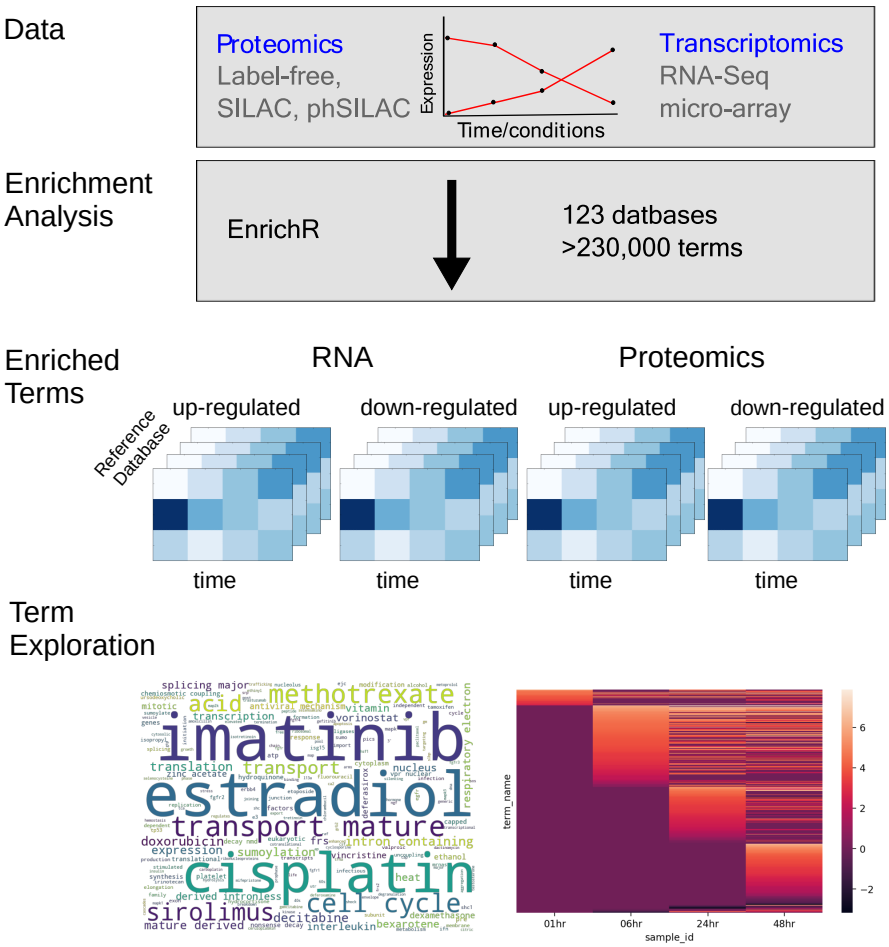


Figure 2.3: **Time series enrichment analysis pipeline.**Automation of running enrichment analysis on multiple time points and experimental platforms. We utilize EnrichR to access various databases. We then organize the data. This allows us to explore the output in various ways. We can use word clouds to compress various databases to see common terms across them all. We can also plot enriched terms over times to see trends of biological processes.

A significant problem in traditional enrichment analyses is the highly varied granularity of terms, which can be very broad (e.g. "biological proces"), or highly specific (e.g. "cysteine-type endopeptidase inhibitor activity involved in apoptotic process"). In addition, each gene often maps to multiple terms. This introduces the possibility of redundancy of terms, which increases the total of number of terms to explore, and hinders human interpretation of results. To address these issues, we introduce a method that allows the aggregation of terms based on their gene content similarity (described in *Methods*). The reduces the overall number of enrichment terms and their redundancy, greatly aiding human interpretation.

A summary of functions available in the enrichment modules is shown in Table 2.2 and example usage can be found online or in Table 5.1.3).

MAGINE's network module allows users to build, query, and visualize molecular and gene annotation networks. A summary of the network module's methods is shown in Table 2.3 and example usage can be found online or in Notebook 5.1.2. The network module utilizes connectivity information from multiple databases, including KEGG (Ogata et al., 1999; Kanehisa et al., 2016), SIGNOR (Perfetto et al., 2016), Reactome Functional Interactions (Wu et al., 2010), BioGrid (Chatr-Aryamontri et al., 2017; Stark, 2006), and HMDB (Wishart et al., 2009, 2007). The graphs underlying these database are merged into a "background" network, which can be used to perform queries or construct context-specific networks based on a user-provided "seed species" list. The list can be created from various sources: significantly changed species, mutational evidence, literature review, or manual curation. We iterate through the databases and add edges and nodes based on connectivity to the seed species, shown in Figure 2.4. The resulting networks are a subset of the background network, focused around the specific molecular species of interest. These networks can be large, ($>$ 20 000 nodes and $>$ 100 000 edges), MAGINE users can use the *Subgraph* class to generate subnetworks based on various queries (Table 2.3). The functions enable users to find paths between species or expanding networks of finding neighbors (upstream or downstream) of nodes of interest. Options are provided to limit the network expansion, such as setting a maximum distance from specific nodes.

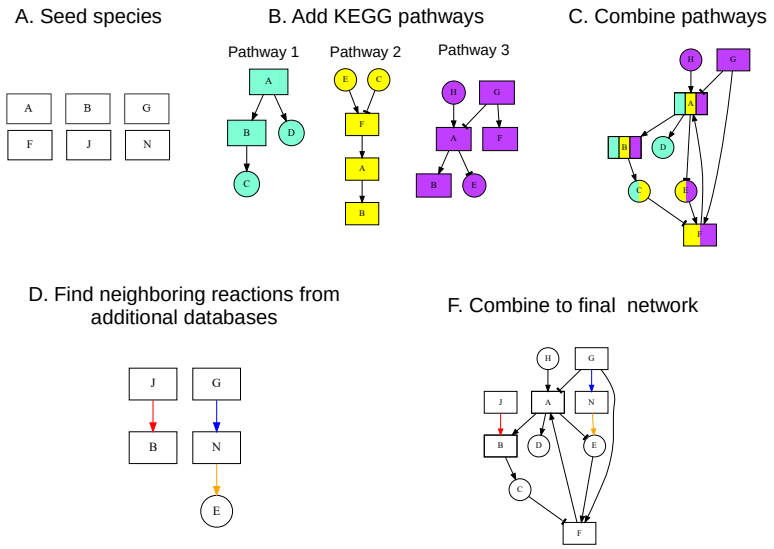


Figure 2.4: **Construction of data seeded network.** Example set of seed species to build network (a) are search across KEGG pathways (b). Pathways in which seed species are contained are merged (c). Next, edges between seed species and nodes added by the KEGG pathways are identifier (d). Finally, all nodes and edges are combined into a single network (e).

Table 2.3: Network methods

Name	**Description**
build_network	Generate network centered around provided species
create_subnetwork	Creates annotated set network from enriched terms and background network
paths_between_pair	Find shortest path(s) between pair
paths_between_list	Find all paths between list
paths_between_two_lists	Find paths between two lists
neighbors	Find neighbors (upstream and/or downstream) of node
expand_neighbors	Add neighbors of node to network
draw	Draw using igraph, matplotlib, graphviz, cytoscape.js
RenderModel	Create Cytoscape instance of network through py2cytoscape

Additionally, we introduce a method to create a coarse-grained network from gene sets that we call an annotated gene set network (AGN). The AGN is motivated by the desire to combine "big picture" information about biological processes and when they occur from enrichment analysis with inter-process communication provided by molecular networks. This results in a coarse-grained network, where nodes are terms and the edges are frequency of connections between the sets of nodes in the molecular network, and a fine-grained network, which contains the species and the connections between them. By using both, we are able explore at a high level

and zoom in on detail when required.

Finally, MAGINE's network module provides various tools for network visualization, allowing users to overlay data or update attributes of the network. Users can visualize networks in Jupyter notebooks using *cytoscape.js* (Franz et al., 2015), modify a *cytoscape* session via *py2cytoscape*, or create a sequence of figures of network activity through *matplotlib*, *igraph*, or *graphviz*. Networks can be exported through Networkx, allowing users to use the networks externally.

MAGINE is implemented in the Python language, leveraging its ease of use, large open source ecosystem, and growing presence in scientific applications (Pérez et al., 2011). It has been tested on Windows, Mac OS, and Linux using Python versions 2.7, 3.6, and 3.7. It can be used within the Python console, interactive Jupyter Notebooks, or within data processing and analysis pipelines. We envisage the majority of users are best served through the Jupyter notebook option, and thus we described that in the most detail and provide a tutorial notebook.

2.1.2.2 Case study: Using MAGINE to explore the temporal response of bendamustine-treated leukemia cells

MAGINE can be utilized on a wide variety of datasets that have case/control measurements from one or more -omics platforms. Here, we demonstrate the capabilities of MAGINE on a recently generated dataset: the response of HL60 cells, a human leukemia cell line, to treatment with bendamustine, a nitrogen mustard alkylating agent used for the treatment of chronic lymphocytic leukemia and non-Hodgkin lymphoma (Cheson and Rummel, 2009). Bendamustine is a cytotoxic agent that can induce DNA interstrand cross links (ICLs) which can prevent replication and transcription, induce cell cycle arrest, and ultimately cell death (Leoni and Hartley, 2011). The main goal of this study was to identify molecular species and biological processes involved in the cellular response to bendamustine. Data were gathered across multiple proteomics platforms and RNA-sequencing, spanning between 30 seconds and 72 hours post-drug exposure (see Methods; Figure 2.5). The analysis is summarized and documented in Notebook 5.1.1.

a

	30s	30m	01h	03h	06h	12h	18h	24h	36h	48h	60h	72h
C18	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
HILIC	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
label_free	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ph_silac	✓	-	✓	-	✓	✓	-	✓	-	-	-	-
silac	✓	-	✓	-	✓	✓	-	✓	-	-	-	-
rna_seq	-	-	✓	-	✓	✓	-	✓	-	-	-	-

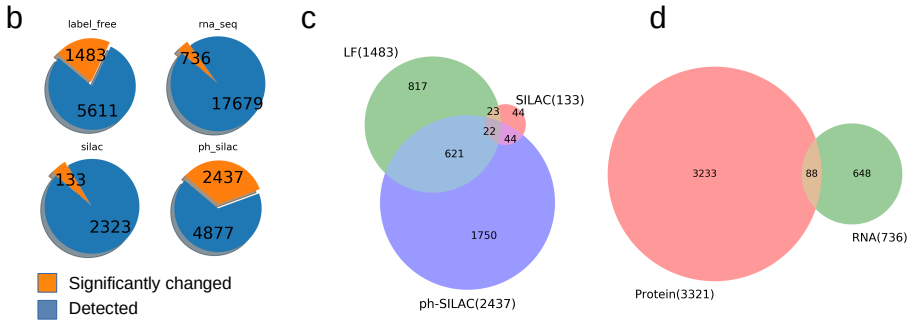


Figure 2.5: **Overlap between experimental measurements.** a.) Table of time points per experimental platform that were measured for the bendamustine study. b.) Summary of number of significant changed species vs detected. c.) Venn diagram comparing ph-silac, label-free, and silac. d.) Venn diagram comparing proteomics (ph-silac, label-free, silac) vs RNAseq.

In total, we detected 54 818 unique molecular species across all platforms, of which 14 426 were significantly changed in at least one time point upon drug treatment. For the proteomics data, we saw 5 115 significant changed species in at least one time point from PH-SILAC, 1 653 from label_free, 133 from SILAC, and 736 from RNA-seq, as shown in Figure 2.5 b. Early changes versus control (untreated) cells were detected at the phosphorylation level; phospho-SILAC detected > 700 such changes. Phosphorylation events occurred relatively even across all sample time points while a gradual increase of changes occurred over time at the protein and RNA level. We calculated the overlap between species measured with each platform as shown in Figure 2.5 c and d. The highest overlap among platforms was between ph-SILAC and label_free (643 species). However, only 88 genes were significantly changed at both RNA and protein (ph-SILAC, SILAC, and label-free) levels. A majority of proteomic measured species were unique to a single platform ($>$2 500), while 688 species were measured in at least two platforms, and only 22 species were measured in all three platforms. Technical replicates and statistical significance suggest within-platform detection is generally reliable and repeatable, thus the low overlap between significant species changes across platforms demonstrates the value of integrative, multi-platform analysis.

We proceeded to further analyze the dataset with a custom MAGINE workflow (2.6). First, we constructed a signaling network centered around the significantly changed species. Next, we performed an enrichment analysis across 52 databases for each time point and experimental platform with the data. Each sample was arranged by up-regulated, down-regulated, and significantly changed species, resulting in 78 total samples across 12 time points and 3 change types. In total, there were 20 758 enriched terms across the 52 reference gene sets with an adjusted p-value ≤ 0.05. Manual exploration of 20 758 terms is not feasible, but using MAGINE, users are able to focus further exploration. Using the *EnrichmentResult* Class, we filtered the terms to only include ph-SILAC enrichment with the Reactome_2016 reference gene set for further analysis, resulting in 561 unique terms. Next, we required terms to be significantly enriched in 3 of 5 time points, resulting in 84 terms. To reduce redundancy of terms (e.g. ‘cell

cycle', 'cell cycle, mitotic'), we applied the *remove_redundancy* method, resulting in 17 terms shown in Figure 2.8. We created a heatmap of these terms to visualize their dynamics over time. Enriched terms include *cell cycle*, *processing of capped intron-containing pre-mrna*, *dna repair*, *apoptosis*, and *hiv infection*. As shown in Figure 2.8c, we are able to expand individual terms such as *cell cycle* to retrieve terms removed in the compression process such as *nuclear envelope breakdown*, *dna replication*, and *g2/m checkpoints*, all of which are known bendamustine responses (Leoni and Hartley, 2011). Thus, MAGINE allows users to easily filter and visualize enrichment results, and to reversibly compress enrichment terms for ease of interpretation.

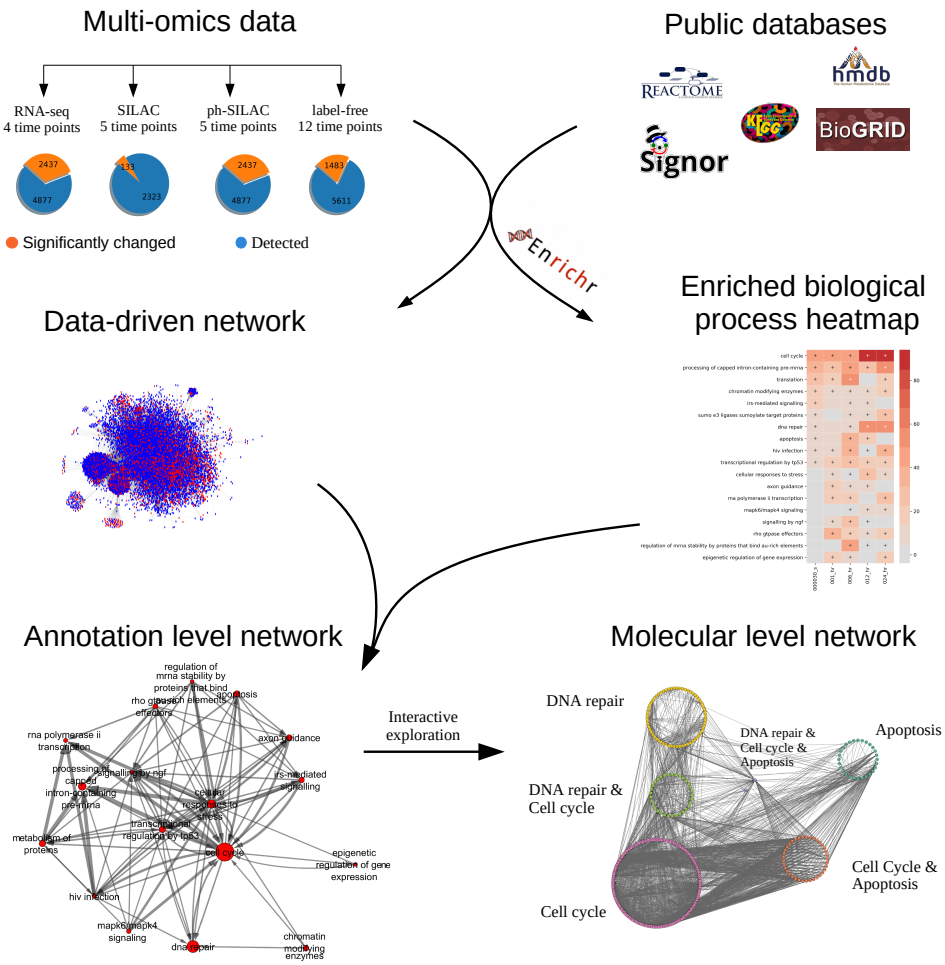


Figure 2.6: Multi-omics datasets (*top left*) are combined with public databases (*top right*) to generate detailed interaction networks (*middle left*) and enriched annotated gene set (AGS) heatmaps (*middle right*). The data-derived network is then pruned by extracting a subnetwork around genes from high-ranking AGS terms and collapsed to produce an annotation-level network (*bottom*), where each node is labeled by an AGS term and scaled according to enrichment of that term; the width of each edge represents the number of edges between the terms. The network can be viewed at each time point or as an animation.

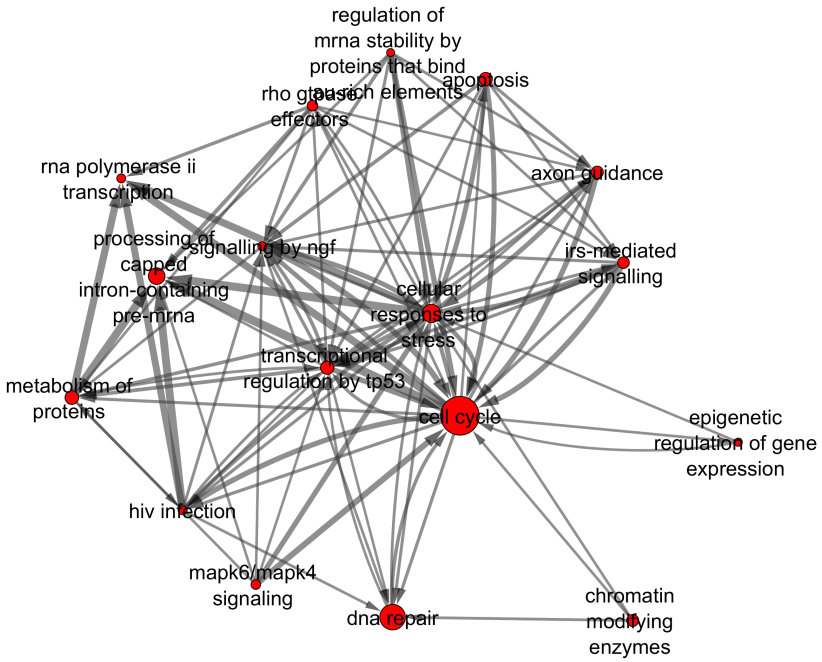
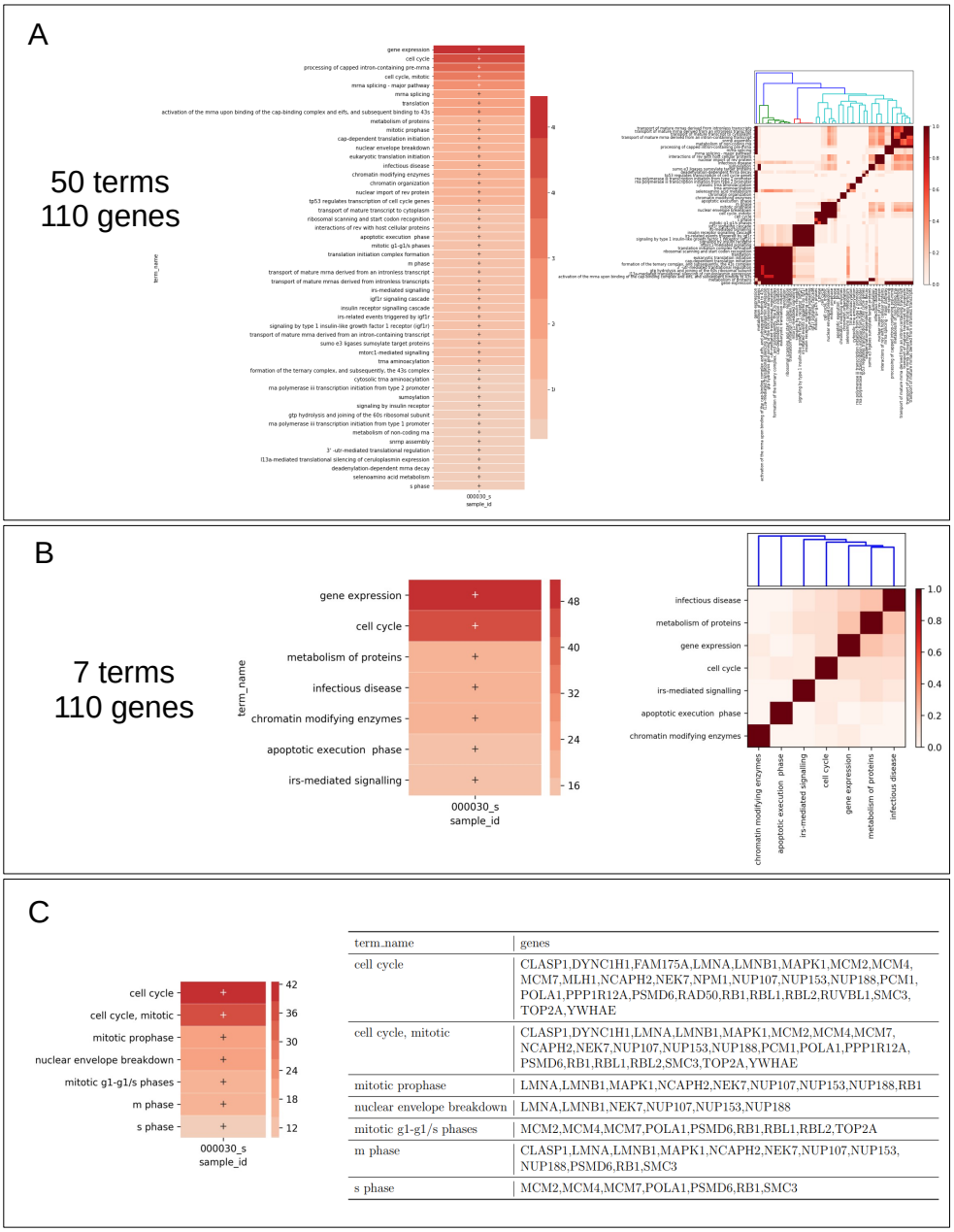


Figure 2.7: Zoom in of Fig. 2.6.



term_name	genes
cell cycle	CLASP1,DYNC1H1,FAM175A,LMNA,LMNB1,MAPK1,MCM2,MCM4, MCM7,MLH1,NCAPH2,NEK7,NPM1,NUP107,NUP153,NUP188,PCM1, POLA1,PPP1R12A,PSMD6,RAD50,RB1,RBL1,RBL2,RUVBL1,SMC3, TOP2A,YWHAE
cell cycle, mitotic	CLASP1,DYNC1H1,LMNA,LMNB1,MAPK1,MCM2,MCM4,MCM7, NCAPH2,NEK7,NUP107,NUP153,NUP188,PCM1,POLA1,PPP1R12A, PSMD6,RB1,RBL1,RBL2,SMC3,TOP2A,YWHAE
mitotic prophase	LMNA,LMNB1,MAPK1,NCAPH2,NEK7,NUP107,NUP153,NUP188,RB1
nuclear envelope breakdown	LMNA,LMNB1,NEK7,NUP107,NUP153,NUP188
mitotic g1-g1/s phases	MCM2,MCM4,MCM7,POLA1,PSMD6,RB1,RBL1,RBL2,TOP2A
m phase	CLASP1,LMNA,LMNB1,MAPK1,NCAPH2,NEK7,NUP107,NUP153, NUP188,PSMD6,RB1,SMC3
s phase	MCM2,MCM4,MCM7,POLA1,PSMD6,RB1,SMC3

Figure 2.8: A. Top 50 terms from Reactome for the 30 second ph-silac timepoint. The left figure shows the terms and their enriched values while the right shows the Jaccard index between all pairs. B. After filtering based on Jaccard index similarity. The resulting left figure has 7 terms, retaining all 110 genes that were contained in the terms from A. The right figure demonstrates that we have minimal overlap between corresponding terms. C. Demonstration of the collapsing of terms to a parent term. The left shows all terms that were collapsed from the term 'cell cycle'. The table on the right provides the genes that make up each term. Not that if a term was too broad ('cell cycle'), the user could remove it and repeat the process to maintain the level of detail which they require.

An enrichment analysis by itself provides no detail on how terms affect one another. Therefore, we constructed an annotated gene set network (AGN) (description of AGN described in methods), as shown in bottom left of Figure 2.6. Each nodes in the AGN represents an enriched term, and the edges between the nodes reflect the connectivity between species that are encompassed in the nodes. As shown in Figure 2.6 this network can be visualized over time, where the node size will be adjusted based on the enriched value of the term at each time point, providing a high level representation of the dynamics of signal flow. For example, *rna polymerase ii transcription* is regulated by four terms (*cellular response to stress*, *cell cycle*, *sumo e3 ligases sumoylate target proteins*, *hiv infection*, but only regulates a single term (*rho gtpase effectors*), suggesting the transcription is highly regulated in response to bendamustine. Additionally, the connectivity of the terms can provide insight into key driving processes in the response. For example, the *cell cycle* node is connected to all other nodes in the AGN, in agreement with its importance in regulating response to bendamustine (Leoni and Hartley, 2011). The AGN provides a coarse representation of the data that allow us to visualize the major processes and how they regulate one another.

