

Algorithms for regulatory network inference and experiment planning in systems biology

Aditya Pratapa

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

T. M. Murali, Chair
Lenwood S. Heath
John Kececioglu
B. Aditya Prakash
John J. Tyson

June 18, 2020
Blacksburg, Virginia

Keywords: network biology, experiment planning, gene regulatory networks, deep learning,
single cell transcriptomics
Copyright 2020, Aditya Pratapa

Algorithms for regulatory network inference and experiment planning in systems biology

Aditya Pratapa

(ABSTRACT)

I present novel solutions to two different classes of computational problems that arise in the study of complex cellular processes. The first problem arises in the context of planning large-scale genetic cross experiments that can be used to validate predictions of multigenic perturbations made by mathematical models.

- (i) I present CrossPlan, a novel methodology for systematically planning genetic crosses to make a set of target mutants from a set of source mutants. CrossPlan is based on a generic experimental workflow used in performing genetic crosses in budding yeast. CrossPlan uses an integer-linear-program (ILP) to maximize the number of target mutants that we can make under certain experimental constraints. I apply it to a comprehensive mathematical model of the protein regulatory network controlling cell division in budding yeast.
- (ii) I formulate several natural problems related to efficient synthesis of a target mutant from source mutants. These formulations capture experimentally-useful notions of verifiability (e.g., the need to confirm that a mutant contains mutations in the desired genes) and permissibility (e.g., the requirement that no intermediate mutants in the synthesis be inviable). I present several polynomial time or fixed-parameter tractable algorithms for optimal synthesis of a target mutant for special cases of the problem that arise in practice.

The second problem I address is inferring gene regulatory networks (GRNs) from single cell transcriptomic (scRNA-seq) data. These GRNs can serve as starting points to build mathematical models.

- (iii) I present BEELINE, a comprehensive evaluation of state-of-the-art algorithms for inferring gene regulatory networks (GRNs) from single-cell gene expression data. The evaluations from BEELINE suggest that the area under the precision-recall curve and early precision of these algorithms are moderate. Techniques that do not require pseudotime-ordered cells are generally more accurate. Based on these results, I present recommendations to end users of GRN inference methods. BEELINE will aid the development of gene regulatory network inference algorithms.
- (iv) Based on the insights gained from BEELINE, I propose a novel graph convolutional neural network (GCN) based supervised algorithm for GRN inference from single-cell gene expression data. This GCN-based model has a considerably better accuracy than existing supervised learning algorithms for GRN inference from scRNA-seq data and can infer cell-type specific regulatory networks.

Algorithms for regulatory network inference and experiment planning in systems biology

Aditya Pratapa

(GENERAL AUDIENCE ABSTRACT)

A small number of key molecules can completely change the cell's state, for example, a stem cell differentiating into distinct types of blood cells or a healthy cell turning cancerous. How can we uncover the important cellular events that govern complex biological behavior? One approach to answering the question has been to elucidate the mechanisms by which genes and proteins control each other in a cell. These mechanisms are typically represented in the form of a gene or protein regulatory network. The resulting networks can be modeled as a system of mathematical equations, also known as a mathematical model. The advantage of such a model is that we can computationally simulate the time courses of various molecules. Moreover, we can use the model simulations to predict the effect of perturbations such as deleting one or more genes. A biologist can perform experiments to test these predictions. Subsequently, the model can be iteratively refined by reconciling any differences between the prediction and the experiment. In this thesis I present two novel solutions aimed at dramatically reducing the time and effort required for this build-simulate-test cycle. The first solution I propose is in prioritizing and planning large-scale gene perturbation experiments that can be used for validating existing models. I then focus on taking advantage of the recent advances in experimental techniques that enable us to measure gene activity at a single-cell resolution, known as scRNA-seq. This scRNA-seq data can be used to infer the interactions in gene regulatory networks. I perform a systematic evaluation of existing computational methods for building gene regulatory networks from scRNA-seq data. Based on the insights gained from this comprehensive evaluation, I propose novel algorithms that can take advantage of prior knowledge in building these regulatory networks. The results underscore the promise of my approach in identifying cell-type specific interactions. These context-specific interactions play a key role in building mathematical models to study complex cellular processes such as a developmental process that drives transitions from one cell type to another.

Dedication

To mom and dad.

Acknowledgments

Foremost, I want to express my deepest appreciation to Murali for inviting me to pursue my PhD at Virginia Tech. I consider myself very fortunate for the series of events that led to his meeting with my Masters advisor Dr. Karthik in 2014. My PhD has been a great learning experience and I cannot thank him enough for being there every step of the way. From preparing me for my first conference talk in Arizona to working even late at night on paper submissions, his patience and attention to detail are truly inspiring. He has been a great mentor and his approach of algorithmic thinking, performing easily accessible and reproducible research contributed a great deal to what I consider a successful graduate experience.

There are several people I am indebted to who have contributed in meaningful ways and have prepared me for my graduate school. I want to thank Dr. Karthik for introducing me to the world of computational biology and being my first academic mentor. My experience at IIT with Dr. Karthik is what led me to pursue a PhD. I am forever indebted to him for showing me how enjoyable and rewarding pursuing research can be. I also want to thank my amazing friends outside of Blacksburg for their support. Special thanks to Aarthi, Kartik, and Madhuri for always lending a ear and supporting me.

I also want to thank many people in Blacksburg who have made my time here at Virginia Tech a thoroughly enjoyable one. I want to thank Alan for always being there for me and helping me navigate graduate school and life in general. I would also like to thank Brittany for making me an unofficial GBCB member so I could attend their potlucks. I would like to thank the current and past members of the CSB research group for their support and feedback throughout my PhD. Special thanks to Jeff Law for patiently helping me with the many bash commands and all the useful discussions on my research. I would like to thank Mitch Wagner for introducing me to Docker. I thoroughly enjoyed working with Mitch and learned a great deal of Python and code organization, which has helped me a lot during my own research. I also want to thank Aditya Bharadwaj for his support and feedback and also introducing me to Jupyter notebooks. Special thanks to Amogh Jalihal for being a great friend and colleague throughout. I had the pleasure of working with Amogh on two of my projects. I am forever grateful for the many discussions and his help with several aspects of my research. I thoroughly enjoyed our extended coffee breaks, even though I do not particularly enjoy coffee as much as he does. I learned a great deal about mathematical modeling and yeast biology from him, which has helped me in many ways.

I owe a great deal of gratitude to my family for their infinite love and support. I could not have asked for more supportive parents. They have always worked very hard so that I could pursue my own dreams. I am also grateful to my sister Mounika, for always being there for me. My thoughts are also with my late grandfather who did not have the opportunity to submit his PhD thesis.

On a more formal note, this work would not have been possible without contributions from several people I have had the pleasure of working with during my PhD. I want to thank Murali for his inputs on every aspect of this research. I would like to thank Dr. Neil Adames for taking the time to explain experimental aspects of yeast biology, and Dr. Pavel Kraikivski for the many discussions on mathematical modeling. I also would like to thank Dr. Tyson and Dr. Peccoud for the HGTV meetings that eventually led to my first publication at Virginia Tech (Chapter 2). I would like to thank Nicholas Franzese for helping me with the NP-completeness proofs for CrossPlan (Chapter 2). I would also like to extend my gratitude to Dr. S. S. Ravi for his contributions in formulating problems for efficient synthesis of mutants (Chapter 3). I want to thank Amogh Jalihal for performing cell cycle model simulations (Chapter 3) and implementing BoolODE (Chapter 4). I want to express my gratitude to Jeff Law for implementing parameter search and Aditya Bharadwaj for implementing Borda aggregation and performing several stability analyses for algorithms in BEELINE (Chapter 4). I would like to thank my committee members Dr. Tyson, Dr. Heath, Dr. Prakash and Dr. Kececioglu for their valuable insights and feedback on my research. I would also like to thank the National Science Foundation awards CCF-1617678 and DBI-1759858 and the National Institute of General Medical Sciences of the National Institutes of Health award R01-GM095955-01 for supporting this work.

Contents

List of Figures	xi
List of Tables	xiv
1 Introduction	1
1.1 Mathematical modeling in systems biology	2
1.1.1 Mathematical modeling cycle	2
1.1.2 Scaling up the mathematical modeling cycle	4
1.1.3 Cell cycle model	6
1.2 Gene regulatory network inference	7
1.3 Organization and contributions of this thesis	8
2 CrossPlan	10
2.1 Introduction	10
2.2 Methods	12
2.2.1 Experimental workflow	12
2.2.2 Problem formulation	13
2.2.3 An ILP for CROSSPLAN	16
2.2.4 Extensions and Alternate Approaches	18
2.3 Results	18
2.3.1 Datasets	19
2.3.2 Results for CROSSPLAN	21
2.3.3 Analysis of CROSSPLAN output for $k = 12$	22
2.3.4 Analysis of mutants obtained using CROSSPLAN	23
2.3.5 Results for CROSSPLANDELAY	24
2.3.6 CROSSPLAN and CROSSPLANMARKERS with limited horizon	25

2.3.7	Comparison with manual planning	26
2.3.8	Extending CROSSPLAN to <i>Drosophila melanogaster</i>	28
2.4	Conclusions	28
3	Efficient Mutant Synthesis	31
3.1	Introduction	31
3.2	Related research	32
3.3	Definitions and problem formulations	33
3.4	Verifiable and permissible synthesis	35
3.4.1	EVS is NP-complete	35
3.4.2	An efficient algorithm for VS-2	37
3.4.3	An ILP for the DCS problem	38
3.4.4	A fixed parameter tractability result for VS	40
3.4.5	A fixed parameter tractability result for VPS	42
3.4.6	Running Time Analysis of EVPS Problem	44
3.5	Results	46
3.5.1	Evaluation on cell cycle model simulations	47
3.5.2	Evaluation on synthetic datasets	52
3.6	Conclusions	54
4	BEELINE	56
4.1	Introduction	56
4.2	Results	58
4.2.1	Overview of algorithms	58
4.2.2	Datasets from synthetic networks	58
4.3	Discussion	75
4.4	Methods	79
4.4.1	Regulatory network inference algorithms	79
4.4.2	BoolODE	81

4.4.3	Datasets	85
4.4.4	Evaluation pipeline	94
5	Supervised GRN Inference	99
5.1	Introduction	99
5.2	Methods	100
5.2.1	GCN-based autoencoders	101
5.2.2	Methods evaluated	103
5.2.3	Datasets	104
5.2.4	Evaluation	105
5.3	Implementation details	106
5.4	Results	106
5.4.1	Identifying the best GCN-based autoencoder architecture	106
5.4.2	Comparison to other supervised GRN inference methods	108
5.4.3	Application to human embryonic stem cell scRNA-seq dataset	111
5.5	Discussion	113
6	Conclusions	115
	Bibliography	117
	Appendices	132
	Appendix A CrossPlan Proofs	133
A.1	CROSSPLAN is NP-complete	133
A.2	Proof of correctness for the CROSSPLAN ILP	139
	Appendix B BEELINE Supplementary Results	147
B.1	Datasets from synthetic networks	147
B.1.1	Generation of simulated datasets	147
B.1.2	Parameter Search	147

B.1.3	Effect of number of cells on AUPRC	152
B.1.4	Fidelity of simulations from inferred GRNs	154
B.1.5	Effect of using trajectory information	156
B.2	Datasets from curated models	158
B.2.1	BoolODE simulations capture model steady states	158
B.2.2	Parameter Search	164
B.2.3	Effect of dropouts on AUPRC	164
B.2.4	Comparing similarities of algorithm outputs	165
B.2.5	Network motifs	167
B.2.6	Explaining false positives in top- k networks	168
B.3	Experimental single-Cell RNA-seq datasets	171
B.3.1	Parameter search	171
B.3.2	Correspondence of clusters in GRNs and gene expression datasets	173
B.3.3	Effect of gene selection strategy	175
B.3.4	Stability across multiple runs	177
B.3.5	Ensembles of GRNs	179
B.4	Software Details	181

List of Figures

1.1	Steps involved in mathematical modeling cycle	2
1.2	Steps involved in scaled-up mathematical modeling cycle	5
2.1	Steps involved in a genetic cross	12
2.2	Example of a genetic cross graph	13
2.3	Example of a batch	15
2.4	Example of model simulations	19
2.5	CROSSPLAN results for different number of batches planned	21
2.6	Analysis of CROSSPLAN output for $k = 12$	23
2.7	Analysis of mutants obtained using CROSSPLAN	24
2.8	Results for various extensions of CROSSPLAN	25
2.9	A set of crosses suggested by CROSSPLAN that require > 4 markers.	25
2.10	Comparison of CROSSPLAN results with manually created plan	27
2.11	Extended comparison of CROSSPLAN results with manually created plan	29
3.1	Illustration of different types of synthesis	34
3.2	Reduction of VS to DCS	37
3.3	Distribution of the number of crosses in optimal synthesis	49
3.4	Example of a mutant for which there is no verifiable permissible synthesis	50
3.5	Example of a verifiable but a non-permissible synthesis	51
3.6	VS-FL results for synthetic data	52
3.7	VPS results for synthetic data	53
4.1	BEELINE overview	57
4.2	Comparison of BoolODE and GeneNetWeaver	59
4.3	Summary of results for datasets from synthetic networks	61

4.4	Box plots of AUPRC values for synthetic networks	62
4.5	Box plots of AUROC values for synthetic networks	63
4.6	t-SNE projections of simulations for datasets from curated models	65
4.7	Summary of results for datasets from curated models	66
4.8	Box plots of AUPRC values for curated models	67
4.9	Box plots of AUROC values for curated models	68
4.10	Box plots of EPR values for curated models	69
4.11	Scalability of GRN algorithms on experimental single-cell RNA-Seq datasets	71
4.12	Summary of results for experimental single-cell RNA-seq datasets	72
4.13	Summary of results for datasets 500 and 1000 genes.	73
4.14	Summary of properties of GRN algorithms and results from BEELINE	76
4.15	Effect of pseudotime shuffling on AUPRC values for synthetic networks	78
5.1	Heatmap of median early precision values for various GCN architectures . . .	107
5.2	Boxplots showing the effect of the number of GCN layers	108
5.3	Box plots summarizing the results 10-fold edge cross-validation	109
5.4	Box plots summarizing the results 10-fold TF-holdout cross-validation	110
5.5	Violin plot summarizing results for hESC dataset	112
A.1	Input graph to Steiner tree problem	135
A.2	Input and output genetic cross graphs for a feasible solution	140
A.3	Illustration of input genetic cross graph	144
B.1	t-SNE visualizations of simulations for synthetic networks	148
B.2	Parameter search results for datasets from synthetic networks	150
B.3	Variation in AUPRC values with respect to number of cells	152
B.4	Summary of results on fidelity for datasets from synthetic networks.	156
B.5	Effect of using trajectory information	157
B.6	t-SNE visualizations of simulated trajectories from the mCAD model	160
B.7	t-SNE visualizations of simulated trajectories from the VSC model	161

B.8	t-SNE visualizations of simulated trajectories from the HSC model	162
B.9	t-SNE visualizations of simulated trajectories from the GSD model	163
B.10	Summary of parameter search results for datasets from curated models	165
B.11	Variation in AUPRC values with respect to dropout rates	166
B.12	PCA projections of edge rank vectors	167
B.13	Summary of output similarity for datasets from curated models	168
B.14	Summary of network motif analysis	170
B.15	Parameter search results for experimental single-cell RNA-seq datasets	172
B.16	Correspondence of clusters in GRNs and gene expression datasets	174
B.17	Variation in EPR and AUPRC with respect to number of genes	176
B.18	Spearman correlation results for experimental single-cell RNA-seq datasets	177
B.19	Summary of Borda aggregation results	179
B.20	Summary of Jaccard index for experimental single-cell RNA-seq datasets	180

List of Tables

1.1	Statistics on model simulations	6
2.1	Statistics on simulations.	19
2.2	Statistics on ILP sizes	26
3.1	Statistics on simulations for up-to-six gene mutations	47
3.2	List of source mutants used for evaluating proposed algorithms	48
4.1	Kinetic parameters used in BoolODE.	84
4.2	Summary of synthetic networks	86
4.3	Parameters used to generate datasets from synthetic networks	88
4.4	Summary of published Boolean models.	89
4.5	Parameters used to generate datasets from curated models	91
4.6	Statistics on experimental single-cell RNA-seq datasets	93
4.7	Statistics on networks for experimental single-cell RNA-seq datasets	94
5.1	Statistics on scRNA-seq datasets	105
B.1	Algorithm parameters used for datasets from synthetic networks	151
B.2	Statistical significance analysis results for Figure B.3	153
B.3	Correlation between simulation time and pseudotime	159
B.4	Algorithm parameters used for datasets from curated models	164
B.5	Statistical significance analysis results for Figure B.11	166
B.6	Parameter used for experimental single-cell RNA-seq datasets	171

Chapter 1

Introduction

Mathematical models of gene and protein regulatory networks are widely used to investigate cellular processes [1]. A standard approach in this field is to start by constructing the model of a specific cellular process, e.g., the cell cycle, from relevant experimental data. Subsequently, we computationally simulate the model in a new scenario, e.g., a combination of gene knockouts that has not been characterized experimentally. Next, we design and perform an experiment to validate the model’s prediction of the effect of the knockouts on the process. Finally, we modify or expand the model to reconcile any differences between the prediction and experiment, and continue this cycle. This process is usually slow and painstaking with high-quality models taking years of effort to build.

In this thesis, we present solutions to two different problems that arise in the context of this mathematical modeling cycle. The first problem arises when we attempt to scale up the cycle, i.e., to substantially decrease the times it takes to develop and improve models. In this context, we envision two main bottlenecks: a) the number of possible combinations of gene knockouts that we can simulate grows explosively, complicating the search and prioritization of informative mutants, (b) it is impossible to manually plan experiments to make and characterize thousands of mutants. In this thesis, we propose a computational framework that solves these challenges of prioritization and experiment planning, thereby significantly contributing to the acceleration of the process of mathematical modeling. The second problem we address deals with leveraging recent advances in experimental techniques that can measure gene expression at a single cell resolution [2, 3] to infer regulatory mechanisms in the form of a gene regulatory network (GRN). These GRNs can potentially serve as the wiring diagram for mathematical models aimed at building models of complex cellular behaviors [4, 5]. In this thesis, we investigate whether we can accurately infer the GRNs underlying complex cellular processes from single cell transcriptomic data.

In this chapter, we summarize the key steps in the mathematical modeling cycle and provide a broad overview of scaling-up strategies we propose in Section 1.1. In Section 1.2 we describe current approaches and challenges in GRN inference from single cell transcriptomic data and our proposed solutions. Finally, Section 1.3 describes the organization of this thesis and the contributions of this research.

1.1 Mathematical modeling in systems biology

Cellular processes include several molecules of different types (e.g., mRNA, proteins, complexes, metabolites) that interact with and regulate each other in intricate ways. As our knowledge of a process increases, it becomes increasingly challenging to reason about its dynamical properties intuitively. A mathematical model of the process has the potential to allow a scientist to study the behavior of the process precisely and quantitatively. Such a model is an abstract representation of the process that aims to describe its behavior through a set of equations.

1.1.1 Mathematical modeling cycle

Mathematical modeling involves five steps (Figure 1.1): (i) building the model of a biological process, e.g., the cell cycle, based on existing data, (ii) simulating the model to predict the phenotype under various genotypical perturbations, e.g., combinatorial knockouts of the genes in the model, (iii) prioritizing model predictions for validation, (iv) performing experiments to test these key model predictions, and (v) modifying or expanding the model to address any differences between the predictions of the simulations and new experimental data. The updated model can be the basis for new simulations, thus perpetuating this build-predict-validate-reconcile cycle of mathematical modeling. We expand on each step in the next few paragraphs.

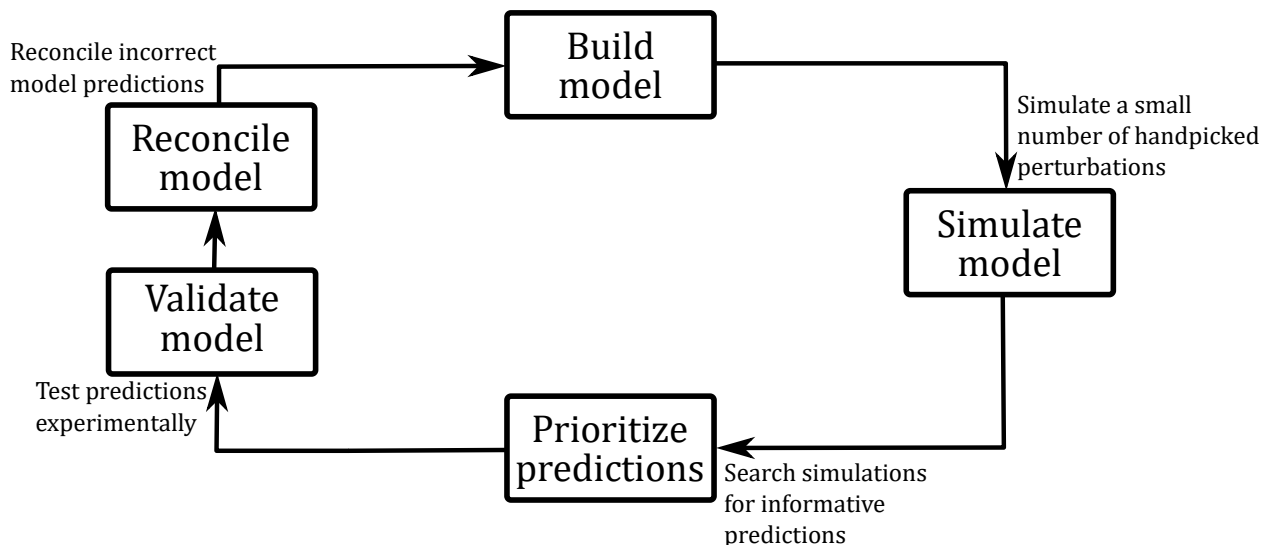


Figure 1.1: The five main steps involved in a traditional mathematical modeling cycle.

Build the model. Building a model involves making decisions about the variables in it, their relationships, and parameters [1]. In a typical mathematical model, each variable corresponds to a biological entity such as a gene, protein, or protein complex. Depending on the system we are modeling, the variables can represent gene expression levels [6, 7], protein concentrations and activity levels [8, 9, 10], or reaction fluxes [11, 12]. The relationships between various variables are often represented in terms of a wiring diagram, examples include gene regulatory networks [6, 7], protein interaction networks [8, 9, 10], and metabolic networks [13, 14]. The parameters in a model can be various kinetic parameters such as reaction or equilibrium constants [15, 16].

As mentioned earlier, one of the crucial aspects of modeling a biological system involves the representation of variables and their relationships in the system in the form of a wiring diagram. A graph is a natural way to represent the wiring diagram. In such a graph, a node represents a variable in the model and an edge represents the relationship between two variables. There are several ways to obtain one of the above mentioned representations of the biological system being modeled. For example, relationships between genes in the model can be obtained by studying the literature to obtain causal mechanisms [8, 9, 10]. Alternatively, we can also build the wiring diagram from existing databases such as KEGG [17], NetPath [18], or Reactome [19].

Once we obtain a wiring diagram, we can convert the relationships between variables into a set of equations. These equations may be Boolean functions to represent logical relationships between variables [20, 21, 22, 23], or ordinary differential equations (ODEs) to describe chemical kinetics [8, 11]. In the case of ODE-based models, the parameters in these equations are either identified from biochemical reaction databases [24, 25] or estimated as a part of global optimization that minimizes the difference between the values of model variables and their experimentally-observed values [1]. The model building process is considered complete once all the parameters have been estimated, so that the model predictions are within a reasonable degree of accuracy when compared to existing literature.

Simulate the model. Once we build the model, we can then simulate it under various extracellular or mutational conditions. Model simulations serve two main purposes. First, they can be used to evaluate how well the model can recapitulate the data it has been trained on. Second, they are useful to make new predictions by simulating the model under various perturbations that have not yet been tested experimentally. A common type of model simulation involves predicting phenotypes under certain hand-picked combinatorial gene knockouts. Studies have previously focused on a small number of selected simulations that suggest new hypotheses about the biological process being modeled [8, 9], or showcase the model's capability in correctly predicting experimental data that was not used while building the model [26].

Prioritize model predictions for experimental Validation.

Experimental data for estimating model parameters is usually sparse. Therefore, multiple models may fit the data equally well: these variants may differ in either the interactions in

the underlying wiring diagram or in their parameter sets or both. Early research focused on prioritizing predictions based on how well they promised to distinguish between Boolean network models that fit the available data equally [27, 28, 29, 30]. Under this approach, researchers use the mathematical models to first evaluate various hypothetical regulatory relationships and then use these simulations to suggest performing the most informative experiments first. Scientists have also prioritized model predictions to guide experimental design for parameterizing ODE-based models [31]. These studies have primarily focused on model predictions that guide experiments by identifying specific enzyme concentrations and time points for adding enzymes in a way that minimizes uncertainty in reaction kinetic parameter estimation [32, 33]. Other studies have focused on identifying model predictions for resolving model ambiguity [34, 35, 36, 37].

Validate the model. Model validation phase primarily involves performing experiments identified in the prioritization step. We can then compare this new experimental data to the model predictions.

Reconcile model. Experimental testing may reveal that one or more model predictions are incorrect. The modeler can then attempt to modify the model in order to reconcile differences between model predictions and the new experimental data [38, 39]. Strategies that are useful here include identifying changes to the wiring diagram, modifying the equations that govern the model, or constraining the parameter space.

1.1.2 Scaling up the mathematical modeling cycle

Section 1.1.1 describes the standard approach used in mathematical modeling (Figure 1.1). We start by constructing the model. We then simulate the model in a few handpicked scenarios. Next, we prioritize model predictions for experimental validation. Finally, we modify or expand the model to reconcile any differences between the prediction and experiment, and continue this cycle. However, this is a very slow and painstaking process.

One of the goals of this thesis is to dramatically scale up this mathematical modeling cycle. Figure 1.2 describes the steps involved in the scaled-up version of the mathematical modeling cycle. Rather than simulate the model for a small number of scenarios, we will expand the scope to predict phenotypes for all single, double, triple, and quadruple genetic perturbations, and so on. Previous studies have used *in silico* models of metabolism in budding yeast to explore the effects of multiple gene knockouts. One analysis showed that considering perturbation of genes individually in a metabolic model does not reveal at least one-third of the genes that contribute to the growth potential of this organism [40]. Other studies have used combinatorial gene knockouts to make informative predictions about biological phenomena such as epistatic interactions [41], metabolic robustness [42], and evolutionary aspects of gene duplication [43]. Researchers have also studied up-to-four gene knockouts in Boolean models of the epithelial-to-mesenchymal transition (EMT) and performed experiments with all knockout combinations that suppressed TGF- β -driven EMT [10].

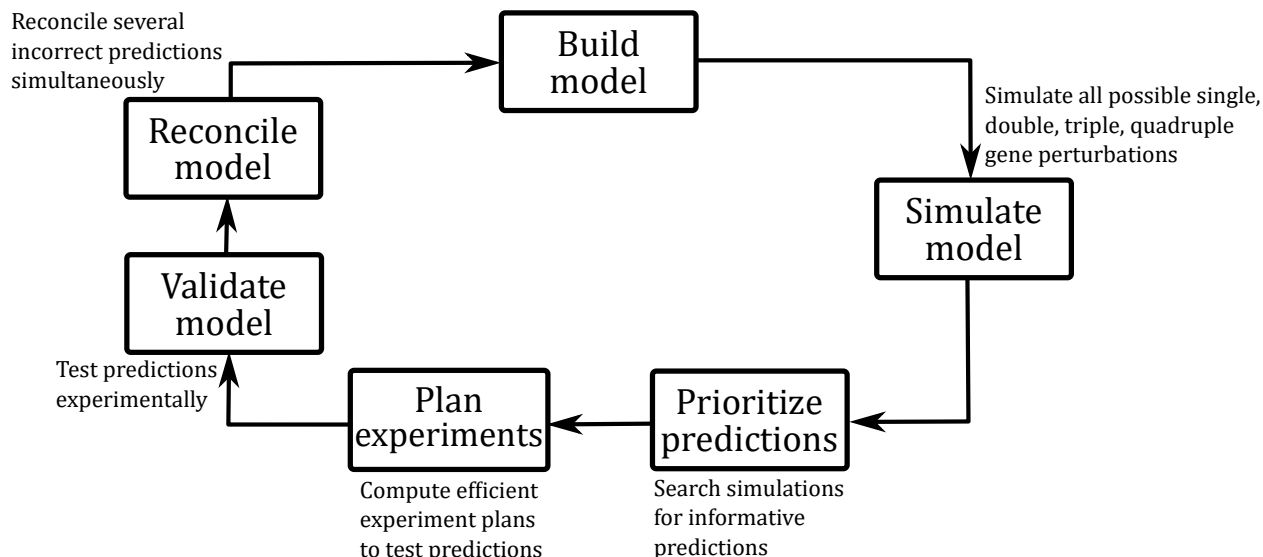


Figure 1.2: The steps in the scaled-up version of the mathematical modeling cycle.

For a model with 35 genes, simulating all up-to-six gene knockouts requires about two million¹ simulations (box titled ‘Simulate model’ in Figure 1.2). Performing experiments to test the predictions made by all these simulations is infeasible, especially due to time and budget constraints. Therefore, as mentioned in Section 1.1.1, previous research has focused mainly on prioritizing the next-best or a small number of experiments. In contrast, to truly scale-up the modeling cycle, we propose to explicitly perform all these simulations, record the predicted phenotypes, and find predictions that are potentially informative about the model. In Chapter 2, we describe how we define an informative prediction (box titled ‘Prioritize predictions’ in Figure 1.2). We anticipate that the number of such predictions that we will prioritize for experimental validation may range in the thousands.

It is possible that the predictions so prioritized may involve arbitrary combinations of double, triple, and quadruple gene knockouts. Making these mutants is experimentally challenging. For example, we cannot use high-throughput experimental screens such as Synthetic Genetic Arrays (SGA) which are appropriate for generating all double knockout strains among a set of genes [44]. Therefore, planning the sequence of experiments to make these mutants is daunting. Hence, our focus in Chapters 2 and 3 is on developing new methods for computational synthesis of experimental plans to make a large set of desired mutants. These solutions correspond to the box titled ‘Plan experiments’ in Figure 1.2. We adapt our algorithms to experimental workflows developed by our collaborators to make a large number of mutants efficiently.

Based on the experiment plans so obtained, one can perform experiments to test these

¹ $\sum_{i=1}^6 \binom{35}{i} = 2,007,327$

predictions (box titled ‘Validate Model in Figure 1.2). The final challenge we may face is that experiments may reveal that tens or even hundreds of model predictions that are incorrect. There is a need for new algorithms to modify the model to rapidly reconcile such large numbers of inconsistencies between model and experiment (box titled ‘Reconcile model’ in Figure 1.2). We did not address the model reconciliation aspect in this thesis.

1.1.3 Motivating example: Mathematical model of the budding yeast cell cycle

To ensure that the proposed solutions in Chapters 2 and 3 are practical and useful, we will develop them in the context of the highly successful model of the cell cycle control system [9]. This model addresses the detailed phenotypic properties of more than 250 yeast strains carrying mutations that interfere with the cell cycle progression in yeast. The model consists of 59 species (proteins, their modified forms, and complexes) and 60 differential and algebraic equations, involving 133 adjustable parameters (e.g., rate constants and binding constants). These 59 species correspond to 29 unique genes.

We simulated this model for mutations in all combinations of up-to-6 genes, ignoring combinations that contained redundant pairs of deletions, e.g., two different ways to knock out Cdc14. For each mutant strain, we recorded whether the simulation predicted the phenotype as “viable” or “inviable.” A strain is “viable” if simulated cells grow and divide with a stable cell size at division, and “inviable” otherwise. Table 1.1 summarizes over 1.6 million mutant simulations we performed using the cell cycle model. Of these, only 126,848 ($\approx 8\%$) of the mutants were observed to be viable. We use these model simulations as a case study for assessing applicability of proposed solutions in Chapters 2 and 3.

Number of mutations	Mutants	Viable	Inviable
1	35	26 (74%)	9
2	586	285 (49%)	301
3	6,250	1,896 (30%)	4,354
4	47,705	8,872 (18%)	38,833
5	277,531	31,060 (11%)	246,471
6	1,279,720	84,709 (7%)	1,195,011
Total	1,611,827	126,848 (8%)	1,484,979

Table 1.1: Statistics on cell cycle model simulations with up-to-6 gene mutations.

1.2 Gene regulatory network inference from single-cell transcriptomic data

Our next focus was on computational methods for building mathematical models from existing experimental data. As a first step towards achieving that goal, we focused on building the wiring diagram from gene expression data. It is now possible to measure the transcriptional states of individual cells in a sample due to advances in single-cell RNA-sequencing technology (scRNA-seq). Although these measurements are noisy, they permit several new applications. One prominent direction is clustering of cells based on shared expression patterns leading to the discovery of new cell types [2, 3]. A central question that arises now is whether we can discover the gene regulatory networks (GRNs) that control cellular differentiation into diverse cell types and drive transitions from one cell type to another. In the GRNs we consider here, each edge connects a transcription factor (TF) to a gene it regulates. Ideally, an edge is directed from the TF to the target gene, represents direct rather than indirect regulation, and has a sign denoting activation or inhibition.

Single-cell expression data are especially promising for computing GRNs because, unlike bulk transcriptomic data, they do not obscure biological signals by averaging over all the cells in a sample. However, these data have features that pose significant difficulties, e.g., substantial cellular heterogeneity [45], considerable cell-to-cell variation in sequencing depth, the high sparsity of the data caused by dropouts [46], and cell-cycle-related effects [47].

Despite these challenges, over a dozen methods have been developed or used to infer GRNs from single-cell data [5, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59]. Some of these methods were originally developed for bulk transcriptional data but have since been applied to single-cell data [48, 49, 57, 60]. Other publications have presented approaches to reverse engineer Boolean networks from single cell transcriptomic data [5, 61, 62, 63]. The criteria for evaluation and comparison of algorithms often change from one paper to another. Moreover, there are no widely-accepted ground truth datasets for assessing the accuracy of these methods. As a result, an experimentalist seeking to analyze a new dataset faces a daunting task in selecting an appropriate inference method. Which method is appropriate for my dataset? How should its parameters be tuned? What might the running time be? How can I test the accuracy of the resulting network? Given the rapid developments in this field, it is critical as well as timely to design a comprehensive evaluation framework to assess the accuracy, robustness, and efficiency of GRN inference methods based on clearly-defined benchmark datasets. However, three challenges arise when we evaluate GRN inference algorithms for single-cell RNA-seq data.

- (i) First, the “ground truth”, i.e. the network of regulatory interactions governing the dynamics of genes of interest, is usually unknown. Consequently, it is a common practice to create artificial graphs or extract subnetworks from large-scale transcriptional networks.

- (ii) Second, there is no accepted strategy to accurately simulate single-cell gene expression data from these networks. Methods such as GeneNetWeaver [64] are widely used to synthesize bulk transcriptomic data from GRNs. GeneNetWeaver has also been applied in single-cell analysis [52, 54, 55, 62, 65] but has limitations, as we demonstrate in Chapter 4.
- (iii) Third, many algorithms need temporal ordering of cells as part of the input. It is quite often the case that the single cell experiment used to generate the gene expression data does not provide this information. While pseudotime is the most popular surrogate for experimental time, there are nearly a hundred techniques available for estimating it [66] and their limitations may propagate to inaccuracies in GRN reconstruction.

In Chapter 4 we present ways to overcome these challenges and perform a comprehensive benchmarking of the current GRN inference methods. Based on our observations on the performance of current techniques, we propose a new method for GRN inference algorithm in Chapter 5.

1.3 Organization and contributions of this thesis

In this section, we report the organization of the upcoming chapters in this thesis and our main contributions.

Chapter 2: In this chapter, we introduce the novel problem of computationally synthesizing an optimal sequence of genetic cross experiments to be performed in order to produce multiple mutant strains that carry multi-gene knockouts of interest. We introduce the genetic cross graph, a useful abstraction for organizing mutants and the experiments needed to make them. We present CROSSPLAN, a novel methodology for systematically planning genetic crosses to make a set of target mutants from a set of source mutants. We base our approach on a generic experimental workflow used in performing genetic crosses in budding yeast. We prove that the CROSSPLAN problem is NP-complete. We develop an integer-linear-program (ILP) to maximize the number of target mutants that we can make under certain experimental constraints. We apply our method to the comprehensive mathematical model of the protein regulatory network controlling cell division in budding yeast described in Section 1.1.3. We also extend our solution to incorporate other experimental conditions such as a delay factor that decides the availability of a mutant and genetic markers to confirm gene deletions.

Chapter 3: In this chapter, we build upon the experiment planning problem presented in Chapter 2. We make the following three main contributions in this chapter. First, we formulate several natural problems related to efficient synthesis of a target mutant from source mutants without the need for constructing a genetic cross graph. The formulations presented in this chapter capture experimentally-useful notions of verifiability (e.g., the need to confirm that a mutant contains mutations in the desired genes) and permissibility (e.g., the requirement that no intermediate mutants in the synthesis be inviable). Second, we develop

combinatorial techniques to solve these problems. We prove that checking the existence of a verifiable, permissible synthesis is **NP**-complete in general. We complement this result with three polynomial time or fixed-parameter tractable algorithms for optimal synthesis of a target mutant for special cases of the problem that arise in practice. Third, we apply these algorithms to simulated data and to synthetic data. We use results from simulations of the mathematical model of the cell cycle to replicate realistic experimental scenarios where a biologist may be interested in creating several mutants in order to verify model predictions.

Chapter 4: In this chapter, we present a comprehensive evaluation of state-of-the-art algorithms for inferring gene regulatory networks (GRNs) from single-cell gene expression data. As the ground truth for assessing accuracy, we use synthetic networks with predictable trajectories, literature-curated Boolean models and diverse transcriptional regulatory networks. We develop a strategy to simulate single-cell transcriptional data from synthetic and Boolean networks that avoids pitfalls of previously used methods. In addition, we investigate GRN inference from five experimental single-cell RNA-seq datasets, two in human and three in mouse cells, comprising seven cell types. We use three different types of networks to evaluate the GRNs inferred from these datasets: cell-type-specific ChIPseq, non-cell-type-specific ChIPseq and functional interaction networks. We develop a systematic evaluation framework called BEELINE that incorporates 12 diverse algorithms for GRN inference. We provide an easy-to-use and uniform interface to each method in the form of a Docker image. We evaluate the GRN inference algorithms in terms of accuracy, robustness and efficiency. Based on the results, we present recommendations to users of GRN inference algorithms.

Chapter 5. In this chapter, we build upon our major observations from our comprehensive evaluation of current state-of-the-art techniques for GRN inference from scRNA-seq data. We present a novel supervised learning technique that uses GCN-based autoencoders for GRN inference. We present an evaluation scheme for supervised GRN inference that relies on node and edge withholding. We propose several encoder-decoder combinations and evaluate their accuracy in predicting GRNs across both evaluations. We investigate GRN inference from three experimental single-cell RNA-seq datasets using non-cell-type-specific ChIP-seq network as the ground-truth. We show that our approach compares favorably to existing supervised GRN inference techniques for scRNA-seq data in terms of reconstruction accuracy. We demonstrate that our proposed method can identify cell-type specific gene regulatory edges even when trained on a non-cell type specific ChIP-seq network.

Chapter 6. In this final chapter, we present concluding remarks and possible directions for future study.

Chapter 2

CrossPlan: Systematic Planning of Genetic Crosses to Validate Mathematical Models

Aditya Pratapa, Neil Adames, Pavel Kraikivski, John Tyson, Jean Peccoud, T. M. Murali (2018) CrossPlan: Systematic Planning of Genetic Crosses to Validate Mathematical Models. *Bioinformatics*. 34 (13): pp. 2237–2244, Oxford University Press.

2.1 Introduction

Mathematical models of gene and protein regulatory networks are widely used to investigate cellular processes [1]. A standard approach used in mathematical modeling is to start by constructing the model of a specific cellular process, e.g., the cell cycle, from relevant experimental data. Subsequently, we simulate the model in a new scenario, e.g., a combination of gene knockouts that has not been characterized experimentally. Next, we design and perform an experiment to validate the model’s prediction in this scenario. Finally, we modify or expand the model to reconcile any differences between the prediction and the experiment, and continue this cycle.

The motivation for this chapter is to dramatically scale up this build-simulate-test cycle. First, we simulate the model to make systematic predictions of the system’s behavior under various perturbations, e.g., all single gene, double gene, triple gene knockouts, and so on. Next, we analyze these predictions to select a subset that are likely to be the most informative about the model and the process under study. Finally, we seek to perform the experiments that can test these informative predictions. The last step can be very challenging, since planning experiments to make the genetic mutants to test these predictions becomes unmanageable and daunting, even for more than a dozen or so mutants. Therefore, we focus in this chapter on developing new algorithms that can automatically synthesize efficient experimental plans to make the desired mutants.

Previous research has focused mainly on prioritizing the next-best or a small number of experiments, e.g., to distinguish between Boolean network models that fit the available data equally [27, 28, 29, 30, 67], to estimate the kinetic parameters of an ODE-based model [31,

32, 33], or to resolve model ambiguity [34, 35, 36, 37].

The problem we are addressing has several unique characteristics that render existing techniques inapplicable. Our primary challenge is that we desire to plan a set of experiments that can test several (hundreds or thousands) promising predictions made by the model. Testing each prediction requires making a multi-gene mutant by performing genetic crosses. Additional challenges arise because we must account for many criteria that are potentially in conflict with each other: (i) there may be many sequences of crosses that can make the same mutant, (ii) a strain to be generated may be parental to multiple mutants, and (iii) some parental mutants may be known or predicted to be inviable. Further, we need techniques that can operate under cost-effective experimental workflows and efficiently utilize mutant strains that are available in the lab. Therefore, we focus on the novel problem of computationally synthesizing an optimal sequence of experiments to be performed in order to produce multiple mutant strains carrying multi-gene knock-outs of interest.

More specifically, we develop a novel methodology, `CROSSPLAN`, to compute a sequence of genetic cross experiments organized into batches such that we can perform the crosses in each batch in parallel. `CROSSPLAN` takes as input a source set S of mutants that are available in the lab, a set T of target mutants whose phenotypes we are interested in characterizing experimentally, and the number k of batches (which reflects the experimental budget). The plan computed by `CROSSPLAN` maximizes the number of target mutants that can be made from the source set in k batches.

We design an integer-linear program (ILP) that captures all the constraints we have stated earlier and can solve the problem optimally. We apply our method to a comprehensive ODE-based mathematical model of the budding yeast cell cycle [9]. We use this model to simulate the phenotypes of mutants carrying mutations in up-to-4 genes. We analyze the model predictions to identify *rescued* mutants: a mutant is rescued if the model predicts it to be viable but there is a strain with one fewer gene deletion is known or predicted to be inviable. We apply `CROSSPLAN` to generate experimental designs that can produce the maximum number of rescued mutants starting from single gene mutations.

We also consider two important extensions mandated by experimental requirements. First, we account for a delay factor that decides when a newly-synthesized mutant strain is available for subsequent crosses. Second, each gene deletion in a mutant strain carries a marker that we can use to verify that the gene has indeed been deleted in that strain. Our analysis shows that incorporating delays or the requirement that each gene mutation be associated with a unique marker has only a marginal effect on the number of planned mutants. We also suggest other time-efficient ILP formulations that are nearly optimal in terms of the number of target mutants that can be made.

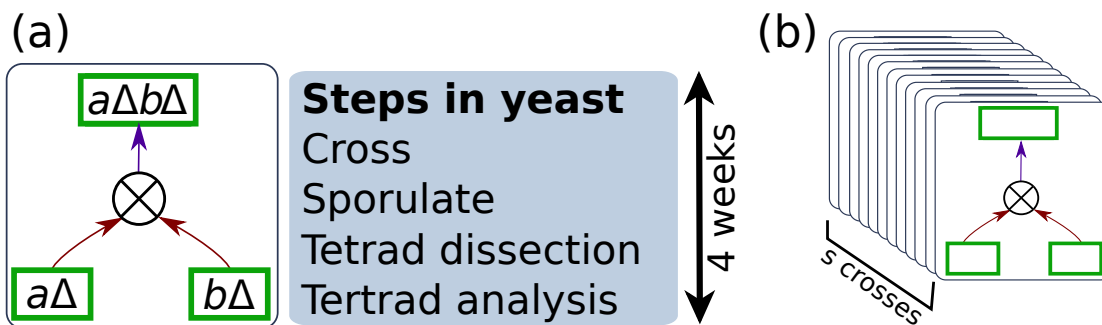


Figure 2.1: (a) Steps required to genetically cross two mutants to create new mutants. Each genetic cross requires three weeks to complete. (b) A batch of genetic crosses that can be performed in parallel. Each batch takes three weeks to complete. Costs of labour and supplies are fixed from one batch to another.

2.2 Methods

In this section, we first describe our experimental methodology (Section 2.2.1). Next, we define the CROSSPLAN problem and its variants (Section 2.2.2). Finally, we design and describe the ILPs that we use to solve these problems (Sections 2.2.3 and 2.2.4).

2.2.1 Experimental workflow

Our strategy for making mutants uses the genetic cross, a standard and widely-used technique in budding yeast and several other model organisms [68, 69, 70]. In this experiment, we start with two mutants that are already available (e.g., $a\Delta B$ and $Ab\Delta$ in Figure 2.1(a)) and cross them to produce a new mutant ($a\Delta b\Delta$ in this case). The process of crossing strains, sporulating the heterozygous diploids, performing tetrad dissection (micromanipulation of the four haploid meiotic products from each diploid), and analyzing these tetrads takes about three weeks (Figure 2.1(a)).

We base CROSSPLAN on a generic experimental workflow in which we perform multiple genetic crosses in parallel in batches. Let s be the number of genetic crosses included in each batch (Figure 2.1(b)). Each cross in a batch involves two viable mutant strains that must have either been made in an earlier batch or are already available to the experimenter (i.e., they are members of source set, S). In this workflow, the experimenter performs all s crosses and characterizes their phenotypes in parallel. In the case of budding yeast, each batch consists of a set of 12 crosses (i.e., $s = 12$); by sporulating batches of 12 crosses, we can perform tetrad analysis on all cultures before spore viability is reduced. Each batch uses approximately the same resources (supplies and labor). Thus, the number of batches we plan to perform determines the experimental cost.

With some modifications, the basis of this workflow can be applied to genetic crosses of other model organisms, for combinatorial siRNA (gene silencing) or CRISPR (genome editing) experiments, or even for testing combinations of bioactive molecules/drugs [71, 72]. For example, for genetic crosses in diploid organisms, such as *Drosophila melanogaster* or mice, we would need to add backcrosses between offspring and parents to obtain homozygous mutations (see Section 2.3.8). For combinatorial siRNA, CRISPR, or drug experiments, the workflow would be simpler, involving single-step co-introduction of RNAs, expression vectors, or drugs to obtain the desired perturbing combinations.

2.2.2 Problem formulation

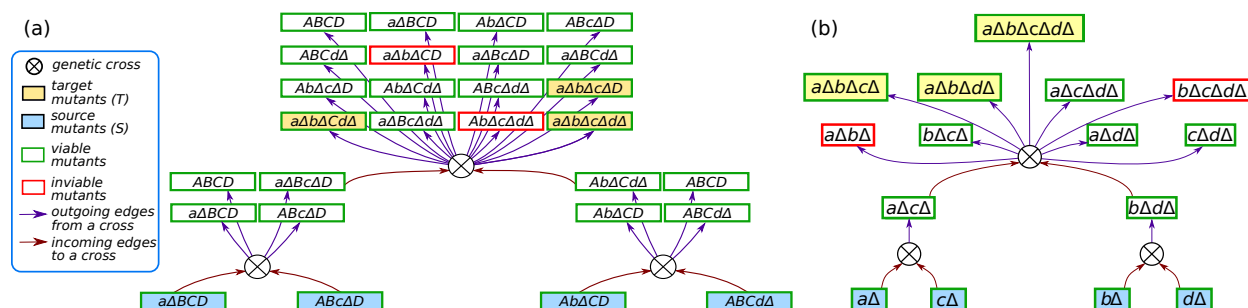


Figure 2.2: Illustration of three crosses in a genetic cross graph. Each rectangle is a mutant. Each circle is a genetic cross with two incoming edges (brown) and multiple outgoing edges (purple). (a) The figure illustrates three crosses: $a\Delta BCD$ with $ABc\Delta D$, $Ab\Delta CD$ with $ABCd\Delta$, and $a\Delta Bc\Delta D$ with $Ab\Delta Cd\Delta$. (b) Simplified version of the genetic cross graph (for illustrative purposes only). See the text for more details.

We introduce a useful abstraction for organizing mutants and the experiments needed to make them. Let G be a set of genes in an organism. A mutant m contains a combination of mutations (e.g., deletions) in multiple genes. We use $G(m)$ to denote the set of genes mutated in m . The *genetic cross graph* $\mathcal{G} = (M, X, E)$ is a bipartite directed graph where M is a set of mutant nodes, X is a set of cross nodes, and E is a set of edges. Each node in M corresponds to a mutant. Each node in X corresponds to a genetic cross experiment. Each edge e in E is either directed from a node in M to a node in X or *vice-versa*. More specifically, each node in X has two incoming edges (from nodes in M) and one or more outgoing edges (to other nodes in M). The incoming edges reflect the two mutants involved in a cross while the outgoing edges reflect all the mutants that are products of the cross. Edges incident on a cross node must satisfy two rules. If there are edges incoming to a cross node x from mutant nodes m_1 and m_2 , then

- (a) the corresponding mutants must contain distinct single gene deletions, i.e., $G(m_1) \cap G(m_2) = \emptyset$ and

- (b) there is an outgoing edge from x to every mutant in $2^{G(m_1) \cup G(m_2)}$, since due to Mendelian inheritance, the cross produces every mutant involving some combination of the genes mutated in m_1 or m_2 .

For example, Figure 2.2(a) illustrates three genetic crosses. The first cross produces $a\Delta Bc\Delta D$ from the strains $a\Delta BCD$ and $ABc\Delta D$ while the second cross produces $Ab\Delta Cd\Delta$ from $Ab\Delta CD$ and $ABCd\Delta$. Each cross also produces the original single mutants and the wild-type strain (shown as $ABCD$). The third cross combines the double mutants $a\Delta Bc\Delta D$ and $Ab\Delta Cd\Delta$ to produce one quadruple mutant $a\Delta b\Delta c\Delta d\Delta$, four triple mutants, six double mutants, four single mutants, and the wild type strain. For the sake of clarity and brevity, Figure 2.2(b) illustrates a simplified version (used for graphical purposes only) of the same set of crosses as in Figure 2.2(a). In Figure 2.2(b), only the mutated genes are shown for each of the mutants, and the only outgoing edges are from a cross to the new mutants that are produced as a result of that cross. In general, this graph represents all possible ways in which we can make new mutants from previously-made mutants. Note that we can construct a mutant using several alternative crosses, which correspond to multiple incoming edges to the corresponding mutant node. Further, we delete from the genetic cross graph any edges that leave mutants that we know are inviable from previous experimental evidence or from model predictions.

Next, we use the terminology of the genetic cross graph to define what we mean by a batch in the workflow illustrated in Figure 2.1. A *batch* is simply a set W of s cross nodes, i.e., $W \subset X$ and $|W| = s$. We will also find it useful to define the input and output mutant nodes for a batch. The *inputs* $In(W)$ to W are the mutant nodes with incoming edges to the cross nodes in W , i.e., $In(W) = \{m \mid (m, w) \in E, w \in W\}$. Informally, we need these mutants in order to perform the crosses in W . Analogously, the *outputs* $Out(W)$ of W are the mutant nodes with incoming edges from the cross nodes in W , i.e., $Out(W) = \{m \mid (w, m) \in E, w \in W\}$. Informally, these are the mutants produced by the crosses in W . Figure 2.3 illustrates the inputs, $In(W)$ and outputs, $Out(W)$ to a batch W . Note that the number of mutant nodes in $Out(W)$ may be larger than the number of cross nodes in W since each genetic cross can produce more than one mutant. Moreover, a single mutant in $Out(W)$ may be the product of more than one cross in W . With these definitions, we can now formulate the CROSSPLAN problem.

CrossPlan Problem. *Given the genetic cross graph $\mathcal{G} = (M, X, E)$, a set $S \subset M$ of source mutants, a set $T \subset M$ of target mutants, the batch size s , and the number of batches k , compute k s -batches $\{W_i, 1 \leq i \leq k\}$ such that*

1. *for each $1 \leq i \leq k$, $In(W_i) \subseteq S \cup (\bigcup_{1 \leq j \leq i-1} Out(W_j))$, and*
2. *the size of $T \cap (\bigcup_{1 \leq j \leq k} Out(W_j))$ is maximized over all possible sets of k s -batches.*

The first condition states that for the set W_i of crosses being performed in batch i , every input in $In(W_i)$ should be one of the source mutants or have been constructed in one of the

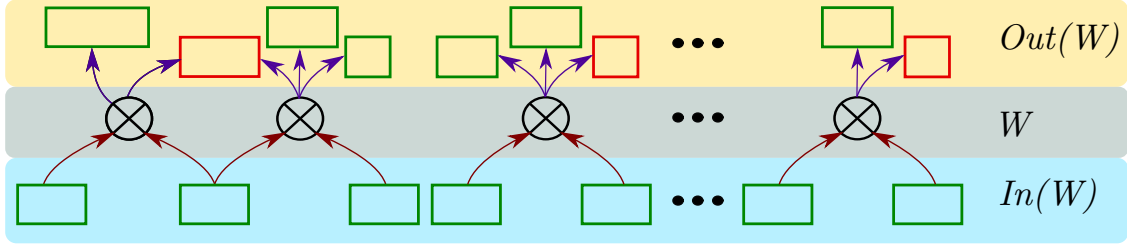


Figure 2.3: An example of a batch W . The input mutants $In(W)$ are the set of mutants required to perform the crosses in batch W ; every mutant in this set is viable (green rectangles). The output mutants $Out(W)$ are the mutants that are produced by performing crosses in the batch; the mutants here may be viable or inviable (red rectangles).

earlier batches. The second condition states that the union of the outputs of the k s -batches should contain as many target mutants as possible.

We now define two extensions of **CROSSPLAN**. The first problem supports a modification to the experimental workflow where we allow multiple batches to be executed in parallel, e.g., by starting a new batch every week. Hence, the mutants we make in a specific batch will not be available for the very next batch but only after a certain delay. We use a positive integer $d \geq 1$ to model this scenario, and require that for every batch i , the mutants made in it are available only for batch $i + j$, where $j \geq d$.

CrossPlanDelay Problem. *The definition of **CROSSPLANDelay** is exactly the same as **CROSSPLAN** itself with an additional parameter $d \geq 1$ and the following modification to the first condition:*

1. for each $1 \leq i \leq k$, $In(W_i) \subseteq S \cup (\bigcup_{1 \leq j \leq i-d} Out(W_j))$

When $d = 1$, this condition is identical to the one for **CROSSPLAN**, i.e., a mutant made in a batch is available for use in the next batch.

The second extension is specific to budding yeast and takes into consideration the markers used to verify gene deletions. Each gene deletion construct in a mutant strain must have a unique selectable marker that can confirm the deletion, e.g., $a\Delta::kanR$ represents a deletion of gene A replaced by the $kanR$ G418 resistance gene. Only strains with gene A deleted will grow in the presence of the antibiotic G418, which normally kills budding yeast cells. No two genes participating in a specific cross can share the same marker. We enforce this constraint as follows. Suppose K is the set of possible markers. We associate three sets with a mutant m : the set $P(m)$ of gene-marker pairs in m , the set $G(m)$ of genes mutated in m , and the set $K(m)$ of markers associated with these genes. We define the *genetic cross graph with markers* $\mathcal{H} = (M', X', E')$ as follows:

1. each node in M' is a subset of $G' = G \times K$ with the property that $|G(m)| = |K(m)|$, i.e., each gene in $G(m)$ has a distinct marker,

2. each node in X' is a genetic cross,
3. if a cross node in X' has two incoming edges from two nodes m and m' in M' , then
 - (a) both the genes and the markers associated with m_1 and m_2 are disjoint, i.e., $G(m_1) \cap G(m_2) = \emptyset$ and $K(m_1) \cap K(m_2) = \emptyset$, and
 - (b) the outgoing edges from the cross are to the mutants in $2^{P(m_1) \cup P(m_2)}$.

CrossPlanMarkers Problem. *The definition of CROSSPLANMARKERS is similar to CROSSPLAN except we use the genetic cross graph with markers.*

2.2.3 An ILP for CrossPlan

We can prove that CrossPlan is NP-complete (see Appendix A.1). Therefore, we solve it using an ILP. An attractive feature of the ILP is that it is easy to state and extend. The ILP contains three sets of variables.

- (i) The first set of variables records in which batch we perform a particular genetic cross experiment. Specifically, for each cross node $x \in X$, we introduce k 0-1 variables $c_{x,i}$ where $1 \leq i \leq k$. We set $c_{x,i} = 1$ if and only if we perform that cross experiment in batch i .
- (ii) The second set of variables records in which batch we make a particular mutant. Specifically, for each mutant $m \in M$, we introduce $k + 1$ 0-1 variables $b_{m,i}$ where $0 \leq i \leq k$. We set $b_{m,i} = 1$ if and only if we make mutant m in batch i . Over the course of several batches of experiments, we may make mutant m multiple times. Hence, $b_{m,i}$ may be set to 1 for multiple values of i .
- (iii) The third set of variables records if we make a target mutant m in *any* batch. For each mutant $m \in T$, we introduce one 0-1 variable a_m , which we set to 1 if and only if $b_{m,i} = 1$ for at least one value of i . We use the variables a_m to define the function we optimize below.

The ILP contains five sets of constraints.

- (i) **Source mutants constraints:** Since the mutants in the source set S are already available, we can set their (and only their) b values in batch 0 to be unity:

$$b_{m,0} = \begin{cases} 1, & \text{for all } m \in S \\ 0, & \text{for all } m \notin S \end{cases} \quad (2.1)$$

There are a total of $|M|$ such constraints, one for every mutant.

- (ii) **Batch size constraints:** We can perform at most s crosses in any batch.

$$\sum_{x \in X} c_{x,i} \leq s, \text{ for each } 1 \leq i \leq k \quad (2.2)$$

There are a total of k such constraints, one for every batch.

- (iii) **Cross input constraints:** If we perform a genetic cross x in batch i , then we must have made both the mutants being crossed in x in one of the earlier batches.

$$c_{x,i} \leq \sum_{j=0}^{i-1} b_{m,j} \text{ for every } m \text{ such that } (m,x) \in E, \quad (2.3)$$

and for every $1 \leq i \leq k$

Note that each $c_{x,i}$ variable appears in two cross input constraints, one for each of the mutants that are inputs to cross x , i.e., one constraint for each mutant m such that (m,x) is an edge in E . The value on the right hand side is zero only if the mutant m is not made in any of the batches between 0 and $i-1$. There are a total of $2k|X|$ of these constraints, two for every cross in every batch.

- (iv) **Mutant input constraints:** If we make a mutant m in batch i , then we must also perform at least one of the genetic crosses that produces m in that batch.

$$b_{m,i} \leq \sum_{\substack{x \in X \\ (x,m) \in E}} c_{x,i}, \text{ for every } 1 \leq i \leq k \quad (2.4)$$

Note that (x,m) is an edge in the genetic cross graph if and only if m is one of the mutants that are the outputs of the cross x . If the right-hand side is zero, then we cannot make mutant m in batch i . There are $k|M|$ such constraints, one for every mutant in every batch.

- (v) **Making target mutant constraints:** For a target mutant m , $a_m = 1$ only if we make m in at least one batch.

$$a_m \leq \sum_{i=0}^k b_{m,i}, \text{ for each } m \in T \quad (2.5)$$

There are $|T|$ such constraints, one for every target mutant.

In total, the ILP contains $(k|X| + (k+1)|M| + |T|)$ variables and $(2k|X| + (k+1)|M| + |T| + k)$ constraints. Our objective is to maximize the number of target mutants in T that we can make in k s -sized batches. In other words, our objective function is the following:

$$\max \sum_{m \in T} a_m \quad (2.6)$$

Note that if some $c_{x,i}$ is set to 1 in a solution to this ILP, then it is not necessary that $b_{m,i}$ is set to 1 for every mutant m such that $(x,m) \in E$, i.e., m is a product of x . The solution will set such a $b_{m,i}$ to 1 only if m is “on the path” to some target mutant l for which $a_l = 1$. Hence, in any solution to this ILP, for each $1 \leq i \leq k$, batch i is given by

$$W_i = \{x \mid c_{x,i} = 1\}$$

In Appendix A.2 we prove that maximizing this objective function under the specified constraints solves the CROSSPLAN problem correctly.

2.2.4 Extensions and Alternate Approaches

In this section, we describe how to solve `CROSSPLANDELAY` and `CROSSPLANMARKERS`. We also introduce an alternative algorithm.

ILP for CrossPlanDelay. In the `CROSSPLANDELAY` problem, we impose the requirement that for every batch i , every cross in that batch can only use mutants made in batches $i - j$, where $j \geq d$. To solve this problem, we modify eq. (2.3) in the ILP for `CROSSPLAN` as follows:

$$c_{x,i} \leq b_{m,0} + \sum_{j=1}^{i-d} b_{m,j} \text{ for every } m \text{ such that } (m,x) \in E \quad (2.7)$$

Note that we can use the mutants in the source set S in any batch irrespective of the value of d .

ILP for CrossPlanMarkers. We simply apply the ILP for `CROSSPLAN` to the genetic cross graph with markers.

Planning with Limited Horizons. Suppose we want to solve `CROSSPLAN` for k batches. In the “limited horizon” approach, we select a *horizon* $h < k$, solve the ILP for h batches, add the mutants made in the solution for h batches to S , solve the ILP for h batches again, and repeat this process until we have obtained a solution for k or more batches. In practice, this approach is likely to be faster but may be sub-optimal since we do not plan all k batches simultaneously.

2.3 Results

We start by describing how we simulate the dynamical model of the cell cycle and compute target mutants (Section 2.3.1) and how we construct the genetic cross graph (Section 2.3.1). Next, we perform a detailed analysis of `CROSSPLAN` results for these data (Section 2.3.2 and Section 2.3.3) before discussing `CROSSPLANDELAY` (Section 2.3.5). We also compare the results of `CROSSPLAN` and `CROSSPLANMARKERS` using their limited-horizon counterparts (Section 2.3.6). In Section 2.3.7, we discuss a case study where we compare `CROSSPLAN`’s results to a manually-created plan for 13 target mutants.

2.3.1 Datasets

Using the ODE model to identify target mutants

We used a dynamic model of the budding yeast cell cycle [9] for our analyses. This model addresses the detailed phenotypic properties of more than 250 yeast strains carrying mutations that interfere with the G1–S and/or the FINISH transition (metaphase-anaphase-telophase-cell division). The model consists of 59 species (proteins, their modified forms, and complexes) and 60 differential and algebraic equations, involving 133 adjustable parameters (e.g., rate constants and binding constants). These 59 species correspond to 29 unique genes. We simulated this model for mutations in all combinations of up-to-4 genes, ignoring combinations that contained redundant pairs of deletions, e.g., two different ways to knock out Cdc14. For each mutant strain, we recorded whether the simulation predicted the phenotype as “viable” or “inviable.” A strain is “viable” if simulated cells grow and divide with a stable cell size at division, and “inviable” otherwise.

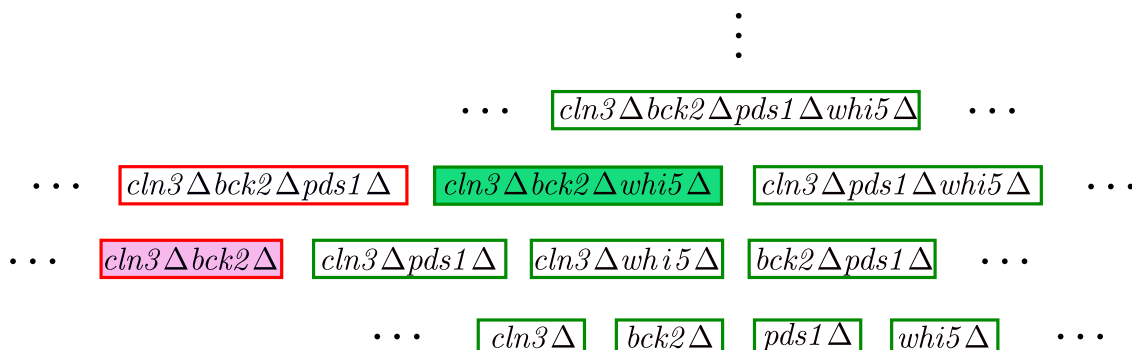


Figure 2.4: An example of rescued mutant. The mutant highlighted in green is the rescued mutant, and the mutant highlighted in pink is the inviable mutant it rescues. Note that the $whi5\Delta$ also rescues the inviable mutant $cln3\Delta bck2\Delta pds1\Delta$. However, this is a redundant rescue mutant.

We defined a mutant b to be a *parent* of mutant a if a carried one additional single gene mutation than b . We defined a mutant a to be *rescued* if a is predicted to be viable but had

#mutations	#mutants	#viable	#rescued	#rescued non-redundant
1	35	26 (74%)	-	-
2	586	285 (49%)	14	14
3	6,250	1,896 (30%)	344	170
4	47,705	8,872 (18%)	3,170	516
Total	54,576	11,079 (20%)	3,528	700

Table 2.1: Statistics on simulations.

a parent b that is either experimentally known or predicted by the cell cycle model to be inviable, e.g., $cln3\Delta bck2\Delta$ is inviable and $cln3\Delta bck2\Delta whi5\Delta$ is rescued (see Figure 2.4). We noted that a rescued mutant may be *redundant*. For example, $whi5\Delta$ rescues the inviable mutants $cln3\Delta bck2\Delta$ and $cln3\Delta bck2\Delta pds1\Delta$. Since $pds1\Delta$ is itself a viable strain, the loss of Pds1 is irrelevant to the rescue phenotype. Hence, we considered only non-redundant rescued mutants. Rescued mutants are highly informative because converting an inviable strain to a viable strain usually occurs only by combining mutations in genes that have opposite functions within the same essential biochemical network. Thus, the prediction of new rescue mutants can assist in generating hypotheses about gene function and the architecture of the regulatory network.

Table 2.1 summarizes our results. Only 20% of all up-to-four gene deletions of the 54,576 mutants we simulated were viable. The percentage of viable combinations decreased with an increase in the number of genes deleted. Of these viable mutants, the number of rescued ones were 3,528, of which 700 (nearly 20%) were non-redundant. We designated all non-redundant rescued mutants as the target set T . For the source set S , we used 35 strains that have been studied in the literature carrying mutations in one of the genes in the model. Note that this number is greater than the number of genes in the model, since some genes have been mutated in different ways.

