dyngen: benchmarking with in silico single cells

Robrecht Cannoodt
Wouter Saelens
Louise Deconinck
Yvan Saeys

01 December 2019

Purpose: A common problem of pioneering computational tools is that during their development, there are rarely sufficient datasets available for adequately quantitatively assessing its performance.
Results: We developed dyngen, a multi-modality simulator of single cells. In dyngen, the biomolecular state of an *in silico* cell changes over time according to a set of reactions defined by the cell's gene regulatory network. By simulating single cells in terms of its biomolecular state and reactions, the simulator is easily extendible for adding new modalities or experimental procedures. We demonstrate dyngen's flexibility by simulating snapshot, time-series and perturbation experiments.
Conclusion: dyngen lays the foundations for benchmarking a wide variety of computational single-cell tools and can be used to help kick-start the development of future types of analyses.

Author contributions:

- R.C. and W.S. designed the study.
- R.C., W.S., and L.D. performed the experiments and analysed the data.
- R.C. and W.S. implemented the dyngen software package.
- R.C. wrote the original manuscript.
- R.C., W.S., L.D., and Y.S. reviewed and edited the manuscript.
- Y.S. supervised the project.

Introduction

Continuous technological advancements to single-cells omics are having profound effects on how researchers can validate biological hypotheses. Early experimental technologies typically only allowed profiling a single modality (e.g. DNA sequence, RNA or protein expression). However, recent developments permit profiling multiple modalities simultaneously, and every modality added allows for new types of analyses that can be performed.

This presents method developers with a problem. The majority of the 250+ peer-reviewed computational tools for analysing single cell omics data were published without a quantitative assessment of the accuracy of the tool. This is partially due to low availability of suitable benchmarking datasets; even if there are sufficient suitable input datasets available, these are often not accompanied by the necessary metadata to serve as ground-truth for a benchmark.

Here, synthetic data plays a crucial role in asserting minimum performance requirements for novel tools in anticipation of adequate real data. Generators of scRNA-seq data (e.g. splatter[1], powsimR[2], PROSSTT[3] and SymSim[4]) have already been widely used to explore the strengths and weaknesses of computational tools, both by method developers[5, 6, 7, 8] and independent benchmarkers[9, 10, 11]. However, a limitation of scRNA-seq profiles generators is that they would require significant methodological alterations to add additional modalities.

An ideal experiment would be able to observe all aspects of a cell, including a full history of its molecular states, spatial positions and environmental interactions[12]. While this falls outside the reach of current experimental technologies, generating synthetic data in anticipation of new experimental technologies would allow already developing the next wave of computational tools.

We introduce dyngen, a multi-modality simulator of single cells and their dynamics (Figure 1A). A cell is simulated using Gillespie's Stochastic Simulation Algorithm (SSA)[13] where a cell consists of a set of molecules. Throughout a simulation, reactions (e.g. transcription, splicing) modify the abundance of these molecules. The likelihood of a reaction occurring is again dependent on molecule abundance.

By simulating a cell over time in terms of its molecular state and the reactions that are allowed to occur, the simulator is more extendible to new modalities or experimental procedures (Figure 1B). We demonstrate dyngen's flexibility by

simulating snapshot, time-series and perturbation experiments, allowing to benchmark a wide variety of computational tools such as trajectory alignment and differential network inference methods.

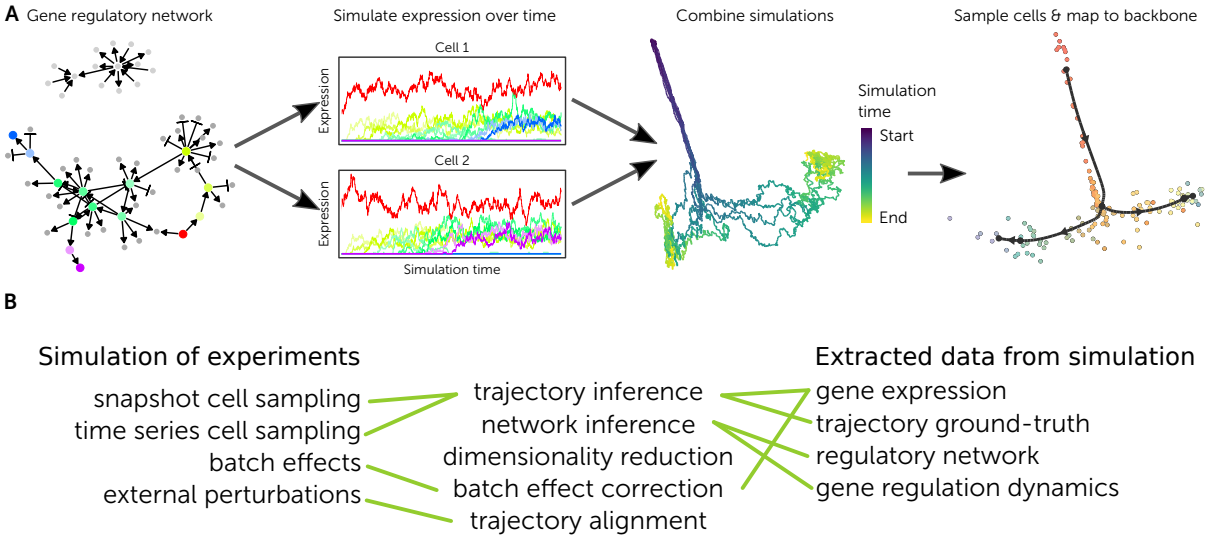

Figure 1: **Showcase of dyngen functionality. A: B:** Evaluating different types of computational tools requires simulating different types of experiments and extracting different layers of information from the simulation.

Results

A cell consists of a set of molecules, the abundance of which are affected by a set of reactions: transcription, splicing, translation, and degradation (Figure 2A). A gene regulatory network (GRN) defines the reactions that are allowed to occur (Figure 2B) and is constructed such that a cell will develop over time (Figure 2C–D). The likelihood of a reaction occurring is a function of the abundance of key molecules involved in each reaction (Figure 2E).

dyngen returns many modalities throughout the whole simulation: molecular abundance, cellular state, number of reaction firings, reaction likelihoods, and regulation activations (Figure 2C–F). These modalities can serve both as input data and ground truth for benchmarking many types of computational approaches. For example, a network inference method could use mRNA abundance and cellular states as inputs and its output could be benchmarked against the gold standard GRN.

Depending on how the GRN is designed, different cellular developmental processes can be simulated. dyngen includes generators of GRNs which result in many different developmental topologies (Figure 3), including branching, converging, cyclic and even disconnected.

Custom-defined GRNs offer more fine-grained control over the simulation. Aside from simulating topologies currently not supported by dyngen, this has several other important use-cases. Simulations of the same GRN with small perturbations allow to emulate batch effects or perturbation experiments (Figure 4). Simulating perturbed GRNs allow evaluating trajectory alignment methods – which attempt to map two or more trajectories onto each other – or differential network inference methods – which infer differential regulatory interactions between two or more groups of profiles.

dyngen can be used to simulate different experimental conditions. By default, dyngen supports snapshot experiments (uniformly sampling from an asynchronous dynamic process) and time-series experiments (sampling cells from different intervals in the simulation). However, it is possible to imagine and implement other sampling strategies, such as sampling a cell at a certain time point and once more at a later time point. This would allow evaluating the performance of RNA velocity approaches – which predict the future state of a cell by looking at differences in pre-mRNA and mRNA abundance levels.

