dyngen: simulating developing single cells

The *dynverse* guys

September 3, 2019

Abstract:

1 Introduction

Continuous technological advancements to high-throughput profiling of single cells are having profound effects on how researchers can validate biological hypotheses. For example, single-cell RNA sequencing (scRNA-seq) directly resulted in the development of a new type of computational method called trajectory inference (TI). By profiling the transcriptomics profiles of developing cells, TI methods attempt to reconstruct and characterise the underlying dynamic processes [1]. While early experimental technologies allowed to profile one single modality (e.g. DNA sequence, RNA or protein expression), recent developments permit profiling multiple modalities simultaneously.

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 [2]. While this falls outside the reach of current experimental technologies, *in silico* simulations of single cells would allow developing the next wave of computational techniques in anticipation of new experimental technologies.

A few generators of scRNA-seq profiles have already been developed (e.g. splatter [3], powsimR [4] and PROSSTT [5]). These can be used to evaluate the performance of computational tools, and to explore their strengths and weaknesses. A limitation of directly simulating a scRNA-seq profile (instead of a single cell) is that extending the simulation to other aspects of the cell – such as tracking the full history of molecular states – becomes difficult.

We introduce dyngen, a multi-modality simulator of single cells (Figure 1). dyngen was initially developed as part of a comprehensive benchmark of TI methods [6] but has since been extended to be applicable in a much broader context.

Inspiration was drawn from GeneNetWeaver's [7] workflow to generate *in silico* bulk profiles for evaluating network inference methods [8]. GeneNetWeaver translates a known gene regulatory network (GRN) into sets of Ordinary Differential Equations (ODE) in order to simulate .... There are two significant differences in dyngen in comparison to GeneNetWeaver. dyngen simulates a single-cell instead of a bulk profile. However, taking random walks through ODE systems in which key molecules occur with low abundance can result in inaccurate results. Gillespie's Stochastic Simulation Algorithm (SSA) [9] allows to simulate low

abundance systems. Running a simulation with SSA is an iterative process where at each iteration one reaction is allowed to take place. Each reaction consists of its propensity – a formula to calculate the probability of occurring during an infinitesimal time interval – and the effect – how it will affect the current state if triggered. Secondly, the GRN is constructed such that it mimics a dynamic process of interest, such as cell differentiation into multiple cell types.

We demonstrate dyngen's flexibility by simulating numerous different types of biological experiments, and using these simulations to develop new benchmarking techniques for computational tools.

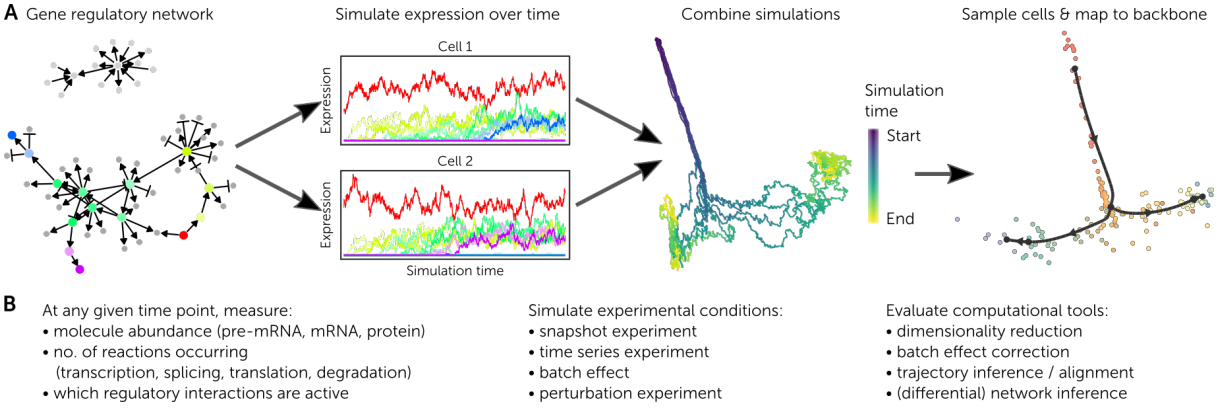

Figure 1: Showcase of dyngen functionality. **TODO: change to pdf.** Remove B?