Constructing the genetic cross graph

To construct the genetic cross graph \mathcal{G} input to the CROSSPLAN problem, we needed to define the set G of single gene mutations, the set M of mutant nodes, the set X of cross nodes, and the input and output edges for each cross in X . We set G to be the set of 35 single gene deletions. Each mutant node in M corresponded to one of the gene mutation combinations we simulated with the cell cycle model (Table 2.1). For every pair of mutants m_1 and m_2 such that $G(m_1)$ and $G(m_2)$ are disjoint, we created a cross node x in X . We added edges from m_1 and from m_2 to x and from x to each of the mutants corresponding to the power set of $G(m_1) \cup G(m_2)$.

We further pruned \mathcal{G} as follows: Consider the set $G(T)$ of genes that are deleted in at least one target mutant, i.e., $G(T) = \bigcup_{l \in T} G(l)$. We deleted every mutant m that contained at least one gene not present in $G(T)$. An alternative way of phrasing this step is that we retained a mutant m in T if and only if (the set of genes mutated in) m was a subset of $G(T)$. We also deleted any cross nodes that were disconnected as a result. After these modifications, \mathcal{G} contained 2,064 mutant nodes, 4,958 cross nodes, and 59,040 edges.

For the CROSSPLANMARKERS problem, we constructed the genetic cross graph with markers \mathcal{H} as follows. Recall that K is the set of markers. We used a set of four markers (hph, kan, nat, and ura). For every mutant $m \in M$ carrying mutations in one gene, we created $|K|$ copies of m , pairing it with each marker in turn. In general, for every mutant m , we created $\binom{|K|}{|G(m)|} |G(m)|!$ copies of m , one for each of the ways to assign distinct markers to the genes

mutated in m . For example, for single gene mutants, we created $\binom{|K|}{1}1! = |K|$ copies for each mutant, one for each marker. We created cross nodes as we did for \mathcal{G} but with the added condition that the two mutants being crossed should not share any markers. We pruned this graph as described above and retained 43,648 mutant nodes, 112,720 cross nodes, and 1,383,192 edges.

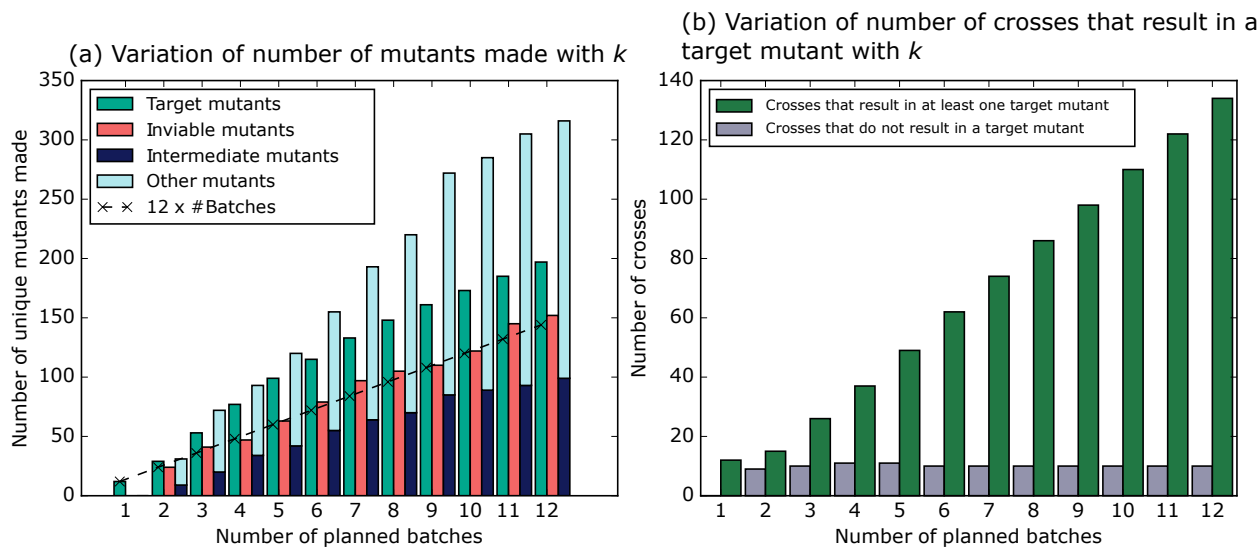


Figure 2.5: CROSSPLAN results for different number of batches planned.

2.3.2 Results for CrossPlan

First, we investigated the number of target mutants we could plan with $s = 12$ (we provided a rationale for this value in Section 2.2.1) and with different values of k . In the solution for each value of k , we identified four sets of mutants made across all the batches: (i) target mutants, (ii) inviable mutants that are rescued by one of the target mutants made, (iii) (viable) intermediate mutants that we make in order to produce a target mutant, and (iv) other mutants that result from the crosses but are not used in any cross in any batch. Note that any cross that produces a target mutant m (which must be viable by our construction of T) will also automatically produce all inviable parents that m rescues. In constructing these sets, we ignored the source mutants in S and counted each other mutant only once.

Figure 2.5(a) summarizes these results for $1 \leq k \leq 12$. For every value of k , we succeeded in planning exactly sk crosses (the dashed line in Figure 2.5(a)). When $k = 1$, we only planned 12 double mutants, all of which rescued inviable single gene deletions. For $k > 1$, we planned about 1.4 times as many target mutants (green bars in Figure 2.5(a)) as the number of crosses. For every value of k , the number of inviable mutants (red bars in Figure 2.5(a)) rescued by the target mutants we planned was almost the same as the number of crosses.

We also noted that the number of intermediate mutants (dark blue bars in Figure 2.5(a)) we needed to produce was approximately 0.4 times the number of target mutants. The figure also displays other mutants that we planned (light blue bars); these mutants result from the planned crosses but are not used to make any other mutants. Figure 2.2(b) illustrates why the number of target mutants may be larger the number of crosses: when we cross $a\Delta c\Delta$ with $b\Delta d\Delta$, we simultaneously produce the target mutants $a\Delta b\Delta c\Delta$, $a\Delta b\Delta d\Delta$, and $a\Delta b\Delta c\Delta d\Delta$.

The trends in Figure 2.5(a) suggest that most crosses yield at least one target mutant (e.g., the cross between $a\Delta c\Delta$ and $b\Delta d\Delta$ in Figure 2.2(b)) and very few crosses produce only intermediate mutants (e.g., the cross between $a\Delta$ and $c\Delta$ in Figure 2.2(b)). To investigate this further, we counted the number of crosses of each type for each value of k . Figure 2.5(b) shows that when $k = 1$, all 12 crosses we plan result in target mutants (green bars). In contrast, for $2 \leq k \leq 12$, at most 11 crosses did not yield any target mutants (gray bars). For $k = 12$, we further analyzed each cross in the computed plan. We observed that not only did the majority of crosses produce at least one target mutant, but the entire plan depended on finding an appropriate set of double gene mutants to make in batch one. Complete details of this analysis are in Section 2.3.3.

Finally, we investigated how many more batches we could plan if we started with the CROSSPLAN solution for k batches. Accordingly, we solved a simpler version of CROSSPLAN: we solved the ILP for k batches, added all the mutants in the solution to S , set $s = \infty$ (i.e., made the batch size unlimited), and then re-solved the ILP for one batch. Our analysis showed that starting from the CROSSPLAN solution for 12 batches, we could make as many as 315 additional target mutants simply by crossing pairs of mutants in S , a number enough to occupy 26 batches of size 12. Additional details of this analysis are in Section 2.3.4.

2.3.3 Analysis of CrossPlan output for $k = 12$

In this section, we discuss an extended analysis of the CROSSPLAN result for $k = 12$. We divided the crosses planned into two groups: crosses that resulted in at least one target mutant and crosses that produced only non-target mutants. Further, for each cross x , we recorded two batch indices: i_x , the batch in which the plan used x and j_x , the earliest batch in which x could have been used (i.e., the plan had created both mutants that were input to x in batch $j_x - 1$ or earlier). Note that $1 \leq j_x \leq i_x \leq k = 12$.

We observed that only 10 crosses yielded only non-target mutants (shown using black borders in Figure 2.6). Notably, we planned all these crosses in the very first batch, suggesting that the entire plan depended on finding an appropriate set of double gene mutants to make in batch one. In Figure 2.6, we also plot the value $i_x - j_x$ for each cross x made in batch i . The darker the color, the larger the value of $i_x - j_x$, i.e., the earlier we could have performed cross x . Figure 2.6 shows that many of the crosses planned in batches indices 9–12 could have been performed much earlier were it not for the restriction on the size of a batch.

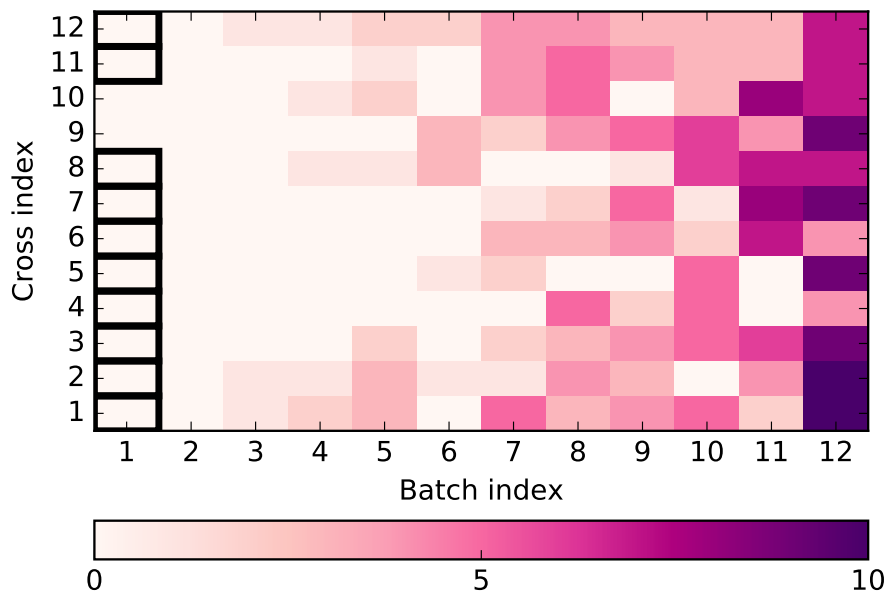


Figure 2.6: For the solution for $k = 12$, a heatmap showing the difference between the batch index in which CROSSPLAN performed a cross and the earliest batch in which it could have performed that cross. Each column corresponds to a batch in the solution for $k = 12$. Every cell in a column corresponds to a cross in that batch (ordered arbitrarily).

2.3.4 Analysis of mutants obtained using CrossPlan

The previous analysis suggests that many crosses could have been performed earlier if not for the restriction of the number of crosses per batch. Therefore, we investigated how many more target mutants we could make by merely crossing mutants obtained in the CROSSPLAN solution for k batches and without imposing any restriction on the number of crosses per batch. While there can be multiple optimal solutions to the CROSSPLAN problem, we based this analysis on one such solution that we computed for each value of k . Accordingly, we first solved CROSSPLAN for k batches where k ranged between 1 and 12. For each value of k , we then set S to be the set of mutants made in the computed plan for k batches, and solved another CROSSPLAN problem with $s = \infty$ (i.e., an unlimited batch size) and for one batch. By design, solving this new ILP will give us the maximum number of mutants that we can make just by crossing mutants obtained in the solution for k batches. We named the target mutants so obtained as “free target mutants”, in the sense that these mutants are readily available by performing a single cross. We also computed the number of batches these crosses would require, assuming 12 crosses per batch, and called them “free batches”.

Figures 2.7(a) and 2.7(b) summarize the results of this analysis. For $k = 1$, CROSSPLAN made 12 target mutants, which yielded only two free target mutants. All 14 target mutants were double mutants. The number of free target mutants increased substantially thereafter. For $k = 2$, where CROSSPLAN made 29 target mutants (see fig. 2.5), we could create 82

free target mutants, enough to occupy 7 free batches. The number of free target mutants increased linearly with k until $k = 9$. For $k = 9$, where CROSSPLAN made 161 target mutants (see fig. 2.5), we could create 327 free target mutants enough to occupy 28 batches. Thereafter, the number of free mutants hit a plateau. This analysis demonstrates that a viable approach to solving CROSSPLAN for k batches may be to solve it instead for a smaller number k' of batches and then check if the number of free target mutants suffices to use the remaining $k - k'$ batches.

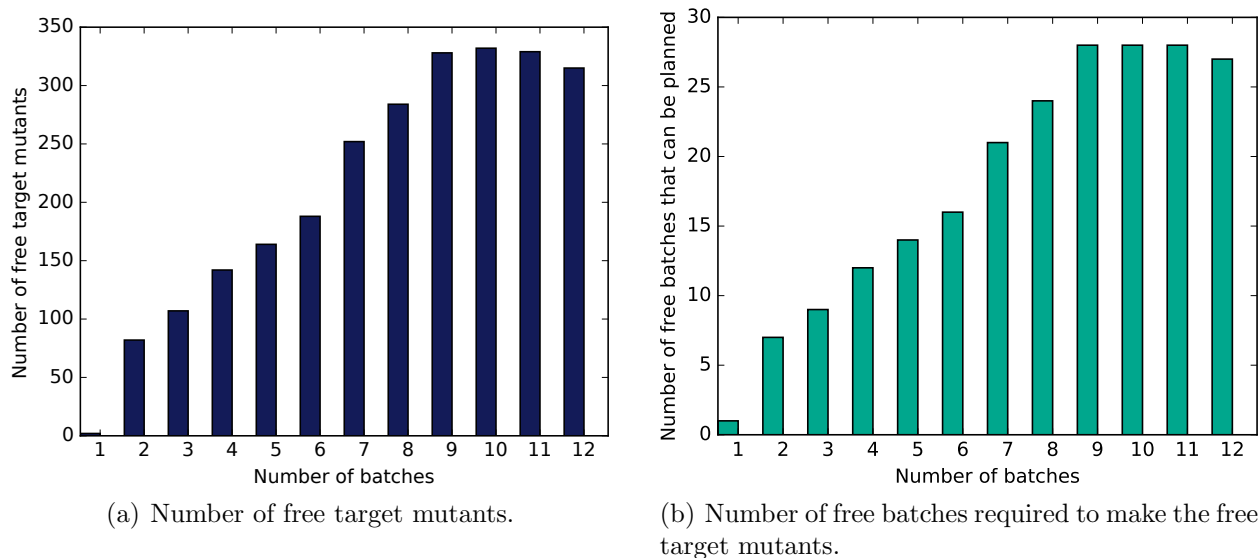


Figure 2.7: Analysis of target mutants that we can make simply by crossing the mutants obtained in the CROSSPLAN solution for k batches, $1 \leq k \leq 12$.

2.3.5 Results for CrossPlanDelay

We solved the ILP for CROSSPLANDELAY for $s = 12$, the same values of k , i.e., $1 \leq k \leq 12$ and for different values of the delay d . Figure 2.8(a) summarizes our results. Here we compare the number of target mutants planned for different numbers of batches as we varied d from 1 to 5; the plot labeled “Original” corresponds to $d = 1$. For any value of k , increasing the delay by unity decreased the number of planned target mutants by 12 on average, which is the size of a batch. Even with $d = 5$, with 12 batches, we could plan 149 target mutants, which is more than 75% of 197 mutants we could plan with $d = 1$ in the same number of batches. The most striking difference occurred for batches $1 \leq k \leq d$ (nearly horizontal lines in Figure 2.8(a)). There are 14 target mutants in T that are double gene deletions. Since all single gene deletions are present in S , for all values of d , we planned 12 of these double gene deletions in the first batch. When $d > 1$, we could plan only the last two double gene deletions in the second batch. We had to wait till $k > d$ to plan any other target mutants.

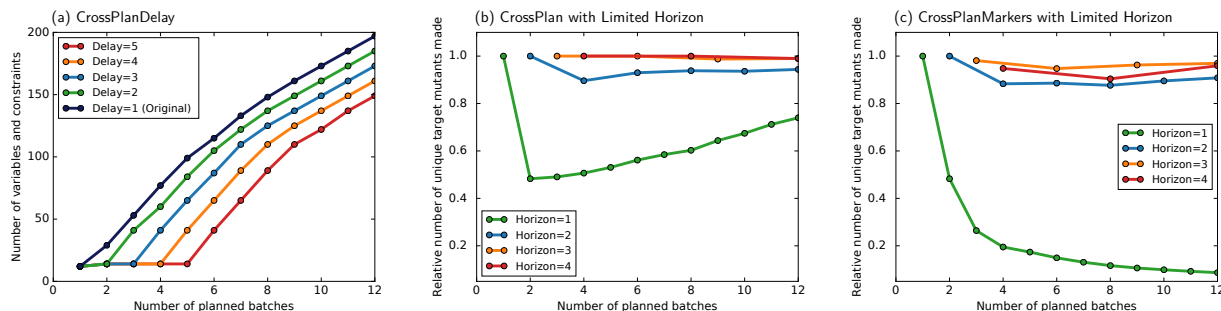


Figure 2.8: (a) Variation of number of mutants made with k for CROSSPLANDELAY. (b) Variation with k of the ratio of the number of target mutants planned by the limited-horizon strategy and the number planned by CROSSPLAN. (c) Variation with k of the ratio of the number of target mutants planned by the CROSSPLANMARKERS with limited-horizon strategy and the number planned by CROSSPLAN.

2.3.6 CrossPlan and CrossPlanMarkers with limited horizon

We compared the performance of planning experiments for all k batches together with solutions obtained for the limited horizon approach with $1 \leq h \leq 4$. Figure 2.8(b) displays the ratio of the total number of unique target mutants we could plan with different horizons with the number planned with an unlimited horizon (i.e., CROSSPLAN). Clearly, when we plan experiments only for one batch at a time $h = 1$, we obtain very inefficient solutions. The most striking difference occurs for $k \leq 4$. Surprisingly, the best solutions for $h > 1$ are quite close to the results for CROSSPLAN, with $h = 3, 4$ being virtually indistinguishable from CROSSPLAN.

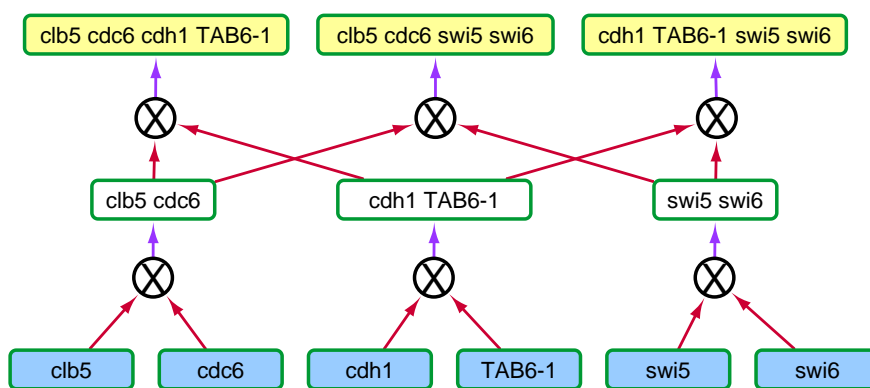


Figure 2.9: A set of crosses suggested by CROSSPLAN that require > 4 markers.

When we attempted to solve CROSSPLANMARKERS with $k \geq 5$, we found that ILP sizes were too large, causing the solvers (CPLEX and Gurobi) to use up all the available RAM (32GB). Rather than use a computer with more RAM, we used the limited horizon approach to solve CROSSPLANMARKERS as well. In Figure 2.8(c), for different values of k , we compare

k	#Variables	#Constraints	Time (in min.)
CROSSPLAN			
1	9,744	14,703	0.01
4	30,810	50,646	10.21
12	86,986	146,494	58.34
CROSSPLAN, $h = 4$			
4	30,810	50,646	10.21
12	30,810	50,646	24.63

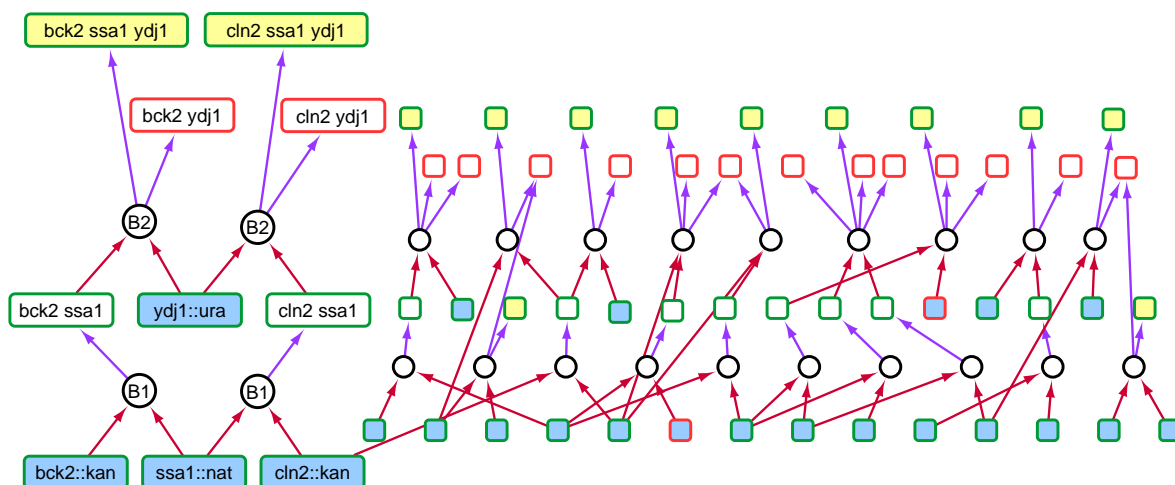
Table 2.2: Statistics on ILP sizes and running times for CROSSPLAN and CROSSPLAN with Limited Horizon.

the number of target mutants planned by this version of CROSSPLANMARKERS to the number planned by CROSSPLAN (with no limit on the horizon). Remarkably, other than for $h = 1$, we could plan at least 85% as many target mutants as CROSSPLAN for all other values of h . However, for $h = 3, 4$, CROSSPLAN succeeded in planning more target mutants than CROSSPLANMARKERS (compare orange and red curves between Figure 2.8(b) and Figure 2.8(c)). The primary reason is that CROSSPLAN is not constrained by the requirement that each gene in a cross be associated with a unique marker. For example, for $k = 3$, CROSSPLAN made 53 target mutants, which was one more than CROSSPLANMARKERS. Figure 2.9 illustrates the point with a set of crosses suggested by CROSSPLAN that require more than four markers or an additional cross.

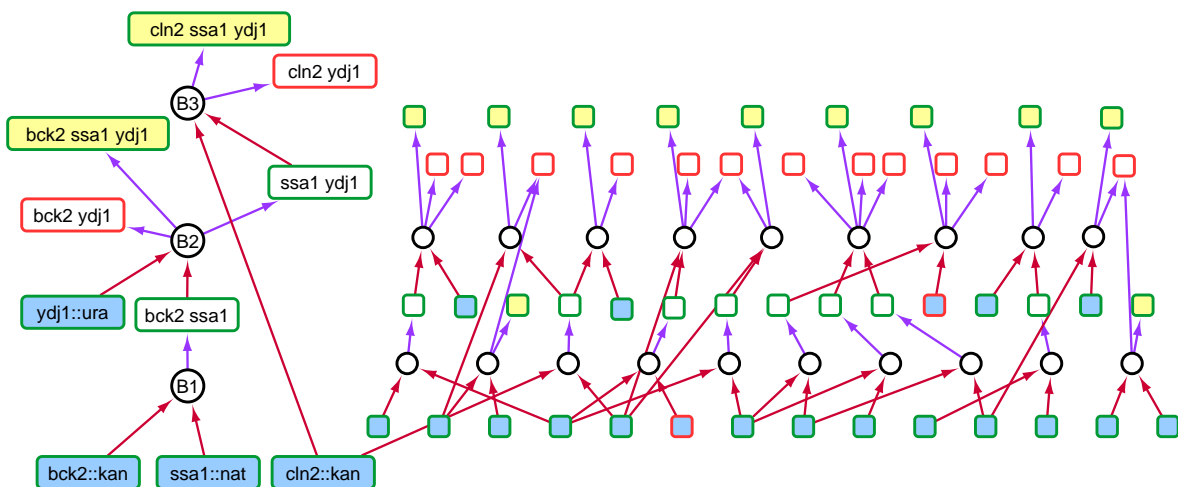
An advantage of the limited-horizon approach is that the sizes of the ILPs are much smaller than for CROSSPLAN (Table 2.2). Additionally, the time taken to solve the planning problem with a limited horizon is much lower than for CROSSPLAN. For example, it takes 58 minutes to solve the ILP for CROSSPLAN for 12 batches, where as it takes only 24 minutes to solve three problems with $h = 4$.

2.3.7 Comparison with manual planning

We manually searched for testable mutant rescue combinations that contained gene deletions known to alter the concentration of cell cycle activators/inhibitors. Moreover, these gene deletions should occur in some known and previously characterized double (or more) mutants that were used to parameterize the model. Therefore, we have high confidence in these model predictions, which are quite intuitive/logical, and include inviable combinations of cell cycle activators that could be counteracted by further removal of cell cycle inhibitors to rescue lethality. For example, *Ssa1* and *Ydj1* (Figure 2.10) have opposite effects on the G1 cyclin *Cln3* (inhibitor and activator, respectively). The model predicts that *cln2Δ ydj1Δ* mutants are inviable because the activity of the G1 cyclins (*Cln2* and *Cln3*) is too low to allow cell cycle progression from G1 to S phase. However, by removing *Ssa1*, *Cln3* activity is elevated enough to support cell cycle progression. We found 13 such target mutants.



(a) Result from CROSSPLAN.



(b) Manually-created solution.

Figure 2.10: Comparison of CROSSPLAN’s results and a manual solution for 13 target mutants. The terms “B1,” “B2,” and “B3” indicate that we plan a cross in batches one, two, and three, respectively. We only display a product of a cross if it is a target mutant (yellow with green border), an inviable parent of a target mutant (red border), or a viable mutant used in a subsequent cross. We indicate the marker for each gene in a starting mutant (blue).

Here, we study this set of 13 target mutants and compare two approaches to plan them: one computed by the ILP for CROSSPLAN and the other manually derived by one of the authors who is an expert yeast geneticist (Adames). For the set of starting mutants, we used 35 single gene mutants with four markers each, and three different double mutants, of which one had two different markers sets. Figure 2.10(a) and 2.10(b) compare the two plans. In these figures, we show mutant names only where the plans differ and illustrate the full plans in fig. 2.11. We see that the CROSSPLAN ILP could plan all target mutants in two batches of

twelve crosses each whereas the manual solution required three batches, with the third batch required to plan the twelfth target mutant. The key differences appear on the left-hand side of the figures, which depict how the two plans produce the mutant $cln2\Delta ssa1\Delta ydj1\Delta$. To produce this mutant, CROSSPLAN crosses $ssa1\Delta$ with $cln2\Delta$ in the first batch and then crosses $cln2\Delta ssa1\Delta$ with $ydj1\Delta$ in the second batch. In contrast, the manual solution does not produce any double mutant that is useful for $cln2\Delta ssa1\Delta ydj1\Delta$ in batch one. Instead, it crosses $ssa1\Delta ydj1\Delta$ (from batch two) with $cln2\Delta$ to produce the desired mutant only in batch three. All the other crosses are identical. Another advantage of CROSSPLAN is that we could automatically compute a total of 17 different ways in which to plan these target mutants in two batches. Note that some other expert might create a plan identical to the one generated by CROSSPLAN. Our aim in this analysis was to show that there are clear advantages of using CROSSPLAN over manually planning many genetic cross experiments.

2.3.8 Extending CrossPlan to *Drosophila melanogaster*

To show how we can use CROSSPLAN for combinatorial genetics in other model organisms, we present an example of a series of crosses in *Drosophila* designed to obtain homozygous mutants in two loci starting with heterozygous single loss-of-function mutants. The homozygous single mutants both show a similar mild embryonic defect while the double mutant is expected to show synthetic embryonic lethality.

In the first cross (F0), the two parents are AABb and AaBB. The resulting offspring are AABB, AABb, AaBB, and AbBb, and all are viable. In the second cross (F1), the AbBb siblings are crossed to yield AABB, AABb, AaBB, AaBb, aaBB, aaBb, AAbb, Aabb, and aabb. If the two mutations result in embryonic lethality, we expect to see no adults with the aabb genotype. It is also possible that haploinsufficiency of one locus is sufficient for embryonic lethality when combined with the other locus as a homozygous null. In this case, we might obtain no offspring with the aabb and aaBb genotypes (b is haploinsufficient when a is homozygous), or aabb and Aabb genotypes (a is haploinsufficient when b is homozygous).

In summary, we can replace a single cross to obtain the necessary mutant in budding yeast with two crosses (F0 and F1) to get aabb from AABb and AaBB in *Drosophila*. With this change, the batch-based workflow and the CROSSPLAN algorithm are also useful for performing high-throughput genetics in diploid model organisms such as *Drosophila*.

2.4 Conclusions

We have introduced the novel problem of efficiently synthesizing experimental plans to produce a desired set of mutant strains. Our methodology uses a generic experimental workflow in which we can make mutants using genetic crosses that are organized into batches. We develop an integer linear program to optimally solve the problem of constructing the largest

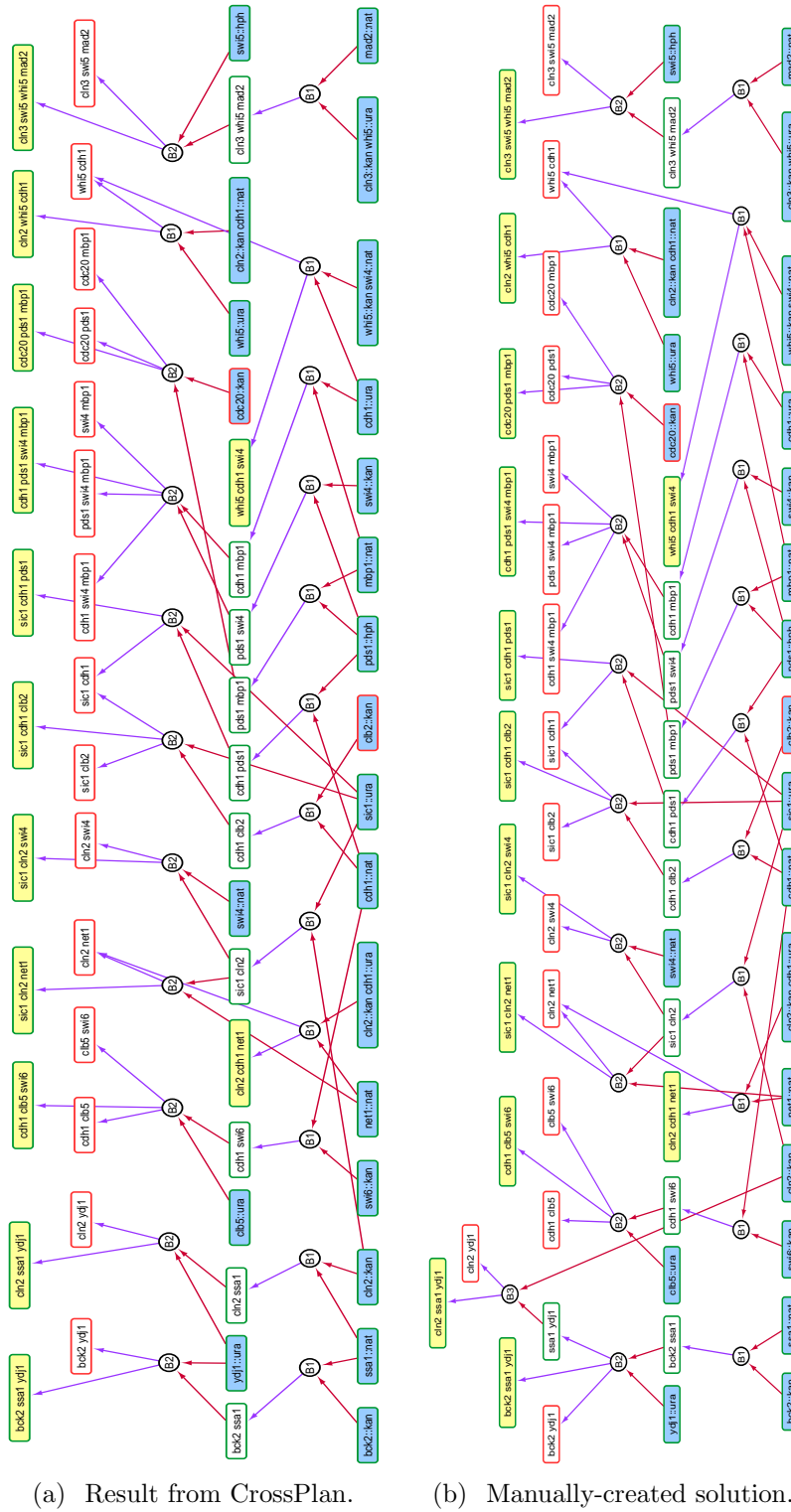


Figure 2.11: The complete genetic cross graphs corresponding to CrossPlan’s result and a manual solution for 13 target mutants.

possible number of target mutants given the batch size and number as inputs. Our results show the effectiveness of our approach. We were easily able to generalize our method to handle natural experimental constraints such as delays and genetic markers.

There are several interesting avenues for future research. The size of the genetic cross graph can be exponential in the maximum number of mutations in a single target mutant. Since CROSSPLAN took the genetic cross graph as an input, the resulting ILPs were very large, e.g., the ILP for CROSSPLANMARKERS contained 780K variables and 11M constraints. We would like to design more compact ILPs or develop alternative ways of formulating and solving the problem that do not directly use the genetic cross graph. Since the problem is NP-complete, we would also like to ascertain the fixed parameter tractability of the problem, e.g., develop polynomial-time algorithms if k or s is fixed.

We also plan to apply this methodology to other organisms where siRNA or CRISPR-based screens are effective. Our method does not rely only predictions from ODE-based mathematical models. It can be applied to Boolean models [10] and to other approaches that can predict the phenotypes of combinatorial perturbations [73]. Ultimately, we seek to complete the entire modeling cycle by performing the experiments that we plan, studying the discrepancies with model predictions, and using them to improve and extend the mathematical models themselves.

Chapter 3

Efficient Synthesis of Mutants Using Genetic Crosses

Aditya Pratapa, Amogh Jalihal, S.S. Ravi, T.M. Murali (2018) Efficient Synthesis of Mutants Using Genetic Crosses. *In Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*. pp. 53-62.

3.1 Introduction

Engineering mutations in multiple genes is a powerful way to dissect cellular mechanisms, discover pathway structure, and understand the genetic basis of complex diseases. The genetic cross is a classical technique to create strains with mutations (e.g., knock-outs or overexpression) in multiple genes, especially in model organisms such as baker's yeast, the fruit fly, mice, and *Arabidopsis thaliana*. Traditionally, biologists have created new strains by crossing existing mutants that they already have in the lab or can purchase easily. Strains carrying mutations in four, six, and even as many as 11 genes have been made in this manner [39, 74, 75]. More recently, developments in genome editing tools such as CRISPR-Cas9 have made it feasible to create mutations at multiple sites in a single experiment. This field is advancing very rapidly, with CRISPR-Cas9-like systems becoming available in several model organisms and with the development of new methods for multiplex gene editing [76].

Despite these advances, the genetic cross continues to remain a fundamentally useful experimental tool, including in combination with genome editing [77]. Since CRISPR-Cas9 edits require considerable efforts for verification, crosses are preferable if the mutants are already readily available. Moreover, in model organisms that have short generation times, performing a cross is often quicker than trying to edit two wild-type genes using CRISPR-based techniques. Therefore, in conjunction with high-throughput experimental techniques [44], genetic crosses open up the possibility of routine construction and characterization of mutants carrying alterations in several genes.

These observations inspire our research. We envision a scenario where a biologist has access in the lab to several source mutants, each carrying mutations in one or more genes; these mutants may have been created using a variety of techniques including crosses and genome editing. The biologist seeks to synthesize a target mutant using the genetic cross as the

primary experimental tool.

Our contributions. Our primary contributions are the definition of problems related to efficient synthesis of a target mutant from a set of source mutants using genetic crosses (Section 3.3) and combinatorial algorithms to solve these problems (Section 3.4). We first prove that when each source mutant contains mutations in at most three genes the problem of determining if there exists a synthesis of a target mutant is **NP**-complete. Next, we present a polynomial time algorithm to minimize the size of the synthesis (the number of crosses) of a target mutant when each source mutant contains mutations in at most two genes (Section 3.4.2). Furthermore, we provide a fixed-parameter tractable (FPT) algorithm for this problem with respect to the number of source mutants with three or more gene mutations (Section 3.4.4). Our final result deals with checking the existence of a permissible synthesis of a target mutant, i.e., ensuring that every intermediate mutant in the synthesis belongs to a permissible set of mutants. This problem is also **NP**-complete, in general. We provide a FPT algorithm with respect to the number of mutated genes in the target mutant (Section 3.4.5). This algorithm also minimizes the number of crosses.

These problems match an experimentalist’s growing repertoire of mutants. Initially, when there are only single gene mutants, the synthesis problem is trivial. Our first algorithm is useful when the experimentalist has made some double gene mutants. As triple mutants become available, the second algorithm is applicable, since it is FPT in the number of such mutants. When the set of available mutants increases further, the issue of permissibility becomes important, which our final algorithm addresses.

Two challenges confront us when we evaluate our algorithms: there is a dearth of prior literature on this problem (see Section 3.2) and there are no databases describing multi-gene (three or more) mutant strains that have been made in the lab. We use two complementary strategies to circumvent these gaps. First, we create a rich dataset of rescued and synthetic lethal target mutants by simulating a mathematical model of the cell cycle [9] for mutations in up-to-six genes (Section 3.5.1). We use mutants predicted to be viable by this model as permissible mutants. We apply our algorithms to compute syntheses for four-, five-, and six-gene target mutants. Permissibility has a material impact on a small fraction of mutants by either precluding the existence of a synthesis or by increasing the number of crosses needed in optimal syntheses. Second, we create synthetic datasets to assess the running times of our algorithms as functions of the fixed parameters (Section 3.5.2). Our results on these datasets show that the running time does grow with an increase in the value of the relevant fixed parameter.

3.2 Related research

Mathematical models of cellular processes permit *in silico* simulations of multiple gene mutations [9, 10, 40, 41, 43]. Research on designing experiments based on model predictions has

focused mainly on prioritizing the next-best or a small number of experiments, e.g., to distinguish between Boolean network models that fit the available data equally [27, 28, 29, 30, 67], to estimate the kinetic parameters of an ODE-based model [31, 32, 33], or to resolve model ambiguity [34, 35, 36, 37]. However, simulations of mathematical models or searching for informative experiments do not solve the problem of how to efficiently synthesize a strain carrying multiple mutations from existing strains.

The only closely related research of which we are aware appeared in Chapter 2. There, we considered the problem of optimizing the number of crosses needed in a verifiable, permissible synthesis of multiple target mutants when the input also contains a “genetic cross graph.” This graph represents all possible ways of crossing source and permissible mutants. Thus, the worst case size of this graph can be quadratic in the number of source and permissible mutants. We proved that the problem was **NP**-complete and developed an integer-linear program (ILP) to solve it, but did not formally analyze the running time of the ILP. In contrast, in the current chapter, we formulate problems that are more natural since they avoid the specification of the genetic cross graph in the input and the concomitant quadratic increase in the input size. We also provide fixed-parameter tractable algorithms in natural parameters. In practice, we show that our algorithm for computing an optimal permissible synthesis is over 28 times faster than our previous approach (Section 3.5.2).

3.3 Definitions and problem formulations

A *mutant* μ is equivalent to a set of genes, each of which has been mutated in some manner experimentally, e.g., a deletion (knock-out), disruption, point mutation, over-expression, etc. We will use the term *representative set* to denote the set of genes deleted in a mutant. An element of such a set represents the mutated gene as well as any other necessary experimental information, e.g., the name of the selectable marker introduced with a gene deletion (in budding yeast; see below) or the recessive deleterious mutations and balancers that accompany a mutation (in the fruit fly).

A *cross* is an operation that takes two mutants as input and produces another mutant as output. If μ_1 and μ_2 are the inputs to a cross with the corresponding representative sets R_1 and R_2 , then the output is a mutant whose representative set is $R_1 \cup R_2$. In formulating this definition, we assume that an experimentalist can select for this mutant from other output genotypes of a cross using selectable markers or other strategies such as recessive mutations and balancers. While this definition abstracts away several biological issues, it cleanly captures the essential features of a cross and also leads to challenging computational problems.

We say that a cross is *verifiable* if the representative sets of the two input mutants to the cross are disjoint. In many organisms, e.g., budding yeast, each gene deletion construct in a mutant strain must have a unique selectable marker that can confirm the deletion, e.g., $a\Delta::kanR$

represents a deletion of gene A replaced by the $kanR$ G418 resistance gene. Only strains with gene A deleted will grow in the presence of the antibiotic G418, which normally kills budding yeast cells. The notion of verifiability of a cross captures the natural requirement that when we cross two mutants, each gene in a representative set is (implicitly) associated with a marker that is not associated with any other gene in that or the other mutant.

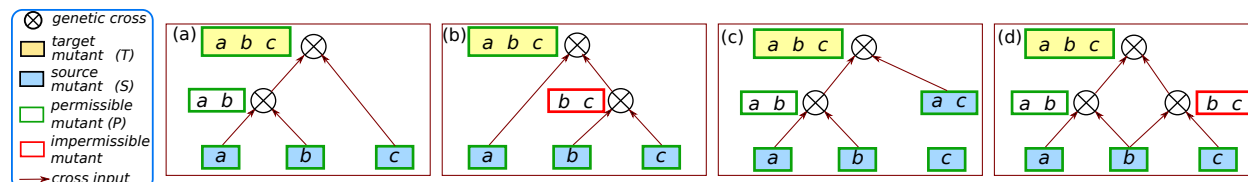


Figure 3.1: Illustration of different types of synthesis. Each rectangle denotes a mutant. Each figure shows a synthesis of $\{a, b, c\}$ from source mutants (blue). The rectangle next to a cross denotes the output mutant of that cross. (a) A verifiable, permissible synthesis. (b) A verifiable synthesis that is not permissible because the intermediate mutant $\{b, c\}$ is not permissible. (c) A permissible synthesis that is not verifiable since the two input mutants to the second cross both have gene a in their representative sets. (d) A synthesis that is neither verifiable nor permissible.

Given a set of source mutants S and a set T of target mutants, we say that a sequence of crosses $\langle x_1, x_2, \dots, x_k \rangle$ is a *synthesis of T from S* if (i) for every mutant $\mu \in T$, there is a unique cross x_i such that μ is the output of x_i , (ii) both the inputs to x_1 are from S and (iii) for every i , $2 \leq i \leq k$, each input to x_i is either a member of S or an output of one of the crosses x_1, x_2, \dots, x_{i-1} .

The *size* of a synthesis is the number of crosses it contains. We say that a synthesis is *verifiable* if every cross in it is verifiable. In such a synthesis, we say that a mutant is *intermediate* if it is not in $S \cup T$ and is used as the input to another cross in the synthesis. Given a set P of mutants, we say that a synthesis is *permissible with respect to P* if every intermediate mutant in the synthesis is in P . The notion of permissibility takes inspiration from applications where we are interested in computing syntheses that avoid particular intermediate mutants, e.g., when we know that mutants involving a certain gene are certain to be lethal or when mathematical models predict that some mutants are likely to be inviable. Without loss of generality, we assume that a synthesis is *minimal*, i.e., for every cross x_i , $1 \leq i \leq k$, where the output of x_i is not a member of T , this output is used as an input to a subsequent cross x_j , $i < j \leq k$. Figure 3.1 illustrates several of these definitions.

We now formulate the main problem that we study followed by several special cases:

Minimum Verifiable and Permissible Synthesis (VPS): Given a set S of source mutants, a target mutant μ , and a set P of intermediate mutants, compute a verifiable synthesis of μ of smallest size from S that is permissible with respect to P , provided such a synthesis exists.

VS The inputs to this problem are the source set S and the target mutant μ . We seek to compute a smallest synthesis of μ starting from S . In other words, this problem is identical to VPS except that all mutants are permissible.

VS-2 This version of VS assumes that the representative set of each mutant in S contains at most two elements. This problem simulates a scenario where an experimentalist has several single-gene mutants and has used them to make double-gene mutants as well.

VS-FL This version of VS assumes that a fixed number q of mutants in S whose representative sets are of size three or more (i.e., q is a constant independent of the problem size). The name is an abbreviation for “VS with a Few Large representative sets.” Here, the experimentalist also has access to a small number of mutants with mutations in three or more genes.

We are also interested in determining whether a synthesis with the appropriate properties exists. We will use **EVS** to denote the existence version of VS.

3.4 Verifiable and permissible synthesis

We first consider the VS problem and its variants. In Section 3.4.1, we show that EVS is **NP**-complete even when each source mutant has at most three deleted genes. It follows that EVPS is **NP**-complete and both VS and VPS are **NP**-hard, in general. Next, we show that VS-2 can be solved in polynomial time (Section 3.4.2). Using this algorithm as a sub-routine, we establish that VS-FL is FPT, where the parameter is the number of source mutants with three or more deleted genes (Section 3.4.4). Finally, we develop a FPT dynamic programming algorithm for VPS, where the parameter is the largest size of a representative set of the target mutant (Section 3.4.5).

We assume that all the problems take as input a starting set S of s mutants, specified by representative sets R_1, R_2, \dots, R_s and a target mutant μ' with the representative set R' , where $r = |R'|$. We further assume that $\bigcup_{i=1}^s R_i = R'$ since any mutant with a representative set containing a gene that is not in R' may be safely removed from S . For the VPS problem, we assume that P is the set of permissible mutants.

3.4.1 EVS is NP-complete

Existence of a Verifiable Synthesis (EVS)

Instance: A set $S = \{\mu_1, \mu_2, \dots, \mu_s\}$ of s source mutants, the representative set R_i for each mutant μ_i , $1 \leq i \leq s$; the representative set R' for a target mutant μ' .

Question: Is there a verifiable synthesis of μ' from the given set S of mutants?

To prove the **NP**-completeness of EVS, we use a reduction from the following problem.

Exact Cover by 3-Sets (X3C)

Instance: A ground set $Y = \{y_1, y_2, \dots, y_{3n}\}$ of $3n$ elements for some positive integer n , a collection $Z = \{Z_1, Z_2, \dots, Z_m\}$, where $Z_j \subseteq Y$ and $|Z_j| = 3$, $1 \leq j \leq m$.

Question: Is there a subcollection $Z' \subseteq Z$ such that $|Z'| = n$ and $\bigcup_{z \in Z'} z = Y$?

It is well known that X3C is **NP**-complete [78]. Note that in the specification of X3C, the ground set Y has $3n$ elements, the required subcollection Z' is of size n and each subset in Z' has exactly three elements. Therefore, when there is a solution Z' to an X3C instance, the subsets in Z' are pairwise disjoint.

Theorem 1. *EVS is **NP**-complete even when the representative set for each source mutant has at most three elements.*

Proof: It is easy to see that EVS is in **NP** since one can guess a subset S' of S and verify in polynomial time that the sets in S' are pairwise disjoint and that their union is equal to S' , the representative set of the target mutant μ' .

To prove **NP**-hardness, we use a reduction from X3C. Let an instance of X3C be specified by the set Y (with $3n$ elements) and the collection Z of m 3-element subsets of Y . We construct an instance of EVS as follows.

1. The set $S = \{\mu_1, \mu_2, \dots, \mu_m\}$ of available mutants is in one-to-one correspondence with the collection of sets Z of the X3C instance. For mutant μ_j , the representative set is Z_j , $1 \leq j \leq m$. Thus, the number of source mutants is m and the representative set for each input mutant has exactly three elements.
2. The representative set S' for the target mutant μ' is Y .

It is easy to see that this construction can be carried out in polynomial time.

Suppose there is a solution $Z' = \{Z_{i_1}, Z_{i_2}, \dots, Z_{i_n}\}$ to the X3C instance. A verifiable synthesis of μ' consisting of the sequence $\langle x_1, x_2, \dots, x_{n-1} \rangle$ of crosses is as follows:

- (a) The inputs to x_1 are mutants μ_{i_1} and μ_{i_2} .
- (b) For each j , $2 \leq j \leq n-1$, the inputs to x_j are the mutant $\mu_{i_{j+1}}$ and the output of x_{j-1} .

Since the union of the sets in Z' is equal to Y , the output of x_{n-1} is the mutant μ' . Further, since the sets in Z' are pairwise disjoint, this is a verifiable synthesis. In other words, we have a solution to the EVS instance.

Suppose we have a solution to the EVS instance. Let $\mu_{i_1}, \mu_{i_2}, \dots, \mu_{i_r}$ denote the source mutants used in the synthesis and let $Z_{i_1}, Z_{i_2}, \dots, Z_{i_r}$ denote their respective representative sets. Thus, $\bigcup_{j=1}^r Z_{i_j} = Z'$. Also, since we have a verifiable synthesis of μ' , each pair of representative sets used in the synthesis is disjoint. Further, each representative set has exactly three elements and their union is the set Z' , where $|Z'| = 3n$. Thus, the number of representative sets used must be exactly n . In other words, the sets $Z_{i_1}, Z_{i_2}, \dots, Z_{i_n}$ constitute a solution to the X3C instance, and this completes the proof of Theorem 1. ■

3.4.2 An efficient algorithm for VS-2

Here, we consider VS-2, a restricted version of EVS, where the representative set of each input mutant has *at most two* elements. We show that VS-2 can be solved efficiently, thus providing a clear delineation between the NP-hard and efficiently solvable versions of EVS. Our polynomial time algorithm for VS-2 relies on a known algorithm for the following problem.

Degree Constrained Subgraph (DCS): Given an undirected graph $G(V, E)$ with $|V| = n$, and non-negative integers a_i and b_i for each node $v_i \in V$, $1 \leq i \leq n$, is there a subgraph $G'(V, E')$ of G , where $E' \subseteq E$, such that for each node v_i , $1 \leq i \leq n$, the degree of v_i in G' , denoted by $d'(v_i)$, satisfies the condition $a_i \leq d'(v_i) \leq b_i$?

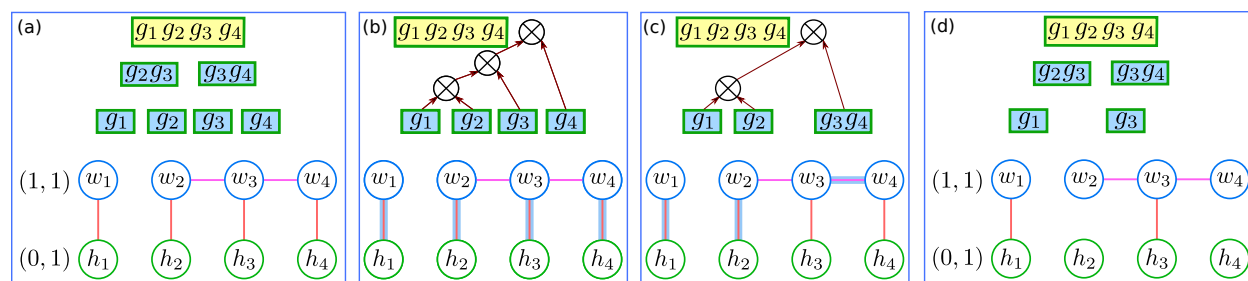


Figure 3.2: Reduction of VS to DCS. (a) The top show the input to VS with six source mutants (blue boxes) and one target mutant (yellow box). The bottom shows the corresponding DCS instance. Numbers in parentheses indicate bounds on degrees. (b) Feasible solution to DCS (edges with a blue outline). This solution is equivalent to three crosses to make the target mutant (g_1, g_2, g_4, g_4) . (c) An optimal solution to DCS that yields two crosses to make (g_1, g_2, g_4, g_4) . (d) An instance of VS that does not have a solution. Such an input may arise when single gene deletions of g_2 and g_4 make the cell inviable but the double deletions (g_2, g_3) and (g_3, g_4) are obtained via transformation.

Algorithm 1 An algorithm for VS-2.

Input: A collection S of s source mutants and a target mutant μ' . Assume that the representative sets for S are divided into two groups: R_1, R_2, \dots, R_q are singleton sets while $R_{q+1}, R_{q+2}, \dots, R_s$ are doubleton sets, and the value of q may be zero.

Output: A verifiable synthesis (if it exists) of the smallest size of the target mutant μ' .

Steps of the algorithm:

1. Construct an instance of DCS as follows.
 - (a) Node set $V = V_1 \cup V_2$: Let $R' = \{g_1, g_2, \dots, g_r\}$. Node set $V_1 = \{w_1, w_2, \dots, w_r\}$ is in one-to-one correspondence with the elements of R' . Node set $V_2 = \{h_1, h_2, \dots, h_q\}$ contains a special node for each singleton representative set R_i , $1 \leq i \leq q$. (Thus, $|V_1| = |R'|$ and $|V_2| = q$.)
 - (b) Edge set E : For each doubleton set $R_y = \{g_i, g_j\}$, E contains the edge $e_y = \{w_i, w_j\}$ between the nodes corresponding to g_i and g_j . For each singleton set $R_y = \{g_j\}$, E contains the edge $\{w_j, h_y\}$ between the node corresponding to g_j and the special node h_y corresponding to R_y . (Thus, $|E| = s$.)
 - (c) Degree bounds: For each node $w_i \in V_1$, set both the lower bound a_i and the upper bound b_i on the degree of w_i to 1. For each node $h_i \in V_2$, set the lower bound a_i and the upper bound b_i on the degree of h_i to 0 and 1 respectively.
 2. Use the algorithm from [79] to compute a subgraph with the smallest number of edges that is a solution to this DCS instance. If there is no solution, output “No” and stop. Otherwise, let the solution $E' = \{e_1, e_2, \dots, e_\ell\}$ consist of ℓ edges.
 3. Define the mutants for synthesizing μ' from E' as follows. For each edge $e_j \in E'$, if e_j joins two nodes in V_1 , choose the mutant corresponding to the doubleton set R_j represented by e_j . If e_j joins a node w_i in V_1 to a node $h_y \in V_2$, choose the mutant corresponding to the singleton set represented by e_j . Let $\mu_1, \mu_2, \dots, \mu_\ell$ denote the mutants chosen in this step.
 4. The synthesis of μ' is the sequence of $\ell - 1$ crosses $\langle x_1, x_2, \dots, x_{\ell-1} \rangle$, where the inputs to x_1 are μ_1 and μ_2 , and for each j , $2 \leq j \leq \ell - 1$, the inputs to x_j are the output of x_{j-1} and the mutant μ_{j+1} .
-

3.4.3 An ILP for the DCS problem

In Section 3.4.2 we showed that VS-2 can be solved in polynomial time by a reduction to the DCS problem. Since our problem instances are reasonably small, in our experiments, we solved the DCS problem using a simple $\{0, 1\}$ integer linear programming (ILP) formulation developed below.

Consider a DCS problem instance specified by a graph $G(V, E)$, where $V = \{v_1, v_2, \dots, v_n\}$ and $E = \{e_1, e_2, \dots, e_r\}$. Let the lower and upper bounds on the degree of node v_i be a_i and b_i respectively; that is, in the subgraph $G'(V, E')$ to be found, degree of v_i must satisfy the condition $a_i \leq \text{degree of } v_i \leq b_i$, $1 \leq i \leq n$. We also want to minimize the number of edges in E' .

The ILP has a $\{0, 1\}$ variable x_i corresponding to each edge e_i , $1 \leq i \leq r$, with the following

interpretation: if $x_i = 1$, edge e_i is included in the solution; otherwise, e_i is not in the solution. So, the objective of the ILP is:

$$\text{Minimize } \sum_{i=1}^r x_i$$

For each node v_i , there are two constraints, corresponding to the upper and lower bounds on the degree of v_i in G' . Let $Q_i \subseteq E$ be set of edges of G which are incident on node v_i . The constraints for node v_i are:

$$\sum_{e_j \in Q_i} x_j \geq a_i \quad \text{and} \quad \sum_{e_j \in Q_i} x_j \leq b_i$$

Thus, we get a total of $2n$ constraints. For any node v_i whose the lower bound a_i is 0, we omit the corresponding constraint.

The DCS problem can be solved in time $O(|E|\sqrt{\sum_{i=1}^n b_i})$ [80]. Moreover, this algorithm can be modified to compute a subgraph with the fewest edges that satisfies the conditions of the problem with the same running time [79]. We obtain our result for VS-2 by reducing it to DCS and using the algorithm for DCS in [79]. The details of our algorithm for VS-2 appear in Algorithm 1. See Figure 3.2 for an illustration.

Theorem 2. *Algorithm 1 solves the VS-2 problem in $O(s\sqrt{r+s})$ time using $O(r+s)$ space, where s is number of source mutants and r is the size of the representative set of the target mutant.*

Proof: We establish the theorem by proving the correctness of Algorithm 1 and estimating its running time. We will refer to the steps of this algorithm throughout this proof.

Correctness: We first consider the case where the algorithm produces a solution and prove that the representative sets of the mutants in the solution are pairwise disjoint and their union is the set R' . Let $\mu_{i_1}, \mu_{i_2}, \dots, \mu_{i_\ell}$ be the mutants chosen in the solution; their respective representative sets are $R_{i_1}, R_{i_2}, \dots, R_{i_\ell}$. Suppose there is a pair of representative sets in the solution, say R_{i_y} and R_{i_z} , which are not disjoint. Let g_p be an element that appears in both R_{i_y} and R_{i_z} . Since each representative set represents an edge in the DCS instance, it follows that the degree of the node $w_p \in V_1$ corresponding to g_p is at least two. This contradicts the requirement that the upper bound on the degree of each node in V_1 is one. Thus, the representative sets in the solution are indeed pairwise disjoint. Now suppose that $\bigcup_{j=1}^{\ell} R_{i_j} \neq R'$. Thus, some element $g_q \in R'$ does not appear in any of the sets in the solution. In other words, in the solution to the DCS instance, the degree of node $w_q \in V_1$ corresponding to x_q is zero. This contradicts the requirement that the lower bound on the degree of each node in V_1 is one. Thus, whenever the algorithm produces a solution, the chosen mutants can be used to construct a verifiable synthesis of the required mutant μ' .

Now, suppose the algorithm outputs “No”. We show by contradiction that there is no solution to the VS-2 instance. Assume to the contrary that there is indeed a solution with mutants $\mu_{i_1}, \mu_{i_2}, \dots, \mu_{i_\ell}$ with corresponding representative sets $R_{i_1}, R_{i_2}, \dots, R_{i_\ell}$. Since

each of these sets is represented by an edge in the DCS instance, let $E' = \{e_{i_1}, e_{i_2}, \dots, e_{i_\ell}\}$ represent the corresponding edges in G . We argue that the subgraph $G'(V, E')$ is a solution to the DCS instance by considering each node in V . There are two cases depending on whether the node is in V_1 or in V_2 .

Case 1: Consider any node $w_p \in V_1$. We need to show that the degree of w_p in G' is 1. To see this, note that node w_p appears in some edge of E' since $\bigcup_{j=1}^{\ell} R_{i_j} \neq R'$. Thus, the degree of w_p in G' is at least 1. If degree of w_p is ≥ 2 , then the corresponding element x_p appears in two or more sets of the solution; this contradicts the assumption that the solution sets are pairwise disjoint. Thus, the degree of w_p is 1.

Case 2: Consider any node $h_q \in V_2$. The degree of h_q in G' is at most 1 since each node h_q in V_2 has a degree of 1 in G .

Thus, we contradict the assumption that the algorithm outputs “No”. This argument completes the proof of correctness.

Analysis of running time and space used: The constructed DCS instance has $r + q$ nodes (where $r = |R'|$ and $q \leq s$ is the number of singleton representative sets of source mutants) and s edges. Thus, we can construct the graph in $O(r + s)$ time. The upper bound on the degree of each node is 1. Thus, we can obtain the optimal solution to the DCS instance in $O(s\sqrt{r + s})$ time [79, 80]. Since the solution produced by the algorithm for DCS has at most s edges, we can find the representative sets in the solution in $O(s)$ time. Thus, the overall running time of the algorithm is $O(s\sqrt{r + s})$. Further, since the size of each representative set of a source mutant is at most two and that of the target mutant is r , the space used by the algorithm is $O(r + s)$. This discussion completes the proof of the theorem. ■

3.4.4 A fixed parameter tractability result for VS

Here, we consider VS-FL, another restricted version of VS, where the number q of mutants whose representative sets are of size 3 or more is *fixed* (i.e., a constant independent of the problem size). We assume that $q > 0$; otherwise, we have the VS-2 problem which can be solved in polynomial time (Section 3.4.2). We show that VS-FL can be solved in $O(q2^q r(s + \sqrt{r + s}))$ time using $O(rs)$ space, where $r = |R'|$ and s is the number of source mutants. More importantly, this result points out that VS-FL is *fixed parameter tractable*, with respect to the parameter q [81], since the component of the running time that is independent of q is polynomial in the input size.

The principle underlying our algorithm for VS-FL is simple. Let L denote the collection of representative sets (of source mutants) whose size is at least three. We use q to denote the fixed size of L . We consider all the possible 2^q subcollections (including the empty subcollection) of L . For each such subcollection L' , the union of the sets in L' contributes

Algorithm 2 An algorithm for VS-FL.

Input: A collection S of s source mutants and a target mutant μ' . Assume that the representative sets for S are divided into two groups: R_1, R_2, \dots, R_q are the sets with three or more elements, while the other sets $R_{q+1}, R_{q+2}, \dots, R_s$ have at most two elements.

Output: A verifiable synthesis (if it exists) of the target mutant μ' .

Steps of the algorithm:

1. Let $L = \{R_1, R_2, \dots, R_q\}$ and $B = \{R_{q+1}, \dots, R_s\}$.
 2. For each subcollection L' of L , if the sets in L' are pairwise disjoint then
 - (i) Let A be the set obtained by the union of the sets in L' and let $R'' = R' - A$.
 - (ii) Let B' denote the subcollection of B such that the intersection between any set in B' and any set in L' is empty. The set collection B' and the set R'' constitute an instance of VS-2.
 - (iii) Use Algorithm 1 to check whether the VS-2 instance constructed in Step 2(ii) has a solution. If it does, output a synthesis of μ' using the mutants returned in Step 4 of Algorithm 1 for VS-2 along with the mutants corresponding to the sets in L' and stop.
 3. (At this point, all subcollections of L have been tried.) Output “No”.
-

some of the elements of the representative set R' of the target mutant μ' . The remaining elements of R' must be synthesized using mutants whose representative sets are of size at most two, giving rise to an instance of the VS-2 problem, which can be solved using the algorithm of Section 3.4.2. The details of our method appear in Algorithm 2. Note that this algorithm determines if a verifiable synthesis of μ' exists. It is easy to modify it to compute a verifiable synthesis of smallest size: In Step 2(iii), instead of stopping when we find a synthesis, we record the size of the synthesis and return a smallest synthesis computed over all executions of this step.

Theorem 3. *Algorithm 2 solves the VS-FL problem in $O(q2^q s(r + \sqrt{r+s}))$ time using $O(rs)$ space. Thus, the VS-FL problem is fixed parameter tractable in q , the number of representative sets of input mutants with three or more elements.*

Proof: We prove the theorem by establishing the correctness of Algorithm 2 and then showing that its running time has the appropriate form to imply the fixed parameter tractability of VS-FL.

Correctness: It is obvious that when the algorithm returns a solution, the representative sets in the solution are pairwise disjoint and their union is equal to R' , the representative set of the target mutant μ' . So, we need only consider the case when the algorithm outputs “No”. For the sake of contradiction, assume that the algorithm outputs “No”, yet there is a solution to the VS-FL instance. Suppose the solution uses a subcollection L' of L (where L' may be empty) and subcollection B' of $B = \{R_{q+1}, \dots, R_s\}$. Since the algorithm considers all subcollections of L (including the empty subcollection), it considers L' at some stage.

The sets in the solution are pairwise disjoint and the union of the sets in $B' \cup L'$ is equal to R' . Let the union of the sets in L' be denoted by A . Thus, the union of the sets in B' must be equal to $R'' = R' - A$. Note that each set in B' has size at most two. Hence, the subcollection B' and set R'' constitute an instance of VS-2 for which a solution is B' . Therefore, the algorithm for VS-2 cannot output “No”. This contradiction completes the correctness proof.

Analysis of running time and space used: Since $|L| = q$, the number of iterations of the for loop in Step 2 of Algorithm 2 is at most 2^q . Let $|R'| = r$. Thus, each representative set has size at most r . For each subcollection L' (which has at most q sets), the set R'' in Step 2(i) of Algorithm 2 can be computed in $O(rq)$ time. Further, the subcollection B' of $B = \{R_{q+1}, \dots, R_s\}$ such that the intersection between any set in B' and any set in L' is empty can be constructed in $O(qrs)$ time. The resulting instance of VS-2 has at most s sets (each of size one or two) and a representative set R'' of size at most r . Therefore, by Theorem 2, the VS-2 instance can be solved in time $O(s\sqrt{r+s})$ time. Thus, the running time for each subcollection L' is $O(qrs + s\sqrt{r+s}) = O(qs(r + \sqrt{r+s}))$. Since the algorithm tries at most 2^q subcollections, the overall running time is $O(q2^q s(r + \sqrt{r+s}))$. This running time has the form $O(h(q)N^{O(1)})$, where $h(q) = q2^q$ depends only on the parameter q and $N = rs$ is the size of the VS-FL instance. Hence, VS-FL is fixed parameter tractable with respect to the parameter q . Further, since the size of the representative set of each mutant is at most r and the total number of mutants (i.e., source and target mutants) is $s + 1$, the space used by the algorithm is $O(rs)$. This completes the proof of the theorem. ■

3.4.5 A fixed parameter tractability result for VPS

We now turn our attention to the VPS problem. Recall that we are given a set S of s source mutants, a set P of p permissible mutants, and a target mutant μ' . Our goal is to compute a verifiable synthesis of μ' of smallest size from the mutants in S that is permissible with respect to P , if such a synthesis exists. We present an FPT algorithm for this problem in the parameter $k = |R'|$ (R' is the representative set of μ').

We assume that every mutants in S and in P has a representative set of size at most k , since only such a mutant can be used in the synthesis of μ' . We assume that each source mutant is permissible, as is the target mutant. We start with a simple definition that captures the notion of verifiable and permissible synthesis.

Definition 3.4.1. *A target mutant μ' is P-realizable (for “Physically realizable”) if (a) μ' is a source mutant or (b) there is a verifiable synthesis of μ' from the source mutants such that each intermediate mutant in the synthesis is permissible.*

Thus, our goal is to determine whether a given target mutant μ' is P-realizable. In developing our algorithm, it is useful to define a computational notion of realizability, which we call C-realizability. A recursive definition of this notion is as follows.