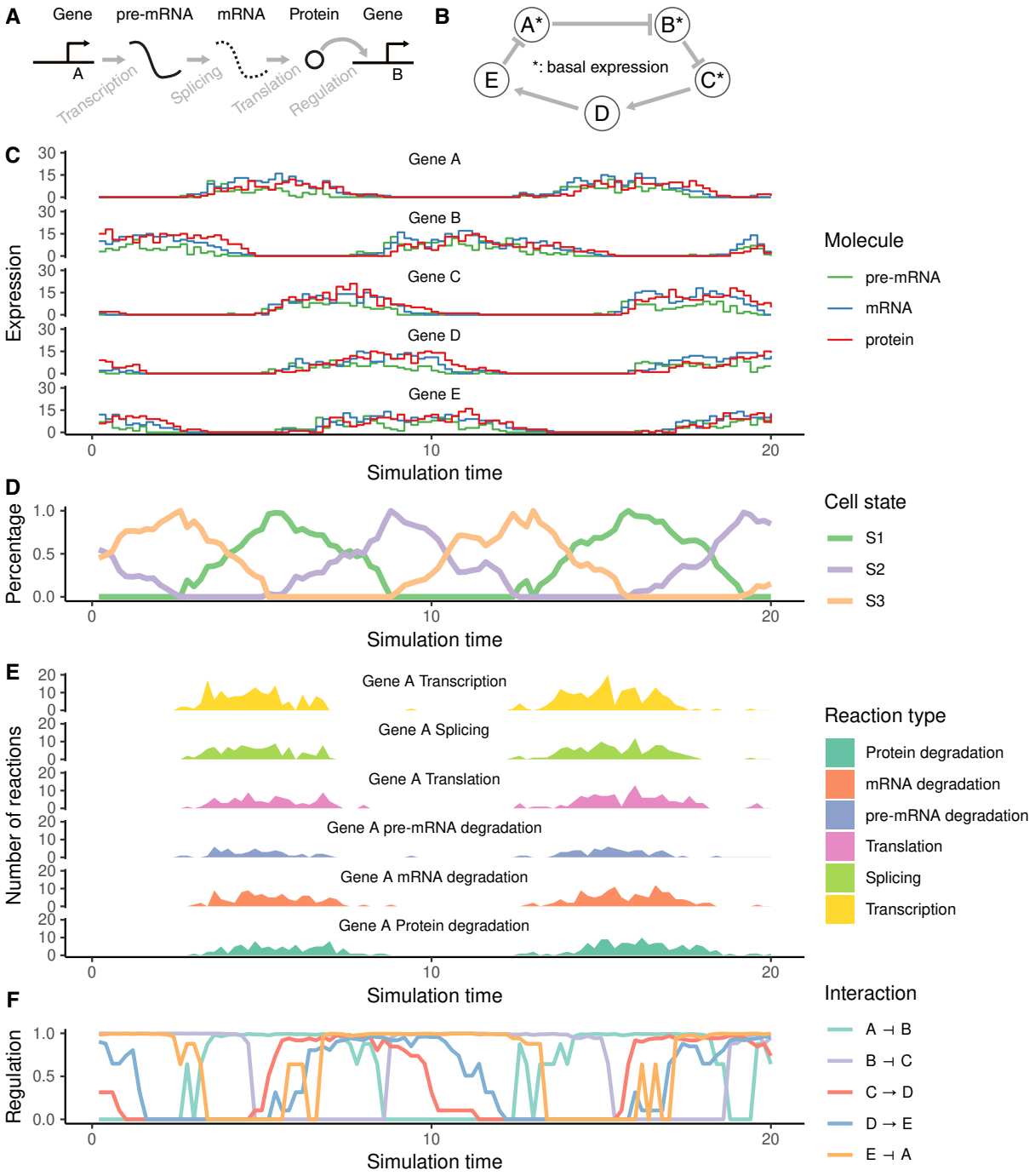

Figure 2: **Key features of dyngen are illustrated using a cyclic toy example.** While this example comprises only of a single cell and 5 genes, dyngen is able to simulate thousands of cells with GRNs containing thousands of genes.

Example use cases

Trajectory alignment

Single-cell network inference

RNA velocity

Discussion

As is, dyngen's single cell simulations can be used to evaluate common single-cell omics computational methods such as clustering, batch correction, trajectory inference and network inference. However, the combined effect of

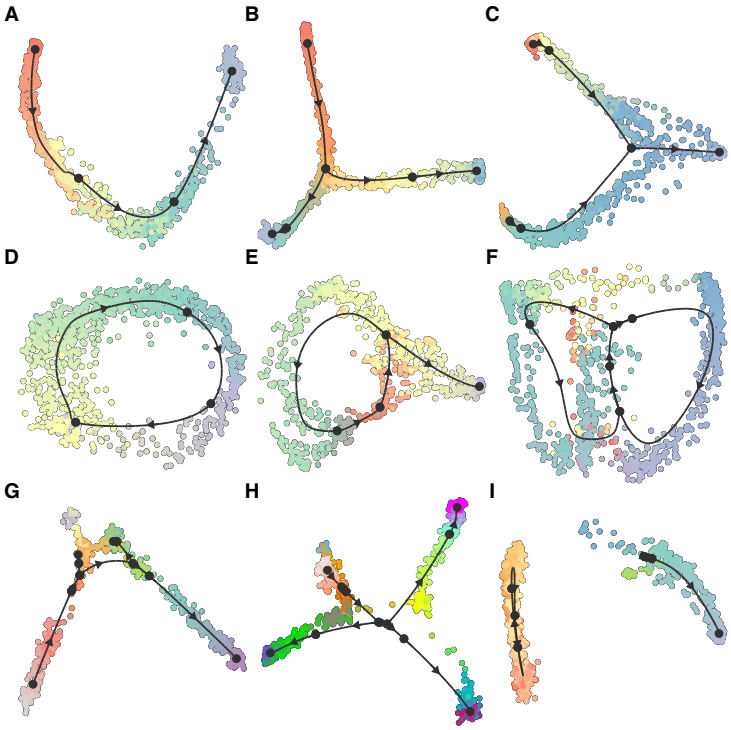

Figure 3: **Multiple executions of dyngen with different predefined backbones.** From each simulation of about 200 genes, 1000 cells were sampled. **A:** Linear. **B:** Bifurcating. **C:** Converging. **D:** Cyclic. **E:** Bifurcating loop. **F:** Bifurcating converging. **G:** Consecutive branching. **H:** Binary tree. **I:** Disconnected.