Importantly, the AGN simplifies the true network structure in order to allow users to identify terms of importance that can then be used as a starting point for more detailed exploration. For example, to understand the data's ability to capture the canonical mechanism of bendamustine, we extracted a molecular network focused on the terms most commonly associated with bendamustine response (Leoni and Hartley, 2011) : *dna repair*, *cell cycle*, and *apoptosis* as shown in the bottom right of Fig 2.6 and center left of Figure 2.9. There is a total of 65, 146, 28 genes for the terms *dna repair*, *cell cycle*, and *apoptosis* respectively, 33 genes classified as both *dna repair* and *cell cycle*, 37 genes were classified as *cell cycle* and *apoptosis*, two genes in all three terms, while no genes were classified as both *dna repair* and *apoptosis*. We saw higher connectivity between *dna repair* and *cell cycle* (132 outgoing and 117 edges) and *cell cycle* and *apoptosis* (154 outgoing and 609 incoming edges) compared to *dna repair* and *apoptosis* (24 outgoing and 56 incoming edges). This agrees with the canonical understanding in how *dna repair* leads to changes in the *cell cycle*

and ultimately *apoptosis*, however also sheds light on the complexity of the cross talk between terms that make the signal non-linear (there is not a clear sequence of events).

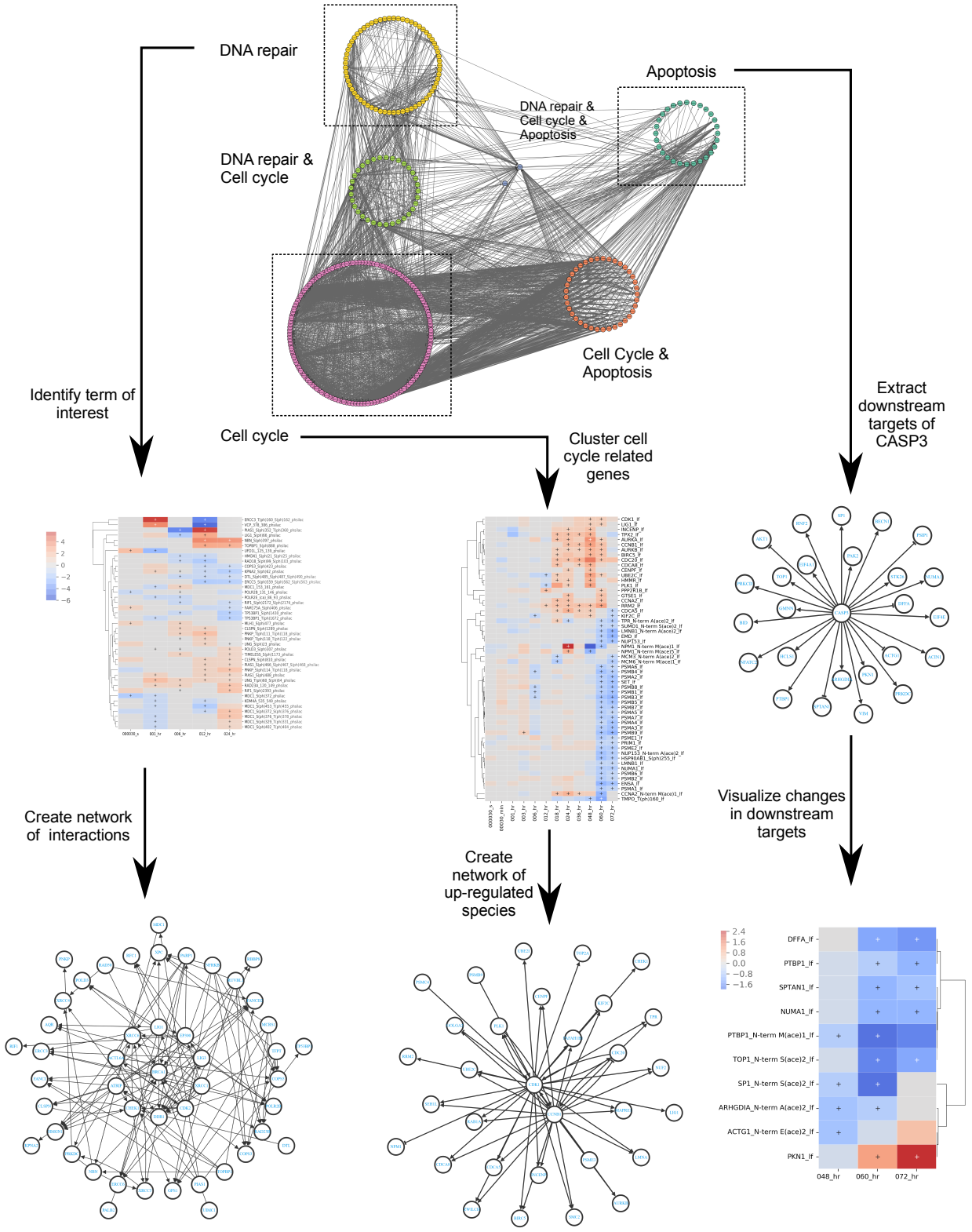

DNA repair
Apoptosis
DNA repair &
Cell cycle &
Apoptosis
DNA repair &
Cell cycle
Cell Cycle &
Apoptosis
Identify term of
interest
Cell cycle
Cluster cell
cycle related
genes
Extract
downstream
targets of
CASP3
Create network
of interactions
Create
network of
up-regulated
species
Visualize changes
in downstream
targets
DFFA_lf
PTBP1_lf
SPTAN1_lf
NUMA1_lf
PTBP1_N-term M(ace)1_lf
TOP1_N-term S(ace)2_lf
SP1_N-term S(ace)2_lf
ARHGDIA_N-term A(ace)2_lf
ACTG1_N-term E(ace)2_lf
PKN1_lf
048_hr
060_hr
072_hr

Figure 2.9: **Custom workflows for exploring known biological processes of bendamustine.** Molecular network constructed from DNA repair, cell cycle, and apoptosis classified species, the canonical processes involved in bendamustine response (*center left*). The network is an expanded version of the annotated gene network from Figure 2.6. From the term "DNA repair", MAGINE can visualize (*top left*) as well as construct a signaling network corresponding to the terms molecular species(*top right*). Due to CASP3 importance in executing apoptosis, we constructed a network of downstream species (*top right*). We used these species to filter our experimental data to generate the heatmap shown in *center right*, Only one species is up-regulated (PKN1) while eight species are down-regulated, suggesting CASP3 activation. Subsetting and visualizing the experimental data based on cell cycle related genes over time *bottom left*. We then constructed a molecular network based on CCNB1 and CDK1 (*center right*), two genes responsible for G2-M transition of cell cycle.

We can further explore the data by leveraging MAGINE's ability to extract out subsets of the data and focus on molecular level resolution of interactions among thing, simplifying the complexity. For example, based on the term *dna repair*, we extracted 173 phosphorylation events across 80 proteins, 43 changes in protein expression, and 5 changes of RNA expression. Of these, we identified proteins involved in DNA ligase (LIG1, LIG2), required to alleviate ICLs, double stranded break response MRN complex (RAD50, MRE11A, NBN), suggesting that the ICLs are corrected during DNA replication creation of double stranded breaks. We also saw proteins involved with base excision repair pathways (DDB1, XRCC1, XRCC2, XRCC5, XCCC6) and nucleotide excision repair (ERCC3, ERCC5, ERRC6), suggesting the the pyramide analog properties of bendamustine also contribute to DNA damage in addition to ICL. This allows users to identify molecular species specific to their drug and by looking at interactions can provide insight into targets to increase/decrease the desired effect. Another example, as shown in Figure 2.9, is our ability to extract out and visualize changes in cell cycle status. For cell cycle, we saw 444 phosphorylation events across 153 proteins, 111 changes in protein expression, and 17 changes of RNA expression. We clustered protein expression changes and create an influence network between up regulated species. We saw up regulation of CDK1, AURKA, AURKB, CCNB1, CDC20, and PLK1. CDC20, CDK1, and CCNB1 regulate chromosome separation while expression of AURKA/AURKB/PLK1 regulate progress through G2-M checkpoint. Without this checkpoint, activated apoptotic signals could lead to Mitotic catastrophe during metaphase/anaphase transition, a mechanism known to occur in response to bendamustine (Leoni and Hartley, 2011). Finally, we can use MAGINE's *Subgraph* module to explore downstream targets of CASP3, an effector caspase that cleavage of proteins marks commitment to apoptosis (Slee et al., 1999). Though our data did not measure changes in CASP3, we wanted to test if its downstream targets were changed in any capacity. As shown in the middle right of Figure 2.9, we created a network of down stream targets of CASP3 and visualize their expression levels over time. We saw 9 proteins that were down regulated only and one protein up regulated at late time points, after apoptosis was induced. Thus, despite not measuring changes in CASP3,

MAGINE allowed us to gather evidence suggesting its activation, although this remains subject to be further experimental validation.

2.1.2.2.1 Bendamustine affects HIV infection-related genes Throughout the exploration of the HL-60 response to bendamustine dataset, we noticed that the term *HIV infection* was a significantly enriched ontology term that apparently played no role in the cell-death response associated with bendamustine. However, since HIV infection hijacks and modulates cell cycle and DNA repair Chaurushiya and Weitzman (2009); Humphries and Temin (1972), we examined whether a plausible explanation of the enrichment in proteins associated with HIV infection is the overlap of underlying species with these expected biological process terms. The data showed that this overlap did not completely account for the finding (Figure 2.10). Proteins down-regulated in response to bendamustine marked with the *HIV infection* term include those related to the nuclear pore (NUP107, NUP133, NUP153, NUP155, NUP160, NUP188, NUP210, NUP214, NUP35, NUP37, NUP50, NUP62, NUP88, NUP93, NUP98) as well as regulators of the trafficking through the nuclear pore (RANBP1, RANBP2). Perturbations to RANBP2 are known to disrupt the HIV virus' ability to shuttle the HIV-1 Rev protein between the cytoplasm and nucleus, making it a potential target for inhibition of HIV Zhang et al. (2010). Additionally, CCNT1, a co-factor of the HIV-1 Tat protein necessary for full activation of viral transcription, is down-regulated at the 60 and 72 hour time points. Thus, the ability of bendamustine to perturb HIV infection-related genes could have clinical relevance. While treatment with bendamustine (for other disease indications) in HIV-positive patients is rare, case reports state that two HIV-positive patients with chronic lymphocytic leukemia received bendamustine without adverse outcomes Shimada et al. (2015). Therefore, unbiased analysis of this dataset enabled side effects exploration that could lead to further leads in HIV treatment. Although such results clearly require further validation, they demonstrates the ability of MAGINE to identify non-canonical cellular pathways effected by drug treatment. Since this study was performed without the ability for valdiation or further exploratory experiments, we were unable to test any of our findings.

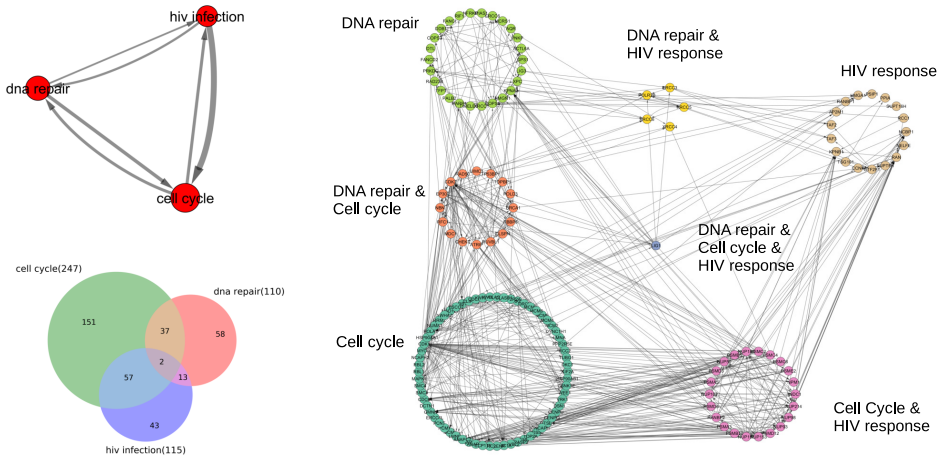


Figure 2.10: **Similarity between Bendamustine and HIV infection.** An unexpected outcome of the analysis revealed that proteins related to HIV infection were modulated with bendamustine. HIV infection depends on regulating cell cycle and DNA repair in order to hijack the replication machinery. Yet, proteins classified as only HIV infection, that is outside of DNA repair and cell cycle, are somehow regulated by bendamustine. An annotated gene set network of the interaction between *dna repair*, *hiv infection*, and *cell cycle* (top left). The overlap between genes labeled in each pathway labeled by Reactome (bottom left). Molecular interaction network of all significantly changed species grouped according to their classified pathway (right).

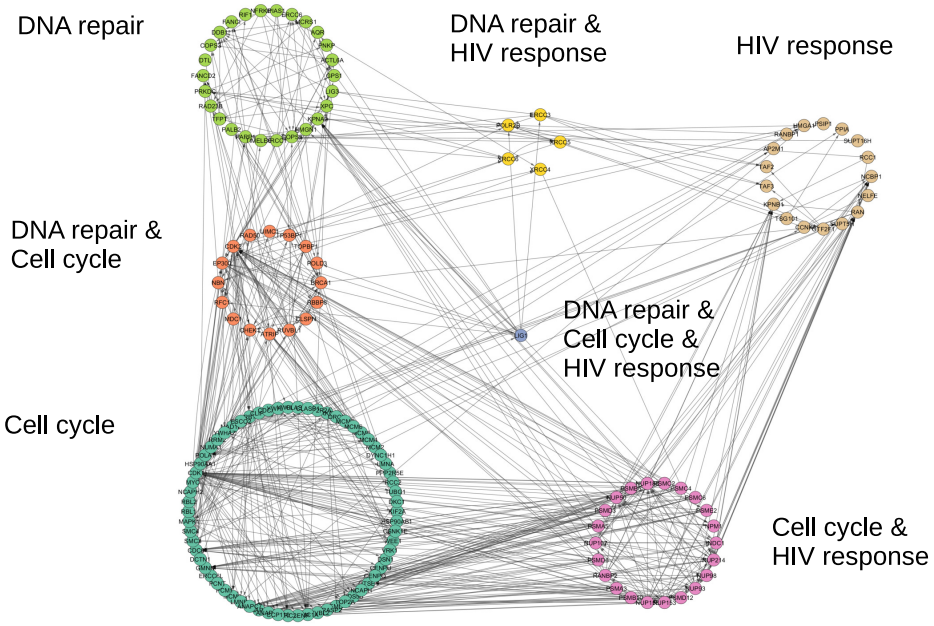


Figure 2.11: **Zoom in of Fig 2.10.** The molcular interaction between *dna repair*, *cell cycle*, and *hiv infection* related genes.

2.1.3 Methods

2.1.3.1 MAGINE software implementation

MAGINE is implemented as Python package. It leverages several existing Python packages including *pandas* (McKinney et al., 2010), *NumPy* (Van Der Walt et al., 2011), *Bioservices* (Cokelaer et al., 2013), *Matplotlib* (Hunter, 2007), *Seaborn* (Waskom et al., 2017), *matplotlib-venn*, and *networkx* (Hagberg et al., 2005) packages. Optional but recommended dependencies include *Jupyter notebook* (Kluyver et al., 2016), *python-igraph* (Csardi et al., 2006), *py2cytoscape* (Ono et al., 2015), and *plotly* (Sievert et al., 2016) Python packages and *graphviz* (Ellson et al., 2001) and *Cytoscape* (Shannon et al., 2003) software for additional visualization features.

2.1.4 Discussion

Experimental advances in omics data generation has provided a wealth of data and opportunities to advance currently available tools. The most common analysis types include network and enrichment analysis, which were designed for single sample, single 'omic' studies. Although network construction and enrichment analysis have been widely used, tools allowing their merger and utilization in multi-sample, multi-omics data have been lacking.

The software presented here provides new opportunities to integrate multi-sample omic data with enrichment analysis and data specific signaling networks within a single environment. MAGINE provides an intuitive flow between enrichment analysis, molecular interaction networks, and experimental data. It enables users to construct custom analysis protocols in Jupyter notebooks that are both reproducible and transferable. Recently, the Biojupies (Torre et al., 2018) and PaDua (Ressa et al., 2019) packages were created to streamline RNAseq and phopsho-protemoimcs analysis in Jupyter notebooks. We expect such approaches utilizing Jupyter notebooks to increase and see MAGINE as an ideal addition to such pipelines.

Importantly, MAGINE saves users time by automating enrichment analysis across multiple

samples, removing the tedious task of manually uploading data to web-based tools. For instance, to recreate the above analysis through EnrichR, it would have required *78 uploads* of gene lists, *78 downloads for each* of the 52 libraries, manual collation of the results, and some form of manual visualization and analysis. Instead, MAGINE performed all enrichment analysis with one function call, retrieving and storing the results in a reproducible and transparent manner, as well as performed down stream analysis, all of which documented and packaged for other users.

Additionally, since multi-omics data provides more cellular coverage than typical *in vitro*/in cell measurements (western blots), measuring significantly changed species that were not "expected" is common, resulting in enriched terms that seem out of place. This could be due to noise in measurements, multiple classifications of species, or completely new context in biology. Further exploration of such terms can provide critical context in deciphering these signals but rate limiting due to the time required for exploration. Here we demonstrated MAGINE's ability to quickly explore *hiv infection* and its relationship to known bendamustine responses. We expect this utility to be useful in identifying and exploring non-canonical cellular pathways that are often disregarded or ignored.

Our results show the complexity in interpreting multi-omics data as the response to bendamustine is not a clear linear chain of events. Single time point measurements might not capture an entire mechanism, with some events occurring early (phosphoylation events of DNA repair proteins), some occur later (up-regulation of cell cycle proteins), and some occur late (cleavage events of apoptotic CASP3). Interactive exploration of the data enables users to piece together the mechanism and design further experiments to validation.

In summary, MAGINE enables users to easily switch between enrichment analysis, networks, and experimental data in an iterative cycle. This allows users to vary resolutions dependent on the question at hand, generate new hypothesis, and switch resolutions to focus exploration. As improvements with omic data generation allow multiple sample acquisitions per perturbation, we envision the need for tools such as MAGINE will grow.

2.1.4.1 Limitations

One of the main bottlenecks we encountered with the above mentioned data and analysis encircled the limited ability to incorporate metabolomics data into our analysis pipeline. enrichR allows an easy point of access for gene based enrichment analysis, allowing large lists to be distilled into summarized information. As annotations and reference databases become developed, we envision easily extending MAGINE to incorporate them into the pipeline.

2.1.5 Notes

All scripts, Jupyter notebooks, and data are available online at https://github.com/LoLab-VU/MAGINE_Supplement_notebooks. Scripts were created using Python 3.7 and tested on Windows 10 x64. Installation of MAGINE can be performed with *pip install magine*.

2.1.6 Additional Methods

2.1.6.1 Data format

MAGINE is designed for fold change quantification datasets. These data are loaded from a single file in a tabular format, with columns identifying the sample, time point, and experimental platform. Due to the large number of possible experimental platforms, we generalized naming of columns to support a wide variety of data types. MAGINE requires the following column labels: *identifier*, *label*, *species_type*, *significant*, *fold_change*, *p_value*, *source*, and *sample_id*, with a sample table shown in Table 2.4. This format uses a *stacked* data file notation (*denormalized*, in database parlance), which provides ease of editing and compatibility (a single, text format table is all that's needed) at the expense of data redundancy. For *identifier*, we use HGNC (Bruford et al., 2007) and Human Metabolite Database (HMDB) (Wishart et al., 2007) IDs for genes and compounds respectively (we also provide tools to map other identifiers to these formats). The *label* column can be used to store any other information, such as post-translational modifications, aliases,

or molecular weights. The *source* and *sample_id* are used to label the experimental platform and sample condition (i.e. timfe point). The *species_type* column is used to state if the species is a metabolite or gene (we use 'gene' to label all possible gene products and allow the *source* column to specify the specific type). The *fold_change* column and *p_value* column are for the fold-change and statistical significance of the data compared to control (supplied from the raw data processing step performed before data are loaded into MAGINE). Note that the *p_value* should be the adjusted value if using any form of multiple hypothesis testing. Finally, the *significant* column is used to identify if a given measurement is significant compared to control. The definition of significant is deliberately left to the user: in the situation where each experimental platform is handled by a different team, those teams can apply their own standards or software to defining significance, if desired.

Table 2.4: Example data input used in MAGINE

identifier	**label**	**type**	**significant**	**fold_change**	**p_value**	**source**	**sample_id**
BAX	BAX_S(ph)292	gene	True	4	0.01	SILAC	01hr
HMDB00012	Deoxyuridine	metabolite	True	-2	0.01	HILIC	02hr

Once loaded, data are stored in an instance of MAGINE's *ExperimentalData* class, built on the Pandas DataFrame with added methods designed for biological data. Our goal was to simplify and provide shorthand properties/methods to minimize the code needed to perform common operations such as filtering, querying, aggregating, and visualization of the data. A summary of these methods is shown in Table 2.1 and example usage can be found online or in Notebook 5.1.1). Python *properties* are constructed around the *significant*, *source* and *sample_id* columns, allowing users to gather only significant changes (*data.sig*), extract only label_free proteomics data (*data.label_free*), extract out only sample_id rows labeled "wild_type" (*data.wild_type*), or filter up regulated fold changed species (*data.up*). Importantly, filters can be chained together to create complex queries, with less complex syntax, and methods directly applied to the resulting information (*data.label_free.sig.up.heatmap()*). Identifiers or labels of these queries can then be aggregated (*data.id_list* or *data.label_list*) and passed to enrichment analysis or to construct a network. Additionally, the class has direct access to common biological plotting functions such as

volcano plots (*data.volcano()*), heatmaps (*data.heatmap()*), or individual species plots (*data.plot_species(...)*).

2.1.6.2 Enrichment term aggregation

MAGINE contains a method, *remove_redundancy*, for reducing the number of gene enrichment terms by aggregating redundant terms. First, it calculates the ratio of the sizes of the intersection and union (Jaccard index (Gilbert, 1972)) between genes within all term pairs. It then ranks all terms based on either their combined score (from EnrichR), number of genes in the term, or p-value. Starting from the highest ranked, it compares all lower ranked terms and remove them if their similarity is above a user defined threshold, as shown in Text 1 and demonstrated in Figure 2.8. This allows the user to minimize the number of total terms while maintaining a level of information content that preserves total information.

2.1.6.3 Annotated gene set network construction

Enriched terms provide us with big picture information about processes and when they occur, yet they do no provide us with details about the interconnection among them. *Why is DNA damage repair up regulated at 1 hour and apoptosis up-regulated at 24? Is there a cause and effect? Are they independent?* To answer these questions, we created an algorithm to extract trimmed signaling networks derived from the enriched terms. The terms can be selected based on expert knowledge, rank of enrichment, all compressed terms, or any other criteria.

Once the terms are selected, we find the nodes in the network for each term. From there, we search through all possible combinations of pairs between the terms. For example, if term 1 has genes (A, B, C) and term 2 has (D, E), we count the number of edges from the possible sets ((A, D), (A, E), (B, D), (B, E), (C, D), (C, E)) that are found in the network edges. We then do the reverse (term 2 to term 1). If a node is in both sets, we consider only the edges that connect the other term, not edges that are within the term. This is demonstrated in 2.12 (N). This results in one coarse-grained network, where nodes are terms and the edges are frequency of edges between any

two terms, and the fine-grained network, which contains the species and the connections between them. By using both, we are able explore at a high level and zoom in on detail when required.

	Enrichment	Genes
Term 1	10	A, B, C
Term 2	15	C, D, E

		A to B	B to A
# edges	Actual	4	1
	Possible	6	6

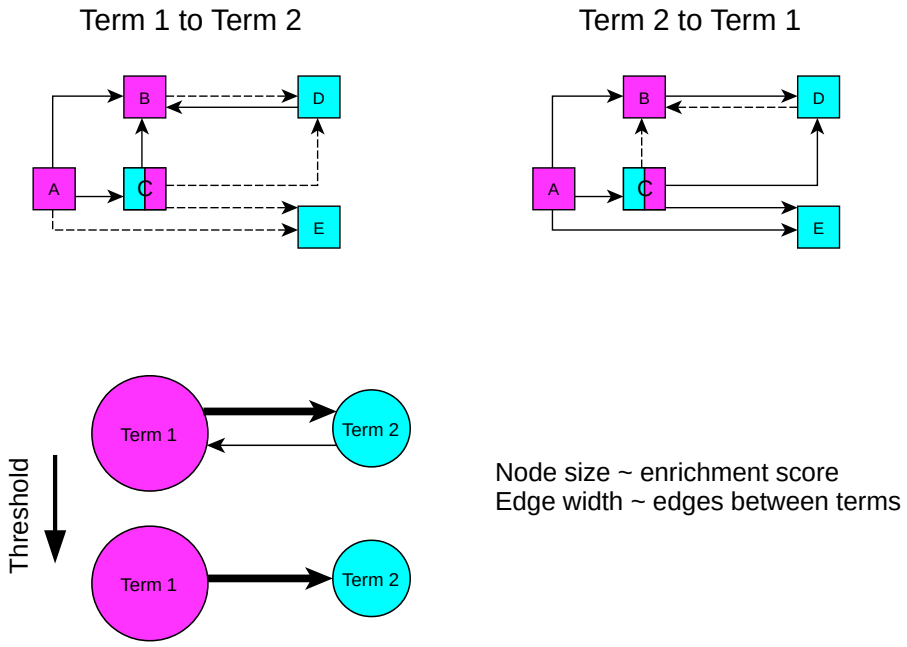


Figure 2.12: **Annotated gene set network construction** Nodes belonging to Term 1 are labeled as pink, Term 2 as blue, and nodes in both are colored both. We calculate the number of edges between nodes in Term 1 to Term 2 and Term 2 and Term 1. We then create nodes for each Term, with the size of the node corresponding to the enrichment value. Edges are created between the terms and width set according to the number of edges between the terms. A thresholding can be applied to remove edges with low number of connecting edges.