2 Results

dyngen is a simulator for single cells that develop over time. Throughout this section, a simple simulation of a cell undergoing a cyclic process is used to illustrate key strengths of dyngen (Figure 2). This example only comprises of a single cell containing 5 genes, but dyngen can easily scale up to thousands of simulations containing thousands of genes.

In dyngen, 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). These reactions are determined from a predefined set of gene regulatory interactions (Figure 2B), henceforth referred to as a gene regulatory network (GRN). The likelihood of a reaction occurring at any given point in time is defined by the GRN and by the abundance of molecules involved each reaction.

One of dyngen's main advantages is that through careful engineering of the GRN, different cellular developmental processes can be obtained. Different GRNs can result in branching, converging, cyclic, or even disconnected developmental topologies. Multiple simulations with slightly different GRNs can emulate rewiring events in disease or perturbation experiments. Multiple simulations with different initial molecule abundance levels can be used to replicate batch effects.

Another advantage is that dyngen returns many modalities throughout the whole simulation: molecular abundance, cellular state, number of reaction firings, reaction likelihoods,

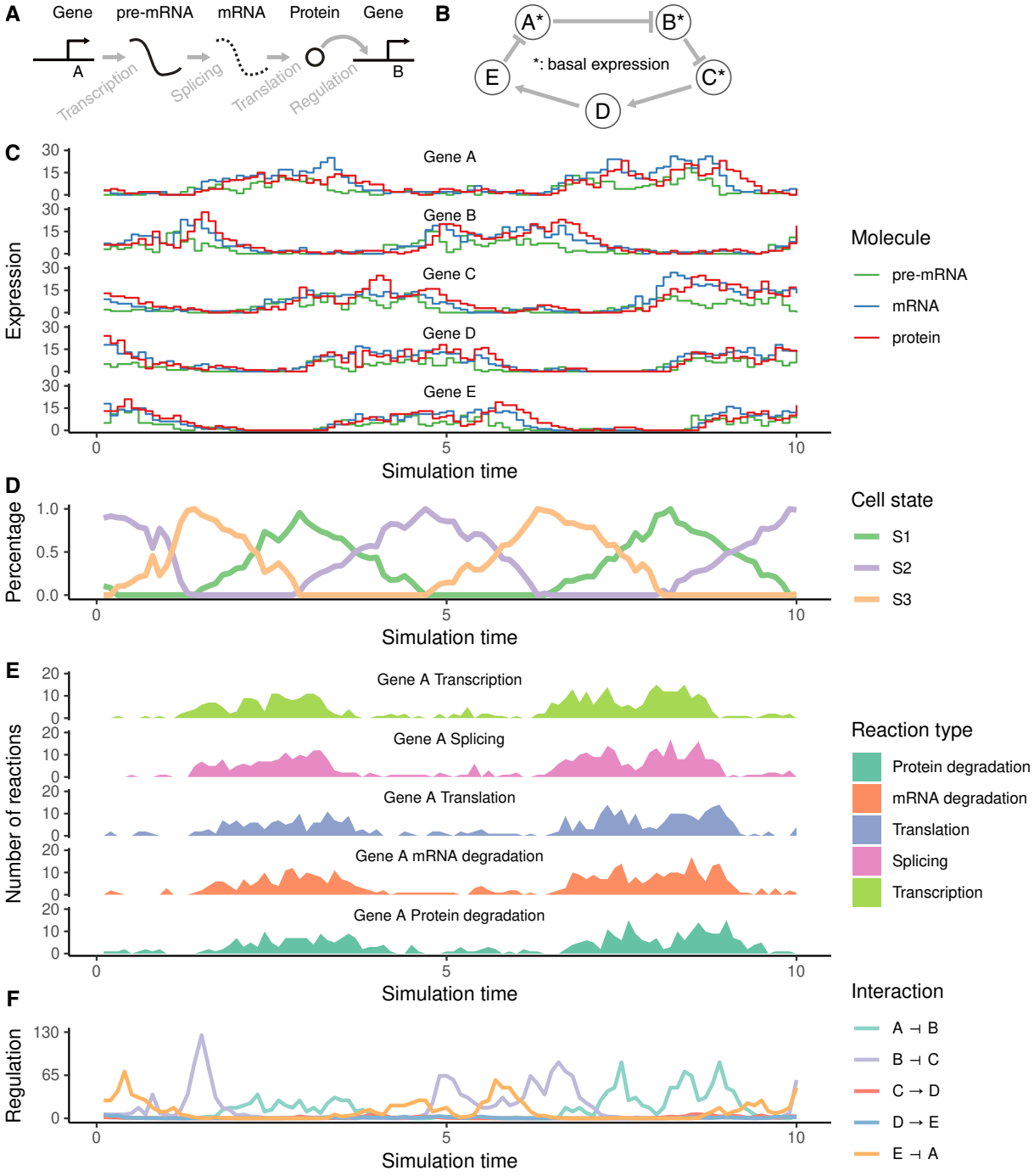

Figure 2: Showcase of dyngen functionality. A time resolution of 0.1 was used, but this can be increased or decreased without effect on performance of the execution of the simulation. **TODO: perhaps it's better to replace Figure 2 with one subfigure for each of the paragraphs in this text.**

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.

The final main advantage is that by making alterations to the simulation pipeline, multiple types of experiments (sampling technique or profiling technique) can be simulated. 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). It is possible to implement other experimental protocols (which perhaps do not exist in real life), such as sampling the same cell at regular intervals.

3 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 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 [7, 10].

Adding batch effects to snapshot simulations of linear (or even branching) trajectories allows evaluating trajectory alignment methods – which attempt to map two or more trajectories onto each other. Adding perturbations to the GRN allows evaluating the performance of differential network inference methods – which predict differential regulatory interactions between two or more groups of profiles. Sampling a cell at a particular time point and once more at a later time point allows 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.

dyngen ultimately also 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. Similarly, it could also be used to compare which experimental technique will likely produce the most accurate result. For example, is it possible to infer directionality of regulatory interactions from snapshot experiments only, or are time series or knockdown experiments a necessity in order to infer high-quality regulatory networks?

Currently, dyngen focuses on simulating cells as standalone entities. Future developments include extending the framework to simulate multiple cells in a virtual environment. Allowing cells to receive and react to environmental and intercellular stimuli would enable simulating essential cellular processes such as cell division and migration.

4 Methods

NOTE: The method section is really rough at this stage.

The workflow to generate *in silico* single cell data consists of six main steps (Figure 3).