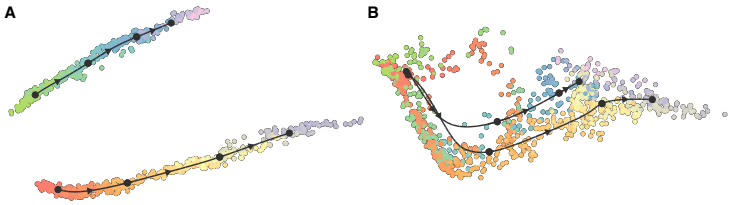

Figure 4: **Examples of simulations with perturbed GRNs. A:** The cells in the top half were simulated with the same GRN as the cells on the bottom half, except that all parameters (e.g. strength of the interaction) were randomised. **B:** Only 10 interactions in the GRN were randomised. In this example, the effect of the GRN perturbation is more subtle.

these advantages results in a framework that is flexible enough to adapt to a broad range of applications. This may include methods that integrate clustering, network inference and trajectory inference. In this respect, dyngen may promote the development of new tools in the single-cell field similarly as other simulators have done in the past[14, 15].

dyngen ultimately allows anticipating technological developments in single-cell multi-omics. In this way, it is possible to design and evaluate the performance and robustness of new types of computational analyses before experimental data becomes available. In addition, it could also be used to compare which experimental protocol is the most cost-effective in producing the qualitative and robust results in downstream analysis.

Currently, dyngen focuses on simulating cells as standalone entities that are well mixed. Splitting up the simulation space into separate subvolumes could pave the way to better study key cellular processes such as cell division, intercellular communication and migration[16].

Methods

The workflow to generate *in silico* single cell data consists of six main steps (Figure 5).

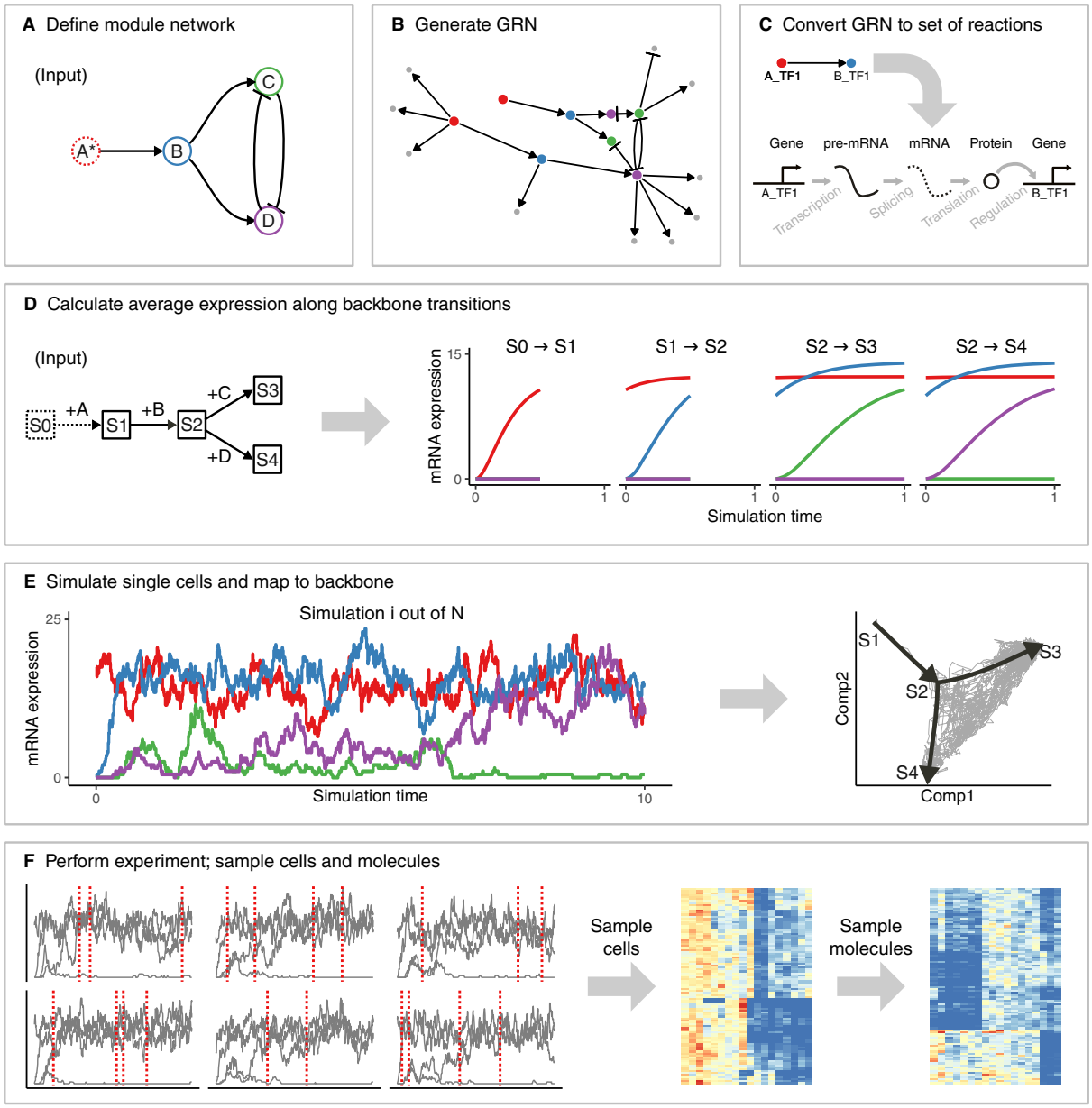

Figure 5: **The workflow of dyngen is comprised of six main steps. A:** The user needs to specify the desired module network or use a predefined module network. **B:** Each gene in a module is is regulated by one or more transcription factors from the upstream module. Additional target genes are generated. **C:** Each gene regulatory interaction in the GRN is converted to a set of biochemical reactions. **D:** Along with the module network, the user also needs to specify the backbone structure of expected cell states. The average expression of each edge in the backbone is simulated by activating a restricted set of genes for each edge. **E:** Multiple Gillespie SSA simulations are run using the reactions defined in step C. The counts of each of the molecules at each time step are extracted. Each time step is mapped to a point in the backbone. **F:** The molecule levels of multiple simulations are shown over time (left). From each simulation, multiple cells are sampled (from left to middle). Technical noise from profiling is simulated by sampling molecules from the set of molecules inside each cell (from middle to right).

Defining the backbone: modules and states

One of the main processes involved in cellular dynamic processes is gene regulation, where regulatory cascades and feedback loops lead to progressive changes in expression and decision making. The exact way a cell chooses a certain path during its differentiation is still an active research field, although certain models have already emerged and been tested *in vivo*. One driver of bifurcation seems to be mutual antagonism, where two genes strongly repress each other[17, 18], forcing one of the two to become inactive[19]. Such mutual antagonism can be modelled and

simulated[20, 21]. Although the two-gene model is simple and elegant, the reality is frequently more complex, with multiple genes (grouped into modules) repressing each other[22].