2.1.6.4 Software availability and requirements

MAGINE is released as an open source software under the GNU General Public License, version 3. Source code for *MAGINE* can be found at github.com/LoLab-VU/Magine. Full documentation can be found at magine.readthedocs.io. MAGINE has a full software test suite, which is executed using continuous integration, and checked for code coverage. The project is version controlled on GitHub. We encourage the community to post issues, questions, enhancement requests, and code contributions.

Enriched term compression

We created a method that filters the number of enriched terms based on redundancy in gene content.Starting with a ranked array with columns term_name, rank, and set of genes. We then rank all terms based on either their combined score (from EnrichR), number of genes in the term, or p-value. Starting from the highest ranked, we compare all lower ranked terms and remove them if their similarity is above a user defined threshold. We use the Jaccard index(Gilbert, 1972), which is the size of the intersection divided by the size of the union of any two sets, as our similarity metric.

Using this approach, we are able to minimize the total number of terms while maintaining their enrichment content, as shown in 1 and demonstrated in figure 2.8. It should also be noted that we can take the compressed terms and back calculate from the original enrichment array which terms were removed as demonstrated in 2.8 C.

Algorithm 1 Enrichment compression

```
1: procedure FILTER
2:     entry = [term, rank, geneSet]
3:     array = sorted list of entries                    ▷ EnrichmentResult class
4:     to_remove = set()
5:     for i in array do
6:         for j in array not including i do
7:             score = jaccard_index(i.geneSet,j.geneSet)
8:             if score > threshold then          ▷ i contains more information than j
9:                 to_remove.add(j)                    ▷ remove to_remove from array
```

2.1.6.5 Experimental methods

Briefly, HL60 cells were plated and exposed to 100 μM bendamustine in triplicate for each time point as described previously (Gutierrez et al., 2018). After introduction of drug, measurements were collected at 12 time points ranging from 30 seconds to 72 hours across 6 experimental platforms shown in Fig. 2.5a. Raw data was processed according to a previously described protocol (Norris et al., 2017). Sample preparation for transcriptomics, label-free proteomics, and metabolomics analyses were performed as previously described (Gutierrez et al., 2018). SILAC and phSILAC samples were prepared in a similar manner as reported (Norris et al., 2017), with the exception that SILAC samples were run by 1D chromatography rather than MudPIT. Each data set was then formatted and combined into a single CSV to be used with MAGINE.

Chapter 3

Advancements in mechanistic modelling

3.1 Background

3.1.1 Bottoms up approach

Kinetic modeling of complex biochemical systems is central to the emerging field of systems biology (Kitano, 2002; Le Novere, 2015). Kinetic models require definition of numerous free parameters, usually obtained by calibration to experimental data, that specify initial species concentrations and kinetic rate constants. This process involves simulating the model and comparing its output to experimental data. Generally this has been done by hand (Albeck et al., 2008), through exhaustive, computationally inefficient methods (Kim et al., 2010), custom single use code (Lopez et al., 2013), or recently through packages designed for biological models (Shockley et al., 2018). Biological models can have many parameters that can vary over order of magnitudes of space, leading to near infinite number of parameter sets that fit the model equally well (Gutenkunst et al., 2007).

Once calibrated, a model should be analyzed for its sensitivity and predictive power over ranges of parameter values (Fisher and Henzinger, 2007).This could result in identifying molecular species that negatively affect the desired outcome (key anti-apoptotic proteins), co-druggable targets (amplify death signal through alternative signaling pathways), or eliminate drug targets due to the models insensitivity to changes in their concentration.

Both model calibration and analysis can require thousands to millions of model simulations for statistical convergence and significance (Gutenkunst et al., 2007; Eydgahi et al., 2013). In many cases, the computational expense of simulation at this scale makes detailed model analysis infeasible.

This chapter addresses these three issues. First, in Section 3.2, I introduce simulation methods

that offer orders of magnitude more simulation faster simulations. In Section 3.3, I introduce a particle swarm optimization toolkit that allow PySB users an efficient way to link their models to experimental data. In Section 3.4, I introduce a sensitivity analysis approach and apply it to both models of EARM.

3.1.2 Stochastic and deterministic simulations

As mentioned above, both model training and characterization requires hundreds to thousands of simulations to be performed. Models can be simulated deterministically or stochastically. Which formalism to use is based on model assumptions. If the system is well mixed, can be regarded as continuous concentration of species, and reactions occur frequently, then the model can be simulated using deterministic methods. This is done by solving a system of ordinary differential equations(ODEs). ODEs are typically solved using numerical methods such as LSODAPetzold (1983). ODE simulations are more cost efficient than stochastic methods. However, when having to do thousands of simulations, simulation of the model is a rate limiting step.

If the model has low species copies or infrequent events (low rates), then the model should be formulated stochastically, discussed in section 3.2.2. This is often done using the Gillespie algorithm (Gillespie, 1976), otherwise called the stochastic simulation algorithm (SSA). Each stochastic simulations take substantially longer than an ODE simulation. In addition, hundreds to thousands of simulations are required to properly explore the state space of the model. Though many of these assumptions have not been extensively validated, the computational cost of the alternative approach to ODEs (stochastic simulations) has allowed this assumption to pass without notice (Spencer et al., 2009).

Thus, there is a need to decrease the time spent simulating models. In sections 3.2.1 and 3.2.2, I introduce simulation methods that increase the number of simulations to be performed compared to current methods. These improvement have direct impact on the time required for both parameter optimization and sensitivity analysis.

3.1.3 Extrinsic apoptosis reaction model (EARM)

In this section, I introduce extrinsic apoptosis reaction model (EARM), a family of models of extrinsic apoptosis that links a death signal to cell destruction (Albeck et al., 2008; Lopez et al., 2013). The models are encoded in PySB, a Python package for rule based modeling (Lopez et al., 2013). Using PySB, a model can be constructed in a modular manner, allowing the ability to construct various models in a systematic way.

Tumor necrosis factor (TNF)-related apoptosis-inducing ligand (TRAIL) binds to a death receptor on the cell membrane, leading to the formation of death-inducing signaling complex (DISC), which if not inhibited by FLIP can cleave and activate caspase-8 (CASP8). Active caspase-8 truncates Bid (into tBid), targeting it to the mitochondrial outer membrane. When at the membrane, tBid recruits cytoplasmic BAX to be inserted into the membrane, where tBid can further activate BAX and the closely related membrane-bound protein BAK. Active BAX/BAK then homo-oligomerize, causing pore formation in the mitochondria. This process, known as mitochondrial outer membrane permeabilization (MOMP), is the "point of no return" in apoptosis (Kuwana et al., 2002; Certo et al., 2006). MOMP releases mitochondrial proteins, in particular cytochrome c (CYTOC) and Smac. CYTOC binds to the cytosolic protein Apaf-1 and to caspase-9, forming the "apoptosome," which activates caspase-3 (CASP3). Smac binds to XIAP, preventing XIAP from binding and inhibiting active CASP3. CASP3 activation marks the beginning of the end: DNA and structural proteins degrade and the cell eventually collapses in a controlled manner (Taylor et al., 2008).

Apoptosis is extensively regulated via the BCL-2 family of proteins (Volkmann et al., 2014). Their main functions are to control mitochondrial outer membrane permeabilization (MOMP) (Chipuk et al., 2008; Martinou and Youle, 2011; Kuwana et al., 2002), considered the most critical step of commitment to apoptosis (Brenner and Mak, 2009; Borner and Andrews, 2014; Elkholi et al., 2014; Gillies et al., 2015).

In this work, I focus on EARM 1.0 and EARM 2.0-M1a 'embedded together' model. The main differences of these models are their treatment of the BCL-2 proteins. EARM 2.0 is an expanded

version of EARM 1.0 (Albeck et al., 2008). In EARM 1.0, the pro-apoptotic (BAX and BAK) and anti-apoptotic species (BCL2, BCL-XL, MCL1) are represented as a single species, BAX and BCL2 respectively. The implications of this simplification will be discussed in section 3.4.1.

3.2 Model simulation

3.2.1 Deterministic simulation

3.2.1.1 Enabling high throughput simulations using GPUs

[1] Kinetic modeling of complex biochemical systems is central to the emerging field of systems biology (Kitano, 2002; Le Novere, 2015). Kinetic models require definition of numerous free parameters, usually obtained by calibration to experimental data, that specify initial species concentrations and kinetic rate constants. Once calibrated, a model should be analyzed for its sensitivity and predictive power over ranges of parameter values (Fisher and Henzinger, 2007). Both model calibration and analysis can require thousands to millions of model simulations for statistical convergence and significance (Gutenkunst et al., 2007; Eydgahi et al., 2013). In many cases, the computational expense of simulation at this scale makes detailed model analysis infeasible.

Recently, efforts have been made to leverage the highly parallel structure of graphics processing units (GPUs) to accelerate scientific computations (Dematté and Prandi, 2010; Nobile et al., 2016). GPUs are well suited for applications in which the same arithmetic operations are applied to many independent data elements, e.g., solving independently parameterized systems of ordinary differential equations (ODEs). GPU-based kinetic simulators thus hold great promise for accelerating tasks such as model calibration and analysis, but are challenging for non-experts to use because they require specialized settings and inputs.

We have created a user-friendly interface between the GPU-based kinetic simulator cupSODA (Nobile et al., 2013, 2014) and PySB, a Python-based modeling and simulation

[1] Section modified from Harris et al. (2017)

platform (Lopez et al., 2013). cupSODA is built around the well-known adaptive stiff/non-stiff ODE integrator LSODA (Petzold, 1983). It is designed to perform thousands of parallel simulations, each independently parameterized, of mass-action kinetic models by leveraging the high-performance memories on the GPU, specifically the cached and non-mutable *constant memory* and the low-latency on-chip *shared memory*. PySB is a rule-based modeling (Chylek et al., 2014, 2015) platform for constructing and analyzing complex models of biochemical systems. Models can be constructed in native Python code or imported from various formats, including the Systems Biology Markup Language (SBML) (Hucka et al., 2003). PySB leverages powerful libraries within the Python ecosystem, such as NumPy, SymPy, and SciPy (Pérez et al., 2011), and provides user-friendly interfaces to numerous third-party simulation and analysis tools, including BioNetGen (Faeder et al., 2009; Harris et al., 2016), KaSim (Suderman and Deeds, 2013), and StochKit (Sanft et al., 2011).

Below, we briefly describe the main features of the PySB/cupSODA interface and showcase its utility by performing run time and sensitivity analyses for three model systems of varying size (Table 3.1).

3.2.1.2 Features and Implementation

cupSODA is designed to exploit the massive parallelism of the CUDA architecture (Nickolls et al., 2008). To run simulations with cupSODA, one must construct multiple input files containing, e.g., the reaction stoichiometries and the initial species concentrations and rate parameter values for each specified simulation. Numerous simulator-specific parameters must also be defined, such as the number of CUDA "blocks" to use and the desired cupSODA memory configuration (see Supplementary Information). The number of simulations that cupSODA can run in parallel is limited by the number of CUDA "cores" on the GPU (usually a few thousand; see Supplementary Table S1), but the number of simulations that can be loaded onto the GPU at one time is usually many more than this, limited by the available memory (Nobile et al., 2013, 2014).

The PySB/cupSODA interface simplifies the use of cupSODA via a `CupSodaSimulator`

class, available within the PySB package. The class constructor accepts the following arguments:

- `model`: A PySB model object (*required*)
- `tspan`: A list of output time points (*default*: `None`)
- `initials`: A list or dictionary of initial species concentrations for each simulation (*default*: `None`)
- `param_values`: A list or dictionary of rate parameter values for each simulation (*default*: `None`)
- `verbose`: Verbose output (*default*: `False`)

The `CupSodaSimulator` constructor also recognizes numerous keyword arguments (kwargs), such as `n_blocks`, the number of CUDA blocks, and `memory_usage`, the desired memory configuration. Importantly, default values are defined for each kwarg, removing the need for user input. For example, if a user-defined value is not provided, the number of CUDA blocks is automatically calculated by querying the specifications of the GPU in use.

The `CupSodaSimulator.run()` method performs the simulations by constructing the cupSODA input files and invoking cupSODA as a subprocess (the method takes `tspan`, `initials`, and `param_values` as optional arguments). Additionally, the method reads into a three-dimensional array the results of the simulations (species time courses), which cupSODA outputs to (typically thousands of) separate text files. The user then has the ability to analyze and/or visualize the results using tools available within the Python ecosystem, e.g., plotting the time courses using the Matplotlib library (Pérez et al., 2011). For convenience, a `run_cupSODA` wrapper function has also been implemented that combines invocations of the `CupSodaSimulator` constructor and `run` method into a single step. A workflow diagram and example Python script using the `run_cupSODA` function are provided in Supplementary Figs. S1 and S2, respectively.

Table 3.1: Models used for PySB/cupSODA performance testing

Model	Species	Reactions	End time	N steps
Cell cycle Tyson (1991)	5	7	100	100
Ras/cAMP/PKA Besozzi et al. (2012)	33	39	1500	100
EARM Lopez et al. (2013)	77	105	20 000	100

3.2.1.3 Result

In Fig. 3.1A–C we compare the run time efficiency of PySB/cupSODA to the CPU-bound ODE integrator LSODA, available in the Python package SciPy (Oliphant, 2007), for three example models listed in Table 3.1. These include models of the eukaryotic cell cycle (Tyson, 1991), the Ras/cAMP/PKA signaling pathway in *Saccharomyces cerevisiae* (Besozzi et al., 2012), and extrinsically induced apoptosis in mammalian cells (EARM: extrinsic apoptosis reaction model) (Lopez et al., 2013). Run time comparisons show that in all cases SciPy/LSODA is faster for small numbers of simulations but PySB/cupSODA overtakes it for large numbers of simulations, achieving a maximum speedup of approximately one order of magnitude.

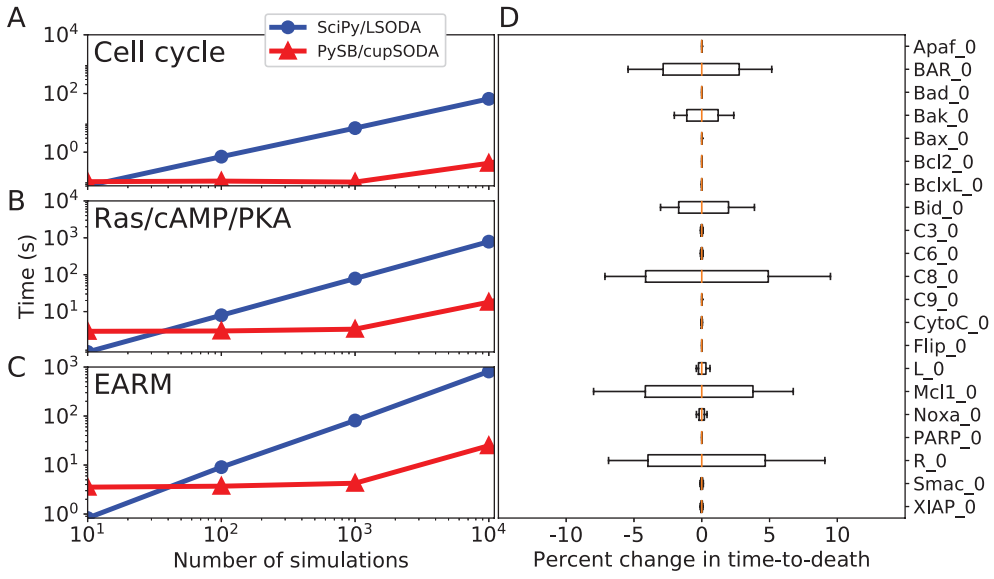


Figure 3.1: **Run time comparisons between PySB/cupSODA and SciPy/LSODA.**(A–C) Run time comparisons between PySB/cupSODA and SciPy/LSODA for the example models in Table 3.1 (all simulations performed with the same initial protein concentrations and rate parameters). (D) Sensitivity in time-to-death in EARM to variations (±20%; 25 410 total simulations) in the initial protein concentrations (gold lines are medians; boxes range from the first to third quartile; whiskers extend to the minimum and maximum values). PySB/cupSODA simulations were run using cupSODA 1.0.0 on a GeForce GTX 980 Ti GPU (2816 cores, 16 threads/block); SciPy/LSODA simulations were run on an Intel Xeon E5-2667 v3 @ 3.20 GHz CPU (see Supplementary Table S1).

For each model in Table 3.1, we also performed sensitivity analyses (Fig. 3.1D) by quantifying changes in defined model outputs to variations ($\pm$20%) in initial protein concentrations around a set of reference values. The ability to efficiently perform such analyses is critical since non-genetic variability within isogenic cell populations has been attributed to significant variations in protein concentrations across cells (Spencer et al., 2009). In Fig. 3.1D, we analyze the sensitivity in "time-to-death" in EARM (defined as the time at which Smac reaches 50% cleavage; demonstrated in Appendix Figure 5.5) for a specific set of rate parameters. Our results show that time-to-death is sensitive to the initial levels of six of the 21 proteins considered. Of particular interest is the sensitivity to Bak. The same analysis performed for a different set of rate parameters (Appendix Figure 5.7) shows insensitivity to Bak but sensitivity to Bax. This indicates that the model harbors at least two alternative pathways to apoptosis induction. The analysis comprised 25 410 total simulations and took $\sim$11 min with PySB/cupSODA and $\sim$35 min with SciPy/LSODA (for both parameter sets). Similar accelerations were seen for the cell cycle and Ras/cAMP/PKA models (Appendix 5.3 and 5.1).

3.2.1.4 Conclusion

The PySB/cupSODA interface provides the modeling community with a high-performance GPU-based kinetic simulator, that can run thousands of parallel simulations on a common desktop workstation, within the easy-to-use framework of a full-fledged, open-source programming and analysis environment in Python. This will greatly accelerate and streamline the process of analyzing complex biochemical models for systems biology applications.

3.2.2 Efficient stochastic simulations

3.2.2.1 Introduction

Chemical kinetic simulations can be either deterministic or stochastic. Which formalism to use is based on model assumptions. If the system is well mixed(can be regarded as continuous concentration of species, and reactions occur frequently), then the model can be considered deterministic and represented as a set of ordinary differential equations(ODEs). Otherwise, if the number of particles are low or the reactions that occur are infrequent, the model should be described with the chemical master equation (CME). Solving the CME is analytically intractable for most systems, which has led to development of sampling or simulation method, most notably the Doob-Gillespie stochastic simulation algorithm (SSA) (Gillespie, 1976, 2007). The Gillespie algorithm, otherwise called the stochastic simulation algorithm (SSA), is one such method to sample the CME (Gillespie, 1976). One realization of the CME can be understood as one possible set of reactive events and the resulting changes in the molecular population. In order to approximate the CME, a large number of simulations of the SSA are required. This has made the use of the SSA computationally prohibitive for many problems (Gillespie, 2001; Rathinam et al., 2003; Gillespie, 2007; McCollum et al., 2006; Gillespie, 2001; Harris et al., 2009).

From a computational perspective, the independence of each individual simulation in the SSA formalism makes parallelization an appealing approach for simulation speedup. Software packages such as StochKit (Sanft et al., 2011) provide this form of parallelism using Central Processing Units (CPUs). In these implementations, the total simulation time decreases linearly with the number of CPU cores. However, for large and complex biochemical reaction systems, significant computational resources are required beyond typically available CPUs in a research setting. To address this limitation, researchers have increasingly turned to Graphical Processing Units (GPUs) to accelerate numerical simulations (Dematté and Prandi, 2010). Originally designed for rendering (thus the name), GPUs are equipped with potentially thousands of cores and perform the same calculations across data very efficiently. Previous implementations of the

SSA algorithm in GPUs have already reported significant speedup, up to 200 times faster compared to CPU-based implementations (Li and Petzold, 2010). However, these implementations have faced challenges associated with writing and maintaining specialized code in hardware-specific programming languages, as well as restricting the access of GPUs for novice users.

To address this challenge, we developed GPU_SSA, an implementation of the SSA solver in Python that can be used within the PySB (Lopez et al., 2013) modeling framework developed in our lab, leveraging middle layer programming (Klöckner et al., 2012) that allows minimal user involvement and easier code maintenance. PySB enables users to encode biochemical reaction networks using a rule-based formalism in Python and supports import of models from other platforms including SBML (Tapia and Faeder, 2013). We believe that our implementation will enable quantitative biologists, that lack specialized GPU knowledge, to carry out SSA-based simulations routinely. Our implementation includes two versions of GPU-SSA simulators: a CUDA version, specific to NVIDIA branded GPUs; and an OpenCL version that enables the use of parallel hardware independent of hardware platform. We demonstrate the performance of the implementation version BioNetGen (Faeder et al., 2009) CPU implementation.

3.2.2.2 Implementation

For $\mathbf{N}(t) = (N_1(t), N_2(t), ..., N_n(t))$, where N_i is the number of molecules of type i and n is the number of distinct chemical species, the chemical master equation (CME)

$$\frac{dP(\mathbf{N},t)}{dt} = \sum_{r \in \mathscr{R}} a_r(\mathbf{N} - \nu_r) P(\mathbf{N} - \nu_r, t) - \sum_{r \in \mathscr{R}} a_r(\mathbf{N}) P(\mathbf{N}, t)$$

describes the change of all possible states $P(\mathbf{N},t)$ in which the system can be over time. Let $\mathscr{R}$ be the full set of all reactions occurring in the chemical system.

In Gillespies direct method (Gillespie, 1976), the time development of each possible state depends on the reaction propensities $a_r(\mathbf{N})$, which are the probabilities per unit time, that reaction

Algorithm 2 Gillespie's Direct Method

Set $t = 0$. Initialize the rate constants $c_1, \ldots, c_v$ and the initial molecule numbers $N_1, \ldots, N_u$.
while $t < T_{max}$ **do**
    Calculate each propensity $a_i(\mathbf{N}, c_i)$, $i = 1, \ldots, v$ based on the current state, $\mathbf{N}$.
    Calculate total propensity $a_{tot} = \sum_{i=1}^{N} a_i(\mathbf{N}, c_i)$.
    Generate two random numbers u_1, u_2
    Calculate time until next reaction $\tau = -log(u_1)/a_{tot}$
    if $t + \tau > t_{end}$ **then**
        end
    Select reaction $\mu = \min(k : \sum_{i=1}^{k} a[i] > a_{tot} u_2)$.
    Update $t += \tau'$
    Update N according to reaction μ $N = N + S^{\mu}$.
    Output N and t based on desired sampling.

$r \in \mathscr{R}$ occurs given that the composition of the system is of the form $\mathbf{N}$. The rate of change is updated via the stoichiometric vector ν_r, which describes the change in the numbers of each species as a result of given reaction r.

The propensities $a_r(\mathbf{N})$ can be directly calculated via the reaction rate κ_r. For a first-order reaction of type i, the propensity is $a_r = \kappa_r N_i$. The second-order reaction of the same type i has the propensity $a_r = \kappa_r N_i (N_i - 1)/2$, and for reactions of two different types i and j $a_r = \kappa_r N_i N_j$. The propensities at a certain time point are representing the probabilities of a reaction happening per unit time. To generate random reactive events, a random reaction time τ, which is exponentially distributed, and a random reaction μ have to be selected. Every time a reaction is firing, the state of $\mathbf{N}$ changes and therefore the propensities. Algorithm 2 shows the basic step of the direct method.

The GPU-SSA algorithm has been implemented using PyCUDA and PyOpenCL as a simulator class within the Python based framework for building mathematical models of biochemical systems PySB (Lopez et al., 2013). PySB enables the usage of rule-based languages such as BioNetGen (Faeder et al., 2009) or Kappa (Danos et al., 2007) within a domain specific programming language. Using PySB, the user can encode the chemical reactions of the biochemical system by representing each reaction with a rule. Users can then simulate their model using various simulators in PySB, including stochastic simulations with Kappa or

BioNetGen, network-free simulations with NFSim (Sneddon et al., 2011), or deterministic simulations using Scipy or cupSODA (Harris et al., 2017). The CUDASimulator is designed for NVIDIA GPUs while the OpenCLSimulator can be ran on GPUs (AMD or NVIDIA), CPUs, and other supported devices. Both simulators generate and compile model specific code to be executed on the devices. Memory allocation (for parameters, initials values, time courses) is automatically handled without requiring any user input. Once compiled, the SSA algorithm is performed across the device and results returned after all simulations have been completed. The entire simulation is performed on the GPU, saving communication time between the CPU and GPU, often the most time consuming task in GPU acceleration. A schematic workflow is outlined in Figure 3.2.

If the user has no GPU available, OpenCL enables the available parallelism of the multi-core CPU. In Algorithm 3.1, we demonstrate a use case of the CUDASimulator in PySB.

```
from pysb.simulators import CUDASSASimulator
from pysb.examples.michment import model

# Set the simulation length and the number of time points
t = np.linspace(0, 20, 11)

# assign the CUDASimulator simulator to the model and run
sim = CUDASimulator(model)
# Run 1000 simulations
traj = sim.run(tspan=t, number_sim=1000)
```

Listing 3.1: Usage of the GPUSimulator in PySB.