Definition 3.4.2. A target mutant μ' with representative set R' is C-realizable (for “Computationally realizable”) if (a) μ' is a source mutant or (b) there is a partition of R' into two subsets R'_1 and R'_2 such that the two corresponding mutants μ'_1 and μ'_2 are both permissible and C-realizable.

The following theorem, shows that the notions of P-realizability and C-realizability are *equivalent*.

Theorem 4. A permissible target mutant μ' is C-realizable if and only if it is P-realizable.

Proof:

Part 1 (If): Suppose μ' is permissible and P-realizable. We use induction on the size of R' , the representative set of μ' to show that μ' is C-realizable. Let $\alpha \geq 1$ be the minimum size of the representative set of a source mutant. Since μ' is permissible, $|R'| \geq \alpha$.

Basis: $|R'| = \alpha$. In this case, since μ' is permissible, it is a source mutant. By definition, each source mutant is C-realizable. Hence, the basis holds.

Inductive hypothesis: Assume that for some $r \geq \alpha$, every permissible and P-realizable mutant μ whose representative set is of size $\leq r$ is also C-realizable.

Inductive proof: Let μ' be a permissible and P-realizable mutant whose representative set R' is of size $r + 1$. We need to prove that μ' is also C-realizable.

If μ' is a source mutant, then it is C-realizable by definition. So, assume that μ' is not a source mutant. Thus, there is a verifiable synthesis of μ' such that each intermediate mutant in the synthesis is permissible. Consider the last cross operation x in this synthesis. Let μ'_1 and μ'_2 denote the mutants which form the inputs to x . Hence, both μ'_1 and μ'_2 are permissible and the corresponding representative sets R'_1 and R'_2 are disjoint. Thus, there is a verifiable synthesis of μ'_1 and μ'_2 such that each intermediate mutant in the synthesis is permissible. In other words, both μ'_1 and μ'_2 are permissible and P-realizable. Further, the sizes of the corresponding representative sets R'_1 and R'_2 are $\leq r$. Therefore, by the inductive hypothesis, both μ'_1 and μ'_2 are C-realizable. In addition, sets R'_1 and R'_2 are disjoint and their union is R' . In other words, R'_1 and R'_2 form a partition of R' and the two corresponding mutants μ'_1 and μ'_2 are both C-realizable. Thus, by definition, μ' is also C-realizable. This completes the proof of Part 1.

Part 2 (Only If): Suppose μ' is permissible and C-realizable. We again use induction on the size of R' to show that μ' is also P-realizable. As before, let $\alpha \geq 1$ be the minimum size of the representative set of a source mutant. Since μ' is permissible, $|R'| \geq \alpha$.

Basis: $|R'| = \alpha$. In this case, since μ' is permissible, it is a source mutant. By definition, each source mutant is P-realizable. Hence, the basis holds.

Inductive hypothesis: Assume that for some $r \geq \alpha$, every permissible and C-realizable mutant μ whose representative set is of size $\leq r$ is also P-realizable.

Inductive proof: Let μ' be a permissible and C-realizable mutant whose representative set R' is of size $r + 1$. We will prove that μ' is also P-realizable.

If μ' is a source mutant, then it is P-realizable by definition. So, assume that μ' is not a source mutant. Since μ' is C-realizable, there is a partition of R' into two two subsets R'_1 and R'_2 such that the two corresponding mutants μ'_1 and μ'_2 are both permissible and C-realizable. Note that $|R'_1| \leq r$ and $|R'_2| \leq r$. Thus, by the induction hypothesis, both μ'_1 and μ'_2 are permissible and P-realizable. Now, construct a synthesis of μ' by adding a new cross operation x whose inputs are the mutants μ'_1 and μ'_2 . (Note that these may be source mutants or outputs of cross operations.) It is easy to see that this is a verifiable synthesis of μ' where each intermediate mutant is permissible. In other words, μ' is P-realizable, and this completes the proof of Part 2 as well as that of the theorem. ■

In view of the above theorem, we can use the notion of C-realizability to develop a dynamic programming (DP) algorithm to solve the EVPS problem for a target mutant μ' . Let $|R'| = k$ and let $\mathbb{P}(R') = \langle \beta_1, \beta_2, \dots, \beta_{2^k-1} \rangle$ denote an ordered collection of all the non-empty subsets of R' , where the subsets are in *non-decreasing* order of size; thus, for each i , $1 \leq i \leq k - 1$, all subsets of size i precede all those with size $i + 1$. The DP algorithm uses a Boolean array B of size $2^k - 1$. For convenience, we will index the elements of B with the subsets in $\mathbb{P}(R')$; that is, $B[\beta]$ denotes the entry of the array corresponding to the subset $\beta \in \mathbb{P}(R')$. The significance of B is as follows. For each $\beta \in \mathbb{P}(R')$, $B[\beta] = \text{TRUE}$ if the mutant whose representative set is β is permissible and C-realizable; otherwise, $B[\beta] = \text{FALSE}$.

Algorithm 3 shows how the entries of B can be computed using DP. We can then determine whether μ' can be synthesized from the value of $B[R']$. The correctness of Algorithm 3 is a direct consequence of Theorem 4. In Section 3.4.6, we show that the algorithm can be implemented to run in time $O(k 2^{2k} + k 2^k (p + s))$ time, which establishes that EVPS is FPT with respect to k . In Section 3.4.6, we also explain how to modify the algorithm to produce a permissible and verifiable synthesis with a minimum number of crosses; the running time and space requirements remain the same. The following theorem summarizes the above discussion.

Theorem 5. *The VPS problem can be solved in $O(k 2^{2k} + k 2^k (p + s))$ time using $O(k 2^k)$ space.*

3.4.6 Running Time Analysis and Extensions of Algorithm 3 for the EVPS Problem

Throughout this section, we will refer to the steps of the dynamic programming algorithm for the EVPS problem shown in Algorithm 3.

Algorithm 3 A dynamic programming algorithm for EVPS.

Input: A set S of s source mutants, a set P of p permissible intermediate mutants and a target mutant μ' .

Output: “Yes” if there is a verifiable and permissible synthesis of μ' and “No” otherwise.

Note: Please see the text for the definitions of the Boolean array B and the ordered collection of sets $\mathbb{P}(R')$.

Steps of the algorithm:

1. For each $\beta \in \mathbb{P}(R')$ such that $|\beta| = 1$, if β is the representative set of a source mutant, then set $B[\beta] = \text{TRUE}$. Otherwise set $B[\beta] = \text{FALSE}$.
 2. For each $\beta \in \mathbb{P}(R')$ with $|\beta| \geq 2$ (in the order specified by $\mathbb{P}(R')$)
 - (a) Let μ be the mutant whose representative set is β .
 - (b) If μ is not permissible then set $B[\beta] = \text{FALSE}$ and start the next iteration of of Step 2.
 - (c) If μ is a source mutant, then set $B[\beta] = \text{TRUE}$ and start the next iteration of of Step 2.
 - (d) Check if there is a partition of β into two subsets β_1 and β_2 such that $B[\beta_1] = B[\beta_2] = \text{TRUE}$. If so, set $B[\beta] = \text{TRUE}$; otherwise, set $B[\beta] = \text{FALSE}$.
 3. (Here, all the entries of the array B have been computed.) If $B[R']$ is TRUE , then output “Yes”; otherwise, output “No”.
-

Analysis of running time and space used. We start with a discussion of a data structure that can be used in the implementation of the algorithm. Since $|R'| = k$, each subset of R' can be represented by a k -bit vector. The Boolean array B can be implemented as a k -level **binary trie** [82] where the leaves represent the array elements. The amount of memory used by this data structure is $O(2^k)$. Given the k -bit vector for a particular subset β of R' , the leaf node corresponding to $B[\beta]$ can be accessed in $O(k)$ time by following the path from the root of the trie to the leaf node.

Using the trie data structure for the array B , Step 1 of the algorithm can be implemented to run in time $O(k^2)$. The number of iterations of the for loop of Step 2 is at most 2^k . In each iteration, testing whether the subset β is permissible (i.e., whether it is in P) can be done in $O(kp)$ time. Likewise, in each iteration, testing whether the subset β represents a source mutant can be done in $O(ks)$ time. Thus, over all the 2^k iterations of the loop in Step 2, these checks use $O(2^k k(p + s))$ time.

Now, we estimate the time used in Step 2(d). In each iteration, Step 2(d) considers a subset β of R' , with $|\beta| \leq k$. For such a subset β , there are at most 2^{k-1} partitions into two subsets. For any partition of β into β_1 and β_2 , we can determine the values of $B[\beta_1]$ and $B[\beta_2]$ in $O(k)$ time. Thus, the time for Step 2(d) in each iteration is $O(2^{k-1} k)$. Since the number of iterations of the loop is at most 2^k , the running time of Step 2(d) over all the iterations is $O(2^k 2^{k-1} k) = O(k 2^{2k})$. Therefore, the total time for Step 2 is $O(k 2^{2k} + k 2^k(p + s))$. Since Step 2 is the dominant part of the algorithm, the running time of the algorithm is

also $O(k 2^{2k} + k 2^k(p + s))$. The algorithm uses $O(2^k)$ space for the array B and $O(k2^k)$ space for the ordered set collection $\mathbb{P}(R')$. Further, the representative set of each source and permissible mutant is at most k . So, the space for all the representative sets is $O(k(p + s))$. Since p and s are both at most 2^k , the space requirement is dominated by that of the ordered collection $\mathbb{P}(R')$. Therefore, the space used by the algorithm is $O(k2^k)$.

Extension to obtain a synthesis with a minimum number of crosses. We can extend the dynamic programming algorithm so that whenever there is a permissible and verifiable synthesis, it finds one that uses the smallest number of crosses. To achieve this extension we use an enhanced version of array B . In this version, each element $B[\beta]$ is a record with three fields which we denote by `flag`, `min-crosses` and `partition`. The `flag` field indicates whether β can be synthesized in a permissible and verifiable manner. When $B[\beta].\text{flag}$ is `TRUE`, the `min-crosses` field stores the minimum number of cross operations needed to obtain a permissible and verifiable synthesis of β and the `partition` field stores one partition of β into two subsets β_1 and β_2 such that $B[\beta_1] = B[\beta_2] = \text{TRUE}$ and the value $B[\beta_1].\text{min-crosses} + B[\beta_2].\text{min-crosses}$ is the smallest among all the partitions of β . Since the dynamic programming algorithm considers all partitions of any set β , the values of the fields of each record $B[\beta]$ can be suitably updated. At the end, if $B[R'].\text{flag}$ is `TRUE`, then the value $B[R'].\text{min-crosses}$ gives the minimum number of cross operations needed to obtain a synthesis of μ' . It is not difficult to see that the running time of and the space used by used by this version remain $O(k 2^{2k} + k 2^k(p + s))$ and $O(k2^k)$ respectively.

3.5 Results

As we noted in the introduction, there are no published databases that contain several mutants with three or more mutations that have been made experimentally. Therefore, we had to create novel datasets to test and evaluate our algorithms. We generate two complementary types of datasets: simulated and synthetic. Our evaluations on simulated datasets seek to replicate realistic scenarios faced by experimentalists who make new mutants from existing strains in order to validate computational predictions [9, 38, 39] (Section 3.5.1). To this end, we simulate mathematical models of the cell cycle in budding yeast in order to generate biologically interesting and meaningful target and permissible mutants. Synthetic datasets allow us to assess the variation in the performance of Algorithms 2 and 3 for the VS-FL and VPS problems as we vary the value of the corresponding fixed parameter (Section 3.5.2).

To generate the results for VS-FL, we used a 0-1 integer linear program (ILP) to solve the DCS instances (see Section 3.4.3). The ILP was easier to implement than the algorithm in [79]; moreover, solving the ILP using Gurobi was very efficient for our problem sizes. However, for some datasets and parameter values, a significant fraction of the time was spent in reading and writing files for this ILP. Therefore, we ignore this cost when we report

the running times of our algorithms. For the VPS problem, we implemented a recursive algorithm with caching, as an alternative to the DP.

3.5.1 Evaluation on cell cycle model simulations

Simulations to Compute Target and Permissible Mutants. We selected a dynamic model of the budding yeast cell cycle [9] since we use its predictions in an ongoing collaboration on making and characterizing novel cell cycle mutants [9, 39, 83]. This model accurately simulates the phenotypic properties of more than 250 cell cycle mutants in budding yeast that have been published in the literature. Starting from 35 single-gene mutations, we simulated this model for mutations in all combinations of up-to-six genes. For each mutant, we recorded whether the simulation predicted the phenotype as “viable” or “inviable.”

Given two mutants a and b with respective representative sets R_a and R_b , we defined b to be a *parent* of a if $|R_a - R_b| = 1$, i.e., a carries one more single gene mutation than b . We say that b is an *ancestor* of a if $|R_a - R_b| \geq 1$. We further define two specific classes of mutants, namely **rescued** and **synthetic lethal**. We define a mutant a to be *rescued* if a is predicted to be viable but had a parent b that is either experimentally known or predicted by the model to be inviable. We note that a rescued mutant may be *redundant*. For example, $whi5\Delta$ ¹ rescues the inviable mutants $bck2\Delta cln3\Delta$ and $bck2\Delta cln3\Delta pds1\Delta$. Since $pds1\Delta$ is itself a viable strain, the loss of Pds1 is irrelevant to the rescue phenotype. Hence, we consider only non-redundant rescued mutants, i.e., $bck2\Delta cln3\Delta whi5\Delta$ but not $bck2\Delta cln3\Delta pds1\Delta whi5\Delta$. We defined a mutant a to be *synthetic lethal* (abbreviated as SL) if a is inviable and all of a ’s parents are viable.

Only 8% of the 1.6M mutants we simulated were viable (Table 3.1) The percentage of viable combinations decreased with an increase in the number of genes mutated, which supports

¹This notation represent the deletion of the Whi5 gene.

#Mutations	Mutants	Viable	Rescued	Rescued non-red.	Synthetic Lethals
1	35	26 (74%)	-	-	-
2	586	285 (49%)	14	14	49
3	6,250	1,896 (30%)	344	170	55
4	47,705	8,872 (18%)	3,170	516	57
5	277,531	31,060 (11%)	16,722	1,098	58
6	1,279,720	84,709 (7%)	56,993	243	27
Total	1,611,827	126,848 (8%)	77,243	2,041	246

Table 3.1: Statistics on simulations. We ignored combinations that contained redundant pairs of mutations, e.g., $cdh1\Delta$ and CDH1_constitutive. We abbreviate “non-redundant” as “non-red.”

Single gene mutants	Double gene mutants	Triple gene mutants
<i>cln3</i> Δ	<i>cln3</i> Δ <i>bck2</i> Δ	<i>cln3</i> Δ <i>bck2</i> Δ <i>whi5</i> Δ
<i>bck2</i> Δ	<i>cln3</i> Δ <i>whi5</i> Δ	<i>cln3</i> Δ <i>whi5</i> Δ <i>mbp1</i> Δ
<i>sic1</i> Δ	<i>cln3</i> Δ <i>mbp1</i> Δ	<i>cln3</i> Δ <i>mbp1</i> Δ <i>swi6</i> Δ
<i>cki</i> Δ	<i>bck2</i> Δ <i>whi5</i> Δ	
<i>cln2</i> Δ	<i>bck2</i> Δ <i>mbp1</i> Δ	
<i>whi5</i> Δ	<i>cln2</i> Δ <i>whi5</i> Δ	
<i>cdh1</i> Δ	<i>cln2</i> Δ <i>mbp1</i> Δ	
CDH1_constitutive	<i>whi5</i> Δ <i>swi4</i> Δ	
<i>clb2</i> Δ	<i>whi5</i> Δ <i>mbp1</i> Δ	
CLB1 <i>clb2</i> Δ	<i>whi5</i> Δ <i>swi6</i> Δ	
<i>cdc20</i> Δ	<i>mbp1</i> Δ <i>swi6</i> Δ	
<i>clb5</i> Δ		
<i>swi5</i> Δ		
<i>apc^{ts}</i>		
APCA		
<i>tem1</i> Δ		
<i>cdc15</i> Δ		
<i>net1^{ts}</i>		
<i>cdc14^{ts}</i>		
TAB6-1		
<i>pds1</i> Δ		
<i>ppx</i> Δ		
<i>mad2</i> Δ		
<i>bub2</i> Δ		
<i>swi4</i> Δ		
<i>mbp1</i> Δ		
<i>clb5</i> Δ		
<i>swi6</i> Δ		
<i>cdc5</i> Δ		
<i>cdc6</i> Δ		
<i>cdc14-1</i> Δ		
<i>ssa1</i> Δ		
<i>ydj1</i> Δ		
<i>lte1</i> Δ		
<i>swe1</i> Δ		

Table 3.2: The 49 source mutants used in Sec. 3.5.1. CDH1_constitutive is a mutant that expresses CDH1 constitutively. CLB1 *clb2*Δ refers to a strain with *clb2* deletion with wildtype CLB1 (CLB1 and CLB2 are paralogs that are redundant in function). *apc^{ts}* refers to a non-functional Anaphase-promoting Complex, which is achieved by using a temperature sensitive strain. TAB6-1 refers to *telophase arrest bypass* mutants representing altered Cdc14:Net1 stoichiometries at mitotic exit.

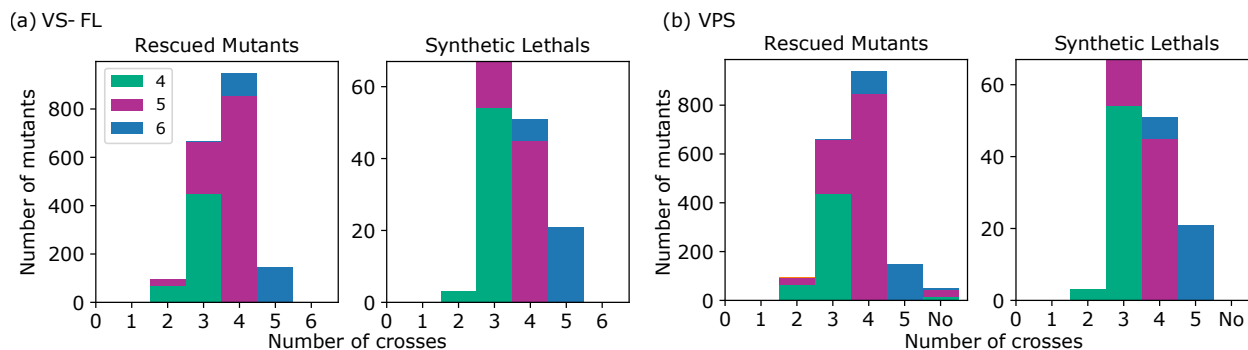


Figure 3.3: Distribution of the number of crosses in optimal syntheses for rescued mutants and for SL mutants (x -axis: number of crosses, y -axis: number of mutants with an optimal synthesis of that size, legend: the number of mutations in each target mutant). (a) Results for Algorithm 2 for VS-FL. (b) Results for Algorithm 3 for VPS. x -axis label of “No:” mutants without a verifiable, permissible syntheses.

our consideration of permissible mutants in the VPS problem. There were 77,243 rescued mutants of which 2,041 (2.6%) were non-redundant. Out of 1.48M inviable mutants, only 246 (0.016%) were SL. We specified inputs to our algorithms as follows:

Target mutants: Two sets: all four-, five-, and six-gene (i) non-redundant rescued mutants and (ii) SL mutants.

Source mutants: 49 mutants available to our experimental collaborators [39] (Neil Adames and Jean Peccoud, personal communication), including 35 single, 11 double, and three triple mutants (Table 3.2).

Permissible mutants: All viable mutants in our simulations.

Results for VS-FL. We used Algorithm 2 for VS-FL to compute the optimal synthesis for each target mutant. The total running time for these computations was 128.75 seconds (average of 0.07 second per mutant). Figure 3.3(a) summarizes these results for rescued mutants and for SL mutants. The majority of target mutants required one fewer cross than the number of mutated genes (e.g., four crosses for a five-gene target mutant). The VS-FL algorithm capitalized on double mutants in the source set to compute syntheses with three crosses for 217 five-gene rescued mutants and syntheses with two crosses for 68 four-gene rescued mutants. For 26 five-gene mutants, the algorithm could compute syntheses with just two crosses, thus taking advantage of triple gene mutants in S . We observed similar trends for SLs.

Results for VPS. We used Algorithm 3 to compute an optimal synthesis for each target mutant in the presence of permissible mutants. The total running time for these computations was 18.5 seconds (average of 0.01 second per mutant). Figure 3.3(b) shows the results.

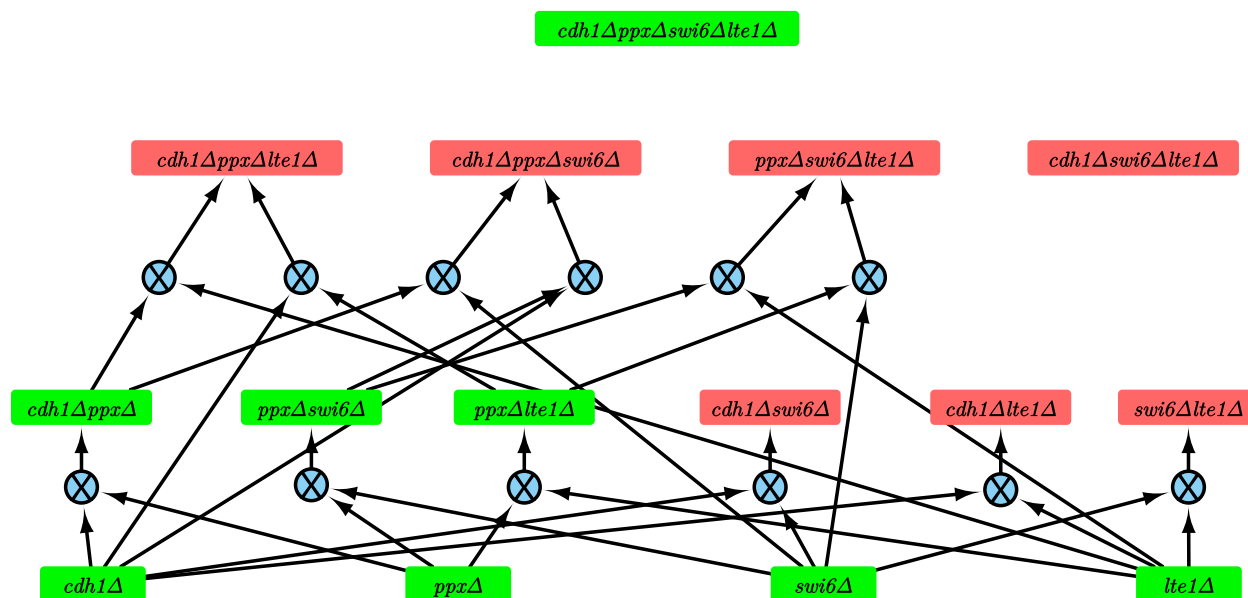


Figure 3.4: Example of a target rescued mutant for which there is no verifiable permissible synthesis. Each rectangular node is a mutant. A permissible mutant is green (the simulations predict it to be viable). A non-permissible mutant is red (the simulations predict it to be inviable). The label on a node lists the genes deleted in that mutant. Each blue circle represents a verifiable cross. Each such node has two incoming edges and one outgoing edge. This network contains all crosses involving the displayed mutants; it is identical to the genetic cross graph used by our ILP-based algorithm in Chapter 2. The node at the top of the network is the target rescued mutant. It does not have a permissible synthesis since (i) all its parents (triple mutants) are inviable and (ii) only one mutant is viable in every pair of double mutants that could be crossed in a verifiable synthesis of the target mutant. An alternative way of stating the second reason is that every pair of viable double mutants has one common gene, which means that crossing these double mutants is not a verifiable synthesis.

We found that 1,992 of the 2,041 (97.6%) of the rescued mutants and all SL mutants had a verifiable, permissible synthesis. Of the 49 target mutants that did not have such a synthesis, there were 18 four-gene, 26 five-gene, and 5 six-gene mutants. Figure 3.4 displays such a mutant. In the caption of the figure, we explain why this mutant does not have a verifiable, permissible synthesis.

For each mutant with a verifiable, permissible synthesis, we compared the syntheses computed by the VS-FL and the VPS algorithms to determine if the restrictions imposed by permissible mutants increased the number of crosses in the synthesis. The size of the synthesis did not increase for most target mutants. However, four rescued mutants did have a larger synthesis for the VPS problem than for the VS-FL problem. Figure 3.5(a) displays a synthesis with two crosses that is not permissible and Figure 3.5(b) displays a verifiable, permissible synthesis of the same mutant with three crosses.

These results show that permissible mutants can affect the synthesis size, even if only for a small number of target mutants. More importantly, the requirement of permissibility can preclude the existence of a synthesis for several target mutants. Our algorithms can handle the additional restrictions imposed by permissibility.

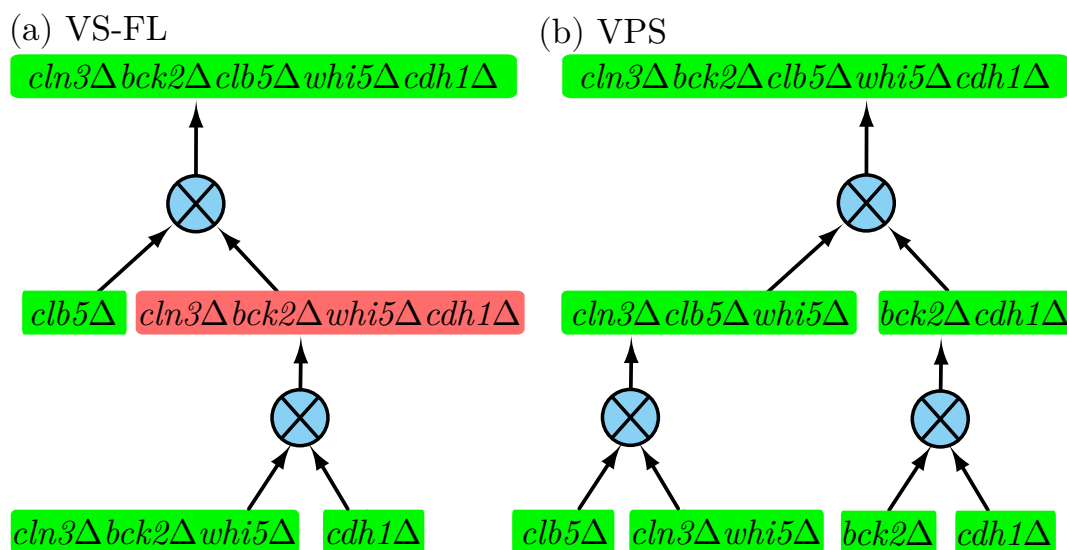


Figure 3.5: (a) Example of a verifiable but non-permissible synthesis with two crosses computed by Algorithm 2 for VS-FL. Node colors are as in Figure 3.4. (b) Example of a verifiable, permissible synthesis with three crosses computed by Algorithm 3 for VPS. Note that even though the triple mutant $cln3\Delta bck2\Delta whi5\Delta$ is a source, Algorithm 3 does not use it in the synthesis since both $cln3\Delta bck2\Delta whi5\Delta cdh1\Delta$ and $cln3\Delta bck2\Delta whi5\Delta clb5\Delta$ (not shown) are not permissible mutants (they are inviable).

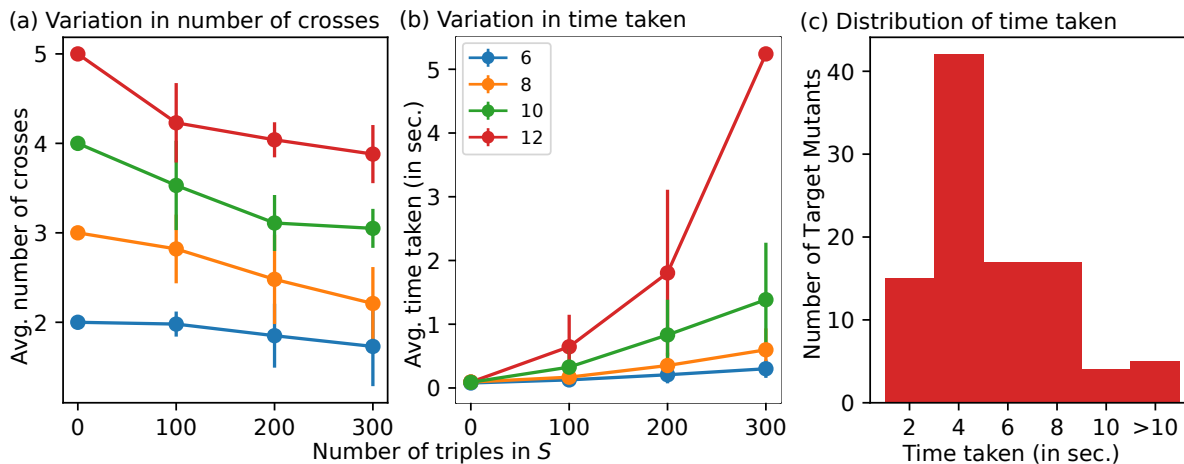


Figure 3.6: VS-FL results for synthetic data. Error bars indicate one standard deviation. The legend in (b) shows k . (a) The average number of crosses in an optimal synthesis as we change q (x -axis). (b) Average time in seconds. (c) Distribution of running time for 100 target mutants, $k = 12$, $q = 300$.

3.5.2 Evaluation on synthetic datasets

Generating Synthetic Data. Recall that Algorithm 2 for the VS-FL problem is FPT in q , the number of source mutants with three or more mutations, while Algorithm 3 for the VPS problem is FPT in k , the number of mutations in the target mutant. We created synthetic datasets that varied both parameters so that we could use the same inputs to both algorithms. We created representative sets for all mutants from a universe U of 35 genes; we used this number to match the cell cycle model.

Target mutants: We generated a random target mutant whose representative set contained $k = 6, 8, 10,$ or 12 genes from U . We used these values of k to test our algorithms on larger target mutants than we used in Section 3.5.1, where every mutant contained at most six mutations. For each of these four values, we report results averaged over 100 target mutants. Let $U' \subseteq U$ be the union of the representative sets of these 100 target mutants.

Source mutants: We started with S containing all singleton and doubleton subsets of U' . This choice ensured that each target mutant had a synthesis with at most $k/2 - 1$ crosses, thereby making the results depend only on the triple mutants added. Next, we varied q from 0 to 300 in steps of 100 and added q triple mutants for S . In selecting the representative sets of these mutants, we sampled subsets of U' uniformly at random. An experimentalist seeking to make a k -gene target mutant is likely to first make one or more triple mutants that are ancestors of the target mutant. We developed this strategy of selecting triple mutants to mimic the experimentalist. We stress that we

use U' only to select source mutants. We still apply our algorithms to each target mutant independently.

Permissible mutants: For each triple mutant μ we added to S , we considered every descendant of μ to be permissible, as long as the representative set of that descendant was a subset of U' .

Results for VS-FL. Figure 3.6(a) plots how the average number of crosses needed to make a target mutant changes with q . When $q = 0$, the algorithm for VS-FL runs Algorithm 1 for VS-2, and computes syntheses that only involve double mutants. As we increase q , the average number of crosses decreases as the synthesis can now take advantage of triple gene mutants. However, as the number of triple gene mutants in the source set increases, the time taken to run the algorithm for VS-FL increases considerably, especially for targets with 12 mutations (Figure 3.6(b)), as we may expect from the exponential dependence on q of the worst-case running time of Algorithm 2. For $k = 6, 8, 10$, a significant fraction of the time was spent in reading and writing files for the DCS ILP. For target mutants with 12 mutations, there is a considerable variation in the running times, especially for $q = 300$. Instead of plotting this error bar in Figure 3.6(b), we display the distribution of running times in Figure 3.6(c). While the algorithm ran in less than four seconds for over 50% of the target mutants, it took 12 seconds for three mutants and 84 seconds for three other mutants.

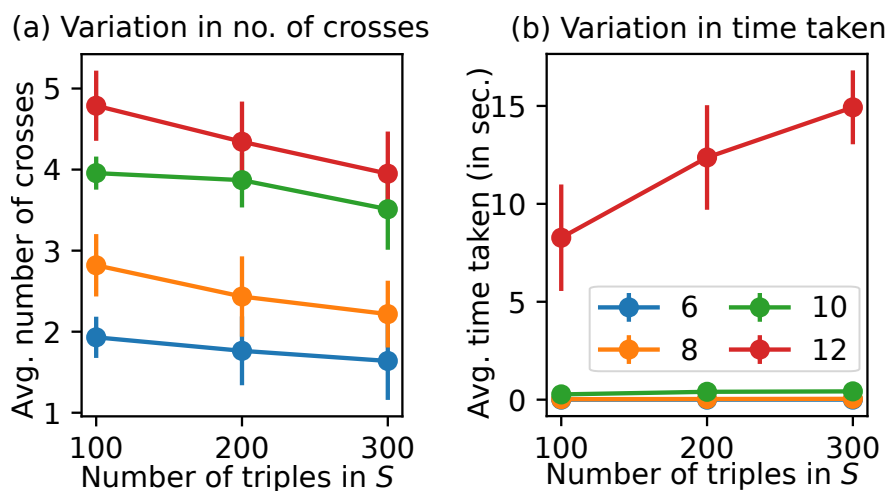


Figure 3.7: VPS results for synthetic data, averaged over 100 target mutants. Error bars indicate one standard deviation. The legend in (b) shows k . The x -axis corresponds to q . (a) Average number of crosses in an optimal synthesis. (b) Average time taken in seconds.

Results for VPS. As we increase q , the number of triples in S , we also increase the size of the permissible set. Therefore, the number of crosses in an optimal synthesis decreases with an increase in q (Figure 3.7(a)). The average time taken by the VPS algorithm is negligible for $k = 6, 8, 10$ (Figure 3.7(b)), in contrast with the result for VS-FL (Figure 3.6(b)). For

$k = 12$, the average time taken by the algorithm for VPS increases considerably compared to smaller values of k (Figure 3.7(b)), since the algorithm is FPT with respect to this parameter. Moreover, the running time also increases with q : the more permissible mutants there are, the longer Algorithm 3 takes to find an optimal synthesis. Nevertheless, all running times are at most 20 seconds. Comparing Figure 3.7(b) and Figure 3.6(b) for $k = 12$ and $q = 300$, the VPS algorithm is about four times slower than the VS-FL algorithm.

Comparing VPS to CrossPlan. Here, we compare the VPS algorithm to our ILP-based approach, CrossPlan (see Chapter 2). We run both algorithms to compute an optimal verifiable synthesis for each target mutant with $k = 12$. For a specific target mutant μ , in addition to the inputs to VPS, CrossPlan takes two additional inputs: (a) l , an upper bound on the number of crosses needed and (b) a genetic cross graph $G(\mu)$. We set $l = k/2 - 1$, since S contains all double mutants. The graph $G(\mu)$ contains one *mutant node* for each permissible mutant whose representative set is a subset of μ 's set. For every pair of permissible mutants with disjoint representative sets, $G(\mu)$ contains a *cross node* that represents the corresponding genetic cross. In addition, $G(\mu)$ contains two incoming edges and one outgoing edge for each cross. See Figure 3.4 for an example. We compute this graph independently for each target mutant. On average, a graph contains $4,095 = 2^{12} - 1$ mutant nodes and 66,000 cross nodes.

Both algorithms report optimal synthesis of the same size or that no verifiable, permissible synthesis exists. However, the VPS algorithm is over 28 times faster (24.83 minutes vs 700.77 minutes) in computing optimal syntheses. It was 10 times faster than CrossPlan (9 minutes vs. 90.08 minutes) for the question of determining if a verifiable, permissible synthesis exists. The CrossPlan ILP contained 300K variables and 600K constraints, on average. In fact, due to the size of the ILP, we set a time limit of six minutes on the Gurobi solver for each target mutant since in our experience the solver computes a heuristic solution quickly but can spend hours in proving its optimality. These results show the benefit of the strategy we have adopted in this chapter of formulating synthesis problems that do not need the explicit construction of the genetic cross graph.

3.6 Conclusions

We introduced a fundamental problem in computational biology: how to use genetic crosses to efficiently synthesize a target mutant from source mutants. We formalized this question in several ways that incorporated key experimental considerations of verifiability and permissibility. We showed that checking the existence of a synthesis is **NP**-complete. We presented one polynomial time and two FPT algorithms for these problems. On simulated and synthetic data, these algorithms ran efficiently and provided useful results.

There are several interesting directions for future research. Developing FPT algorithms that compute an optimal synthesis for multiple target mutants is an important open problem.

These methods will complement the ILP-based solution we have proposed [84].

It may also be useful to study the scenario when permissible intermediate mutants are specified implicitly rather than explicitly. A simple example of an implicit specification is to allow every intermediate mutant to be permissible, which is identical to the VS problem. Another example is the following: an intermediate mutant is permissible if and only if it has no gene from the set $\{g_1, g_2, g_3\}$; this set could correspond to genes whose products form an essential protein complex. Such a description can specify an exponentially large set of permissible intermediate mutants. However, given a target mutant μ , we can efficiently determine whether it is permissible under the given specification. Our algorithm for VPS in Section 3.4.5 can be readily modified to work for such implicit specifications of permissible intermediate mutants.

Our current model for creating new mutants includes only the genetic cross. It is important to extend this model to include other experimental techniques, especially genome editing methods, while considering the relative costs of each approach. We are also interested in studying and incorporating more complex constraints imposed by genetic crosses in model organisms.

Chapter 4

BEELINE: Benchmarking gene regulatory network inference from single-cell transcriptomic data

Aditya Pratapa, Amogh Jalihal, Jeffrey Law, Aditya Bharadwaj, T.M. Murali (2020) Benchmarking algorithms for gene regulatory network inference from single-cell transcriptomic data. *Nature Methods*, 17: pp. 147-154

4.1 Introduction

Single-cell RNA-sequencing technology has made it possible to trace cellular lineages during differentiation and to identify new cell types [2, 3]. A central question that arises now is whether we can discover the gene regulatory networks (GRNs) that control cellular differentiation and drive transitions from one cell type to another. In such a GRN, each edge connects a transcription factor (TF) to a gene it regulates. Ideally, the edge is directed from the TF to the target gene, represents direct rather than indirect regulation and corresponds to activation or inhibition.

Single-cell expression data are especially promising for computing GRNs because, unlike bulk transcriptomic data, they do not obscure biological signals by averaging over all the cells in a sample. However, these data have features that pose significant difficulties; for example, substantial cellular heterogeneity [45], considerable cell-to-cell variation in sequencing depth, the high sparsity of the data caused by dropouts [46], and cell-cycle-related effects [47]. Despite these challenges, over a dozen methods have been developed or used to infer GRNs from single-cell data [5, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59]. An experimentalist seeking to analyze a new dataset faces a daunting task in selecting an appropriate inference method since there are no widely accepted ground-truth datasets for assessing algorithm accuracy and the criteria for evaluation and comparison of methods are varied.

We have developed BEELINE, a comprehensive evaluation framework to assess the accuracy, robustness and efficiency of GRN inference techniques for single-cell gene expression data based on well-defined benchmark datasets (Figure 4.1). BEELINE incorporates 12 diverse GRN inference algorithms. It provides an easy-to-use and uniform interface to each method

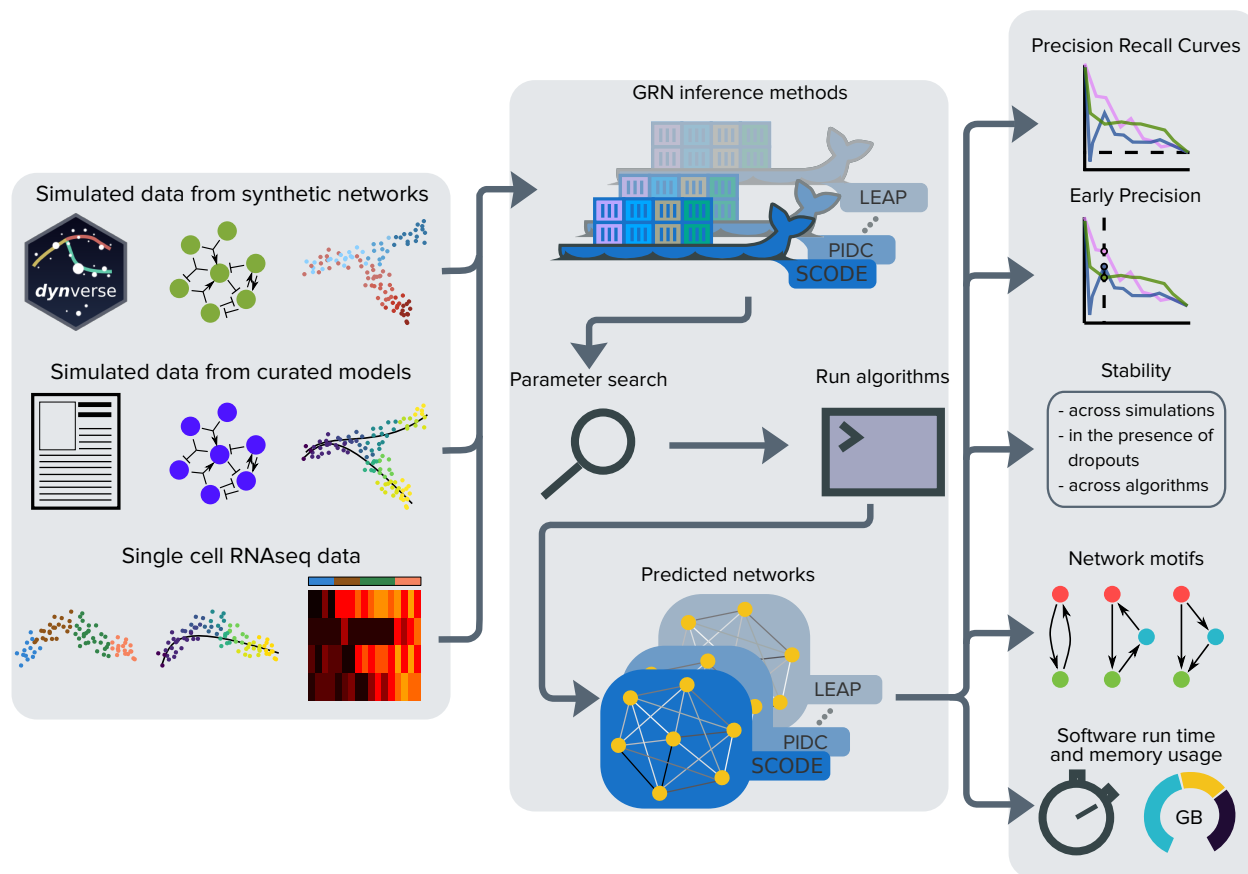


Figure 4.1: An overview of the BEELINE evaluation framework. We apply GRN inference algorithms to three types of data: datasets from synthetic networks, datasets from curated Boolean models from the literature and experimental single-cell transcriptional measurements. We process each dataset through a uniform pipeline: preprocessing, Docker containers for 12 GRN inference algorithms, parameter estimation, postprocessing and evaluation. We compare algorithms based on accuracy (AUPRC and early precision), stability of results (across simulations, in the presence of dropouts and across algorithms), analysis of network motifs and scalability.

in the form of a Docker image. BEELINE implements several measures for estimating and comparing the accuracy, stability and efficiency of these algorithms. Thus, BEELINE facilitates reproducible, rigorous and extensible evaluations of GRN inference methods for single-cell gene expression data.

4.2 Results

4.2.1 Overview of algorithms

We surveyed the literature and bioRxiv preprints for papers that either published a new GRN inference algorithm or used an existing approach. We ignored methods that did not assign weights or ranks to the interactions, required additional datasets or supervision, or sought to discover cell-type-specific networks. We selected 12 algorithms using these criteria (see Section 4.4.1).

We used BEELINE to evaluate these approaches on over 400 simulated datasets (across six synthetic networks and four curated Boolean models) and five experimental human or mouse single-cell RNA-seq datasets. Since eight algorithms require pseudotime-ordered cells, we used datasets (both simulated and real) that focus on cell differentiation and development, processes in which there is a meaningful temporal progression of cell states. We did not study GRNs relevant to other biological processes; for example, changes in disease states or differences among cell types.

4.2.2 Datasets from synthetic networks

Our motivations for using synthetic networks were two-fold. First, we wanted to use a known GRN that could serve as the ground truth. Second, we desired to create *in silico* single-cell gene expression datasets that were isolated from any limitations of pseudotime inference algorithms. Therefore, we started with six synthetic networks (Figure 4.2(a)). Simulating these networks should produce a variety of different trajectories seen in differentiating and developing cells [66]. Several recent studies on GRN inference [52, 54, 55, 62, 65] have used GeneNetWeaver [85], a method originally developed for generating time courses of bulk-RNA datasets from a given GRN. Accordingly, we simulated the six synthetic networks using GeneNetWeaver via the procedure outlined in Chan *et al.* [52] (see Section 4.4.2 for details). However, we observed no discernible trajectories when we visualized two-dimensional projections of these data (Figure 4.2(d)).

We therefore used our BoolODE approach to simulate the six synthetic networks (see Sections 4.4.2 and 4.4.3). For each gene in a GRN, BoolODE requires a Boolean function that specifies how that genes regulators combine to control its state. We represent each Boolean function as a truth table, which we convert into a nonlinear ordinary differential equation (ODE). This approach provides a reliable method to capture the logical relationships among the regulators precisely in the components of the ODE. We add noise terms to make the equation stochastic [66, 86].

For each network, we applied BoolODE by sampling ODE parameters ten times and generating 5,000 simulations per parameter set (Section 4.4.3). We created five datasets per

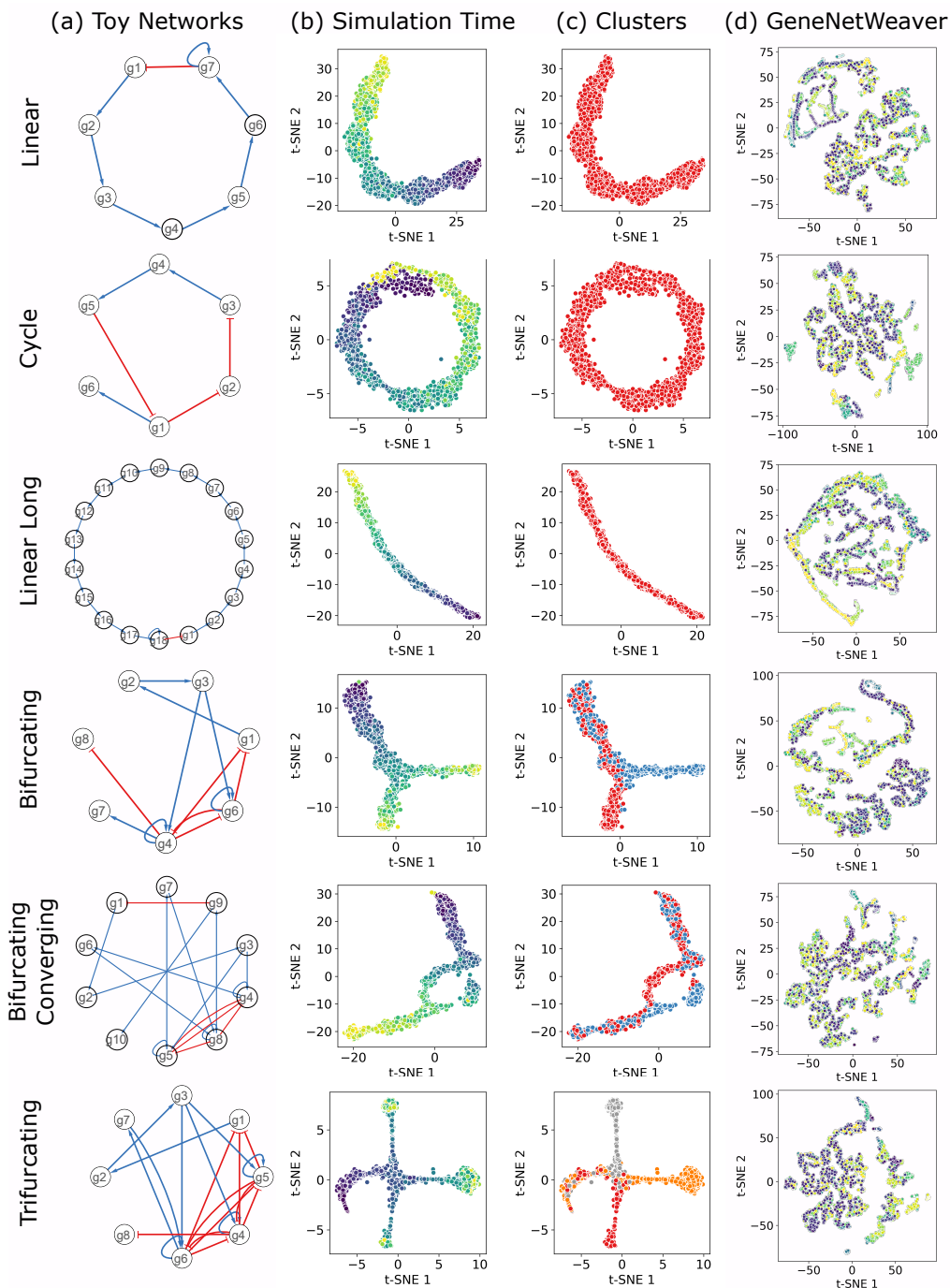


Figure 4.2: A comparison of datasets from synthetic networks simulated using BoolODE and GeneNetWeaver. Each row corresponds to the network indicated by the label on the left. (a) The network itself, with red edges representing inhibition and blue edges representing activation. (b) A 2D t-SNE visualization of one BoolODE-generated dataset for 2,000 cells. The color of each point indicates the simulation time: blue for earlier, green for intermediate, and yellow for later times. (c) Each color corresponds to a different subset of cells obtained by using k -means clustering of the BoolODE-generated dataset, with k set to the number of expected steady states. (d) A 2-D t-SNE visualization of one GeneNetWeaver output.

parameter set, one each with 100, 200, 500, 2,000 and 5,000 cells by sampling one cell per simulation, to obtain 50 different expression datasets. Analyzing the two-dimensional projections of these simulations reassured us that BoolODE was successful in correctly simulating the network models (Figure 4.2(b), (c), Appendix B.1.1).

Setting each network as the ground truth, we executed the 12 algorithms on every one of the 50 simulated datasets. For those GRN inference methods that required time information, we provided the simulation time at which we sampled each cell. For the bifurcating, bifurcating converging and trifurcating networks, we ran the algorithms that need time information on each trajectory individually and combined the outputs (Section 4.4.4). Six algorithms required one or more parameters to be specified. We performed a parameter sweep to determine the values that gave the highest median area under the precision-recall curve (AUPRC) (see Appendix B.1.2)

For each network–algorithm pair, Figure 4.3 displays the median AUPRC ratio (the AUPRC divided by that of a random predictor). Figures 4.4 and 4.5 show the box plots of AUPRC and area under the receiver operating characteristic curve (AUROC) values. The methods performed best for the linear network: 10 out of 12 algorithms had a median AUPRC ratio greater than 2.0. Seven methods had a median AUPRC ratio greater than 5.0 for the linear long network. The cycle, bifurcating converging, bifurcating and trifurcating networks were progressively harder to infer, with no algorithm achieving an AUPRC ratio of two or more on the last network. Single-cell regularized inference using time-stamped expression profiles (SINCERITIES) obtained the highest median AUPRC ratio for four out of the six networks. Single-cell inference of networks using Granger ensembles (SINGE) had the highest median AUPRC ratio for cycle and partial information decomposition and context (PIDC) for trifurcating.

We examined the effect of the number of cells on AUPRC by comparing the values for 100, 200, 500 and 2,000 cells to those for 5,000 cells (Appendix B.1.3). As the number of cells increased from 100 to 500, the number of algorithms with significantly lower AUPRC values in comparison to 5,000 cells decreased from seven to four. The number of cells had no significant effect on five algorithms: gene network inference with ensemble of trees (GENIE3), GRN variational Bayesian expectation-maximization (GRNVBEM), lag-based expression association for pseudotime-series (LEAP), single-cell network synthesis (SCNS) and SCODE.

To examine the stability of the results, we considered the GRNs formed by the k edges with the highest ranks, with k set to the number of edges in each synthetic network, computed the Jaccard indices of all pairs of GRNs, and recorded the medians of these values (Figure 4.3). While SINCERITIES, SINGE and SCRIBE had the three highest median-of-median AUPRC ratios, the networks they predicted were relatively less stable (median Jaccard index between 0.28 and 0.35). PPCOR and PIDC, the other two methods in the top five, had higher median Jaccard indices of 0.62 each.

We simulated the inferred GRNs to see if they had the same number of steady states as the

ground-truth networks. Apart from the linear network, we found that more than 60% of the GRNs yielded more steady states than in the ground truth (Appendix B.1.4). For networks with multiple steady states, there was no clear benefit to computing a single GRN after combining all trajectories over merging the GRNs inferred for each trajectory (Appendix B.1.5).

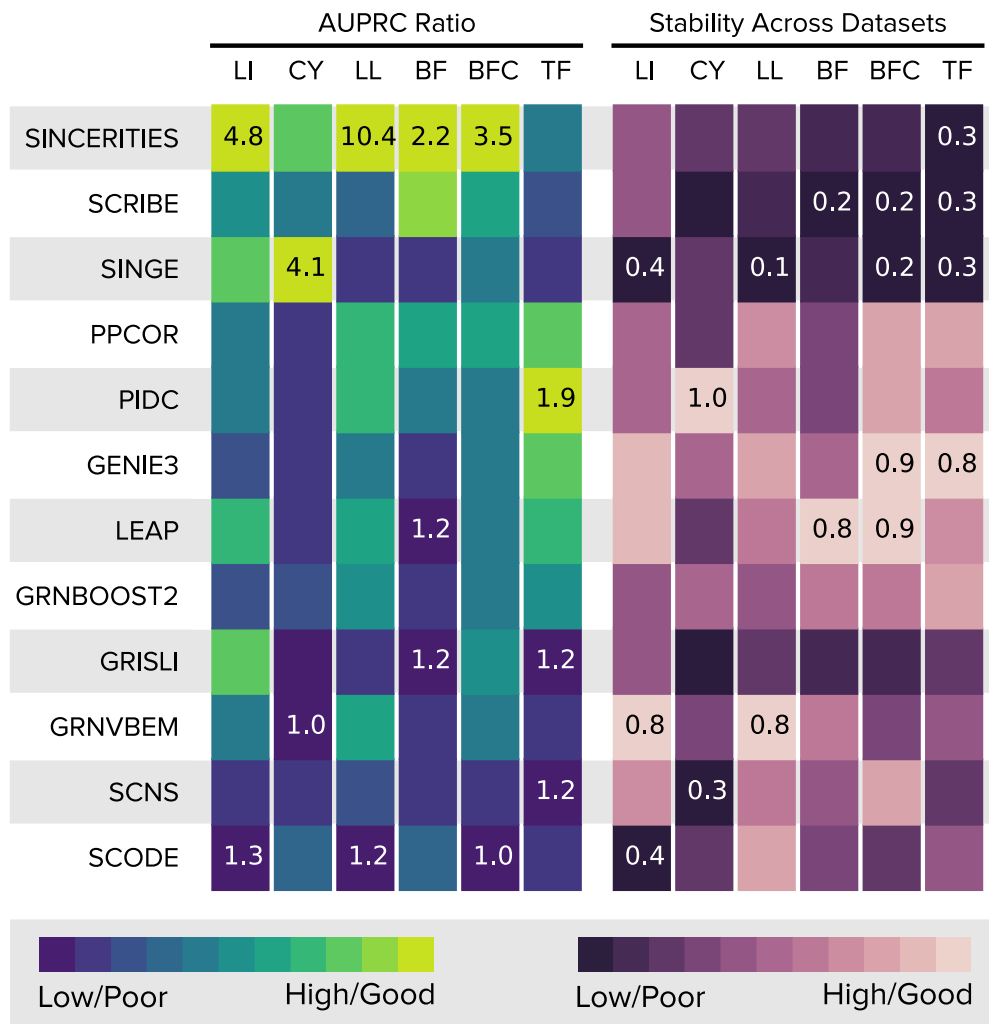


Figure 4.3: Summary of results for datasets from synthetic networks. The first set of six columns displays the median AUPRC ratio. The next three sets of six columns display the median stability score across multiple datasets, the median edge ratios, and median ratios of reachable steady states of the proposed GRN reconstructions for each of the six simulated datasets. The color is proportional to the corresponding value after scaling it between 0 and 1. For each dataset, we display the highest and lowest values inside the corresponding cells. We computed the medians over 20 simulated datasets for each synthetic network. Abbreviations: LI: Linear, CY: Cycle, LL: Linear Long, BF: Bifurcating, BFC: Bifurcating Converging, TF: Trifurcating.

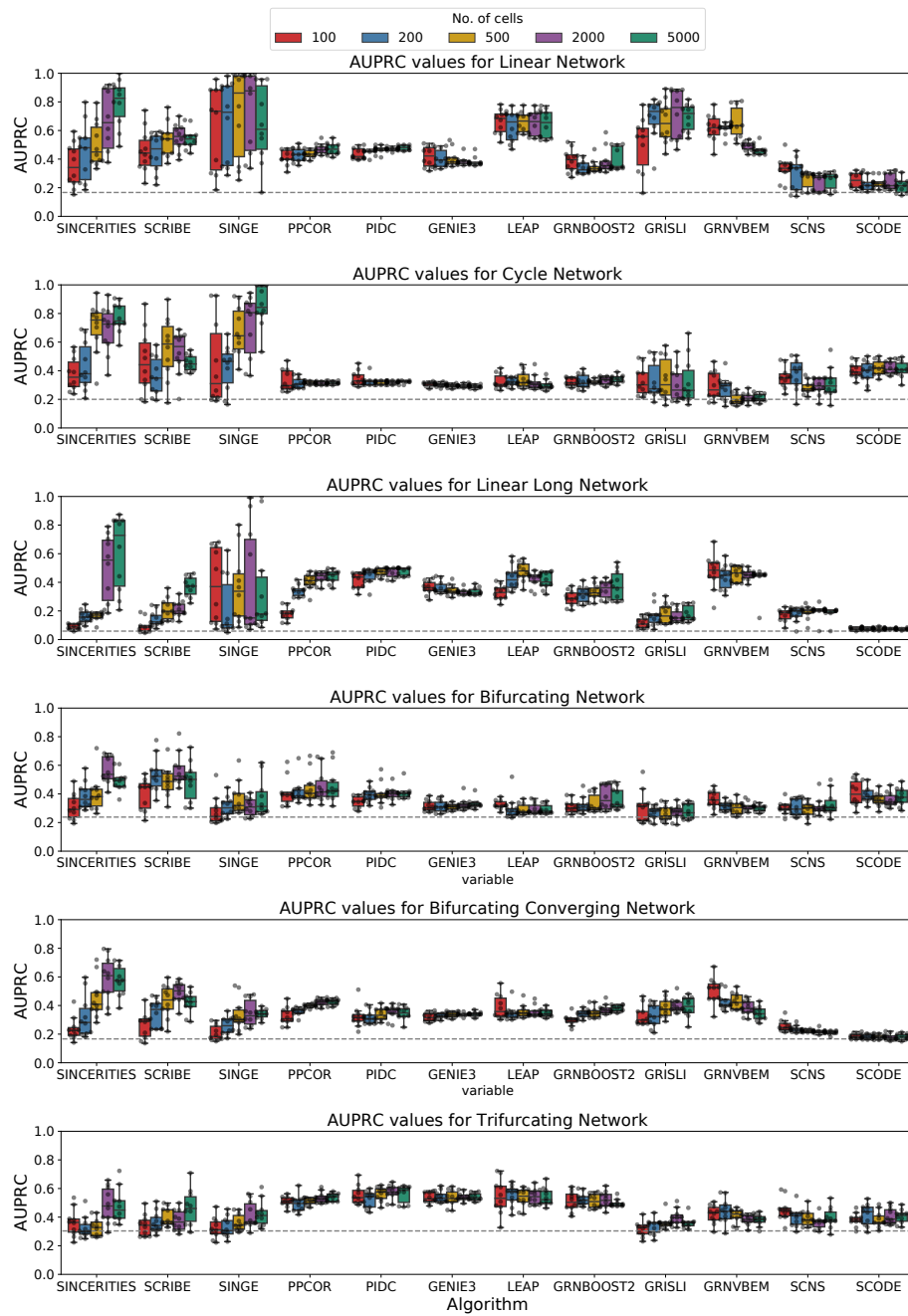


Figure 4.4: Box plots of AUPRC values. Each row corresponds to one of the six synthetic networks. Each column corresponds to an algorithm. Red, blue, green, purple and orange box plots corresponds to datasets with 100, 200, 500, 2,000, and 5,000 cells, respectively. The gray dotted line indicates the AUPRC value for a random predictor (network density).

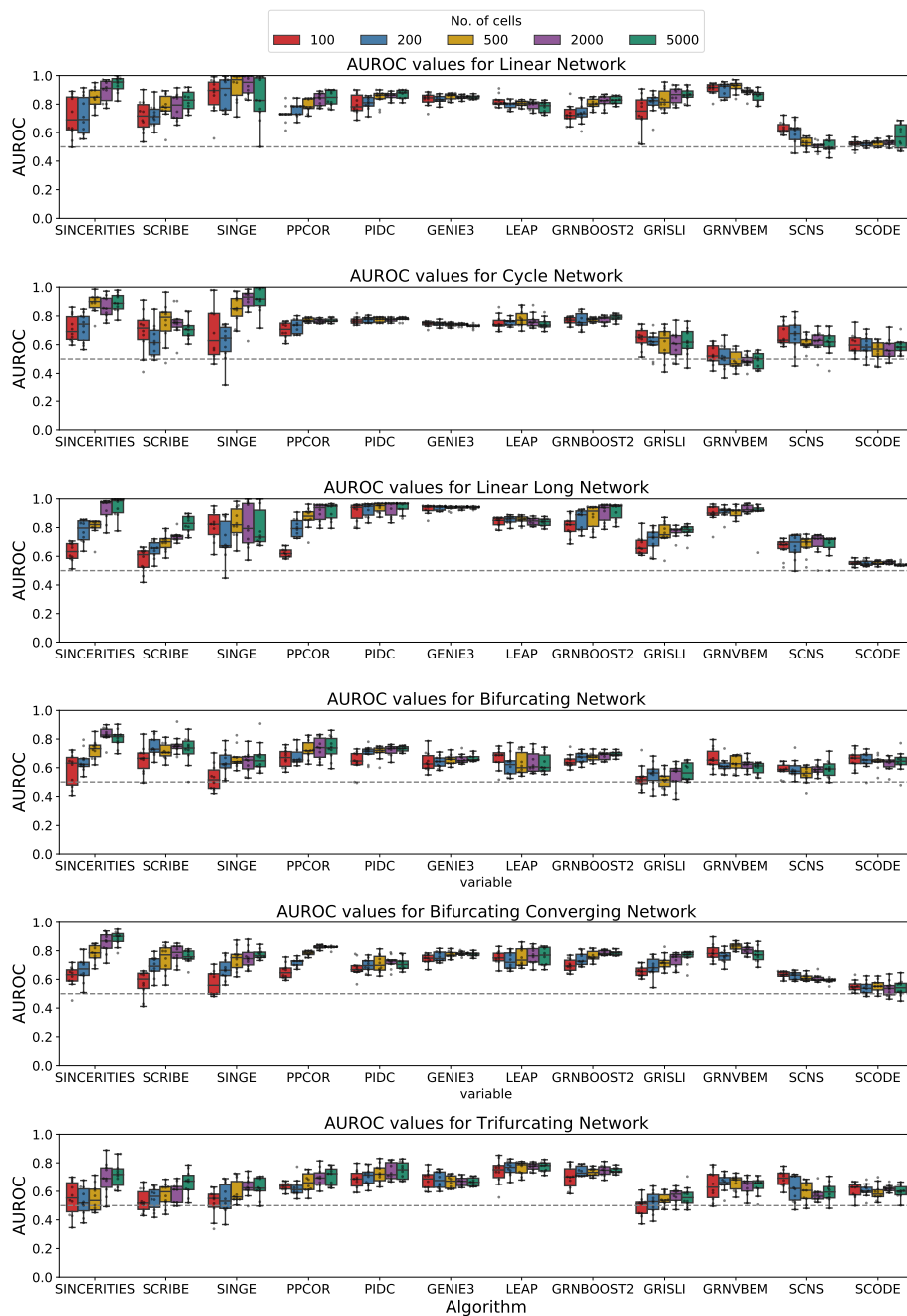


Figure 4.5: Box plots of AUROC values. Each row corresponds to one of the six synthetic networks. Each column corresponds to an algorithm. Red, blue, green, purple and orange box plots corresponds to datasets with 100, 200, 500, 2,000, and 5,000 cells, respectively. The gray dotted line indicates the AUROC value for a random predictor (0.5).

Datasets from curated models

Dense subnetworks of large-scale GRNs that have been used to generate simulated datasets of single-cell gene expression [52, 54, 55, 62, 65] may not capture the complex regulation in any specific developmental process. To avoid this pitfall, we selected four published Boolean models: mammalian cortical area development (mCAD) [87], ventral spinal cord development (VSC) [88], hematopoietic stem cell differentiation (HSC) [89], and gonadal sex determination (GSD) [90]. Figure 4.6(a) displays these networks.

We confirmed that the BoolODE-simulated datasets for each Boolean model (1) captured the same number of steady states as in the model (Figure 4.6(b)) and (2) matched the unique gene expression pattern that characterized each steady state of that model, as reported in the corresponding publication (Figure 4.6(c), Appendix B.2.1).

Encouraged by these results, we used BoolODE to create 10 different datasets with 2,000 cells for each model. We computed pseudotime using Slingshot [91] and provide these values to the GRN algorithms, so as to mimic a real single-cell gene expression data processing pipeline (Figure 4.6(d)). For each dataset, we also generated a version that included varying levels of dropouts (q) [52]. We obtained 30 datasets for each of the four models, 10 datasets with no dropout, 10 datasets with dropouts at $q = 50$, and 10 datasets with dropouts at $q = 70$. We provide more details on dataset generation in Section 4.4.3. We then ran each of the 12 GRN inference algorithms and analyzed each of the 30 reconstructions proposed by these methods for the four curated models.

Figure 4.7 summarizes our findings for the datasets without dropouts. Only four methods (gene regulation inference for single-cell with linear differential equations and velocity inference (GRISLI), SCODE, SINGE and SINCERITIES) had a median AUPRC ratio greater than one for the mCAD model. The reason may be the high density of the underlying network (Table 4.4). For the VSC model, which only has inhibitory edges, three methods (PIDC, GRNBoost2 and GENIE3) had an AUPRC ratio greater than 2.5. These three methods also had an AUPRC ratio close to 2.0 for the HSC model. PPCOR, GRISLI and SCRIBE tied for the highest median AUPRC ratio of 1.4 for the GSD model. Overall, GENIE3, GRNBoost2 and PIDC had among the highest median AUPRC ratios for two out of the four models. SINCERITIES, SCRIBE and SINGE, which were the best algorithms according to the AUPRC ratios for the datasets from synthetic networks, had a close to random median AUPRC ratio for all four curated models. We address this trend in Section 4.3.