To start a dyngen simulation, the user needs to define a module network and a backbone. The module network defines how sets of co-regulated genes, called modules, regulate each other. The module network is what mainly determines which dynamic processes occur within the simulated cells. The backbone is a separate set of simulations in which the ground-truth topology of the dynamic processes are defined, as it is difficult to determine the topology of the dynamic processes from the module network itself.

A module network consists of modules connected together by regulatory interactions. A module may have basal expression, which means genes in this module will be transcribed without the presence of transcription factor molecules. A module marked as "active during the burn phase" means that this module will be allowed to generate expression of its genes during an initial warm-up phase (See section). At the end of the dyngen process, cells will not be sampled from the burn phase simulations. Interactions between modules have a strength (which is a positive integer) and an effect (+1 for upregulating, -1 for downregulating).

Several examples of module networks are given (Figure 6). A simple chain of modules (where one module upregulates the next) results in a *linear* process. By having the last module repress the first module, the process becomes *cyclic*. Two modules repressing each other is the basis of a *bifurcating* process, though several chains of modules have to be attached in order to achieve progression before and after the bifurcation process. Finally, a *converging* process has a bifurcation occurring during the burn phase, after which any differences in module regulation is removed.

Note that these examples represent the bare minimum in terms of number of modules used. Using longer chains of modules is typically desired. In addition, the fate decisions made in this example of a bifurcation is reversible, meaning cells can be reprogrammed to go down a different differentiation path. If this effect is undesirable, more safeguards need to be put in place to prevent reprogramming from occurring (Section).

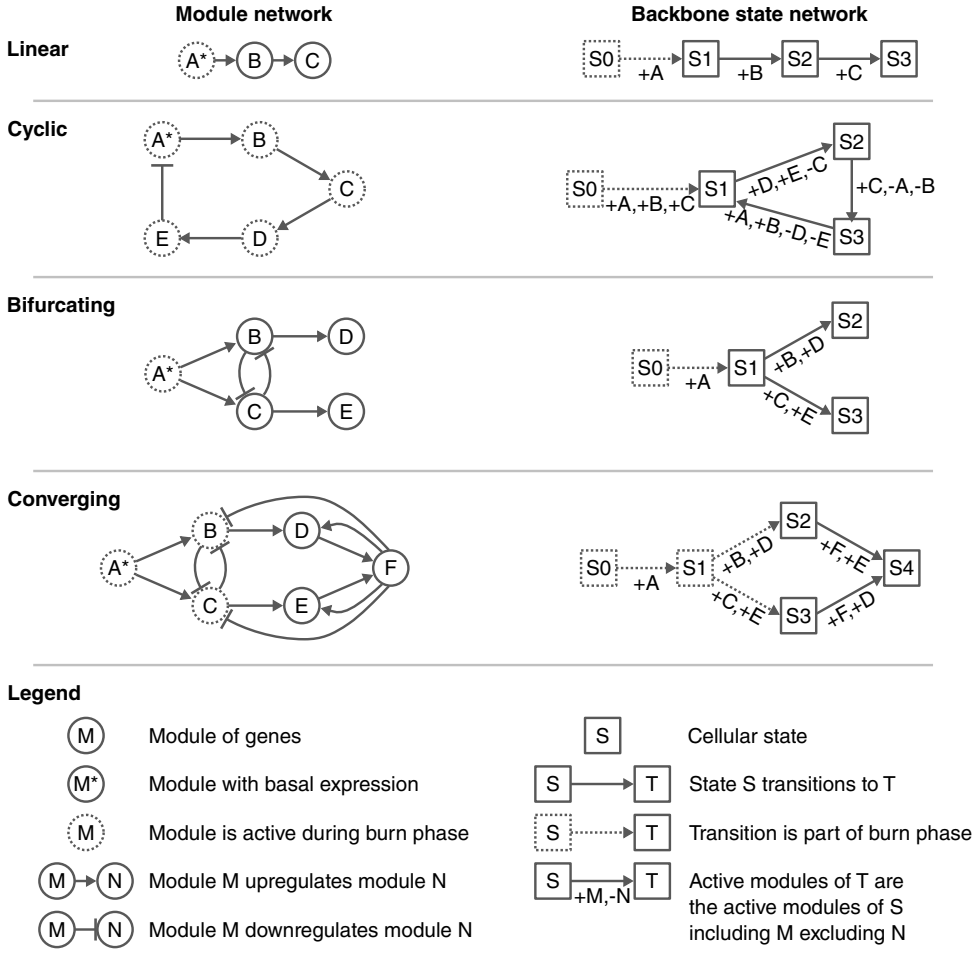

Figure 6: **Example module networks.**

In addition to the module network, the user also needs to define a network of cellular states called the "backbone".

Before simulating any cells, each transition in the backbone is simulated separately to obtain the average changes in expression along that transition (Figure 5D). As part of the backbone, the user needs to specify which modules are allowed to alter its expression from one state to another. For example, in order to transition from state S0 to S1 in the cyclic example, gene modules A, B and C are turned on and a simulation is allowed to run. To transition from S1 to S2, gene modules D and E are turned on, and expression of gene module C is kept constant. To transition from S2 to S3, C is turned on again and now A and B are fixed. Finally, to transition from S3 to S1 again, A and B are turned on again and D and E are fixed again. Demonstrations of the backbone will be explained in more detail in section .

Backbone lego

The backbone can make use of one or more "backbone lego" (BBL) pieces (Figure 7). A BBL consists of one or more modules which regulate each other such that the output modules present a specific behaviour, depending on the input module (Figure 7A). Parameters allow determining the number of modules involved in the process and the number of outputs. Multiple BBLs can be chained together in order to intuitively create module networks and corresponding state networks (Figure 7B). Note that not all dynamic processes can be represented by a combination of BBLs, but they can serve as common building blocks to aid the construction of the backbone.