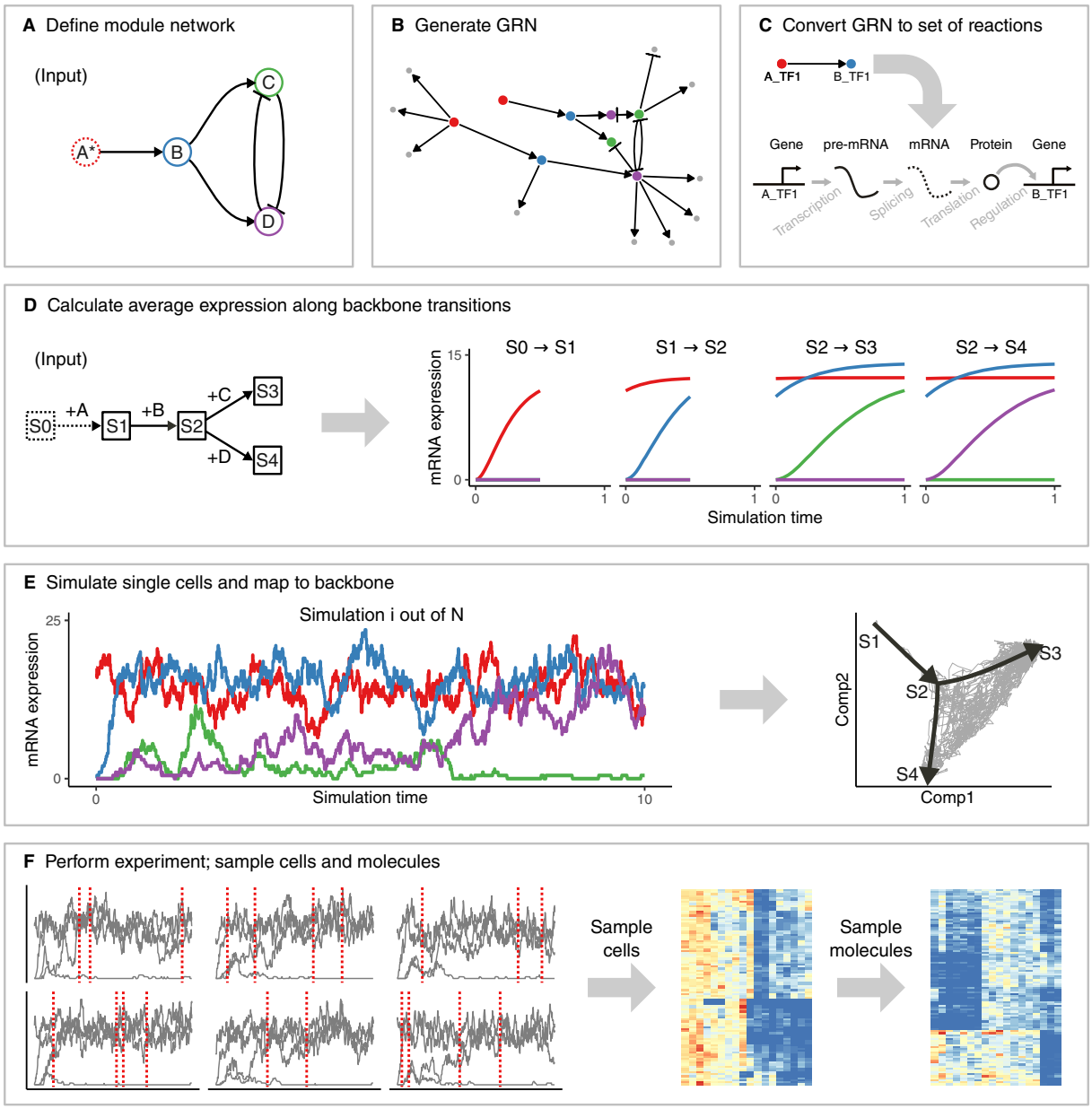

Figure 3: The workflow of dyngen is comprised of six main steps. A: The user needs to specify the desired module network or use a predefined module network. **B:** Each gene in a module is is regulated by one or more transcription factors from the upstream module. Additional target genes are generated. **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 mocules at each time step are extracted. Each time step is mapped to a point in the backbone. **F:** Multiple cells are sampled from each simulation. Molecules are sampled from each cell.

4.1 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 [11] strongly repress each other, forcing one of the two to become inactive [12]. Such mutual antagonism can be modelled and simulated [13, 14]. Although the two-gene model is simple and elegant, the reality is frequently more complex, with multiple genes (grouped into modules) repressing each other [15].

In dyngen, the user defines the behaviour of the simulation by defining how sets of genes, called modules, are regulating eachother. A module may have basal expression, which means that pre-mRNA of the 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 4.5). At the end of the dyngen process, cells will not be sampled from the burn phase simulations.

Several examples of module networks are given (Figure 4). 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 eachother 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 4.1.1).

TODO: mention strength and cooperativity.

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 3D). 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 4.4.

4.1.1 Backbone lego

The backbone can make use of one or more "backbone lego" (BBL) pieces (Figure 5). 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 5A). Parameters allow