Figures 4.8 and 4.9 show distributions of the AUPRC and AUROC values for all dropout rates. To study the effect of dropouts, for each algorithm, we compared the distributions of AUPRC scores across all Boolean models between $q = 0$ and $q = 50$ and between $q = 0$ and $q = 70$. Four and seven GRN inference methods had a statistically significant difference in AUPRC values for the 0–50 and the 0–70 comparison, respectively (Appendix B.2.3). The four algorithms that were unaffected by dropout rates (GRNVBEM, LEAP, SCRIBE and SINCERITIES) had worse-than-random AUPRC values on the mCAD and VSC datasets.

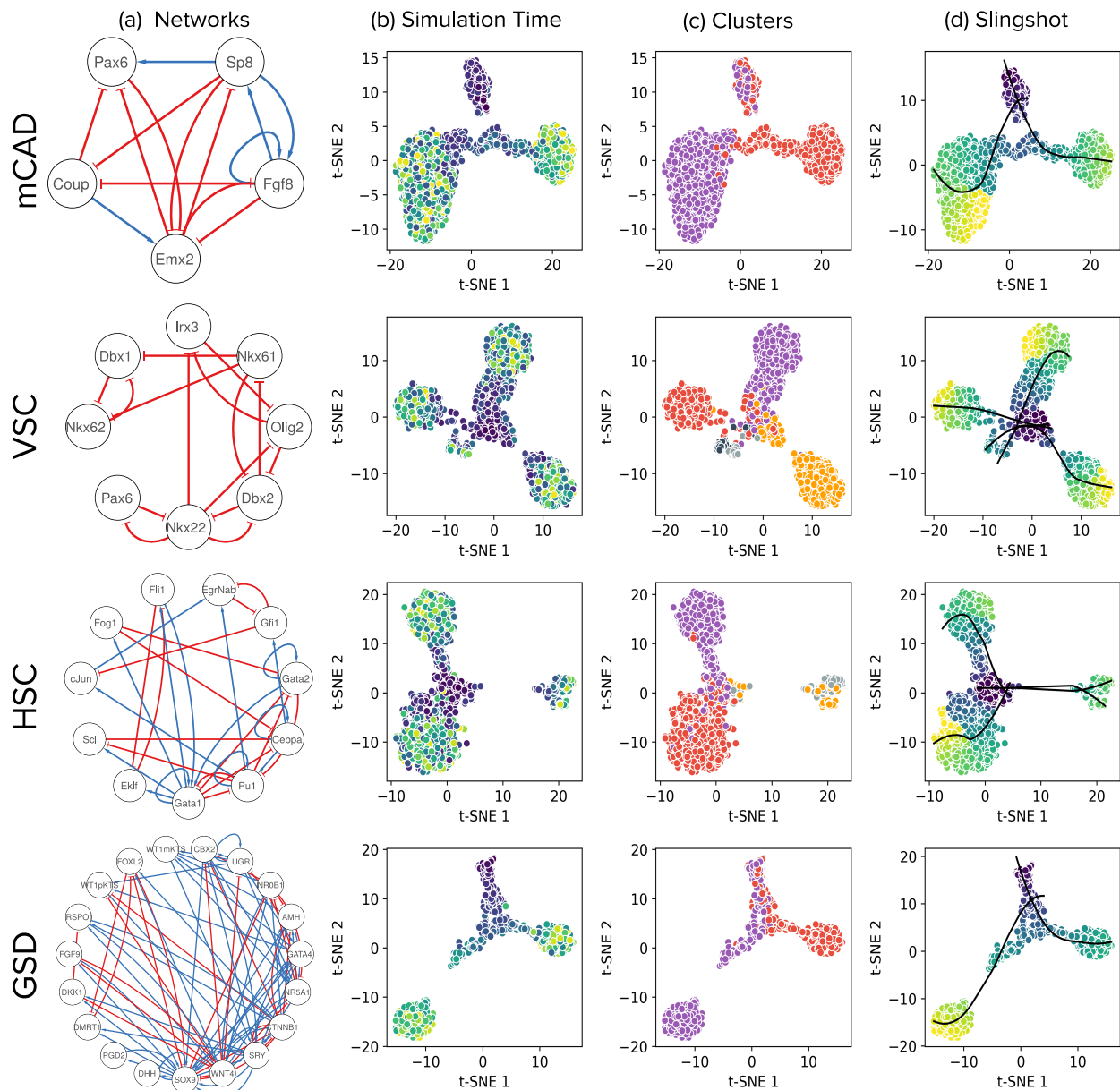


Figure 4.6: Visualization of t-SNE projections of simulations reveals trajectories leading to steady states that correspond to those of the curated models. Each row in the figure corresponds to a model, indicated on the left: mCAD, VSC, HSC differentiation and GSD determination. (a) Network diagrams of the models. (b) t-SNE visualizations of 2,000 cells sampled from the BoolODE output. The color of each point indicates the corresponding simulation time. (c) Each color corresponds to a different subset of cells obtained by using k-means clustering of simulations, with k set to the number of steady states reported in the relevant publication (two for mCAD, five for VSC, four for HSC and two for GSD). (d) Pseudotimes and principal curves (black) computed by Slingshot showing correspondence with simulation times in b and clusters in c, respectively. Colors of simulation time and pseudotime: blue for early, green for intermediate and yellow for later.

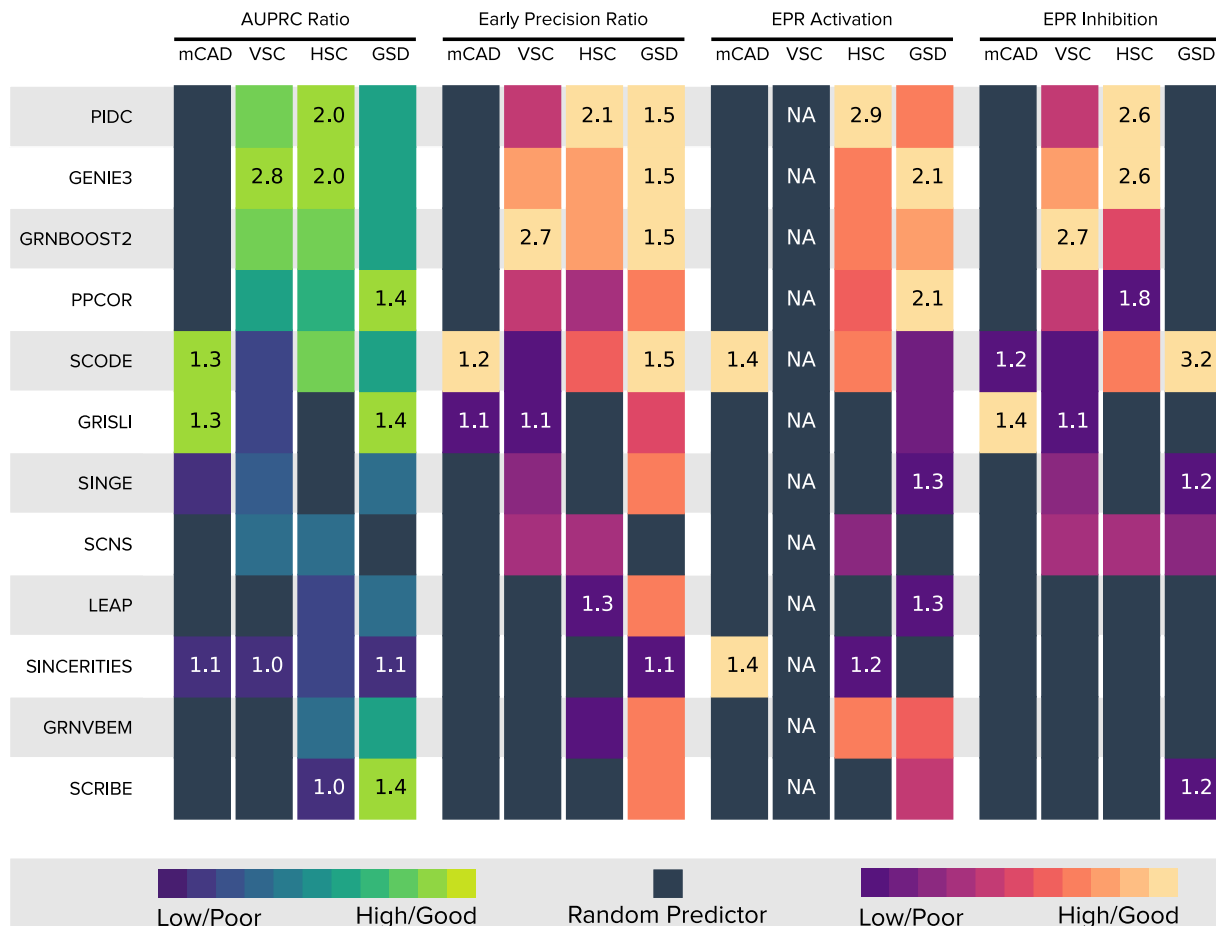


Figure 4.7: Summary of results for ten datasets without dropouts from curated models. Rows correspond to algorithms ordered by decreasing median of the per-model median AUPRC ratios. The four sets of four columns each display the median AUPRC ratios, median EPR, median EPR for activating edges and median EPR for inhibitory edges. For each model, the color in each cell is proportional to the corresponding value (scaled between 0 and 1, ignoring values that are less than that of a random predictor, shown as black squares). We display the highest and lowest values for each model inside the corresponding cells.

Next, we studied the early precision and early precision ratio (EPR) values of the top- k predictions (Section 4.4.4) for each of the four models. In at least one of the four models, 11 algorithms had a median EPR less than or close to one; that is, similar to a random predictor (black squares in Figure 4.7). In the 29 cases when the median EPR was at least one, it was 1.5 or larger only 16 times. The mCAD model had the smallest number of algorithms (two, GRISLI and SCODE) with median EPR larger than one. For datasets with dropouts, we did not see any clear trends in terms of smaller or larger early precision values compared to the dropout-free results (Figure 4.10).

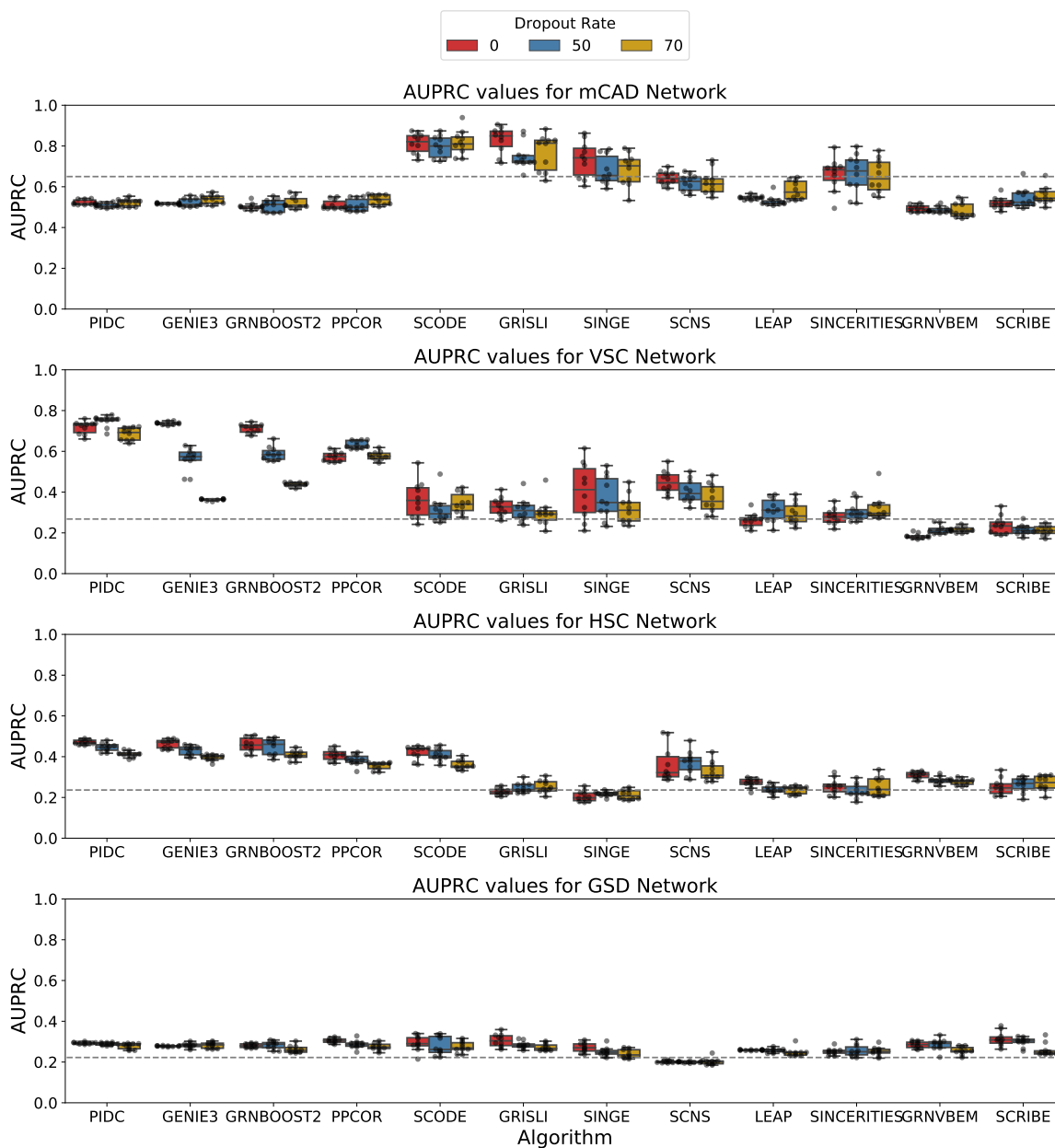


Figure 4.8: Box plot of AUPRC values. Each row corresponds to one of the four Boolean networks. Each column corresponds to an algorithm. Each box plot represents the 10 AUPRC values (10 datasets each containing 2,000 cells). Red, blue and green box plots correspond to datasets with no dropouts, a dropout rate of 50%, and a dropout rate of 70%, respectively. The gray dotted line indicates the AUROC value for a random predictor (network density).

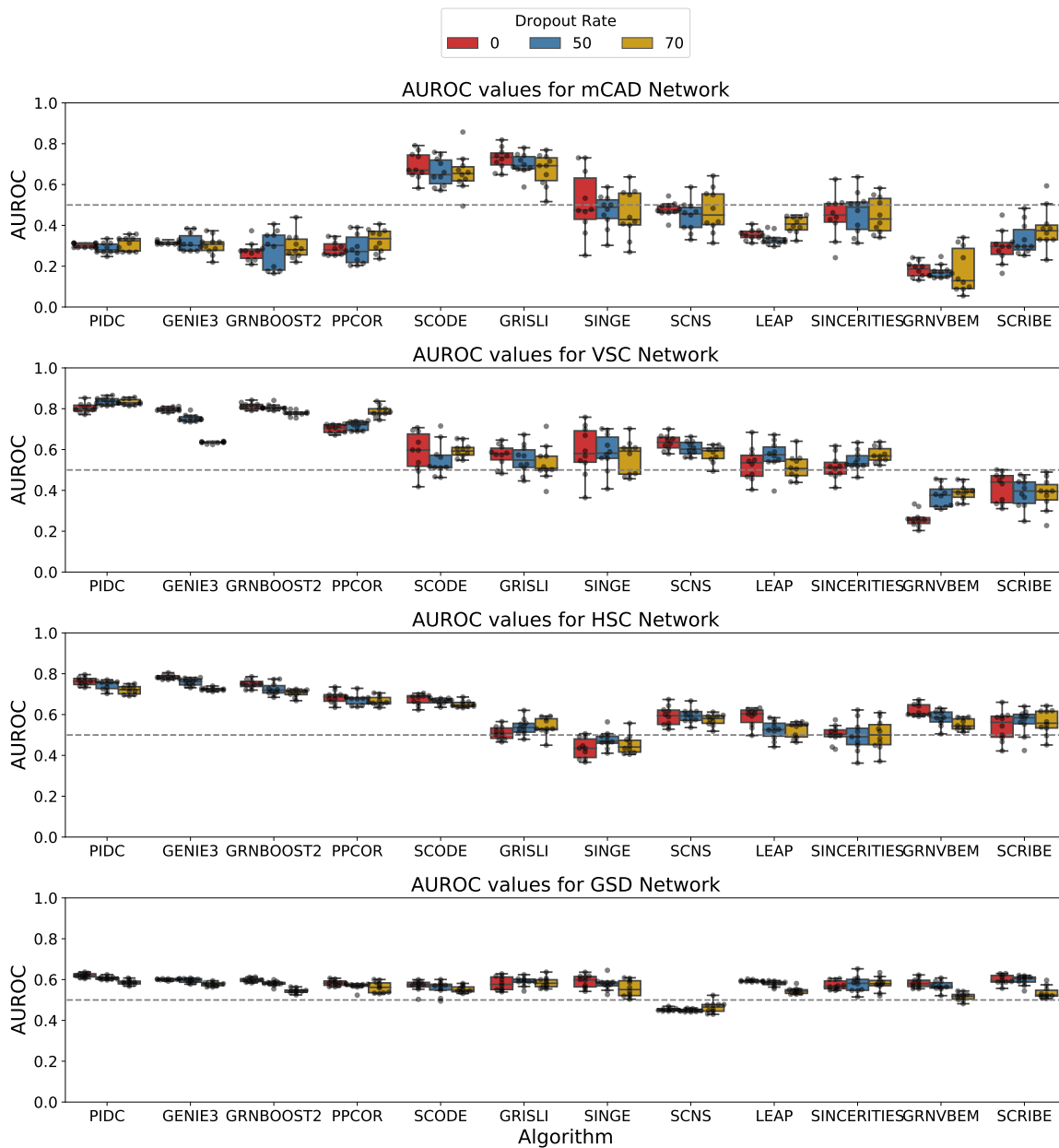


Figure 4.9: Box plot of AUROC values. Each row corresponds to one of the four Boolean networks. Each column corresponds to an algorithm. Each box plot represents the 10 AUROC values (10 datasets each containing 2,000 cells). Red, blue and green box plots correspond to datasets with no dropouts, a dropout rate of 50%, and a dropout rate of 70%, respectively. The gray dotted line indicates the AUROC value for a random predictor (0.5).

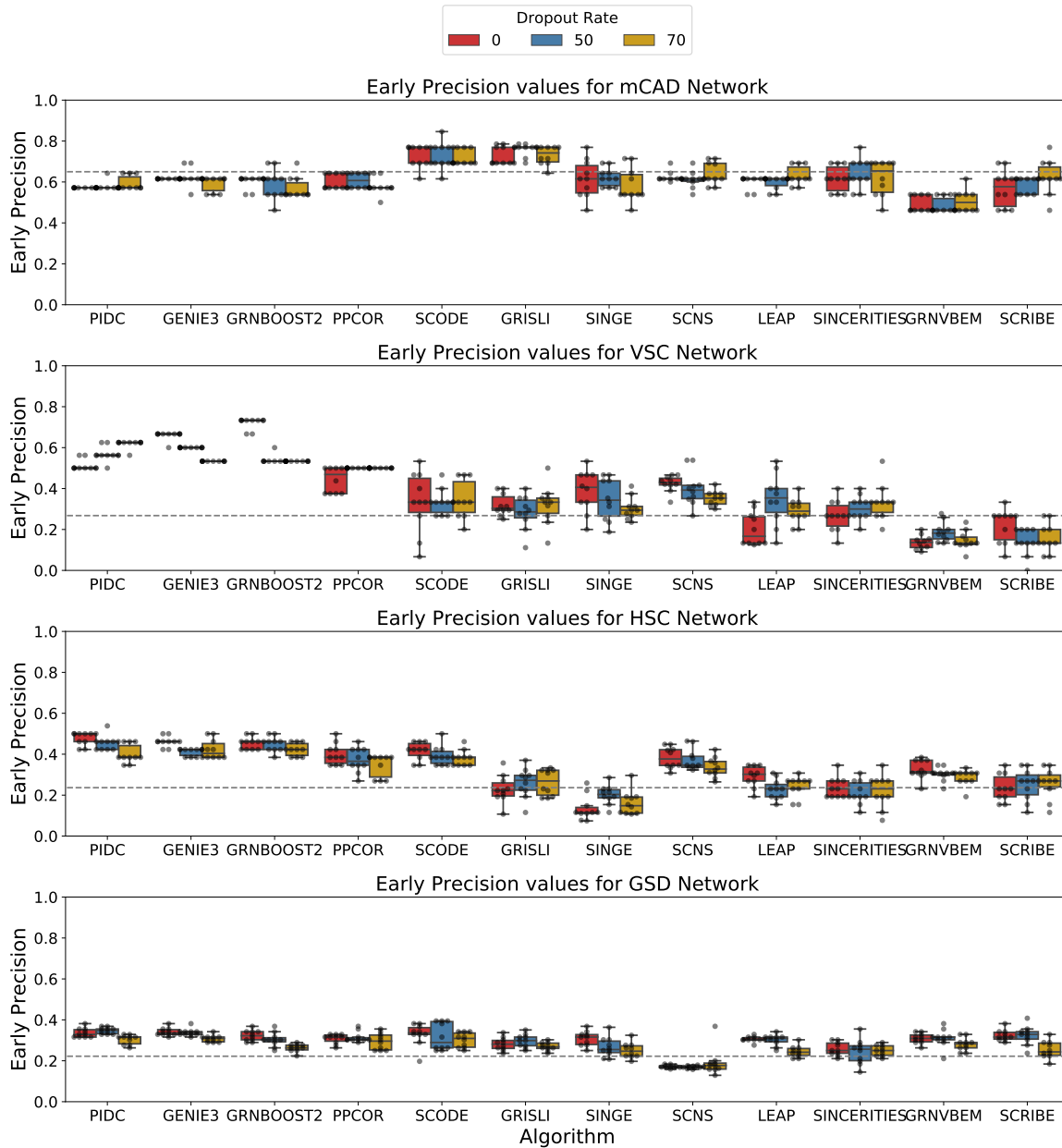


Figure 4.10: Box plot of early precision values. Each row corresponds to one of the four synthetic networks. Each column corresponds to an algorithm. Each box plot represents the 10 early precision values (10 datasets each containing 2,000 cells). Red, blue and green box plots correspond to datasets with no dropouts, a dropout rate of 50%, and a dropout rate of 70%, respectively. The gray dotted line indicates the baseline early precision value for a random predictor (network density).

We also investigated if the GRN inference methods were better at recovering activating edges or inhibitory edges. The mCAD model was again an outlier for EPR for activating and inhibitory edges, with only SCODE having slightly better-than-random scores for both. Overall, the GRN inference algorithms performed poorly when it comes to recovering the true edges within the top- k predictions.

We examined which algorithms produced similar reconstructions. For every model, the three best-performing methods (PIDC, GENIE3 and GRNBoost2) had similar outputs (Appendix B.2.4). In addition, LEAP and PPCOR were similar to the first three methods for the mCAD and GSD models. Pairwise similarities were poor for the other algorithms.

The GRNs formed by the top- k edges contained a higher than expected number of feedforward loops and lower than expected feedback loops and mutual interactions (Appendix B.2.5). Further, a very large fraction of false positives in the top- k edges corresponded to paths of length two in the ground-truth networks (Appendix B.2.6). This tendency to predict ‘indirect’ interactions could be the reason for the low EPR values.

Experimental single-Cell RNA-seq datasets

The final test of these algorithms is how well they can predict GRNs when applied to experimentally-derived single-cell gene expression datasets. We selected five published experimental single-cell RNA-seq datasets, two in human and three in mouse cells, comprising seven cell types [92, 93, 94, 95, 96]. We collected three different types of ground truth networks [97, 98, 99, 100]. We provide details on the RNA-seq data and the networks in Section 4.4.3. We evaluated and compared algorithms on running time and memory usage, the stability of the results to multiple executions, and accuracy in recovering interactions in the ground truth sets. We also compared the clusters present in the inferred GRNs with those computed directly from the gene expression data.

To measure the running time of the algorithms, we selected three cell types, namely, human mature hepatocytes (hHEP), human embryonic stem cells (hESCs) and erythroid-lineage mouse hematopoietic stem cells (mHSC-E), which contained different numbers of cells. We executed the algorithms on multiple subsets of highly varying genes in each dataset. We did not consider SCNS here since it took more than a day to complete for the 19-gene GSD Boolean model. We executed each of the remaining 11 algorithms on multiple subsets of highly-varying genes (10, 100, 500, 1,000, 2,000, and 5,000 genes) in three datasets, namely, mESCs, hESCs and mHSCs, which contain approximately 400, 750 and 1000 cells, respectively. Figure 4.12(a) summarizes our observations. Missing values indicate that the method either did not complete even after running for over a day or it gave a run-time error. For 5,000 genes, the running times ranged from minutes (PPCOR, SINCERITIES, SCODE) to hours (GRNBOOST2) to nearly a day (GENIE3). GRNVBEM, SCRIBE, and SCINGE were the slowest methods taking a few hours to nearly a day for 1,000 genes. There was little to no variation in the running times of these methods with increasing number of cells.

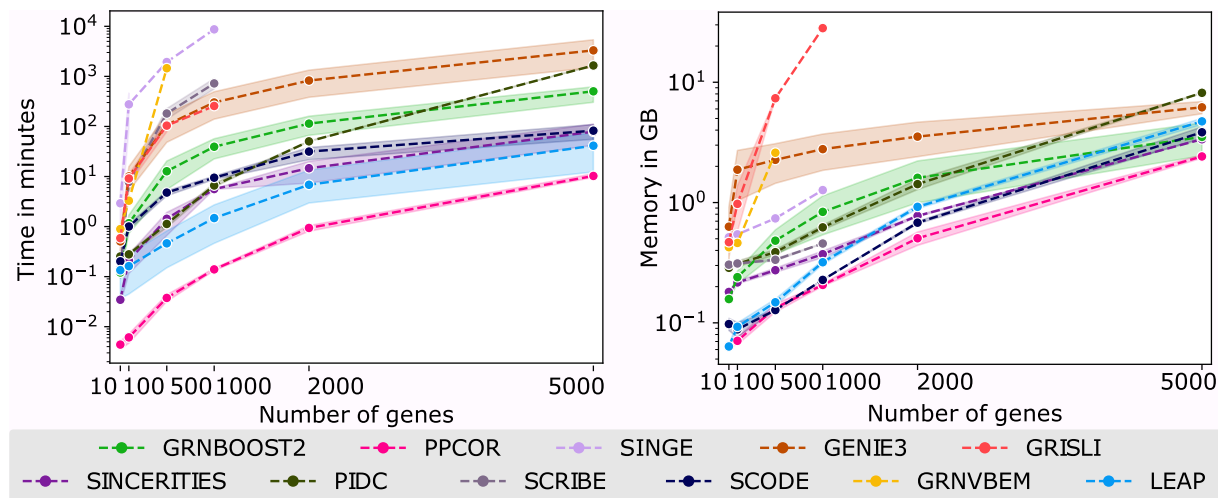


Figure 4.11: Scalability of GRN algorithms on experimental single-cell RNA-Seq datasets. Variation in running time and memory usage of GRN inference algorithms with respect to number of genes for three experimental single-cell RNA-Seq datasets. Each point represents the mean running time or memory across all three datasets and the shaded regions correspond to one standard deviation around the mean. Missing values indicate that the method either did not complete after one day or gave a runtime error. We did not consider SCNS since it took over a day on the 19-gene GSD Boolean model. We obtained these results on a computer with a 32-core 2.0GHz processor and 32GB of memory running Ubuntu 18.04.

The implementations of GENIE3, GRNBOOST2, and SCINGE can take advantage of multi-threading, with the wall-clock times of these methods being lowered by a factor proportional to the number of threads available. In terms of memory usage, most of the algorithms did not require more than 6GB of RAM. We were unable to run PIDC for 5,000 genes and GRISLI for over 500 genes due to memory exceptions. We obtained these results on a computer with a 32-core 2.0 GHz processor and 32GB of memory running Ubuntu 18.04.

For further analysis, we selected the five algorithms with the highest median AUPRC in each of the earlier two datasets: SINCERITIES, SCRIBE, SINGE, PPCOR and PIDC for synthetic networks and PIDC, GENIE3, GRNBoost2, PPCOR and SCODE for Boolean models; PIDC and PPCOR were in both sets. We did not retain SINGE and SCRIBE because of the time taken for parameter search.

For each experimental single-cell RNA-seq dataset, we created four subsets of genes: (a, b) containing all the significantly-varying TFs and either the 500 or 1,000 most varying genes and (c, d) only the 500 or 1,000 most varying genes. We intersected each ground truth network with each of these sets of genes. We observed that the densities of the resulting networks varied in a wide range, reaching as high as 0.38 for cell-type specific ChIP-Seq datasets and as low as 0.01 for non-cell-type specific ChIP-Seq networks (columns titled “Network statistics” in Figure 4.12). We applied each of the seven selected algorithms to

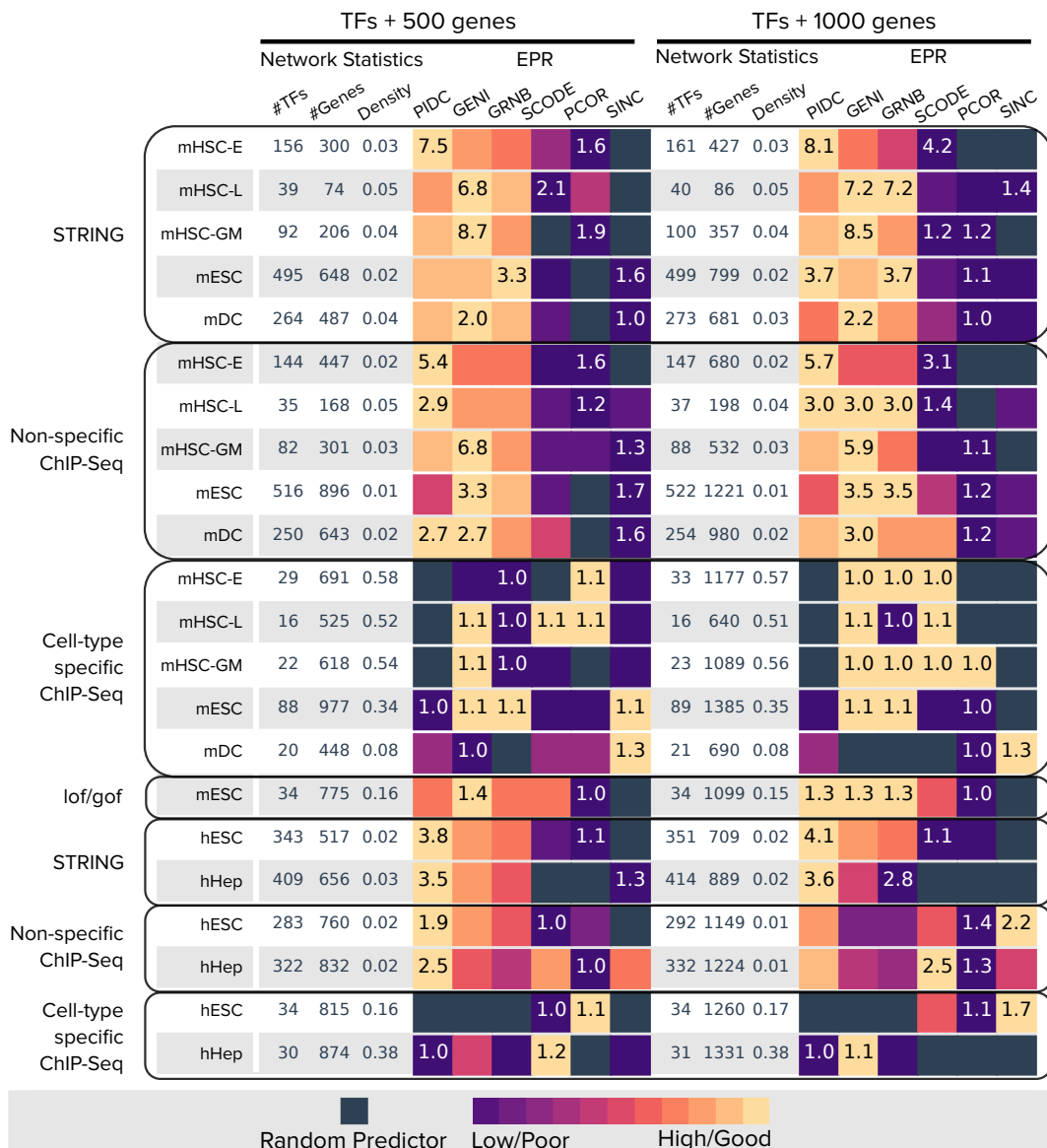


Figure 4.12: Summary of EPR results for experimental single-cell RNA-seq datasets with TFs+500 and TFs+1000 genes. The left half of the figure (TFs+500 genes) shows results for datasets composed of all significantly varying TFs and the 500 most-varying genes. The right half of the figure (TFs + 1000 genes) shows results for datasets composed of all significantly varying TFs and the 1000 most-varying genes. Each row corresponds to one of scRNA-seq dataset-network combination. A blackened square in the heatmap represents a value that is less than that of a random predictor. The EPR columns are sorted by the median EPR achieved across all rows for TFs + 500 dataset. In both sections, algorithm Abbreviations: GENI, GENIE3; GRNB, GRNBoost2; lof/gof, loss-of-function/gain-of-function; PCOR, PPCOR and SINC, SINCERITIES.

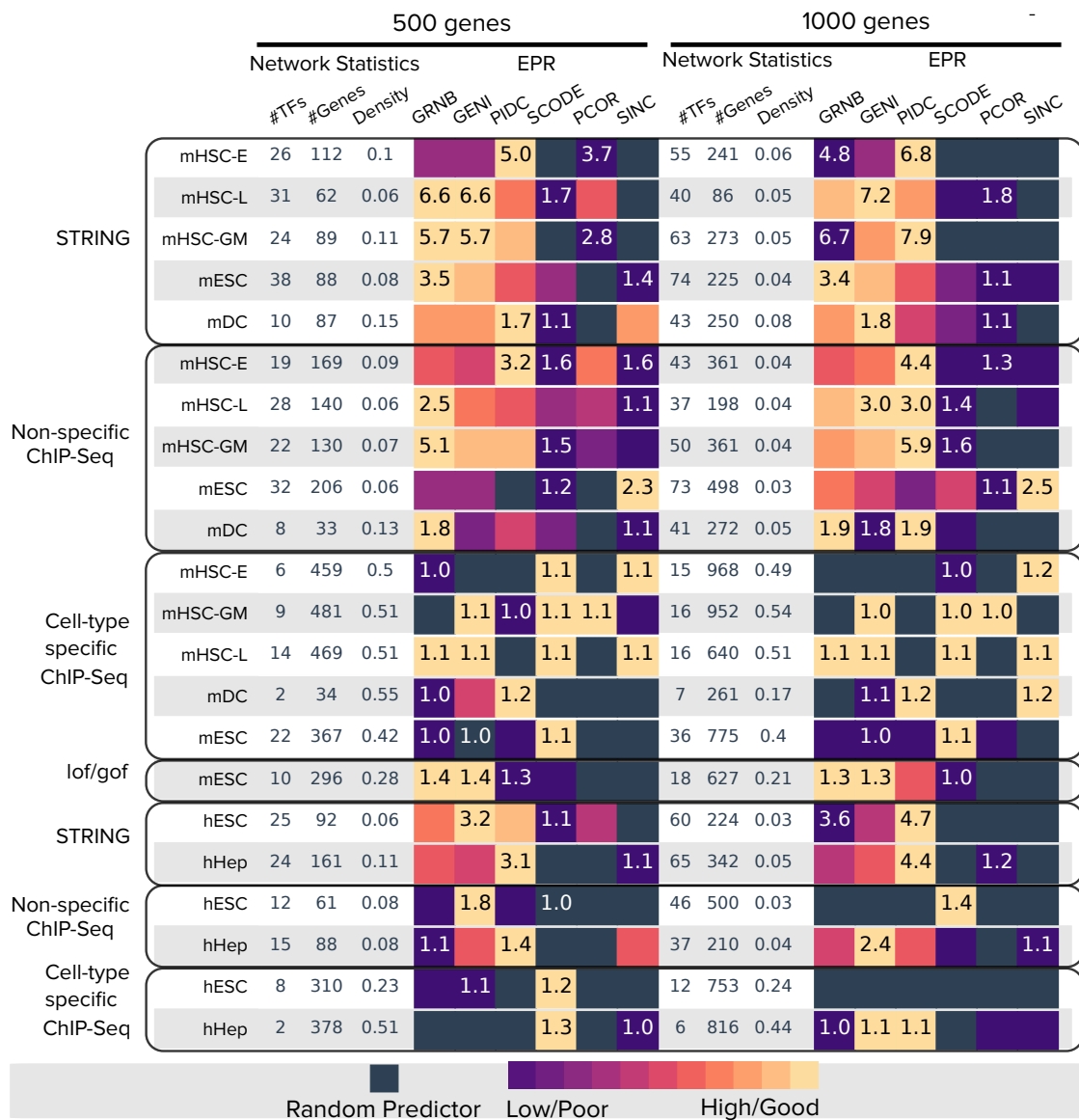


Figure 4.13: Summary of EPR results for experimental single-cell RNA-seq datasets with 500 and 1000 genes. The left half of the figure (500 genes) shows results for datasets composed of the 500 most-varying genes and TFs. The right half of the figure (1000 genes) shows results for datasets composed of the 1000 most-varying genes and TFs. Each row corresponds to one of scRNA-seq dataset-network combination. A blackened square in the heatmap represents a value that is less than that of a random predictor. The EPR columns are sorted by the median EPR achieved across all rows for 500 dataset. In both sections, algorithm Abbreviations: GENI, GENIE3; GRNB, GRNBoost2; lof/gof, loss-of-function/gain-of-function; PCOR, PPCOR and SINC, SINCERITIES.

infer a GRN for each experimental single-cell RNA-seq dataset. We compared the algorithms based on the EPR, reasoning that predicted interactions of higher confidence will be more interesting to experimentalists. We used the top- k networks, setting k equal to the number of edges in the corresponding reduced ground-truth dataset. We performed parameter search to optimize the EPR (Appendix B.3.1).

GENIE3, PIDC and GRNBoost2 were the three methods with the highest EPR values for experimental datasets (Figure 4.12 and Figure 4.13), just as they were for curated models (Figure 4.7). When we computed modules in the GRNs output by these methods, we found that they had a good concordance with clusters determined directly from the gene expression data (Appendix B.3.2). PPCOR, which was consistently in the top-four methods for both synthetic networks and curated models, had only a slightly better-than-random EPR across all experimental datasets. PPCORs AUPRC decreased moderately as we increased dropouts in datasets from curated models (Figure 4.8). We speculate that dropouts in experimental single-cell RNA-seq datasets had a severe effect on PPCOR.

To examine the effect of the number of genes, as well as including all significantly varying TFs, we evaluated the three top performing methods, PIDC, GENIE3 and GRNBoost2, on the nonspecific ChIP-seq and STRING networks. We found that the median EPR had a statistically significant improvement when we included all significantly varying TFs in the analysis (Appendix B.3.3). However, the addition of highly varying genes (500 versus 1,000) did not lead to a significant improvement in EPR values. The AUPRC ratio did not vary with the number of genes.

In terms of the ground-truth network, we observed that the performance of almost all the methods was similar to that of a random predictor when we evaluated them on cell-type specific ChIP-Seq networks. While the corresponding network densities varied within a large range (from 0.08 to 0.57), the EPR values did not exceed 1.5 (for SCODE and SINCERITIES on the hESC dataset). In contrast, for non-specific ChIP-Seq networks with comparable densities, the EPR was as high as 2.9 (PIDC on the mHSC dataset). The performance of algorithms on the loss-or-gain of function (lof/gof) reference network was similar to that on the cell-type specific ChIP-Seq networks, with most values being close to 1.

Almost every algorithm produced nearly identical outputs when we ran it on the same dataset multiple times (Supplementary Note 3.4). However, for the same dataset, the results varied substantially from one algorithm to another (Supplementary Note 3.5), in contrast to curated models where the top performing methods yielded similar results (Appendix B.3.4). Moreover, ensembles of the algorithm outputs did not perform systematically better than the best method for each dataset (Appendix B.3.5). This result stands in contrast to the success of ensembles in inferring GRNs from bulk transcriptional data [101].

4.3 Discussion

We have presented BEELINE, a framework for benchmarking algorithms that infer GRNs from single-cell gene expression data. Although a few reviews of GRN inference methods have appeared [102, 103, 104], there has been only one quantitative evaluation of these approaches [65]. This paper compared eight approaches of which six were originally developed or use methods created for bulk transcriptomic data. In contrast, we include eight new algorithms that have been developed exclusively for single-cell gene expression data while not including some of the previously-compared approaches for reasons given in “Methods”. Most of the techniques we have considered have been developed since 2017. The earlier comparison used GeneNetWeaver to create simulated datasets. Thus, our evaluation is up-to-date and timely while also being more relevant for single-cell data.

Figure 4.14 summarizes the properties of the GRN inference algorithms (columns two to six) and the key insights from this study (column seven onwards). We found considerable variation in the performance of the algorithms across the ten networks (six synthetic and four Boolean) and five experimental scRNA-seq datasets that we analyzed. Nevertheless, we noted a few trends. In general, the synthetic networks were easier to recover than the Boolean models. This trend may be due to the fact that the synthetic networks have simpler and more well-defined trajectories, which our simulations capture accurately, than the Boolean models. For datasets from curated models, each of which has multiple trajectories, we found that methods that do not require pseudotime information (specifically, GENIE3, GRNBoost2, and PIDC) performed the best. For the other techniques, we input pseudotime values for each trajectory independently. We found that these approaches succeeded in discovering mutual inhibition loops, with each arm of the loop inferred from a different trajectory. Methods that performed well for datasets from curated models also inferred GRNs of good accuracy for experimental single-cell RNA-seq datasets. Nevertheless, the overall performance of these approaches was much less than ideal. Note that three methods (GRISLI, SCINGE, and SCRIBE) have appeared only as preprints. Their final published versions may show improved performance.

A surprising trend we observed was that the best performing algorithms for datasets from synthetic networks (SINCERITIES, SCRIBE, and SCINGE) had very poor results on datasets from curated models (ranks 10, 12, and 7, respectively); SCRIBE and SINCERITIES had close to or worse-than-random EPRs on experimental single-cell RNA-seq datasets data as well (Figure 4.12). To investigate the reason for this reversal of fortunes, we repeated the analysis for datasets from synthetic networks with single trajectories with the original and with shuffled pseudotimes (see “Shuffling simulation times” in Section 4.4.4 for details). We only considered the seven methods that required pseudotimes, ignoring SCNS for reasons given earlier. We observed a general decrease in performance with an increase in the size of the window over which we shuffled the pseudotime values, with the effect being most pronounced for SINCERITIES, SCRIBE, and SCINGE (column titled “Pseudotime” in Figures 4.14 and 4.15). This analysis suggests that these algorithms may be very sensitive to

accurate pseudotime imputation. We expect that the accuracy of the GRNs inferred by these method will improve as the techniques for pseudotime imputation become better or through the use of other surrogates for experimental time such as RNA velocity [105].

		Properties				Accuracy			Stability			Scalability (genes)							
Category	Addl. Inputs	Time ordered?	Directed?	Signed?	Synthetic	Curated	scRNA-Seq	Datasets	Runs	Dropouts	Pseudotime	Time				Memory			
												100	500	1k	2k	100	500	1k	2k
PIDC	MI	-	X	X	X	High	High	High	High	High	High	1s	1m	5m	30m	0.1G	0.1G	0.5G	1G
GENIE3	RF	-	X	✓	X	High	High	High	High	High	High	5m	1h	3h	12h	1G	2G	2G	2G
GRNBOOST2	RF	-	X	✓	X	High	High	High	High	High	High	1m	10m	30m	1h	0.1G	0.1G	0.5G	1G
SCODE	ODE+Reg	ODE parameters	✓	✓	✓	High	High	High	High	High	High	1m	5m	5m	30m	1M	0.1G	0.1G	0.5G
PPCOR	Corr	-	X	X	✓	High	High	High	High	High	High	1s	1s	1s	1s	1M	0.1G	0.1G	0.1G
SINCERITIES	Reg	-	✓	✓	✓	High	High	High	High	High	High	1s	1m	5m	10m	0.1G	0.1G	0.1G	0.5G
SCRIBE	MI	Type of RDI	✓	✓	X	High	High	High	High	High	High	5m	2h	6h	-	0.1G	0.1G	0.1G	-
SINGE	GC	Regression parameters	✓	✓	X	High	High	High	High	High	High	3h	>1d	>1d	-	0.5G	0.5G	1G	-
LEAP	Corr	Lag	✓	✓	X	High	High	High	High	High	High	1s	1s	1m	5m	1M	0.1G	0.1G	0.5G
GRISLI	ODE+Reg	Regression parameters	✓	✓	X	High	High	High	High	High	High	5m	1h	3h	-	0.5G	>4G	>4G	-
GRNVBEM	Reg	-	✓	✓	✓	High	High	High	High	High	High	1m	>1d	-	-	0.1G	2G	-	-
SCNS	Bool	Boolean model parameters	✓	✓	✓	High	High	High	High	High	High	-	-	-	-	-	-	-	-

Low/Poor High/Good

Low/Poor High/Good

Figure 4.14: Summary of properties of GRN algorithms and results from BEELINE. Each row corresponds to one of the algorithms included in our evaluation. See text for more details. Abbreviations: MI: Mutual Information, RF: Random Forest, Corr: Correlation, Reg: Regression, GC: Granger Causality, Bool: Boolean Model.

Despite the widely varying performance of GRN inference algorithms from one ground truth network to another, we can make some specific recommendations for end users seeking to apply these methods to an experimental single-cell RNA-seq dataset.

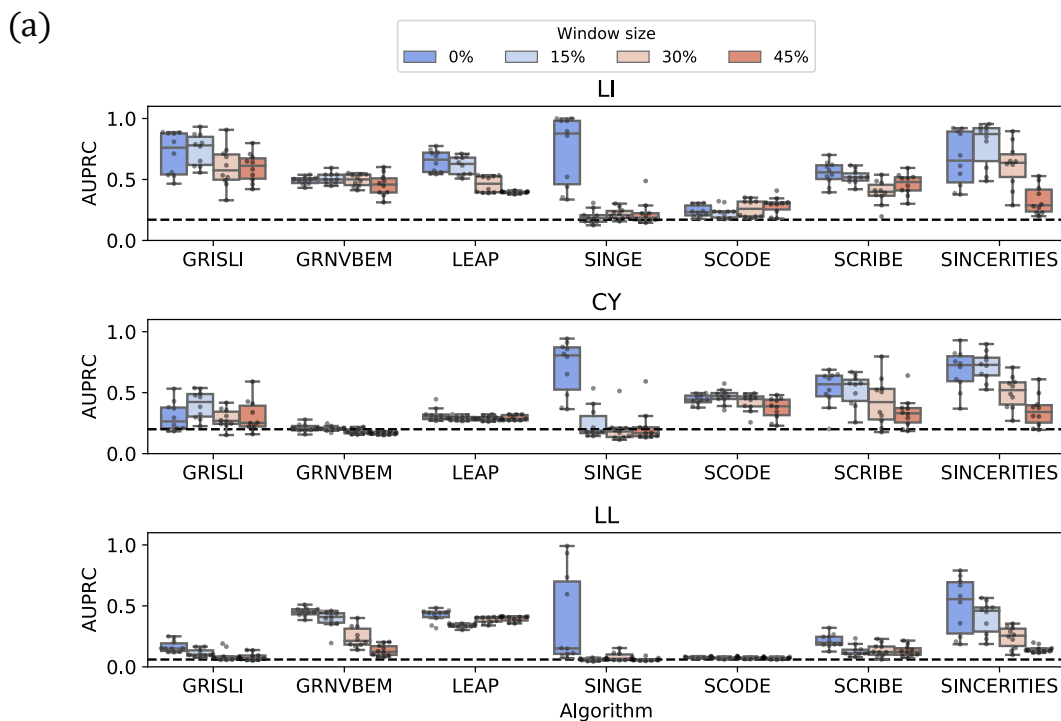
- (i) PIDC, GENIE3, and GRNBoost2 were leading and consistent performers in our evaluations of datasets from curated models and experimental single-cell RNA-seq datasets. GENIE3 and PIDC have better stability across multiple runs whereas GRNBoost2 is less sensitive to the presence of dropouts. While neither of these methods can take advantage of pseudotime-ordered cells in the input, they are also immune to any errors incurred during pseudotime computation.

- (ii) LEAP, GENIE3, and GRNBoost2 may be suitable in scenarios where we do not expect too many feedforward loops in the true regulatory network.
- (iii) If efficiency is important, PIDC and GRNBoost2 run considerably faster than GENIE3. In addition, as the quality of pseudotime inference improves, SINCERTIES may also be a good choice, especially since it is also quite stable to multiple runs and presence of dropouts.
- (iv) The strategy for selection of genes on which to infer GRNs is an important consideration. Our results on experimental single-cell RNA-seq datasets suggest that adding more highly-varying genes (1,000 rather than 500) and/or considering all significantly-varying TFs contribute to significant improvements in the EPR of the best-performing algorithms. However, there is no effect on AUPRC. This step merits further analysis in the future.

Inference of GRNs has been active area of research for more than 20 years. Our evaluation shows that GRN inference remains a challenging problem despite the variety of innovative ideas that researchers have brought to bear upon the problem. There could be several factors that cause the poor performance that we have observed. First, even though single-cell RNA-seq techniques have advanced rapidly both in the number of cells that can be measured and in the depth of coverage [106], these methods may not still provide sufficient resolution and variation in expression for the reliable and robust inference of GRNs. Second, there may be inherent shortcomings to the underlying assumption that statistical relationships between expression patterns correspond to regulatory interactions.

In this context, we observed that false positive edges form feedforward loops when added to ground truth networks. There is a history of GRN inference algorithms that explicitly try to avoid indirect effects [56, 107, 108, 109], using techniques such as partial correlation coefficients [107], the data processing inequality [108, 109], and conditioning the score of a pair of genes on all other genes [56]. We suggest that new ideas for avoiding the prediction of such interactions may assist in improving the accuracy of GRN inference algorithms for single cell gene expression data. To this end, it may be important to integrate additional types of data such as known transcription factor binding sites or ChIP-seq measurements in order to distinguish between direct and indirect interactions, e.g., as done in the SCENIC algorithm [53]. Finally, a target gene’s expression level may change even if the regulating TF does not vary in abundance, e.g., the TF may translocate to the nucleus in order to carry out its regulatory program. Recent approaches that interrogate single cells along multiple modalities [110, 111] and integrate these data may be critical for the next generation of GRN inference algorithms.

BoolODE was a critical component of our analysis. We developed BoolODE after studying the performance on GRN algorithms on real single-cell RNA-seq datasets; when we examined the results in the literature [48, 51, 54, 59, 60], we found that the AUROC or precision at early recall values of these reconstructions were quite poor, often being close to that of a



(b)

Network	Window size	Correlation
LI	15%	0.99
	30%	0.96
	45%	0.90
CY	15%	0.99
	30%	0.96
	45%	0.90
LL	15%	0.99
	30%	0.96
	45%	0.90

Figure 4.15: (a) Comparison of AUPRC values for networks inferred with original pseudotimes (blue) and shuffled within three window sizes 15% (light blue), 30% (beige) or 45% (orange). Each row corresponds to one of the three synthetic networks with single trajectories (Linear, Cycle, and Linear Long). Each box plot represents 10 AUPRC values (10 datasets each containing 2,000 cells). The gray dotted line indicates the AUPRC value for a random predictor (i.e., the network density). In each box plot, the box shows the 1st and 3rd quartile, and whiskers denote 1.5 times the interquartile range. (b) Median Pearson correlation between original and shuffled pseudotimes. For each network and window-size the median is over 10 datasets.

random predictor; our results are in line with these previous reports. Therefore, we reasoned that it would be valuable to the community to benchmark GRN algorithms by applying them to carefully simulated Boolean models with predictable or computable trajectories. BoolODE is very successful at this task and promises to be useful as an independent tool. Future developments on simulators such as BoolODE can include more general frameworks for constructing ODE equations out of Boolean rules, fully-automated verification of steady states, increasing its robustness to variations in parameters, and extensions for generating large-scale datasets.

As single-cell experiments become more complex, we expect that transcriptional trajectories will also be more intricate, perhaps involving multiple stages of bifurcation and/or cycling. A key challenge that lies ahead is accurately computing GRNs that give rise to such complex trajectories. We hope that scientists will use BEELINE in conjunction with BoolODE as they develop new approaches for GRN inference.

4.4 Methods

4.4.1 Regulatory network inference algorithms

We briefly describe each algorithm we have included in this evaluation. We have ordered the methods chronologically by year and month of publication. Every software package had an open source license, other than GRNVBEM and GRISLI, which did not have any license.

- 1. GENIE3 (Gene Network Inference with Ensemble of trees) [48].** Developed originally for bulk transcriptional data, GENIE3 computes the regulatory network for each gene independently. It uses tree-based ensemble methods such as Random Forests to predict the expression profile of each target gene from profiles of all the other genes. The weight of an interaction comes from the importance of an input gene in the predictor for a target gene's expression pattern. Aggregating these weighted interactions over all the genes yields the regulatory network. This method was the top performer in the DREAM4 In Silico Multifactorial challenge.

- 2. PPCOR [49].** This R package computes the partial and semi-partial correlation coefficients for every pair of variables (genes, in our case) with respect to all the other variables. It also computes a p -value for each correlation. We use this package to compute the partial coefficients. Since these values are symmetric, this method yields an undirected regulatory network. We use the sign of the correlation, which is bounded between -1 and 1, to signify whether an interaction is inhibitory (negative) or activating (positive).

- 3. LEAP (Lag-based Expression Association for Pseudotime-series) [50].** Starting with pseudotime-ordered data, LEAP calculates the Pearson's correlation of normalized mapped-read counts over temporal windows of a fixed size with different lags. The score

recorded for a pair of genes is the maximum Pearson's correlation over all the values of lag that the method considers. The software includes a permutation test to estimate false discovery rates. Since the correlation computed is not symmetric, this method can output directed networks.

4. SCODE [51]. This method uses linear Ordinary Differential Equations (ODEs) to represent how a regulatory network results in observed gene expression dynamics. SCODE relies on a specific relational expression that can be estimated efficiently using linear regression. In combination with dimension reduction, this approach leads to a considerable reduction in the time complexity of the algorithm.

5. PIDC (Partial Information Decomposition and Context) [52]. This method uses concepts from information theory. For every pair of genes x and y , given a third gene z , the method partitions the pairwise mutual information between x and y into a redundant and a unique component. It computes the ratio between the unique component and the mutual information. The sum of this ratio over all other genes z is the proportional unique contribution (PUC) between x and y . The method then uses per-gene thresholds to identify the most important interactions for each gene. The resulting network is undirected since the PUC is symmetric.

6. SINCERITIES (SINgle CELL Regularized Inference using TIme-stamped Expression profileS) [54]. Given time-stamped transcriptional data, this method computes temporal changes in each gene's expression through the distance of the marginal distributions between two consecutive time points using the KolmogorovSmirnov (KS) statistic. To infer regulatory connections between TFs and target genes, the approach uses Granger causality, i.e., it uses the changes in the gene expression of TFs in one time window to predict how the expression distributions of target genes shift in the next time window. The authors formulate inference as a ridge regression problem. They infer the signs of the edges using partial correlation analyses.

7. SCNS [5]. This method takes single-cell gene expression data taken over a time course as input and computes logical rules (Boolean formulae) that drive the progression and transformation from initial cell states to later cell states. By design, the resulting logical model facilitates the prediction of the effect of gene perturbations (e.g., knockout or overexpression) on specific lineages.

8. GRNVBEM [55]. This approach infers a Bayesian network representing the gene regulatory interactions. It uses a first-order autoregressive model to estimate the fold change of a gene at a specific time as a linear combination of the expression of the gene's parents in the Bayesian network at the previous time point. It infers the gene regulatory network within a variational Bayesian framework. This method can associate signs with its directed edges.

9. SCRIBE [56]. Similar to PIDC, this approach uses ideas from information theory. The relevant concept here is conditioned Restricted Directed Information (cRDI), which measures

the mutual information between the past state (expression values) of a regulator and the current state of a target gene conditioned on the state of the target at the previous time point. To obtain efficiency for large datasets, the authors use an unconditioned version called uRDI, followed by the Context Likelihood of Relatedness (CLR) algorithm [109] to remove edges that correspond to indirect effects. We used this strategy for experimental single-cell RNA-seq datasets.

10. GRNBoost2 [57]. GRNBoost2 is a fast alternative for GENIE3, especially suited for datasets with tens of thousands of observations. Like GENIE3, GRNBoost2 trains a regression model to select the most important regulators for each gene in the dataset. GRNBoost2 achieves its efficiency by using stochastic Gradient Boosting Machine regression with early-stopping regularization to infer the network.

11. GRISLI [58]. Like SCODE, this approach uses a linear ODE-based formalism. GRISLI estimates the parameters of the model using different ideas. Taking either the experimental time of the cells or estimated pseudotime as input, it first estimates the velocity of each cell, i.e., how each genes expression value changes as each cell undergoes a dynamical process [105]. It then computes the structure of the underlying gene regulatory network by solving a sparse regression problem that relates the gene expression and velocity profiles of each cell.

12. SCINGE (Single-Cell Inference of Networks using Granger Ensembles) [59]. The authors observe that while many gene inference algorithms start by computing a pseudotime value for each cell, the distribution of cells along the underlying dynamical process may not be uniform. To address this limitation, SCINGE uses kernel-based Granger Causality regression to alleviate irregularities in pseudotime values. SCINGE performs multiple regressions, one for each set of input parameters, and aggregates the resulting predictions using a modified Borda method.

Other methods. We now discuss other papers on this topic and our rationale for not including them in the comparison. We did not consider a method if it was supervised [112] or used additional information, e.g., a database of transcription factors and their targets [53], a lineage tree [61].

We did not include methods that output a single GRN without any edge weights [62, 86], since any such approach would yield just a single point on a precision-recall or curve.

4.4.2 BoolODE: Converting boolean models to Ordinary Differential Equations

We start this section by giving an overview of GeneNetWeaver [85, 113], a popular method for simulating bulk gene expression datasets. It is being used for simulating single cell transcriptional data as well [52, 54, 55, 62, 65]. Next, we describe the BoolODE framework that we have developed and highlight its differences with GeneNetWeaver. We compare the

data simulated by BoolODE and GeneNetweaver to demonstrate that the latter approach does not yield expected trajectories. We end this section by summarizing BoolODE and the reasons we prefer it over GeneNetWeaver.

GeneNetWeaver. This method starts with a network of regulatory interactions among transcription factors (TFs) and their targets. It computes a connected, dense subnetwork around a randomly selected seed node and converts this network into a system of differential equations. To express this network in the form of Ordinary Differential Equations (ODEs), it assigns each node i in the network a ‘gene’ variable (x_i) representing the level of mRNA expression and a ‘protein’ variable (p_i) representing the amount of transcription factor produced by protein translation as follows:

$$\begin{aligned}\frac{d[x_i]}{dt} &= mf(R_i) - l_x[x_i] \\ \frac{d[p_i]}{dt} &= r[x_i] - l_p[p_i],\end{aligned}$$

where m is the mRNA transcription rate, l_x is the mRNA degradation rate, r is the protein translation rate, and l_p is the protein degradation rate. In the first equation, R_i denotes the set of regulators of node i . The non-linear input function $f(R_i)$ captures all the regulatory interactions controlling the expression of node i [64]; we specify it below.

If there are N regulators for a given gene, there are 2^N possible configurations of how the regulators can bind to the gene’s promoter. Considering cooperative effects of regulator binding, the probability of each configuration $S \in 2^{R_i}$, the powerset of R_i , is given by the following equation [114]:

$$\Pr(S) = \frac{\prod_{q \in S} ([q]/k)^n}{1 + \sum_S \prod_{q \in S} ([q]/k)^n}, \quad (4.1)$$

where k and n are the Hill threshold and Hill coefficient respectively. Here, we use q to denote a regulator in the configuration S . In the summation in the denominator of this equation, the set S ranges over all members of the power set 2^{R_i} other than the empty set. Here the product in the numerator ranges over all regulators that are present (bound) in a specific configuration S , and the sum in the denominator runs over all configurations in the power set, other than the empty set. GeneNetWeaver further introduces a randomly-sampled parameter $\alpha_S \in [0, 1]$ to specify the efficiency of transcription activation by a specific configuration S of bound regulators. Thus, the function $f(R_i)$ thus takes the following form:

$$f(R_i) = \sum_{S \in 2^{R_i}} \alpha_S \Pr(S) \quad (4.2)$$

Next, GeneNetWeaver adds a noise term to each equation to mimic stochastic effects in gene expression [85]. In addition, in order to create variations among individual experimental

samples, GeneNetWeaver recommends adopting a multifactorial perturbation [85] that increases or decreases the basal activation of each gene in the GRN simultaneously by a small, randomly-selected value. GeneNetWeaver removes this perturbation after the first half of the simulation. Simulating this system of SDEs generates the requisite gene expression data.

BoolODE uses Boolean models to create simulated datasets. In order to generate simulated time course data for our analysis, we used the GeneNetWeaver framework with one critical difference and one minor variation. The form of the equations used by BoolODE is identical to that of GeneNetWeaver. The critical difference is that we do not sample the α_S parameters in Equation 4.2 randomly, i.e., we do not combine the regulators of each gene using a random logic function. Instead, we use the fact that in both the artificial networks and the literature-curated models, we know the Boolean function that specifies how the states of the regulators control the state of the target genes. Moreover, we can express any arbitrary Boolean function in the form of a truth table relating the input states (i.e., activities of transcription factors) to the output state (the activity of target gene). For a gene with N regulators in its Boolean function, we explore all 2^N combinations of transcription factor states and evaluate the transcriptional activity of each specific regulator configuration. Since the value of the Boolean function is the logical disjunction ('or') of all these values, we set the α value to one (respectively, zero) for every configuration that evaluates to 'on' (respectively, 'off'). The following example illustrates our approach. Consider a gene X with two activators (A and B) and one inhibitor (C), represented by the following rule:

$$X = (P \vee Q) \wedge \neg(R)$$

The truth table corresponding to this rule along with the α parameters appears below.

P	Q	R	X	Parameter name	Parameter value
0	0	0	0	α_0	0
1	0	0	1	α_P	1
0	1	0	1	α_Q	1
0	0	1	0	α_R	0
1	1	0	1	α_{PQ}	1
1	0	1	0	α_{PR}	0
0	1	1	0	α_{QR}	0
1	1	1	0	α_{PQR}	0

Therefore, the ODE governing the time dynamics of gene X is

$$\begin{aligned} \frac{d[X]}{dt} &= m \left(\frac{\alpha_0 + \alpha_P[P] + \alpha_Q[Q] + \alpha_R[R] + \alpha_{PQ}[P][Q] + \alpha_{PR}[P][R] + \alpha_{QR}[Q][R] + \alpha_{PQR}[P][Q][R]}{1 + [P] + [Q] + [R] + [P][Q] + [P][R] + [Q][R] + [P][Q][R]} \right) - l_x[X] \\ &= m \left(\frac{[P] + [Q] + [P][Q]}{1 + [P] + [Q] + [R] + [P][Q] + [P][R] + [Q][R] + [P][Q][R]} \right) - l_x[X], \end{aligned}$$

since only α_P, α_Q , and α_{PQ} have the value one and every other parameter has the value zero.

Next, we discuss the minor variation of BoolODE from GeneNetWeaver, which is in how we sample kinetic parameters. The GeneNetWeaver equations use four kinetic parameters: one each for mRNA transcription, protein translation, and mRNA and protein degradation rates. Saelens *et al.* [66] sample them uniformly from parameter-specific intervals.

Independently for every dataset, we sample each parameter from a normal distribution using the value shown in Table 4.1 as the mean and a standard deviation of 10% of this mean value. Within a single dataset and for all simulations for that dataset, we fix each parameter (e.g., mRNA degradation rate) for all genes.

Parameter	Symbol	value
mRNA transcription rate	m	20
mRNA degradation rate	l_x	10
Protein translation rate	r	10
Protein degradation rate	l_p	1
Hill threshold	k	10
Hill coefficient	n	10

Table 4.1: Kinetic parameters used in BoolODE.

In order to create stochastic simulations, we use the formulation proposed by Saelens *et al.* [66] to modify the ODE expressions as follows:

$$\begin{aligned}\frac{d[x_i]}{dt} &= m f(R_i) - l_x[x_i] + s\sqrt{[x_i]}\Delta W_t \\ \frac{d[p_i]}{dt} &= r[x_i] - l_p[p_i] + s\sqrt{[p_i]}\Delta W_t \\ \Delta W_t &= \mathcal{N}(0, h),\end{aligned}$$

where s is the noise strength. We use $s = 10$ in our simulations. We use the Euler-Maruyama scheme for numerical integration of the SDEs [115] with time step of $h = 0.01$.

Defining a single cell. We define the vector of gene expression values corresponding to a particular time point in a model simulation as a single cell. For every analysis, we sample one time point, i.e. one cell from a single simulation. Using this procedure, for a dataset generated from 5000 simulations, we can obtain up to 5000 cells.

Creating GeneNetWeaver simulations for comparison with BoolODE. In order to simulate a synthetic network using GeneNetWeaver, we used its the edge list as the input network to GeneNetWeaver. In order to create the simulations, we used the default options of the noise parameter (0.05) and multifactorial perturbations. We only performed wildtype simulations and used the DREAM4 time series output format for comparison with the BoolODE output.

Summary. We developed the BoolODE approach to convert Boolean functions specifying a

GRN directly to ODE equations. Our proposed BoolODE pipeline accepts a file describing a Boolean model as input, creates an equivalent ODE model, adds noise terms, and numerically simulates a stochastic time course. Different model topologies can produce different numbers of steady states. Since we carry out stochastic simulations, we perform a large number of simulations in an attempt to ensure that we can reach every steady state. Our earlier analysis of the trajectories computed by BoolODE on datasets from curated models demonstrates the success of our approach in this regard. We prefer BoolODE over a direct application of GeneNetWeaver to create datasets from synthetic networks and datasets from curated models for three reasons: (a) A dense regulatory subnetwork computed around a randomly-selected node, as used by GeneNetWeaver, may not correspond to a real biological process. (b) GeneNetWeaver introduces a random, initial, multifactorial perturbation and removes it halfway in order to create variations in the expression profiles of genes across samples. This stimulation may not correspond to how single-cell gene expression data is collected. (c) GeneNetWeaver’s SDEs do not appear to capture single-cell expression trajectories, as we have shown in Figure 4.2.