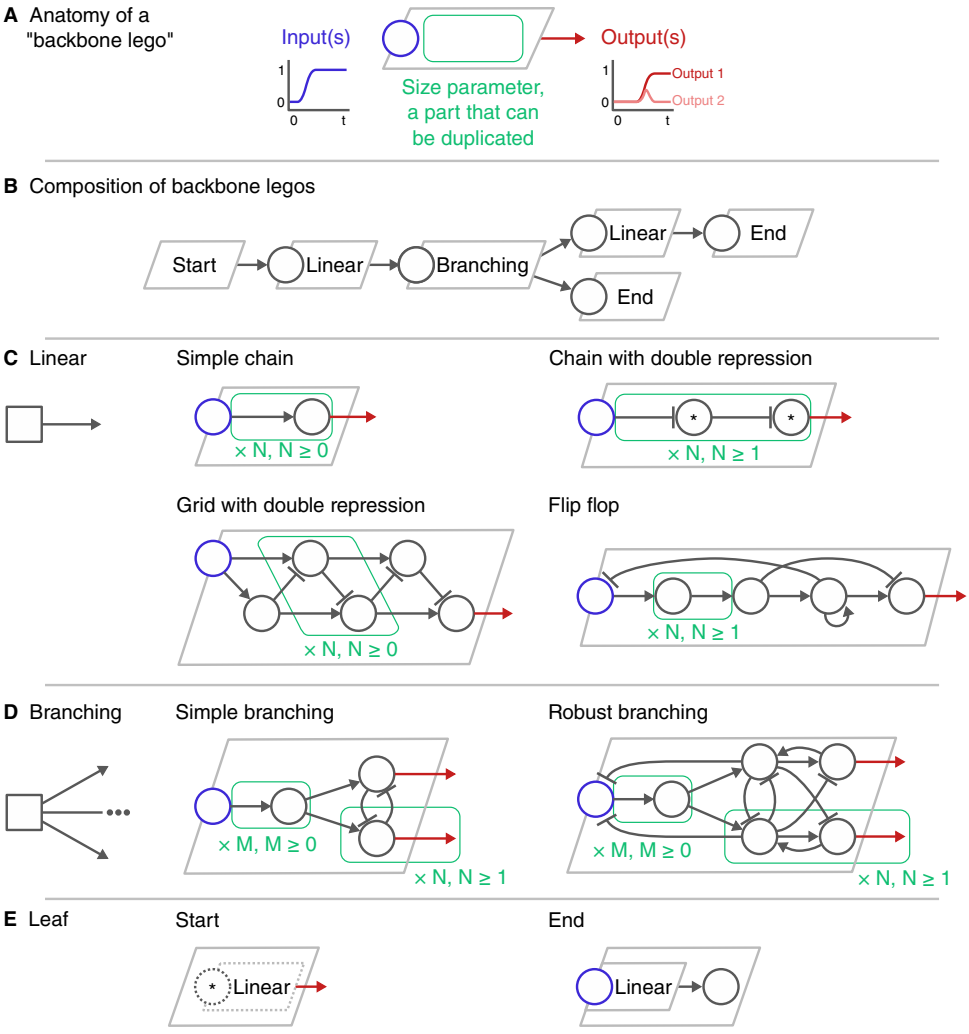

Figure 7: **Backbone lego.**

When the input node of a **linear BBL** (Figure 7C) is upregulated, the module the BBL is connected to will be upregulated. A *simple chain* is a set of modules where a module upregulates the next. A *chain with double repression* has an uneven number of modules forming a chain where each module downregulates the next but all modules (except the input) have basal expression. A *grid with double repression* is similar; except that modules do not have basal expression but instead get upregulated by an upstream module in the chain. Finally, a *flip flop*} consists of a simple chain where first the modules (except the last) are upregulated. Once the second to last module is upregulated, that

module upregulates itself and the first module is strongly repressed, causing all other modules to lose expression and finally the last module to be upregulated. The *flip flop* retains this output state, even when the input changes.

When the input node of a **branching BBL** (Figure 7D) is upregulated, a subset of its output modules will eventually be upregulated. A *simple branching* uses reciprocal inhibition to drive the upregulation of one of the output modules. Due to its simplicity, however, multiple output modules might be upregulated simultaneously and over long periods of simulation time it might be possible that the choice of upregulated module changes. A *robust branching* improves upon the simple branching by preventing upregulation of output modules until an internal branching decision has been made, and by repressing the decision mechanism to avoid other output modules being upregulated other than the one that has been chosen.

A **leaf BBL** (Figure 7E) is a linear BBL that has either no inputs or no outputs. A *start* BBL is a linear BBL where the first module has basal expression, and all modules in this module will be active during the burn-in phase of the simulation (Section). An *end* BBL is also a linear BBL with its output regulating one final module.

Generating the gene regulatory network

The GRN is generated based on the given backbone in four main steps (Figure 8).

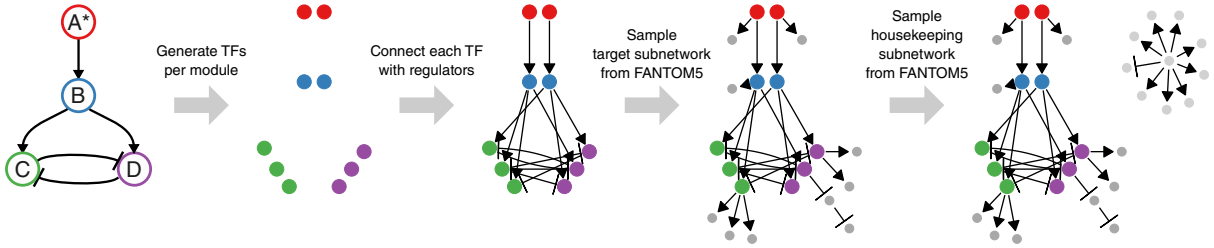

Figure 8: **Generating the feature network from a backbone consists of four main steps.**

Step 1, sampling the transcription factors (TF). The TFs are the main drivers of the molecular changes in the simulation. The user provides a backbone and the number of TFs to generate. Each TF is assigned to a module such that each module has at least x parameters (default $x = 1$). A TF inherits the 'burn' and 'basal expression' from the module it belongs to.

Step 2, generating the TF interactions. Let each TF be regulated according to the interactions in the backbone. These interactions inherit the effect, strength, and cooperativity parameters from the interactions in the backbone. A TF can only be regulated by other TFs or itself.

Step 3, sampling the target subnetwork. A user-defined number of target genes are added to the GRN. Target genes are regulated by a TF or another target gene, but is always downstream of at least one TF. To sample the interactions between target genes, one of the many FANTOM5 [23] GRNs is sampled. The currently existing TFs are mapped to regulators in the FANTOM5 GRN. The targets are drawn from the FANTOM5 GRN, weighted by their page rank value. For each target, at most x regulators are sampled from the induced FANTOM5 GRN (default $x = 5$). The interactions connecting a target gene and its regulators are added the GRN.

Step 4, sampling the housekeeping subnetwork. Housekeeping genes are completely separate from any TFs or target genes. A user-defined set of housekeeping genes are also sampled from the FANTOM5 GRN. The interactions of the FANTOM5 GRN are first subsampled such that the maximum in-degree of each gene is x (default $x = 5$). A random gene is sampled and a breadth-first-search is performed to sample the desired number of housekeeping genes.

Convert gene regulatory network to a set of reactions

Simulating a cell's GRN makes use of a stochastic framework which tracks the abundance levels of molecules over time in a discrete quantity. For every gene G, the abundance levels of three molecules are tracked, namely of corresponding pre-mRNAs, mature mRNAs and proteins, which are represented by the terms w_G, x_G and y_G respectively. The GRN defines how a reaction affects the abundance levels of molecules and how likely it will occur. Gibson and Bruck[24] provide a good introduction to modelling gene regulation with stochastic frameworks, on which many of the concepts below are based.

For every gene in the GRN a set of reactions are defined, namely transcription, splicing, translation, and degradation. Each reaction consists of a propensity function – a formula $f(.)$ to calculate the probability $f(.) \times \mathrm{d}t$ of it occurring during a time interval $\mathrm{d}t$ – and the effect – how it will affect the current state if triggered.