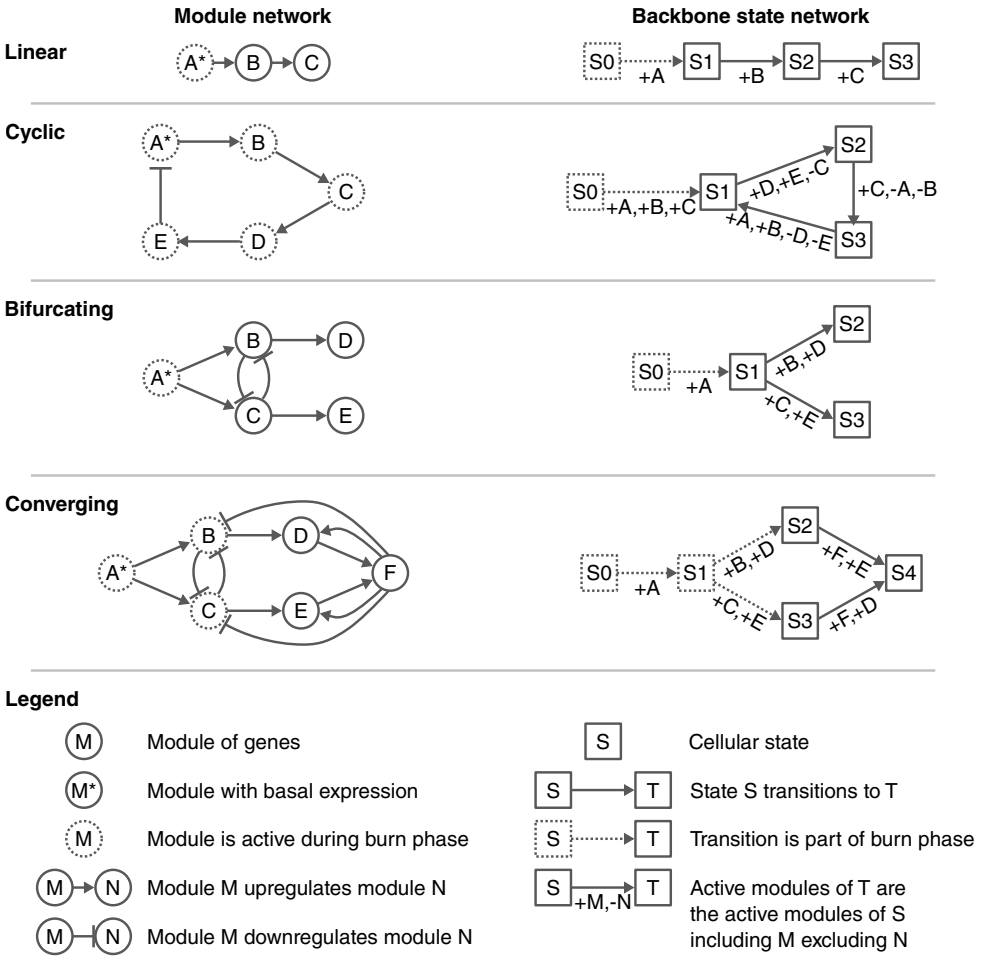

Figure 4: Example module networks

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 milestone networks and corresponding state networks (Figure 5B). 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.

When the input node of a **linear BBL** (Figure 5C) 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 5D) is upregulated, a subset of its output modules will eventually be upregulated. A *simple branching* uses reciprocal inhibition to