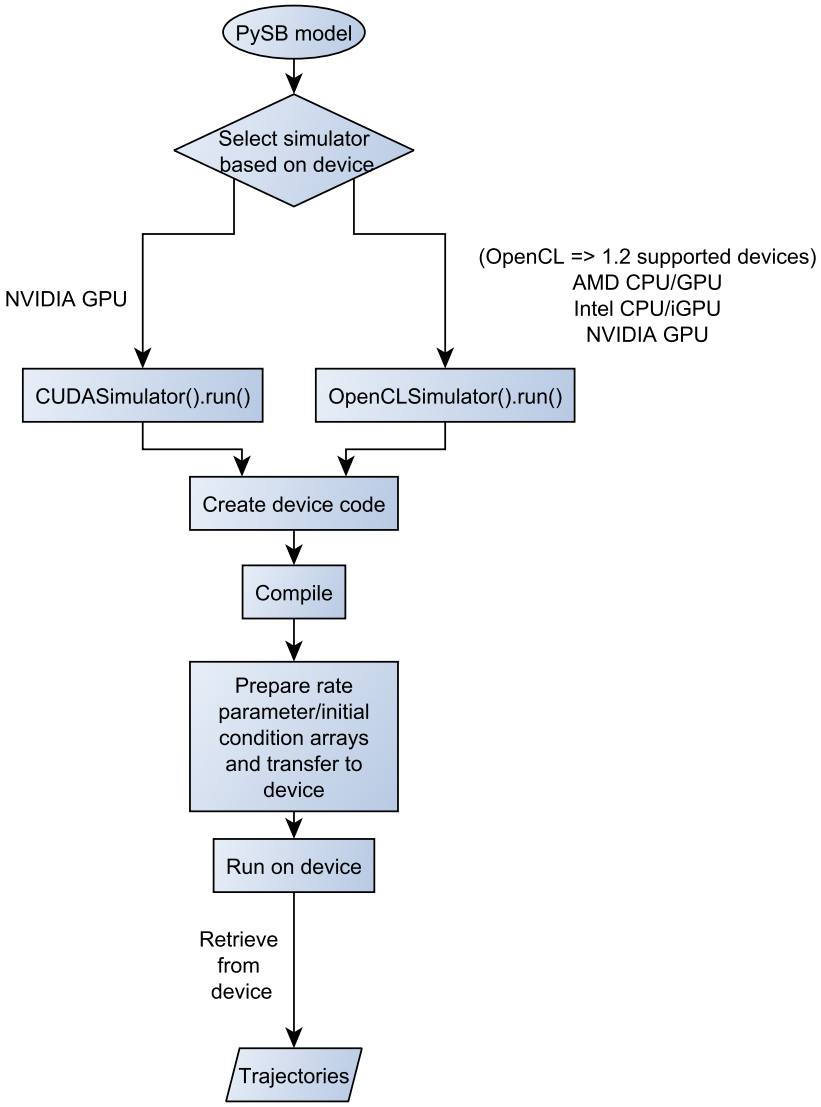


Figure 3.2: **Workflow for the GPU-SSA implementation.** A PySB model is imported and the GPU simulator is chosen. Both the PyCUDA version as well as the PyOpenCL version declare the working arrays for the SSA kernel, manage the memory allocation on the GPU, create the SSA kernel in C and pull the results from the SSA from the GPU back to the CPU. The kernel is created as a C-file that runs on every available hardware core. For each simulation, a thread is created that runs the full kernel, i.e. performs one realization of the stochastic simulation algorithm. After every thread is done with the simulation, the result vector is pulled back from the device and the device-memory gets cleaned up.

Since every simulation on the GPU is independent, each simulation starts with its own initial conditions and parameters, allowing users to simulate numerous initial conditions and/or parameter sets. The **.run** call also accepts *param_values* and/or *initials* input matrices as arguments, where every row holds the parameters/initial conditions for the particular simulation.

3.2.2.3 Models

In this subsection, we introduce example models implemented in PySB and demonstrate the performance of our implementation versus BioNetGen and StockKit, the two SSA methods currently available to PySB users. All models are written in PySB and are available as examples on the github repository of PySB. The computational size of the models is summarized in Table 3.2.

3.2.2.3.1 Michaelis-Menten Michaelis-Menten kinetics can be seen as one of the most prominent test cases for biochemical reactions. It represents the reversible binding of an enzyme E with a substrate S to release a product P in a non-reversible process. The process can be written as

$$E + S \underset{k_r}{\overset{k_f}{\rightleftharpoons}} ES \overset{k_c}{\rightarrow} E + P,$$

where k_f, k_r, k_c, denote the rate constants forward rate, reverse rate, and catalytic rate respectively.

3.2.2.3.2 Schlögl In (Schlögl, 1972), Friedrich Schlögl introduced chemical reaction models that describe the transition of a steady state into non-equilibrium steady states, a phenomenon at that time observed in thermodynamics. The chemical species in this example are representing rarefied gases and homogeneity is assumed due to constant stirring of the system.
One possible representation of such a non-equilibrium phase transition is

$$\begin{aligned} 2X &\underset{k_2}{\overset{k_1}{\rightleftharpoons}} 3X \\ I &\underset{k_4}{\overset{k_3}{\rightleftharpoons}} I + X \end{aligned}$$

	Number of reactions	Number of species	Number of parameters
Michaelis-Menton	3	4	4
Schlögl	4	3	6
Kinase Cascade	30	21	36
EARM1.0	70	58	88

Table 3.2: Size of the different models used in this section.

Note, that the concentration of I remains constant over the simulation time, while the concentration of X changes over time.

The Schlögl model is considered to be a standard test model to capture bi-stable states.

3.2.2.3.3 Kinase Cascade This model is adapted from (Chen et al., 2009)

3.2.2.3.4 EARM1.0 The extrinsic apoptosis reaction model (EARM) was introduced by (Albeck et al., 2008) which describes the model pathway of apoptosis induced by an extrinsic signal. The system captures the permeabilization of the outer mitochondrial membrane (MOMP) resulting in the inhibition of apoptosis inhibitors, as well as the activation of apoptosome caspases.

3.2.2.4 Results

3.2.2.4.1 Validation of implementation We validate the simulation output by comparing them with the well-established SSA implementation provided by BioNetGen (BNG) (Harris et al., 2016) and Stochkit (Sanft et al., 2011). Since SSA is a stochastic algorithm, comparing the results must be done statistically. The Kolmogorov-Smirnov test is a measure whether two samples are drawn from the same distribution. A significant p-value would reject the hypothesis, that the samples come from two different distributions. As shown on the left of Figure 3.3, our two implementations can not be distinguished from StochKit and BNG according to the Kolmogorov-Smirnov test.

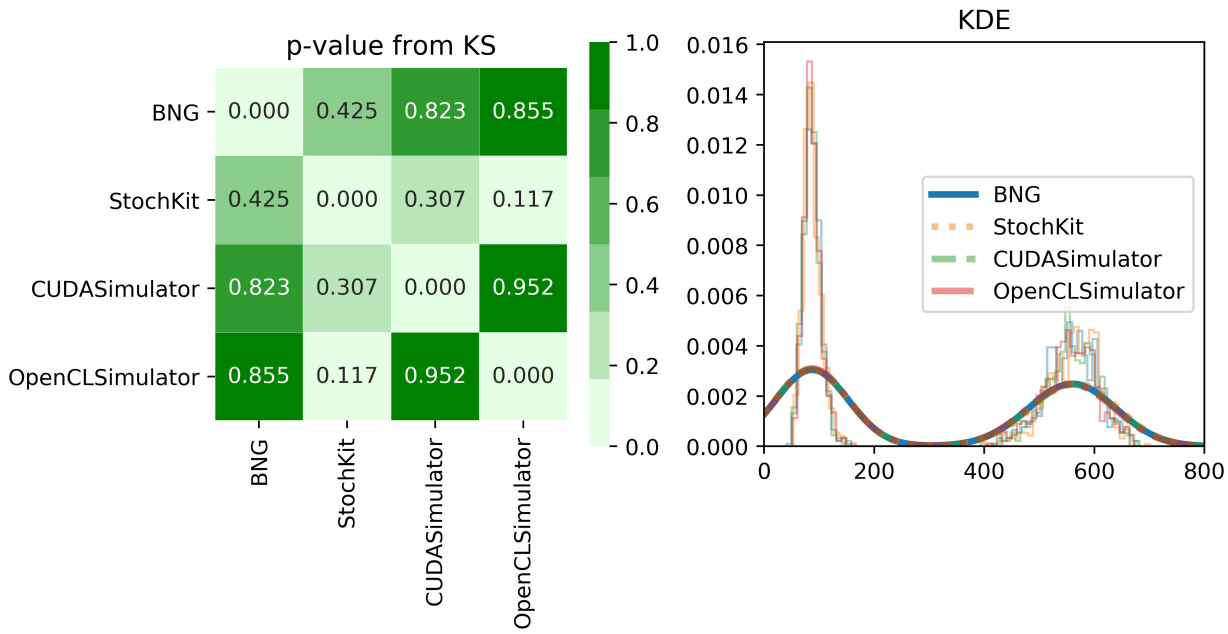


Figure 3.3: **Validation of the CUDASimulator and OpenCLSimulator.** The Schlögl model was simulated 1000 times and the end time point (shown on right) was used to compute Kolmogorov-Smirnov P-values for all pairs of simulators.

3.2.2.4.2 Benchmarks In this section, we compare the run-time of our method with BNG as well as the CPU-parallelised version of SSA provided by StochKit (Sanft et al., 2011). CUDA and Opencl simulations were performed on a NVIDIA Volta V100 (https://www.nvidia.com/en-us/data-center/v100/). CPU simulations were performed on the IBM POWER9. For StochKit simulations, we performed single CPU as well as running across 64 cores. Figure 3.4 shows the timing results for the different models introduced in section 3.2.2.3 and values can be found in Table 3.3.

As shown in 3.4, the GPU implementations are faster by orders of magnitude for all models in the single core implementations of BNG and StochKit. StochKit on 64 cores is fast when performing small number of simulations ($<$1024). This is because each CPU simulation is faster than on the GPU, requiring multiple simulations to be performed in order to gain from the GPUs parallelism. However, as the number of simulations on CPU doubles, do does the simulation time. On the contrary, there is a plateau in the GPU run-time until it is at full capacity (roughly 8192 simulations). Then, the GPU times share the same pattern (double the simulations = double the the simulation time). Thus, the GPU simulator is nearly always better than the CPU counter part, with the exception of running smaller number of simulations (1000) across many cores (64).

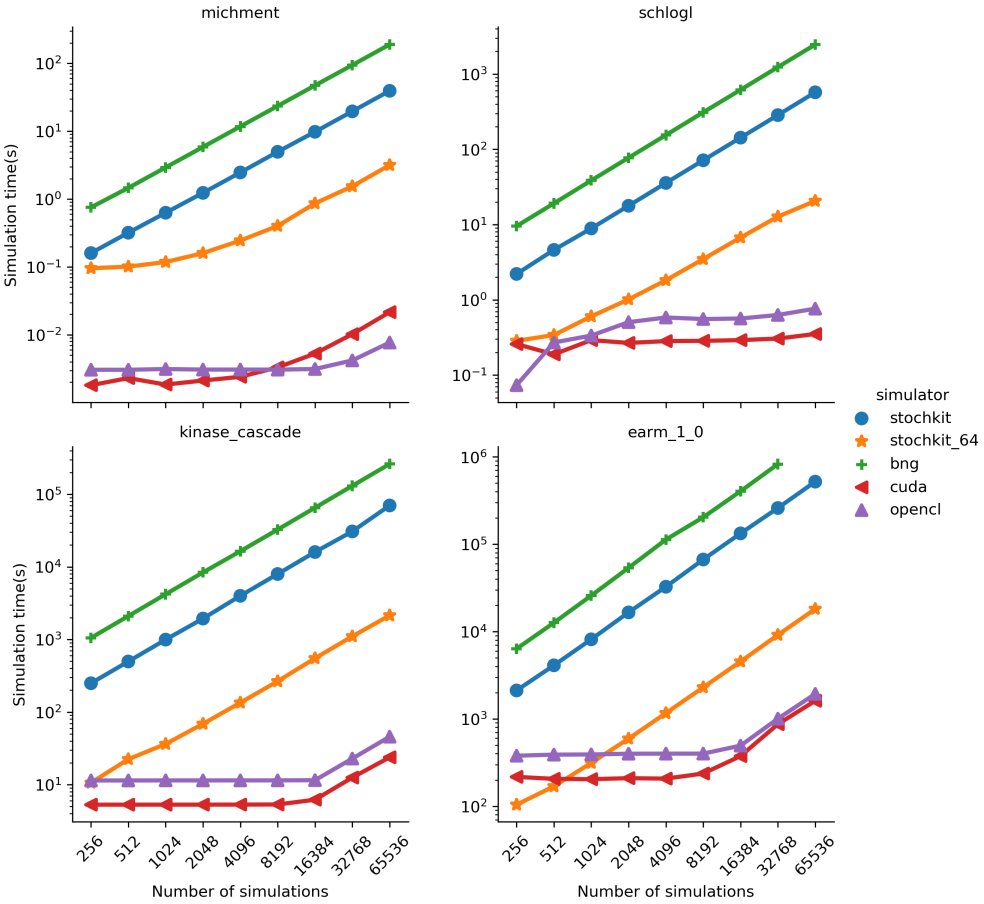


Figure 3.4: Comparing BioNetGen, StochKit and GPU-SSA timings run on the NVIDIA V100. On the $x-$axis, we see the number of simulations of each model. On the $y-$axis, we see the amount of time in seconds the simulation of the model takes for this number of simulations. Each simulation is a realization of a possible state of the system. BioNetGen depicted in green is implemented to run on only one CPU. Doubling the number of simulations from 2048 to 4096 would result in a run-time of more than two days. In blue and orange, we depict the run-time for StochKit on 1 CPU, as well as a parallelization on 64 CPU cores, respectively.

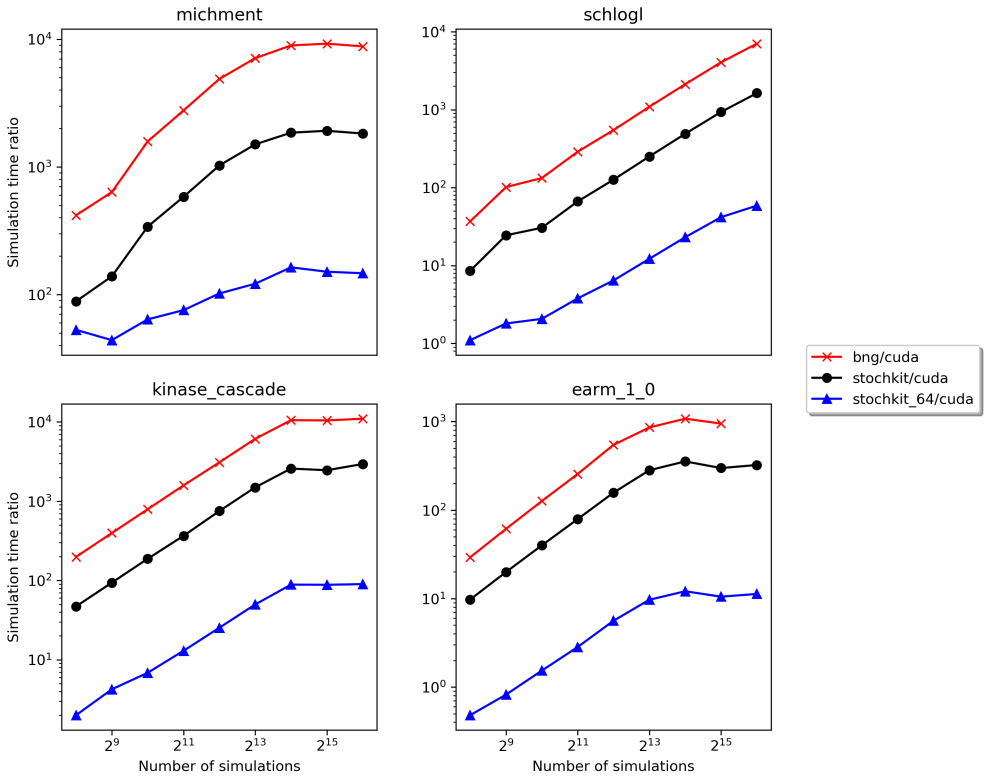


Figure 3.5: **Fold change improvements of the CUDASimulator compare to CPU implementation.** The ratio of BNG, StochKit, or Stockit on 64 cores simulation time divided by our CUDA simulator time.

3.2.2.4.3 Additional benchmarks Though users are able to access the NVIDIA Volta GPU (their current flagship compute GPU) via Amazon Web Services, we don't expect or require users to have the newest and fastest GPUs. Thus, we also evaluated older generations of the less expensive GeForce consumer graphics cards. As shown in Figure 3.6, we saw improvements in timings across each generation of GPU. All GPU simulations were faster for large number of simulations ($>$ 20k) than the sequential CPU implementation on each machine (data not shown). Thus users are not required to have the flagship Volta GPU to gain improvements from our simulators. We also saw improvements in simulation times across GPU generations, giving promise that our simulators will also improve with the next generations of GPUs to come.

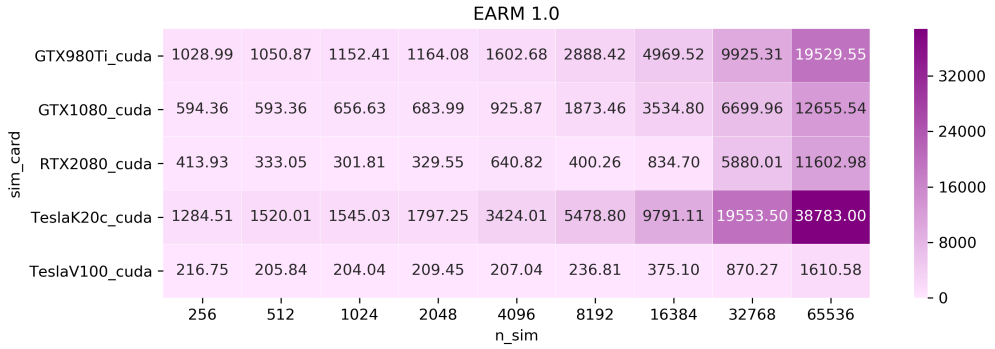


Figure 3.6: Timing analysis for multiple generations of GPUs.The GPUs represent the last three generation of consumer 'gaming' cards (GTX980TI, GTX1080, RTX2080) and two generations of data center cards (Tesla K20C and Tesla Volta V100).

3.2.2.4.4 OpenCL CUDA is only accessible via NVIDIA GPUs. OpenCL is able to run on NVIDIA or AMD GPUs and it possible to run on multi-core CPUs. Thus, we created an OpenCL implementation to make GPU-SSA more accessible to the user. As shown in Figure 3.7, we performed simulations across various GPUs using both OpenCL and CUDA simulators. Since CUDA is optimized for NVIDIA GPUs, it is not surprising that the CUDA implementation runs more efficient than OpenCL on NVIDIA devices3.7. Sadly, we were not able to test on any modern AMD GPUS and were only able to test on a decade old HD7970 AMD card (that was found for 30 dollars on Craigslist). AMD consumer cards are known to have higher double precision than NVIDIA consumer GPUs, which is required for stochastic simulations. Despite the

old AMD card, it still performs better than the CPU implementations (8.63 seconds vs 2467.47 seconds for 65536 simulations of the Schlögl model). Our benchmarks show that we do not lose much performance using OpenCL, and that even non-NVIDIA specific GPUs can give the user a significant benefit over a sequential CPU implementation.

Additionally, since OpenCL is capable of using CPUs, we ran the benchmark for various CPUs that were available. We used three different CPUs, Intel i5-6500T, Ryzen 1600x, and Ryzen 3900x, each having 4, 12, and 24 compute cores (Ryzen have 6 and 12 core, but hyper threading allows them to double capacity). We compare run times for the Schlögl across various CPUs versus the BNG and StochKit timings, shown in Figure 3.8. Our implementation outperform the BNG and single CPU StochKit implementation. The 64 core StochKit times are within the 12 core Ryzens timing and even beat by the 24 core Ryzen 3900x. Note that despite their improvement over the BNG and StochKit, our OpenCL CPU timings are still slower than the of the slowest GPU runs (12.74 seconds vs 8.63 seconds) and two orders of magnitude than the fastest times (12.74 seconds vs .76 seconds),

Figure 3.7: **OpenCL vs CUDA timings.** With the exception of the H7970 (only OpenCL capable), all GPUs were benchmarked with CUDA and OpenCL. In nearly all cases, the OpenCL timings are slightly more than the CUDA timings. This can be attributed to CUDA being specific for NVIDIA devices.

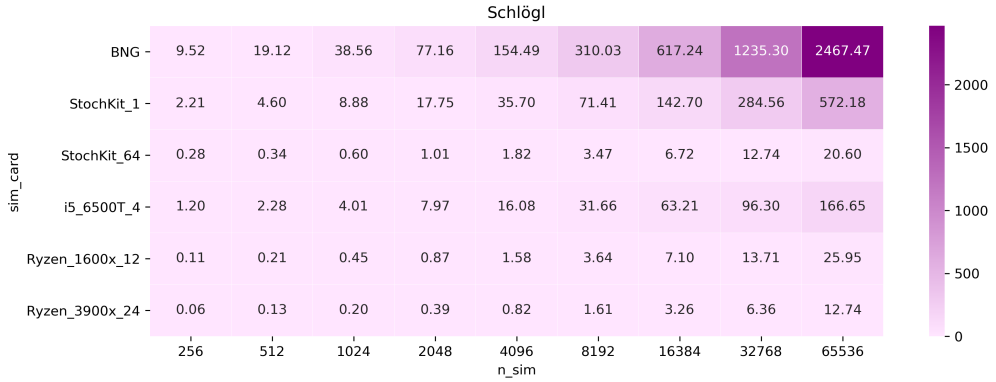


Figure 3.8: Timing analysis for CPU implementations. BNG, StochKit_1, and StochKit_64 timings are performed using their respective software. The timings for the remaining CPUs were performed using the OpenCLSimulator. Each rows name reflects the name of the CPU and the number of cores (name_Ncores).

3.2.2.5 Discussion

We introduced an implementation of the stochastic simulation algorithm (SSA) on GPUs, as a newly integrated simulator for the rule-based modeling framework PySB. This enables the user to gather statistics about the chemical master equation underlying the chemical kinetics system in a more feasible time frame, than currently accessible simulators accessible to the PySB community (BioNetGen and StochKit). Both GPU implementations enables orders of magnitude faster simulations than the CPU implementations, even beating StochKit running across 64 CPUs. We also saw an increase in speed up across each newer generation of GPUs, suggesting that the time required to run many SSA simulations will continue to decrease as GPUs improve in the future. Finally, by creating an OpenCL based simulators, users are able to leverage AMD GPUs and multi-core systems without requiring them to write specialized code.