4.4.3 Datasets

Creating Datasets from synthetic networks

We now describe how we selected synthetic networks, converted these networks into systems of stochastic differential equations (SDEs), simulated these systems, and pre-processed the resulting datasets for input to GRN inference algorithms.

Selecting networks. In order to create synthetic datasets exhibiting diverse temporal trajectories, we use six “toy” networks created by Saelens *et al.* [66] in their comparison of pseudotime inference algorithms (Figure 4.2 and Table 4.2). When simulated as SDEs, we expect the models produce to trajectories with the following qualitative properties:

Linear: a gene activation cascade that results in a single temporal trajectory with distinct final and initial states.

Linear long: similar to Linear but with a larger number of intermediate genes.

Cycle: an oscillatory circuit that produces a linear trajectory where the final state overlaps with the initial state.

Bifurcating: a network that contains a mutual inhibition motif between two genes resulting in two distinct branches starting from a common trajectory.

Trifurcating: three consecutive mutual inhibition motifs in this network result in three distinct steady states.

Bifurcating Converging: an initial bifurcation creates two branches, which ultimately converge to a single steady state.

We do not consider three other networks (Consecutive Bifurcating, Bifurcating Loop, and Converging) created by Saelens *et al.* since we were unable to recapture the expected trajectories using the approach described below.

Converting networks into SDE models and simulating them. We manually convert each of these networks to a Boolean model: we set a node to be ‘on’ if and only if at least one activator is ‘on’ and every inhibitor is ‘off’. We simulate these networks using BoolODE, a method we have developed to convert Boolean models into systems of differential equations (Section 4.4.2). We use the initial conditions specified in the Dynverse software created by Saelens *et al.* (Table 4.3). In order to sample cells from the simulations that capture various locations along a trajectory, we limit the duration of each simulation according to the characteristics of the model (Table 4.3). For example, we simulated the Linear Long network for 15 steps but the Linear network, which has fewer nodes, for 5 steps.

Comparing simulated datasets with the expected trajectories from synthetic networks. It is common to visualize simulated time courses from ODE/SDE models as time course plots or phase plane diagrams with two or three dimensions. The latter are useful to qualitatively explore the state-space of a model at hand. The recent popularity of t-SNE as a tool to visualize and cluster high dimensional scRNAseq data motivated us to consider this technique to visualize simulated single-cell data. Figure 4.2 shows two-dimensional t-SNE visualizations of each of the toy models. The color of each point on the t-SNE scatter plot reflects the corresponding simulation timepoint (blue for early, green for intermediate, and yellow for later time points). We find that the t-SNE visualizations succeed in capturing the qualitative steady states of each of the models we simulate. Specifically, the number of steady state clusters we expect from each synthetic network (e.g., one steady state for linear, two for bifurcating, etc.) are visually apparent in the t-SNE plots.

Pre-processing datasets from synthetic networks for GRN algorithms. Saelens *et*

Name of network	Number of genes	Number of edges		Network density	Number of Steady States
		Activation	Inhibition		
Linear	7	7	1	0.17	1
Linear Long	18	18	1	0.06	1
Cycle	6	3	3	0.2	1*
Bifurcating	7	7	5	0.24	2
Bifurcating Converging	10	13	5	0.17	1
Trifurcating	8	10	10	0.30	3

Table 4.2: Summary of synthetic networks. We ignored self loops while computing network density. An asterisk indicates limit cycle oscillations.

al. [66] have shown that pseudotime inference methods perform well for linear and bifurcating trajectories. However, even the best performing pseudotime algorithm fails to accurately identify more complex trajectories such as Cycle and Bifurcating Converging. Therefore, we sought to develop an approach for pre-processing synthetic datasets that mimicked a real single cell gene expression pipeline while isolating the GRN inference algorithms from the limitations of pseudotime techniques. Accordingly, we used the following four-step approach to generate single-cell gene expression data from each of the six synthetic networks:

1. Using the values in Table 4.3 as the means, and 10% of these values as the standard deviations, we sampled a parameter set. We then used BoolODE to perform 5000 simulations using this parameter set.
2. We represented each simulation as a $|G| \times |C|$ matrix, where G is the set of genes in the model and C is the set of cells in the simulation. We converted each of the 5,000 matrices into a 1-D vector of length $|G| \times |C|$ and clustered the vectors using k -means clustering with k set to the number of expected trajectories for each network. For example, the Bifurcating network in Figure 4.2 has two distinct trajectories, so we used $k = 2$.
3. We then randomly sampled one set each of 100, 200, 500, 2000, and 5000 cells from the 5,000 simulations.
4. Finally, we set as input to each algorithm the $|G| \times |D|$ matrix, where G is the set of genes in the model and D is the set of randomly sampled cells. For those methods that require time information, we specified the simulation time at which the cell was sampled.

We repeated this procedure on 10 different sampled parameter sets to obtain 50 datasets. We ran each algorithm on these 50 datasets.

Note that we used the same set of sampled parameters for all simulation in a dataset and that we varied parameters only across datasets. As an alternative, we considered sampling a different set of parameters per simulation since they may vary from cell to cell. However, this approach caused so much variation that we could not recapitulate the steady states of the Boolean models in the BoolODE-created data.

Note that we clustered simulations themselves (with each simulation represented by the full time courses of all genes) before we sampled cells. The reason we adopted this ordering is that the goal of the clustering was to partition the cells such that each group would (a) correspond to a distinct steady state of the network and (b) contain cells sampled from the entire time course of the simulations. Clustering the simulations helped us to compute these types of clusters. Subsequently sampling one cell from each simulation permitted us to assign each cell to the cluster to which its simulation belonged and satisfy both properties we desired.

In contrast, if we had sampled cells and then clustered them, we were concerned that some cells would have belonged to a cluster corresponding only to early timepoints and some only to intermediate, resulting in the possibility that some steady states would not have any clusters. Alternatively, we would have had to increase the number of clusters we sought to compute but then have been forced to compute pseudotime for the clusters corresponding to early or intermediate timepoints.

Network	# of simulation steps	Initial conditions
Linear	5	$g1 = 1$
Linear long	15	$g1 = 1$
Bifurcating	5	$g1 = 1$
Trifurcating	8	$g1 = 1; g7 = 1$
Cycle	8	$g1 = 1; g2 = 1; g3 = 1$
Bifurcating Converging	8	$g1 = 1$

Table 4.3: Parameters used to generatedatasets from synthetic networks. The final column lists the nodes set to a value of one at the beginning of the simulation. We set all the other nodes to a small value ($=0.01$) close to zero to avoid numerical errors.

Creating Datasets from curated models

While the synthetic models presented above are useful for generating simulated data with a variety of specific trajectories, these networks do not correspond to any real cellular process. In order to create simulated datasets that better reflect the characteristics of single-cell transcriptomic datasets, we turned to published Boolean models of gene regulatory networks, as these models are reflective of the real “ground truth” control systems in biology.

Since tissue differentiation and development are active areas of investigation by single cell methods, we examined the literature from the past 10 years to look for published Boolean models of gene regulatory networks involved in these processes. We selected four published models for analysis. Table 4.4 lists the size of the regulatory networks and the number of steady states. Below, we discuss the biological background, the interpretation of model steady states, and the expected type of trajectories for each of these Boolean models.

Mammalian Cortical Area Development. Giancomantonio *et al.* explored mammalian Cortical Area Development (mCAD) as a consequence of the expression of regulatory transcription factors along an anterior-posterior gradient [87]. The model contains five transcription factors connected by 14 interactions, captures the expected gene expression patterns in the anterior and posterior compartments respectively, and results in two steady states. Figure 4.6 displays the regulatory network underlying the model along with a t-SNE visualization of the trajectories simulated using BoolODE. Figure B.6 shows that the two clusters observed in the t-SNE visualization correspond to the two biological states captured by the Boolean model.

Name	Number of genes	Number of edges		Network density	Number of steady states	Reference
		Activation	Inhibition			
mCAD	5	5	9	0.65	2	[87]
VSC	8	0	15	0.27	5	[88]
HSC	11	15	15	0.24	4	[89]
GSD	19	27	59	0.25	2	[90]

Table 4.4: Summary of published Boolean models. We ignored self loops when we computed network density.

Ventral Spinal Cord Development. Lovrics *et al.* investigated the regulatory basis of ventral spinal cord development [88]. The model consisting of 8 transcription factors involved in ventralization contains 15 interactions, all of which are inhibitory. It succeeds in accounting for five distinct neural progenitor cell types. We expect to see five steady states from this model. Figure 4.6 shows the regulatory network underlying the model along with the t-SNE visualization of the trajectories simulated using BoolODE. Figure B.7 shows that the five steady state clusters observed in the t-SNE visualization correspond to the five biological states captured by the model.

Hematopoietic Stem Cell Differentiation. Krumsiek *et al.* investigated the gene regulatory network underlying myeloid differentiation [89]. The proposed model has 11 transcription factors and captures the differentiation of multipotent Myeloid Progenitor (CMP cells) into erythrocytes, megakaryocytes, monocytes and granulocytes. The Boolean model exhibits four steady states, each corresponding to one of the four cell types mentioned above. Figure 4.6 shows the regulatory network of the HSC model along with the t-SNE visualization of the trajectories simulated using BoolODE. Figure B.8 shows that the four steady-state clusters observed in the t-SNE visualization correspond to the four biological states captured by the Boolean model.

Gonadal Sex Determination. Rios *et al.* 2015 modeled the gonadal differentiation circuit that regulates the maturation of the Bipotential Gonadal Primordium (BGP) into either male (testes) or female (ovary) gonads [90]. The model consists of 18 genes and a node representing the Urogenital Ridge (UGR), which serves as the input to the model. For the wildtype simulations, the Boolean model predominantly exhibits two steady states corresponding to the Sertoli cells (male gonad precursors) or the Granulosa cells (female gonad precursor), and one rare state corresponding to a dysfunctional pathway. Figure 4.6 shows the regulatory network of the GSD model along with the t-SNE visualization of the trajectories simulated using BoolODE. Figure B.9 shows that the two steady state clusters observed in the t-SNE visualization correspond to the two predominant biological states capture by the Boolean model.

Pre-processing datasets from curated models for GRN algorithms. We simulated the four curated models using parameters mentioned in Table 4.5. As with the datasets from synthetic networks, we performed 150 simulations, and sampled 100 cells per simulation for

a total of 15,000 cells. For each model, we then randomly sampled ten different datasets, each containing 2,000 cells. In order to closely mimic the preprocessing steps performed for real single-cell gene expression data, we use the following steps for each dataset:

1. *Pseudotime inference using Slingshot.* In the case of datasets from synthetic networks, we used the simulation time directly for those GRN inference methods that required cells to be time-ordered. In contrast, for datasets from curated models data, we ordered cells by pseudotime, which we computed using Slingshot [91]. We selected Slingshot for pseudotime inference due to its proven success in correctly identifying cellular trajectories in a recent comprehensive evaluation of this type of algorithm [66]. Slingshot needs a lower dimensional representation of the gene expression data as input. In addition, if the cells belong to multiple trajectories, Slingshot needs a vector of cluster labels for the cells, as well as the cluster labels for cells in the start and end states in the trajectories. To obtain these additional input data for Slingshot, we used the following procedure:
 - (a) We use t-SNE on the $|G| \times |C|$ matrix representing the data to obtain a two-dimensional representation of the cells.
 - (b) We performed k -means clustering on the lower dimensional representation of the cells with k set to one more than the expected number of trajectories. For example, since we knew that the GSD model had a bifurcating trajectory, we used $k = 3$.
 - (c) We computed the average simulation time of the cells belonging to each of the k clusters. We set the cluster label corresponding to the smallest average time as the starting state and the rest of clusters as ending states.
 - (d) We then ran Slingshot with the t-SNE-projected data and starting and ending clusters as input. We obtained the trajectories to which each cell belonged and its pseudotime as output from Slingshot.

Figure 4.6 display the results for one dataset of 2,000 cells for each of the four model. We observed that Slingshot does a very good job of correctly identifying cells belonging to various trajectories. Further, the pseudotime computed for each cell by Slingshot is highly correlated with the simulation time at which the cell was sampled (data not shown).

2. *Inducing dropouts in datasets from curated models.* We used the same procedure as Chan *et al.* [52] in order to induce dropouts, which are commonly seen in single cell RNA-Seq datasets, especially for transcripts with low abundance [116]. We created a dataset with a dropout rate of q as follows: for every gene, we sorted the cells in increasing order of that gene's expression value. We set the expression of that gene in each of the lowest q^{th} percentile of cells in this order to zero with a $q\%$ chance. For example, choosing $q = 50$ resulted in a 50% chance of setting a gene's expression values below the 50th percentile to 0, which affected about 25% of the dataset. Note that we

used the same parameter twice simply for convenience; we do not expect the trends we find to deviate considerably if we had used two different parameters. We applied two different dropout rates: $q = 50$ and $q = 70$. We used Slingshot to recompute pseudotime for each dataset with dropouts.

At the end of this step, we had 30 datasets with 2,000 cells for each of the four models: 10 datasets without dropouts, 10 with dropouts at $q = 20$, and 10 with dropouts at $q = 50$.

Model	# of simulation steps	Initial conditions
mCAD	5	Fgf8 = 1
VSC	5	—
HSC	8	Cebpa = 1, Pu1 = 1, Gata2 = 1
GSD	8	UGR = 1

Table 4.5: Parameters used to generate datasets from curated models. The final column lists the nodes set to a value of one at the beginning of the simulation. We set all the other nodes to a small value ($=0.01$) close to zero to avoid numerical errors. For the VSC model, the original publication carried out simulations from all possible states. We carry out VSC simulations by setting the initial conditions of all variables to 1.

Collecting Experimental single-Cell RNA-seq datasets and ground truth networks

Experimental single-Cell RNA-seq datasets. We obtained five different single cell RNA-seq datasets for evaluating GRN inference algorithms. These datasets are from three different mouse cell types and two different human cell types. We pre-processed each dataset using the procedure described in the corresponding paper. In general, if the publication did not provide normalized expression values, we log-transformed the TPM or FPKM counts using a pseudocount of 1 and used the results as the expression values. We additionally filtered out any genes that were expressed in fewer than 10% of the cells. We describe the details of the datasets and the Slingshot pseudotime computation below:

1. Mouse Hematopoietic Stem and Progenitor Cells (mHSC) [94]: We obtained the normalized expression data for 1,656 HSPCs across 4,773 genes from the supplementary data provided by Nestorowa *et al.* [94]. As in the that publication, we used the first three dimensions from DiffusionMap in order to compute pseudotime values. In order to mimic the pseudotime computation performed in the original publication, we used the E-SLAM population as the starting cell type and computed pseudotime along three lineages, namely, erythroid (E), granulocyte-monocyte (GM), and lymphoid (L). We inferred GRNs for each lineage independently.
2. Mouse Embryonic Stem Cells (mESC) [96]: This dataset contains scRNA-seq expression for 421 primitive endoderm (PrE) cells differentiated from mESCs, collected at five

different time points (0h, 12h, 24h, 48h, until 72 hours). SCODE, SINGE and GRISLI used this dataset to evaluate their performance. We computed the pseudotime with cells measured at 0h as the starting cluster and the cells measured at 72h as the ending cluster.

3. Mouse Dendritic Cells (mDC) [93]: This dataset contains scRNA-seq expression for over 1,700 bone-marrow derived dendritic cells under various conditions. Using the same approach as in the SCRIBE pre-print [56], we used the LPS stimulated wild-type cells measured at 1h, 2h, 4h, and 6h. We then computed the pseudotime with cells measured at 1h as the starting cluster and the cells measured at 6h as the ending cluster.
4. Human Mature Hepatocytes (hHep) [95]: This dataset is from a scRNA-seq measurement performed on cells during hepatocyte-like differentiation from induced pluripotent stem cells (iPSCs) in 2D culture. The dataset contains 425 scRNA-seq measurements from multiple time points: days 0 (iPS cells), 6, 8, 14, and 21 (mature hepatocyte-like (MH)). We computed the pseudotime with cells measured on day 0 (iPSCs) as the starting cluster and the cells measured on 21 (MHs) as the ending cluster.
5. Human Definitive Endoderm (hESC) [92]: This dataset is from a time course scRNA-seq experiment derived from 758 cells along the differentiation protocol to produce DE cells from human embryonic stem cells, measured at 0h, 12h, 24h, 36h, 72h and 96h. We computed the pseudotime with cells measured at 0h as the starting cluster and the cells measured at 96h as the ending cluster. SCODE used this dataset to evaluate its performance.

Once we obtained the pseudotime values for the cells in each dataset, we then computed which genes had expression values that varied along the pseudotime. We used the general additive model (GAM) obtained from the ‘gam’ R package to compute the variance and the p -value of this variance. We used the Bonferroni method to correct for testing multiple hypotheses. Table 4.6 provides the statistics on the number of genes and TFs in each dataset after using a corrected p -value cutoff of 0.01. We selected genes for GRN inference in two different ways.

- (i) We considered all genes with p -value less than a threshold. We selected thresholds so that we obtained 500 and 1,000 genes. We recorded the number of TFs in these sets.
- (ii) We started by including all TFs whose variance had p -value at most 0.01. Then, we added 500 and 1,000 additional genes as in the previous option. This approach enabled the GRN methods to consider TFs that may have a modest variation in gene expression but still regulate their targets.

After applying a GRN inference algorithm to a dataset, we only considered interactions outgoing from a TF in further evaluation.

Dataset	Reference	Species	# of Cells	# of Genes	# of TFs
mHSC-E	Nestorowa <i>et al.</i> [94]	Mouse	1,071	2,634	204
mHSC-L	Nestorowa <i>et al.</i> [94]	Mouse	847	692	60
mHSC-GM	Nestorowa <i>et al.</i> [94]	Mouse	889	1,595	132
mESC	Hayashi <i>et al.</i> [96]	Mouse	421	8,150	620
mDC	Shalek <i>et al.</i> [93]	Mouse	383	3,755	321
hHep	Camp <i>et al.</i> [95]	Human	425	4,336	311
hESC	Chu <i>et al.</i> [92]	Human	758	4,406	330

Table 4.6: Statistics on experimental single-cell RNA-seq datasets. The column “#Genes” displays the number of genes whose expression values varied along the pseudotime trajectory with a Bonferroni-corrected p -value of at most 0.01, as computed by GAM. Abbreviations: HSC: Hematopoietic stem cells, E: Erythroid, L: Lymphoid, GM: Granulocyte-Macrophage, ESC: Embryonic stem cells, DC: Dendritic cells, Hep: hepatocytes.

Ground truth networks collection and processing. In the GRN inference literature, a common practice is to evaluate the accuracy of a resulting network by comparing its edges to an appropriate database of TFs and their targets. For example, SCODE used the RikenTFdb and animalTFDB resources to define ground truth networks for mouse and human gene expression data, respectively [51]. We used three types of ground truth datasets.

1. Cell-type specific: For each experimental scRNA-seq dataset, we searched the ENCODE, ChIP-Atlas, and ESCAPE databases for ChIP-Seq data from the same or similar cell type. We also included the loss-of-function/gain-of-function dataset from the ESCAPE database.
2. Non-specific: Here, we used the following resources:
 - (a) DoRothEA integrates information from multiple sources [117]. We considered two levels of evidence in this database: A (curated/high confidence) and B (likely confidence).
 - (b) RegNetwork incorporates genome-wide TF-TF, TF-gene, TF-miRNA regulatory relationships in human and mouse collected from various sources [98]. We used the TF-TF and TF-gene interactions for our analysis.
 - (c) TRRUST contains TF-target interactions collected based on text-mining followed by manual curation for human and mouse [99].
3. Functional: Finally, we used the human and mouse STRING networks. An interaction here is functional and need not correspond to transcriptional regulation. We selected this type of ground truth network due to our observation that many GRN methods predict indirect interactions for Boolean models.

We found that the ChIP-seq networks had very few edges when we overlapped with the highly-varying subsets from the gene expression data. Therefore, we considered the next two types of ground truth networks. We provide details on the number of interactions from each source in Table 4.7.

Sp	Source	#TFs	#Genes (incl. TFs)	#Edges	Density	Expression dataset
Mouse	HSC, E, L, G-M CA	137	19,324	1,078,888	0.407	mHSC [94]
Mouse	DC, ChIP-Atlas	36	11,092	30,658	0.077	mDC [93]
Mouse	mESC, ESCA + CA	247	25,703	6,348,394	0.154	mESC [96]
Mouse	mESC, LOGOF, ESCA	57	18,427	104,797	0.100	mESC [96]
Mouse	TRRUST + RN	1,455	17,852	100,139	0.004	
Mouse	STRING	1,350	7,771	157,134	0.015	
Human	hESC, ChEA + CA	130	18,104	436,563	0.186	hESC [92]
Human	HEPG2, ChEA + CA	84	16,822	342,862	0.243	HEPG2 [95]
Human	TRRUST + RN	2,165	23,566	386,293	0.008	
Human	+ DoRothEA STRING	1,489	8,806	198,285	0.015	

Table 4.7: Statistics on networks used to evaluate experimental single-cell RNA-seq datasets. To compute the density, we divided the number of edges by the total number of edges that can be outgoing from a TF. Abbreviations: HSC: Hematopoietic stem cells, E: Erythroid, L: Lymphoid, G-M: Granulocyte-Macrophage, ESC: Embryonic stem cells, DC: Dendritic cells, Hep: hepatocytes, ESCA: ESCAPE database, CA: ChIP-Atlas, RN: RegNetwork.

4.4.4 Evaluation pipeline

One of the major challenges we faced was that the GRN inference methods we included in this evaluation were implemented in a variety of languages such as R, MATLAB, Python, Julia and F#. In order to obtain an efficient and reproducible pipeline, we Dockerized the implementation of each of these algorithm. Supplementary File 1 contains details on the specific software or GitHub commit versions we downloaded and used in our pipeline. For methods implemented in MATLAB, we created MATLAB executable files (.mex files) that we could execute within a Docker container using the MATLAB Runtime. We further studied the publications and the documentation of the software (and the source code, on occasion) to determine how the authors recommended that their methods be used. We have incorporated these suggestions as well as we could. We provide more details in Appendix B.4.

Inputs. For datasets from synthetic networks and for datasets from curated models, we provided the gene expression values obtained from the simulations directly after optionally inducing dropouts in the second type of data. As shown in Figure 4.3, eight out of the 12 methods also required some form of time information for every cell in the dataset. Of these, two methods (GRNVBEM and LEAP) only required cells to be ordered according to their

pseudotime and did not require the pseudotime values themselves.

Parameter estimation. Six of the methods also required one or more parameters to be specified. To this end, we performed parameter estimation for each of these methods separately for datasets from synthetic networks, datasets from curated models, and real datasets and provided them with the parameters that resulted in the best AUPRC values. See Appendix B.1.2 for details.

Output processing. Ten of the GRN inference methods output a confidence score for every possible edge in the network, either as an edge list or as an adjacency matrix, which we converted to a ranked edge list. We gave edges with the same confidence score the same rank.

Performance evaluation. We used a common evaluation pipeline across all the datasets considered in this paper. We evaluated the result of each algorithm using the following criteria:

1. **AUPRC, AUROC:** We computed areas under the Precision-Recall (PR) and Receiver Operating Characteristic (ROC) curves using the edges in the relevant ground truth network as ground truth and ranked edges from each method as the predictions. We ignored self-loops for this analysis since some methods such as PPCOR always assigned the highest rank to such edges and some other methods such as SCINGE and always ignored them. Most of the networks we considered have a density of 0.3 or less, i.e., the positive-to-negative ratio is worse than 1:3. Since the GRN inference problem for these networks is moderately imbalanced, we focus on AUPRC scores in the main text [118]. We provide AUROC plots in the supplement for readers who are interested in these performance statistics.
2. **Stability across multiple runs:** We executed each algorithm 10 times on a dataset to ask if the inferred networks changed from one run to another. We represented every result by the corresponding ranked list of edges. For every pair of results, we computed the Spearman’s correlation of the ranked lists.
3. **Identifying top- k edges:** We first identified top- k edges for each method, where k equaled the number of edges in the ground truth network (excluding self loops). In multiple edges were tied for a rank of k , we considered all of them. If a method provided a confidence score for fewer than k edges, we used only those edges. For experimental single-cell RNA-seq datasets, this number varied from one ground truth dataset to another.
4. **Stability across multiple datasets:** Once we obtained the set of top- k edges for each method and each dataset, we then computed the Jaccard index of every pair of these sets. We used the median of the values as an indication of the robustness of a method’s output to variations in the simulated datasets from a given synthetic network or Boolean model.

5. **Early precision (EP) and Early precision ratio (EPR):** We defined early precision as the fraction of true positives in the top- k edges. We also computed the early precision ratio, which represents the ratio of early precision value and the early precision for a random predictor for that model. A random predictor’s precision is the edge density of the ground truth network.
6. **Early precision of signed edges:** We desired to check whether there were any differences in how accurately a GRN inference algorithm identified activating edges in comparison to inhibitory edges. To this end, we computed the top- k_a edges from the ranked list of edges output by each method, where k_a is the number of activating edges in the ground-truth network. In this step, we ignored any inhibitory edges in the ground truth network. We defined the early precision of activating edges as the fraction of true edges of this type in the top k_a edges. We used an analogous approach to compute early precision of inhibitory edges. We also computed the early precision ratio for these values.
7. **Network motifs:** In each proposed network formed using the top- k predicted edges, we counted the number of three-node feedback loops, feedforward loops and mutual interactions involving every edge and divided this number by their corresponding values in the ground truth networks. For this analysis, we ignored algorithms that predict only undirected edges (PIDC and PPCOR).
8. **Connectivity of false positive edges:** We were interested in determining if GRN methods were successful in weeding out indirect interactions from their predictions. To this end, for each false positive edge in a top- k network, we computed the length of the shortest path between the endpoints of that edge in the relevant ground truth network. We plotted histograms of the distributions of these distances. We ignored algorithms that predict only undirected edges for this analysis as well.

Datasets with multiple trajectories. Seven methods we evaluated require pseudotime-ordered cells but cannot directly handle data with branched trajectories (Figure 4.14). In these cases, as suggested uniformly by many of these methods, we separated the cells into multiple linear trajectories using Slingshot, the top performing pseudotime algorithm in a recent comparison [66], and applied the algorithm to the set of cells in each trajectory individually. To combine the GRNs, for each interaction, we recorded the largest score for it across all the networks, and ranked the interactions by these values. In the case of GRISLI, which outputs ranked edges, for each interaction, we took the best (smallest) rank for it across all the networks.

As an alternative, we merged all the trajectories into one set of cells and executed each algorithm on this dataset. In this case, two cells (from different categories) may have the same pseudotime values. It is also possible for a cell to be part of two or more trajectories. In this situation, we used the smallest pseudotime value assigned to it.

Shuffling simulation times. We investigated the effect of shuffling the pseudotime values on the performance of methods that require this information. We used three window sizes, namely, 15%, 30% and 45%, which defined the range of indices to sample from as a fraction of the total number of cells in a dataset. Thus, for a dataset with 2,000 cells, using a window size of 30% resulted in a range of $2000 \times 0.3 = 600$. Therefore, after sorting the cells in increasing order of pseudotime, for every index i , we sampled a new index from the interval $[\max(0, i - 300), \min(i + 300, 2000)]$ and swapped the cells at these two indices. We performed this analysis for datasets from synthetic networks; here each original (unshuffled) pseudotime was equal to the simulation time.

Fidelity of simulations from inferred GRNs. We sought to determine if a GRN inferred by an algorithm from datasets from synthetic networks, when represented as a Boolean network, would exhibit dynamics and steady states identical to the original models. We first verified that the synthetic networks yielded the expected number of steady states under asynchronous updates, namely, zero for Cycle, one each for Linear, Linear Long, and Bifurcating Converging, two for Bifurcating, and three for Trifurcating network. We first removed all self loops from the ranked edges lists output by each algorithm. Then, we traversed the ranked list of edges output by each algorithm until each gene had at least one incoming interaction. For the output from GRNVBEM, SCODE, and SINCERITIES, we used the signs of interactions (activation or repression) predicted by that algorithm. For every other algorithm, we assigned the signs of interactions based on the values computed by PPCOR. Finally, we added self interactions if they were present in the ground truth network. Next, we defined the rule for each gene as follows: a gene is ON iff at least one activator is ON and every inhibitor is OFF. We simulated this Boolean model asynchronously, i.e., at any state, we selected a gene uniformly at random and applied the rule for that gene to compute the next state. We computed the network formed by the transitions among all the possible states. With both the Boolean network and this “state space network” in hand, we computed multiple statistics.

- (a) We recorded the number of the interactions in the Boolean network created from the computed GRN and its ratio with the number of interactions in the ground truth network.
- (b) We had simulated each ground truth network starting from a specific initial state, e.g., for the Linear model, we set g_1 to ON and all other genes to OFF. We asked how many steady states we could reach from this starting state in the Boolean network created from the computed GRN, and computed the ratio of this number with the number of steady states in the ground truth network.
- (c) Each steady state in a ground truth network corresponded to a specific configuration of genes, e.g., only g_7 is ON in the steady state for the Linear network. For each reachable state in the Boolean network created from the computed GRN, we determined whether its genes were in the same configuration as any steady state in the ground truth network. We recorded the number of these states.

Note that we did not perform this analysis for datasets from curated models since the Boolean rules in these networks were quite complex. We did not expect that the relatively simple rules used by BoolODE would suffice to recapture the properties of the Boolean networks we curated from the literature.

Correspondence of clusters in GRNs and gene expression datasets. Here, we investigated how much clusters in the predicted GRNs agreed with those in the gene expression datasets. We first computed a co-expression network using the absolute value of the Pearson's correlation coefficient as the weight of the edge between two genes. Next, we ran Louvain clustering algorithm on the predicted GRN (considering all predicted edges and their weights) and on co-expression network to compute clusters. In order to compute clusters using Louvain algorithm we converted the GRNs into undirected networks: if a method predicted both the edge (a, b) and (b, a) , we used only the higher of the two edge weights. Next, we measured the similarity between these clusterings using the Adjusted Rand Index [119]. We recorded the values of this index for each pair of experimental single-cell RNA-seq dataset and algorithm.

Procedure for creating Figure 4.14. We chose the following measures presented in earlier sections for summarize our key findings:

Accuracy: We computed the median of the per-network median AUPRC value obtained for datasets containing 2,000 and 5,000 cells for the six synthetic networks (Figure 4.3). For datasets from curated models, we used the median of the per-model median AUPRC value obtained for 10 datasets without dropouts for each the four datasets from curated models (Figure 4.7). For experimental single-cell RNA-seq datasets, we used the median EPR for obtained for each dataset-evaluation network combination (all significantly-varying TFs and the 1000 most-varying genes) (Figure 4.12). We have ordered the algorithms by their median EPR score across experimental single-cell RNA-seq datasets followed by median AURPCs in datasets from synthetic networks.

Stability: We present four such measures: (a) *across datasets*: the median score of the Jaccard index obtained for all six datasets from synthetic networks (Figure 4.3); (b) *across runs*: median Spearman's correlation of outputs for each of the four datasets from curated models (one dataset each, Figure B.18); (c) *across dropouts*: median percentage decrease in AUPRC for the $q = 50$ dropout rate compared to no dropouts (Figure 4.8); (d) *across pseudotime*: median percentage decrease in AUPRC after shuffling time values within a 30% window compared to the original simulation time for datasets from synthetic networks (Figure 4.15).

Scalability: Median running times and memory consumption for three different scRNA-Seq datasets, namely mESCs, hESCs and mHSCs, which contain 400, 750 and 1000 cells, respectively (Figure 4.12(a)). Missing values indicate that either the method did not complete even after running for over a day or it gave a run-time error.

Chapter 5

Supervised inference of gene regulatory networks using graph convolutional neural networks

5.1 Introduction

All the methods evaluated in BEELINE used unsupervised and association-based strategies. For the most part, these methods had low area under the precision-recall curve (AUPRC) or early precision scores when we applied them to experimental scRNA-seq datasets to reconstruct GRNs. This key finding has motivated us investigate supervised learning methods for GRN inference from experimental single-cell RNA-seq (scRNA-seq) datasets. In principle, these techniques can take advantage of known regulatory interactions to predict new connections. While there are several supervised learning techniques for constructing GRNs from bulk RNA-seq data [120, 121, 122, 123, 124], this problem is under-explored in the context of scRNA-seq data [125].

Broadly, there are two types of supervised GRN inference algorithms [126]. A ‘local’ approach trains a classifier for every transcription factor to distinguish between its target and non-target genes [120]. A ‘global’ approach learns a single classifier that can distinguish between interacting and non-interacting TF-gene pairs [121]. In this work, we seek to develop a global classifier since gene regulation involves multiple TFs and it is conceivable that we can learn from the interactions of one TF to infer the interactions of another TF.

Existing global supervised learning methods for GRN inference usually encode the input gene expression matrix into feature vectors for all gene pairs of interest. To generate a feature vector for two genes, one strategy is to simply concatenate their expression vectors [127, 128, 129]. An alternative is to take the outer-product of the gene expression vectors [130]. Other approaches for learning GRNs included kernel-based methods [131, 132, 133]. CNNC computes a normalized empirical probability distribution function (NEPDF) for each gene pair [125].

It is unclear which of these encoding approaches is appropriate for supervised GRN inference from scRNA-seq data. Moreover, using a pre-defined, fixed feature vectors may not be beneficial to the classifier performance. Therefore, we seek to explore methods that can

automatically learn the encoding of the gene expression data for supervised GRN inference. To this end, we take advantage of recent advances in deep learning on graphs [134, 135, 136, 137, 138, 139]. We focus on developing supervised GRN inference methods that uses graph convolutional networks (GCNs) [137]. GCNs learn a lower dimensional embedding of the input node features (here, gene expression data) by propagating these features along the edges in the input graph. Moreover, one can learn node features using a GCN in a way that can reconstruct original graph [136, 139]. This formulation can be thought of as a graph autoencoder [136]. An autoencoder consists of two parts: an *encoder* that learns a low-dimension embedding of the nodes, and a *decoder* that reconstructs the original graph.

In our context, the encoder learns a lower dimensional representation by performing convolutions over the graph formed by known regulatory edges. We then train a decoder that uses this lower dimensional representation of the expression data to predict a confidence score for every possible regulatory edge. The network formed by the top scoring edges. Sorting all possible regulatory edges in decreasing order of edge confidence score and selecting the top edges gives us the predicted GRN. We investigate several encoder-decoder combinations and identify the best architectures for supervised GRN inference. We compare the performance of GCN-based autoencoders to other supervised GRN inference algorithms, namely, SVM [127, 128, 129], multi-layer perceptron (MLP), and CNNC [125]. We show that our approach outperforms existing methods for supervised GRN inference across various evaluations. We also reveal the ability of GCN-based autoencoders to predict cell-type-specific regulatory edges, even when trained on non-specific ChIP-seq networks.

5.2 Methods

Problem setting. The inputs to the problem are (a) a directed, unweighted graph $G = (V, E)$, where each node in V is a TF or a gene and every edge connects a TF to a gene it regulates, and (b) the single-cell gene expression matrix, X , of size $|V| \times |C|$, i.e., for each $i \in V$ we have a vector of gene expression values $x_i \in \mathbb{R}^{|C|}$ measured across the set C of cells in a given sample. The node set can be further subdivided into a set of TFs, T and a set of genes B , such that $V = T \cup B$, $T \cap B = \emptyset$. Since G is a GRN it only contains edges of type $E \subseteq T \times V$.

We can now define the supervised GRN inference problem as follows: Given the graph $G = (V, E)$, the gene expression matrix X . Let A be the adjacency matrix representation of G , and \bar{E} be the set of edges in $(T \times V) \setminus E$. Our goal is to compute a new adjacency matrix \bar{A} , where a_{st} , the st^{th} element in \bar{A}_{ij} has a value $0 \leq \bar{a}_{ij} \leq 1$ for every node pair (i, j) . Here, the value corresponding to \bar{a}_{ij} represents the probability of existence of a regulatory interaction between the two nodes i and j . We want to compute a matrix \bar{A} such that the

reconstruction loss, \mathcal{L} , is minimized:

$$\mathcal{L} = \frac{1}{|E|} \left(- \sum_{(i,j) \in E} \log \bar{a}_{ij} - \sum_{(p,q) \in \bar{E}} \log(1 - \bar{a}_{pq}) \right) \quad (5.1)$$

We use GCN-based autoencoders to predict the matrix \bar{A} . Note that G need not be bipartite since a TF can regulate another. We do not impose any restriction on connectivity of this network in that G may contain more than one connected component. However, we require that there is an edge incident on every node in G .

5.2.1 GCN-based autoencoders

We use a graph convolutional neural network (GCN) based encoder [137] since it enables learning on graphs such as GRNs. Typically, a GCN offers better computational complexity than a multi-layer perceptron or a convolutional neural network with the same number of layers [134, 137]. A GCN-based encoder learns a lower dimensional representation of each node by aggregating information along the edges incident upon the node. Depending on the application, this aggregation can be along all the edges incident upon a node (undirected GCN) or only along the incoming edges to the node (directed GCN). GRNs are directed graphs with edges going out of TFs to the genes they regulate. However, we also considered undirected GCN-based encoders in this study since it is conceivable that we can learn the lower dimensional representation of a TF by aggregating information from the genes they regulate. Therefore, we investigated the following two types of encoders, depending on whether we represent the input GRNs as undirected or directed:

- (i) **Undirected GCN-based encoder (GCN):** Here, we first convert the given input graph G to undirected, i.e., for each edge $(i, j) \in E$, we add an edge (j, i) to E . Let A be the adjacency matrix of this undirected version of the graph G . We add self-edges to A and obtain $\hat{A} = A + I$, I is the identity matrix. We then compute, \hat{D} , the diagonal node degree matrix of \hat{A} , where the diagonal entry in row i is given as $d_i = \sum_{j=0}^n \hat{a}_{ij}$. We are also given the feature matrix X , a number n representing the number of graph convolutional layers, and a list K representing the number of features in each layer $K = \{k_0, k_1, k_2, \dots, k_n\}$, where $k_0 = |C|$. The GCN consists of n layers with the first layer taking X as input and the final layer producing an output matrix Z of size $|V| \times k_n$, where k_n is the number of output features per node. The neural network weights in each layer are represented by a weight matrix $W^{(l)}$ with the dimensions $k_{l-1} \times k_l$. We use $\text{ReLU}(x) = \max(0, x)$, i.e., the rectified linear unit, as the activation function for each layer.

For every $1 \leq l \leq n - 1$, the output of GCN layer l can be written as $H^{(l+1)} = f_u(H^{(l)}, A)$. Here $H^{(0)} = X$, $H^{(n)} = Z$, and $f_u(\cdot)$ is a non-linear propagation function defined as follows:

$$f_u(H^{(l)}, A) = \text{ReLU} \left(\hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}} H^{(l)} W^{(l)} \right),$$

where $\widehat{A} = A + I$, I is the identity matrix, \widehat{D} is the diagonal node degree matrix of \widehat{A} . Alternatively, an individual node v_i 's value in hidden layer l can be obtained using the following relation:

$$h_i^{(l)} = \sum_{j \in \mathcal{N}_i} \frac{1}{\sqrt{d_i d_j}} h_j^{(l)} W^{(l)}$$

where, \mathcal{N}_i is the set of nodes connected to v_i with an edge in the undirected version of G .

- (ii) **Directed GCN-based encoder (DGCN):** The DGCN is similar to a GCN, but as the name suggests, the adjacency matrix A represents the input directed graph G [139]. Further, in order to represent the directed information flow from TFs to genes, the non-linear propagation function at each hidden layer, $f_d(\cdot)$, defined as follows:

$$f_d(H^{(l)}, A) = \text{ReLU} \left((\widehat{D}_{out}^{-1} \widehat{A})^T H^{(l)} W^{(l)} \right),$$

Alternatively, an individual node v_i 's value in hidden layer l can be obtained using the following relation:

$$h_i^{(l)} = \sum_{j \in \mathcal{N}_i} \frac{1}{\sqrt{d_j}} h_j^{(l)} W^{(l)}$$

where, \mathcal{N}_i are the set of nodes with an outgoing edge to the node v_i in G .

Once we encode the input node feature matrix X of size $|V| \times |C|$ into a lower dimensional matrix $Z \in \mathbb{R}^{|V| \times k_n}$, the next step in an autoencoder is to reconstruct the original adjacency matrix from Z using a decoder. In other words, using Z as input, we wish to obtain a reconstructed adjacency matrix \bar{A} , such that $\bar{a}_{st} = g(z_s, z_t)$, where $g(\cdot)$ is a decoder function, z_s and z_t represent rows s and t of Z , respectively. Inspired by the recent factorization techniques in relational learning [138, 140, 141], we considered the following three types of decoders:

- (a) **Inner Product decoder (IP):** This is the simplest form of the decoder, where we compute the matrix $\bar{A} = \sigma(ZZ^T)$ and use its entries as the edge scores, where, $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid activation function. In other words, the confidence score for the edge (s, t) is

$$\bar{a}_{st} = \sigma(\langle z_s, z_t \rangle)$$

where, $\langle \cdot \rangle$ represents the inner product between two vectors. Note that the output adjacency matrix \bar{A} here is symmetric, representing an undirected graph.

- (b) **Node weight-based decoder (NW):** Since GRNs are directed graphs, we propose a modification to the IP decoder, where we learn a node weight v_s corresponding to each node $s \in V$. We then obtain the confidence score for the edge (s, t) as follows:

$$\bar{a}_{st} = \sigma(v_s \langle z_s, z_t \rangle)$$

If the learned weights for two nodes s and t are unequal, then \bar{A} is asymmetric, representing a directed graph.

- (c) **RESCAL decoder (RS):** Finally, we adopted RESCAL [140], a factorization-based approach for relational learning in large-scale data, as the decoder, where we learn a

matrix of weights $M \in \mathbb{R}^{k \times k}$, and obtain the output adjacency matrix $\bar{A} = \sigma(ZMZ^T)$. In other words, we obtain the confidence score corresponding to the edge (s, t) using the following relation:

$$\bar{a}_{st} = \sigma(\langle z_s \times M, z_t \rangle)$$

This formulation also results in a non-symmetric adjacency matrix representing a directed graph.

We use the sigmoid activation function at the decoder to obtain edge scores between 0 and 1. With the above two encoders and three decoders, we obtained a total of six GCN-based autoencoders. The goal of the learning procedure then is to find the weights $W^{(l)}$ for each hidden layer of the encoder and weights for the decoders (if any) such that the cross-entropy reconstruction loss \mathcal{L} in eq. (5.1) is minimized:

5.2.2 Methods evaluated

In this study, we compare the performance of the following approaches for supervised GRN inference from scRNA-seq data:

- (i) We considered all six variants of GCN-based autoencoders, with two encoders (GCN, DGCN), and three decoders (IP, NW, RS). In addition, we evaluated two versions of the feature matrix X :
 - (a) **Gene-by-cell expression matrix (E)**: As defined earlier, X is a $|V| \times |C|$ gene expression matrix where V is the set of genes and C is the set of cells in a scRNA-seq dataset.
 - (b) **Reduced dimension expression matrix using principal component analysis (P)**: X is a reduced $|V| \times |P|$ matrix computed by performing principal component analysis (PCA) on the gene expression matrix and selecting the top- $|P|$ principal components that can explain at least 70% of the variance. Here 70% is a parameter.
- (ii) **MLP-C**: This is a multi-layer perceptron with as many hidden layers as in the GCN and with concatenated expression vectors as input features. Here, we are given as input the graph $G = (V, E)$ and expression matrix X . Let X_{ui} represent the expression of gene u in cell i . For every possible edge of type $(u, v) \in T \times V$, we first create an edge-feature matrix F , such that the $F_{uv} = [X_{u1}, X_{u2}, \dots, X_{u|C|}, X_{v1}, X_{v2}, \dots, X_{v|C|}]$ by concatenating the expression vectors corresponding to the genes u and v . In MLP-C, each hidden layer is a fully connected neural network. We used the same parameters as in the GCN encoder for the number of hidden layers n and number of features in each layer $K = k_0, k_1, \dots, k_n$, with the exception that, $k_0 = 2 \times |C|$. Similar to GCN-based encoder, we used the $\text{ReLU}(\cdot)$ activation function for the hidden layers. Let the neural

network weights in the hidden layer l be $M^{(l)}$. For every $1 \leq l \leq n - 1$, the output of MLP-C at layer l can be written as $H^{(l+1)} = g(H^{(l)})$. Here, $H^{(0)} = F$, and $g(\cdot)$ is a propagation function defined as follows:

$$g(H^{(l)}) = \text{ReLU}(H^{(l)}M^{(l)})$$

In order to obtain a probability score for each pair of nodes as output, we add another fully connected layer $M^{(n)}$ of size $k_n \times 1$, as the output layer to obtain a probability value as follows:

$$H^{(out)} = \sigma(H^{(n)}M^{(n)})$$

We then learn the weights for this neural network by minimizing the loss function in Equation (5.1).

- (iii) **SVM-C**: Here, for every pair of nodes $(u, v) \in T \times V$ we set its feature vector F_{uv} to be the concatenated expression vector of the two genes u and v , same as in MLP-C. We assign the label 1 to each edge (u, v) if $(u, v) \in E$, and 0 otherwise. We then train a linear kernel using the `LinearSVC` function in the `scikit-learn` Python package with default parameters.
- (iv) **CNNC**: a convolutional neural network with NEPDF-based features [125].
- (v) **GRNBoost2**: We considered one of the top-performing unsupervised methods from BEELINE [142] for directed GRN inference as the baseline.

5.2.3 Datasets

We considered four different scRNA-seq expression datasets with varying number of cells and genes for evaluating GRN inference algorithms described above. We used three different mouse scRNA-seq datasets, namely, embryonic stem cells (mESC) dataset [96], hematopoietic stem cells (mHSC) dataset [143], and bone-marrow derived macrophages [144]. In addition, we analyzed the GCN-based autoencoder’s performance on the human embryonic stem cell dataset [92]. We obtained the mESC, mHSC, and, hESC expression datasets from our BEELINE evaluation framework [142]. We obtained the mMac expression dataset from Alavi *et al.* [144], which was one of the datasets used by Yuan *et al.* [125] to evaluate the performance of their method, CNNC. For each of the datasets, we trained the algorithms with the mouse or human non-cell type specific ChIP-seq networks [98, 99, 117] obtained from the BEELINE pipeline [142]. This non-specific ChIP-seq network contains TF-gene edges observed across multiple cell-types. Table 5.1 provides details on each expression dataset and the number of nodes and edges in the corresponding ground-truth networks. The number of nodes for each of the datasets varied between 896 (mESC) to 7,428 (mMac). The column “#TFs” in Table 5.1 represents the number of transcription factors among the nodes in the network, i.e., nodes with at least one outgoing edge.

Dataset	Ref.	Species	#Cells	#Nodes	#Edges	# TFs
mESC	Hayashi <i>et al.</i> [96]	Mouse	471	896	6,893	516
mHSC	Nestorowa <i>et al.</i> [94]	Mouse	3,175	4,158	17,309	445
mMac	Alavi <i>et al.</i> [144]	Mouse	6,283	7,428	35,347	747
hESC	Chu <i>et al.</i> [92]	Human	758	1,142	4,597	292

Table 5.1: Statistics on single-cell RNA-seq datasets. Abbreviations: mESC, mouse embryonic stem cells; mHSC, mouse hematopoietic stem cells, mMac, mouse macrophages, hESC, human embryonic stem cells

5.2.4 Evaluation

For each dataset X containing measurements across V genes and C cells, we remove genes from X if they are not present in the non-specific ChIP-seq network. We can subdivide V into a set of TFs T , set of genes B such that $V = T \cup B$, $T \cap B = \emptyset$. We define the ground-truth network G for genes in V as follows. For every edge (u, v) in the non-specific ChIP-Seq network, we include (u, v) in G if and only if, $u \in T$ and $v \in V$. Let $G = (V, E)$ be the graph so obtained. We treat the edges in E as positives. Since G is a GRN it only contains edges of type $E \subseteq T \times V$. Let $\bar{E} = (T \times V) \setminus E$. We treat the edges in \bar{E} as negatives. We then use the following two strategies to obtain positive and negative examples for training and testing:

- (a) **10-fold edge cross-validation:** We randomly divide the set of edges E in the ground-truth network into 10 sets. Iteratively, we hold out each set as the test positive examples P_{test} and use the remaining nine sets as the training positive examples P_{train} . In order to obtain the negative examples, we use the set of negatives \bar{E} . We choose uniformly at random as many edges as there are training positives from \bar{E} and use the selected edges as training negatives, $N_{train} \subset \bar{E}$, and $|N_{train}| = |P_{train}|$. We then sample as many test negatives, N_{test} , as there are test positives from the remaining negative examples i.e., $N_{test} \subset \bar{E} \setminus N_{train}$, and $|N_{test}| = |P_{test}|$. Since some architectures treat edges as undirected, we remove edges of type (u, v) from a training set, if the edge (v, u) appears in the test set.
- (b) **10-fold TF holdout cross-validation:** Here we first randomly divide all the TFs, T , in the ground-truth network into 10 sets, use nine sets of TFs for training (T_{train}), and hold out one set for testing (T_{test}). For each evaluation, we retain all the edges incident on the test TFs in G as the test positives, i.e., $P_{test} = \{(t, v) \in E | t \in T_{test}\}$. We use the remaining edges from the ground-truth network as train positives, i.e., $P_{train} = E \setminus P_{test}$. In order to obtain the negative examples, we use the set of negatives \bar{E} . To obtain test negatives, N_{test} , for each TF $t_e \in T_{test}$, we select uniformly at random from \bar{E} as many edges of the form (t_e, v) as there are in P_{test} . From the remaining negatives, we choose uniformly at random as many training negatives, N_{train} , as there are training positives, i.e., $N_{train} \subset \bar{E} \setminus N_{test}$ and $|N_{train}| = |P_{train}|$.

5.3 Implementation details

We use the `PyTorch-Geometric` package [145] to implement various neural network architectures. For every fold in the 10-fold cross-validation, we randomly select 20% of the training edges (positive and negative) for validation. We set the maximum number of epochs to 2,000. At each epoch, we compute the mean validation loss of 10 most recent epochs. After the first 1,000 epochs, we terminated the training if the validation loss in the current epoch is lower than the mean validation loss of the last 10 epochs. In order to minimize the loss function in eq. (5.1), we use the Adam optimizer [146] implemented in PyTorch [147] with a learning rate of 0.003. We performed a parameter search for the number of GCN layers in $n = \{1, 2, 3, 5, 7, 9, 10\}$.

5.4 Results

5.4.1 Identifying the best GCN-based autoencoder architecture

We first focused our attention on identifying the best encoder-decoder combination out of the six architectures considered. We performed 10-fold edge cross validation and 10-fold TF-holdout cross validation for the mESC and mHSC datasets with non-specific ChIP-seq network as the ground-truth network. We obtained the median early precision on the test data across the 10 folds. Figure 5.1 summarizes our results. Across both the evaluations, the the undirected GCN encoder (GCN) consistently performed better than the directed GCN-encoder (DGCN). In addition, the expression matrix-based features (top row in Figure 5.1) only had a slightly lower median early precision compared to PCA-based features. However, the decoders' median early precision varied depending on the type of evaluation performed.

As shown in Figure 5.1(a), the undirected GCN encoder (GCN) with RESCAL decoder (RS) had the highest median early precision across all test sets considered for 10-fold edge cross-validation for both datasets. Of the three decoders, the IP decoder consistently underperformed the RS and NW decoders for expression and PCA-based features. Interestingly, we observed the opposite trend for decoders in the case of 10-fold TF-holdout cross-validation (Figure 5.1(b)). The IP decoder consistently had the highest median early precision across all test sets.

A possible explanation for the under performance of weight-based decoders for TF holdout cross-validation (RS and NW) could be the type of evaluation being performed itself. In case of the NW decoder, since we remove all edges adjacent to test TFs from the training set, this decoder has no information about the test TFs in order to learn their corresponding node weights. The similar lack of information could result in overfitting the RS decoder weights to best predict edges adjacent to training TFs alone.

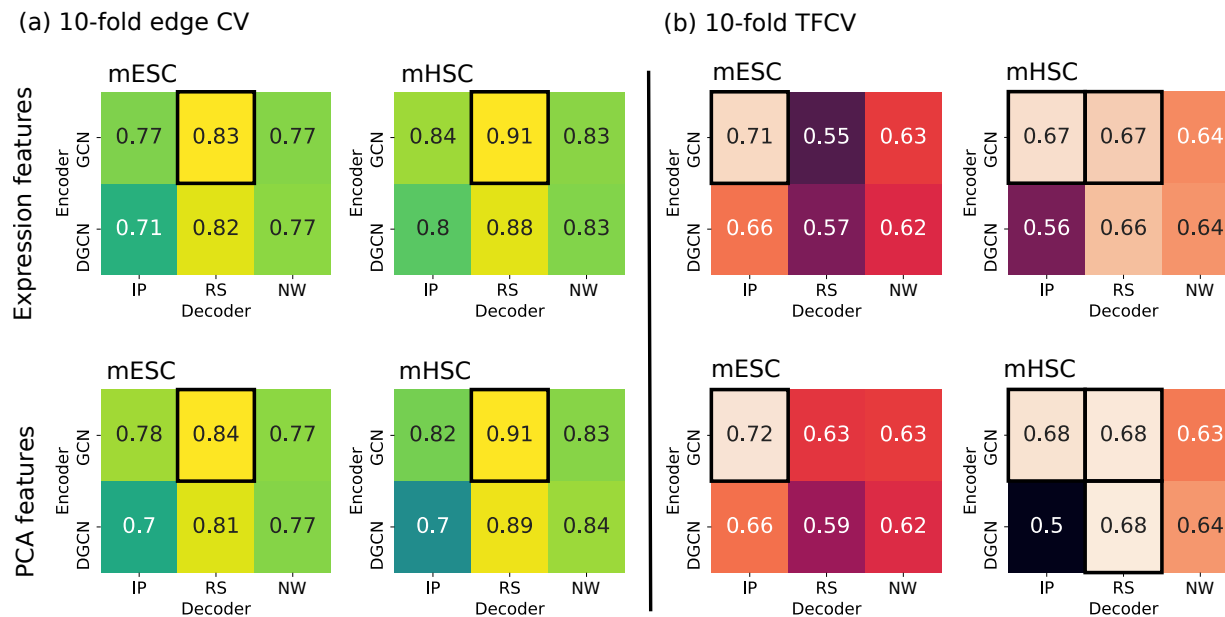


Figure 5.1: Heatmap summarizing the median early precision obtained for (a) 10-fold edge cross-validation, (b) 10-fold TF holdout cross-validation for various encoder-decoder combinations considered for the mESC and mHSC datasets. The top row corresponds to nodes feature being constructed from the expression matrix while the bottom row corresponds to the PCA matrix being used for node features. The colors in the heatmap are anchored at the highest median early precision (bright) and the early precision of a random predictor (dark). A black border around a cell indicates the best median early precision.

Based on these observations, we propose using the GCN-IP autoencoder in cases where there is no known genes regulated by a TF and the GCN-RS autoencoder otherwise. Specifically, we choose the GCN-RS encoder for 10-fold edge holdout evaluation and the GCN-IP encoder for 10-fold TF holdout evaluation for the rest of this study.

Effect of the number of graph convolutions. For the two best performing architectures (GCN-RS, GCN-IP), we wanted to further investigate the effect of the number of graph convolution layers on early precision. To this end, we varied the number of GCN layers for the GCN-RS architecture for 10-fold edge cross-validation for both expression and PCA-based features. Similarly, we varied the number of GCN layers for the GCN-IP architecture for 10-fold TF-holdout cross-validation. Figure 5.2 summarizes our results for the mESC dataset. In general, we did not observe any benefit to increasing the number of GCN layers beyond 2. Moreover, when we increased the number of GCN layers to 10 for GCN-RS Figure 5.2(a), it resulted in a near random performance in some cross-validation evaluations due to numerical instability issues arising from the large size of the neural network. Based on these observations, we set the number of GCN layers to two for the rest of this study.

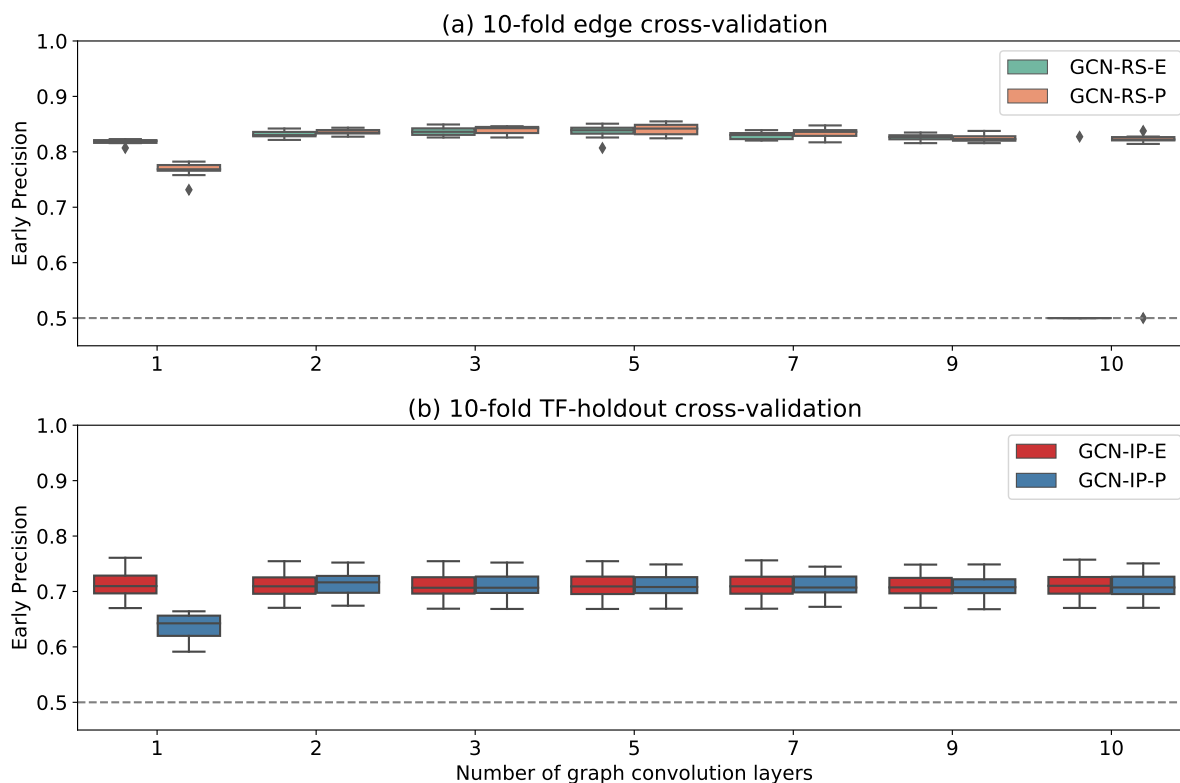


Figure 5.2: Boxplots summarizing the results on the effect of the number of GCN layers (x -axis) on early precision (y -axis) for (a) 10-fold edge cross-validation, (b) 10-fold TF-holdout cross-validation, with the ratio of test positives and negatives set to 1:1. The boxes in green and orange represent the GCN-RS architecture with expression features (GCN-RS-E) and PCA-matrix based features (GCN-PS-P), respectively. The boxes in red and blue represent the GCN-IP architecture with expression features (GCN-IP-E) and PCA-matrix based features (GCN-IP-P), respectively. The gray, dashed horizontal line represents the early precision of a random classifier.

5.4.2 GCN-based autoencoders outperform other techniques for GRN inference

We next analyzed how GCN-based approaches would compare to other approaches for GRN inference from scRNA-seq data. In addition to the mESC and mHSC scRNA-seq datasets, we evaluated performance on the bone-marrow derived macrophages dataset as well [125, 144] (see section 5.2.3 for more details). We chose the mouse non-specific ChIP-seq network as ground-truth and performed 10-fold edge cross-validation and 10-fold TF-holdout cross-validation. Based on our observations on various encoder-decoder combinations, we chose the undirected GCN with RESCAL decoder (GCN-RS) as the preferred architecture for 10-

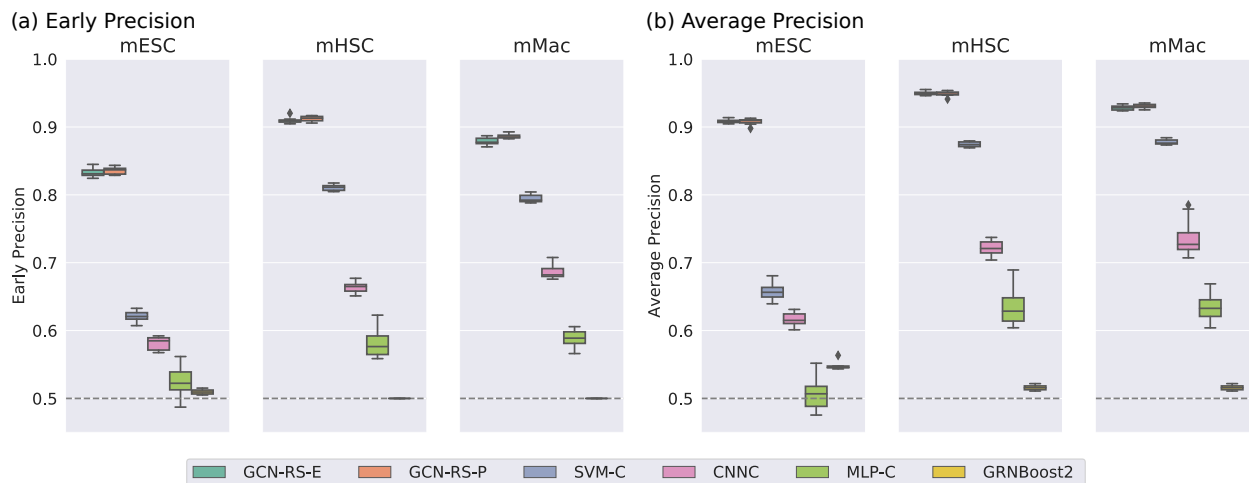


Figure 5.3: Box plots summarizing the results on (a) early precision and (b) average precision for 10-fold edge cross-validation. The subplots show the performance of various algorithms for three mouse scRNA-seq datasets, namely, embryonic stem cell (mESC), hematopoietic stem cell (mHSC), and bone-marrow derived macrophages (mMac), when using non-cell type specific ChIP-seq network as the ground truth. The distributions correspond to the six algorithms tested, namely, GCN-RS-E (teal), GCN-RS-P (orange), SVM-C (blue), CNNC (pink), MLP-C (green), and, GRNBoost2 (yellow). In every boxplot, the box shows the 1st and 3rd quartile, and whiskers denote 1.5 times the interquartile range. Each gray, dashed horizontal line represents the early precision of a random classifier.

fold edge cross-validation, and the undirected GCN with Inner Product decoder (GCN-IP) for 10-fold TF-holdout cross-validation. We included in our analyses both the expression based features as well as the PCA-based features to evaluate if the earlier results held over multiple datasets. We named the four resulting combinations of GCN-based approaches as GCN-RS-E, GCN-RS-P, GCN-IP-E, and GCN-IP-P, where the suffix -E or -P represents expression matrix as node features or the reduced dimensional representation of expression matrix using PCA as node features. We compared the performance of the GCN-based approach to three supervised GRN inference methods for scRNA-seq data, namely, SVM-C, MLP-C, and CNNC [125]. In addition, we also considered GRNBoost2 [57], one of the top performing unsupervised learning approach for GRN inference from scRNA-seq data based on our BEELINE framework [142].

Figure 5.3 summarizes our results on 10-fold edge cross-validation for the three datasets under consideration. We ordered the datasets by the number of single cells measured. The mESC dataset has the lowest number of cells (around 400) and the mMac dataset has the highest number of cells (around 6,000). In general, the GCN-based approaches (GCN-RS-E and GCN-RS-P) achieved the highest early precision for 10-fold edge cross validation, with median values ranging from 0.83 (mESC) to 0.91 (mHSC) (Figure 5.3(a)). This analysis

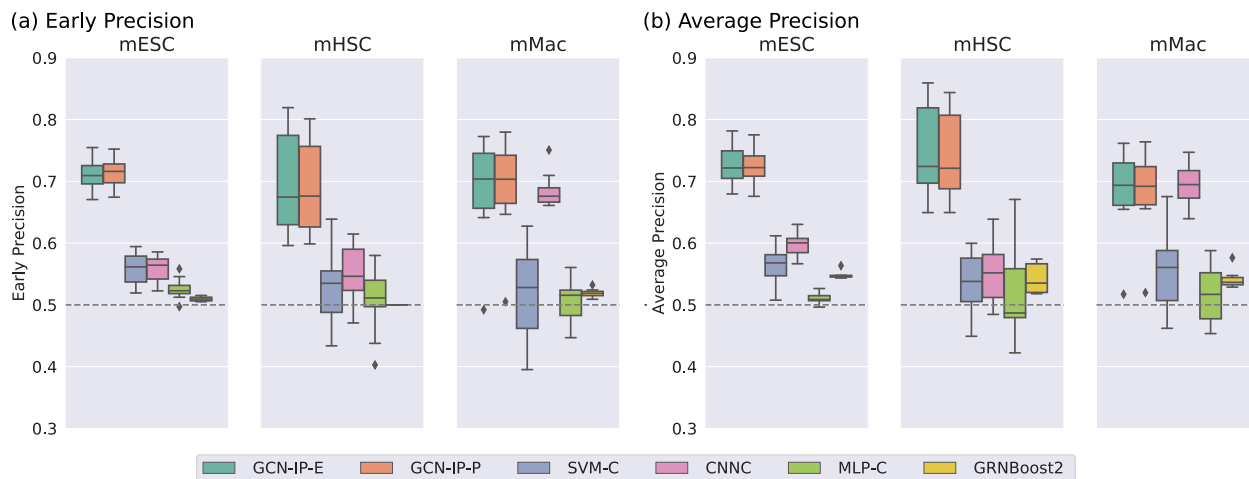


Figure 5.4: Box plot summarizing the results on (a) early precision, and (b) average precision for 10-fold TF-holdout cross-validation. The subplots show the performance of various algorithms for three mouse scRNA-seq datasets, namely, embryonic stem cell (mESC), hematopoietic stem cell (mHSC), and bone-marrow derived macrophages (mMac), when using non-cell type specific ChIP-seq network as the ground truth. The distributions correspond to the six algorithms tested, namely, GCN-IP-E (teal), GCN-IP-P (orange), SVM-C (blue), CNNC (pink), MLP-C (green), and, GRNBoost2 (yellow). In every boxplot, the box shows the 1st and 3rd quartile, and whiskers denote 1.5 times the interquartile range. Each gray, dashed horizontal line represents the early precision of a random classifier.

shows that GCN-RS-P might be a useful alternative when there are a large number of cells in the scRNA-seq dataset, since the reduction in the number of dimensions in the input matrix decreases the number of weights to be learned for the GCN encoder. SVM-C achieved the third best early precision values with median values in the range of 0.62 (mESC) to 0.81 (mHSC). While there was a general increase in the performance of the methods under consideration with the increase in the number of cells in the underlying dataset, SVM-C seems to benefit the most with this increase. The unsupervised learning approach, GRNBoost2, achieved a near random early precision across all three datasets.