The effects of each reaction mimic the respective biological processes (Table 1, middle). Transcription of gene G results in the creation of a single pre-mRNA molecule w_G. Splicing turns one pre-mRNA w_G into a mature mRNA w_G. Translation uses a mature mRNA x_G to produce a protein y_G. Pre-mRNA, mRNA and protein degradation results in the removal of a w_G, x_G, and y_G molecule, respectively.

The propensity of all reactions except transcription are all linear functions (Table 1, right) of the abundance level of some molecule multiplied by a constant drawn from a normal distribution (Table 2). The propensity of transcription of a gene G depends on the abundance levels of its TFs.

In the most simple case, G is regulated by a single TF H, which can either be bound (state S_0) or unbound (state S_1). The probability $P(S_1)$ that H is bound to the promoter region of G is given by the abundance levels of H.

Table 1: **Reactions affecting the abundance levels of pre-mRNA w_G, mature mRNA x_G and proteins y_G of gene G.** Define the set of regulators of G as R_G, the set of upregulating regulators of G as R_G^+, and the set of downregulating regulators of G as R_G^-. Parameters used in the propensity formulae are defined in Table 2.

Reaction	Effect	Propensity
Transcription	$\emptyset \to \mathrm{w}_G$	$\mathrm{wpr}_G \times \frac{\mathrm{ba}_G - \mathrm{co}_G^{\lvert \mathrm{R}_G^+ \rvert} + \prod_{H \in \mathrm{R}_G^+} (\mathrm{co}_G + \chi_{G,H})}{\prod_{H \in \mathrm{R}_G} (1 + \chi_{G,H})}$
Pre-mRNA degradation	$\mathrm{w}_G \to \emptyset$	$\mathrm{wdr}_G \times \mathrm{w}_G$
Splicing	$\mathrm{w}_G \to \mathrm{x}_G$	$\mathrm{wsr}_G \times \mathrm{w}_G$
Mature mRNA degradation	$\mathrm{x}_G \to \emptyset$	$\mathrm{xdr}_G \times \mathrm{x}_G$
Translation	$\mathrm{x}_G \to \mathrm{x}_G + \mathrm{y}_G$	$\mathrm{ypr}_G \times \mathrm{x}_G$
Protein degradation	$\mathrm{y}_G \to \emptyset$	$\mathrm{ydr}_G \times \mathrm{y}_G$

Table 2: **Default parameters defined for the calculation of reaction propensity functions.**

Parameter	Symbol	Definition
Transcription rate	wpr_G	$\in N(50, 10),\ \geq 10$
Splicing rate	wsr_G	$\in N(5, 1),\ \geq 1$
Translation rate	ypr_G	$\in N(5, 1),\ \geq 1$
Pre-mRNA half-life	whl_G	$\in N(0.15, 0.03),\ \geq 0.05$
Mature mRNA half-life	xhl_G	$\in N(0.15, 0.03),\ \geq 0.05$
Protein half-life rate	yhl_G	$\in N(0.25, 0.05),\ \geq 0.1$
Interaction strength	$\mathrm{str}_{G,H}$	$\in 10^{U(0,2)}$ *
Hill coefficient	$\mathrm{hill}_{G,H}$	$\in U(0.5, 2)$ *
Cooperativity factor	co_G	$\in [0, 1]$ *
Pre-mRNA degradation rate	wdr_G	$= \ln(2) \,/\, \mathrm{whl}_G$
Mature mRNA degradation rate	xdr_G	$= \ln(2) \,/\, \mathrm{xhl}_G$
Protein degradation rate	ydr_G	$= \ln(2) \,/\, \mathrm{yhl}_G$
Dissociation constant	dis_H	$= 0.5 \times \frac{\mathrm{wpr}_H \times \mathrm{ypr}_H}{\mathrm{wdr}_H \times \mathrm{xdr}_H \times \mathrm{ydr}_H}$
Binding	$\chi_{G,H}$	$= (\mathrm{str}_{G,H} \times \mathrm{y}_H \,/\, \mathrm{dis}_H)^{\mathrm{hill}_{G,H}}$
Basal expression	ba_G	$= \begin{cases} 1 & \text{if } \mathrm{R}_G^+ = \emptyset \\ 0.0001 & \text{if } \mathrm{R}_G^- = \emptyset \text{ and } \mathrm{R}_G^+ \neq \emptyset \text{ *} \\ 0.5 & \text{otherwise} \end{cases}$

*: unless G is a TF, then the value is determined by the backbone.

Compute average expression along backbone transitions

When simulating the developmental backbone, we go through the edges of the backbone state network defined in an earlier step (Section), starting from the root state. It is assumed the root state has no modules active and has

no expression of any molecules. To get to the next state, we follow a transition starting from the root state, activate and deactivate the modules as indicated by the transition, and compute the average molecule abundance along the transition. To compute the average abundance, we perform small time steps $t = 0.001$ and let each reaction (Section) occur t times its propensity.

Simulate single cells

dyngen uses Gillespie's stochastic simulation algorithm (SSA)[13] to simulate dynamic processes. An SSA simulation is an iterative process where at each iteration one reaction is triggered.

Each reaction consists of its propensity – a formula to calculate the probability of the reaction occurring during an infinitesimal time interval – and the effect – how it will affect the current state if triggered. Each time a reaction is triggered, the simulation time is incremented by $\tau = \frac{1}{\sum_j prop_j} \ln\left(\frac{1}{r}\right)$, with $r \in U(0, 1)$ and $prop_j$ the propensity value of the jth reaction for the current state of the simulation.

GillespieSSA2 is an optimised library for performing SSA simulations. The propensity functions are compiled to C++ and SSA approximations can be used which allow to trigger many reactions simultaneously at each iteration. The framework also allows to store the abundance levels of molecules only after a specific interval has passed since the previous census. By setting the census interval to 0, the whole simulation's trajectory is retained but many of these time points will contain very similar information. In addition to the abundance levels, also the propensity values and the number of firings of each of the reactions at each of the time steps can be retained, as well as specific sub-calculations of the propensity values, such as the regulator activity level $reg_{G,H}$.

Map SSA simulations to backbone

We compute the Pearson correlation between the state vectors in the simulation and the average expression levels along a transition in the backbone. Each timepoint in the SSA simulation is mapped to the point in the backbone that has the highest correlation value.

Simulate experiment

From the SSA simulation we obtain the abundance levels of all the molecules at every state. We need to replicate technical effects introduced by experimental protocols in order to obtain data that is similar to real data. For this, the cells are sampled from the simulations and molecules are sampled for each of the cells. Real datasets are used in order to achieve similar data characteristics.

Sample cells

In this step, N cells are sampled the simulations. Two approaches are implemented: sampling from an unsynchronised population of single cells (snapshot) or sampling at multiple time points in a synchronised population (time series).

Snapshot The backbone consists of several states linked together by transition edges with length L_i, to which the different states in the different simulations have been mapped (Figure 9A). From each transition, $N_i = N / \frac{L_i}{\sum L_i}$ cells are sampled uniformly, rounded such that $\sum N_i = N$.