Table 3.3: Simulation run times.

model	n_sim	Simulator times (seconds)					Timing ratio		
		BNG	CUDA	OpenCL	StockKit	StochKit_64	BNG \CUDA	StochKit \CUDA	StochKit64 \CUDA
earm_1_0	**256**	6348.13	216.74	378.81	2115.36	104.16	29.28	9.75	0.48
	512	12628.22	205.83	389.54	4104.66	168.92	61.35	19.94	0.82
	1024	25805.27	204.04	391.39	8150.94	313.07	126.47	39.94	1.53
	2048	53670.92	209.45	399.20	16577.10	593.39	256.24	79.14	2.83
	4096	113075.98	207.04	399.75	32606.80	1163.14	546.14	157.48	5.61
	8192	203555.37	236.80	400.28	66849.00	2296.05	859.58	282.29	9.69
	16384	404603.91	375.10	494.27	133109.00	4535.86	1078.65	354.86	12.09
	32768	827438.20	870.26	1005.73	259961.00	9144.92	950.78	298.71	10.50
	65536	NaN	1610.57	1933.24	520748.00	18158.80	NaN	323.32	11.27
kinase_cascade	**256**	1051.37	5.29	11.41	249.35	10.58	198.47	47.07	1.99
	512	2102.81	5.29	11.41	497.49	22.44	396.98	93.92	4.23
	1024	4199.72	5.29	11.43	995.78	36.28	792.66	187.94	6.84
	2048	8393.11	5.30	11.43	1939.52	68.77	1583.29	365.87	12.97
	4096	16418.15	5.31	11.43	3993.20	134.53	3092.10	752.05	25.33
	8192	32634.33	5.33	11.45	7976.93	265.45	6122.02	1496.42	49.79
	16384	65257.74	6.20	11.51	16007.70	550.34	10524.15	2581.57	88.75
	32768	130748.75	12.51	22.84	30782.20	1104.34	10450.20	2460.29	88.26
	65536	261432.90	23.79	45.76	70123.80	2142.31	10986.69	2946.94	90.03
michment	**256**	0.755	0.002	0.003	0.160	0.096	417.16	88.42	52.88
	512	1.462	0.002	0.003	0.319	0.101	634.79	138.69	43.90
	1024	2.927	0.002	0.003	0.629	0.118	1581.74	340.03	63.85
	2048	5.858	0.002	0.003	1.234	0.160	2763.85	582.21	75.40
	4096	11.741	0.002	0.003	2.467	0.245	4881.25	1025.42	101.88
	8192	23.522	0.003	0.003	4.968	0.402	7102.32	1500.02	121.40
	16384	47.253	0.005	0.003	9.797	0.864	8932.86	1851.96	163.28
	32768	94.376	0.010	0.004	19.612	1.543	9226.00	1917.22	150.86
	65536	189.680	0.022	0.008	39.380	3.167	8786.72	1824.22	146.71
Schlögl	**256**	9.519	0.258	0.073	2.206	0.285	36.87	8.54	1.10
	512	19.118	0.188	0.271	4.602	0.340	101.47	24.42	1.80
	1024	38.555	0.291	0.335	8.881	0.602	132.52	30.52	2.06
	2048	77.159	0.267	0.504	17.755	1.012	288.67	66.42	3.78
	4096	154.493	0.282	0.581	35.702	1.819	547.08	126.42	6.44
	8192	310.031	0.285	0.555	71.408	3.472	1089.10	250.85	12.19
	16384	617.244	0.291	0.564	142.701	6.716	2119.73	490.06	23.06
	32768	1235.297	0.305	0.629	284.565	12.736	4049.66	932.88	41.75
	65536	2467.469	0.351	0.762	572.178	20.598	7025.57	1629.15	58.64

3.3 Fitting models to data

Ideally, model parameters would reflect those that occur naturally within the cell. Though binding constants can be measured in isolated systems *in vitro*, experimentally measuring rate constants within live cells is difficult. Even parameters that are measured experimentally, they may not reflect the exact system conditions. This paired with the total number of parameters of large models (binding rates, synthesis and degradation rates, diffusion rates) makes measuring all parameters for a model is currently beyond experimental limitations. Thus, models are often fit to experimental data using parameter optimization (Eydgahi et al., 2013; Lopez et al., 2013). This calibration process involves simulating the model and comparing it to experimental protein concentration time course data. A common metric used to measure the distance between experiment and simulation is χ^2, described by the function χ^2

$$\chi^2 = \sum_{t=0}^{t_{end}} \sum_{i=1}^{N} \frac{(X^i_{simulated}(t,\Theta) - X^i_{measured}(t))^2}{\sigma^2_{measured}(t)} \tag{3.1}$$

where X is the species of interest, t is time-points of experimental measurement, i to N is species, and Θ the parameter vector. Thus, the goal is to find parameter sets that minimize the difference between simulations and experiment. Typically this has been performed by hand (Albeck et al., 2008) or by problem specific programs (Lopez et al., 2013). Due to the non-linearity of the system, local optimization methods require expensive compute time to find optimal solutions (Kim et al., 2010). Markov Chain Monte Carlo methods can be used but still require extensive computer time (Vrugt and Ter Braak, 2011; Vrugt, 2014; Eydgahi et al., 2013). Though general optimization packages exist (Fortin et al., 2012), biologists may not have the experience required to implement complex algorithms. Thus, the systems biology field would benefit from a model optimization package that enables users to easily and quickly train models to data. [2]

Here, we present simplePSO, a simple and efficient implementation of the Particle Swarm

[2]This was one of the earlier projects in this thesis work. Since then, a package call PyNetFit has been published that offers a lot more than this implementation. Though the design of this solution was to be as simple to use as possible, a further exploration of PyNetFits offerings should be considered. Future work should entail comparisons.

Optimization algorithm for PySB models. It is designed for PySB models, thus alleviating the requirements of designing custom use optimize scripts. It can be used to train both deterministic and stochastic models, allowing the use of GPU simulators mentioned in 3.2.13.2.2. We demonstrate its performance in finding better fit parameters than a commonly used simulated annealing method used in Lopez et al. (2013).

3.3.1 Introduction to particle swarm optimization

Particle swarm optimization (PSO) was developed to mimic the behavior or a flock of birds finding food (Kennedy and Eberhart, 1995). Particles, representing birds, spread out over a defined search space. Each particle then assesses its fitness, cost function score (or the availability of food), and reports to the rest of the population. Then each particle adjusts its course based on the population's best position. However, each particle also remembers where it had its best cost function and takes a step towards its best position. This is repeated until all the particles are within a threshold of distance from one another or a predefined number of iterations.

It requires basic addition, subtraction and multiplication operators, and is computational inexpensive in both memory and computation standards. Compared to other methods of optimization, PSO has strengths in convergence, easy implementation and a global search (Coello Coello and Reyes-Sierra, 2006), making it an ideal candidate for biological models. The steps of the algorithm are shown in Figure 3. Each solution of the optimization is called a *particle* (*P*). Each particle has a position (*P.pos*), velocity (*P.vel*), and a personal best (*P.pb*a vector of parameters where the cost function had its best value). A group of *particles* is contained within a *swarm*. The algorithm begins by randomly choose starting positions for each particle. Then, for each iteration, each particles position is scored using the user defined cost function, the *currentbest* (*CB*) particle (particle with the lowest cost function) is distinguished from the others, and each particle compares its current cost versus its own personal best. Each particles velocity is calculated by creating a vector towards the global best particles as well as towards its best location. The particles position is updated by adding the particles velocity to its current position.

Algorithm 3 Particle swarm optimization

Generate starting positions and velocities for each particle.
while stopping criteria not met **do**
    Evaluate each particles fitness.
    Identify global best particle *GB*
    for P in *swarm* **do**
        if $P.fit < P.pb$ **then**
            $P.pb = P.pos$
    for P in *swarm* **do**
                ▷ Update particle velocity
        $P.vel = w * P.vel(t) + c_1 r_1 (P.pb - P.pos) + c_2 r_2 (GB - P.pos))$
        $P.pos = P.pos + P.vel$ ▷ Update particle position

▷ where w is the acceleration weight, c_1 and c_2 are learning constants, and r_1 and r_2 are random numbers from [0,1].

This process is repeated until a maximum number of iterations has been performed or the particles converge to a solution.

3.3.2 simplePSO: Fast modeling calibration for Rule based models.

simplePSO is a Python package designed to train PySB models to experimental data. It leverage multi-core systems to evaluate each particles cost function in parallel, speeding up each iteration of the algorithm. It can be use to train deterministic or stochastic models. It is implemented in Python and available for install via PyPi (*pip install simplepso*).

To demonstrate simplePSO, we trained the EARM2.0 model using the same cost function and experimental data used in Lopez et al. (2013). Example fits are shown in Figure 3.9. We wanted to compare the efficiency of simplePSO to the simulated annealing protocol in Lopez et al. (2013). We used the exact objective function, where the cleavage of PARP, tBID trained EARM2.0 with exact conditions as done in . We compare our ability to find equally fitting parameter sets as the well as the number of iterations to find them. Since each iteration of simplePSO uses multiple cores, we allowed the four times (the number of cores used) as many iterations of the simulated annealing steps to provide a fair comparison. As shown in Figure 3.11, simplePSO is able to find better fits (lower score) and can reach those values faster.

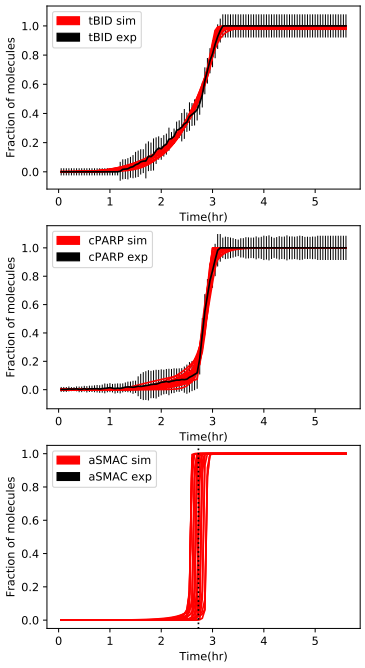


Figure 3.9: **Best fit parameters from simplePSO.** Example fits obtained from 100 runs of simplePSO of the EARM2.0 model.

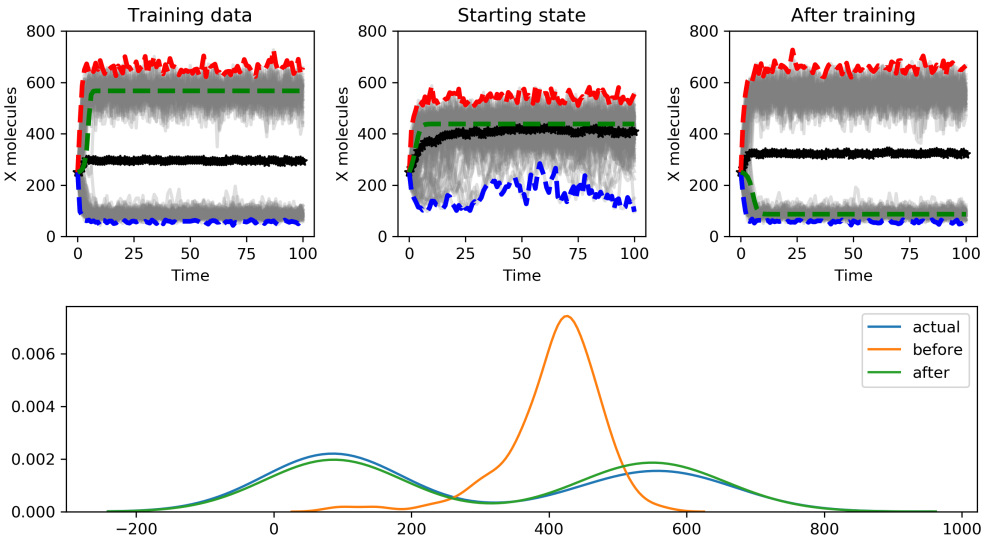


Figure 3.10: **Example output using simplePSO on with stochastic simulations.** Top right) Trajectories of Schlögl with ideal parameters. Top middle) Trajectories of random parameters used as the starting position, Top right) Trajectories of the best fit parameters obtained from simplePSO. Bottom) Distribution of the total number of X molecules.

Additionally, as shown in Figure 3.10, simplePSO can be used to train to train models using stochastic simulations. We demonstrate this using the Schlögl model, with an attempt to recreate distribution of X molecules at the end of the simulation. For an objective function, we calculated the distance between the distributions of each particle with ideal parameters using one minus the p-value obtained by the Kolmogorov-Smirnov (Massey Jr, 1951).

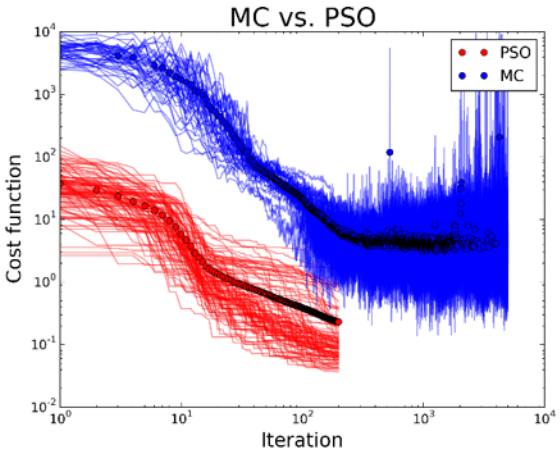


Figure 3.11: **Performance of PSO vs simulated annealing**. The objective function is shown versus iteration number for each individual runs of the simulated annealing(blue) and the best fit particle from PSO(red).

3.4 Quantifing the effects of initial protein concentrations on the model's prediction of cell fate.

[3]

After fitting the model to experimental data, the model can be characterize to predict sensitivity to changes in initial concentrations. This analysis can be used to answer questions such as *How much does each species contribute to variability in time of death?*. Similar to knockdown or over expression experiments, we expect the models output to change based on the initial concentrations. This can be done by perturbing a species (i.e. decrease by initial concentration by 10 %), run a simulation, and compare the result to a control (simulation without a perturbation).

To systematically characterize the models, we created a sensitivity analyses with respect to initial species populations. The approach is similar to that found in (Gaudet et al., 2012). Let $\Theta = [\theta_0 \ldots \theta_{N-1}]^{\mathrm{T}} \in \mathbb{R}^N_{\geq 0}$ be a vector of N initial populations of chemical species, such that $\theta_k \neq 0$ for at least one value of $k \in \{0, \ldots, N-1\}$. Let also $F : \Theta \to \mathbb{R}$ be a model output given Θ (e.g., the population of a species at a given time point, the time required for a species population to reach a predefined threshold). We denote by $\Theta^0 = [\theta^0_0 \ldots \theta^0_{N-1}]^{\mathrm{T}}$ a vector of *reference* initial population values obtained. The model *sensitivity* to changes in the initial species populations is then defined as

$$\begin{aligned} S: \ & \Theta \to \mathbb{R} \\ & \Theta \mapsto \frac{F(\Theta) - F(\Theta^0)}{F(\Theta^0)} \cdot 100. \end{aligned} \tag{3.2}$$

Let $\mathbf{J} = [0 \ldots N-1]^{\mathrm{T}} \in \mathbb{N}^N$ be a vector of N species indices and $\Delta = [\delta_0 \ldots \delta_{M-1}]^{\mathrm{T}} \in \mathbb{R}^M_{\geq 0}$, $M \geq 1$, be a vector of M multiplicative factors representing perturbations to each initial species population. We generate an NM-length vector of 2-tuples $\mathbf{A} \equiv \mathrm{vec}(\Delta \times \mathbf{J})$, where $\times$ denotes the Cartesian product and *vec* is the matrix vectorization. We then generate the $NM \times NM$ matrix $\mathbf{B} \equiv \mathbf{A} \times \mathbf{A}$ by taking the Cartesian product of $\mathbf{A}$ with itself. Element B_{ij} of this matrix is a 2-tuple of 2-tuples, where the first element of each inner 2-tuple is a perturbation (an element of Δ) and the

[3] Parts of this section are modified from (Harris et al., 2017)

second element is a species index (an element of $\mathbf{J}$). The perturbations and species indices relate to the row and column indices of $\mathbf{B}$ via

$$B_{ij} = \left((\delta_{i\%M}, J_{\lfloor i/M \rfloor}), (\delta_{j\%M}, J_{\lfloor j/M \rfloor})\right), \;\; i,j \in \{0,\ldots,NM-1\}, \tag{3.3}$$

where % is the modulo operator and $\lfloor \cdot \rfloor$ is the floor operator. For convenience, we also define the vector $\mathbf{A}' \equiv \mathrm{vec}(\mathbf{1}_M \times \mathbf{J})$, where $\mathbf{1}_M = [1\ldots1]^{\mathrm{T}}$ is an all-ones vector of length M. This allows us to define the matrix $\mathbf{B}' \equiv \mathbf{A}' \times \mathbf{A}$, the ij-th element of which is

$$B'_{ij} = \left((1, J_{\lfloor i/M \rfloor}), (\delta_{j\%M}, J_{\lfloor j/M \rfloor})\right), \;\; i,j \in \{0,\ldots,NM-1\}, \tag{3.4}$$

i.e., the initial population of species $J_{\lfloor i/M \rfloor}$ is unperturbed.

We define the function

$$\begin{aligned}
&G:\ B_{ij} \to \mathbb{R}^N_{\geq 0} \\
&\begin{cases}
\text{if } J_{\lfloor i/M \rfloor} \neq J_{\lfloor j/M \rfloor}: & \\
\quad \theta^0_{J_{\lfloor i/M \rfloor}} & \mapsto \quad \delta_{i\%M} \cdot \theta^0_{J_{\lfloor i/M \rfloor}} \\
\quad \theta^0_{J_{\lfloor j/M \rfloor}} & \mapsto \quad \delta_{j\%M} \cdot \theta^0_{J_{\lfloor j/M \rfloor}} \\
\quad \theta^0_k & \mapsto \quad \theta^0_k, \quad k \in \mathbf{J} \setminus \{J_{\lfloor i/M \rfloor}, J_{\lfloor j/M \rfloor}\} \\
\text{else}: & \\
\quad \theta^0_k & \mapsto \quad \theta^0_k, \quad k \in \mathbf{J}
\end{cases},
\end{aligned} \tag{3.5}$$

which takes an element B_{ij} and returns a vector of initial species populations that differs from the reference vector Θ^0 at indices $J_{\lfloor i/M \rfloor}$ and $J_{\lfloor j/M \rfloor}$ if and only if $J_{\lfloor i/M \rfloor} \neq J_{\lfloor j/M \rfloor}$. We then define the

function

$$H:\ \mathbf{B} \to \mathbb{R}_{\geq 0}^{NM \times NM}$$
$$B_{ij} \mapsto S(G(B_{ij})),\ \forall\ i,j \in \{0,\ldots,NM-1\}, \tag{3.6}$$

which sequentially applies Eqs. 3.5 and 3.2 to each element of $\mathbf{B}$ to produce an $NM \times NM$ matrix of sensitivity values. Let $\mathbf{P} \equiv H(\mathbf{B})$ be the *pairwise sensitivity matrix*. Note that $\mathbf{P}$ is symmetric and that the elements of the block diagonal are all zero. Therefore, $N \cdot (N-1) \cdot M^2/2 < (NM)^2$ evaluations of F (i.e., simulations) are needed to construct it. Further, let $\mathbf{P}' \equiv H(\mathbf{B}) - H(\mathbf{B}')$ be the *normalized pairwise sensitivity matrix*, i.e., each sensitivity value is normalized by subtracting the corresponding sensitivity value in which the initial population of species $J_{\lfloor i/M \rfloor}$ is unperturbed (see Eq. 3.4). In principle, to construct $\mathbf{P}'$ an additional $N \cdot (N-1) \cdot M$ evaluations of F are required.However, these can be avoided if 1 is contained within Δ.

Finally, we define the *single-parameter sensitivity multiset*[4]

$$\mathbf{Q}_k = \bigcup_{\substack{kM \leq i < (k+1)M \\ kM > j \geq (k+1)M}} P'_{ij},\ \ k \in \mathbf{J}, \tag{3.7}$$

which can be plotted as a boxplot to visualize the range of model outputs due to changes in the initial population of species k.

Example: *Two species, three perturbations* ($N = 2, M = 3$)

$$\Theta^0 = \begin{bmatrix} 100 & 100 \end{bmatrix}^{\mathrm{T}}$$
$$\mathbf{J} = \begin{bmatrix} 0 & 1 \end{bmatrix}^{\mathrm{T}}$$
$$\Delta = \begin{bmatrix} 0.8 & 1.0 & 1.2 \end{bmatrix}^{\mathrm{T}}$$

[4] A multiset is a generalization of a set that allows for duplicate elements (Knuth, 1997). It is defined as a 2-tuple (A, m), where A is a set and $m\colon A \to \mathbb{N}_{\geq 1}$ is the *multiplicity*.

$$\mathbf{A} = \text{vec}\left(\begin{bmatrix} (0.8,0) & (0.8,1) \\ (1.0,0) & (1.0,1) \\ (1.2,0) & (1.2,1) \end{bmatrix}\right)$$

$$= \begin{bmatrix} (0.8,0) & (1.0,0) & (1.2,0) & (0.8,1) & (1.0,1) & (1.2,1) \end{bmatrix}^{\mathrm{T}}$$

$$\mathbf{B} = \left[\begin{array}{ccc|ccc} ((0.8,0),(0.8,0)) & ((0.8,0),(1.0,0)) & ((0.8,0),(1.2,0)) & ((0.8,0),(0.8,1)) & ((0.8,0),(1.0,1)) & ((0.8,0),(1.2,1)) \\ ((1.0,0),(0.8,0)) & ((1.0,0),(1.0,0)) & ((1.0,0),(1.2,0)) & ((1.0,0),(0.8,1)) & ((1.0,0),(1.0,1)) & ((1.0,0),(1.2,1)) \\ ((1.2,0),(0.8,0)) & ((1.2,0),(1.0,0)) & ((1.2,0),(1.2,0)) & ((1.2,0),(0.8,1)) & ((1.2,0),(1.0,1)) & ((1.2,0),(1.2,1)) \\ \hdashline ((0.8,1),(0.8,0)) & ((0.8,1),(1.0,0)) & ((0.8,1),(1.2,0)) & ((0.8,1),(0.8,1)) & ((0.8,1),(1.0,1)) & ((0.8,1),(1.2,1)) \\ ((1.0,1),(0.8,0)) & ((1.0,1),(1.0,0)) & ((1.0,1),(1.2,0)) & ((1.0,1),(0.8,1)) & ((1.0,1),(1.0,1)) & ((1.0,1),(1.2,1)) \\ ((1.2,1),(0.8,0)) & ((1.2,1),(1.0,0)) & ((1.2,1),(1.2,0)) & ((1.2,1),(0.8,1)) & ((1.2,1),(1.0,1)) & ((1.2,1),(1.2,1)) \end{array}\right]$$

$$\mathbf{B}' = \left[\begin{array}{ccc|ccc} ((1.0,0),(0.8,0)) & ((1.0,0),(1.0,0)) & ((1.0,0),(1.2,0)) & ((1.0,0),(0.8,1)) & ((1.0,0),(1.0,1)) & ((1.0,0),(1.2,1)) \\ ((1.0,0),(0.8,0)) & ((1.0,0),(1.0,0)) & ((1.0,0),(1.2,0)) & ((1.0,0),(0.8,1)) & ((1.0,0),(1.0,1)) & ((1.0,0),(1.2,1)) \\ ((1.0,0),(0.8,0)) & ((1.0,0),(1.0,0)) & ((1.0,0),(1.2,0)) & ((1.0,0),(0.8,1)) & ((1.0,0),(1.0,1)) & ((1.0,0),(1.2,1)) \\ \hdashline ((1.0,1),(0.8,0)) & ((1.0,1),(1.0,0)) & ((1.0,1),(1.2,0)) & ((1.0,1),(0.8,1)) & ((1.0,1),(1.0,1)) & ((1.0,1),(1.2,1)) \\ ((1.0,1),(0.8,0)) & ((1.0,1),(1.0,0)) & ((1.0,1),(1.2,0)) & ((1.0,1),(0.8,1)) & ((1.0,1),(1.0,1)) & ((1.0,1),(1.2,1)) \\ ((1.0,1),(0.8,0)) & ((1.0,1),(1.0,0)) & ((1.0,1),(1.2,0)) & ((1.0,1),(0.8,1)) & ((1.0,1),(1.0,1)) & ((1.0,1),(1.2,1)) \end{array}\right]$$

$$G(B_{00}) = G(\,((0.8,0),(0.8,0))\,) = \begin{bmatrix} 100 & 100 \end{bmatrix}^{\mathrm{T}} = \Theta^0$$

$$G(B_{05}) = G(\,((0.8,0),(1.2,1))\,) = \begin{bmatrix} 80 & 120 \end{bmatrix}^{\mathrm{T}}$$

$$G(B_{50}) = G(\,((1.2,1),(0.8,0))\,) = \begin{bmatrix} 80 & 120 \end{bmatrix}^{\mathrm{T}} = G(B_{05})$$

$$G(B_{55}) = G(\,((1.2,1),(1.2,1))\,) = \begin{bmatrix} 100 & 100 \end{bmatrix}^{\mathrm{T}} = \Theta^0$$

$$S(G(B_{00})) = S(\Theta^0) = \frac{F(\Theta^0) - F(\Theta^0)}{F(\Theta^0)} \cdot 100 = 0$$

$$S(G(B_{05})) = S([80 \;\; 120]^{\mathrm{T}}) = \frac{F\left([80 \;\; 120]^{\mathrm{T}}\right) - F(\Theta^0)}{F(\Theta^0)} \cdot 100$$

$$S(G(B_{50})) = S(G(B_{05}))$$

$$S(G(B_{55})) = S(\Theta^0) = 0$$

$$\mathbf{P} \equiv H(\mathbf{B})$$

$$= \left[\begin{array}{ccc|ccc}
0 & 0 & 0 & S([80\ \ 80\ ]^{\mathrm{T}}) & S([80\ \ 100]^{\mathrm{T}}) & S([80\ \ 120]^{\mathrm{T}}) \\
0 & 0 & 0 & S([100\ 80\ ]^{\mathrm{T}}) & 0 & S([100\ 120]^{\mathrm{T}}) \\
0 & 0 & 0 & S([120\ 80\ ]^{\mathrm{T}}) & S([120\ 100]^{\mathrm{T}}) & S([120\ 120]^{\mathrm{T}}) \\
\hline
S([80\ \ 80\ ]^{\mathrm{T}}) & S([100\ 80\ ]^{\mathrm{T}}) & S([120\ 80\ ]^{\mathrm{T}}) & 0 & 0 & 0 \\
S([80\ \ 100]^{\mathrm{T}}) & 0 & S([120\ 100]^{\mathrm{T}}) & 0 & 0 & 0 \\
S([80\ \ 120]^{\mathrm{T}}) & S([100\ 120]^{\mathrm{T}}) & S([120\ 120]^{\mathrm{T}}) & 0 & 0 & 0
\end{array}\right]$$

$$\mathbf{P}' \equiv H(\mathbf{B}) - H(\mathbf{B}')$$

$$= \left[\begin{array}{ccc|ccc}
0 & 0 & 0 & S([80\ \ 80\ ]^{\mathrm{T}}) - S([100\ 80\ ]^{\mathrm{T}}) & S([80\ \ 100]^{\mathrm{T}}) & S([80\ \ 120]^{\mathrm{T}}) - S([100\ 120]^{\mathrm{T}}) \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & S([120\ 80\ ]^{\mathrm{T}}) - S([100\ 80\ ]^{\mathrm{T}}) & S([120\ 100]^{\mathrm{T}}) & S([120\ 120]^{\mathrm{T}}) - S([100\ 120]^{\mathrm{T}}) \\
\hline
S([80\ \ 80\ ]^{\mathrm{T}}) - S([80\ \ 100]^{\mathrm{T}}) & S([100\ 80\ ]^{\mathrm{T}}) & S([120\ 80\ ]^{\mathrm{T}}) - S([120\ 100]^{\mathrm{T}}) & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
S([80\ \ 120]^{\mathrm{T}}) - S([80\ \ 100]^{\mathrm{T}}) & S([100\ 120]^{\mathrm{T}}) & S([120\ 120]^{\mathrm{T}}) - S([120\ 100]^{\mathrm{T}}) & 0 & 0 & 0
\end{array}\right]$$

$$\mathbf{Q}_0 = \Big\{0,\ 0,\ 0,\ S([80\ \ 100]^{\mathrm{T}}),\ S([120\ 100]^{\mathrm{T}}),$$
$$S([80\ \ 80\ ]^{\mathrm{T}}) - S([100\ 80\ ]^{\mathrm{T}}),\ \ S([80\ \ 120]^{\mathrm{T}}) - S([100\ 120]^{\mathrm{T}}),$$
$$S([120\ 80\ ]^{\mathrm{T}}) - S([100\ 80\ ]^{\mathrm{T}}),\ \ S([120\ 120]^{\mathrm{T}}) - S([100\ 120]^{\mathrm{T}})\Big\}$$

$$\mathbf{Q}_1 = \Big\{0,\ 0,\ 0,\ S([100\ 80\ ]^{\mathrm{T}}),\ S([100\ 120]^{\mathrm{T}}),$$
$$S([80\ \ 80\ ]^{\mathrm{T}}) - S([80\ \ 100]^{\mathrm{T}}),\ \ S([120\ 80\ ]^{\mathrm{T}}) - S([120\ 100]^{\mathrm{T}}),$$
$$S([80\ \ 120]^{\mathrm{T}}) - S([80\ \ 100]^{\mathrm{T}}),\ \ S([120\ 120]^{\mathrm{T}}) - S([120\ 100]^{\mathrm{T}})\Big\}$$

3.4.1 Results and Discussion

In (Harris et al., 2017), we performed sensitively analysis on EARM 2.0 across two best fit parameter sets obtained from simplePSO. Despite each parameter set fitting the model equally well, the sensitivities of changing the initial concentration for each parameter set varied across all parameter To further explore if that was a consequence of the two parameter sets we found, or a property of the model, we performed sensitivity analysis on the top fit parameter sets obtained from running simplePSO. The results of each individual are shown in Figure 3.13 and summarized in Figure 3.12. Initial concentrations of BAD, SMAC, NOXA, C9 (caspase 9), and APAF make minimal impact on model outcome across all parameter sets, while the initial conditions of C8, C3, C6 (caspases), R (receptor), L (trail), and BID impact all model outcomes.