The general trends in the results on average precision for 10-fold edge cross-validation (Figure 5.3(b)) were similar to that of early precision. GCN-RS-E and GCN-RS-P achieved the highest average precision, with the median values in the range of 0.91 (mESC) to 0.95 (mHSC). Interestingly, MLP-C marginally under-performed the unsupervised learning method, GRNBoost2, on the mESC dataset, with GRNBoost2 having a slightly higher median average precision of 0.54 and MLP-C achieving a median average precision of 0.51. However, MLP-C’s performance relatively improved in datasets with higher number of cells. Even though CNNC, a method developed specifically for GRN inference from scRNA-seq datasets, performed better than MLP-C and GRNBoost2, it consistently under-performed

the GCN-based approaches and SVM-C.

Next we analyzed how GCN-based encoders would compare against other methods under 10-fold TF holdout evaluation. As mentioned earlier, for this analysis we used the undirected GCN encoder with Inner product decoder (GCN-IP). Figure 5.4 summarizes our results on early precision and average precision for the three datasets under consideration. GCN-IP-E and GCN-IP-P generally had a higher early precision than other methods (Figure 5.4(a)), with median early precision values in the range of 0.67 (mHSC) to 0.71 (mESC). The average precision values of GCN-based approaches also showed a similar trend with values ranging between 0.69 (mMac) and 0.72 (mHSC). The SVM-C method, which was the third best performing method for 10-fold edge cross validation, under-performed GCN-based methods and CNNC across all the datasets, with most median early precision and average precision values close to that of a random predictor (0.5). Interestingly, CNNC also compared favorably to GCN-based approaches in terms of average precision for the mMac dataset, with a median average precision around 0.69 for all three approaches. The results from 10-fold edge and TF holdout cross-validations show that the GCN-based approaches generally outperform other algorithms for GRN inference from scRNA-seq data.

In general, the early precision and average precision values for all algorithms under 10-fold TF holdout cross-validation were lower than that of 10-fold edge cross-validation. In addition, the inter-quartile ranges of early precision and average precision values in the TF holdout cross-validation were generally higher. These result shows that predicting edges for a TF without any prior knowledge is challenging. However, the supervised approaches can still learn some information from other TF-gene pairs in the ground-truth network, and thereby outperform unsupervised learning approaches.

5.4.3 Application to human embryonic stem cell scRNA-seq dataset

To study the properties of the predicted network in more detail, we applied the GCN-RS-E autoencoder to the human embryonic stem cell dataset (hESC) [92]. The expression dataset was processed using the procedure outlined in BEELINE [142]. We trained GCN-RS-E with all the edges in the ground-truth ChIP-seq network as training positives and all other possible outgoing edges from TFs as training negatives. We then obtained the predicted edge weights for every possible edge for all 1,142 nodes in this dataset, of which 292 were TFs. Figure 5.5 summarizes our main observations.

We first analyzed the edge weight distribution for the different types of edges in the predicted network (Figure 5.5(a)). Specifically, we divided all possible edges into three categories: (i) edges going out of genes (red), (ii) edges going out of TFs that are not present in the ground-truth network (blue), and (iii) edges going out of TFs that are present in the ground-truth network (green). Since all 4,597 edges in the ground-truth were used as training positives, it is not surprising that the median score of those edges is the highest at 0.87. All other edges outgoing from TFs that are used as training negatives (328,575 edges) have a median

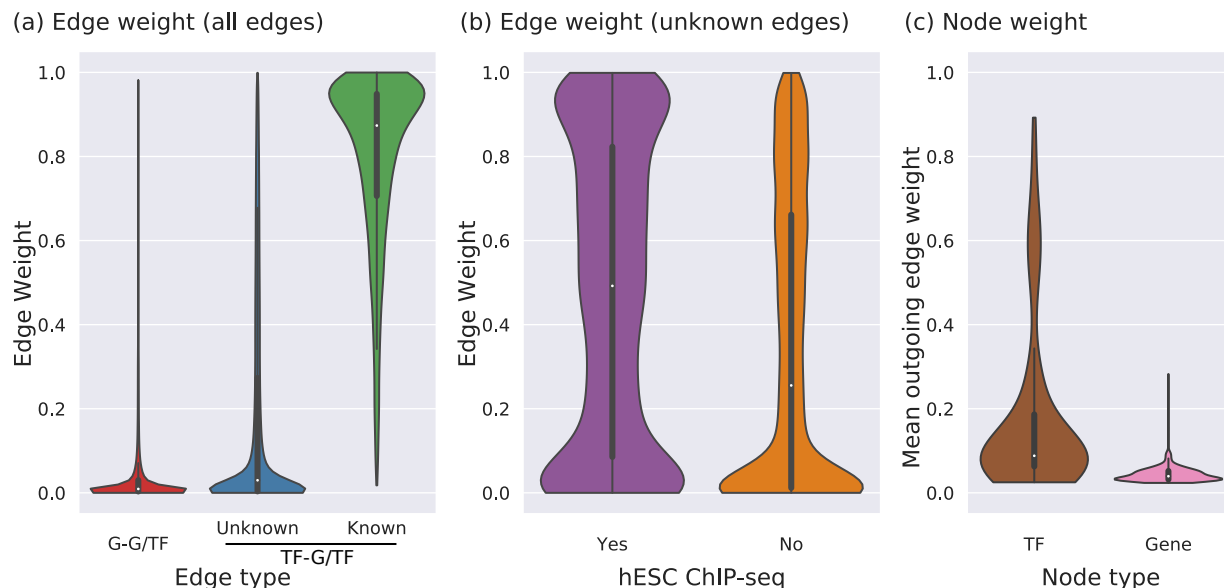


Figure 5.5: Violin plot summarizing results for GCN-RS-E predictions for the hESC dataset. (a) Edge weight distribution of various types of edges in the predicted network, i.e., edges going out of genes (red), unknown edges going out of TFs (blue), edges going out of TFs in the ground-truth network (green) (b) Edge weight distributions of the unknown edges going out of TFs that are present in the hESC ChIP-seq network (purple), or otherwise (orange), restricted to the TFs measured in the ChIP-seq experiments. (c) Mean outgoing edge weight for TF nodes (brown) and gene nodes (pink) in the predicted network. Abbreviations: TF: Transcription Factor, G: Gene.

predicted edge weight of 0.03. Interestingly, edges outgoing from genes, which were neither training positives nor training negatives, have the lowest median edge weight of (0.009). This analysis reveals that the GCN-RS-E can implicitly distinguish between the edges going out of genes from those leaving TFs.

Next, we focused on the edges outgoing from TFs that were not in the ground-truth network, which we used as training negatives. We refer to these as *unknown* edges. Even though we used them as training negatives, it is conceivable that a subset of them correspond to true regulatory interactions that have not yet been experimentally observed. We specifically focused on the 28 TFs for which ChIP-seq measurements on hESCs were available [100, 142]. Of the 31,948 possible edges going out of these 28 TFs (excluding self-loops), 1,750 were present in the non-specific ChIP-seq network used as the ground truth. After ignoring the edges in the ground-truth network, we were left with a total of 30,198 unknown edges. The hESC cell-type specific ChIP-seq network contained 4,855 of these unknown edges. Figure 5.5(b) shows the predicted edge weight distribution of the unknown edges present in hESC ChIP-seq network (purple) and the remaining 25,343 edges that are not present in the hESC ChIP-seq network (orange). Even though all these edges were presented as training

negatives for GCN-RS-E, the median edge weight of the edges present in the hESC ChIP-seq network (0.49) was nearly twice that of edges not present in the hESC ChIP-seq network (0.25), with a p -value $\approx 10^{-76}$ on the two-sample Kolmogorov-Smirnov (K-S) test. This analysis reveals that our GCN-based approach can be a powerful tool to identify cell-type specific gene regulatory edges even when trained on a non-cell type specific network.

Finally, we also analyzed if GCN-RS-E could distinguish between the different node types in the network. For each node in the predicted network, we computed the mean outgoing edge weight. Figure 5.5(c) shows the distribution of these node scores with TF scores shown in brown and gene scores in pink. Even though the node type information was not presented to the GCN-RS-E model, it was able to distinguish between TFs and genes, with the genes having a median node score of 0.03 and TFs having nearly three times that score with a median of 0.09, p -value $\approx 10^{-16}$ on the two-sample K-S test. In addition, we found that only 15 (5.1%) TFs had a lower node score than the median gene node score.

5.5 Discussion

We have presented a novel application of the graph convolutional neural network-based autoencoders to the problem of supervised gene regulatory network inference. We investigated two encoder and three different decoders, i.e., a total of six autoencoders with different encoder-decoder combinations. We performed 10-fold edge holdout and 10-fold TF holdout cross-validations and evaluated the methods based on their median early precision scores. We have made the following key observations. First, the undirected GCN-based encoder generally performed better than the directed GCN-based encoder. This result suggests that even though the underlying GRN is a directed graph, the encoder step can better utilize information flow in both directions between TFs and genes in order to learn a lower dimensional representation of input node features. Second, the choice of decoder depends on whether there are some genes known to be regulated by a TF. Specifically, the RESCAL decoder performs the best when there are some genes known TF's targets. However, if there are no genes known to be regulated by a TF, it may be difficult to learn neural network parameters for the RESCAL decoder. In such cases, the non-parameteric Inner Product decoder offers better predictive performance. Third, the GCN-based autoencoders outperformed other supervised learning techniques in terms of both early precision and average precision values.

Despite the better performance of GCNs compared to other techniques for GRN inference, predicting target genes for TFs when there is no information on its targets still remains a challenging problem. However, our analysis on training the GCN-RS-E autoencoder with non-specific ChIP-seq network with hESC scRNA-seq expression dataset has some encouraging results. The GCN-RS-E autoencoder was able to predict higher scores for edges in the hESC cell-type specific ChIP-seq network, even though only a small fraction (that also appeared in the non-specific network) of those edges were marked as positive examples while

training. This analysis shows that even if cell-type specific regulatory interactions are poorly known, a non-specific ChIP-seq network might be useful as the prior knowledge network. Another avenue for further development would be to integrate multiple cell-type specific scRNA-seq datasets using a tool such as Scanorama [143], and systematically evaluate if the GCN-RS-E autoencoder can better identify cell-type specific interactions even when trained on non-specific ChIP-seq network. One concern with integrating multiple datasets is that the node feature vector might have tens or hundreds of thousands of dimensions. In such cases, the GCN-RS-P with the PCA-based features might be useful since it offers more flexibility on the number of parameters in the neural network with comparable performance as the GCN-RS-E.

There are several ways to further improve on the current GCN-autoencoders. Our choice of a GCN-based encoder was primarily guided by its simple architecture and performance compared to other architectures [148, 149] on node classification. In this study, though, we have presented a comprehensive analysis of various encoder-decoder combinations for link prediction. Therefore, other graph neural networks such as GraphSAGE [150], graph attention networks [151] might be worth exploring systematically as autoencoders. Another avenue for improvement is to use relational GCN-based autoencoders [138] as a framework for integrating multiple relation types between TFs and genes into the ground-truth network, such as physical and genetic interactions [97], or co-expression data. This multi-edge formulation will provide more ways to integrate information in the encoder, potentially resulting in better predictive power. In addition, a gene regulatory interaction can either be activatory or inhibitory. The sign of the regulatory edge can be also be learned by modifying existing decoders, for example, learning a different weight matrix in the RESCAL decoder that represents sign of the edge. Overall, GCNs are a flexible and promising framework for addressing the next frontier of challenges in GRN inference.

Chapter 6

Conclusions

In this thesis, we have presented solutions to two problems that arise in the context of mathematical modeling of biological systems. These problems we addressed lie broadly in the areas of efficiently improving existing mathematical models and developing wiring diagrams underlying new mathematical models that can leverage recent advances in experimental techniques for single cell measurements.

In Chapter 2, we presented our novel approach CROSSPLAN, for scaling-up the mathematical modeling cycle, with a focus on prioritizing and planning large scale genetic cross experiments aimed at model validation. CROSSPLAN uses genetic cross graph, an abstraction for organizing mutants and the genetic cross experiments needed to make them. Subsequently in Chapter 3, we formulated several natural problems related to efficient synthesis of a target mutant from a set of source mutants without the need for constructing a genetic cross graph. In both chapters, we apply our experiment planning algorithms to simulations of a mathematical model of the budding yeast cell cycle.

We next focused on inferring GRNs from scRNA-seq data. These GRNs can serve as the wiring diagram for mathematical models of complex biological process such as cellular differentiation. In Chapter 4, we presented our comprehensive evaluation framework, BEELINE, to investigate current state-of-the-art techniques for GRN inference in terms of accuracy, stability, and scalability. In this context we developed BoolODE, a strategy to simulate single-cell transcriptional data from synthetic and Boolean networks. Based on the observation that the performance of many of the existing methods for GRN inference was less than ideal, we presented several recommendations to the users of these methods. BEELINE provides a flexible interface for benchmarking new methods for GRN inference against existing techniques.

The methods evaluated in BEELINE were unsupervised, so in order to take advantage of prior knowledge in terms of known regulatory interactions, we developed a novel supervised learning technique using GCN-based autoencoders for GRN inference from scRNA-seq data in Chapter 5. Our GCN-based method compares favorably to other supervised learning techniques, as well as unsupervised techniques for GRN inference from scRNA-seq. We also demonstrated that our supervised approach can identify cell-type specific regulatory interactions even when trained on non-cell type specific GRN.

We envision several directions of future research for the work that we have presented in this thesis. Our comprehensive benchmarking of existing scRNA-seq methods in BEELINE (Chapter 4) was primarily focused on evaluating unsupervised techniques. One of the key results in Chapter 5 was that the methods that can leverage prior knowledge offer superior performance in GRN inference compared to the unsupervised techniques. A natural extension of BEELINE would be to extend the benchmarking framework to supervised learning methods.

Recent advances in experimental techniques for single-cell measurements across multiple modalities [152] offer tremendous opportunity in our ability to dissect complex biological behaviors. For instance, GRN inference methods could incorporate chromatin accessibility data using the single-cell Assay of Transposase Accessible Chromatin sequencing (scATAC-seq) [153]. The scATAC-seq, along with scRNA-seq data measured in the same cell or a matching sample, can be used to confirm whether the predicted TF-gene interactions are supported by the cell-type specific ATAC-seq peaks. This type of data integration could result in a more accurate GRN inference.

The GRN so obtained would serve as a wiring diagram for the mathematical model that we would like to build to study the underlying biological system. One could employ BoolODE, a tool we developed for automatic conversion of GRNs to ODE models, to obtain a model that can be simulated. However, we also showed in Chapter 4 that the current GRN inference methods do not accurately capture the underlying system dynamics or steady states when we simulated the top- k predicted networks using BoolODE. An exciting direction for future research would involve leveraging graph neural networks in combination with BoolODE to build predictive models that can accurately capture underlying system dynamics. The autoencoder architectures that combine graph neural networks with ODE-based decoders presented by Kipf *et al.* [154] might be a useful starting point for developing such methods.

A mathematical model obtained from scRNA-seq or by integrating multiple data types, would still be an initial guess reflecting our current understanding of the biological system under consideration. The predictive capabilities of such a model can be further investigated by performing model simulations under new scenarios, e.g., combinatorial gene knockouts. We envision that one can leverage experiment planning techniques such as the ones presented in Chapters 2 and 3, by adapting them to the relevant experimental workflow. Finally, one can develop novel frameworks and algorithms for reconciling any differences between predictions and experiments and repeat this process thereby scaling up the mathematical modeling cycle.

Bibliography

- [1] N. Le Novère. Quantitative and logic modelling of molecular and gene networks. *Nature Reviews Genetics*, 16(3):146–158, Mar 2015.
- [2] A. C. Villani, R. Satija, G. Reynolds, S. Sarkizova, K. Shekhar, J. Fletcher, M. Griesbeck, A. Butler, S. Zheng, S. Lazo, L. Jardine, D. Dixon, E. Stephenson, E. Nilsson, I. Grundberg, D. McDonald, A. Filby, W. Li, P. L. De Jager, O. Rozenblatt-Rosen, A. A. Lane, M. Haniffa, A. Regev, and N. Hacohen. Single-cell RNA-seq reveals new types of human blood dendritic cells, monocytes, and progenitors. *Science*, 356(6335), 04 2017.
- [3] O. Rozenblatt-Rosen, M. J. T. Stubbington, A. Regev, and S. A. Teichmann. The Human Cell Atlas: from vision to reality. *Nature*, 550(7677):451–453, 10 2017.
- [4] Victoria Moignard, Steven Woodhouse, Laleh Haghverdi, Andrew J. Lilly, Yosuke Tanaka, Adam C. Wilkinson, Florian Buettner, Iain C. Macaulay, Wajid Jawaid, Evangelia Diamanti, Shin-Ichi Nishikawa, Nir Piterman, Valerie Kouskoff, Fabian J. Theis, Jasmin Fisher, and Berthold Göttgens. Decoding the regulatory network of early blood development from single-cell gene expression measurements. *Nature Biotechnology*, 33(3):269–276, Mar 2015.
- [5] Steven Woodhouse, Nir Piterman, Christoph M Wintersteiger, Berthold Göttgens, and Jasmin Fisher. Scns: a graphical tool for reconstructing executable regulatory networks from single-cell genomic data. *BMC systems biology*, 12(1):59, 2018.
- [6] Nicholas M Luscombe, M Madan Babu, Haiyuan Yu, Michael Snyder, Sarah A Teichmann, and Mark Gerstein. Genomic analysis of regulatory network dynamics reveals large topological changes. *Nature*, 431(7006):308, 2004.
- [7] Jason Ernst, Oded Vainas, Christopher T Harbison, Itamar Simon, and Ziv Bar-Joseph. Reconstructing dynamic regulatory maps. *Molecular systems biology*, 3(1):74, 2007.
- [8] Katherine C. Chen, Laurence Calzone, Attila Csikasz-Nagy, Frederick R. Cross, Bela Novak, and John J. Tyson. Integrative analysis of cell cycle control in budding yeast. *Molecular Biology of the Cell*, 15(8):3841–3862, August 2004.
- [9] Pavel Kraikivski, Katherine C. Chen, Teeraphan Laomettachit, T. M. Murali, and John J. Tyson. From START to FINISH: Computational analysis of cell cycle control in budding yeast. *NPJ Systems Biology and Applications*, 1:15016, 2015.

- [10] Steven Nathaniel Steinway, Jorge Gomez Tejeda Zañudo, Paul J Michel, David J Feith, Thomas P Loughran, and Reka Albert. Combinatorial interventions inhibit tgf [beta]-driven epithelial-to-mesenchymal transition and support hybrid cellular phenotypes. *NPJ Systems Biology and Applications*, 1:15014, 2015.
- [11] JS Edwards, R Ramakrishna, CH Schilling, and BO Palsson. Metabolic flux balance analysis. *Bioprocess Technology*, 24:13–58, 1997.
- [12] Jeremy S Edwards and Bernhard O Palsson. Metabolic flux balance analysis and the in silico analysis of *Escherichia coli* k-12 gene deletions. *BMC Bioinformatics*, 1(1):1, 2000.
- [13] Daniel Segre, Dennis Vitkup, and George M Church. Analysis of optimality in natural and perturbed metabolic networks. *Proceedings of the National Academy of Sciences*, 99(23):15112–15117, 2002.
- [14] Steffen Klamt and Jörg Stelling. Combinatorial complexity of pathway analysis in metabolic networks. *Molecular biology reports*, 29(1-2):233–236, 2002.
- [15] Radhakrishnan Mahadevan, Jeremy S Edwards, and Francis J Doyle III. Dynamic flux balance analysis of diauxic growth in *Escherichia coli*. *Biophysical journal*, 83(3):1331–1340, 2002.
- [16] Jong Min Lee, Erwin P Gianchandani, James A Eddy, and Jason A Papin. Dynamic analysis of integrated signaling, metabolic, and regulatory networks. *PLoS computational biology*, 4(5):e1000086, 2008.
- [17] M. Kanehisa, Y. Sato, M. Kawashima, M. Furumichi, and M. Tanabe. KEGG as a reference resource for gene and protein annotation. *Nucleic Acids Research*, 44(D1):D457–462, Jan 2016.
- [18] K. Kandasamy, S. S. Mohan, R. Raju, S. Keerthikumar, G. S. Kumar, A. K. Venugopal, D. Telikicherla, J. D. Navarro, S. Mathivanan, C. Pecquet, S. K. Gollapudi, S. G. Tattikota, S. Mohan, H. Padhukasahasram, Y. Subbannayya, R. Goel, H. K. Jacob, J. Zhong, R. Sekhar, V. Nanjappa, L. Balakrishnan, R. Subbaiah, Y. L. Ramachandra, B. A. Rahiman, T. S. Prasad, J. X. Lin, J. C. Houtman, S. Desiderio, J. C. Renauld, S. N. Constantinescu, O. Ohara, T. Hirano, M. Kubo, S. Singh, P. Khatri, S. Draghici, G. D. Bader, C. Sander, W. J. Leonard, and A. Pandey. NetPath: a public resource of curated signal transduction pathways. *Genome Biology*, 11(1):R3, 2010.
- [19] A. Fabregat, K. Sidiropoulos, P. Garapati, M. Gillespie, K. Hausmann, R. Haw, B. Jassal, S. Jupe, F. Korninger, S. McKay, L. Matthews, B. May, M. Milacic, K. Rothfels, V. Shamovsky, M. Webber, J. Weiser, M. Williams, G. Wu, L. Stein, H. Hermjakob, and P. D’Eustachio. The Reactome pathway Knowledgebase. *Nucleic Acids Research*, 44(D1):D481–487, Jan 2016.

- [20] A Ryll, J Bucher, A Bonin, S Bongard, E Goncalves, J Saez-Rodriguez, J Niklas, and S Klamt. A model integration approach linking signalling and gene-regulatory logic with kinetic metabolic models. *Bio Systems*, 124:26–38, October 2014.
- [21] Claudine Chaouiya, Duncan Brenguier, Sarah M Keating, Aurlien Naldi, Martijn P van Iersel, Nicolas Rodriguez, Andreas Drger, Finja Bchel, Thomas Cokelaer, Bryan Kowal, Benjamin Wicks, Emanuel Goncalves, Julien Dorier, Michel Page, Pedro T Monteiro, Axel von Kamp, Ioannis Xenarios, Hidde de Jong, Michael Hucka, Steffen Klamt, Denis Thieffry, Nicolas Le Novre, Julio Saez-Rodriguez, and Tom Helikar. SBML qualitative models: a model representation format and infrastructure to foster interactions between qualitative modelling formalisms and tools. *BMC Systems Biology*, 7:135, December 2013.
- [22] Rainer Poltz, Raimo Franke, Katrin Schweitzer, Steffen Klamt, Ernst-Dieter Gilles, and Michael Naumann. Logical network of genotoxic stress-induced nf-b signal transduction predicts putative target structures for therapeutic intervention strategies. *Advances and applications in bioinformatics and chemistry : AABC*, 2:125–138, 2009.
- [23] Julio Saez-Rodriguez, Leonidas G Alexopoulos, Jonathan Epperlein, Regina Samaga, Douglas A Lauffenburger, Steffen Klamt, and Peter K Sorger. Discrete logic modelling as a means to link protein signalling networks with functional analysis of mammalian signal transduction. *Molecular Systems Biology*, 5:331, 2009.
- [24] Ulrike Wittig, Renate Kania, Martin Golebiewski, Maja Rey, Lei Shi, Lenneke Jong, Enkhjargal Alгаа, Andreas Weidemann, Heidrun Sauer-Danzwith, Saqib Mir, et al. Sabio-rkdatabase for biochemical reaction kinetics. *Nucleic Acids Research*, 40(D1):D790–D796, 2011.
- [25] Sandra Placzek, Ida Schomburg, Antje Chang, Lisa Jeske, Marcus Ulbrich, Jana Tillack, and Dietmar Schomburg. Brenda in 2017: new perspectives and new tools in brenda. *Nucleic acids research*, page gkw952, 2016.
- [26] Tobias S Christensen, Ana Paula Oliveira, and Jens Nielsen. Reconstruction and logical modeling of glucose repression signaling pathways in *saccharomyces cerevisiae*. *BMC Systems Biology*, 3(1):7, 2009.
- [27] T E Ideker, V Thorsson, and R M Karp. Discovery of regulatory interactions through perturbation: inference and experimental design. *Pacific Symposium on Biocomputing*, pages 305–316, 2000.
- [28] Chen H. Yeang, H. Craig Mak, Scott McCuine, Christopher Workman, Tommi Jaakkola, and Trey Ideker. Validation and refinement of gene-regulatory pathways on a network of physical interactions. *Genome Biology*, 6(7):R62+, 2005.

- [29] Christian L Barrett and Bernhard O Palsson. Iterative reconstruction of transcriptional regulatory networks: an algorithmic approach. *PLoS Computational Biology*, 2:e52, May 2006.
- [30] Ewa Szczurek, Irit Gat-Viks, Jerzy Tiuryn, and Martin Vingron. Elucidating regulatory mechanisms downstream of a signaling pathway using informative experiments. *Molecular Systems Biology*, 5:287, 2009.
- [31] Clemens Kreutz and Jens Timmer. Systems biology: experimental design. *The FEBS journal*, 276:923–942, Feb 2009.
- [32] Samuel Bandara, Johannes P Schloder, Roland Eils, Hans Georg Bock, and Tobias Meyer. Optimal experimental design for parameter estimation of a cell signaling model. *PLoS Computational Biology*, 5:e1000558, Nov 2009.
- [33] Edouard Pauwels, Christian Lajaunie, and Jean-Philippe Vert. A bayesian active learning strategy for sequential experimental design in systems biology. *BMC Systems Biology*, 8:102, Sep 2014.
- [34] Joshua F Apgar, Jared E Toettcher, Drew Endy, Forest M White, and Bruce Tidor. Stimulus design for model selection and validation in cell signaling. *PLoS Computational Biology*, 4:e30, Feb 2008.
- [35] Andreas Kremling, Sophia Fischer, Kapil Gadkar, Francis J Doyle, Thomas Sauter, Eric Bullinger, Frank Allgower, and Ernst D Gilles. A benchmark for methods in reverse engineering and model discrimination: problem formulation and solutions. *Genome Research*, 14:1773–1785, Sep 2004.
- [36] Bence Melykuti, Elias August, Antonis Papachristodoulou, and Hana El-Samad. Discriminating between rival biochemical network models: three approaches to optimal experiment design. *BMC Systems Biology*, 4:38, Apr 2010.
- [37] Heather A Harrington, Kenneth L Ho, Thomas Thorne, and Michael P H Stumpf. Parameter-free model discrimination criterion based on steady-state coplanarity. *Proceedings of the National Academy of Sciences of the United States of America*, 109:15746–15751, Sep 2012.
- [38] Frederick R Cross, Vincent Archambault, Mary Miller, and Martha Klovstad. Testing a mathematical model of the yeast cell cycle. *Molecular Biology of the Cell*, 13(1):52–70, 2002.
- [39] Neil R Adames, P Logan Schuck, Katherine C Chen, TM Murali, John J Tyson, and Jean Peccoud. Experimental testing of a new integrated model of the budding yeast START transition. *Molecular Biology of the Cell*, 26(22):3966–3984, Nov 2015.

- [40] D. Deutscher, I. Meilijson, S. Schuster, and E. Rupp. Can single knockouts accurately single out gene functions? *BMC Systems Biology*, 2:50, Jun 2008.
- [41] D. Segre, A. Deluna, G. M. Church, and R. Kishony. Modular epistasis in yeast metabolism. *Nature Genetics*, 37(1):77–83, Jan 2005.
- [42] David Deutscher, Isaac Meilijson, Martin Kupiec, and Eytan Rupp. Multiple knock-out analysis of genetic robustness in the yeast metabolic network. *Nature Genetics*, 38:993–998, Sep 2006.
- [43] B. Papp, C. Pal, and L. D. Hurst. Metabolic network analysis of the causes and evolution of enzyme dispensability in yeast. *Nature*, 429(6992):661–664, Jun 2004.
- [44] Michael Costanzo, Benjamin VanderSluis, Elizabeth N Koch, Anastasia Baryshnikova, Carles Pons, Guihong Tan, Wen Wang, Matej Usaj, Julia Hanchard, Susan D Lee, et al. A global genetic interaction network maps a wiring diagram of cellular function. *Science*, 353(6306), Sep 2016.
- [45] Allon Wagner, Aviv Regev, and Nir Yosef. Revealing the vectors of cellular identity with single-cell genomics. *Nat. Biotechnol.*, 34(11):1145–1160, Nov 2016.
- [46] Peter V. Kharchenko, Lev Silberstein, and David T. Scadden. Bayesian approach to single-cell differential expression analysis. *Nat. Methods*, 11(7):740–742, May 2014.
- [47] Florian Buettner, Kedar N. Natarajan, F. Paolo Casale, Valentina Proserpio, Antonio Scialdone, Fabian J. Theis, Sarah A. Teichmann, John C. Marioni, and Oliver Stegle. Computational analysis of cell-to-cell heterogeneity in single-cell RNA-sequencing data reveals hidden subpopulations of cells. *Nat. Biotechnol.*, 33(2):155–160, Jan 2015.
- [48] V. A. Huynh-Thu, A. Irrthum, L. Wehenkel, and P. Geurts. Inferring regulatory networks from expression data using tree-based methods. *PLoS ONE*, 5(9), Sep 2010.
- [49] S. Kim. ppcor: An R Package for a Fast Calculation to Semi-partial Correlation Coefficients. *Commun Stat Appl Methods*, 22(6):665–674, Nov 2015.
- [50] A. T. Specht and J. Li. LEAP: constructing gene co-expression networks for single-cell RNA-sequencing data using pseudotime ordering. *Bioinformatics*, 33(5):764–766, 03 2017.
- [51] Hirotaka Matsumoto, Hisanori Kiryu, Chikara Furusawa, Minoru SH Ko, Shigeru BH Ko, Norio Gouda, Tetsutaro Hayashi, and Itoshi Nikaido. Scode: an efficient regulatory network inference algorithm from single-cell rna-seq during differentiation. *Bioinformatics*, 33(15):2314–2321, 2017.
- [52] Thalia E Chan, Michael PH Stumpf, and Ann C Babbie. Gene regulatory network inference from single-cell data using multivariate information measures. *Cell systems*, 5(3):251–267, 2017.

- [53] S. Aibar, C. B. Gonzalez-Blas, T. Moerman, V. A. Huynh-Thu, H. Imrichova, G. Hulselmans, F. Rambow, J. C. Marine, P. Geurts, J. Aerts, J. van den Oord, Z. K. Atak, J. Wouters, and S. Aerts. SCENIC: single-cell regulatory network inference and clustering. *Nature Methods*, 14(11):1083–1086, 2017.
- [54] Nan Papili Gao, SM Minhaz Ud-Dean, Olivier Gandrillon, and Rudiyanto Gunawan. Sincerities: inferring gene regulatory networks from time-stamped single cell transcriptional expression profiles. *Bioinformatics*, 34(2):258–266, 2018.
- [55] M. Sanchez-Castillo, D. Blanco, I. M. Tienda-Luna, M. C. Carrion, and Y. Huang. A Bayesian framework for the inference of gene regulatory networks from time and pseudo-time series data. *Bioinformatics*, 34(6):964–970, 03 2018.
- [56] Xiaojie Qiu, Arman Rahimzamani, Li Wang, Qi Mao, Timothy Durham, José L McFaline-Figueroa, Lauren Saunders, Cole Trapnell, and Sreeram Kannan. Towards inferring causal gene regulatory networks from single cell expression measurements. *bioRxiv*, 2018.
- [57] Thomas Moerman, Sara Aibar Santos, Carmen Bravo Gonzlez-Blas, Jaak Simm, Yves Moreau, Jan Aerts, and Stein Aerts. GRNBoost2 and Arboreto: efficient and scalable inference of gene regulatory networks. *Bioinformatics*, 11 2018.
- [58] Pierre-Cyril Aubin-Frankowski and Jean-Philippe Vert. Gene regulation inference from single-cell rna-seq data with linear differential equations and velocity inference. *bioRxiv*, 2018.
- [59] Atul Deshpande, Li-Fang Chu, Ron Stewart, and Anthony Gitter. Network inference with granger causality ensembles on single-cell transcriptomic data. *bioRxiv*, 2019.
- [60] V. A. Huynh-Thu and G. Sanguinetti. Combining tree-based and dynamical systems for the inference of gene regulatory networks. *Bioinformatics*, 31(10):1614–1622, May 2015.
- [61] H. Chen, J. Guo, S. K. Mishra, P. Robson, M. Niranjana, and J. Zheng. Single-cell transcriptional analysis to uncover regulatory circuits driving cell fate decisions in early mouse development. *Bioinformatics*, 31(7):1060–1066, Apr 2015.
- [62] C. Y. Lim, H. Wang, S. Woodhouse, N. Piterman, L. Wernisch, J. Fisher, and B. Gottgens. BTR: training asynchronous Boolean models using single-cell expression data. *BMC Bioinformatics*, 17(1):355, Sep 2016.
- [63] F. K. Hamey, S. Nestorowa, S. J. Kinston, D. G. Kent, N. K. Wilson, and B. Gottgens. Reconstructing blood stem cell regulatory network models from single-cell molecular profiles. *Proc. Natl. Acad. Sci. U.S.A.*, 114(23):5822–5829, 06 2017.

- [64] D. Marbach, R. J. Prill, T. Schaffter, C. Mattiussi, D. Floreano, and G. Stolovitzky. Revealing Strengths and Weaknesses of Methods for Gene Network Inference. *Proceedings of the National Academy of Sciences*, 107(14):6286–6291, 2010.
- [65] S. Chen and J. C. Mar. Evaluating methods of inferring gene regulatory networks highlights their lack of performance for single cell gene expression data. *BMC Bioinformatics*, 19(1):232, 06 2018.
- [66] Wouter Saelens, Robrecht Cannoodt, Helena Todorov, and Yvan Saeys. A comparison of single-cell trajectory inference methods. *Nature Biotechnology*, 37(5):547–554, 2019.
- [67] Nir Atias, Michal Gershenson, Katia Labazin, and Roded Sharan. Experimental design schemes for learning boolean network models. *Bioinformatics (Oxford, England)*, 30:i445–i452, Sep 2014.
- [68] S L Forsburg. The art and design of genetic screens: yeast. *Nature Reviews Genetics*, 2:659–668, September 2001.
- [69] Damian R Page and Ueli Grossniklaus. The art and design of genetic screens: Arabidopsis thaliana. *Nature reviews. Genetics*, 3:124–136, February 2002.
- [70] Daniel St Johnston. The art and design of genetic screens: Drosophila melanogaster. *Nature Reviews Genetics*, 3:176–188, March 2002.
- [71] John Paul Shen, Dongxin Zhao, Roman Sasik, Jens Luebeck, Amanda Birmingham, Ana Bojorquez-Gomez, Katherine Licon, Kristin Klepper, Daniel Pekin, Alex N Beckett, Kyle Salinas Sanchez, Alex Thomas, Chih-Chung Kuo, Dan Du, Assen Roguev, Nathan E Lewis, Aaron N Chang, Jason F Kreisberg, Nevan Krogan, Lei Qi, Trey Ideker, and Prashant Mali. Combinatorial crispr-cas9 screens for de novo mapping of genetic interactions. *Nature Methods*, 14:573–576, June 2017.
- [72] Patricia G Cipriani and Fabio Piano. RNAi methods and screening: Rnai based high-throughput genetic interaction screening. *Methods in Cell Biology*, 106:89–111, 2011.
- [73] Michael Ku Yu, Michael Kramer, Janusz Dutkowsk, Rohith Srivas, Katherine Licon, Jason F Kreisberg, Cherie T Ng, Nevan Krogan, Roded Sharan, and Trey Ideker. Translation of genotype to phenotype by a hierarchy of cell subsystems. *Cell Systems*, 2(2):77–88, 2016.
- [74] David A. Orlando, Charles Y. Lin, Allister Bernard, Jean Y. Wang, Joshua E. S. Socolar, Edwin S. Iversen, Alexander J. Hartemink, and Steven B. Haase. Global control of cell-cycle transcription by coupled CDK and network oscillators. *Nature*, 453(7197):944–947, May 2008.
- [75] Sahand Jamal Rahi, Kresti Pecani, Andrej Ondracka, Catherine Oikonomou, and Frederick R Cross. The cdk-apc/c oscillator predominantly entrains periodic cell-cycle transcription. *Cell*, 165(2):475–487, 2016.

- [76] B. Zetsche, M. Heidenreich, P. Mohanraju, I. Fedorova, J. Kneppers, E. M. DeGennaro, N. Winblad, S. R. Choudhury, O. O. Abudayyeh, J. S. Gootenberg, W. Y. Wu, D. A. Scott, K. Severinov, J. van der Oost, and F. Zhang. Multiplex gene editing by CRISPR-Cpf1 using a single crRNA array. *Nature Biotechnology*, 35(1):31–34, Jan 2017.
- [77] F. Port, H. M. Chen, T. Lee, and S. L. Bullock. Optimized CRISPR/Cas tools for efficient germline and somatic genome engineering in *Drosophila*. *Proc. Natl. Acad. Sci. U.S.A.*, 111(29):E2967–2976, Jul 2014.
- [78] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman & Co., San Francisco, CA, 1979.
- [79] P. Klein. Lecture notes on combinatorial optimization. Department of Computer Science, Brown University, Providence, RI, 1990.
- [80] H. N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 448–456, 1983.
- [81] R. Niedermeier. *Invitation to Fixed Parameter Algorithms*. Oxford University Press, New York, NY, 2006.
- [82] D. E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison Wesley, Inc., Reading, MA, 1974.
- [83] C. L. Poirel, R. R. Rodrigues, K. C. Chen, J. J. Tyson, and T. M. Murali. Top-down network analysis to drive bottom-up modeling of physiological processes. *Journal of Computational Biology*, 20(5):409–418, May 2013.
- [84] Aditya Pratapa, Neil Adames, Pavel Kraikivski, Nicholas Franzese, John J Tyson, Jean Peccoud, and TM Murali. Crossplan: systematic planning of genetic crosses to validate mathematical models. *Bioinformatics*, 34(13):2237–2244, 2018.
- [85] Thomas Schaffter, Daniel Marbach, and Dario Floreano. GeneNetWeaver: in Silico Benchmark Generation and Performance Profiling of Network Inference Methods. *Bioinformatics*, 27(16):2263–2270, 2011.
- [86] A. Ocone, L. Haghverdi, N. S. Mueller, and F. J. Theis. Reconstructing gene regulatory dynamics from high-dimensional single-cell snapshot data. *Bioinformatics*, 31(12):89–96, Jun 2015.
- [87] Clare E. Giacomantonio and Geoffrey J. Goodhill. A boolean model of the gene regulatory network underlying mammalian cortical area development. *PLoS Computational Biology*, 6(9):e1000936, 2010.

- [88] Anna Lovrics, Yu Gao, Bianka Juhász, István Bock, Helen M. Byrne, András Dinnyés, and Krisztián A. Kovács. Boolean modelling reveals new regulatory connections between transcription factors orchestrating the development of the ventral spinal cord. *PLoS ONE*, 9(11):e111430, 2014.
- [89] Jan Krumsiek, Carsten Marr, Timm Schroeder, and Fabian J. Theis. Hierarchical differentiation of myeloid progenitors is encoded in the transcription factor network. *PLoS ONE*, 6(8):e22649, 2011.
- [90] Osiris Ríos, Sara Frias, Alfredo Rodríguez, Susana Kofman, Horacio Merchant, Leda Torres, and Luis Mendoza. A Boolean network model of human gonadal sex determination. *Theoretical Biology and Medical Modelling*, 12(1):26, 2015.
- [91] K. Street, D. Risso, R. B. Fletcher, D. Das, J. Ngai, N. Yosef, E. Purdom, and S. Dudoit. Slingshot: cell lineage and pseudotime inference for single-cell transcriptomics. *BMC Genomics*, 19(1):477, Jun 2018.
- [92] Li-Fang Chu, Ning Leng, Jue Zhang, Zhonggang Hou, Daniel Mamott, David T. Vereide, Jee Choi, Christina Kendzierski, Ron Stewart, and James A. Thomson. Single-cell RNA-seq reveals novel regulators of human embryonic stem cell differentiation to definitive endoderm. *Genome Biology*, 17(1):173, Dec 2016.
- [93] A. K. Shalek, R. Satija, J. Shuga, J. J. Trombetta, D. Gennert, D. Lu, P. Chen, R. S. Gertner, J. T. Gaublomme, N. Yosef, S. Schwartz, B. Fowler, S. Weaver, J. Wang, X. Wang, R. Ding, R. Raychowdhury, N. Friedman, N. Hacohen, H. Park, A. P. May, and A. Regev. Single-cell RNA-seq reveals dynamic paracrine control of cellular variation. *Nature*, 510(7505):363–369, Jun 2014.
- [94] S. Nestorowa, F. K. Hamey, B. Pijuan Sala, E. Diamanti, M. Shepherd, E. Laurenti, N. K. Wilson, D. G. Kent, and B. Gottgens. A single-cell resolution map of mouse hematopoietic stem and progenitor cell differentiation. *Blood*, 128(8):20–31, 2016.
- [95] J. G. Camp, K. Sekine, T. Gerber, H. Loeffler-Wirth, H. Binder, M. Gac, S. Kanton, J. Kageyama, G. Damm, D. Seehofer, L. Belicova, M. Bickle, R. Barsacchi, R. Okuda, E. Yoshizawa, M. Kimura, H. Ayabe, H. Taniguchi, T. Takebe, and B. Treutlein. Multilineage communication regulates human liver bud development from pluripotency. *Nature*, 546(7659):533–538, 2017.
- [96] T. Hayashi, H. Ozaki, Y. Sasagawa, M. Umeda, H. Danno, and I. Nikaido. Single-cell full-length total RNA sequencing uncovers dynamics of recursive splicing and enhancer RNAs. *Nature Communications*, 9(1):619, 02 2018.
- [97] D. Szklarczyk, A. Franceschini, S. Wyder, K. Forslund, D. Heller, J. Huerta-Cepas, M. Simonovic, A. Roth, A. Santos, K. P. Tsafou, M. Kuhn, P. Bork, L. J. Jensen, and C. von Mering. STRING v10: protein-protein interaction networks, integrated over the tree of life. *Nucleic Acids Research*, 43(Database issue):D447–452, Jan 2015.

- [98] Z. P. Liu, C. Wu, H. Miao, and H. Wu. RegNetwork: an integrated database of transcriptional and post-transcriptional regulatory networks in human and mouse. *Database (Oxford)*, 2015, 2015.
- [99] H. Han, J. W. Cho, S. Lee, A. Yun, H. Kim, D. Bae, S. Yang, C. Y. Kim, M. Lee, E. Kim, S. Lee, B. Kang, D. Jeong, Y. Kim, H. N. Jeon, H. Jung, S. Nam, M. Chung, J. H. Kim, and I. Lee. TRRUST v2: an expanded reference database of human and mouse transcriptional regulatory interactions. *Nucleic Acids Res.*, 46(D1):D380–D386, 01 2018.
- [100] Shinya Oki, Tazro Ohta, Go Shioi, Hideki Hatanaka, Osamu Ogasawara, Yoshihiro Okuda, Hideya Kawaji, Ryo Nakaki, Jun Sese, and Chikara Meno. Chip-atlas: a data-mining suite powered by full integration of public chip-seq data. *EMBO reports*, 19(12), 2018.
- [101] D. Marbach, J. C. Costello, R. Kuffner, N. M. Vega, R. J. Prill, D. M. Camacho, K. R. Allison, M. Kellis, J. J. Collins, G. Stolovitzky, A. Aderhold, K. R. Allison, R. Bonneau, D. M. Camacho, Y. Chen, J. J. Collins, F. Cordero, J. C. Costello, M. Crane, F. Dondelinger, M. Drton, R. Esposito, R. Foygel, A. de la Fuente, J. Gertheiss, P. Geurts, A. Greenfield, M. Grzegorzczuk, A. C. Hauray, B. Holmes, T. Hothorn, D. Husmeier, V. A. Huynh-Thu, A. Irrthum, M. Kellis, G. Karlebach, R. Kuffner, S. Lebre, V. De Leo, A. Madar, S. Mani, D. Marbach, F. Mordelet, H. Ostrer, Z. Ouyang, R. Pandya, T. Petri, A. Pinna, C. S. Poultney, R. J. Prill, S. Rezny, H. J. Ruskin, Y. Saeys, R. Shamir, A. Sirbu, M. Song, N. Soranzo, A. Statnikov, G. Stolovitzky, N. Vega, P. Vera-Licona, J. P. Vert, A. Visconti, H. Wang, L. Wehenkel, L. Windhager, Y. Zhang, and R. Zimmer. Wisdom of crowds for robust gene network inference. *Nature Methods*, 9(8):796–804, Jul 2012.
- [102] Ann C. Babbie, Thalia E. Chan, and Michael P.H. Stumpf. Learning regulatory models for cell development from single cell transcriptomic data. *Current Opinion in Systems Biology*, 5:72 – 81, 2017. Synthetic biology: Development and differentiation.
- [103] Mark W E J Fiers, Liesbeth Minnoye, Sara Aibar, Carmen Bravo Gonzalez-Blas, Zeynep Kalender Atak, and Stein Aerts. Mapping gene regulatory networks from single-cell omics data. *Briefings in Functional Genomics*, 17(4):246–254, 01 2018.
- [104] H. Todorov, R. Cannoodt, W. Saelens, and Y. Saeys. Network Inference from Single-Cell Transcriptomic Data. *Methods in Molecular Biology*, 1883:235–249, 2019.
- [105] G. La Manno, R. Soldatov, A. Zeisel, E. Braun, H. Hochgerner, V. Petukhov, K. Lidschreiber, M. E. Kastriiti, P. Lonnerberg, A. Furlan, J. Fan, L. E. Borm, Z. Liu, D. van Bruggen, J. Guo, X. He, R. Barker, E. Sundstrom, G. Castelo-Branco, P. Cramer, I. Adameyko, S. Linnarsson, and P. V. Kharchenko. RNA velocity of single cells. *Nature*, 560(7719):494–498, 08 2018.

- [106] V. Svensson, R. Vento-Tormo, and S. A. Teichmann. Exponential scaling of single-cell RNA-seq in the past decade. *Nat Protoc*, 13(4):599–604, 04 2018.
- [107] A. de la Fuente, N. Bing, I. Hoeschele, and P. Mendes. Discovery of meaningful associations in genomic data using partial correlation coefficients. *Bioinformatics*, 20(18):3565–74, 2004.
- [108] K. Basso, A.A. Margolin, G. Stolovitzky, U. Klein, R. Dalla-Favera, and A. Califano. Reverse engineering of regulatory networks in human B cells. *Nat Genet*, 37(4):382–90, 2005.
- [109] J. J. Faith, B. Hayete, J. T. Thaden, I. Mogno, J. Wierzbowski, G. Cottarel, S. Kasif, J. J. Collins, and T. S. Gardner. Large-scale mapping and validation of *Escherichia coli* transcriptional regulation from a compendium of expression profiles. *PLoS Biology*, 5(1):e8, Jan 2007.
- [110] T. Stuart, A. Butler, P. Hoffman, C. Hafemeister, E. Papalexi, W. M. Mauck, Y. Hao, M. Stoeckius, P. Smibert, and R. Satija. Comprehensive Integration of Single-Cell Data. *Cell*, 177(7):1888–1902, Jun 2019.
- [111] E. P. Mimitou, A. Cheng, A. Montalbano, S. Hao, M. Stoeckius, M. Legut, T. Roush, A. Herrera, E. Papalexi, Z. Ouyang, R. Satija, N. E. Sanjana, S. B. Koralov, and P. Smibert. Multiplexed detection of proteins, transcriptomes, clonotypes and CRISPR perturbations in single cells. *Nat. Methods*, 16(5):409–412, 05 2019.
- [112] Ye Yuan and Ziv Bar-Joseph. Deep learning for inferring gene relationships from single-cell expression data. *bioRxiv*, 2019.
- [113] D. Marbach, T. Schaffter, C. Mattiussi, and D. Floreano. Generating realistic in silico gene networks for performance assessment of reverse engineering methods. *Journal of Computational Biology*, 16(2):229–239, Feb 2009.
- [114] G. K. Ackers, A. D. Johnson, and M. A. Shea. Quantitative model for gene regulation by lambda phage repressor. *Proceedings of the National Academy of Sciences*, 79(4):1129–1133, 1982.
- [115] Gisiro Maruyama. Continuous markov processes and stochastic equations. *Rendiconti del Circolo Matematico di Palermo*, 4(1):48, 1955.
- [116] P. Brennecke, S. Anders, J. K. Kim, A. A. Kolodziejczyk, X. Zhang, V. Proserpio, B. Baying, V. Benes, S. A. Teichmann, J. C. Marioni, and M. G. Heisler. Accounting for technical noise in single-cell RNA-seq experiments. *Nature Methods*, 10(11):1093–1095, Nov 2013.
- [117] L. Garcia-Alonso, C. H. Holland, M. M. Ibrahim, D. Turei, and J. Saez-Rodriguez. Benchmark and integration of resources for the estimation of human transcription factor activities. *Genome Res.*, 2019.

- [118] T. Saito and M. Rehmsmeier. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLoS ONE*, 10(3):e0118432, 2015.
- [119] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.
- [120] F. Mordelet and J. P. Vert. SIRENE: supervised inference of regulatory networks. *Bioinformatics*, 24(16):76–82, Aug 2008.
- [121] Jérôme Ambroise, Annie Robert, Benoit Macq, and Jean-Luc Gala. Transcriptional network inference from functional similarity and expression data: a global supervised approach. *Statistical applications in genetics and molecular biology*, 11(1):1–24, 2012.
- [122] Fei Liu, Shao-Wu Zhang, Wei-Feng Guo, Ze-Gang Wei, and Luonan Chen. Inference of gene regulatory network based on local bayesian networks. *PLoS computational biology*, 12(8), 2016.
- [123] Alireza F Siahpirani and Sushmita Roy. A prior-based integrative framework for functional transcriptional regulatory network inference. *Nucleic acids research*, 45(4):e21–e21, 2017.
- [124] Dayanne M Castro, Nicholas R De Veaux, Emily R Miraldi, and Richard Bonneau. Multi-study inference of regulatory networks for more accurate models of gene regulation. *PLoS computational biology*, 15(1):e1006591, 2019.
- [125] Ye Yuan and Ziv Bar-Joseph. Deep learning for inferring gene relationships from single-cell expression data. *Proceedings of the National Academy of Sciences*, 116(52):27151–27158, 2019.
- [126] Tobias Petri, Stefan Altmann, Ludwig Geistlinger, Ralf Zimmer, and Robert Küffner. Addressing false discoveries in network inference. *Bioinformatics*, 31(17):2836–2843, 2015.
- [127] Luigi Cerulo, Charles Elkan, and Michele Ceccarelli. Learning gene regulatory networks from only positive and unlabeled data. *BMC Bioinformatics*, 11(1):228, 2010.
- [128] Jean-Philippe Vert. Reconstruction of biological networks by supervised machine learning approaches. *Elements of computational systems biology*, pages 165–188, 2010.
- [129] Ying Ni, Delasa Aghamirzaie, Haitham Elmarakeby, Eva Collakova, Song Li, Ruth Grene, and Lenwood S Heath. A machine learning approach to predict gene regulatory networks in seed development in arabidopsis. *Frontiers in plant science*, 7:1936, 2016.
- [130] S. R. Maetschke, P. B. Madhamshettiwar, M. J. Davis, and M. A. Ragan. Supervised, semi-supervised and unsupervised inference of gene regulatory networks. *Brief. Bioinformatics*, 15(2):195–211, Mar 2014.

- [131] Yoshihiro Yamanishi, J-P Vert, and Minoru Kanehisa. Protein network inference from multiple genomic data: a supervised approach. *Bioinformatics*, 20(suppl_1):i363–i370, 2004.
- [132] Cuong C To and Jiri Vohradsky. Supervised inference of gene-regulatory networks. *BMC bioinformatics*, 9(1):2, 2008.
- [133] Z. Gillani, M. S. Akash, M. D. Rahaman, and M. Chen. CompareSVM: supervised, Support Vector Machine (SVM) inference of gene regularity networks. *BMC Bioinformatics*, 15:395, Nov 2014.
- [134] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *CoRR*, abs/1312.6203, 2013.
- [135] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional networks on graphs for learning molecular fingerprints. *ArXiv*, abs/1509.09292, 2015.
- [136] Thomas Kipf and Max Welling. Variational graph auto-encoders. *ArXiv*, abs/1611.07308, 2016.
- [137] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.
- [138] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018.
- [139] Guillaume Salha, Stratis Limnios, Romain Hennequin, Viet Anh Tran, and Michalis Vazirgiannis. Gravity-inspired graph autoencoders for directed link prediction. In *CIKM '19*, 2019.
- [140] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. Factorizing yago: scalable machine learning for linked data. In *Proceedings of the 21st international conference on World Wide Web*, pages 271–280, 2012.
- [141] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014.
- [142] Aditya Pratapa, Amogh P. Jalihal, Jeffrey N. Law, Aditya Bharadwaj, and T. M. Murali. Benchmarking algorithms for gene regulatory network inference from single-cell transcriptomic data. *Nature Methods*, 17(2):147–154, 2020.
- [143] Brian Hie, Bryan Bryson, and Bonnie Berger. Efficient integration of heterogeneous single-cell transcriptomes using scanorama. *Nature biotechnology*, 37(6):685–691, 2019.

- [144] Amir Alavi, Matthew Ruffalo, Aiyappa Parvangada, Zhilin Huang, and Ziv Bar-Joseph. A web server for comparative analysis of single-cell rna-seq data. *Nature communications*, 9(1):1–11, 2018.
- [145] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.
- [146] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [147] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.
- [148] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.
- [149] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.
- [150] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.
- [151] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.
- [152] Chenxu Zhu, Sebastian Preissl, and Bing Ren. Single-cell multimodal omics: the power of many. *Nature Methods*, 17(1):11–14, 2020.
- [153] Jason D Buenrostro, Beijing Wu, Ulrike M Litzénburger, Dave Ruff, Michael L Gonzales, Michael P Snyder, Howard Y Chang, and William J Greenleaf. Single-cell chromatin accessibility reveals principles of regulatory variation. *Nature*, 523(7561):486–490, 2015.
- [154] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. *arXiv preprint arXiv:1802.04687*, 2018.
- [155] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Complexity of Computer Computations. Springer, 1972.

- [156] Vincent D. Blondel, Jean Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008.
- [157] Tetsutaro Hayashi, Haruka Ozaki, Yohei Sasagawa, Mana Umeda, Hiroki Danno, and Itoshi Nikaido. Single-cell full-length total RNA sequencing uncovers dynamics of recursive splicing and enhancer RNAs. *Nature Communications*, 9(1):619, Feb 2018.

Appendices

Appendix A

CrossPlan Proofs

A.1 CrossPlan is NP-complete

In this section, we prove that the decision version of the CROSSPLAN is NP-complete. We first state this version of the problem.

DecideCrossPlan Problem. *Given a genetic cross graph $\mathcal{G} = (M, X, E)$, a set $S \subset M$ of source mutants, a set $T \subset M$ of target mutants, the batch size s , and the number of batches k , do k s -batches $\{W_i, 1 \leq i \leq k\}$ exist with the following properties?*

1. For each $1 \leq i \leq k$, $In(W_i) \subseteq S \cup (\bigcup_{1 \leq j \leq i-1} Out(W_j))$ and
2. $T \subseteq (\bigcup_{1 \leq j \leq k} Out(W_j))$.

The definitions of a batch and its input and output sets of mutants are the same as in Section 2.2.2. In this decision version, we ask if there are k batches that can make all target mutants, i.e., T is a subset of the union of the outputs of all the batches.

We now state the Steiner tree problem, which we will reduce to the decision problem of the CROSSPLAN problem.

Steiner Tree Problem. *Given an undirected, weighted graph $G = (V, F)$, a set $U \subset V$ of terminal nodes, and an integer l , is there a connected subgraph of G with at most l edges that contains all the nodes in U ?*

This problem is one of the 21 problems proved by Karp to be NP-complete [155]. We will use $\langle G, U, l \rangle$ to denote such an instance of the Steiner tree problem.

Our reduction relies on establishing a correspondence between a traversal of a Steiner tree rooted at an arbitrary terminal in U and a sequence of crosses in an appropriately defined genetic cross graph. To effect this reduction, we construct an instance $\langle \mathcal{G}, S, T, s, k \rangle$ of DECIDECROSSPLAN from $\langle G, U, l \rangle$, as described in Algorithm 4. In this algorithm, we use set notation to indicate which genes are deleted in a mutant instead of defining additional notation to name the mutant. We assume that the wild type (corresponding to the empty set) is viable. Figure A.1 illustrates the crosses and the genetic cross graph constructed by Algorithm 4 for a toy example.

Algorithm 4 Algorithm to construct an instance $\langle \mathcal{G}, S, T, s, k \rangle$ of DECIDECROSSPLAN from an instance $\langle G, U, l \rangle$ of Steiner Tree.

1. Create a genetic cross graph $\mathcal{G} = (M, X, E)$ as follows:
 - (a) For each node v in G , create two genes $g(v)$ and $d(v)$.
 - (b) For each node v in G , create two viable mutant nodes $\{g(v)\}$ and $\{g(v), d(v)\}$ and one inviable mutant node $\{d(v)\}$. Add these mutant nodes to the set M .
 - (c) For every undirected edge (u, v) in G , create five inviable mutant nodes corresponding to the gene sets $\{g(u), g(v)\}$, $\{g(u), d(v)\}$, $\{g(v), d(u)\}$, $\{g(u), g(v), d(v)\}$, and $\{g(v), g(u), d(u)\}$ in M .
 - (d) For every undirected edge (u, v) in G , create two cross nodes in the set X as follows:
 - (i) A cross node $x_{u,v}$ with incoming edges in E from the viable mutants $\{g(u)\}$ and $\{g(v), d(v)\}$ and outgoing edges in the edge set E to the eight mutants corresponding to the elements of the power set of $\{g(u), g(v), d(v)\}$.
 - (ii) A cross node $x_{v,u}$ defined like $x_{u,v}$ but with the roles of u and v reversed.
 2. Select an arbitrary node $u^* \in U$. Create the set S of source mutant nodes containing the mutant nodes $\{g(u^*)\}$ and all the mutant nodes of the form $\{g(v), d(v)\}$, where v is a node in G .
 3. Create the set T of target mutant nodes containing all the mutant nodes of the form $\{g(u)\}$, where $u \in U$.
 4. Set the batch size $s = 1$.
 5. Set the number of batches $k = l$.
-

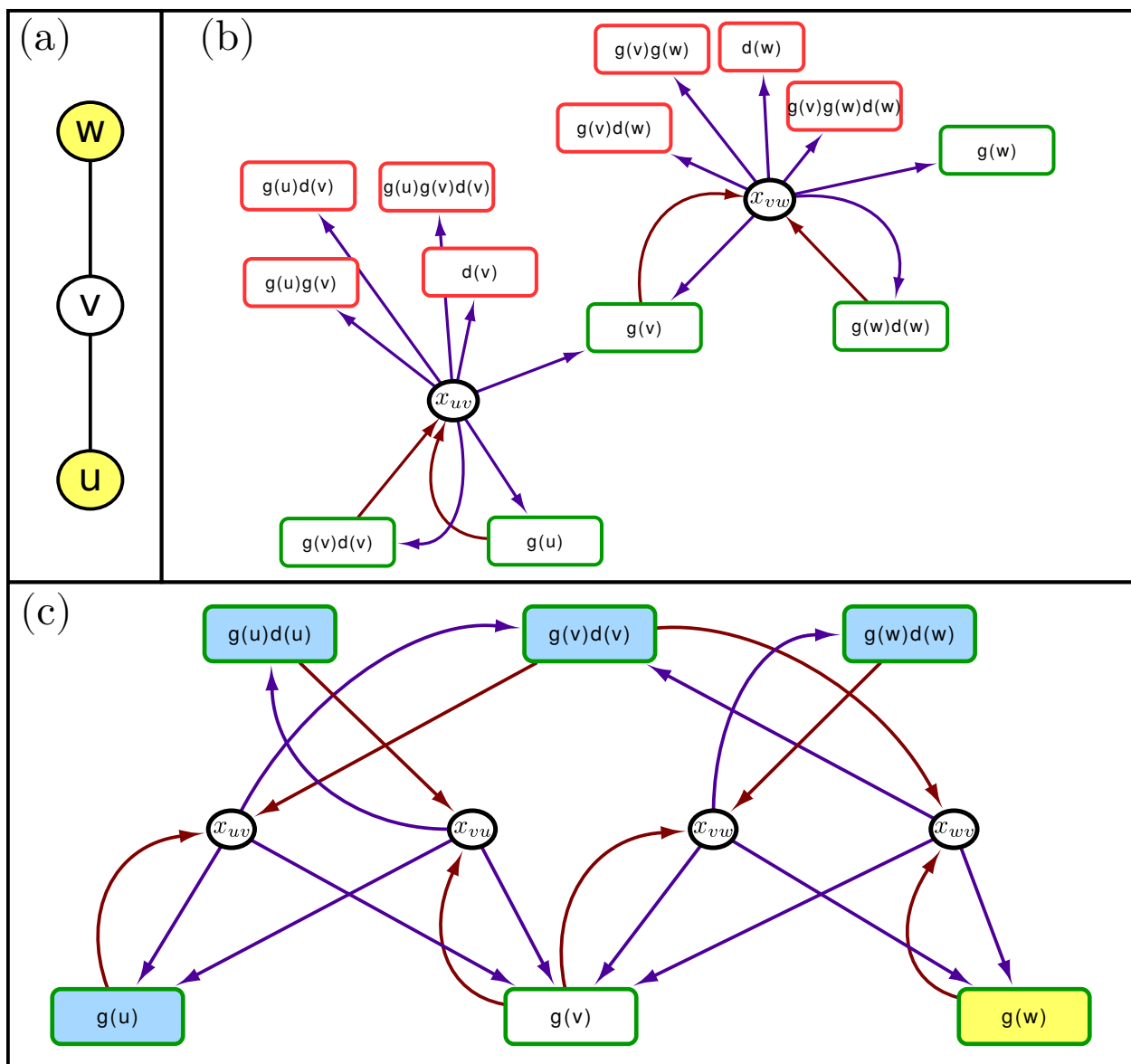


Figure A.1: Illustration of cross nodes constructed by Algorithm 4. (a) Input graph to the Steiner tree problem. Yellow nodes denote terminals in U . (b) The cross nodes x_{uv} and x_{vw} constructed in Step 1d(i) Viable mutant nodes have green borders and inviable mutant nodes have red borders. Brown edges connect input mutant nodes to crosses and purple edges connect crosses to output mutant nodes. (c) The complete genetic cross graph constructed by the algorithm after the deletion of inviable mutant nodes. Blue rectangles correspond to mutant nodes in S and yellow rectangles to mutant nodes in T . In this example, u is the arbitrary node in G selected in Step 2. Note that the mutant node $\{g(u)\}$ is also a member of T but is not shown in yellow since it is a member of S .

We state a series of observations about the instance of DECIDECROSSPLAN constructed by Algorithm 4.

Observation 6. *Every mutant node in M contains three or fewer gene deletions.*

Observation 7. *There are no mutant nodes in M of the form*

1. $\{g(u), d(u), d(v)\}$, where (u, v) is an edge in G .
2. $\{g(u), d(v), d(w)\}$, where u, v, w are nodes in G .
3. $\{g(u), g(v), d(w)\}$, where u, v, w are nodes in G .
4. $\{g(u), g(v), g(w)\}$, where u, v, w are nodes in G .
5. $\{d(u), d(v), d(w)\}$, where u, v, w are nodes in G .

Observation 8. *A mutant node of the form $\{g(u), g(v), d(v)\}$ is present in M iff (u, v) is an edge in G .*

Observation 9. *If $x_{u,v}$ is a cross node in X , then*

1. *both its input mutant nodes $\{g(u)\}$ and $\{g(v), d(v)\}$ are viable and*
2. *apart from $\{g(u)\}$ and $\{g(v), d(v)\}$ (which are also inputs to $x_{u,v}$) and the wild-type strain, the only other viable output mutant node of this cross is $\{g(v)\}$.*

Observation 10. *All mutant nodes in S and T are viable.*

Since no inviable mutant nodes are in T and inviable mutant nodes are not input into any cross nodes, we delete all inviable mutant nodes from \mathcal{G} (Figure A.1(c)). We can show that this modification will not affect any of the arguments we make below. However, the proof is tedious and we leave it to the reader. We can also state the following lemma.

Lemma 11. *The size of \mathcal{G} is linear in the size of G . Algorithm 4 constructs the instance of DECIDECROSSPLAN in time proportional to the size of G .*

We now prove a series of lemmas that show that if the instance of DECIDECROSSPLAN created by Algorithm 4 has a solution, then we can construct a solution to the corresponding Steiner tree problem. Recall that u^* is the arbitrary node in U selected in Step 2 of the algorithm.

Lemma 12. *If $\langle \mathcal{G}, S, T, s, k \rangle$ is a “Yes” instance of DECIDECROSSPLAN, then there is a sequence $\langle x_1, x_2, \dots, x_{k-1}, x_k \rangle$ of k cross nodes and a sequence $\langle v_1 = u^*, v_2, \dots, v_k, v_{k+1} \rangle$ of $k + 1$ mutant nodes in \mathcal{G} such that for every $i, 1 \leq i \leq k$,*

1. the inputs to cross node x_i are the mutant nodes $\{g(v_i)\}$ and $\{(g(v_{i+1}), d(v_{i+1}))\}$
2. the output of x_i is the mutant node $\{g(v_{i+1})\}$, and
3. (v_i, v_{i+1}) is an edge in G .

Proof. Since $s = 1$ (Step 4 in Algorithm 4), each batch in any solution to this instance of DECIDECROSSPLAN contains one cross node. Let x_1, x_2, \dots, x_k denote these cross nodes in batch order. By construction of \mathcal{G} (Step 1d in Algorithm 4), each of these cross nodes corresponds to an edge of G . Therefore, we can rename the mutants involved in these cross nodes as follows: for each $1 \leq i \leq k$, the inputs to x_i are the mutant nodes $\{g(v_i)\}$ and $\{g(v_{i+1}), d(v_{i+1})\}$ and the output of x_i is the mutant node $\{g(v_{i+1})\}$. By construction, this cross node corresponds to the edge (v_i, v_{i+1}) in G .

The first cross node x_1 must have both input mutants in S . Only cross nodes that correspond to edges in G that are incident on u^* have this property. Hence, the node v_1 must be u^* , proving the lemma.