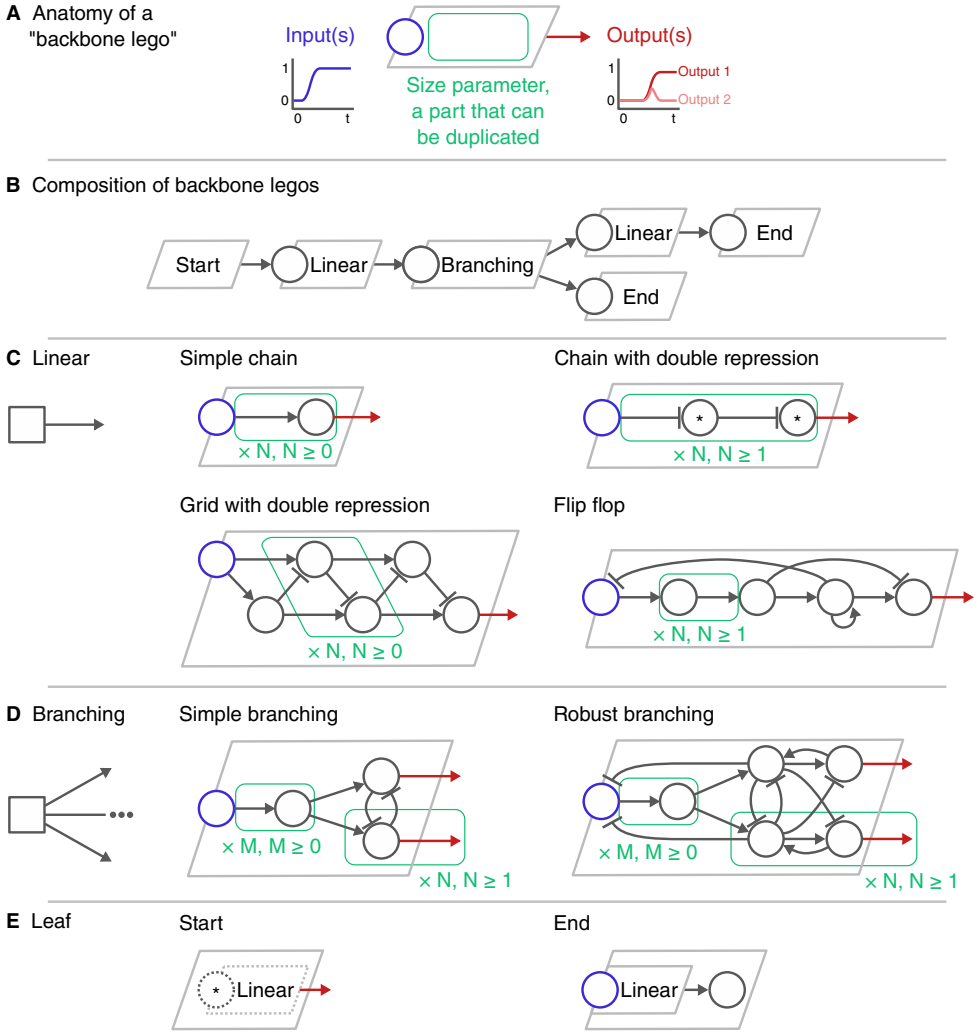

Figure 5: Backbone lego

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 5E) 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 4.4). An *end* BBL is also a linear BBL with its output regulating one final module.

4.2 Generate gene regulatory network

4.2.1 Generate the transcription factor network

Parameters:

- Number of TFs to generate num_tfs

- Minimum TFs per module min_tfs, default = 1
- Number of regulatory interactions per module interaction num_tf_ints, default = 2

Procedure:

- Divide num_tfs TFs amongst modules in backbone such that each module has at least min_tfs TFs.
- For every TF in a particular module M, connect that TF with num_tf_ints TFs for each upstream module of M.
- The strength and cooperativity of interactions created this way are defined by the module network.

4.2.2 Generate targets

Parameters:

- Number of target genes $num\_targets$
- Name of a FANTOM5 GRN
- Damping factor $damping$, default = 0.05
- Target resampling $resamp$, default = $+\infty$
- Maximum in-degree max_in_degree, default = 5

Procedure:

- Download the FANTOM5 GRN if not already available
- Randomly map TFs to the regulators in the GRN
- Compute page rank from the selected regulators in the GRN with damping factor $damping$
- Perform weighted sample of $\min(num\_targets, resamp)$ targets weighted by the page rank
- Select subgraph induced by the TFs and the sampled targets in the GRN, remove edges to the TFs
- Add subgraph to current TF-target network
- If less than $num\_targets$ have been sampled in this way, go back to step 2.
- Remove regulatory interactions if a target has more than max_in_degree edges.

4.2.3 Generate housekeeping genes

Parameters:

- Number of housekeeping genes $num\_targets$
- Name of a FANTOM5 GRN
- Target resampling $resamp$, default = $+\infty$
- Maximum in-degree max_in_degree, default = 5

Procedure:

- Use same FANTOM5 GRN
- Subsample GRN such that each gene has a maximum in-degree of max_in_degree
- Perform breadth-first-search from a random gene in the GRN, select $\min(num\_targets, resamp)$ first genes encountered
- Select subgraph induced by the sampled housekeeping genes in the GRN
- Add subgraph to current TF-target network
- If less than $num\_targets$ have been sampled in this way, go back to step 2.

4.3 Convert gene regulatory network to a set of reactions

Each reaction consists of its propensity – a formula to calculate the probability of occurring during an infinitesimal time interval – and the effect – how it will affect the current state if triggered.

We define the abundance levels of pre-mRNA, mRNA and protein of gene G as w_G x_G and y_G respectively. Five reactions affect the abundance levels of these molecules: transcription, splicing, mRNA degradation, translation, and protein degradation. The effects and propensity functions of these reactions are defined in Table 1.

Table 1: Reactions affecting the abundance levels of pre-mRNA w_G, 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	$\emptyset \to w_G$	$wpr_G \times \frac{ba_G - 1 + \prod_{H \in R_G^+} (1 + reg_{G,H})}{ba_G - 1 + \prod_{H \in R_G} (1 + reg_{G,H})}$,
Splicing	$w_G \to x_G$	$wsr_G \times w_G$
mRNA degradation	$x_G \to \emptyset$	$xdr_G \times x_G$
Translation	$x_G \to x_G + y_G$	$ypr_G \times x_G$
Protein degradation	$y_G \to \emptyset$	$ydr_G \times y_G$

Table 2: Parameters defined for the calculation of reaction propensity functions.