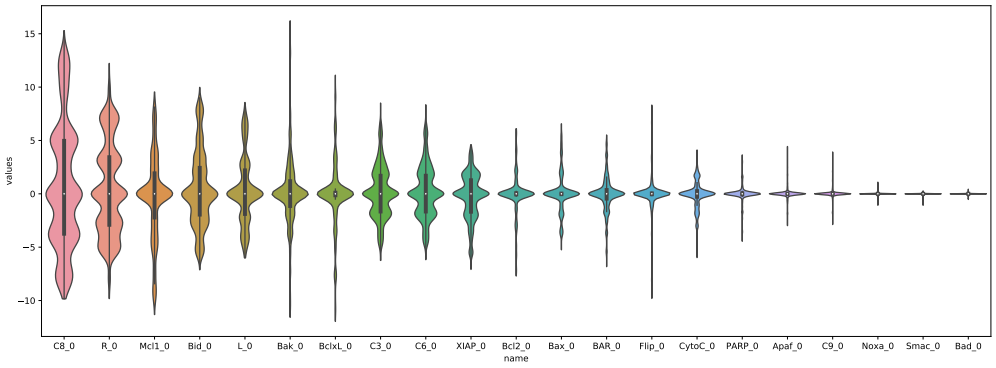


Figure 3.12: **Summary view of EARM 2.0 sensitivity** Violin plot of the percent change of time-to-death per initial concentration across 100 of the best fit parameter sets for EARM 2.0.

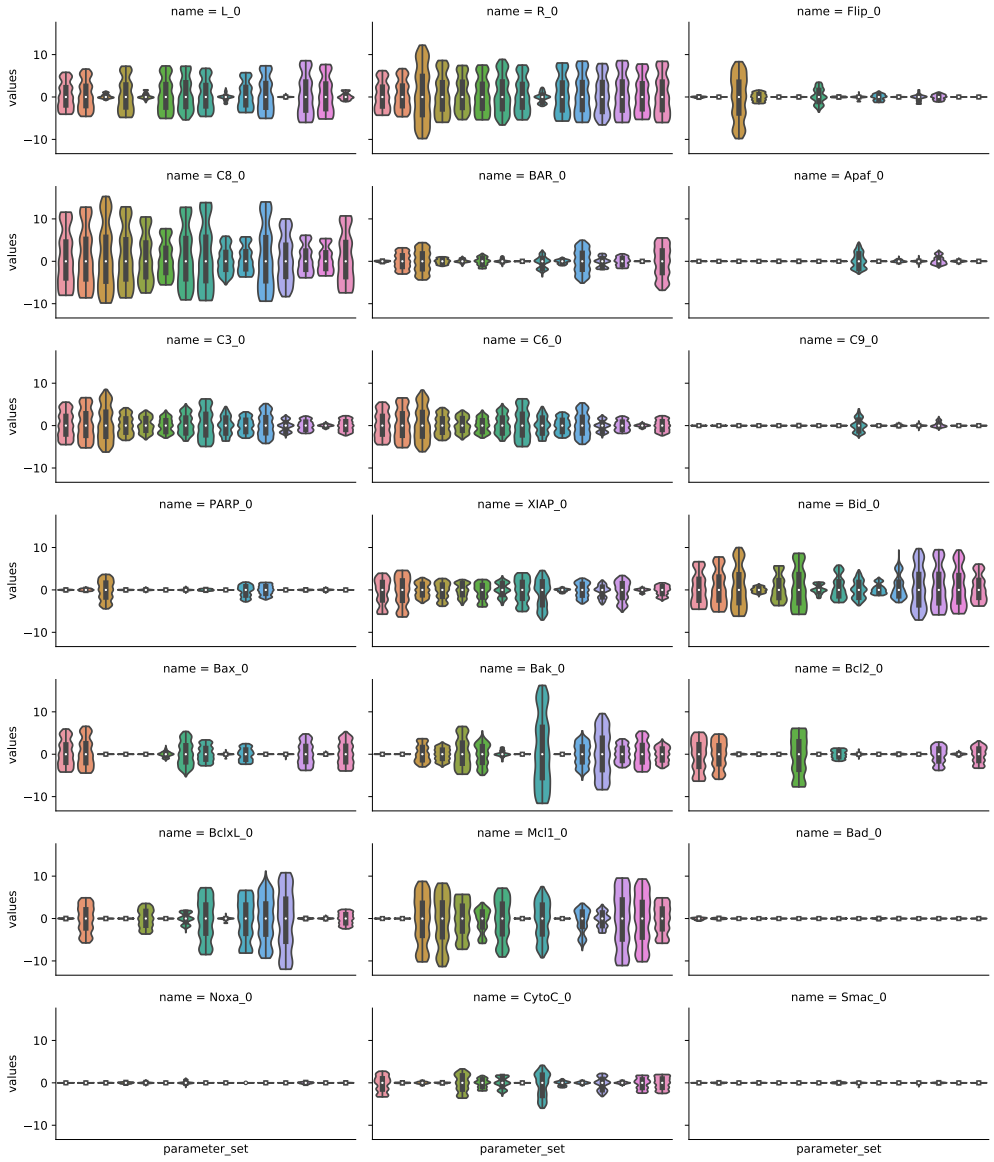


Figure 3.13: **Individual initial concentration sensitivity for EARM 2.0.** Distribution of percent change of time-to-death vs parameter set. The data are grouped by initial condition and visualized in each subplot.

Next, we examine the impact of the BCL2 family of proteins. First, we checked if the model was more sensitive to BAX or BAK, the effector apoptotic signals responsible for MOMP. As shown in Figure 3.15ba, our results suggest that MOMP signal dependence on BAX or BAK is parameter set dependent. Some parameter sets show no dependence on BAX while others show no dependence of BAK. Only two show that changes either would impact the model. Additionally, since these proteins are regulated by either MCL1, BCL2, or BCL-XL, we saw similar trends as

shown in Figure 3.15bb. This could be a consequence of evolution and help explain where there are so many proteins with near identical functions. Does having proteins with redundant properties ensure that the cell fully commit to signals of apoptosis? Or more simply, does it mean that despite equally fitting the model, some of these parameter sets are wrong? Since these species are grouped together in EARM 1.0, we tested to see if parameter sets for the grouped terms (BAX(BAX and BAK) and BCL2(MCL1, BCL2, BCL-XL)) are also parameter set dependent. For this, we performed sensitivity analysis on EARM 1.0 to compare as shown in 3.16. We then averaged across each initial condition, shown in 3.14.

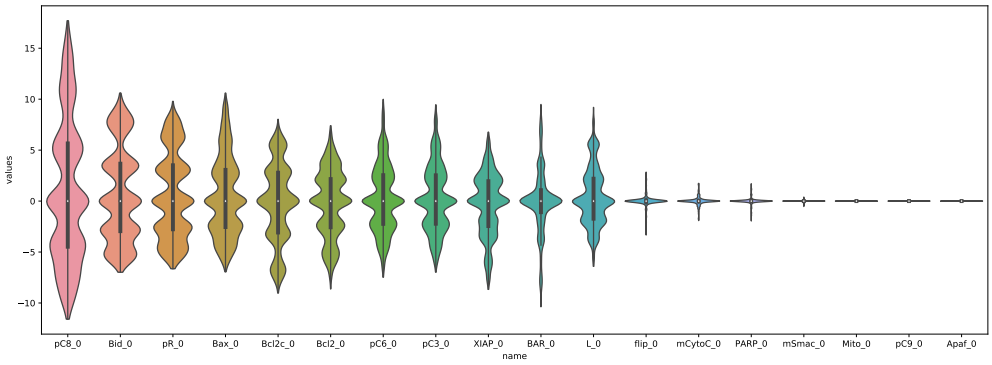
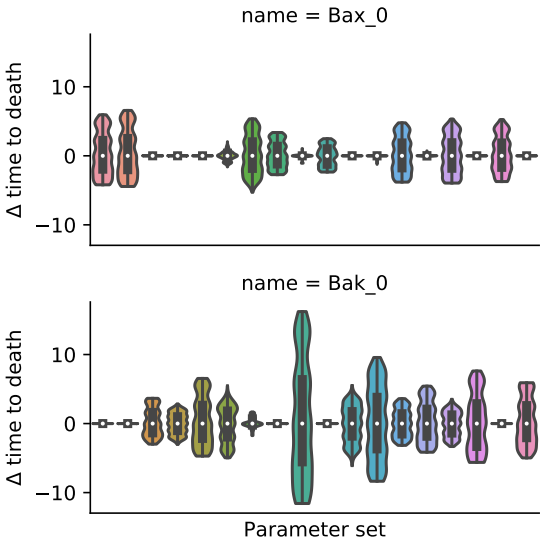
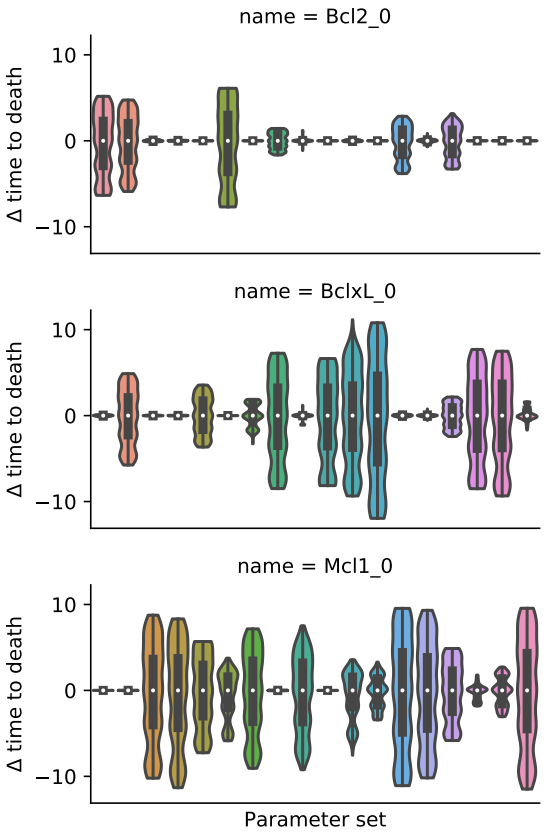


Figure 3.14: **Summary view of EARM 1.0 sensitivity.** Violin plot of the percent change of time-to-death per initial concentration across 100 of the best fit parameter sets for EARM 1.0.



(a) Anti-correlation between initial concentrations of BAX and BAK.



(b) Anti-correlation between initial concentrations of anti-apoptotic proteins.

Figure 3.15: a.) Change in time to death versus parameter set for the pro-apoptotic proteins BAX and BAK. A majority of the parameter sets show that the model is either sensitive to BAX or BAK. b.) Change in time to death versus parameter set for the anti-apoptotic proteins BCL2, BCL-XL, and MCL.

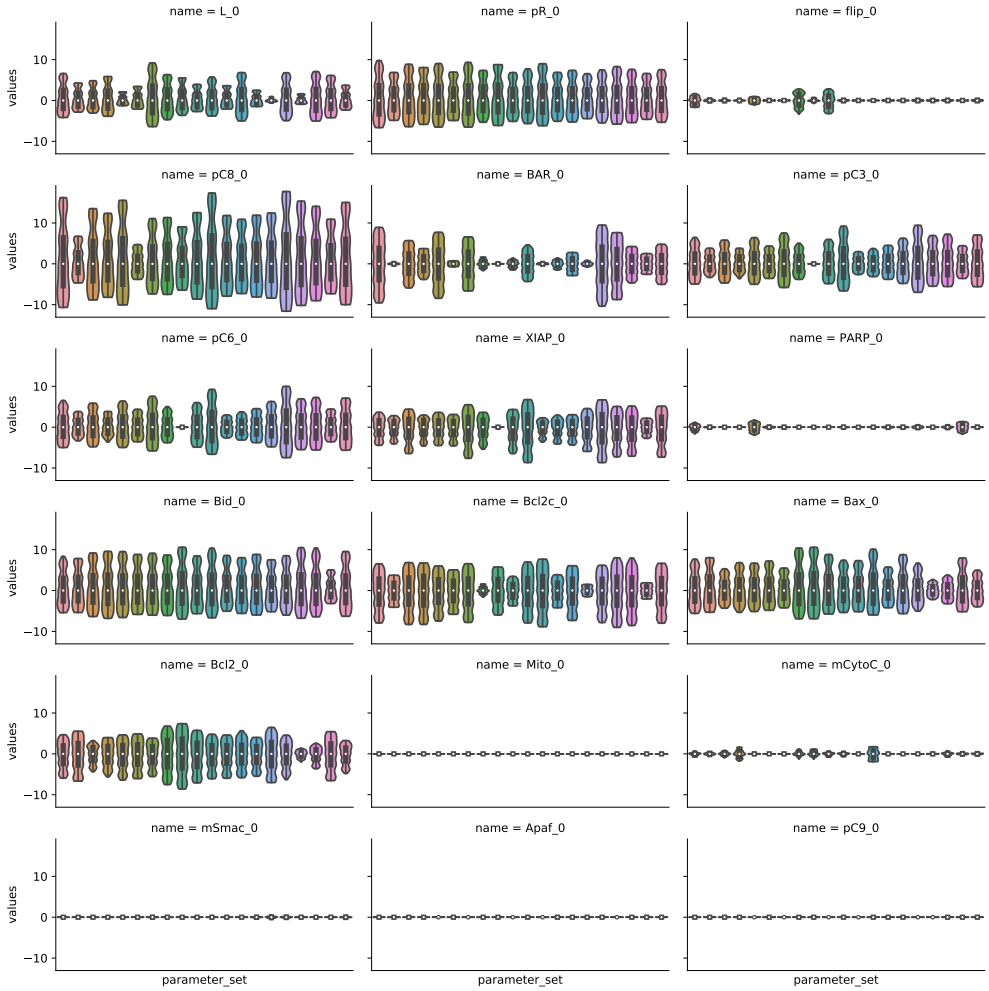


Figure 3.16: **Individual initial concentration sensitivity for EARM 1.0** Distribution of percent change of time-to-death vs parameter set for EARM 1.0. The data are grouped by initial condition and visualized in each subplot.

Interestingly, the grouped BAX terms and BCL impact model outcome for every parameter set. Thus, the added detail of EARM 2.0 compared to EARM 1.0 increases the difficulty in finding key perturbation patterns. If we were to use a single parameter set, rather than an ensemble, we might be lead to think that either BAX or BAK is required for apoptosis to occur. Since EARM 1.0 groups anti-apoptotic proteins and effector proteins, regardless of which parameter set we would see that the group impacts apoptosis. This simplification decreasing the degrees of freedom in the parameter space and easier to interpret the model sensitivities. This means that we must ensure that we use ensembles of parameter sets for such analysis, or try to keep the model as simple as possible. I believe that EARM 1.0 is a better model to use until more experimental constraints can be used to decipher the relative amounts of contribution. It should also be noted that despite the difference in EARM 1.0 and EARM 2.0, the sensitivities to remaining proteins are striking similar. C9, APAF, PARP, and SMAC initial conditions make little impact on model outcome, while C8, C3, C6, R, L, and BID contribute significantly. Regardless of which models complexity is used, we can use these information to prioritize or eliminate targets for future treatments. Our analysis showed that not all proteins in the EARM model impact output, regardless of the optimized parameters used. Such analysis would allow us to prioritize additional targets to gain therapeutic benefits. Proteins in the model that do not change cellular death should be lower in the perturbation priority list. This would be especially useful if the model being characterized was less studied than EARM.

Chapter 4

Conclusion and Future directions

The central contributions of this dissertation are 1.) the development of MAGINE to extract out key features and allow multi-scale exploration of large, time series, multi-omics datasets and 2.) development of modeling tools that greatly decrease the time in simulating, parameterizing, and characterizing models behaviour. Though seemingly distinct units, there are two of the main steps required to automate the creation of models from high throughput omics data.

Envision a scenario where a doctor biopsies a tumor and acquires multi-omic measurements to determine the state of the cell. They then construct a model using MAGINE that describes the main signaling activity that regulate the cancer. This information is then used to construct a mass action model. The model is then parameterized quickly with simplePSO, gathering all possible parameter values that could explain its state. Next, the model is characterized to find targetable molecular species. The doctor uses simulations to test drug combinations to predict outcome and side effects. This information is then used to create a patient specific treatment plan that maximizes the control of the cancer while also minimizing the side effects.

Though this dissertation does not reach this ultimate goal, it does provide advances that make it more realistic. Using MAGINE, it is possible to identify and integrate thousands of data points at multiple resolutions. It is possible to focus attention on these identified components and construct or expand more mechanistic models, such as EARM. Then, it is possible to leverage GPU simulations to characterize the model to quickly train and characterize the model. This information is used to determine drug combinations, which can then be quickly tested via simulation. Once an ideal treatment is identified, it is used in the clinic on the same patient in which the original sample came, providing a better, more personalized treatment.

4.1 Bridging the gap : Reactome2Rules

In order to complete the automatic model creation from data, a few remaining steps must be taken. MAGINE can help identify key components, yet converting that information into a mechanistic model is still a manual process. Expanding a model, such as EARM, based on this information is rather straight forward. This would involve adding the missing components and their interactions as shown in Fig 4.1. In this hypothetical demonstration, both pro and anti-apoptotic proteins are identified based on MAGINEs enrichment analysis and added to the pre-existing EARM model. Using PySB, new components can be built into different modular units. Then, these modules can be loaded and added to the EARM model. This makes the construction of the drug specific model easier than constructing all aspects of the module new each time. Then, the model could be trained, characterized, and used for *in silico* testing to determine an appropriate coarse of action to increase cell death.

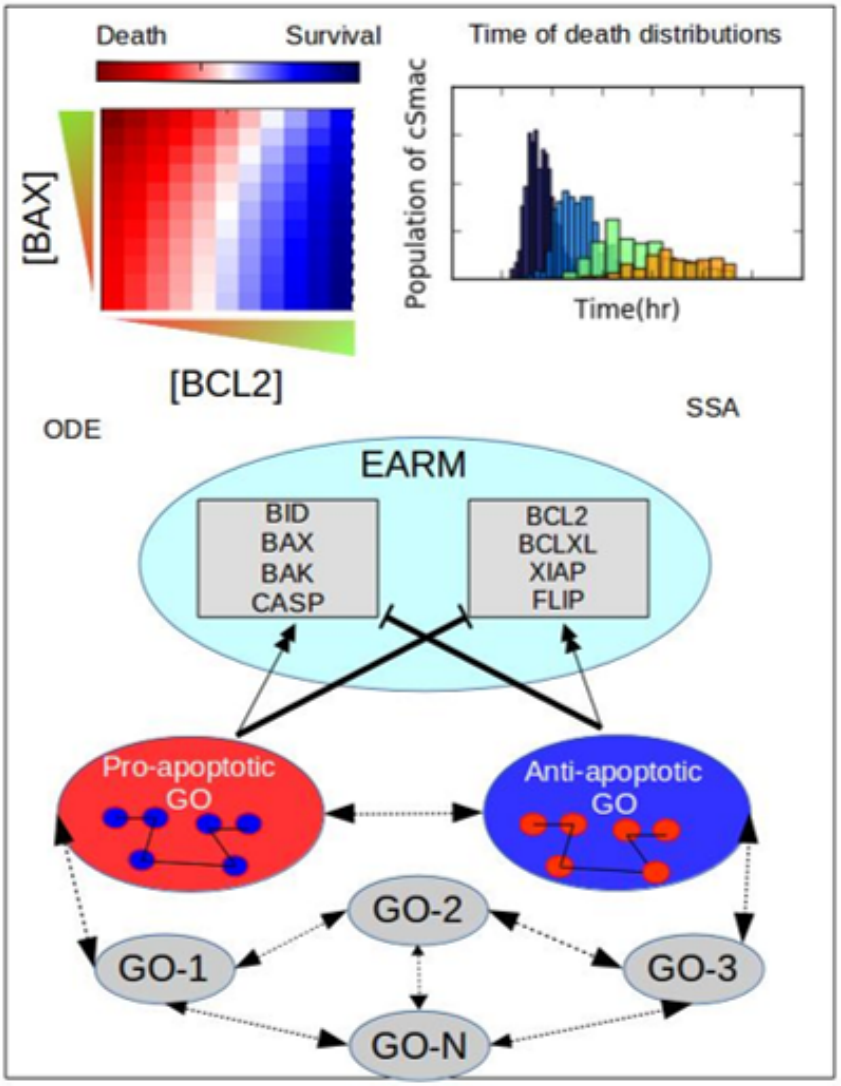


Figure 4.1: **The EARM model expanded based on enriched terms identified by MAGINE.** This figure represents the future goals of this thesis. The EARM model, which is already mechanistically detailed, would be expanded using knowledge obtained from MAGINE. Then, the model would be characterized to understand how each molecular species contributes to its outcome(top left). In this case increasing concentrations of BCL2 would increase likelihood of survival while increased concentrations of BAX would increase change of death. The model can be used to predict probability of cell death (top right), which would allow for *in silico* testing of perturbations for further experimental validation.

Ideally, the level of detail of the interactions, that is the state of each molecule (phosphorylation, cellular compartment, etc), would come from the experimental data. With that information known, it would be possible to reference Reactome (Matthews et al., 2009) and query information about how those species interact with the current model. Reactome provides the required level of detail. For instance, as shown in Figure 4.2, the event of BID activating BAX is shown. BID, which has to be truncated to amino acid 62-195, binds to BAX in the cytosol. It then activates BAX, which translocates into the mitochondria. Once there, BAX oligomerizes with other BAX molecules, become an active BAX species. This level of information is nearly identical to the manually created EARM model, yet was all automated with the what I have been calling 'Reactome2Rules'. It takes a list of species, find connecting interaction among them, then translates it into a PySB model. Though only an initial implementation, this is the most promising outlook of my many projects. Imagine automatically extending models based on new experimental data, quickly testing the impact of adding a few molecules into your model, without the need to extensively research and dig out those interactions. This is all dependent on the information stored in Reactome, and not all interactions are stored there (or known). The remaining steps for this to be fully implemented would involve fully mapping out all the Reactome API capabilities, developing macro level groups of rules to easily be imported into models, create a database for initial concentrations for the new species that are incorporated into the model. The Reactome database model is large. I have spent many hours trying to find a perfect mapping of information they have, and how it maps to information needed for a PySB model. Though time consuming, future students can hopefully endure to complete the Reactome2Rules.