Time series Assuming that the final time of the simulations is T, the interval $[0, T]$ is divided into k equal intervals of width w separated by $k - 1$ gaps of width g. $N_i = N/k$ cells are sampled uniformly from each interval (Figure 9B), rounded such that $\sum N_i = N$. By default, $k = 8$ and $g = 0.75$. For usual dyngen simulations, $10 \leq T \leq 20$. For larger values of T, k and g should be increased accordingly.

Sample molecules

Molecules are sampled from the simulation to replicate how molecules are experimentally sampled. A real dataset is downloaded from a repository of single-cell RNA-seq datasets[25]. For each *in silico* cell i, draw its library size ls_i from

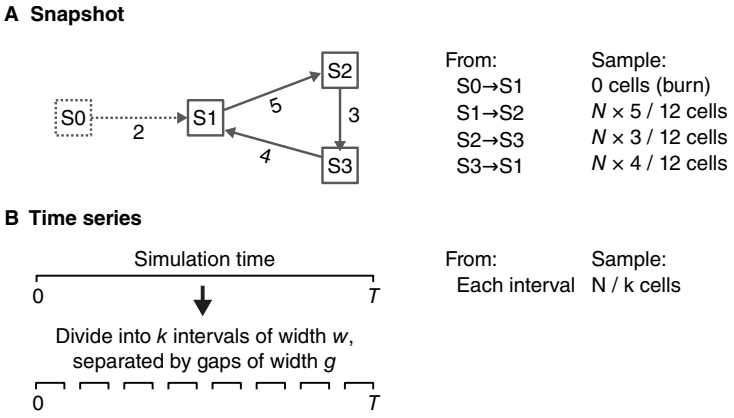

Figure 9: **Two approaches can be used to sample cells from simulations: snapshot and time-series.**

the distribution of transcript counts per cell in the real dataset. The capture rate cr_j of each *in silico* molecule type j is drawn from $N(1, 0.05)$. Finally, for each cell i, draw ls_i molecules from the multinomial distribution with probabilities $cr_j \times ab_{i,j}$ with $ab_{i,j}$ the molecule abundance level of molecule j in cell i.

Determining the casewise ground-truth regulatory network

Calculating the regulatory effect of a regulator R on a target T (Figure 5F) requires determining the contribution of R in the propensity function of the transcription of T (section) with respect to other regulators. This information is useful, amongst others, for benchmarking casewise network inference methods.

The regulatory effect of R on T at a particular state S is defined as the change in the propensity of transcription when R is set to zero, scaled by the inverse of the pre-mRNA production rate of T. More formally:

$$\text{regeffect}_G = \frac{\text{proptrans}_G(S) - \text{proptrans}_G(S[\text{y}_T \leftarrow 0])}{\text{wpr}_G}$$

Determining the regulatory effect for all interactions and cells in the dataset yields the complete casewise ground-truth GRN (Figure 10). The regulatory effect lie between $[-1, 1]$, where -1 represents complete inhibition of T by R, 1 represents maximal activation of T by R, and 0 represents inactivity of the regulatory interaction between R and T.

Comparison of casewise network inference methods

Several datasets were generated using the different predefined backbones. For every cell in the dataset, the transcriptomics profile and the corresponding casewise ground-truth regulatory network was determined (Section).

Several casewise NI methods were considered for comparison: SCENIC[26], LIONESS[27, 28], and SSN[29].

LIONESS[27, 28] computes the Pearson correlation between a regulator and one of its targets for all cells. The possible targets for a given regulator is predefined according to prior motif analysis. To derive casewise GRNs for a particular cell C, the Pearson correlation is recomputed for all cells excluding C. The difference in Pearson correlations results in a ranking of regulatory interactions for C.

SSN[29] TODO: wip.

SCENIC consists of four main steps: classical network inference with arboreto (similar to GENIE3), selection of 'regulons' from the GRN, filtering of the regulons using motif analysis, and casewise scoring of each regulon for each cell using AUCell.

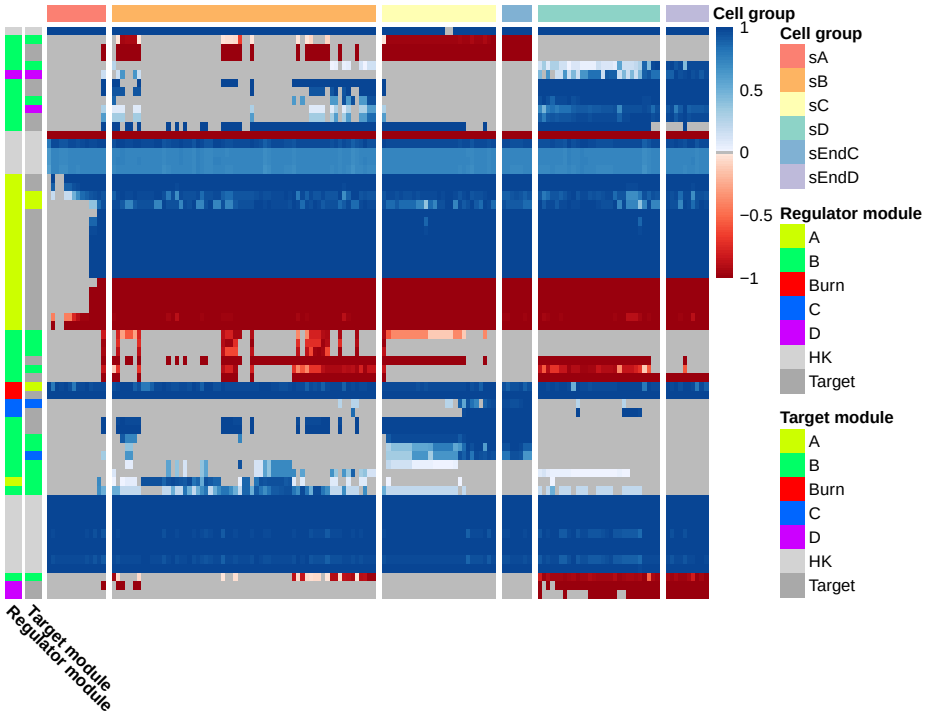

Figure 10: **The casewise regulatory effects of all interactions, computed on cells part of a bifurcation trajectory.** Negative values correspond to inhibitory interactions, positive values to activating interactions, and zero values correspond to inactive interactions.

References

[1] Luke Zappia, Belinda Phipson, and Alicia Oshlack. "Splatter: Simulation of Single-Cell RNA Sequencing Data". In: *Genome Biology* 18 (Sept. 2017), p. 174. ISSN: 1474-760X. DOI: 10.1186/s13059-017-1305-0.

[2] Beate Vieth et al. "powsimR: Power Analysis for Bulk and Single Cell RNA-Seq Experiments". In: *Bioinformatics* 33.21 (Nov. 1, 2017), pp. 3486–3488. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btx435.

[3] Nikolaos Papadopoulos, Rodrigo Gonzalo Parra, and Johannes Soeding. "PROSSTT: Probabilistic Simulation of Single-Cell RNA-Seq Data for Complex Differentiation Processes". In: *bioRxiv* (Jan. 2018), p. 256941. DOI: 10.1101/256941.