Parameter	Symbol	Definition
Transcription rate	wpr_G	$\in \{.\|. \in N(100, 20), . \geq 10\}$
Splicing rate	wsr_G	$\in \{.\|. \in N(10, 2), . \geq 2\}$
mRNA degradation rate	xdr_G	$\in \{.\|. \in N(5, 1), . \geq 2\}$
Translation rate	ypr_G	$\in \{.\|. \in N(5, 1), . \geq 2\}$
Protein degradation rate	ydr_G	$\in \{.\|. \in N(3, 0.5), . \geq 1\}$
Interaction strength	$str_{G,H}$	$\in \{10 \cdot \|. \in U(0, 2)\}$ *
Interaction cooperativity	$co_{G,H}$	$\in U(0.5, 2)$ *
Max protein	$maxy_H$	$= \frac{wpr_H \times ypr_H}{xdr_H \times ydr_H}$
Regulation activity	$reg_{G,H}$	$= \left(str_{G,H} \times \frac{y_H}{0.5 \times maxy_H}\right)^{co_{G,H}}$
Basal expression	ba_G	$= \begin{cases} 1 & \text{if } R_G^+ = \emptyset \\ 0.0001 & \text{if } R_G^- = \emptyset \text{ and } R_G^+ \neq \emptyset \text{ *} \\ 0.5 & \text{otherwise} \end{cases}$

*: unless already defined when G is a TF.

4.4 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 4.1), 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 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 4.3) occur t times its propensity.

4.5 Simulate single cells

dyngen uses Gillespie's Stochastic Simulation Algorithm (SSA) 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 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.

SSA simulations are notoriously slow. We use GillespieSSA2 which contains many optimisations such as translating and compiling all the propensity functions to C++ and implementations of approximations of SSA which allows to trigger many reactions simultaneously at each iteration.

The framework 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 $reg_{G,H}$.

4.5.1 Map SSA simulations to backbone

The cellular state of each timepoint in the SSA simulation is mapped to the state network of the backbone by calculating the 1NN between a state vector in the simulation and the average expression levels along transitions.

4.6 Simulate experiment

From the SSA simulation we obtain the abundance levels of all the molecules at the different time points. 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.

4.6.1 Sample cells

Cells can be sampled from an unsynchronised population of single cells (snapshot) or at multiple time points in a synchronised population (time series).

Snapshot Cells are just sampled randomly from the different time points in the simulation.

Time series The timeline of the simulations is cut up into chunks. From several of these chunks, cells are sampled. For each cell it is known at which time point it was sampled.

4.6.2 Sample molecules

- From real dataset, look at the number of transcripts that was captured per cell. Library size ls_i of cell i is samples from this distribution.
- Capture rate of each molecule type j is drawn from $cr_j \in N(1, 0.05)$
- For each cell i, a multinomial distribution is used to draw ls_i molecules from molecule type j with probability $cr_j \times ab_{i,j}$ with $ab_{i,j}$ the molecule abundance level of molecule j in cell i.

4.7 Example runs of predefined backbones

4.7.1 Linear

4.7.2 Bifurcating

4.7.3 Cycle

4.7.4 Branching

(and binary tree, consecutive bifurcating, trifurcating)

4.7.5 Converging

4.7.6 Bifurcating converging

4.7.7 Bifurcating cycle

4.7.8 Bifurcating loop

4.7.9 Disconnected

4.8 Example use cases

4.8.1 Trajectory alignment

From discussion: Adding batch effects to snapshot simulations of linear (or even branching) trajectories allows evaluating trajectory alignment methods – which attempt to map two or more trajectories onto each other.

4.8.2 Differential network inference

From discussion: Adding perturbations to the GRN allows evaluating the performance of differential network inference methods – which predict differential regulatory interactions between two or more groups of profiles.

4.8.3 RNA velocity

From discussion: Sampling a cell at a certain time point and once more at a later time point allows 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.

4.8.4 Perturbation experiment

References

[1] Robrecht Cannoodt, Wouter Saelens, and Yvan Saeys. "Computational Methods for Trajectory Inference from Single-Cell Transcriptomics". en. In: *European Journal of Immunology* 46.11 (Nov. 2016), pp. 2496–2506. ISSN: 1521-4141. DOI: 10.1002/eji.201646347.

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

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] Daniel Marbach et al. "Wisdom of Crowds for Robust Gene Network Inference". In: *Nature methods* 9.8 (July 2012), pp. 796–804. ISSN: 1548-7091. DOI: 10.1038/nmeth.2016.

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

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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