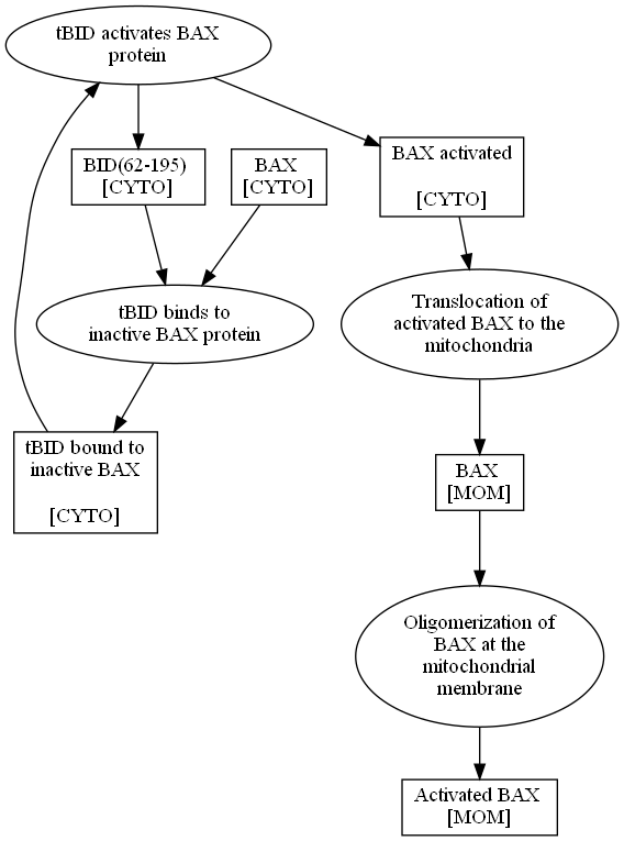


Figure 4.2: **Reactome2Rules creation of MAGINE edge 'BID activate BAX'**. Oval shapes represent 'Rules' and box shapes are molecular components. The state of the molecule is located at the bottom of the box.

Chapter 5

Appendix

Table 5.1: Reference initial species populations for EARM (Lopez et al., 2013). The PySB version of the model, `earm_lopez_embedded_flat.py`, is available at github.com/LoLab-VU/PySB_cupSODA_Bioinfo2017.

Parameter	Reference value
Apaf_0	100 000
Bad_0	1000
Bak_0	20 000
BAR_0	1000
Bax_0	80 000
Bcl2_0	20 000
BclxL_0	20 000
Bid_0	40 000
C3_0	10 000
C6_0	10 000
C8_0	20 000
C9_0	100 000
CytoC_0	500 000
flip_0	100
L_0	3000
Mcl1_0	20 000
Noxa_0	1000
PARP_0	1 000 000
R_0	200
Smac_0	100 000
XIAP_0	100 000

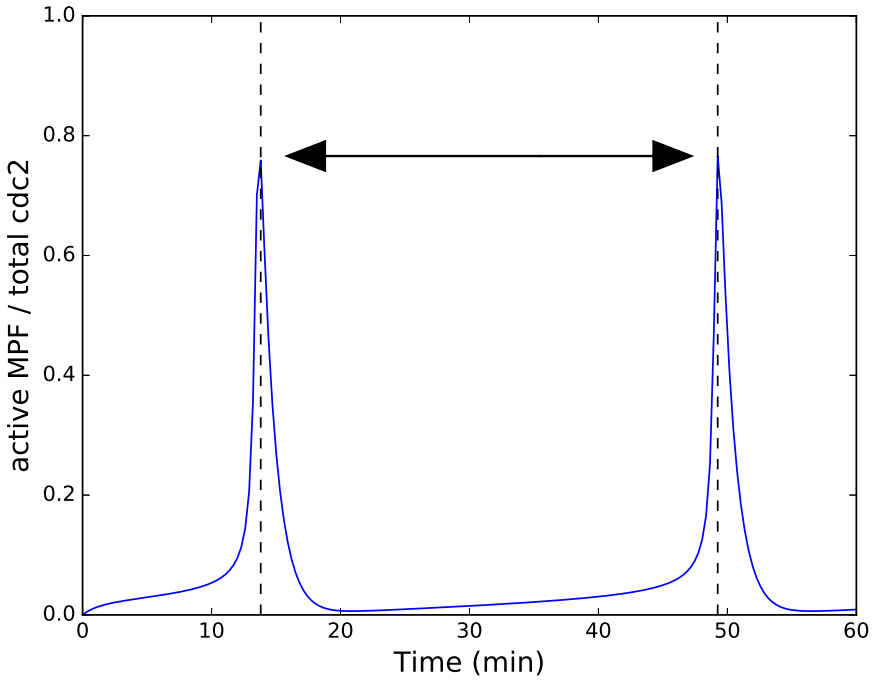


Figure 5.1: [**Time course for active maturation-promoting factor (MPF).**Time course for active maturation-promoting factor (MPF), as a ratio with respect to total cdc2, for the Tyson cell cycle model (Tyson, 1991). Model sensitivity to changes in the initial populations of cyclin and cdc2 was assessed with respect to the period of oscillation of active MPF (double arrow and black dashed lines).

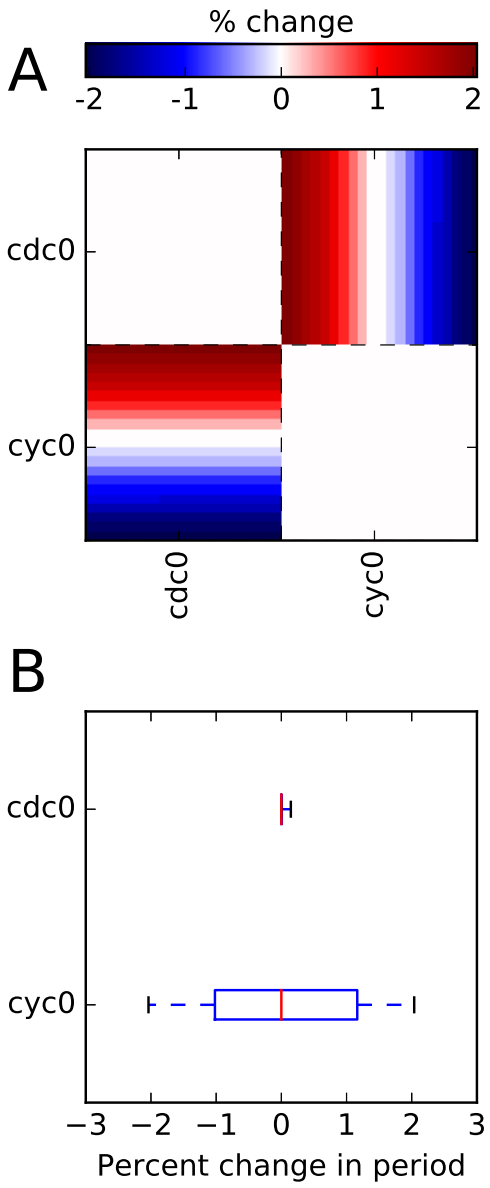


Figure 5.2: **Sensitivity analysis for the Tyson cell cycle model.**Sensitivity analysis for the Tyson cell cycle model Tyson (1991): (A) pairwise sensitivity matrix **P** (Eq. 3.6); (B) single-parameter sensitivity multisets $\mathbf{Q}_k$ (Eq. 3.7), plotted as boxplots (red lines are medians; boxes range from the first to third quartile; whiskers extend to the minimum and maximum values). `cdc0`: initial population of cdc2; `cyc0`: initial population of cyclin.

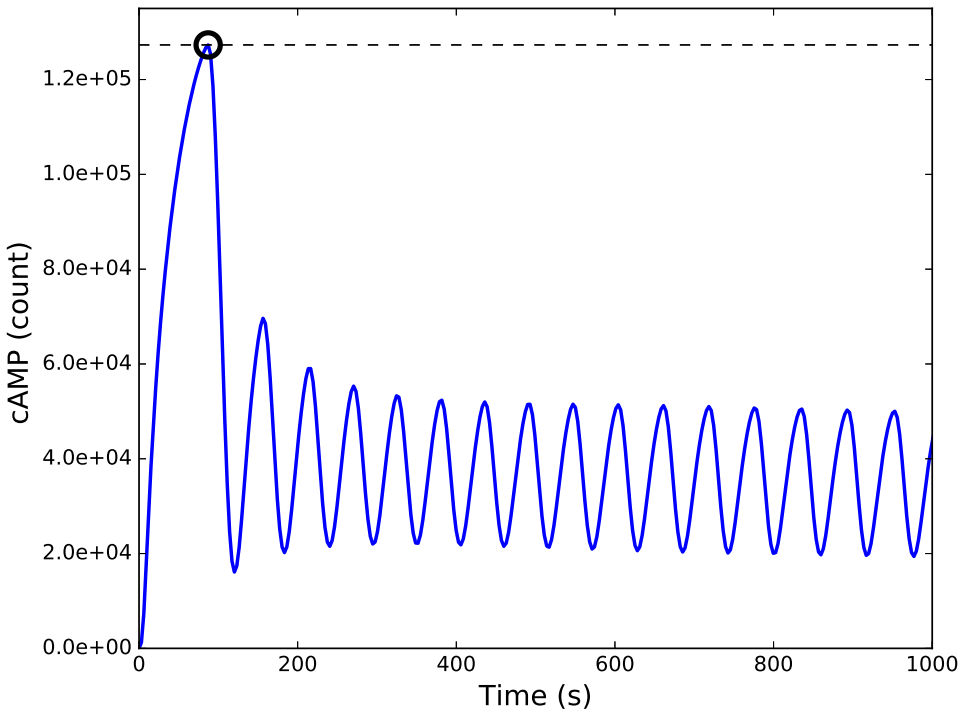


Figure 5.3: **Time course for cAMP in the Ras/cAMP/PKA model (Besozzi et al., 2012)**. Model sensitivity to changes in initial species populations was assessed with respect to the amplitude of the initial peak (the maximum value achieved; black circle and dashed black line).

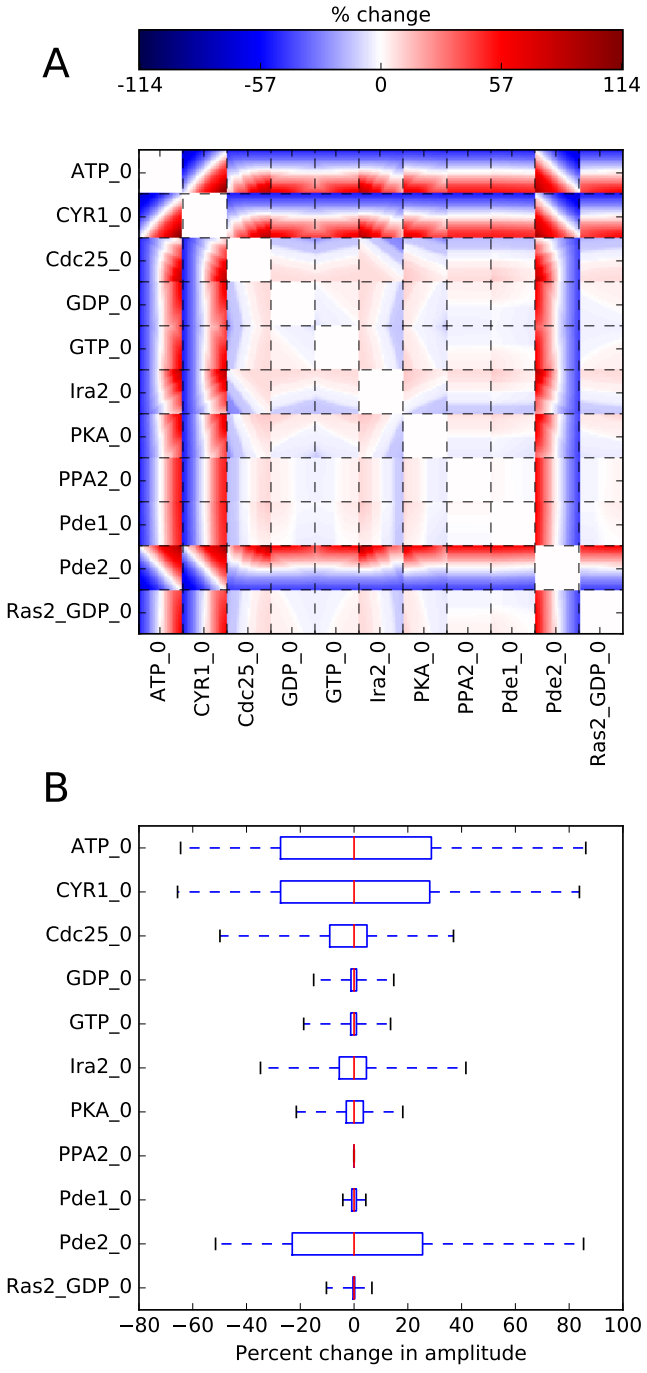


Figure 5.4: Sensitivity analysis for the Ras/cAMP/PKA model (Besozzi et al., 2012): (A) pairwise sensitivity matrix **P** (Eq. 3.6); (B) single-parameter sensitivity multisets $\mathbf{Q}_k$ (Eq. 3.7), plotted as boxplots (red lines are medians; boxes range from the first to third quartile; whiskers extend to the minimum and maximum values).

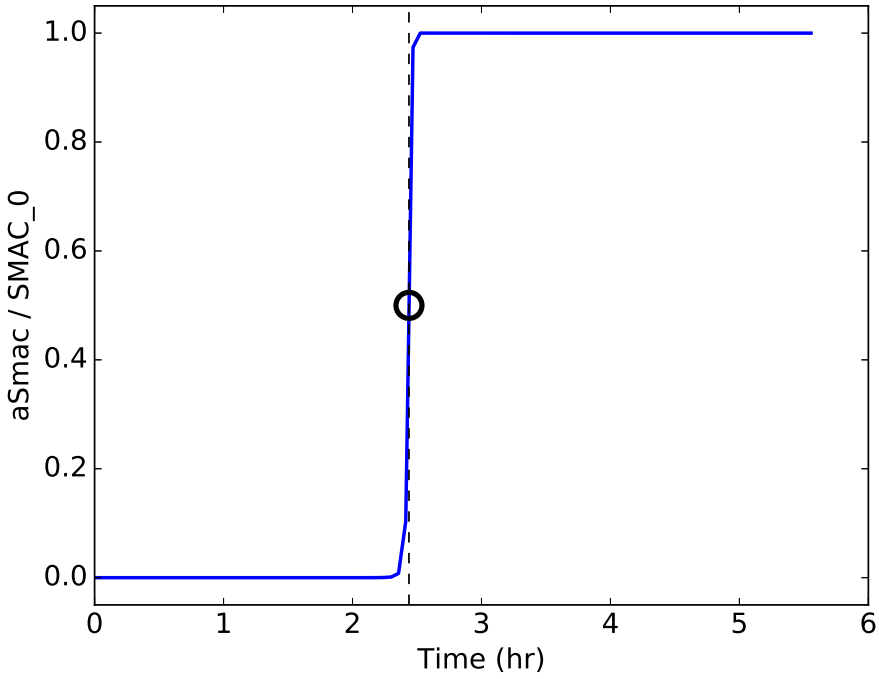


Figure 5.5: Time course for active Smac (aSmac), as a ratio with respect to total Smac (SMAC_0), in EARM (Lopez et al., 2013). Model sensitivity to changes in the initial species populations was assessed with respect to the "time-to-death," defined as the time point at which Smac reaches 50% cleavage (aSmac/SMAC_0 = 0.5; black circle and black dashed line).

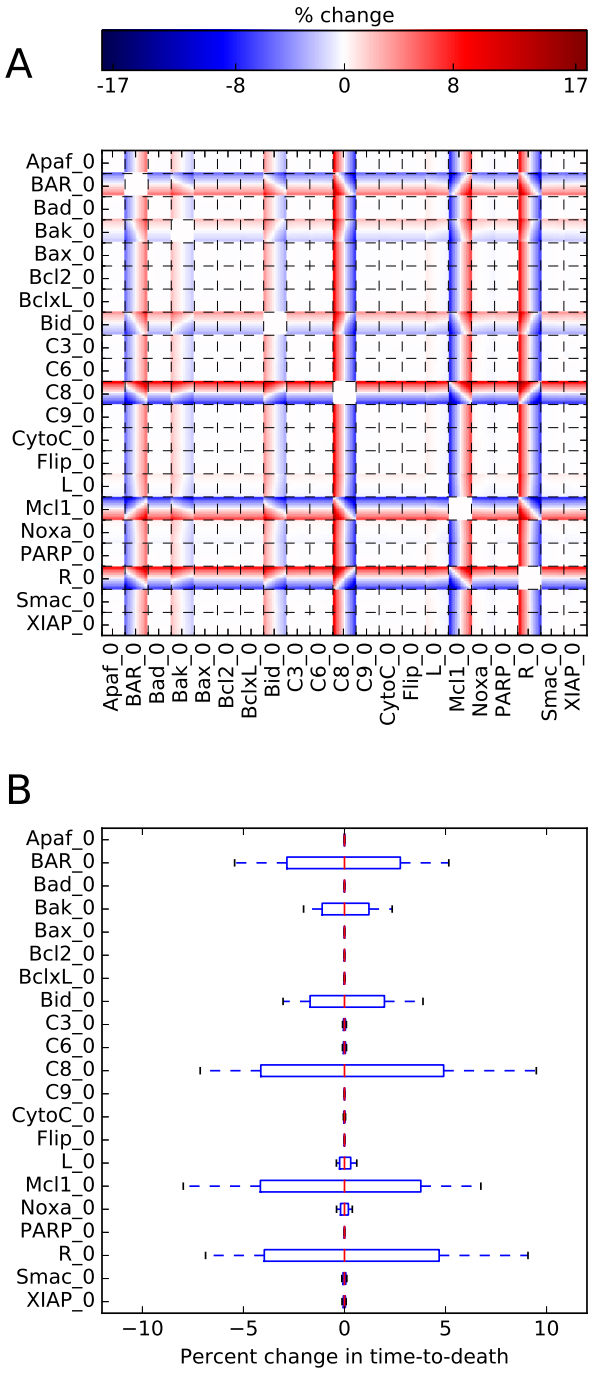


Figure 5.6: **Sensitivity analysis for parameter set 1 of EARM 2.0. (Lopez et al., 2013)**: (A) pairwise sensitivity matrix **P** (Eq. 3.6); (B) single-parameter sensitivity multisets $\mathbf{Q}_k$ (Eq. 3.7), plotted as boxplots (red lines are medians; boxes range from the first to third quartile; whiskers extend to the minimum and maximum values). Note that (B) is the same as Fig. **1**D in the main text.

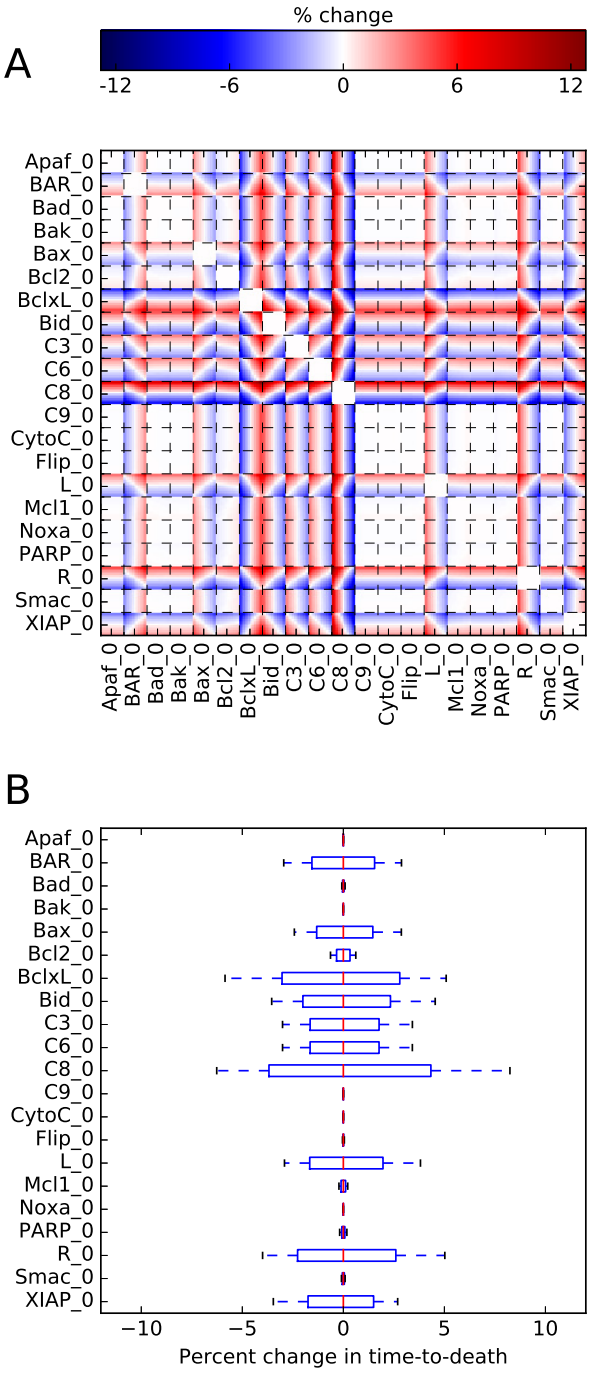


Figure 5.7: **Sensitivity analysis for parameter set 2 of EARM (Lopez et al., 2013)**: (A) pairwise sensitivity matrix **P** (Eq. 3.6); (B) single-parameter sensitivity multisets $\mathbf{Q}_k$ (Eq. 3.7), plotted as boxplots (red lines are medians; boxes range from the first to third quartile; whiskers extend to the minimum and maximum values). Note that this parameter set fits the experimental data comparably to that used in Fig. 5.6, yet shows sensitivity to a significantly different set of initial species populations.

5.1 MAGINE Jupyter notebooks

5.1.1 MAGINE data exploration

5.1.2 MAGINE network exploration

5.1.3 MAGINE enrichment analysis exploration

5.1.4 MAGINE AGN creation exploration

BIBLIOGRAPHY

Aebersold, R. and Mann, M. (2003). Mass spectrometry-based proteomics. *Nature*, 422(6928):198–207.

Aebersold, R. and Mann, M. (2016). Mass-spectrometric exploration of proteome structure and function. *Nature*, 537(7620):347–355.

Albeck, J. G., Burke, J. M., Spencer, S. L., Lauffenburger, D. A., and Sorger, P. K. (2008). Modeling a snap-action, variable-delay switch controlling extrinsic cell death. *PLoS biology*, 6(12):e299.

Aldridge, B. B., Burke, J. M., Lauffenburger, D. A., and Sorger, P. K. (2006). Physicochemical modelling of cell signalling pathways. *Nature cell biology*, 8(11):1195.

Bakker, B. M., Walsh, M. C., Ter Kuile, B. H., Mensonides, F. I., Michels, P. A., Opperdoes, F. R., and Westerhoff, H. V. (1999). Contribution of glucose transport to the control of the glycolytic flux in trypanosoma brucei. *Proceedings of the National Academy of Sciences*, 96(18):10098–10103.

Besozzi, D., Cazzaniga, P., Pescini, D., Mauri, G., Colombo, S., and Martegani, E. (2012). The role of feedback control mechanisms on the establishment of oscillatory regimes in the Ras/cAMP/PKA pathway in S. cerevisiae. *EURASIP J. Bioinforma. Syst. Biol.*, 2012(1):10.

Borner, C. and Andrews, D. W. (2014). The apoptotic pore on mitochondria: are we breaking through or still stuck? *Cell Death Differ.*, 21(2):187–91.

Brenner, D. and Mak, T. W. (2009). Mitochondrial cell death effectors. *Curr. Opin. Cell Biol.*, 21(6):871–877.

Bruford, E. A., Lush, M. J., Wright, M. W., Sneddon, T. P., Povey, S., and Birney, E. (2007). The hgnc database in 2008: a resource for the human genome. *Nucleic acids research*, 36(suppl_1):D445–D448.

Certo, M., Moore, V. D. G., Nishino, M., Wei, G., Korsmeyer, S., Armstrong, S. A., and Letai, A. (2006). Mitochondria primed by death signals determine cellular addiction to antiapoptotic BCL-2 family members. *Cancer Cell*, 9(5):351–365.

Chatr-Aryamontri, A., Oughtred, R., Boucher, L., Rust, J., Chang, C., Kolas, N. K., O'Donnell, L., Oster, S., Theesfeld, C., Sellam, A., Stark, C., Breitkreutz, B. J., Dolinski, K., and Tyers, M. (2017). The BioGRID interaction database: 2017 update. *Nucleic Acids Res.*, 45(D1):D369–D379.

Chaurushiya, M. S. and Weitzman, M. D. (2009). Viral manipulation of dna repair and cell cycle checkpoints. *DNA repair*, 8(9):1166–1176.

Chen, E. Y., Tan, C. M., Kou, Y., Duan, Q., Wang, Z., Meirelles, G. V., Clark, N. R., and Ma'ayan, A. (2013). Enrichr: Interactive and collaborative HTML5 gene list enrichment analysis tool. *BMC Bioinformatics*, 14(1).

Chen, W. W., Schoeberl, B., Jasper, P. J., Niepel, M., Nielsen, U. B., Lauffenburger, D. A., and Sorger, P. K. (2009). Input–output behavior of erbb signaling pathways as revealed by a mass action model trained against dynamic data. *Molecular systems biology*, 5(1):239.

Cheson, B. D. and Rummel, M. J. (2009). Bendamustine: Rebirth of an old drug. *J. Clin. Oncol.*, 27(9):1492–1501.

Chipuk, J. E., Fisher, J. C., Dillon, C. P., Kriwacki, R. W., Kuwana, T., and Green, D. R. (2008). Mechanism of apoptosis induction by inhibition of the anti-apoptotic BCL-2 proteins. *Proc. Natl. Acad. Sci. U. S. A.*, 105(51):20327–20332.

Chylek, L. A., Harris, L. A., Faeder, J. R., and Hlavacek, W. S. (2015). Modeling for (physical) biologists: an introduction to the rule-based approach. *Physical biology*, 12(4):045007.

Chylek, L. A., Harris, L. A., Tung, C.-S., Faeder, J. R., Lopez, C. F., and Hlavacek, W. S. (2014). Rule-based modeling: a computational approach for studying biomolecular site dynamics in cell signaling systems. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 6(1):13–36.