[4] Xiuwei Zhang, Chenling Xu, and Nir Yosef. "Simulating Multiple Faceted Variability in Single Cell RNA Sequencing". In: *Nature Communications* 10.1 (June 13, 2019), pp. 1–16. ISSN: 2041-1723. DOI: 10.1038/s41467-019-10500-w.

[5] Kelly Street et al. "Slingshot: Cell Lineage and Pseudotime Inference for Single-Cell Transcriptomics". In: *BMC Genomics* 19.1 (June 2018), p. 477. ISSN: 1471-2164. DOI: 10.1186/s12864-018-4772-0.

[6] R Gonzalo Parra et al. "Reconstructing Complex Lineage Trees from scRNA-Seq Data Using MERLoT". In: *bioRxiv* (Feb. 2018), p. 261768. DOI: 10.1101/261768.

[7] Edroaldo Lummertz da Rocha et al. "Reconstruction of Complex Single-Cell Trajectories Using CellRouter". In: *Nature Communications* 9.1 (Mar. 1, 2018), p. 892. ISSN: 2041-1723. DOI: 10.1038/s41467-018-03214-y.

[8] Yingxin Lin et al. "scClassify: Hierarchical Classification of Cells". In: *bioRxiv* (Jan. 1, 2019), p. 776948. DOI: 10.1101/776948.

[9] Angelo Duò, Mark D. Robinson, and Charlotte Soneson. "A Systematic Performance Evaluation of Clustering Methods for Single-Cell RNA-Seq Data". In: *F1000Research* 7 (2018), p. 1141. ISSN: 2046-1402. DOI: 10.12688/f1000research.15666.2. pmid: 30271584.

[10] Wouter Saelens et al. "A Comparison of Single-Cell Trajectory Inference Methods". In: *Nature Biotechnology* 37 (May 2019). ISSN: 15461696. DOI: 10.1038/s41587-019-0071-9.

[11] Charlotte Soneson and Mark D. Robinson. "Bias, Robustness and Scalability in Single-Cell Differential Expression Analysis". In: *Nature Methods* 15.4 (Apr. 2018), pp. 255–261. ISSN: 1548-7105. DOI: 10.1038/nmeth.4612. pmid: 29481549.

[12] Tim Stuart and Rahul Satija. "Integrative Single-Cell Analysis". In: *Nature Reviews Genetics* 20.5 (May 2019), pp. 257–272. ISSN: 1471-0064. DOI: 10.1038/s41576-019-0093-7.

[13] Daniel T. Gillespie. "Exact Stochastic Simulation of Coupled Chemical Reactions". In: *The Journal of Physical Chemistry* 81.25 (Dec. 1, 1977), pp. 2340–2361. ISSN: 0022-3654. DOI: 10.1021/j100540a008.

[14] Thomas Schaffter, Daniel Marbach, and Dario Floreano. "GeneNetWeaver: In Silico Benchmark Generation and Performance Profiling of Network Inference Methods." In: *Bioinformatics* 27.16 (Aug. 2011), pp. 2263–2270. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btr373. pmid: 21697125.

[15] Adam D. Ewing et al. "Combining Tumor Genome Simulation with Crowdsourcing to Benchmark Somatic Single-Nucleotide-Variant Detection". In: *Nature Methods* 12.7 (July 2015), pp. 623–630. ISSN: 1548-7105. DOI: 10.1038/nmeth.3407.

[16] Stephen Smith and Ramon Grima. "Spatial Stochastic Intracellular Kinetics: A Review of Modelling Approaches". In: *Bulletin of Mathematical Biology* 81.8 (Aug. 1, 2019), pp. 2960–3009. ISSN: 1522-9602. DOI: 10.1007/s11538-018-0443-1.

[17] N. Rekhtman et al. "Direct Interaction of Hematopoietic Transcription Factors PU.1 and GATA-1: Functional Antagonism in Erythroid Cells". In: *Genes & Development* 13.11 (June 1, 1999), pp. 1398–1411. ISSN: 0890-9369. DOI: 10.1101/gad.13.11.1398. pmid: 10364157.

[18] Heping Xu et al. "Regulation of Bifurcating {B} Cell Trajectories by Mutual Antagonism between Transcription Factors {IRF4} and {IRF8}". In: *Nat. Immunol.* 16.12 (Dec. 2015), pp. 1274–1281.

[19] Thomas Graf and Tariq Enver. "Forcing Cells to Change Lineages". In: *Nature* 462.7273 (Dec. 2009), p. 587. ISSN: 1476-4687. DOI: 10.1038/nature08533.

[20] Jin Wang et al. "Quantifying the Waddington Landscape and Biological Paths for Development and Differentiation". In: *Proceedings of the National Academy of Sciences* 108.20 (May 2011), pp. 8257–8262. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1017017108. pmid: 21536909.

[21] James E Ferrell. "Bistability, Bifurcations, and Waddington's Epigenetic Landscape". In: *Current Biology* 22.11 (June 2012), R458–R466. ISSN: 0960-9822. DOI: 10.1016/j.cub.2012.03.045.

[22] Nir Yosef et al. "Dynamic Regulatory Network Controlling {TH17} Cell Differentiation". In: *Nature* 496.7446 (2013), pp. 461–468.

[23] Marina Lizio et al. "Gateways to the FANTOM5 Promoter Level Mammalian Expression Atlas". In: *Genome Biology* 16.1 (Jan. 5, 2015), p. 22. ISSN: 1465-6906. DOI: 10.1186/s13059-014-0560-6.

[24] Michael A. Gibson and Jehoshua Bruck. "A Probabilistic Model of a Prokaryotic Gene and Its Regulation". In: *Computational Methods in Molecular Biology: From Genotype to Phenotype, MIT press, Cambridge* (2000).

[25] Robrecht Cannoodt et al. "Single-Cell -Omics Datasets Containing a Trajectory". In: *Zenodo* (Oct. 2018). DOI: 10.5281/zenodo.1211532.

[26] Sara Aibar et al. "SCENIC: Single-Cell Regulatory Network Inference and Clustering". In: *Nature Methods* (Oct. 2017). ISSN: 1548-7091. DOI: 10.1038/nmeth.4463.

[27] Marieke Lydia Kuijjer et al. "Estimating Sample-Specific Regulatory Networks". In: (2015), pp. 1–19. URL: http://arxiv.org/abs/1505.06440.

[28] Marieke Lydia Kuijjer et al. "Estimating Sample-Specific Regulatory Networks". In: *iScience* 14 (Mar. 28, 2019), pp. 226–240. ISSN: 2589-0042. DOI: 10.1016/j.isci.2019.03.021. pmid: 30981959.

[29] Xiaoping Liu et al. "Personalized Characterization of Diseases Using Sample-Specific Networks". In: *Nucleic Acids Research* 44.22 (2016), e164–e164. ISSN: 0305-1048. DOI: 10.1093/nar/gkw772. pmid: 27596597.