Note that it is possible that the sequence $\langle v_1 = u^*, v_2, \dots, v_k, v_{k+1} \rangle$ does not contain $k + 1$ distinct nodes, e.g., when there is an edge in G such that both crosses corresponding to it are elements of $\langle x_1, x_2, \dots, x_{k-1}, x_k \rangle$ or when there is an index $j, 1 < j \leq k$, such that the output of x_j is identical to the input of an earlier cross. \square

Let $\langle \mathcal{G}, S, T, s, k \rangle$ be a “Yes” instance of DECIDECROSSPLAN. For each $1 \leq i \leq k$, we use G_i to denote the graph induced by the set of edges $\{(v_j, v_{j+1}), 1 \leq j \leq i\}$, where the edges are as defined in Lemma 12. The next two lemmas prove properties of these graphs.

Lemma 13. *If $\langle \mathcal{G}, S, T, s, k \rangle$ is a “Yes” instance of DECIDECROSSPLAN, then for each $i, 1 \leq i \leq k$, the graph G_i is a connected subgraph of G containing u^* .*

Proof. Using the notation of Lemma 12, let the solution to this instance of DECIDECROSSPLAN be the sequence of k cross nodes $\langle x_1, x_2, \dots, x_{k-1}, x_k \rangle$. We prove the lemma by induction on i . The first cross node x_1 has $u^* = v_1$ as an input mutant node. Moreover G_1 contains one edge, proving the base case.

Assume that the lemma is true for the index $i > 1$, i.e., G_i is connected and contains u^* . We prove the statement for the index $i + 1$. Consider the cross node x_{i+1} , which has the mutant nodes $\{g(v_i)\}$ and $\{g(v_{i+1}), d(v_{i+1})\}$ as inputs. Since the k crosses form a solution to this instance of DECIDECROSSPLAN, each input must either be a member of S or the output of one of the crosses x_1, x_2, \dots, x_i . Of the two inputs to x_{i+1} , only $\{g(v_{i+1}), d(v_{i+1})\}$ is an element of S , by construction (Step 2 of Algorithm 4). Hence, the mutant node $\{g(v_i)\}$ must be an output of a cross x_l , where $l \leq i$. Since G_i is connected, it must contain v_i . Therefore, G_{i+1} , which is the graph formed by the addition of the edge (v_i, v_{i+1}) to G_i , is also connected and contains u^* . Note that (v_i, v_{i+1}) may already be an edge in G_i or that the addition of this edge to G_i may create a cycle. \square

Lemma 14. *If $\langle \mathcal{G}, S, T, s, k \rangle$ is a “Yes” instance of DECIDECROSSPLAN, then the graph G_k is connected and contains all the terminals in U .*

Proof. Lemma 13 proves that G_k is connected and contains u^* . Since the crosses x_1, x_2, \dots, x_k correspond to a solution to the DECIDECROSSPLAN problem, the outputs of these cross nodes must contain all the target mutants of the form $\{g_u\}$, where $u \in U - \{u^*\}$. In other words, for every terminal $u \in U - \{u^*\}$, there is at least one cross node $x_i, 1 \leq i \leq k$, such that the mutant node $\{g(u)\}$ is the output of x_i . Therefore, u is identical to v_{i+1} , i.e., u is a node in G_k , which proves the lemma. \square

This lemma has the following corollary.

Corollary 15. *If $\langle \mathcal{G}, S, T, s, k \rangle$ is a “Yes” instance of DECIDECROSSPLAN, then $\langle G, U, k \rangle$ is a “Yes” instance of Steiner tree, i.e., G_k contains a connected subgraph with at most k edges that contains all the nodes in U .*

Next, we consider how to construct a solution to the DECIDECROSSPLAN problem from a “Yes” instance of Steiner tree.

Lemma 16. *If G contains a Steiner Tree with at most k edges that connects all the terminals in U , then the instance $\langle \mathcal{G}, S, T, s, k \rangle$ computed by Algorithm 4 is a “Yes” instance of DECIDECROSSPLAN problem. Specifically, there are k cross nodes $\langle x_1, x_2, \dots, x_k \rangle$ in \mathcal{G} such that*

1. For each $1 \leq i \leq k$, $In(x_i) \subseteq S \cup (\bigcup_{1 \leq j \leq i-1} Out(x_j))$ and
2. $T \subseteq (\bigcup_{1 \leq j \leq k} Out(x_j))$.

Proof. Let τ denote the Steiner tree in G with at most k edges that connects all the terminals in U . Consider a breadth-first traversal of τ starting at u^* . We rename the nodes in S as $v_1 = u^*, v_2, \dots, v_{l+1}$ in the order of this traversal. We name the edges as follows: if an edge in S connects a node v_i to a node v_j , where $i < j$, we call this edge e_{j-1} . Note that edge indices lie between 1 and k and that the index of an edge is one less than the larger node index incident on that edge. We define a set of k cross nodes in \mathcal{G} as follows: for each $1 \leq i \leq k$, let the edge e_i connect a vertex v_j (for some $j \leq i$) to the vertex v_{i+1} . Then the cross node x_i has the viable mutant nodes $\{g(v_j)\}$ and $\{g(v_{i+1}), d(v_{i+1})\}$ as inputs and the viable mutant node $\{g(v_{i+1})\}$ as output. Note that since e_i is an edge in G , this cross node and its input and output mutant nodes are members of \mathcal{G} by construction (Steps 1b and 1d of Algorithm 4). We claim that this set of k cross nodes is a solution to the instance of DECIDECROSSPLAN computed by Algorithm 4.

First, we prove by induction on i that for each $1 \leq i \leq k$, $In(x_i) \subseteq S \cup (\bigcup_{1 \leq j \leq i-1} Out(x_j))$. For $i = 1$, the inputs to x_1 are the mutant nodes $\{g(u^*)\}$ and $\{g(v_2), d(v_2)\}$, both of which are

members of S by construction. Suppose the statement is true for index i . Consider the cross node x_{i+1} . Its inputs are the mutant nodes $\{g(v_j)\}$, for some $j \leq i$ and $\{g(v_{i+1}), d(v_{i+1})\}$. The second input is a member of S by construction. Since $j \leq i$ and $\{g(v_j)\}$ is the output of the cross x_j , we have shown that the statement holds true for index $i + 1$ as well.

Now we show that $T \subseteq (\bigcup_{1 \leq j \leq k} \text{Out}(W_j))$. Note that the right-hand side of this expression is simply the set of mutant nodes $\{\{g(v_j)\}, 1 \leq j \leq k\}$. Our construction of the crosses x_1, x_2, \dots, x_k from the Steiner tree τ implies that the set of nodes $\{v_k, 1 \leq j \leq k\}$ contain all the terminals in U . Therefore, T , which is the set of mutant nodes $\{\{g(u)\}, u \in U\}$ must be contained in $\{\{g(v_j)\}, 1 \leq j \leq k\}$. \square

We are now ready to prove the main result of this section.

Theorem 17. *The DECIDECROSSPLAN problem is NP-complete.*

Proof. We first prove that the DECIDECROSSPLAN problem is in NP. Given a set of k batches $\{W_1, W_2, \dots, W_k\}$, we perform the following checks for each $1 \leq i \leq k$. To perform these steps efficiently, we maintain a running set Ω of output mutant nodes, which we initialize with the mutants in S :

- (i) We verify that every cross node in batch W_i is in \mathcal{G} and the size of W_i is at most s .
- (ii) For every cross $x \in W_i$, for each of the two mutant nodes that is an input to x , we verify that the mutant node is an element of Ω .
- (iii) For every cross $x \in W_i$, we add all the output mutant nodes of x to Ω .
- (iv) If $i = k$, we check whether T is a subset of Ω .

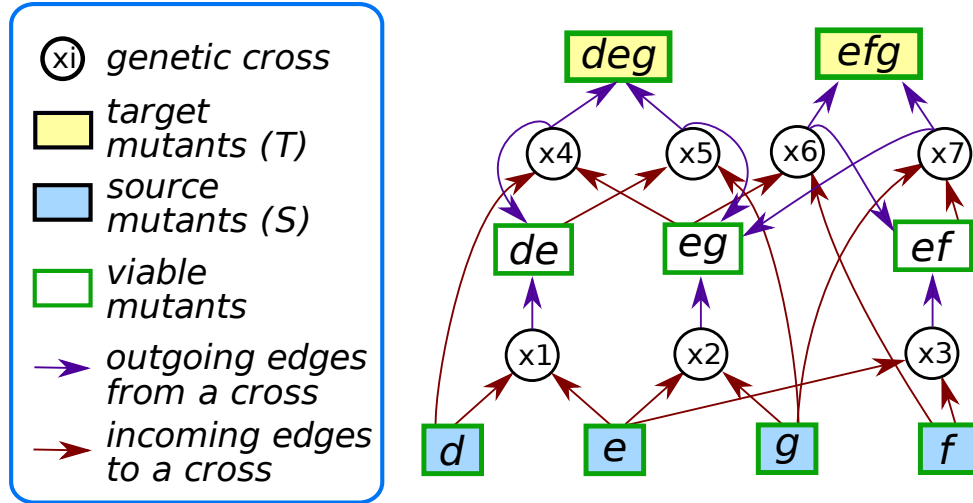
The running time of this algorithm is polynomial in k, s , and the size of \mathcal{G} , proving that the DECIDECROSSPLAN problem is in NP.

Consider the instance of DECIDECROSSPLAN. Corollary 15 and Lemma 16 prove that the instance $\langle \mathcal{G}, S, T, s, k \rangle$ of DECIDECROSSPLAN created by Algorithm 4 is a “Yes” instance of this problem iff $\langle G, U, l \rangle$ is a “Yes” instance of Steiner Tree. Since Steiner Tree is NP-complete, we have proven that DECIDECROSSPLAN is also NP-complete. \square

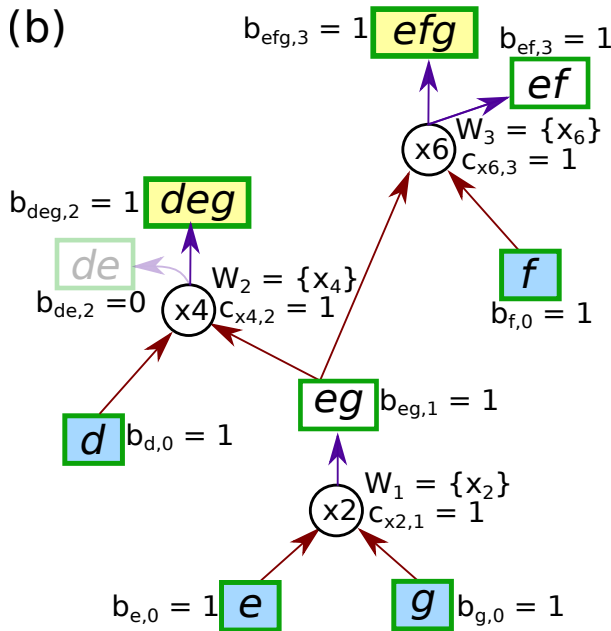
A.2 Proof of correctness for the CrossPlan ILP

We seek to prove that an assignment of binary variables that is feasible, i.e., an assignment that satisfies constraints (2.1)–(2.5) and maximizes the objective function (2.6) is an optimal solution to the CROSSPLAN problem. We first consider the genetic cross graph corresponding to any feasible solution to this ILP and prove some of its properties. Let \mathcal{O} be an assignment

(a)



(b)



(c)

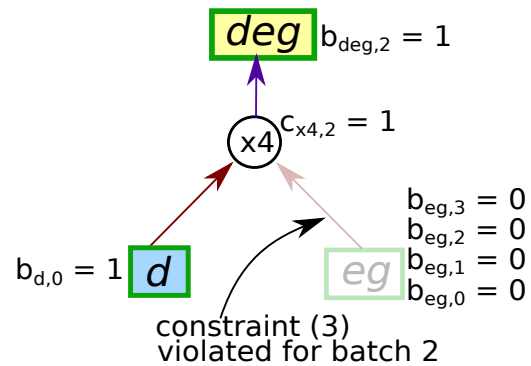


Figure A.2: An illustration of a genetic cross graph and the output genetic cross graphs for a feasible solution (for $s = 1, k = 3$). (a) An input to the CROSSPLAN problem with four mutant nodes in S and two mutant nodes in T . We do not show inviable mutants. (b) The output genetic cross graph for a feasible solution where we perform cross x_2 in batch 1, x_4 in batch 2, and x_6 in batch 3. For the mutant node de , the variable $b_{de,2} = 0$ in this feasible solution. Hence, this mutant node (displayed in a lighter color) is not an output of cross x_4 in the genetic cross graph for this feasible solution. (c) Illustration of the proof of Lemma 18. If the mutant node eg is not in the output genetic cross graph, then constraint (3) is violated for the cross node x_4 and batch 2.

of the binary variables in the ILP in a feasible solution. We define the *output genetic cross graph* $\mathcal{G}(\mathcal{O}) = (M(\mathcal{O}), X(\mathcal{O}), E(\mathcal{O}))$ as follows (Figure A.2(a),(b)):

1. For every mutant node m in $M(\mathcal{O})$, $b_{m,i} = 1$ for at least one value of i , $0 \leq i \leq k$ in \mathcal{O} ,
2. For every cross node x in $X(\mathcal{O})$, $c_{x,i} = 1$ in \mathcal{O} for at least one value of i , $1 \leq i \leq k$, and
3. $E(\mathcal{O})$ is the set of edges in the genetic cross graph \mathcal{G} that are induced by the nodes in $M(\mathcal{O})$ and $X(\mathcal{O})$.

We need two more definitions. For every mutant $m \in M(\mathcal{O})$, we define $\beta(m)$ as the smallest batch index in which mutant m is made, i.e.,

$$\beta(m) = \arg \min_i \{b_{m,i} = 1, 0 \leq i \leq k\}.$$

For every cross $x \in X(\mathcal{O})$, we define $\beta(x)$ as the smallest batch index in which the cross is made, i.e.,

$$\beta(x) = \arg \min_i \{c_{x,i} = 1, 1 \leq i \leq k\}.$$

Note that these values are well-defined for every mutant node and cross node in $\mathcal{G}(\mathcal{O})$ since we include a mutant node m in $M(\mathcal{O})$ (respectively, cross node x in $X(\mathcal{O})$) if and only if $b_{m,i} = 1$ (respectively, $c_{x,i} = 1$) for at least one value of i . We can now state and prove the following lemma:

Lemma 18. *If \mathcal{O} is a feasible solution to the ILP, then the output genetic cross graph $\mathcal{G}_{\mathcal{O}} = (M(\mathcal{O}), X(\mathcal{O}), E(\mathcal{O}))$ has the following properties:*

1. *Every cross node $x \in X(\mathcal{O})$ has in-degree equal to two.*
2. *If (m, x) is an edge in $E(\mathcal{O})$ from a mutant node m to a cross node x , then $\beta(m) < \beta(x)$.*

Proof. For every cross node $x \in X(\mathcal{O})$, the definition of $X(\mathcal{O})$ implies that $\beta(x)$ is defined; otherwise, we would not have included x in $X(\mathcal{O})$. Therefore, $c_{x,\beta(x)} = 1$. Similarly, if there is a mutant node m that is present in M but not in $M(\mathcal{O})$, then $b_{m,i} = 0$ for every $0 \leq i \leq k$; otherwise, we would have included m in $M(\mathcal{O})$.

1. Recall that $E(\mathcal{O})$ is the set of edges induced in \mathcal{G} by the mutant nodes in $M(\mathcal{O})$ and the cross nodes in $X(\mathcal{O})$. Therefore, if there is a cross node $x \in X(\mathcal{O})$ whose in-degree is either zero or one, then at least one of the mutant nodes that are input to x in \mathcal{G} is not present in $M(\mathcal{O})$. Let m be such a node. Note that (m, x) is an edge in \mathcal{G} . Since \mathcal{O} is a feasible solution, constraint (2.3) must apply to this edge for $i = \beta(x)$, i.e., $c_{x,\beta(x)} \leq \sum_{j=0}^{\beta(x)-1} b_{m,j} = 0$, which contradicts the fact that $c_{x,\beta(x)} = 1$ by dint of the inclusion of x in $\mathcal{G}(\mathcal{O})$ (Figure A.2(c)). Therefore, every cross node in $\mathcal{G}(\mathcal{O})$ must have in-degree equal to two.

2. Now suppose that $\beta(m) \geq \beta(x)$ for some $(m, x) \in E(\mathcal{O})$. Since constraint (2.3) must apply to this edge for $i = \beta(x)$, we have $1 = c_{x, \beta(x)} \leq \sum_{j=0}^{\beta(x)-1} b_{m, j}$. Therefore, $b_{m, j} = 1$ for at least one value of $0 \leq j \leq \beta(x) - 1$, which implies that $\beta(m) \leq \beta(x) - 1$, leading to a contradiction. □

Now we define the batches corresponding to a feasible solution \mathcal{O} and prove that each batch contains at most s crosses and that the batches satisfy the rules on input and output mutants in the statement of the CROSSPLAN problem. Given a feasible solution \mathcal{O} to the ILP, for every $1 \leq i \leq k$, we define the *batch* $W_i(\mathcal{O})$ to be the set of crosses with $\beta(x) = i$, i.e., $W_i(\mathcal{O}) = \{x | c_{x, i} = 1, x \in X(\mathcal{O})\}$. Note that a cross may appear in multiple batches, by this definition. We can prove the following lemma.

Lemma 19. *For every $1 \leq i \leq k$, $W_i(\mathcal{O})$ contains at most s crosses.*

Proof. Consider an arbitrary value of i between 1 and k . Since \mathcal{O} is a feasible solution to the ILP, it must satisfy constraint (2.2), i.e.,

$$\sum_{x \in X} c_{x, i} \leq s,$$

Therefore, the number of distinct cross nodes x such that $c_{x, i} = 1$ is at most s , which implies that the size of $W_i(\mathcal{O})$ is also at most s . □

With these lemmas in hand, we are almost ready to prove that the k s -batches $\{W_i(\mathcal{O}), 1 \leq i \leq k\}$ satisfy the rule on inputs and outputs in the statement of the CROSSPLAN problem. However, in Section 2.2.2, we defined the inputs and outputs to a batch with respect to the input genetic cross graph \mathcal{G} whereas we have defined batches for the feasible solution \mathcal{O} with respect to the output genetic cross graph $\mathcal{G}(\mathcal{O})$, which is a subgraph of \mathcal{G} . To account for this difference, we first expand the notation of inputs and outputs to a batch: given a batch W , we will use $In(W, \mathcal{G})$ to represent the mutant nodes in \mathcal{G} that are inputs to the cross nodes in W and $In(W, \mathcal{G}(\mathcal{O}))$ when we are referring to the mutant nodes in $\mathcal{G}(\mathcal{O})$. We use $Out(W, \mathcal{G})$ and $Out(W, \mathcal{G}(\mathcal{O}))$ analogously. We now state and prove some lemmas that relate the input and output sets of a batch in \mathcal{G} and in $\mathcal{G}(\mathcal{O})$.

Lemma 20. *A mutant node m is in $\mathcal{G}(\mathcal{O})$ if and only if it is a member of S or is a member of the output set of some batch, i.e., $m \in S$ or $m \in Out(W_i(\mathcal{O}), \mathcal{G}(\mathcal{O}))$ for some $i, 1 \leq i \leq k$.*

Proof. We first prove the “only if” direction. Consider a mutant node m that is in $\mathcal{G}(\mathcal{O})$. If $m \in S$, then this statement is proved in this direction. Therefore, we assume that $m \notin S$. Since m is in $\mathcal{G}(\mathcal{O})$, there must some value of $i, 1 \leq i \leq k$ such that $b_{m, i} = 1$. This variable must satisfy constraint (2.4), i.e.,

$$1 = b_{m, i} \leq \sum_{\substack{x \in X \\ (x, m) \in E}} c_{x, i}$$

Therefore, there must be a cross node x such that (x, m) is in \mathcal{G} and $c_{x,i} = 1$. By construction, both x and (x, m) are in $\mathcal{G}(\mathcal{O})$ and x is an element of batch $W_i(\mathcal{O})$. Therefore, m is a member of $Out(W_i(\mathcal{O}), \mathcal{G}(\mathcal{O}))$.

We now prove the “if” direction. There are two cases.

1. Case 1: Suppose m is in S . Constraint (2.1) sets $b_{m,0} = 1$. Therefore m is a node in $\mathcal{G}(\mathcal{O})$.
2. Case 2: Suppose m is an element of $Out(W_i(\mathcal{O}), \mathcal{G}(\mathcal{O}))$ for some $i, 1 \leq i \leq k$. There must be a cross node x such that x and the edge (x, m) are in $\mathcal{G}(\mathcal{O})$; otherwise, m cannot be a member of $Out(W_i(\mathcal{O}), \mathcal{G}(\mathcal{O}))$. Therefore m must also be a node in $\mathcal{G}(\mathcal{O})$.

□

Lemma 21. *Given a feasible solution \mathcal{O} to the CROSSPLAN ILP, for each $i, 1 \leq i \leq k$, the s -batch $W_i(\mathcal{O})$ satisfies the following properties (Figure A.3):*

1. $In(W_i(\mathcal{O}), \mathcal{G}(\mathcal{O})) = In(W_i(\mathcal{O}), \mathcal{G})$, i.e., the input mutants for $W_i(\mathcal{O})$ are identical in $\mathcal{G}(\mathcal{O})$ and in \mathcal{G} .
2. $Out(W_i(\mathcal{O}), \mathcal{G}(\mathcal{O})) \subseteq Out(W_i(\mathcal{O}), \mathcal{G})$, i.e., the output mutants for $W_i(\mathcal{O})$ in $\mathcal{G}(\mathcal{O})$ are a subset of the output mutants in \mathcal{G} .
3. $In(W_i(\mathcal{O}), \mathcal{G}(\mathcal{O})) \subseteq S \cup (\bigcup_{1 \leq j \leq i-1} Out(W_j(\mathcal{O}), \mathcal{G}(\mathcal{O})))$, i.e., the input mutants for $W_i(\mathcal{O})$ in $\mathcal{G}(\mathcal{O})$ are a subset of the union of S and the output mutants of the preceding batches in $\mathcal{G}(\mathcal{O})$.

Proof. We prove the claims in order.

1. Since $\mathcal{G}(\mathcal{O})$ is a subgraph of \mathcal{G} , we know that $In(W_i(\mathcal{O}), \mathcal{G}(\mathcal{O})) \subseteq In(W_i(\mathcal{O}), \mathcal{G})$. We now prove that the two sets are identical. Consider any mutant node m that is a member of $In(W_i(\mathcal{O}), \mathcal{G})$. There must be a cross node x in $W_i(\mathcal{O})$ such that (m, x) is an edge in \mathcal{G} . By the definition of $W_i(\mathcal{O})$, x is a node in $\mathcal{G}(\mathcal{O})$ as well. By Lemma 18, the degree of x in $\mathcal{G}(\mathcal{O})$ is two. Therefore, m must be a member of both $M(\mathcal{O})$ and $In(W_i(\mathcal{O}), \mathcal{G}(\mathcal{O}))$, proving the claim.
2. This statement follows from the fact that $\mathcal{G}(\mathcal{O})$ is a subgraph of \mathcal{G} .
3. Let m be an arbitrary mutant node in $In(W_i(\mathcal{O}), \mathcal{G}(\mathcal{O}))$. By definition, there must be a cross node x in $W_i(\mathcal{O})$ such that (m, x) is an edge in $\mathcal{G}(\mathcal{O})$, i.e., m is an input mutant node to x in $\mathcal{G}(\mathcal{O})$. We know that for every $x \in W_i(\mathcal{O})$, we have $1 \leq \beta(x) \leq i$, and from Lemma 18, $\beta(m) < \beta(x) \leq i$. Consider batch $\beta(m)$. We have that $b_{m,\beta(m)} = 1$. There are two cases to consider.

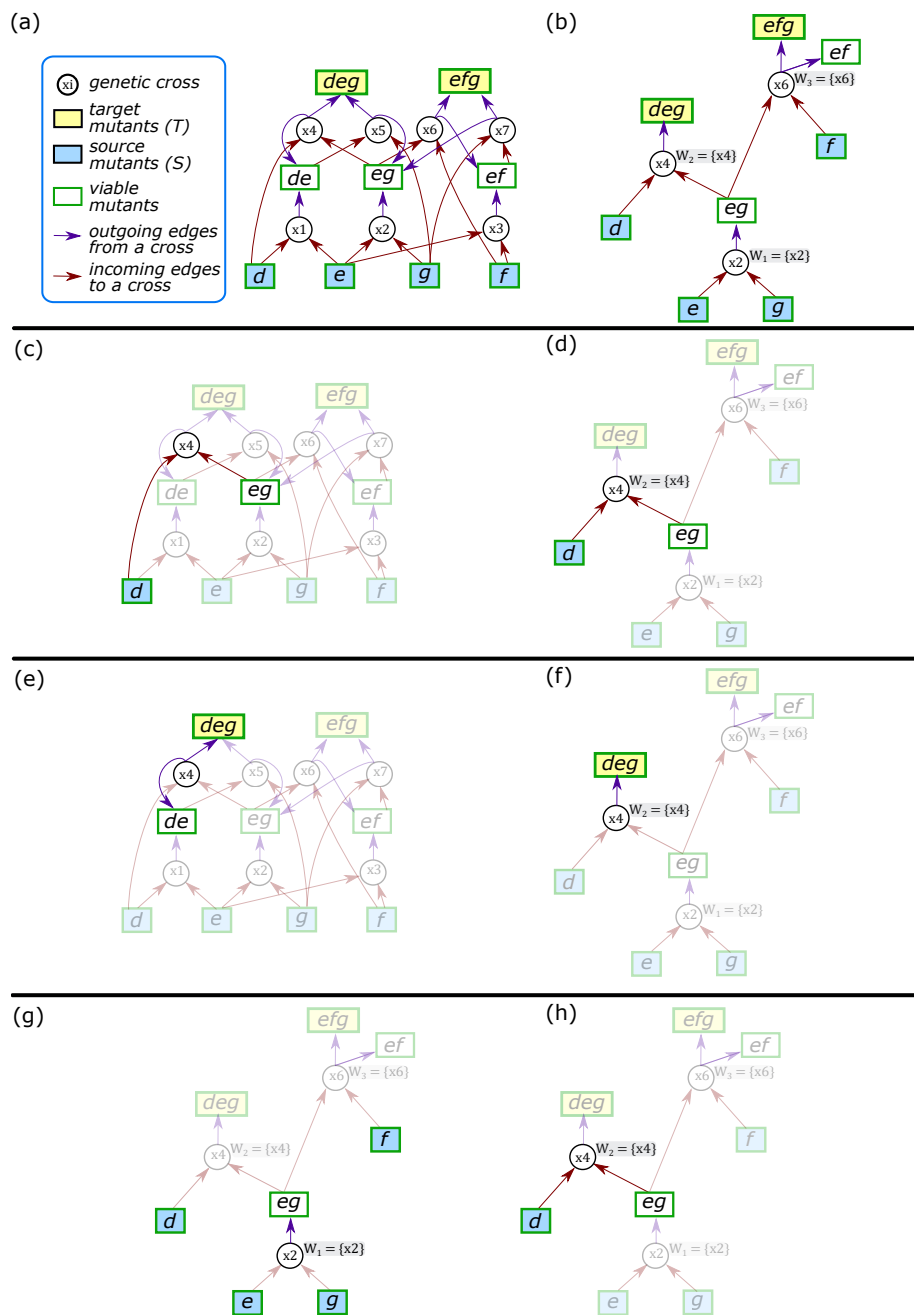


Figure A.3: Illustration of the three statements in Lemma 21. (a) An input genetic cross graph \mathcal{G} to the CROSSPLAN problem (identical to Figure A.2(a)). (b) The output genetic cross graph $\mathcal{G}(\mathcal{O})$ for a feasible solution \mathcal{O} (identical to Figure A.2(b)). Here $s = 1, k = 3$. The remaining figures illustrate the statements in Lemma 21 for batch W_2 , which contains the cross x_4 . (c) The input mutant nodes to W_2 in \mathcal{G} . (d) The input mutant nodes to W_2 in $\mathcal{G}(\mathcal{O})$. (e) The output mutant nodes of W_2 in \mathcal{G} . (f) The output mutant nodes of W_2 in $\mathcal{G}(\mathcal{O})$. (g) The union of S and the outputs of batch W_1 . (h) The inputs to batch W_2 , which are a subset of the mutant nodes in (g).

- (a) $\beta(m) = 0$. In this case, m must be a member of S , which proves this statement in the Lemma.
- (b) $\beta(m) \geq 1$. In this case, we can apply constraint (2.4) for $i = \beta(m)$:

$$1 = b_{m,\beta(m)} \leq \sum_{\substack{x \in X \\ (x,m) \in E}} c_{x,\beta(m)}.$$

Therefore, there must be a cross node y such that m is an output of y and $c_{y,\beta(m)} = 1$, i.e., $y \in W_{\beta(m)}(\mathcal{O})$. Since $b_{m,\beta(m)} = 1$, $(y, m) \in \mathcal{G}(\mathcal{O})$, and $y \in W_{\beta(m)}(\mathcal{O})$, we have $m \in \text{Out}(W_{\beta(m)}(\mathcal{O}))$.

Since $0 \leq \beta(m) < i$ and we chose an arbitrary mutant node m in $\text{In}(W_i(\mathcal{O}), \mathcal{G}(\mathcal{O}))$, we have proven that $\text{In}(W_i(\mathcal{O}), \mathcal{G}(\mathcal{O})) \subseteq S \cup (\bigcup_{1 \leq j \leq i-1} \text{Out}(W_j(\mathcal{O}), \mathcal{G}(\mathcal{O})))$.

□

The following corollary is a consequence of Lemma 21. Note that we state this corollary in terms of \mathcal{G} .

Corollary 22. *Given a feasible solution \mathcal{O} to the CROSSPLAN ILP, for each $1 \leq i \leq k$, the s -batch $W_i(\mathcal{O})$ satisfies $\text{In}(W_i(\mathcal{O}), \mathcal{G}) \subseteq S \cup (\bigcup_{1 \leq j \leq i-1} \text{Out}(W_j(\mathcal{O}), \mathcal{G}))$.*

We have now demonstrated that the s -batches defined by an feasible solution satisfy the first condition in the definition of the CROSSPLAN problem. We now state a lemma relating the expression in the objective function to the number of target mutants made by a feasible solution.

Lemma 23. *Given a feasible solution \mathcal{O} to the CROSSPLAN ILP, the value of the objective function (Equation (2.6)) is at most the number of target mutants that are also members of $\bigcup_{1 \leq j \leq k} \text{Out}(W_j(\mathcal{O}), \mathcal{G}(\mathcal{O}))$.*

Proof. Recall that the variable a_m is defined only if m is a target mutant. By constraint (Equation (2.5)), if $a_m = 1$ then $\sum_{0 \leq i \leq k} b_{m,i} \geq 1$, i.e., $b_{m,i} = 1$ for at least one value of i , $0 \leq i \leq k$. Therefore, by the proof of Lemma 20, m is a member of $\text{Out}(W_i(\mathcal{O}), \mathcal{G}(\mathcal{O}))$. However, it is possible that there is a target mutant m such that $b_{m,i} = 1$ for at least value of $0 \leq i \leq k$ but $a_m = 0$; this setting of variable values does not violate constraint (2.5). Therefore, the value of the objective function is at most the number of target mutants that are members of $\bigcup_{1 \leq j \leq k} \text{Out}(W_j(\mathcal{O}), \mathcal{G}(\mathcal{O}))$.

□

We now prove that maximizing the objective function (2.6) subject to the constraints (2.1)–(2.5) solves the CROSSPLAN problem to optimality.

Theorem 24. *Let \mathcal{O}^* be a feasible solution to the CROSSPLAN ILP that maximizes the objective function (2.6). Then the k batches $\{W_i(\mathcal{O}^*), 1 \leq i \leq k\}$ satisfy the following properties:*

1. *For each $1 \leq i \leq k$, $|W_i(\mathcal{O}^*)| \leq s$, i.e., $W_i(\mathcal{O}^*)$ is an s -batch.*
2. *For each $1 \leq i \leq k$, $In(W_i(\mathcal{O}^*), \mathcal{G}) \subseteq S \cup (\bigcup_{1 \leq j \leq i-1} Out(W_j(\mathcal{O}^*), \mathcal{G}))$.*
3. *The size of $T \cap (\bigcup_{1 \leq j \leq k} Out(W_j(\mathcal{O}^*), \mathcal{G}))$ is maximized over all possible sets of k s -batches.*

Proof. Since \mathcal{O}^* is a feasible solution, the first two parts in the theorem follow from Lemma 19 and Corollary 22, respectively. We now prove the third statement in the theorem.

The value $\nu(\mathcal{O}^*)$ of the objective function for \mathcal{O}^* satisfies the inequalities

$$\begin{aligned} \nu(\mathcal{O}^*) &\leq T \cap \left(\bigcup_{1 \leq j \leq k} Out(W_j(\mathcal{O}^*), \mathcal{G}(\mathcal{O}^*)) \right), \text{ by Lemma 23} \\ &\leq T \cap \left(\bigcup_{1 \leq j \leq k} Out(W_j(\mathcal{O}^*), \mathcal{G}) \right), \text{ by Lemma 21} \end{aligned}$$

Note that first inequality specifies the output sets of the batches with respect to $\mathcal{G}(\mathcal{O}^*)$ and the second inequality with respect to \mathcal{G} . We now prove that

$$\nu(\mathcal{O}^*) = T \cap \left(\bigcup_{1 \leq j \leq k} Out(W_j(\mathcal{O}^*), \mathcal{G}) \right).$$

If this equality does not hold, then there must be a target mutant m and a batch $1 \leq i \leq k$ and such that m is a member of $Out(W_i(\mathcal{O}^*), \mathcal{G})$ but $a_m = 0$. Since m is a member of $Out(W_i(\mathcal{O}^*), \mathcal{G})$, there must be a cross node x in $W_i(\mathcal{O}^*)$ such that (x, m) is an edge in \mathcal{G} . Clearly, x is in $\mathcal{G}(\mathcal{O}^*)$ (otherwise, it would not be a member of $W_i(\mathcal{O}^*)$). Since x in $W_i(\mathcal{O}^*)$, we have $c_{x,i} = 1$. Note that setting $b_{m,i} = 1$ does not violate constraint (2.4). We can now set $a_m = 1$ without violating constraint (2.5). However, we have constructed a new feasible solution to the ILP with an objective function value equal to $\nu(\mathcal{O}^*) + 1$, which contradicts the fact that \mathcal{O}^* maximizes the objective function. Therefore, such a target mutant m cannot exist, which proves that $\nu(\mathcal{O}^*)$ equals the size of $T \cap (\bigcup_{1 \leq j \leq k} Out(W_j(\mathcal{O}^*), \mathcal{G}))$. Since \mathcal{O}^* maximizes the value of the objective function, we have also proven that the size of $T \cap (\bigcup_{1 \leq j \leq k} Out(W_j(\mathcal{O}^*), \mathcal{G}))$ is maximized over all possible sets of k s -batches. \square

Appendix B

BEELINE Supplementary Results

B.1 Datasets from synthetic networks

B.1.1 Generation of simulated datasets

For each network, using the procedure outlined in “Creating Datasets from synthetic networks” (Methods), we applied BoolODE by sampling parameters ten times and creating 5,000 simulations per parameter set. Then we created two datasets per parameter set, one with 2,000 cells and another with 5,000 cells by sampling one cell per simulation, to obtain 20 different expression datasets. We visually inspected the two-dimensional representation of these datasets computed by t-SNE and confirmed that the simulation of each synthetic network yielded the expected trajectory. Next, we clustered the simulations in each dataset using k -means clustering with k set to the expected number of trajectories (Methods).

We plotted each cluster with a different colour in the two-dimensional t-SNE layouts. We visually compared the corresponding plots to confirm that each cluster indeed included cells from a distinct trajectory. Supplementary Figure 1(b) shows a t-SNE plot for an example 2,000-cell dataset for each network, with all these plots appearing in Figure B.1. Each colour in the example t-SNE plots in Supplementary Figure 1(c) corresponds to a distinct cluster. We found that the t-SNE visualizations succeed in capturing the qualitative steady states of each of the models we simulate. Specifically, the number of steady states we expected from each synthetic network (e.g., one steady state for linear, two for bifurcating, etc.) were visually apparent in the t-SNE plots. These results reassured us that BoolODE was successful in correctly simulating the network models.

B.1.2 Parameter Search

Six of the 12 algorithms we tested had no parameters. For each of the other six methods, we performed a parameter sweep as described below. For the synthetic networks, we optimized the parameters on the datasets from the Linear network, separately for the five sets of 10 datasets containing varying number of cells (100, 200, 500, 2,000 and 5,000). For each method and number of cells, we chose the parameter value or combination of values that maximized the median AUPRC computed over the 10 datasets.

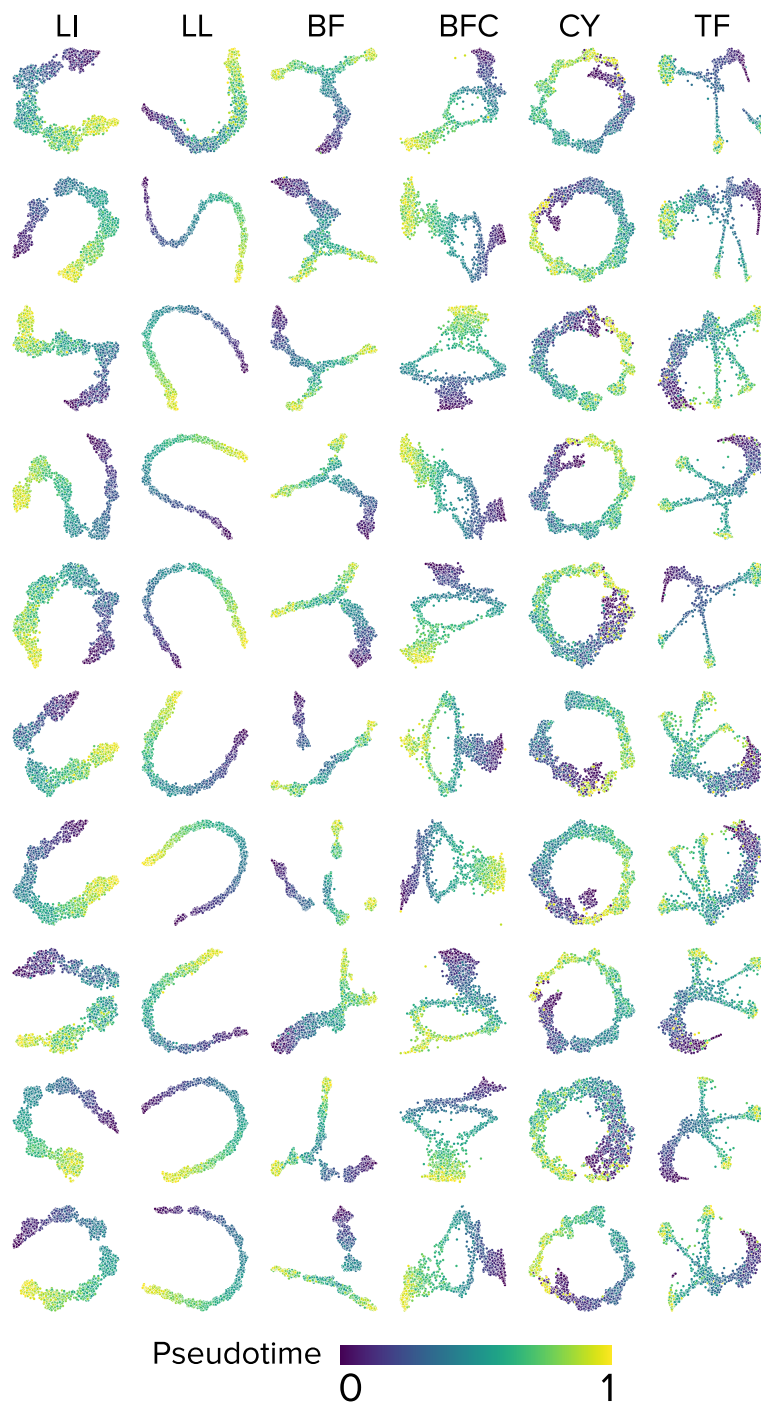


Figure B.1: t-SNE visualizations of simulations of each synthetic network with 10 distinct parameter sets and 2,000 cells. Each column corresponds to one synthetic network, indicated by the abbreviation on the top. Each row corresponds to one of the ten parameter sets. Each visualization is the 2-D t-SNE projection of the simulation output for the corresponding network and parameter set. The colors indicate the simulation time (blue for low, green for intermediate, and yellow for high). Abbreviations: LI: Linear, CY: Cycle, LL: Linear Long, BF: Bifurcating, BFC: Bifurcating Converging, TF: Trifurcating.

Below we describe the parameters tested for each method. We also provide brief comments on the effects of the parameters on the AUPRC.

LEAP. This method has a single parameter, `maxLag`, which controls the maximum amount of lag to consider. The authors of this method suggest not using a value larger than $1/3$. Hence, we tested seven values in the interval $[0, 0.33]$: 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, and 0.33.

SCODE. While SCODE has two parameters, `D`, and the number of iterations, we only varied the parameter `D`. For the number of iterations, we used 100, the same value as used by the authors. We also experimented with 1,000 iterations but found that this value did not improve the AUPRC (data not shown). In the documentation of the SCODE code, the authors suggest averaging the results from multiple runs. Therefore, we averaged the results across five runs. For `D`, we tested the values that they evaluated in their paper: 2, 4, 6, 8, and 10. We found this parameter had a large effect on the AUPRC and thus for each dataset, we used the value that gave the highest AUPRC.

SINCERITIES. SINCERITIES does not need any parameters when using temporal transcriptional data. However, when experimental time points are not available, the authors suggest discretizing pseudotime values into bins. We varied the number of bins between 5, the smallest number of time points their code would accept, and 20 using the specific values of 5, 6, 7, 8, 9, 10, 15, and 20. We did not consider values larger than 20 reasoning that datasets with so many timepoints were unlikely to be available.

SCRIBE. The authors used the combination “5,10,20,25” for a parameter called “delay.” We evaluated this value as well as several other combinations of delays: “5”, “5,10”, “5,10,15,20”, “5,10,15,20,25”, “5,10,20,25”, “5,10,25,50,75,100”, and “20,25,50,75,100”.

GRISLI. This method has three parameters: `R`, `L`, and α . The authors suggest that `R` be as large as possible to reduce random fluctuations of the algorithm. We chose to use `R`= 3,000. Following the authors’ recommendation to test `L` and α over a large grid, we tried values in the same ranges as in their paper: `L` = 1, 5, 10, 25, 50, 75, and 100 and α = 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, and 1.0. In the case of datasets from curated models, we varied α from 0 to 1.0 in steps of 0.25 in order to reduce the number of combinations we had to test.

SINGE. SINGE has a total of seven parameters. For five of these parameters, we used the same values as the authors of SINGE did: δt and `L` combinations of (3,5), (5,9), (9,5), (5,15), (15,5); kernel width σ = 0.5, 1, 2, 4; probability of zero removal and probability of removing samples values = 0, and 0.2 each. We used a reduced set of values for the sparsity parameter λ : we tested λ = 0 or 0.01 and the number of replicates = 2 or 6. We chose these parameter values since our initial, exploratory analysis showed that increasing λ to the other values used in SINGE (0.02, 0.05, 1) often resulted in decreased performance or very few or no output edges. We did not see any improvement

in increasing the number of replicates to 10 either. These parameter values resulted in a total of 320 combinations. While the authors of SINGE used an ensemble of parameter values, we chose to simply use the combination of parameters that resulted in the best AUPRC. Since varying the parameters corresponding to the number of replicates, probability of zero removal and the probability of removing samples did not affect the AUPRC, we chose 2, 0 and 0 for their values. For λ , small values of 0 or 0.01 were best. δt , L and σ had the largest effect on AUPRC.

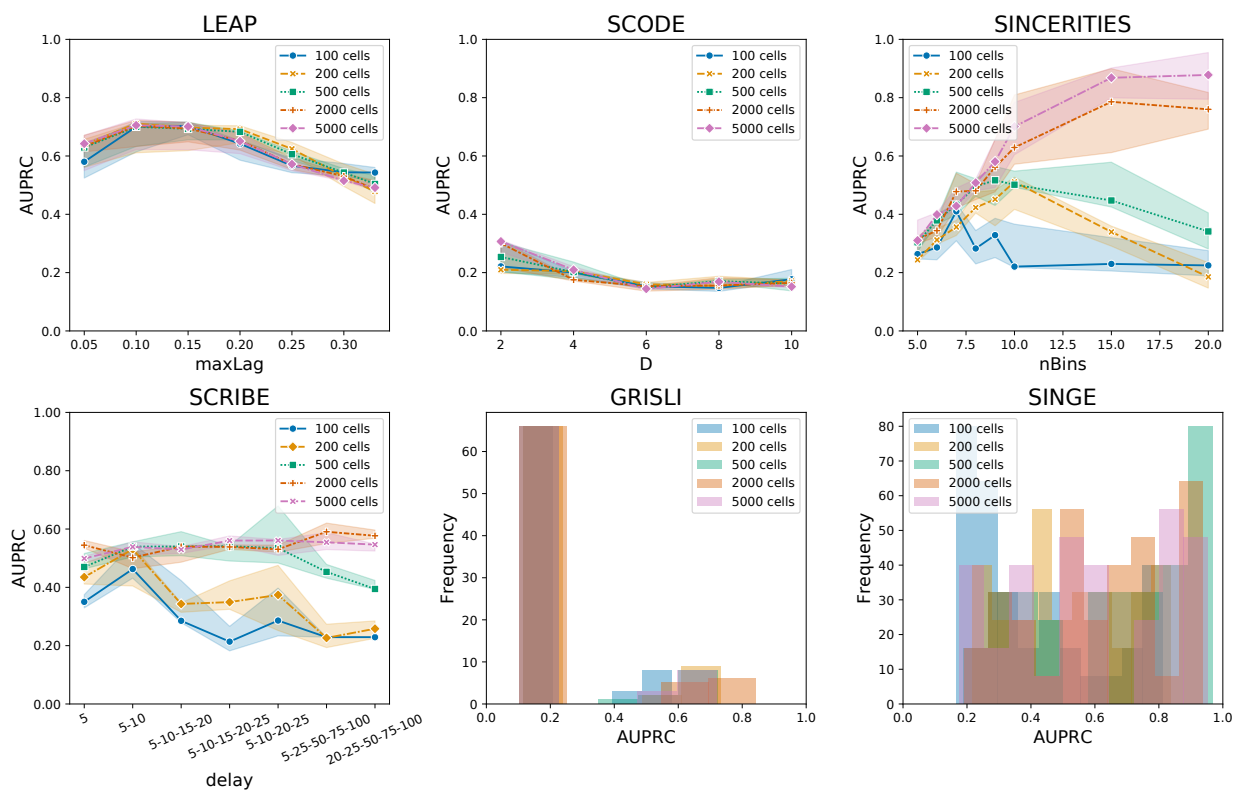


Figure B.2: Results of parameter search for each of the six methods with one or more parameters on the datasets (10 each) with 100, 200, 500, 2,000 and 5,000 cells for the Linear synthetic network. For the four methods with one parameter to vary, SCRIBE, SCODE, SINCERITIES, and LEAP, we show the median AUPRC (center line) as well as the 68% confidence interval of the median (shaded region), estimated using 1,000 bootstrapped samples of the data. For GRISLI and SINGE, which have two and seven parameters, respectively, we show histograms of the AUPRC over all parameter combinations we tested.

Figure B.2 summarizes the results from the parameter sweep and Table B.1 displays the best parameter value(s) for each method. We used these parameter values for the corresponding number of cells for the five other synthetic networks.

No. cells	LEAP	SCODE	SINCERITIES	SCRIBE	GRISLI		SINGE			
	maxLag	D	nBins	delay	L	α	λ	δt	L	σ
100	0.15	2	7	5,10	5	0.5	0.01	3	5	2
200	0.1	2	10	5,10	5	0.5	0	15	5	0.5
500	0.1	2	9	5,10,15,20,25	5	0	0.01	5	15	0.5
2000	0.1	2	15	5,25,50,75,100	5	0	0	9	5	0.5
5000	0.1	2	20	5,10,20,25	5	0	0	9	5	0.5

Table B.1: Parameter values that resulted in the highest median AUPRC for a given number of cells on the Linear synthetic network.

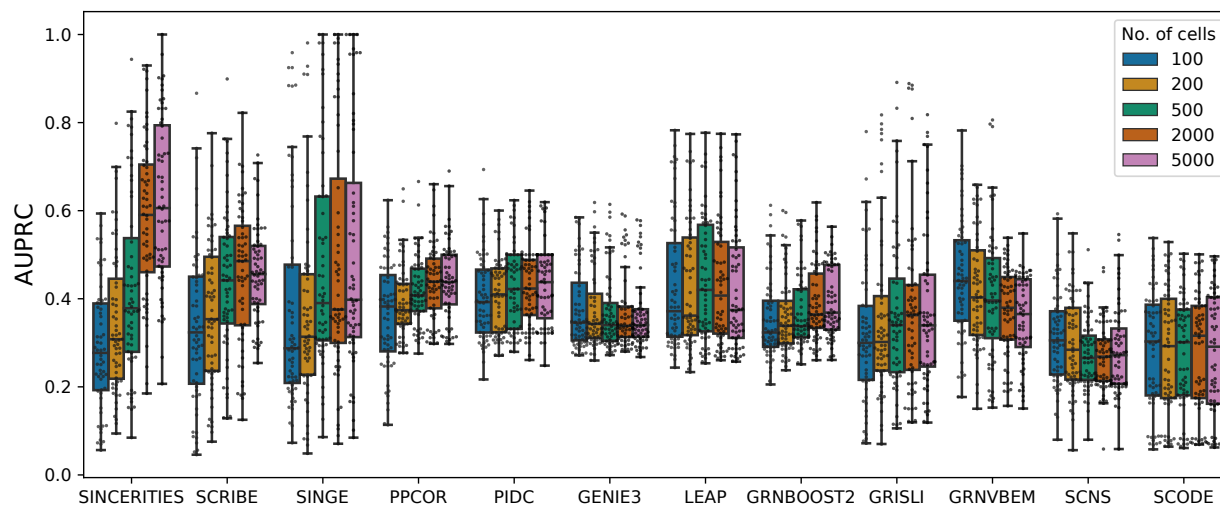


Figure B.3: Comparison of AUPRC values from all datasets from synthetic networks ($n = 60$), for each number of cells. In every boxplot, lower and upper hinges denote the 1st and 3rd quartiles, with whiskers extending to the largest value less than 1.5 times the interquartile range.

B.1.3 Effect of number of cells on AUPRC

To examine the effect of the number of cells on accuracy, for each algorithm, we compared the AUPRC values across all datasets from synthetic networks for 100, 200, 500, and 2,000 cells to those for 5,000 cells (Figure B.3). No algorithm had a statistically significant result for the 2000–5000 cell comparison (Table B.2). Four algorithms (GRNBoost2, PIDC, PPCOR, SINCERITIES) had significantly lower AUPRC values for 500 cells in comparison to 5000 cells. In addition to these four methods, SINGE and SCRIBE were significant in the 200–5000 cell comparison, with these two approaches and GRISLI also being significant in the 100–5000 cell comparison. The number of cells had no significant effect on five algorithms (GENIE3, GRNVBEM, LEAP, SCNS, and SCODE).

# of cells compared	SINCERITIES	SCRIBE	SINGE	PPCOR	PIDC	GENIE3
100–5000	9.18×10^{-10}	7.85×10^{-6}	2.91×10^{-4}	5.28×10^{-6}	8.18×10^{-4}	1.0
200–5000	3.07×10^{-9}	2.49×10^{-5}	2.71×10^{-5}	2.47×10^{-6}	2.62×10^{-5}	1.0
500–5000	6.86×10^{-8}	0.30	0.35	2.36×10^{-6}	2.93×10^{-2}	0.67
2000–5000	0.12	0.71	0.31	0.35	0.40	1.0
# of cells compared	LEAP	GRNBoost2	GRISLI	GRNVBEM	SCNS	SCODE
100–5000	1.0	5.62×10^{-4}	2.26×10^{-3}	1.0	1.0	1.0
200–5000	1.0	2.02×10^{-4}	0.41	1.0	1.0	1.0
500–5000	1.0	5.71×10^{-4}	0.59	1.0	0.91	0.88
2000–5000	1.0	0.87	1.0	1.0	0.66	1.0

Table B.2: Table of Benjamini-Hochberg-corrected one-sided Wilcoxon signed-rank test q -values corresponding to Figure B.3. Bold text highlights q -values ≤ 0.05 .

B.1.4 Fidelity of simulations from inferred GRNs

We sought to determine if a GRN inferred by an algorithm from datasets from synthetic networks, when represented as a Boolean network, would exhibit dynamics and steady states identical to the original models. We first verified that the synthetic networks yielded the expected number of steady states under asynchronous updates, namely, zero for Cycle, one each for Linear, Linear Long, and Bifurcating Converging, two for Bifurcating, and three for Trifurcating network.

We removed all self-loops from the ranked edges lists output by each algorithm. Then, we traversed the ranked list of edges output by each algorithm until each gene had at least one incoming interaction. SCNS produced such a network only for the Linear Long dataset; we attribute this result to the fact that we use a uniform value of threshold parameter for all genes due to the high running time of the method. For the output from GRNVBEM, SCODE, and SINCERITIES, we used the signs of interactions (activation or repression) predicted by that algorithm. For every other algorithm, we assigned the signs of interactions based on the values computed by PPCOR. We added self-interactions if they were present in the ground truth network.

Next, we defined the rule for each gene as follows: a gene is ON if and only if at least one activator is ON and every inhibitor is OFF. We simulated this Boolean model asynchronously, i.e., at any state, we selected a gene uniformly at random and applied the rule for that gene to compute the next state. We computed the network formed by the transitions among all the possible states. With both the Boolean network and this “state space network” in hand, we computed two sets of statistics.

- (a) We recorded the number of the interactions in the Boolean network created from the computed GRN and its ratio with the number of interactions in the ground truth network.
- (b) We had simulated each ground truth network starting from a specific initial state, e.g., for the Linear model, we set g_1 to ON and all other genes to OFF. We asked how many steady states we could reach from this starting state in the Boolean network created from the computed GRN and recorded the ratio of this number with the number of steady states in the ground truth network.

Note that we did not perform this analysis for datasets from curated models since the Boolean rules in these networks were quite complex. We did not expect that the relatively simple rules used by BoolODE would suffice to recapture the properties of the Boolean networks we curated from the literature.

The six columns titled “Edge Ratio” in Figure B.4 present the number of edges in each such network divided by the number in the ground truth network. Almost all the methods required fewer than twice the number of edges in the ground truth networks, except for the

Linear Long network. PPCOR achieved an edge ratio of 1.0 for four of six networks. SCODE had the largest edge ratios for all the networks.

The columns titled “Ratio of Reachable States” in Figure B.4 display the number of steady states reachable from the initial condition used to simulate each ground truth network. Five of the twelve methods had a mean ratio of one for both the Linear and the Bifurcating networks, i.e., the number of steady states were one and two, respectively. While eight of the twelve methods yielded a mean of one steady state for the Linear network, only four methods had the same value for the Linear Long network. In general, most methods showed high edge ratios for the Linear Long network, they also over-predicted the number of steady states. We observed that all the methods predicted steady states for the Cycle network, failing to capture the expected limit cycle behavior. For the Bifurcating Converging network, the methods in general over-predicted the number of steady states, while for the Trifurcating network, all the methods (except LEAP) had either higher or lower median values for the number of reachable states.

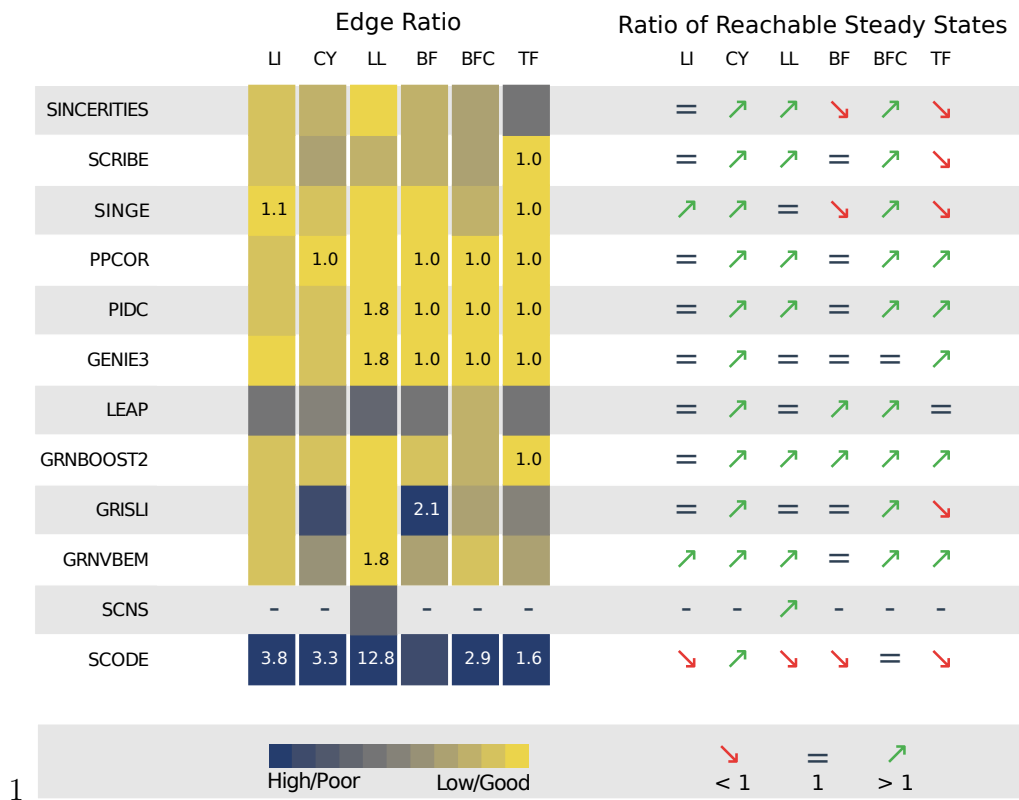


Figure B.4: Summary of results for datasets from synthetic networks. The first set of six columns displays the median edge ratios. The next set of six columns display the median ratios of reachable steady states of the proposed GRN reconstructions for each of the six simulated datasets. The color is proportional to the corresponding value after scaling it between 0 and 1. For each dataset, we display the highest and lowest values inside the corresponding cells. We computed the medians over 20 simulated datasets for each synthetic network containing 2,000 and 5,000 cells. Abbreviations: LI: Linear, CY: Cycle, LL: Linear Long, BF: Bifurcating, BFC: Bifurcating Converging, TF: Trifurcating.

B.1.5 Effect of using trajectory information

Seven algorithms require the cells in the gene expression data to be ordered by (pseudo)time but cannot handle datasets representing multiple lineages. We compared the predictive performance of these algorithms in two situations: (a) split the cells into multiple trajectories, infer the GRN for each lineage, and combine these GRNs into network (see “Methods” for details), and (b) merge the trajectories into one and infer a single network. See “Datasets with multiple trajectories” in “Evaluation Pipeline” for details. We performed this comparison to check if access to information on a cell’s lineage gave these algorithms any advantage. By and large, there was virtually no difference between these two approaches for most algorithms (Figure B.5, Benjamini-Hochberg-corrected one-sided Wilcoxon signed-rank test

q -value > 0.05). An exception was SCRIBE for which splitting trajectories yielded a superior AUPRC to the merged approach for at least two networks (q -value = 0.032). SINGE and SINCERITIES were contrasting exceptions, since merged trajectories performed better for all three networks (q -value = 0.007 and 0.004, respectively). This absence of a clear benefit of splitting over merging or vice versa may be a consequence of the numerous false positive interactions in the results, as indicated by the values of AUPRC.

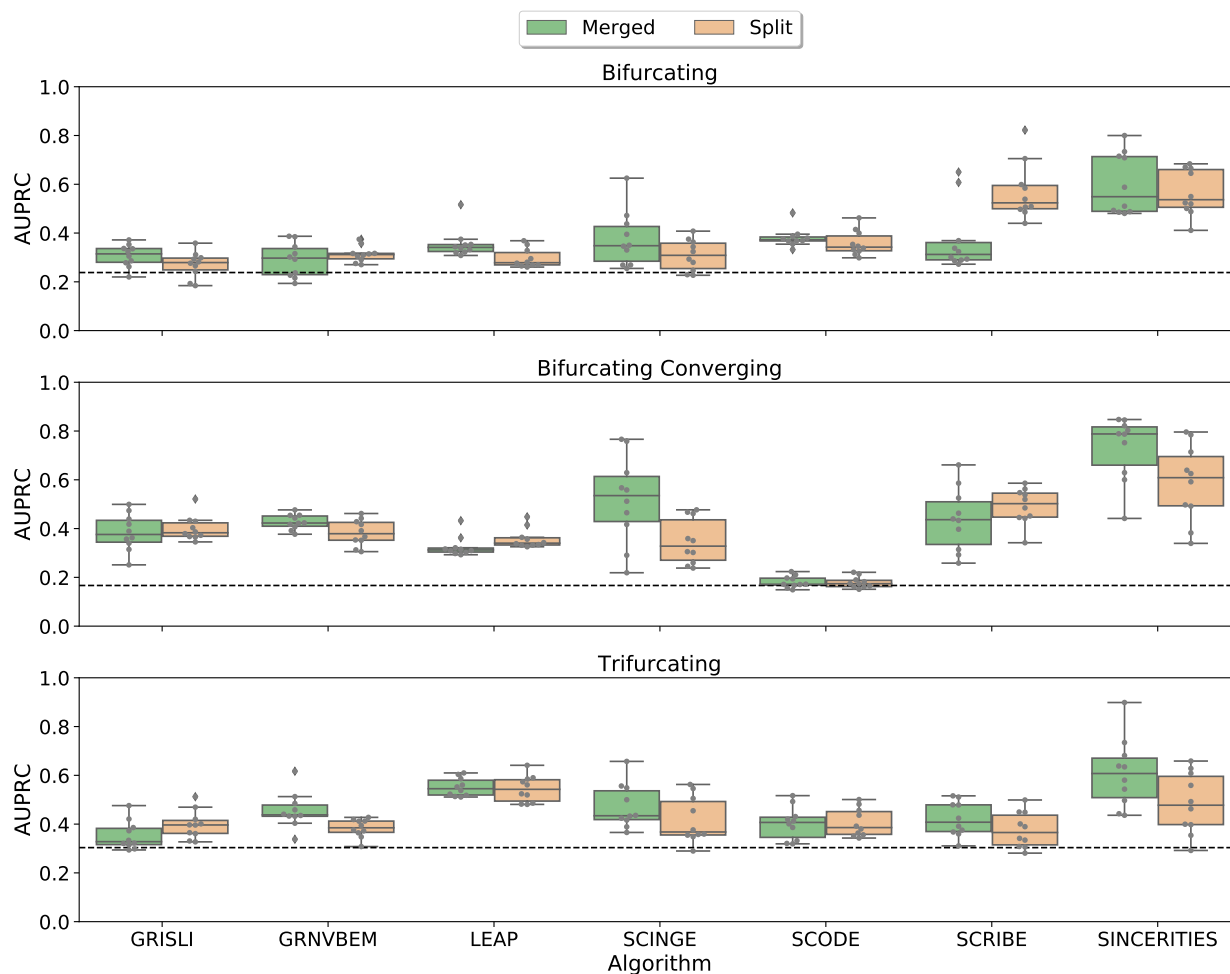


Figure B.5: Comparison of AUPRC values with split (orange) and with merged (green) trajectories. Each row corresponds to one synthetic network that produces at least two trajectories. Each column corresponds to an algorithm. Each boxplot represents 10 AUPRC values (10 datasets each containing 2,000 cells). In every boxplot, lower and upper hinges denote the 1st and 3rd quartiles, with whiskers extending to the largest value less than 1.5 times the interquartile range.

B.2 Datasets from curated models

B.2.1 Correspondence of BoolODE simulations and steady states of Boolean models

We sought to determine if the simulated datasets could capture steady state properties of the Boolean models, as reported in the corresponding publications. We performed this analysis in two stages.

(i) Figure 4.6 shows the two-dimensional t-SNE visualizations colored with simulation time for one of 10 datasets created by BoolODE for each Boolean model. Examining these plots visually, we observed continuous, discernible trajectories that ran along the entire simulation timeline, as seen by the colours of the cells changing gradually from blue to green to yellow. Next, we carried out k -means clustering of simulations with k set to the number of steady states of the model, as mentioned in the corresponding publication. Figure 4.6(c) displays these results, with a different color for each cluster (two for mCAD, five for VSC, four for HSC, and two for GSD). We visually corresponded the plots in Figures 4.6(b) and 4.6(c) to confirm that each cluster contained cells spanning the entire length of the simulation. After running the pseudotime inference algorithm Slingshot [91], we visualized the principal curves computed by this method in Figure 4.6(d); each such curve computed is a smooth representation of the trajectory. The correlation between the pseudotime and the simulation time was high (Table B.3). Therefore, we gave the pseudotime as input to the GRN algorithms.

(ii) Each publication describing a Boolean model specifies a unique gene expression pattern that characterizes each steady state of that model. We asked if our simulated data matched these patterns. To this end, we colored each cell (a point on the 2D t-SNE plot) with the simulated expression value of each gene in that cell, with darker colors corresponding to larger values (Figures B.6 to B.9). We explain the procedure we followed using an example. The mCAD model has two steady states: Anterior (characterized by up-regulation of Pax6, Fgf8, and Sp8) and Posterior (up-regulation of Coup and Emx2). In Figure B.6, we observed that Pax6, Fgf8, and Sp8 had high simulated expression values in a group of cells at the top right of the t-SNE plot whereas Coup and Emx2 had large values in the group in the bottom left. We concluded that these two cell groups corresponded to the Anterior and Posterior steady states of the Boolean model, respectively. We succeeded in finding similar matches for each of the other models (Figures B.7 to B.9).

Network	Dropout rate	Correlation
mCAD	0	0.81
	50%, 0.5	0.78
	70%, 0.7	0.46
VSC	0	0.75
	50%, 0.5	0.33
	70%, 0.7	0.12
HSC	0	0.73
	50%, 0.5	0.49
	70%, 0.7	0.34
GSD	0	0.92
	50%, 0.5	0.90
	70%, 0.7	0.88

Table B.3: Correlation between simulation time and pseudotime for different dropout rates.