Coello Coello, C. a. and Reyes-Sierra, M. (2006). Multi-Objective Particle Swarm Optimizers: A Survey of the State-of-the-Art. *Int. J. Comput. Intell. Res.*, 2(3):287–308.

Cokelaer, T., Pultz, D., Harder, L. M., Serra-Musach, J., Saez-Rodriguez, J., and Valencia, A. (2013). BioServices: A common Python package to access biological Web Services programmatically. *Bioinformatics*, 29(24):3241–3242.

Csardi, G., Nepusz, T., et al. (2006). The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695(5):1–9.

Csete, M. E. and Doyle, J. C. (2002). Reverse engineering of biological complexity. *Science*, 295(5560):1664–9.

Danos, V., Feret, J., Fontana, W., Harmer, R., and Krivine, J. (2007). Rule-based modelling of cellular signalling. In *International conference on concurrency theory*, pages 17–41. Springer.

Dematté, L. and Prandi, D. (2010). Gpu computing for systems biology. *Briefings in bioinformatics*, 11(3):323–333.

Elkholi, R., Renault, T. T., Serasinghe, M. N., and Chipuk, J. E. (2014). Putting the pieces together: How is the mitochondrial pathway of apoptosis regulated in cancer and chemotherapy? *Cancer Metab.*, 2(1):16.

Ellson, J., Gansner, E., Koutsofios, L., North, S. C., and Woodhull, G. (2001). Graphviz—open source graph drawing tools. In *International Symposium on Graph Drawing*, pages 483–484. Springer.

Eydgahi, H., Chen, W. W., Muhlich, J. L., Vitkup, D., Tsitsiklis, J. N., and Sorger, P. K. (2013). Properties of cell death models calibrated and compared using Bayesian approaches. *Mol. Syst. Biol.*, 9:644.

Faeder, J. R., Blinov, M. L., and Hlavacek, W. S. (2009). Rule-based modeling of biochemical systems with bionetgen. In *Systems biology*, pages 113–167. Springer.

Fischer, U. and Schulze-Osthoff, K. (2005). New approaches and therapeutics targeting apoptosis in disease. *Pharmacological reviews*, 57(2):187–215.

Fisher, J. and Henzinger, T. A. (2007). Executable cell biology. *Nat. Biotechnol.*, 25(11):1239–49.

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., and Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175.

Franz, M., Lopes, C. T., Huck, G., Dong, Y., Sumer, O., and Bader, G. D. (2015). Cytoscape. js: a graph theory library for visualisation and analysis. *Bioinformatics*, 32(2):309–311.

Fulda, S. and Debatin, K.-M. (2006). Extrinsic versus intrinsic apoptosis pathways in anticancer chemotherapy. *Oncogene*, 25:4798–4811.

Gaudet, S., Spencer, S. L., Chen, W. W., and Sorger, P. K. (2012). Exploring the contextual sensitivity of factors that determine cell-to-cell variability in receptor-mediated apoptosis. *PLoS Comput. Biol.*, 8(4).

Ghobrial, I. M., Witzig, T. E., and Adjei, A. A. (2005). Targeting apoptosis pathways in cancer therapy. *CA: a cancer journal for clinicians*, 55(3):178–194.

Gilbert, G. (1972). Distance between sets. *Nature*, 239(5368):174.

Gillespie, D. T. (1976). A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of computational physics*, 22(4):403–434.

Gillespie, D. T. (2001). Approximate accelerated stochastic simulation of chemically reacting systems. *The Journal of chemical physics*, 115(4):1716–1733.

Gillespie, D. T. (2007). Stochastic simulation of chemical kinetics. *Annu. Rev. Phys. Chem.*, 58:35–55.

Gillies, L. A., Du, H., Peters, B., Knudson, C. M., Newmeyer, D. D., and Kuwana, T. (2015). Visual and functional demonstration of growing Bax-induced pores in mitochondrial outer membranes. *Mol. Biol. Cell*, 26(2):339–49.

Gutenkunst, R. N., Waterfall, J. J., Casey, F. P., Brown, K. S., Myers, C. R., and Sethna, J. P. (2007). Universally sloppy parameter sensitivities in systems biology models. *PLoS computational biology*, 3(10).

Gutierrez, D. B., Gant-Branum, R. L., Romer, C. E., Farrow, M. A., Allen, J. L., Dahal, N., Nei, Y. W., Codreanu, S. G., Jordan, A. T., Palmer, L. D., Sherrod, S. D., McLean, J. A., Skaar, E. P., Norris, J. L., and Caprioli, R. M. (2018). An Integrated, High-Throughput Strategy for Multiomic Systems Level Analysis. *J. Proteome Res.*, 17(10):3396–3408.

Hagberg, A., Schult, D., and Swart, P. (2005). Networkx: Python software for the analysis of networks. *Mathematical Modeling and Analysis, Los Alamos National Laboratory*.

Hanahan, D. and Weinberg, R. A. (2011). Hallmarks of cancer: the next generation. *cell*, 144(5):646–674.

Harris, L. A., Hogg, J. S., Tapia, J.-J., Sekar, J. A., Gupta, S., Korsunsky, I., Arora, A., Barua, D., Sheehan, R. P., and Faeder, J. R. (2016). Bionetgen 2.2: advances in rule-based modeling. *Bioinformatics*, 32(21):3366–3368.

Harris, L. A., Nobile, M. S., Pino, J. C., Lubbock, A. L., Besozzi, D., Mauri, G., Cazzaniga, P., and Lopez, C. F. (2017). Gpu-powered model analysis with pysb/cupsoda. *Bioinformatics*, 33(21):3492–3494.

Harris, L. A., Piccirilli, A. M., Majusiak, E. R., and Clancy, P. (2009). Quantifying stochastic effects in biochemical reaction networks using partitioned leaping. *Physical Review E*, 79(5):051906.

Hood, L., Heath, J. R., Phelps, M. E., and Lin, B. (2004). Systems biology and new technologies enable predictive and preventative medicine. *Science*, 306(5696):640–643.

Huang, D. W., Sherman, B. T., and Lempicki, R. A. (2008). Bioinformatics enrichment tools: paths toward the comprehensive functional analysis of large gene lists. *Nucleic acids research*, 37(1):1–13.

Huang, D. W., Sherman, B. T., Tan, Q., Kir, J., Liu, D., Bryant, D., Guo, Y., Stephens, R., Baseler, M. W., Lane, H. C., et al. (2007). David bioinformatics resources: expanded annotation database and novel algorithms to better extract biology from large gene lists. *Nucleic acids research*, 35(suppl_2):W169–W175.

Huang, S. (2012). The molecular and mathematical basis of waddington's epigenetic landscape: A framework for post-darwinian biology? *Bioessays*, 34(2):149–157.

Hucka, M., Finney, A., Sauro, H. M., Bolouri, H., Doyle, J. C., Kitano, H., Arkin, A. P., Bornstein, B. J., Bray, D., Cornish-Bowden, A., Cuellar, A. A., Dronov, S., Gilles, E. D., Ginkel, M., Gor, V., Goryanin, I. I., Hedley, W. J., Hodgman, T. C., Hofmeyr, J. H., Hunter, P. J., Juty, N. S., Kasberger, J. L., Kremling, A., Kummer, U., Le Novère, N., Loew, L. M., Lucio, D., Mendes, P., Minch, E., Mjolsness, E. D., Nakayama, Y., Nelson, M. R., Nielsen, P. F., Sakurada, T., Schaff, J. C., Shapiro, B. E., Shimizu, T. S., Spence, H. D., Stelling, J., Takahashi, K., Tomita, M., Wagner, J., and Wang, J. (2003). The systems biology markup language (SBML): A medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531.

Humphries, E. H. and Temin, H. M. (1972). Cell cycle-dependent activation of rous sarcoma virus-infected stationary chicken cells: avian leukosis virus group-specific antigens and ribonucleic acid. *Journal of virology*, 10(1):82–87.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90.

Ideker, T., Galitski, T., and Hood, L. (2001). A new approach to decoding life: systems biology. *Annual review of genomics and human genetics*, 2(1):343–372.

Ingalls, B. P. (2013). *Mathematical modeling in systems biology: an introduction*. MIT press.

Kanehisa, M., Sato, Y., Kawashima, M., Furumichi, M., and Tanabe, M. (2016). KEGG as a reference resource for gene and protein annotation. *Nucleic Acids Res.*, 44(D1):D457–D462.

Kearns, J. D., Basak, S., Werner, S. L., Huang, C. S., and Hoffmann, A. (2006). Iκbε provides negative feedback to control nf-κb oscillations, signaling dynamics, and inflammatory gene expression. *The Journal of cell biology*, 173(5):659–664.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. *1995 IEEE Int. Conf. Neural Networks (ICNN 95)*, 4(1):1942–1948.

Kholodenko, B. N. (2006). Cell-signalling dynamics in time and space. *Nature reviews Molecular cell biology*, 7(3):165–176.

Kim, K. A., Spencer, S. L., Albeck, J. G., Burke, J. M., Sorger, P. K., Gaudet, S., and Kim, D. H. (2010). Systematic calibration of a cell signaling network model. *BMC Bioinformatics*, 11:202.

King, O. D., Foulger, R. E., Dwight, S. S., White, J. V., and Roth, F. P. (2003). Predicting gene function from patterns of annotation. *Genome research*, 13(5):896–904.

Kitano, H. (2002). Computational systems biology. *Nature*, 420(6912):206–210.

Klöckner, A., Pinto, N., Lee, Y., Catanzaro, B., Ivanov, P., and Fasih, A. (2012). Pycuda and pyopencl: A scripting-based approach to gpu run-time code generation. *Parallel Computing*, 38(3):157–174.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B. E., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J. B., Grout, J., Corlay, S., et al. (2016). Jupyter notebooks-a publishing format for reproducible computational workflows. In *ELPUB*, pages 87–90.

Knuth, D. E. (1997). *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Komurov, K., Dursun, S., Erdin, S., and Ram, P. T. (2012). NetWalker: A contextual network analysis tool for functional genomics. *BMC Genomics*, 13(1).

Kremling, A. and Saez-Rodriguez, J. (2007). Systems biology—an engineering perspective. *Journal of biotechnology*, 129(2):329–351.

Kuleshov, M. V., Jones, M. R., Rouillard, A. D., Fernandez, N. F., Duan, Q., Wang, Z., Koplev, S., Jenkins, S. L., Jagodnik, K. M., Lachmann, A., McDermott, M. G., Monteiro, C. D., Gundersen, G. W., and Ma'ayan, A. (2016). Enrichr: a comprehensive gene set enrichment analysis web server 2016 update. *Nucleic Acids Res.*, 44(W1):W90–W97.

Kuwana, T., Mackey, M. R., Perkins, G., Ellisman, M. H., Latterich, M., Schneiter, R., Green, D. R., and Newmeyer, D. D. (2002). Bid, Bax, and lipids cooperate to form supramolecular openings in the outer mitochondrial membrane. *Cell*, 111(3):331–342.

Le Novere, N. (2015). Quantitative and logic modelling of molecular and gene networks. *Nature Reviews Genetics*, 16(3):146–158.

Leoni, L. M. and Hartley, J. A. (2011). Mechanism of Action: The Unique Pattern of Bendamustine-Induced Cytotoxicity. *Semin. Hematol.*, 48(SUPPL. 1):S12–S23.

Li, H. and Petzold, L. (2010). Efficient parallelization of the stochastic simulation algorithm for chemically reacting systems on the graphics processing unit. *The International Journal of High Performance Computing Applications*, 24(2):107–116.

Lockshin, R. (2016). Programmed cell death 50 (and beyond). *Cell Death & Differentiation*, 23(1):10–17.

Lopez, C. F., Muhlich, J. L., Bachman, J. A., and Sorger, P. K. (2013). Programming biological models in python using pysb. *Molecular systems biology*, 9(1):646.

Martinou, J. C. and Youle, R. J. (2011). Mitochondria in Apoptosis: Bcl-2 Family Members and Mitochondrial Dynamics. *Dev. Cell*, 21(1):92–101.

Massey Jr, F. J. (1951). The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78.

Matthews, L., Gopinath, G., Gillespie, M., Caudy, M., Croft, D., de Bono, B., Garapati, P., Hemish, J., Hermjakob, H., Jassal, B., Kanapin, A., Lewis, S., Mahajan, S., May, B., Schmidt, E., Vastrik, I., Wu, G., Birney, E., Stein, L., and D'eustachio, P. (2009). Reactome knowledgebase of human biological pathways and processes. *Nucleic Acids Res.*, 37(SUPPL. 1):619–622.

McCollum, J. M., Peterson, G. D., Cox, C. D., Simpson, M. L., and Samatova, N. F. (2006). The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior. *Comput. Biol. Chem.*, 30(1):39–49.

McKinney, W. et al. (2010). Data structures for statistical computing in python.

Narayan, V., Ly, T., Pourkarimi, E., Murillo, A. B., Gartner, A., Lamond, A. I., and Kenyon, C. (2016). Deep Proteome Analysis Identifies Age-Related Processes in C. elegans. *Cell Syst.*, 3(2):1–16.

Nickolls, J. et al. (2008). Scalable parallel programming with CUDA. *ACM Queue*, 6:40–53.

Nobile, M. S., Besozzi, D., Cazzaniga, P., Mauri, G., and Pescini, D. (2013). CupSODA: A cuda-powered simulator of mass-action kinetics. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, 7979 LNCS:344–357.

Nobile, M. S., Cazzaniga, P., Besozzi, D., and Mauri, G. (2014). GPU-accelerated simulations of mass-action kinetics models with cupSODA. *J. Supercomput.*, 69(1):17–24.

Nobile, M. S. et al. (2016). Graphics processing units in bioinformatics, computational biology and systems biology. *Brief. Bioinform.*, 2016:bbw058.

Norris, J. L., Farrow, M. A., Gutierrez, D. B., Palmer, L. D., Muszynski, N., Sherrod, S. D., Pino, J. C., Allen, J. L., Spraggins, J. M., Lubbock, A. L., et al. (2017). Integrated, high-throughput, multiomics platform enables data-driven construction of cellular responses and reveals global drug mechanisms of action. *Journal of proteome research*, 16(3):1364–1375.

Ogata, H., Goto, S., Sato, K., Fujibuchi, W., Bono, H., and Kanehisa, M. (1999). KEGG: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Res.*, 27(1):29–34.

Oliphant, T. E. (2007). Python for scientific computing. *Comput. Sci. Eng.*, 9:10–20.

Omony, J. (2014). Biological network inference: A review of methods and assessment of tools and techniques. *Annual Research & Review in Biology*, pages 577–601.

Ono, K., Muetze, T., Kolishovski, G., Shannon, P., and Demchak, B. (2015). Cyrest: Turbocharging cytoscape access for external tools via a restful api. *F1000Research*, 4.

Palmer, L. D., Jordan, A. T., Maloney, K. N., Farrow, M. A., Gutierrez, D. B., Gant-Branum, R., Burns, W. J., Romer, C. E., Tsui, T., Allen, J. L., Beavers, W. N., Nei, Y.-W., Sherrod, S. D., Lacy, D. B., Norris, J. L., McLean, J. A., Caprioli, R. M., and Skaar, E. P. (2019). Zinc intoxication induces ferroptosis in A549 human lung cells. *Metallomics*.

Papin, J. A., Hunter, T., Palsson, B. O., and Subramaniam, S. (2005). Reconstruction of cellular signalling networks and analysis of their properties. *Nature reviews Molecular cell biology*, 6(2):99.

Papin, J. A., Stelling, J., Price, N. D., Klamt, S., Schuster, S., and Palsson, B. O. (2004). Comparison of network-based pathway analysis methods. *Trends in biotechnology*, 22(8):400–405.

Pérez, F., Granger, B. E., and Hunter, J. D. (2011). Python: An ecosystem for scientific computing. *Comput. Sci. Eng.*, 13(2):13–21.

Perfetto, L., Briganti, L., Calderone, A., Perpetuini, A. C., Iannuccelli, M., Langone, F., Licata, L., Marinkovic, M., Mattioni, A., Pavlidou, T., Peluso, D., Petrilli, L. L., Pirró, S., Posca, D., Santonico, E., Silvestri, A., Spada, F., Castagnoli, L., and Cesareni, G. (2016). SIGNOR: A database of causal relationships between biological entities. *Nucleic Acids Res.*, 44(D1):D548–D554.

Petzold, L. (1983). Automatic Selection of Methods for Solving Stiff and Nonstiff Systems of Ordinary Differential Equations.

Rathinam, M., Petzold, L. R., Cao, Y., and Gillespie, D. T. (2003). Stiffness in stochastic chemically reacting systems: The implicit tau-leaping method. *J. Chem. Phys.*, 119(24):12784.

Reed, J. C. (2000). Mechanisms of apoptosis. *The American journal of pathology*, 157(5):1415–1430.

Ressa, A., Fitzpatrick, M., Van Den Toorn, H., Heck, A. J., and Altelaar, M. (2019). PaDuA: A Python Library for High-Throughput (Phospho)proteomics Data Analysis. *J. Proteome Res.*, 18(2):576–584.

Sacco, F., Silvestri, A., Posca, D., Pirrò, S., Gherardini, P. F., Castagnoli, L., Mann, M., and Cesareni, G. (2016). Deep Proteomics of Breast Cancer Cells Reveals that Metformin Rewires Signaling Networks Away from a Pro-growth State. *Cell Syst.*, 2(3):159–171.

Sanft, K. R., Wu, S., Roh, M., Fu, J., Lim, R. K., and Petzold, L. R. (2011). Stochkit2: software for discrete stochastic simulation of biochemical systems with events. *Bioinformatics*, 27(17):2457–2458.

Schleich, K. and Lavrik, I. N. (2013). Mathematical modeling of apoptosis. *Cell Communication and Signaling*, 11(1):44.

Schlögl, F. (1972). Chemical reaction models for non-equilibrium phase transitions. *Zeitschrift für physik*, 253(2):147–161.

Shannon, P., Markiel, A., Ozier, O., Baliga, N. S., Wang, J. T., Ramage, D., Amin, N., Schwikowski, B., and Ideker, T. (2003). Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498–2504.

Shimada, N., Uchimaru, K., Yuji, K., Oyaizu, N., Tojo, A., Koibuchi, T., and Ohno, N. (2015). Treatment of chronic lymphocytic leukemia with bendamustine in an HIV-infected patient on antiretroviral therapy: a case report and review of the literature. *Clin. Case Reports*, 3(6):453–460.

Shockley, E. M., Vrugt, J. A., and Lopez, C. F. (2018). Pydream: high-dimensional parameter inference for biological models in python. *Bioinformatics*, 34(4):695–697.

Sievert, C., Parmer, C., Hocking, T., Chamberlain, S., Ram, K., Corvellec, M., and Despouy, P. (2016). plotly: Create interactive web graphics via plotly's javascript graphing library [software].

Slee, E. A., Harte, M. T., Kluck, R. M., Wolf, B. B., Casiano, C. A., Newmeyer, D. D., Wang, H.-G., Reed, J. C., Nicholson, D. W., Alnemri, E. S., et al. (1999). Ordering the cytochrome c–initiated caspase cascade: hierarchical activation of caspases-2,-3,-6,-7,-8, and-10 in a caspase-9–dependent manner. *The Journal of cell biology*, 144(2):281–292.

Sneddon, M. W., Faeder, J. R., and Emonet, T. (2011). Efficient modeling, simulation and coarse-graining of biological complexity with nfsim. *Nature methods*, 8(2):177.

Spencer, S. L., Gaudet, S., Albeck, J. G., Burke, J. M., and Sorger, P. K. (2009). Non-genetic origins of cell-to-cell variability in trail-induced apoptosis. *Nature*, 459(7245):428–432.

Stark, C. (2006). BioGRID: a general repository for interaction datasets. *Nucleic Acids Res.*, 34(90001):D535–D539.

Stuckey, D. W. and Shah, K. (2013). Trail on trial: preclinical advances in cancer therapy. *Trends in molecular medicine*, 19(11):685–694.

Subramanian, A., Tamayo, P., Mootha, V. K., Mukherjee, S., Ebert, B. L., Gillette, M. A., Paulovich, A., Pomeroy, S. L., Golub, T. R., Lander, E. S., et al. (2005). Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *Proceedings of the National Academy of Sciences*, 102(43):15545–15550.

Suderman, R. and Deeds, E. J. (2013). Machines vs. Ensembles: Effective MAPK Signaling through Heterogeneous Sets of Protein Complexes. *PLoS Comput. Biol.*, 9(10):e1003278.

Tang, H., Thomas, P. D., Kang, D., Mills, C., Muruganujan, A., Huang, X., and Mi, H. (2016). PANTHER version 11: expanded annotation data from Gene Ontology and Reactome pathways, and data analysis tool enhancements. *Nucleic Acids Res.*, 45(D1):D183–D189.

Tapia, J.-J. and Faeder, J. R. (2013). The atomizer: extracting implicit molecular structure from reaction network models. In *Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics*, pages 726–727.

Taylor, R. C., Cullen, S. P., and Martin, S. J. (2008). Apoptosis: controlled demolition at the cellular level. *Nat. Rev. Mol. Cell Biol.*, 9(3):231–241.

Thompson, C. B. (1995). Apoptosis in the pathogenesis and treatment of disease. *Science*, 267(5203):1456–1462.

Toby, T. K., Fornelli, L., and Kelleher, N. L. (2016). Progress in top-down proteomics and the analysis of proteoforms. *Annual review of analytical chemistry*, 9:499–519.

Torre, D., Lachmann, A., and Ma'ayan, A. (2018). BioJupies: Automated Generation of Interactive Notebooks for RNA-Seq Data Analysis in the Cloud. *Cell Syst.*, 7(5):556–561.e3.

Tyson, J. J. (1991). Modeling the cell division cycle: cdc2 and cyclin interactions. *Proc. Natl. Acad. Sci. U. S. A.*, 88(16):7328–7332.

Van Der Walt, S., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22.

Volkmann, N., Marassi, F. M., Newmeyer, D. D., and Hanein, D. (2014). The rheostat in the membrane: BCL-2 family proteins and apoptosis. *Cell Death Differ.*, 21(2):206–15.

Vrugt, J. A. (2014). Markov chain Monte Carlo Simulation Using the DREAM Software Package: Theory, Concepts, and MATLAB Implementation. *Environ. Model. Softw.*, page 82.

Vrugt, J. A. and Ter Braak, C. J. F. (2011). DREAM(D): An adaptive Markov Chain Monte Carlo simulation algorithm to solve discrete, noncontinuous, and combinatorial posterior parameter estimation problems. *Hydrol. Earth Syst. Sci.*, 15(12):3701–3713.

Wang, J., Ma, Z., Carr, S. A., Mertins, P., Zhang, H., Zhang, Z., Chan, D. W., Ellis, M. J. C., Townsend, R. R., Smith, R. D., McDermott, J. E., Chen, X., Paulovich, A. G., Boja, E. S., Mesri, M., Kinsinger, C. R., Rodriguez, H., Rodland, K. D., Liebler, D. C., and Zhang, B. (2017). Proteome Profiling Outperforms Transcriptome Profiling for Coexpression Based Gene Function Prediction. *Mol. Cell. Proteomics*, 16(1):121–134.

Wang, Z., Gerstein, M., and Snyder, M. (2009). Rna-seq: a revolutionary tool for transcriptomics. *Nature reviews genetics*, 10(1):57.

Waskom, M. et al. (2017). mwaskom/seaborn: v0.8.1 (september 2017).

Wishart, D. S., Knox, C., Guo, A. C., Eisner, R., Young, N., Gautam, B., Hau, D. D., Psychogios, N., Dong, E., Bouatra, S., Mandal, R., Sinelnikov, I., Xia, J., Jia, L., Cruz, J. A., Lim, E., Sobsey, C. A., Shrivastava, S., Huang, P., Liu, P., Fang, L., Peng, J., Fradette, R., Cheng, D., Tzur, D., Clements, M., Lewis, A., De souza, A., Zuniga, A., Dawe, M., Xiong, Y., Clive, D., Greiner, R., Nazyrova, A., Shaykhutdinov, R., Li, L., Vogel, H. J., and Forsythei, I. (2009). HMDB: A knowledgebase for the human metabolome. *Nucleic Acids Res.*, 37(SUPPL. 1):603–610.

Wishart, D. S., Tzur, D., Knox, C., Eisner, R., Guo, A. C., Young, N., Cheng, D., Jewell, K., Arndt, D., Sawhney, S., Fung, C., Nikolai, L., Lewis, M., Coutouly, M. A., Forsythe, I., Tang, P., Shrivastava, S., Jeroncic, K., Stothard, P., Amegbey, G., Block, D., Hau, D. D., Wagner, J., Miniaci, J., Clements, M., Gebremedhin, M., Guo, N., Zhang, Y., Duggan, G. E., MacInnis, G. D., Weljie, A. M., Dowlatabadi, R., Bamforth, F., Clive, D., Greiner, R., Li, L., Marrie, T., Sykes, B. D., Vogel, H. J., and Querengesser, L. (2007). HMDB: The human metabolome database. *Nucleic Acids Res.*, 35(SUPPL. 1):521–526.

Wu, G., Feng, X., and Stein, L. (2010). A human functional protein interaction network and its application to cancer data analysis. *Genome Biol.*, 11(5):R53.

Zhang, R., Mehla, R., and Chauhan, A. (2010). Perturbation of host nuclear membrane component RanBP2 impairs the nuclear import of human immunodeficiency virus -1 preintegration complex (DNA). *PLoS One*, 5(12).