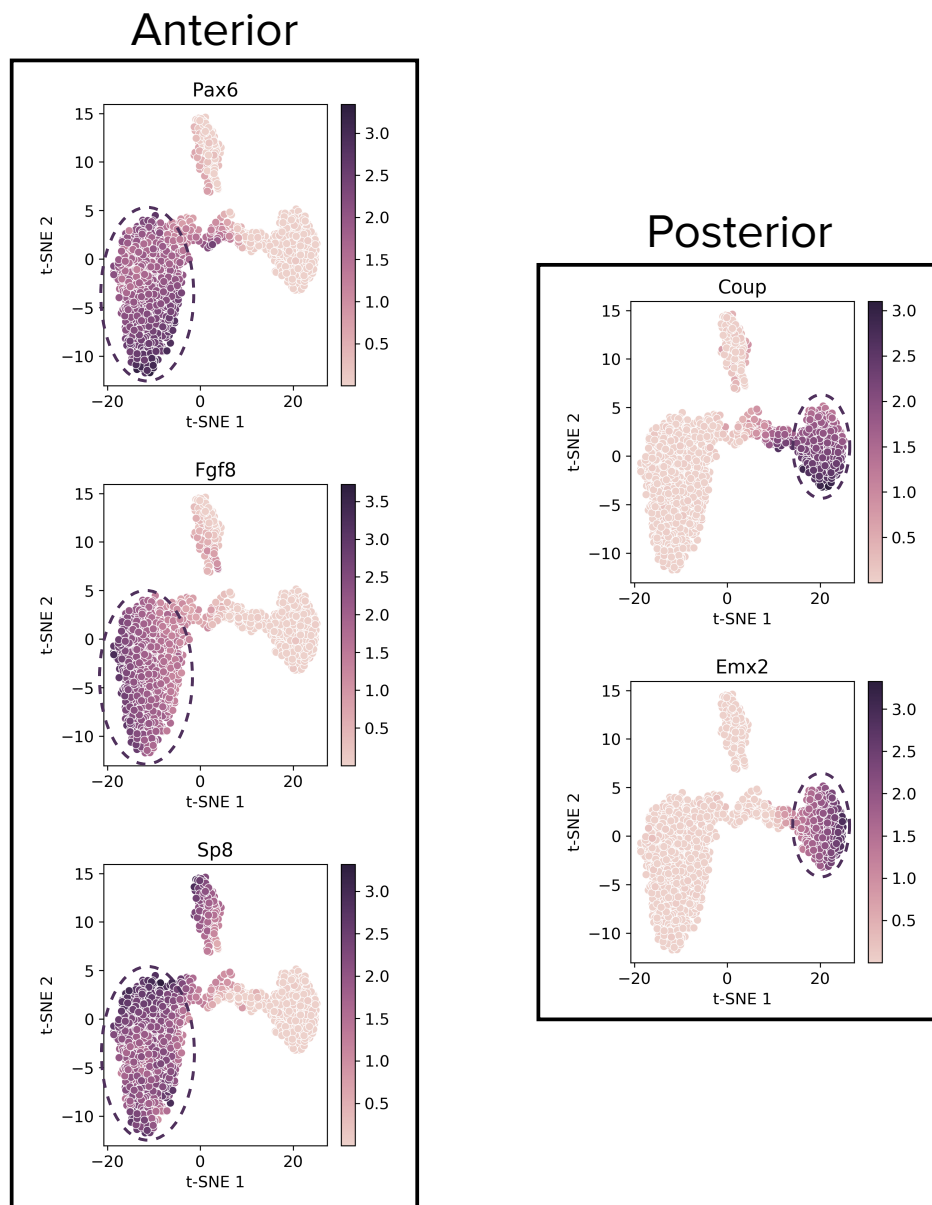


Figure B.6: t-SNE visualizations of simulated trajectories from the mCAD model. The bar on the right of each plot displays the mapping between the expression value of a gene and the corresponding color. The model has two steady states corresponding to the anterior and posterior compartments during cortical area development [87]. The anterior compartment is characterized by high transcriptional activity of Fgf8, Pax6 and Sp8, while the posterior compartment is characterized by the high transcriptional activity of Coup-tf1 and Emx2. The steady states corresponding to the two compartments appear as two distinct clusters in the t-SNE visualizations.

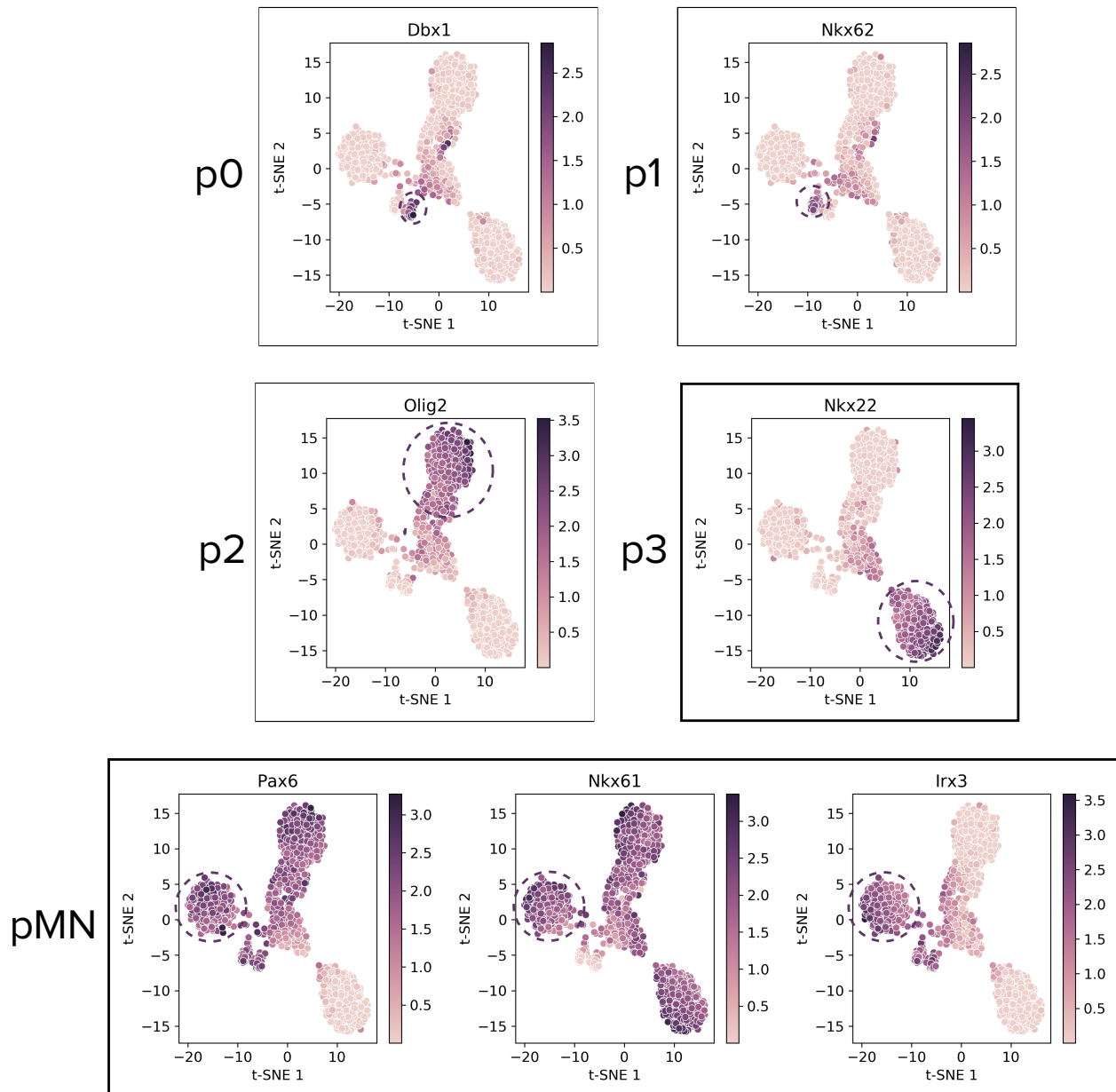


Figure B.7: t-SNE visualizations of simulated trajectories from the VSC model. This model exhibits five distinct steady states corresponding to five distinct cell types in the neural tube [88]. Each of these cell types appears as a distinct cluster in the tSNE visualizations shown above. The transcription factors specific to the p0 (Dbx1), p1 (Nkx6.2), p2 (Olig2), and p3 (Nkx2.2) states are shown in the four plots on top. The pMN state shown in the lower plot is characterized by high activity of Nkx6.1, Pax6 and Irx3.

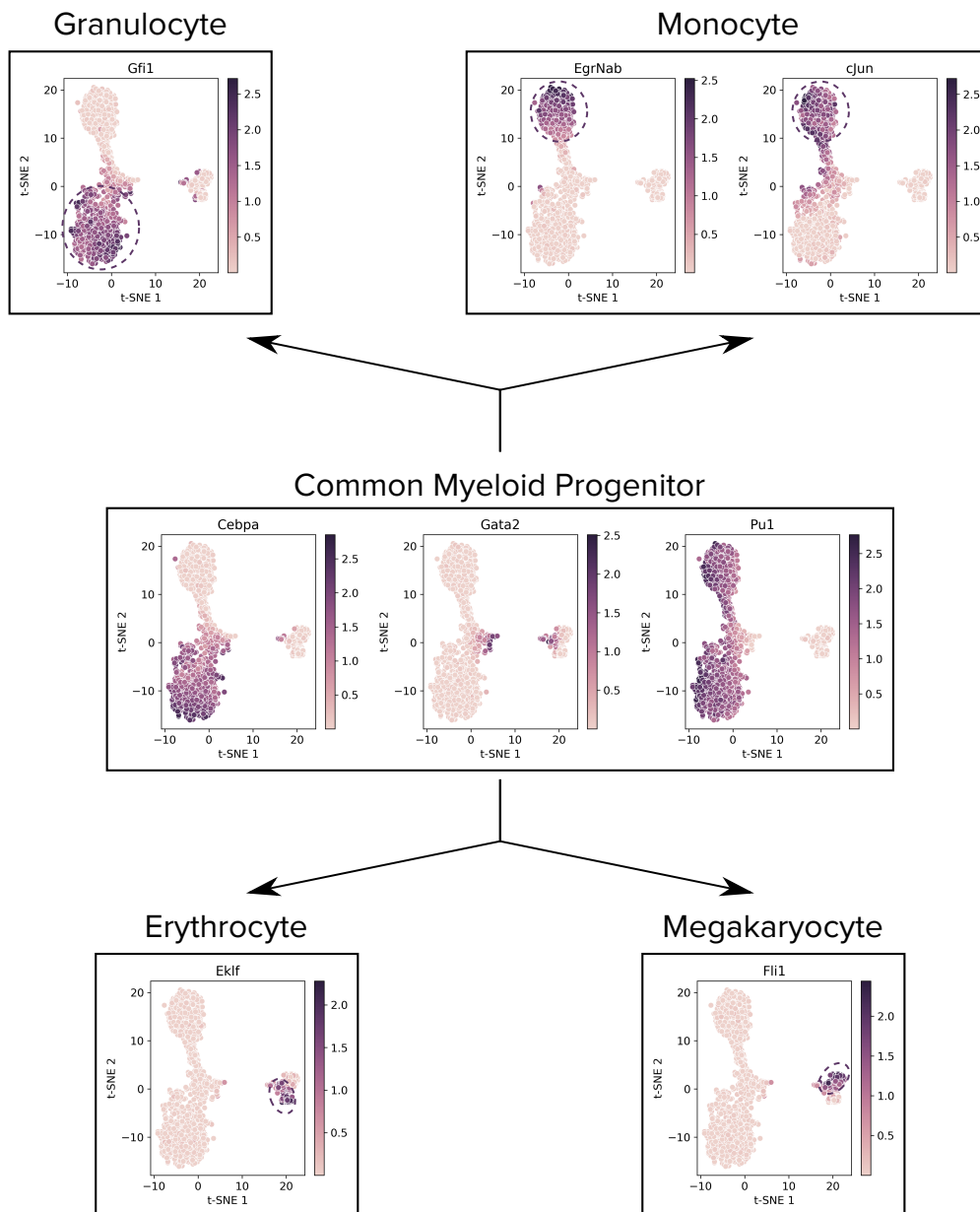


Figure B.8: t-SNE visualizations of simulated trajectories from the HSC model. In the initial state, which corresponds to the Common Myeloid Progenitor (CMP), the activities of GATA-2, PU.1 and C/EBP α are high [89]. There are two main branches of differentiation. The branch leading to the monocyte-granulocyte progenitors is characterized by the inactivation of GATA-2, followed by the activation of either *Gfi1* (granulocyte) or *cJun* and *EgrNab* (monocyte) [89]. The clusters in the t-SNE visualization that correspond to these states are shown in the plots on top. Similarly, the branch leading to the megakaryocyte-erythrocyte progenitors is characterized by the inactivation of C/EBP α and PU.1 along with the concomitant activation of GATA1, FOG1 and SCL [89]. The subsequent inactivation of GATA2 along with activation of *Fli1* leads to the megakaryocyte state while that with the activation of *EKLF* leads to the erythrocyte state [89]. These states are shown at the bottom.

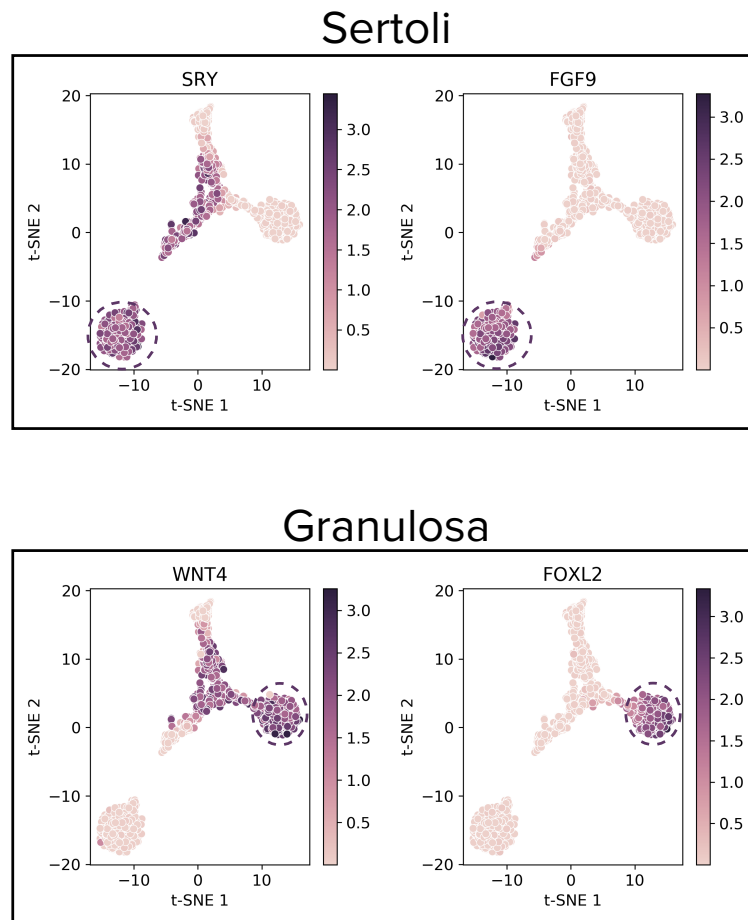


Figure B.9: t-SNE visualizations of simulated trajectories from the GSD model. This model has two steady states corresponding to Sertoli cells (male gonads) or Granulosa cells (female gonads) [90]. The clusters on the tSNE plots correspond to these two states. SRY and FGF9 show male-gonad-specific activity while WNT4 and FOXL2 show female gonad-specific activity. Except for GATA4 and WT1mKTS, which are active in both branches, all other transcription factors in the model belong to one of the two clusters shown above.

B.2.2 Parameter Search

For each of the six methods that require input parameters, we performed a parameter sweep using a procedure similar to the one described in Supplementary Note Section B.1.2. However, there were a few notable differences. Since we only had datasets from curated models with a fixed number of 2,000 cells, we performed parameter sweep once for each model on the 10 non-dropout samples. For those algorithms that additionally use pseudotime information, we ran each algorithm on each individual trajectory for each dataset, combined the ranked lists of edges, and then computed the AUPRC. We chose this approach since every GRN inference algorithm in our evaluation that had parameters and had been developed explicitly for single-cell gene expression data could take only one trajectory as input. Since each of these datasets had 10 replicates, we chose the parameter value or combination of values that maximized the median AUPRC for each method. Figure B.10 shows the results for parameter search for the four different curated models and Table B.4 displays the best parameter value(s) for each method.

Model	LEAP	SCODE	SINC.	SCRIBE	GRISLI		SINGE			
	maxLag	D	nBins	delay	L	α	λ	δt	L	σ
GSD	0.1	2	6	5,25,50,75,100	10	0	0.01	15	5	0.5
HSC	0.05	2	20	5	10	0.25	0.01	3	5	1
mCAD	0.3	6	10	5,25,50,75,100	100	0	0	5	9	0.5
VSC	0.33	10	5	5,10,15,20,25	10	0	0.01	15	5	0.5

Table B.4: Parameter values that resulted in the highest median AUPRC for each curated model. Abbreviations: SINC, SINCERITIES

B.2.3 Effect of dropouts on AUPRC

To study the effect of dropouts, for each algorithm, we compared the distributions of AUPRC scores across all datasets from curated models between the dropout rates $q = 0$ and $q = 50$ and between $q = 0$ and $q = 70$ (Figure B.11). Four GRN inference methods had a statistically significant difference in AUPRC values when comparing $q = 0$ to $q = 50$ (GENIE3, GRNBoost2, SCODE, GRISLI, Table B.5). For the other comparison, four additional algorithms were significant (PIDC, PPCOR, SINGE, SCNS), with SCODE being just above the 0.05 threshold. The four algorithms that were unaffected by dropout rates (LEAP, SINCERITIES, GRNVBEM, SCRIBE) had worse-than-random AUPRC values on the mCAD and VSC datasets.

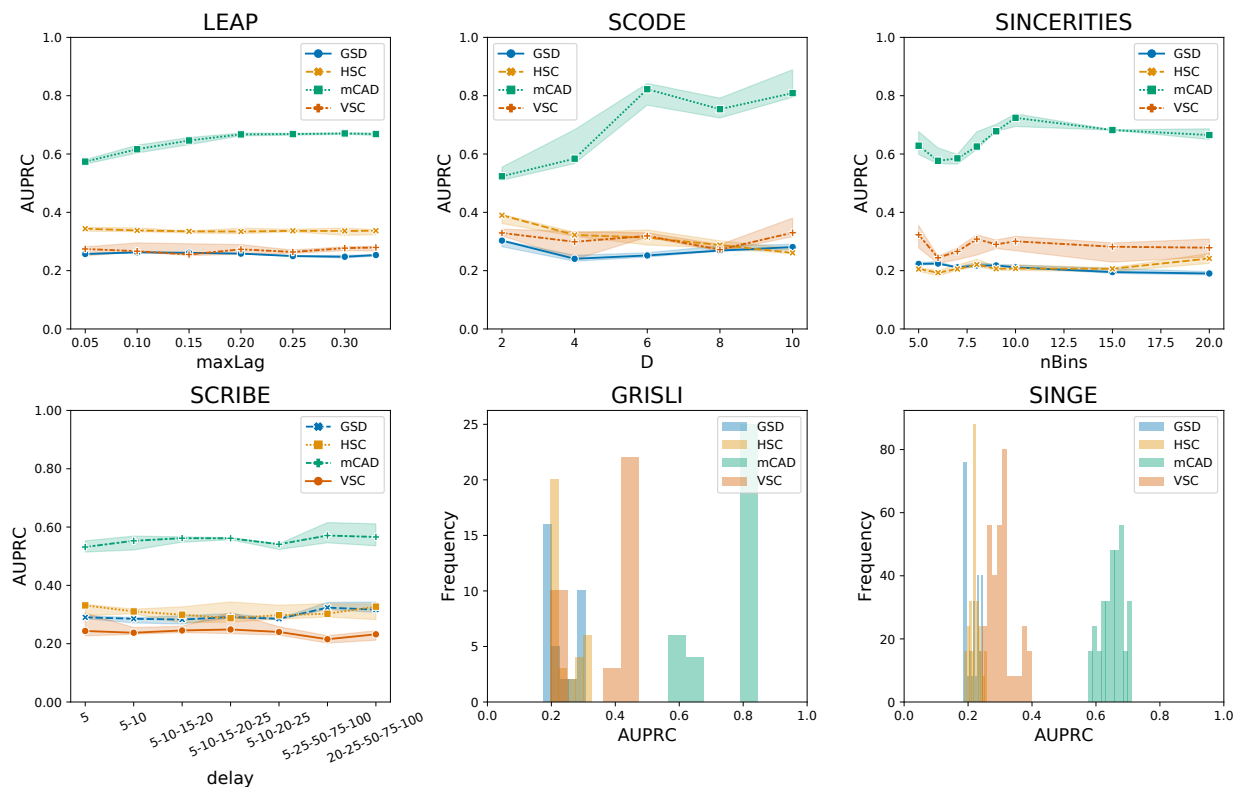


Figure B.10: Results of parameter search for each of the six methods with one or more parameters on the 10 datasets with 2,000 cells and no dropouts for the four curated Boolean models. For the four methods with one parameter to vary, SCRIBE, SCODE, SINCERITIES, and LEAP, we show the median AUPRC (center line) as well as the 68% confidence interval of the median (shaded region), estimated using 1,000 bootstrapped samples of the data. For GRISLI and SINGE, which have two and seven parameters, respectively, we show histograms of the AUPRC over all parameter combinations we tested.

B.2.4 Comparing similarities of algorithm outputs

Our results indicate moderate to poor accuracy of the networks inferred by the 12 methods on datasets from curated models (Figure 4). We were curious if the algorithms made similar errors or if there were some subnetworks of the Boolean models that were easy for most algorithms to infer correctly. We performed three analyses for each model separately. (i) We converted each algorithm’s output into a vector of edges and recorded the edge ranks in this vector. We used PCA on the set of vectors on all 10 datasets and all 12 algorithms to compute a two-dimensional projection (Figure B.12). (ii) For each dataset we computed the Spearman’s correlation coefficients between these vectors for all pairs of algorithms. Figure B.13a presents the results for one representative dataset. (iii) For the top- k edges for

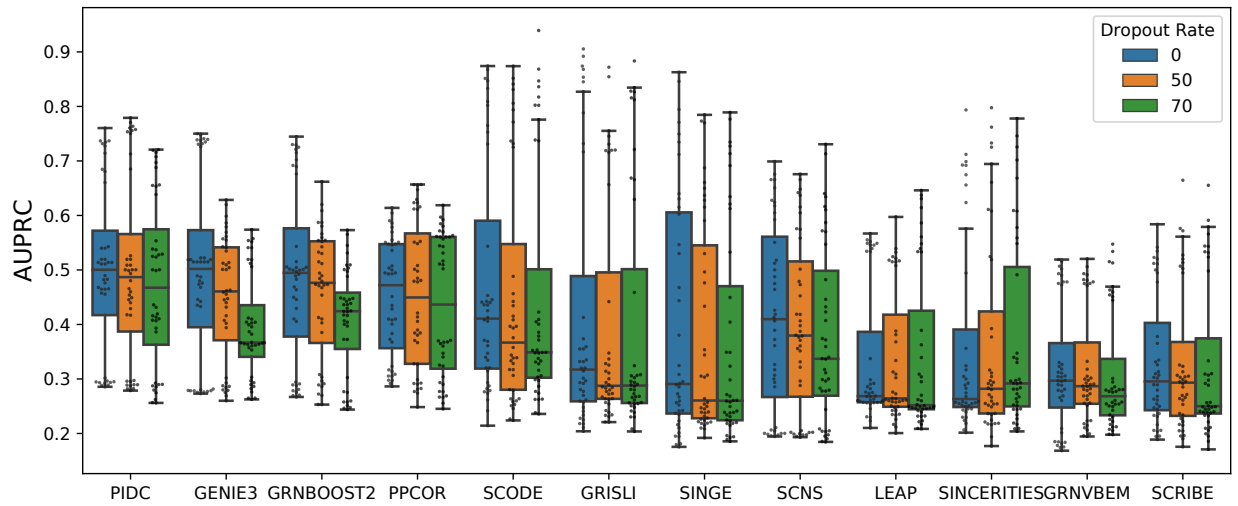


Figure B.11: Comparison of AUPRC values from the four datasets from curated models ($n = 40$), for three dropout rates. In every boxplot, lower and upper hinges denote the first and third quartiles, with whiskers extending to the largest value less than 1.5 times the interquartile range.

Dropout rates compared	PIDC	GENIE3	GRNBoost2	PPCOR	SCODE	GRISLI
0–50	0.15	4.54×10^{-3}	0.03	0.6	0.034	0.03
0–70	4.17×10^{-4}	5.78×10^{-3}	6.87×10^{-4}	0.03	0.051	0.02
Dropout rates compared	SINGE	SCNS	LEAP	SINCERITIES	GRNVBEM	SCRIBE
0–50	0.09	0.85	0.25	0.74	0.71	0.6
0–70	0.04	0.03	0.65	0.82	0.09	0.23

Table B.5: Benjamini-Hochberg-corrected q -values for the one-sided Wilcoxon signed-rank test corresponding to Figure B.11. Bold text highlights q -values ≤ 0.05 .

each dataset, we computed the Jaccard indices between all pairs of algorithms. Figure B.13b presents the results for one representative dataset. The PCA results suggest that some algorithms have somewhat similar outputs (e.g., GENIE3, GRNBoost2, and PIDC for all models, with LEAP and PPCOR each being similar to the first three methods for two of the four models). The Spearman’s correlations and Jaccard indices echo these trends. GENIE3 and GRNBoost2 are both based on random forests, which may explain their similarities. Similarly, both LEAP and PPCOR are based on computing correlations. What is striking is that most pairs of Spearman’s correlations are somewhat negative (between -0.25 and 0) and most pairs of Jaccard indices are less than 0.5, suggesting that the outputs are quite

different from each other.

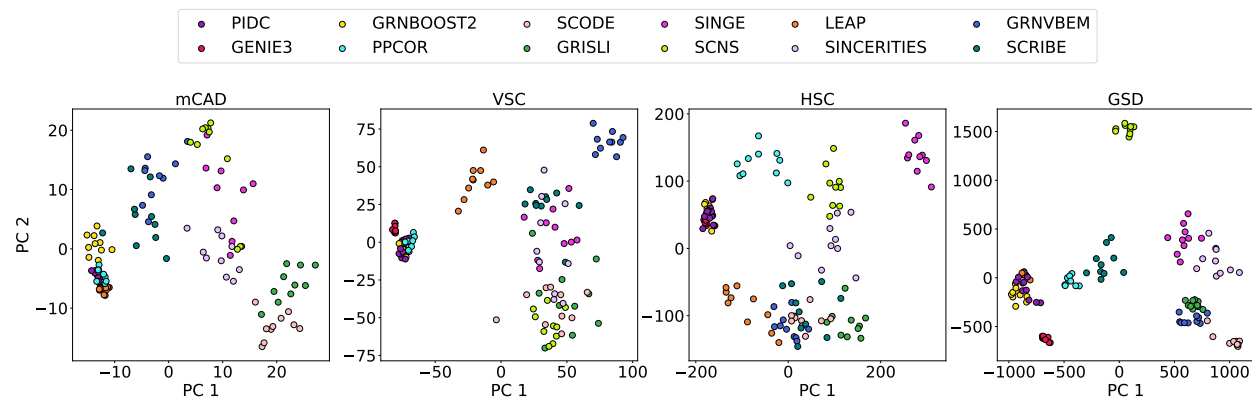
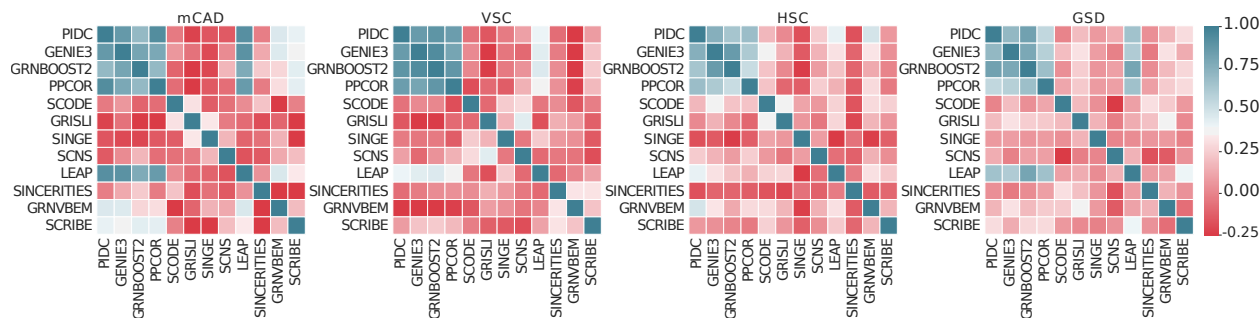


Figure B.12: PCA projections of edge rank vectors. Each point corresponds to a vector of edge ranks predicted by an algorithm for a dataset.

B.2.5 Network motifs

We investigated if certain network motifs were over-represented in the top- k networks. Specifically, we counted the number of three-node feedback and feedforward loops and (two-node) mutual interactions in the top- k networks from each GRN algorithm and reported their ratios with their respective values in the ground truth network. For this analysis, we ignored algorithms that predict only undirected edges (PIDC and PPCOR). We report the medians of these values (over the 10 datasets we simulated for each model) in Figure B.14(a). For the mCAD model, we observed that every GRN inference method yielded close to expected or lower than expected numbers of feedback and feed-forward loops compared to the ground truth network. This trend is likely due to the fact that the mCAD network is fairly dense. For the VSC model, which primarily has mutual inhibitory edges, 9 out of the 10 models had expected or lower than expected number of mutual interactions. For the HSC and GSD networks, we noted an overall decrease in feedback loops and mutual interactions but an increase in feedforward loops. In the case of feedback loops, we noted that LEAP and GRISLI were the only methods that predicted roughly as many or more feedback loops than in the ground truth network for each of the four Boolean models. SCODE and SINCERITIES typically produced fewer feedback loops and mutual interactions but more feedforward loops.

(a) Spearman correlation



(b) Jaccard index

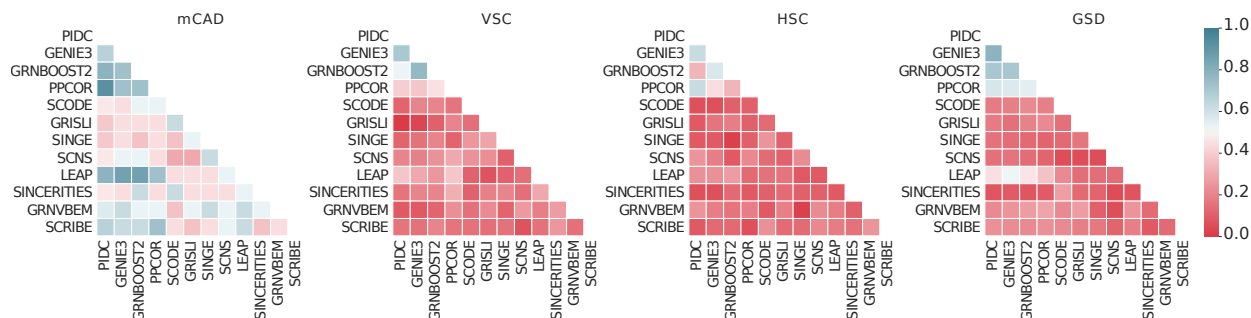


Figure B.13: Summary of output similarity for datasets from curated models. (a) Heatmap of the Spearman correlation matrix for a representative dataset. Each cell represents the correlation value between the edge rank vectors for a pair of algorithms. (b) Heatmap of the Jaccard index for a representative dataset. Each cell in the heatmap represents the Jaccard index between the sets of top- k edges for a pair of algorithms.

B.2.6 Explaining false positives in top- k networks

We reasoned that the poor value of early precision for a GRN inference algorithm could be due to its tendency to predict “indirect” interactions, i.e., if, in the Boolean model, gene a regulates gene b , which in turn regulates gene c , a method may also predict an interaction between a and c . This type of configuration is a feedforward loop, which we have seen to be in excess over the ground truth network for several algorithms (Figure B.14). To confirm the possibility of indirect interactions, we considered each false positive edge (u, v) in a top- k network and computed the length of the shortest path between u and v in the corresponding ground truth network. Figure B.14(b) and (c) summarize the results.

We discuss the results for the VSC, HSC, and GSD models first. We observed that the endpoints of about 5% to 20% of false positive edges were not even connected in the ground truth networks by any path (bars labeled “No” in Figure B.14(b)). Virtually all the remaining false positive edges corresponded to paths of lengths between two and four in the ground truth network (Figure B.14(c)); What was most striking was that the largest proportion of

these false positive edges were connected by paths of length two, i.e., if added to the ground truth network, they would indeed form feed-forward loops. This result lends strong credence to our hypothesis that GRN inference algorithms are predicting indirect interactions.

In the case of the mCAD model, a very large fraction of false positive edges corresponded to paths of length two in the corresponding ground truth network. We reasoned that the density of the mCAD network was the cause for this statistic despite the lower prevalence of feedforward loops in the predicted networks for the mCAD model.

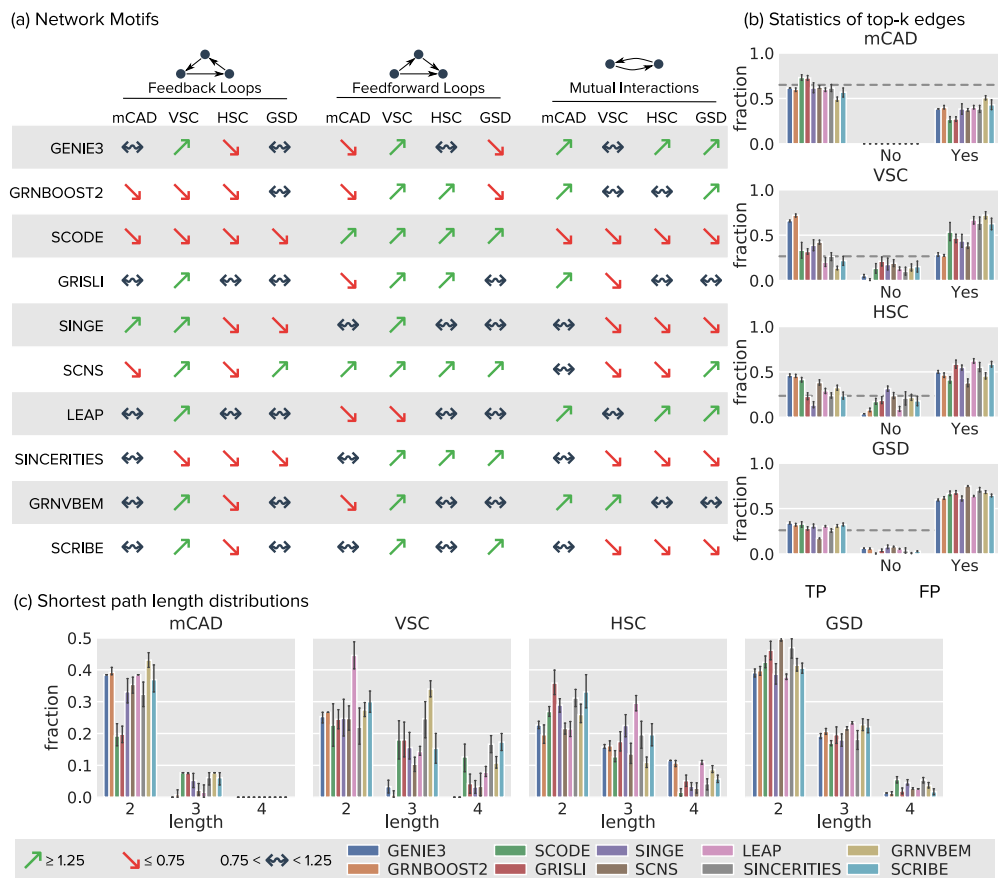


Figure B.14: Summary of network motif analysis. (a) Analysis of network motifs in the predicted top- k networks using datasets from curated models for feedback loops, feedforward loops and mutual interactions. We counted the number of occurrences of each motif type in each predicted top- k network, as well as in the ground truth network, and we report the ratio of these counts. Green upward pointing arrows represent ratios greater than 1.25, red downward pointing arrows represent ratios less than 0.75, and the black arrows represent ratios between 0.75 and 1.25. (b) Statistics on the edges in the top- k network for each method divided into three parts: fraction of top- k edges that are true positives (TP), fraction of top- k edges that are false positives and whose endpoints are not connected by any path in the ground truth network (FP, No), and the fraction of top- k edges that are false positives and whose endpoints are connected by a path (of length two or more) in the ground truth network (FP, Yes). Each colored bar corresponds to one algorithm. The height of a bar represents the mean and the error bar the 95% confidence interval over the 10 datasets. The gray line indicates the density of the network. (c) For every false positive edge in each predicted top- k network, we computed the length of the shortest paths in the ground truth network that connected the endpoints of this edge. Bar plots show the fractions of top- k edges that are false positive edges and whose endpoints have paths of lengths 2, 3, and 4 in the ground truth network in the top- k networks, colored by the method. The height of a bar represents the mean and the error bar the 95% confidence interval over the 10 datasets.

B.3 Experimental single-Cell RNA-seq datasets

B.3.1 Parameter search

For the seven experimental single-cell RNA-seq datasets and each of the four subsets of genes, we performed a parameter sweep for SCODE and SINCERITIES, and selected the parameter that yielded the highest EPR when evaluated on the non-specific ChIP-Seq ground truth network (Figure B.15, Table B.6). We used these parameter values for the other two types of ground truth networks.

In the case of SINCERITIES, which was developed for temporal single cell expression data, we also provided the experimental time as input for the mESC and hESC datasets. For five out of eight subsets of genes (across both datasets), SINCERITIES had a higher EPR when we first computed pseudotime and did parameter estimation than when we used experimental time (Figure B.15). In the case of the TFs+500 gene set for the hESC dataset, SINCERITIES gave a runtime error for experimental time and for every parameter value. For the 1,000 gene set in the same dataset, SINCERITIES ran successfully for experimental time but not for any parameter value. Only for the 500 gene set in the same dataset did the experimental time have a higher EPR than from parameter estimation.

Dataset	SCODE				SINCERITIES			
	D				nBins			
	500	TFs+500	1k	TFs+1k	500	TFs+500	1k	TFs+1k
mHSC-E	4	2	2	2	10	10	8	-
mHSC-L	2	2	2	2	20	15	20	20
mHSC-GM	10	10	2	4	10	10	-	-
mESC	6	8	4	10	6	5	6	6
mDC	10	10	4	6	10	20	15	20
hHEP	2	6	10	6	10	15	15	20
hESC	2	2	4	4	Exp.	-	Exp.	10

Table B.6: Parameter values that resulted in the highest EPR for a given experimental single-cell RNA-seq datasets and gene set, evaluated on the non-specific ChIP-Seq networks. Abbreviations: Exp.: scRNA-seq experiment time, 1k: 1000

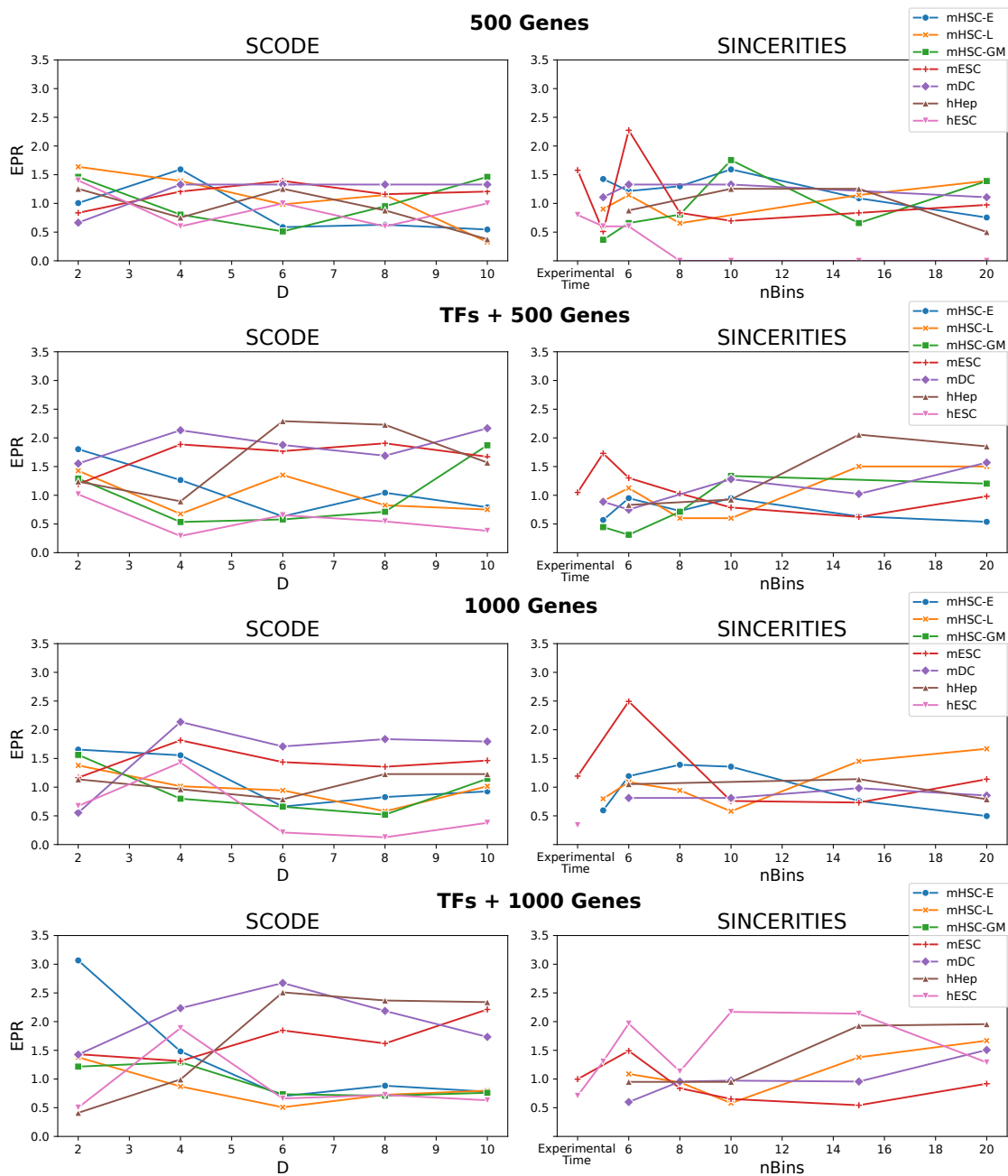


Figure B.15: Results of the parameter search for SCODE and SINCERITIES on experimental single-cell RNA-seq datasets, with four different sets of genes. We used the non-specific ChIP-Seq networks as the ground truth. For SINCERITIES, the point for “Experimental Time” corresponds to using the experimental time points available in the hESC and mESC datasets. Points that are missing correspond to cases where SINCERITIES gave an error and did not produce output.

B.3.2 Correspondence of clusters in GRNs and gene expression datasets

We investigated how much the clusters in the predicted GRNs agreed with those in the gene expression datasets. We first computed a co-expression network using the absolute value of the Pearson's correlation coefficient as the weight of the edge between two genes. Next, we ran the Louvain clustering algorithm [156] on the predicted GRN (considering all edges and their weights) and on the co-expression network to compute clusters. In order to compute clusters using the Louvain algorithm, we converted the GRNs into undirected networks: if a method predicted both the edge (a, b) and (b, a) , we used only the higher of the two edge weights. Next, we measured the similarity between these clusterings using the Adjusted Rand Index [119]. We recorded the values of this index for each pair of experimental single-cell RNA-Seq dataset and algorithm. Values of this index close to 1 indicate that the modules inferred from GRNs are similar to the ones detected using gene expression datasets. Averaged over all experimental single-cell RNA-seq datasets, we observed that GRNBoost2 had the highest ARI value of 0.485 and PPCOR had the lowest ARI value of 0.004, i.e., close to random (Figure B.16). The ordering of the methods by average ARI was close to the ranking by median EPR, suggesting that modules in the more accurate GRNs have a good concordance with clusters computed directly from gene expression data.

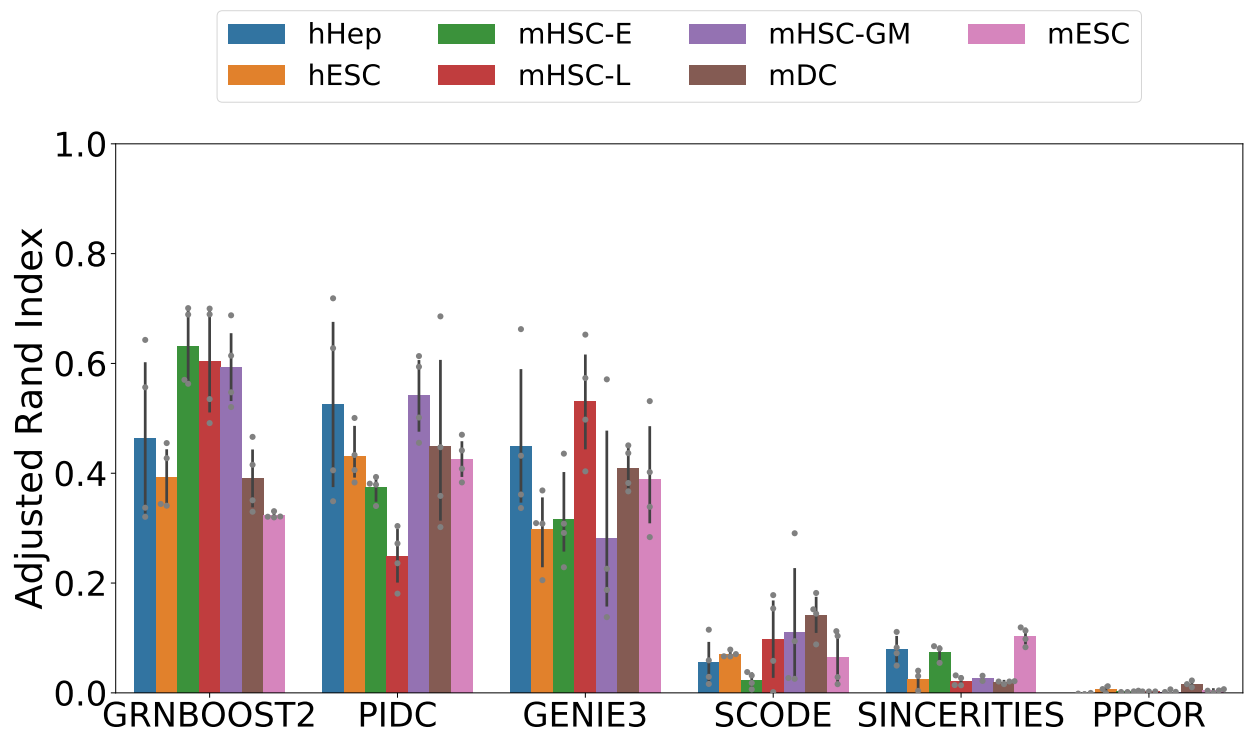


Figure B.16: Bar plot of Adjusted Rand Index values for clusterings generated using real gene expression datasets and GRNs predicted by an algorithm.

B.3.3 Effect of gene selection strategy

We sought to examine the effect of the number of genes (gene sets “500” and “1,000”) as well as including all TFs (sets “TFs+500” and “TFs+1,000”), on the performance of the methods. Therefore, for each gene set, we combined the EPR values of the three top performing methods (PIDC, GENIE3, and GRNBoost2) on the non-specific ChIP-Seq and STRING networks. A p -value of 0.017 for the Kruskal-Wallis test indicated that EPR values for these four sets of genes did not originate from the same distribution. To further investigate if these differences arose from the differences in the number of genes or the inclusion of TFs, we grouped the gene sets as follows:

- (i) To study the effect of addition of TFs, we combined EPR values for the gene sets “500” and “1000” and compared them to the union of the EPR values for “TFs+500” and “TFs+1000” ($n = 84$).
- (ii) To study the effect of the number of genes, we compared the union of the EPR values for “500” and “TFs+500” with the union of “1000” and “TFs+1000” ($n = 84$).

We found that including TFs resulted in a statistically significant improvement (one-sided Mann-Whitney U test p -value 0.003), while including more highly varying genes did not (p -value 0.084). Thus, our analysis suggests that including significantly varying TFs may be more beneficial than adding more highly-varying genes. The AUPRC ratio did not show statistically significant variation with the number of genes or the addition of TFs.

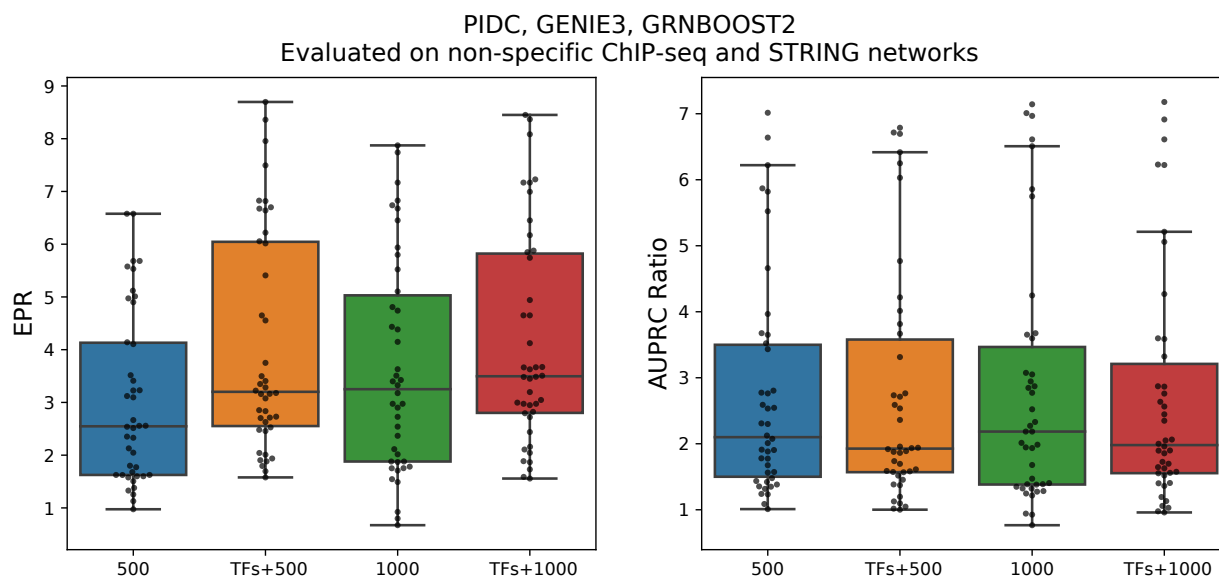
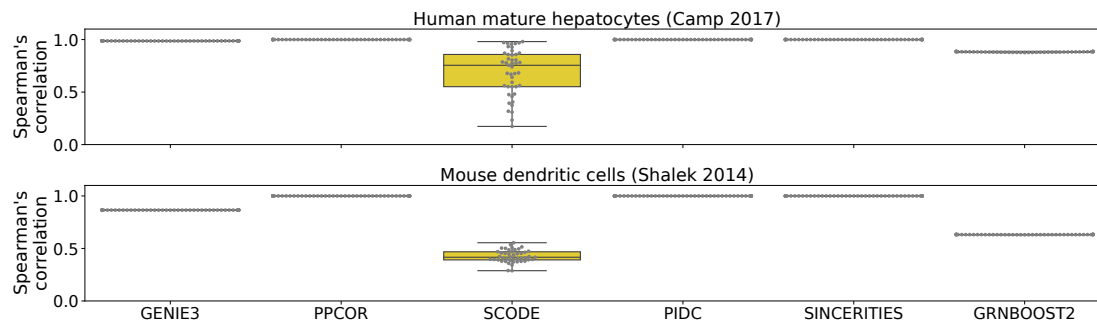


Figure B.17: Boxplots of EPR and AUPRC ratios of PIDC, GENIE3, and GRNBoost2 for experimental single-cell RNA-seq datasets on the 500 most-varying genes, all significantly-varying TFs with the 500 most-varying genes, and similarly defined sets for 1,000 genes evaluated on the non-specific ChIP-Seq and STRING ground truth networks. Each box contains a total of 42 values. In each boxplot, lower and upper hinges denote the 1st and 3rd quartiles, with whiskers extending to the largest value less than 1.5 times the interquartile range.

B.3.4 Stability across multiple runs

(a) Experimental single-cell RNA-seq datasets



(b) Boolean networks

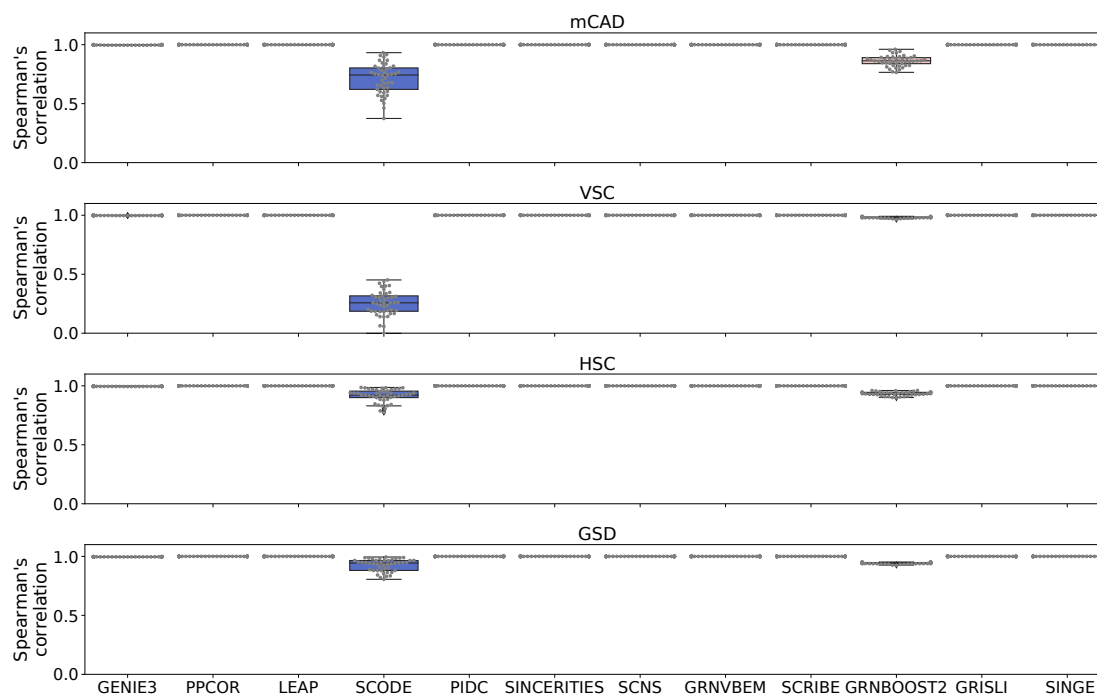


Figure B.18: Box plot of Spearman's correlations of edge ranks for 10 runs on (a) two representative experimental single-cell RNA-seq datasets and (b) on four curated models. Each row corresponds to one dataset or model. Each column corresponds to an algorithm. Each box plot represents 45 Spearman's correlation values. In each boxplot, lower and upper hinges denote the 1st and 3rd quartiles, with whiskers extending to the largest value less than 1.5 times the interquartile range.

We selected two experimental single-cell RNA-seq datasets (human mature hepatocytes [95] and mouse dendritic cells [93]). On each of these datasets, we ran the six algorithms we selected for experimental datasets ten times and computed the Spearman's correlations between all 45 pairs of ranked edge lists output by these runs. We observed that every algorithm

produced almost identical outputs across these multiple runs with the exception of GENIE3, GRNBoost2, and SCODE, which had a median Spearman's correlation of 0.92, 0.75, and 0.55 respectively (Figure B.18(a)). We also performed this analysis with all twelve algorithms for one 2,000 cell dataset for every Boolean model. While GENIE3 and GRNBoost2 had less variation for experimental single-cell RNA-seq datasets than for simulated models, SCODE had highly varying outputs for all types of datasets (Figure B.18(b)).

B.3.5 Ensembles of GRNs

Inspired by the success of ensembles in inferring GRNs from bulk transcriptional data [101], we attempted to combine the results of the algorithms. We tried four approaches:

Borda: re-ranks every edge based on the averages of its ranks computed by each GRN inference algorithm;

mBorda: a modification of the Borda method that computes a weighted rank for each edge after down-weighting worse ranks by assigning a value of $1/n^2$ for the n^{th} ranked edge [59];

sBorda: a version of Borda that combines only the three methods with the highest AUPRC or EPR [101]; and

smBorda: a modification of mBorda that combines only the three methods with the highest AUPRC or EPR.

For sBorda and smBorda, we combined PIDC, GENIE2, and GRNBoost2. We performed this analysis only for the non-specific ChIP-Seq network since the STRING network contains functional (indirect) interactions.

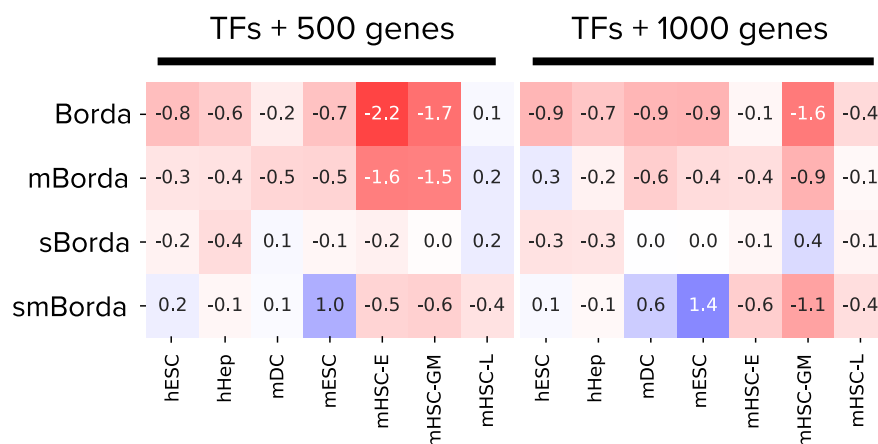


Figure B.19: Difference in EPR value obtained by Borda aggregation and the best performing algorithm’s EPR on the non-specific ChIP-Seq network. Each row corresponds to one of the four Borda-type methods. Each column corresponds to a experimental single-cell RNA-seq dataset. The value in each cell is the difference in EPR value. The color of a cell is a shade of red (respectively, blue) if the difference is negative (respectively, positive).

Figure B.19 displays the performance of these ensembles for the experimental single-cell RNA-seq datasets. The smBorda method was substantially better than the best performing individual algorithm for the mESC dataset [157] (both “TFs+1000 genes” and “TFs+500

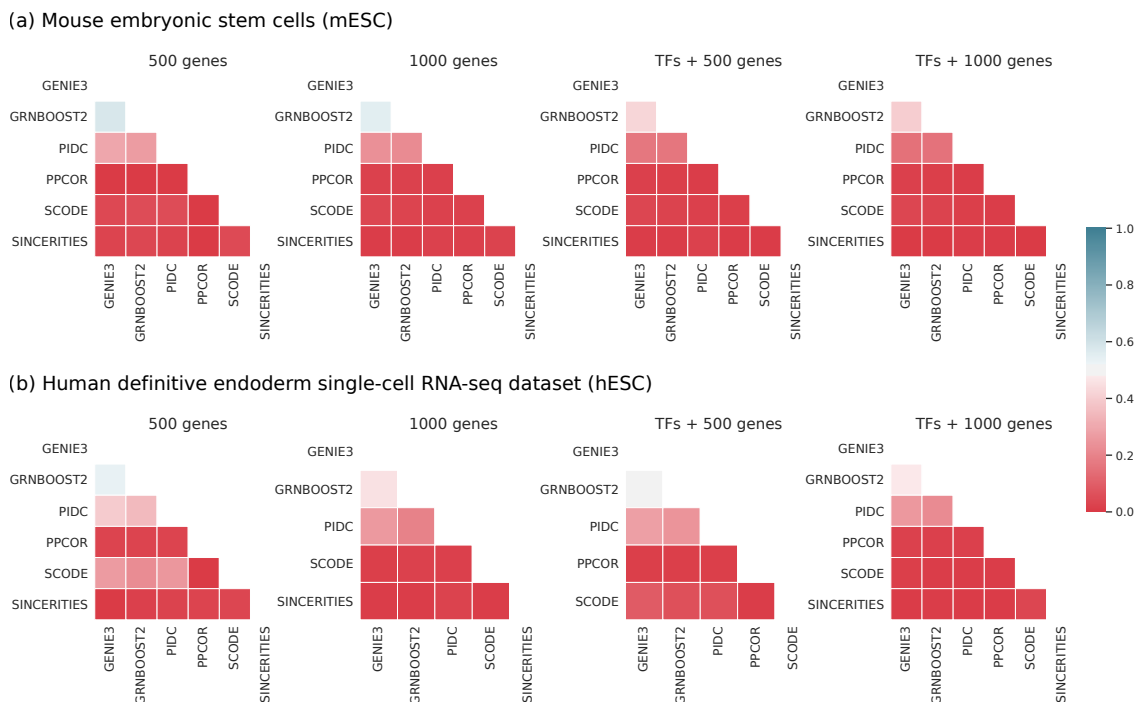


Figure B.20: Heatmap of the Jaccard index for the experimental single-cell RNA-seq dataset in (a) mouse embryonic stem cells (mESC) [96], and (b) human definitive endoderm single-cell RNA-Seq dataset (hESC) [92]. Each cell in the heatmap represents the Jaccard index between the sets of top- k edges for a pair of algorithms. We set k to be the number of edges in the intersection of the corresponding set of genes and the mouse non-specific ChIP-Seq network.

genes” sets) and the mDC dataset [93] (only the “TFs+1000 genes” set) for both the non-specific ChIP-Seq and STRING networks. However, the smBorda method performed worse than or almost the same as the best individual algorithm (difference ≤ 0.1) for ten of the 14 combinations. The other three strategies (Borda, mBorda, and sBorda) either did not perform systematically better than the algorithm with the highest EPR or showed only marginal improvements. We speculate that the low similarities between the outputs of the best methods (Figure B.20) prevented the creation of effective ensembles.

B.4 Software Details

In this section, we provide more information on how we executed each algorithm.

GENIE3 and GRNBoost2. We used the Python package Arboreto [57] for running GENIE3 and GRNBoost2. We choose this implementation of GENIE3 since it is the version used in an application of this method on single-cell data [53]. The current implementation of GENIE3 and GRNBoost2 in Arboreto takes an expression matrix and a list of genes as input. It produces a ranked list of edges as output, which we used directly in our evaluations.

PPCOR. We used the R implementation of PPCOR available on CRAN. The software has options to use nonparametric partial and semi-partial correlation coefficients based on Kendalls and Spearman's rank correlations. We used Spearman's correlation. We used a cut-off of 0.01 on the p -values computed by PPCOR to decide whether to include an interaction in the output or not, similar to the approach used in Pseudotime Network Inference [63]. We ordered the interactions in decreasing order of absolute value of the correlation.

LEAP. We used the R implementation of LEAP. The software has options for cutoffs for maximum lag and maximum absolute correlation (MAC) and whether or not to obtain a directed network as output. We choose the option to obtain a directed network. Since we were interested in obtaining a rank for every edge, we set the MAC cutoff to 0. We searched for the best value of maximum lag as described in the "Parameter Search" section. LEAP outputs an adjacency matrix, which we converted to a ranked list of edges based on the MAC score.

SCODE. SCODE is available both as an R and a Julia package. We used the R implementation. The software has options for the number of reduced dimensions, number of iterations for optimizing ODE parameters, and for running SCODE several times. We describe how we selected these values in the "Parameter Search" section. SCODE outputs an adjacency matrix A , with positive and negative values for edge weights indicating activation or inhibition. In order to obtain the ranked list of edges, we sorted the edges by the absolute values of the edge weights.

PIDC. PIDC is available in a Julia package titled "NetworkInference". This package has options to select various network inference algorithms such as Mutual Information, CLR, PUC and PIDC [52]. We used the maximum likelihood estimator, as recommended by the authors. We did not use any edge-weight cut-off. We converted the resulting undirected network in the form of an adjacency matrix into a ranked edge list.

SINCERITIES. SINCERITIES is available both as an R and a MATLAB package. We used the R implementation. The software has options for the distance parameter, regularization strategy, and whether or not to obtain self loops. We used the Kolmogorov-Smirnov distance and Ridge regularization as described in the SINCERITIES paper. SINCERITIES also recommends binning pseudotime values into a pre-specified number of bins. Therefore, we performed a parameter estimation on the number of bins. SINCERITIES outputs a ranked list of edges along with edge weights, which we used directly in our evaluations.

SCNS. We used the F# implementation of SCNS. SCNS requires the following inputs: binarized gene expression values for each cell, a list of cells in the initial state, a list of cells in final states, maximum numbers of activators and inhibitors for each gene, and a threshold parameter. We first binarized the gene expression data by setting the gene's expression to zero if it was lower than average expression of that gene across all cells. We sorted the cells by pseudotime and specified that the cells in the bottom 10th percentile were in the initial state and those in the top 10th percentile were in the final state. For each gene we specified the same number of activators and inhibitors as in the reference network. We set the threshold parameter to 95. We used this value since lowering it caused SCNS to find a very large number of rules for some genes in some datasets. Moreover, the running time of SCNS made parameter search prohibitively expensive. For each gene, this method outputs a Boolean formula that combines the states (on/off) of the regulators to produce the state of the regulated gene. The formula has a specific structure: the gene is 'on' if and only at least one activator is 'on' and all inhibitors are 'off'. We converted each such formula into a directed network as follows: we created an edge directed from each regulator to the target gene. We ignored whether a regulator acts as an activator or inhibitor, and simply created an edge if the regulator appeared in the rule. For each gene, SCNS also outputs multiple Boolean formulae that satisfy the given input parameters. Therefore, we counted the number of rules in which a regulator appeared and used this quantity as the edge weight. We sorted this list of edges based on these weights.

SCRIBE. SCRIBE is available as an R package. Among the many variants of Restricted Directed Information presented in this paper, we used the ucRDI with $k = 1$, since this alternative appeared to have the highest accuracy in the SCRIBE manuscript; $k = 1$ means computing the RDI value conditioned on each other gene individually in order to remove indirect causal interactions. The paper says that after applying CLR, it is possible to further sparsify the network using a method for regularization. However, we did not include this step since the documentation for their R package did not give any guidelines on what functions to invoke here. SCRIBE outputs an adjacency matrix of RDI values, which we converted to an edge list.

GRISLI. GRISLI is available as a MATLAB package. GRISLI has options for parameters such as R , L and α , for which we performed parameter estimation using values recommended by the authors. GRISLI outputs a list of ranks for each edge as an adjacency matrix, which we then converted to a ranked edge list.

SINGE. SINGE is available as a MATLAB package. The software has options for various parameters, for which we performed parameter estimation. SINGE outputs a ranked list of edges along with edge weights, which we used directly in our analysis.